М.Ф. Пічугін, І.О. Канкін, В.В. Воротніков

КОМП'ЮТЕРНА ГРАФІКА

НАВЧАЛЬНИЙ ПОСІБНИК

Рекомендовано
Міністерством освіти і науки, молоді та спорту України
для студентів вищих навчальних закладів

«Видавництво
«Центр учбової літератури»
Київ – 2013

М.Ф. Пічугін, І.О. Канкін, В.В. Воротніков

КОМП'ЮТЕРНА ГРАФІКА

НАВЧАЛЬНИЙ ПОСІБНИК

Рекомендовано
Міністерством освіти і науки, молоді та спорту України
для студентів вищих навчальних закладів

«Видавництво
«Центр учбової літератури»
Київ – 2013

УДК 681.3.06(075.8)
ББК 32.973-01я73
П 36

Гриф надано
Міністерством освіти і науки, молоді та спорту України
(*лист №1/11-16818 від 29 жовтня 2012 року.*)

Рецензенти:
Ю. К. Зіатдінов – доктор технічних наук, професор, дійсний член Аерокосмічної академії України;
Л. В. Лось – доктор технічних наук, професор, заслужений діяч науки і техніки України;
Ю. Г. Грицюк – доктор технічних наук, доцент.

Пічугін М. Ф.
Комп'ютерна графіка [текст] : навч. посіб. / М. Ф. Пічугін, І. О. Канкін, В. В.
П 36 Воротніков – К. : «Центр учбової літератури», 2013. – 346 с.

ISBN 978-617-673-181-8

Навчальний посібник призначений для курсантів та студентів які вивчають дисципліну “Комп’ютерна графіка”.

У навчальному посібнику висвітлено питання загального ознайомлювального плану, що дозволяє зорієнтуватися в такій області інформатики, як комп'ютерна графіка. Розглянуто математичні та алгоритмічні основи подання об’єктів комп’ютерної графіки та питання ефективності обчислень трудомістких операцій тривимірної графіки. Розглянуті прикладні пакети для роботи з растровими та векторними зображеннями. На прикладах показується використання стандартних алгоритмів машинної графіки, інтерфейсних засобів, сучасного мультимедійного забезпечення тощо.

Автори: кандидат військових наук, професор **М.Ф. Пічугін**; кандидат технічних наук **І.О. Канкін**; кандидат технічних наук, доцент **В.В. Воротніков**.

УДК 681.3.06(075.8)
ББК 32.973-01я73

ISBN 978-617-673-181-8

УДК 681.3.06(075.8)
ББК 32.973-01я73
П 36

Гриф надано
Міністерством освіти і науки, молоді та спорту України
(*лист №1/11-16818 від 29 жовтня 2012 року.*)

Рецензенти:
Ю. К. Зіатдінов – доктор технічних наук, професор, дійсний член Аерокосмічної академії України;
Л. В. Лось – доктор технічних наук, професор, заслужений діяч науки і техніки України;
Ю. Г. Грицюк – доктор технічних наук, доцент.

Пічугін М. Ф.
Комп'ютерна графіка [текст] : навч. посіб. / М. Ф. Пічугін, І. О. Канкін, В. В.
П 36 Воротніков – К. : «Центр учбової літератури», 2013. – 346 с.

ISBN 978-617-673-181-8

Навчальний посібник призначений для курсантів та студентів які вивчають дисципліну “Комп’ютерна графіка”.

У навчальному посібнику висвітлено питання загального ознайомлювального плану, що дозволяє зорієнтуватися в такій області інформатики, як комп'ютерна графіка. Розглянуто математичні та алгоритмічні основи подання об’єктів комп’ютерної графіки та питання ефективності обчислень трудомістких операцій тривимірної графіки. Розглянуті прикладні пакети для роботи з растровими та векторними зображеннями. На прикладах показується використання стандартних алгоритмів машинної графіки, інтерфейсних засобів, сучасного мультимедійного забезпечення тощо.

Автори: кандидат військових наук, професор **М.Ф. Пічугін**; кандидат технічних наук **І.О. Канкін**; кандидат технічних наук, доцент **В.В. Воротніков**.

УДК 681.3.06(075.8)
ББК 32.973-01я73

ISBN 978-617-673-181-8

© . М. Ф. Пічугін, І. О. Канкін, В. В. Воротніков., 2013.
© «Видавництво «Центр учбової літератури», 2013.

ЗМІСТ

Вступ..... 5
РОЗДІЛ 1. БАЗОВІ УЯВЛЕННЯ ПРО КОМП'ЮТЕРНУ ГРАФІКУ..... 7
Глава 1. Основні поняття комп'ютерної графіки..... 8
1.1. Предмет та задачі дисципліни «Комп'ютерна графіка»..... 8
1.2. Історія розвитку комп'ютерної графіки..... 13
1.3. Види комп'ютерної графіки..... 18
1.4. Колірні моделі..... 32
1.5. Апаратні засоби комп'ютерної графіки..... 51

Глава 2. Математичні й алгоритмічні основи двовимірної графіки..... 78
2.1. Зображення та перетворення точок..... 78
2.2. Зображення та перетворення ліній..... 81
2.3.Однорідні координати та матричне подання двовимірних перетворень..... 88
2.4. Ефективність обчислень..... 93

Глава 3. Математичні й алгоритмічні основи тривимірної графіки..... 96
3.1. Просторові перетворення..... 96
3.2. Показ тривимірних зображень на двовимірній площині..... 111
3.3. Перетворення, пов'язані із системою координат..... 118
3.4. Алгоритми растрової графіки..... 119
3.5. Нормуючі перетворення видимого об'єкта..... 123
3.6. Алгоритми видалення невидимих ребер і граней..... 126
3.7. Моделі розрахунку освітленості граней тривимірних об'єктів..... 130
3.8. Кубічні сплайни..... 133
Глава 4. Формати графічних файлів та алгоритми стиснення..... 140
4.1. Типи графічних файлів..... 141
4.2. Елементи графічного файла..... 147
4.3. Растрові формати..... 148
4.4. Векторні формати..... 160
4.5. Метафайли та інші формати..... 162
4.6. Алгоритми стиснення даних..... 176

РОЗДІЛ 2. ІНТЕРАКТИВНА КОМП'ЮТЕРНА ГРАФІКА..... 195
Глава 5. Розробка графічних програм для Windows..... 196

ЗМІСТ

Вступ..... 5
РОЗДІЛ 1. БАЗОВІ УЯВЛЕННЯ ПРО КОМП'ЮТЕРНУ ГРАФІКУ..... 7
Глава 1. Основні поняття комп'ютерної графіки..... 8
1.1. Предмет та задачі дисципліни «Комп'ютерна графіка»..... 8
1.2. Історія розвитку комп'ютерної графіки..... 13
1.3. Види комп'ютерної графіки..... 18
1.4. Колірні моделі..... 32
1.5. Апаратні засоби комп'ютерної графіки..... 51

Глава 2. Математичні й алгоритмічні основи двовимірної графіки..... 78
2.1. Зображення та перетворення точок..... 78
2.2. Зображення та перетворення ліній..... 81
2.3.Однорідні координати та матричне подання двовимірних перетворень..... 88
2.4. Ефективність обчислень..... 93

Глава 3. Математичні й алгоритмічні основи тривимірної графіки..... 96
3.1. Просторові перетворення..... 96
3.2. Показ тривимірних зображень на двовимірній площині..... 111
3.3. Перетворення, пов'язані із системою координат..... 118
3.4. Алгоритми растрової графіки..... 119
3.5. Нормуючі перетворення видимого об'єкта..... 123
3.6. Алгоритми видалення невидимих ребер і граней..... 126
3.7. Моделі розрахунку освітленості граней тривимірних об'єктів..... 130
3.8. Кубічні сплайни..... 133
Глава 4. Формати графічних файлів та алгоритми стиснення..... 140
4.1. Типи графічних файлів..... 141
4.2. Елементи графічного файла..... 147
4.3. Растрові формати..... 148
4.4. Векторні формати..... 160
4.5. Метафайли та інші формати..... 162
4.6. Алгоритми стиснення даних..... 176

РОЗДІЛ 2. ІНТЕРАКТИВНА КОМП'ЮТЕРНА ГРАФІКА..... 195
Глава 5. Розробка графічних програм для Windows..... 196

5.2. Робота з графікою з використанням класів, властивостей і функцій компонент Borland C++ Builder…… 206

Глава 6. Графічна бібліотека OpenGL…… 216
6.1. Загальні відомості про OpenGL…… 216
6.2. Створення та перетворення графічних об'єктів…….. 225
6.3. Моделювання освітлення та текстури…….. 245
6.4. Приклади роботи з OpenGL…….. 267

Глава 7. Характеристика основних можливостей пакету растрової графіки..... 274
7.1. Основні можливості та інструменти…… 274
7.2. Малювання та ретушування зображень…… 280
7.3. Виділені області, маски та фільтри……..... 284
7.4. Шари, об'єкти, текст……..…… 288
7.5. Анімація та обробка відео-файлів…… 290

Глава 8. Характеристика основних можливостей пакету векторної графіки……... 295
8.1. Основні можливості та інструменти…… 295
8.2. Розміщення об'єктів…… 303
8.3. Векторні трансформації і фільтри…… 316
8.4. Робота з шарами…….. 327
8.5. Робота з текстом і шрифтом……..... 330
8.6. Робота з піксельними зображеннями…… 331
8.7. Інформаційна графіка (діаграми)……... 333

Список скорочень ……..…… 338
Глосарій……... 339
Список літератури…… 342
Предметний покажчик…… 344

5.2. Робота з графікою з використанням класів, властивостей і функцій компонент Borland C++ Builder…… 206

Глава 6. Графічна бібліотека OpenGL…… 216
6.1. Загальні відомості про OpenGL…… 216
6.2. Створення та перетворення графічних об'єктів…….. 225
6.3. Моделювання освітлення та текстури…….. 245
6.4. Приклади роботи з OpenGL…….. 267

Глава 7. Характеристика основних можливостей пакету растрової графіки..... 274
7.1. Основні можливості та інструменти…… 274
7.2. Малювання та ретушування зображень…… 280
7.3. Виділені області, маски та фільтри……..... 284
7.4. Шари, об'єкти, текст……..…… 288
7.5. Анімація та обробка відео-файлів…… 290

Глава 8. Характеристика основних можливостей пакету векторної графіки……... 295
8.1. Основні можливості та інструменти…… 295
8.2. Розміщення об'єктів…… 303
8.3. Векторні трансформації і фільтри…… 316
8.4. Робота з шарами…….. 327
8.5. Робота з текстом і шрифтом……..... 330
8.6. Робота з піксельними зображеннями…… 331
8.7. Інформаційна графіка (діаграми)……... 333

Список скорочень ……..…… 338
Глосарій……... 339
Список літератури…… 342
Предметний покажчик…… 344

Вступ

Навчальний посібник призначений для забезпечення навчальної дисципліни «Комп'ютерна графіка» та спрямований на розкриття базових питань, пов'язаних з розробкою, обробкою та описом графічних зображень за допомогою комп'ютера.

Комп'ютерна графіка (КГ) – це область комп'ютерних технологій, яка в наш час набуває стрімкого розвитку. Це обумовлено використанням її результатів у рекламному бізнесі, Інтернеті, телебаченні та кіноіндустрії, в проектуванні та моделюванні різного роду складних систем як у народному господарстві, так і у військовій сфері. Якщо ви плануєте працювати в одній із вищевказаних областей, то ця книга допоможе ознайомитись з математичними основами комп'ютерної графіки, способами побудови, обробки та зберігання графічних зображень.

Охопити усі сфери використання комп'ютерної графіки з достатньою глибиною в одному навчальному посібнику просто не можливо, та напевно і не потрібно. Тому метою видання є висвітлення відомостей про концептуальні основи роботи з графічними зображеннями, що стане підґрунтям для подальшого вивчення, розуміння та використання будь-яких додатків інтерактивної комп'ютерної графіки.

Навчальний посібник складається з двох розділів.

Розділ 1. Базові уявлення про комп'ютерну графіку. Ця частина книги (глави 1–4) присвячена основам комп'ютерної графіки. Це своєрідний «лікбез» по термінології та базових поняттях.

Матеріал першої глави розділу присвячений питанням історії комп'ютерної графіки, задачам, які вона розв'язує, розглянуті колірні моделі та апаратні засоби комп'ютерної графіки.

У другій главі висвітлені математичні й алгоритмічні основи двовимірної графіки, а саме питання зображення та перетворення основних графічних примітивів (точок та ліній), розкриваються поняття

Вступ

Навчальний посібник призначений для забезпечення навчальної дисципліни «Комп'ютерна графіка» та спрямований на розкриття базових питань, пов'язаних з розробкою, обробкою та описом графічних зображень за допомогою комп'ютера.

Комп'ютерна графіка (КГ) – це область комп'ютерних технологій, яка в наш час набуває стрімкого розвитку. Це обумовлено використанням її результатів у рекламному бізнесі, Інтернеті, телебаченні та кіноіндустрії, в проектуванні та моделюванні різного роду складних систем як у народному господарстві, так і у військовій сфері. Якщо ви плануєте працювати в одній із вищевказаних областей, то ця книга допоможе ознайомитись з математичними основами комп'ютерної графіки, способами побудови, обробки та зберігання графічних зображень.

Охопити усі сфери використання комп'ютерної графіки з достатньою глибиною в одному навчальному посібнику просто не можливо, та напевно і не потрібно. Тому метою видання є висвітлення відомостей про концептуальні основи роботи з графічними зображеннями, що стане підґрунтям для подальшого вивчення, розуміння та використання будь-яких додатків інтерактивної комп'ютерної графіки.

Навчальний посібник складається з двох розділів.

Розділ 1. Базові уявлення про комп'ютерну графіку. Ця частина книги (глави 1–4) присвячена основам комп'ютерної графіки. Це своєрідний «лікбез» по термінології та базових поняттях.

Матеріал першої глави розділу присвячений питанням історії комп'ютерної графіки, задачам, які вона розв'язує, розглянуті колірні моделі та апаратні засоби комп'ютерної графіки.

У другій главі висвітлені математичні й алгоритмічні основи двовимірної графіки, а саме питання зображення та перетворення основних графічних примітивів (точок та ліній), розкриваються поняття

однорідних координат та матричного подання двомірних перетворень.

Третя глава охоплює питання, пов'язані з математичними й алгоритмічними основами тривимірної графіки. Розглянуто методику просторових перетворень тривимірних об'єктів, геометричні проекції, що застосовуються в комп'ютерній графіці, та базові алгоритми растрової та векторної графіки.

Завершується перший розділ четвертою главою, де розглядається класифікація та сутність форматів графічних файлів та алгоритми стиснення зображень.

Розділ 2. Інтерактивна комп'ютерна графіка. Під терміном інтерактивної комп'ютерної графіки будемо розуміти здатність комп'ютерної системи створювати графіку та вести діалог з людиною.

Історично першими інтерактивними системами вважаються системи автоматичного проектування [3, 19]. Нині стають усе більш популярними геоінформаційні системи, які є відносно новим різновидом систем інтерактивної комп'ютерної графіки. Сьогодні майже будь-яку програму можна вважати системою інтерактивної КГ.

Тому у другому розділі надано матеріал, що є основою для інтерактивної КГ, та розглядаються принципи роботи з графічними редакторами растрової та векторної графіки.

Глава п'ята присвячена використанню API функцій для побудови графічних зображень.

У шостій главі детально розглядається графічна бібліотека OpenGL, наведено основні можливості цієї бібліотеки, методики створення та перетворення графічних об'єктів, моделювання освітлення та текстура. Наведено численні приклади використання функцій цієї бібліотеки.

Сьома та восьма глави присвячені принципам роботи з графічними редакторами растрової (Adobe Photoshop) та векторної (Adobe Illustrator) графіки відповідно.

однорідних координат та матричного подання двомірних перетворень.

Третя глава охоплює питання, пов'язані з математичними й алгоритмічними основами тривимірної графіки. Розглянуто методику просторових перетворень тривимірних об'єктів, геометричні проекції, що застосовуються в комп'ютерній графіці, та базові алгоритми растрової та векторної графіки.

Завершується перший розділ четвертою главою, де розглядається класифікація та сутність форматів графічних файлів та алгоритми стиснення зображень.

Розділ 2. Інтерактивна комп'ютерна графіка. Під терміном інтерактивної комп'ютерної графіки будемо розуміти здатність комп'ютерної системи створювати графіку та вести діалог з людиною.

Історично першими інтерактивними системами вважаються системи автоматичного проектування [3, 19]. Нині стають усе більш популярними геоінформаційні системи, які є відносно новим різновидом систем інтерактивної комп'ютерної графіки. Сьогодні майже будь-яку програму можна вважати системою інтерактивної КГ.

Тому у другому розділі надано матеріал, що є основою для інтерактивної КГ, та розглядаються принципи роботи з графічними редакторами растрової та векторної графіки.

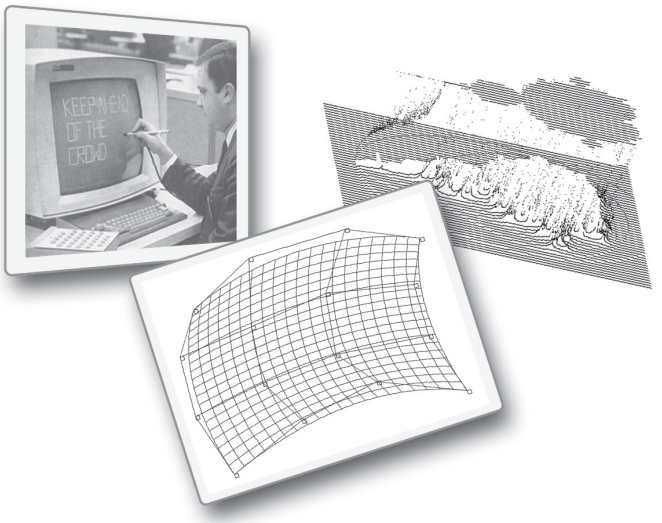
Глава п'ята присвячена використанню API функцій для побудови графічних зображень.

У шостій главі детально розглядається графічна бібліотека OpenGL, наведено основні можливості цієї бібліотеки, методики створення та перетворення графічних об'єктів, моделювання освітлення та текстура. Наведено численні приклади використання функцій цієї бібліотеки.

Сьома та восьма глави присвячені принципам роботи з графічними редакторами растрової (Adobe Photoshop) та векторної (Adobe Illustrator) графіки відповідно.

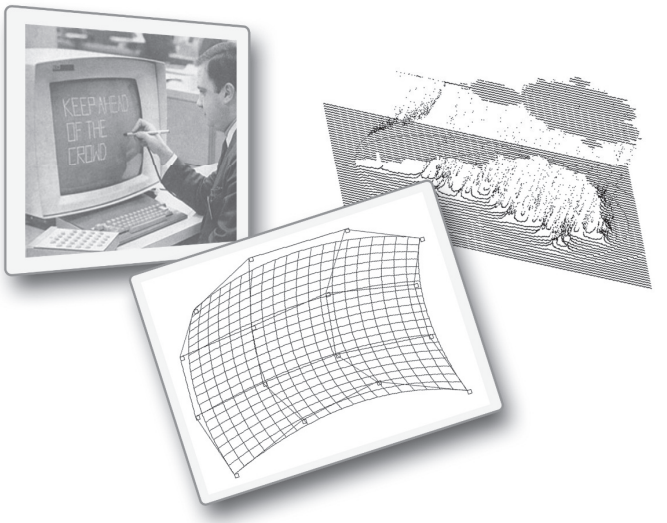
Розділ 1
БАЗОВІ УЯВЛЕННЯ ПРО КОМП'ЮТЕРНУ ГРАФІКУ

- Основні поняття комп'ютерної графіки
- Математичні й алгоритмічні основи двовимірної графіки
- Математичні й алгоритмічні основи тривимірної графіки
- Формати графічних файлів та алгоритми стиснення



Розділ 1
БАЗОВІ УЯВЛЕННЯ ПРО КОМП'ЮТЕРНУ ГРАФІКУ

- Основні поняття комп'ютерної графіки
- Математичні й алгоритмічні основи двовимірної графіки
- Математичні й алгоритмічні основи тривимірної графіки
- Формати графічних файлів та алгоритми стиснення



Глава 1

Основні поняття комп'ютерної графіки

1.1. Предмет та задачі дисципліни комп'ютерна графіка

Разом із широким впровадженням персональних електронно-обчислювальних машин (ПЕОМ) знайшла своє застосування і комп'ютерна графіка.

У військовій області можна виділити такі сфери: навчання і тренування обслуговуючого персоналу; діагностика й оцінка функціонування зразків озброєння у військовій техніці; відображення на автоматизованих робочих місцях (АРМ) і колективних засобах відображення інформації (КЗВІ); інтелектуальні системи підтримки й прийняття рішення командиром, під час оцінки ефективності прийнятих рішень; розробка нових зразків озброєння і військової техніки.

У народному господарстві виділяються: архітектурне проектування; розпізнавання відеообразів; автоматизоване проектування у машинобудуванні; медична діагностика, у тому числі для відновлення форми прихованих об'єктів, для розробки відеотренажерів та імітаторів складних сцен і об'єктів; реклама і мультиплікація.

Як приклад комплексного використання комп'ютерної графіки можна привести американську систему Lansat глобального моніторингу Землі. Один знімок, переданий зі штучного супутника Землі, містить залежно від типу приладу від 200 до 2000 Мбіт інформації. Далі знімки коректуються від радіометричних і геометричних перешкод. Відкоректовані знімки у цифровій формі вводяться в ЕОМ для аналізу. Про широту використання знімків говорить статистика: 25-30% - фірми, 20-25% - урядові заклади: 15-20% - навчальні та наукові організації; 30-35% - за межами США [5, 22].

Глава 1

Основні поняття комп'ютерної графіки

1.1. Предмет та задачі дисципліни комп'ютерна графіка

Разом із широким впровадженням персональних електронно-обчислювальних машин (ПЕОМ) знайшла своє застосування і комп'ютерна графіка.

У військовій області можна виділити такі сфери: навчання і тренування обслуговуючого персоналу; діагностика й оцінка функціонування зразків озброєння у військовій техніці; відображення на автоматизованих робочих місцях (АРМ) і колективних засобах відображення інформації (КЗВІ); інтелектуальні системи підтримки й прийняття рішення командиром, під час оцінки ефективності прийнятих рішень; розробка нових зразків озброєння і військової техніки.

У народному господарстві виділяються: архітектурне проектування; розпізнавання відеообразів; автоматизоване проектування у машинобудуванні; медична діагностика, у тому числі для відновлення форми прихованих об'єктів, для розробки відеотренажерів та імітаторів складних сцен і об'єктів; реклама і мультиплікація.

Як приклад комплексного використання комп'ютерної графіки можна привести американську систему Lansat глобального моніторингу Землі. Один знімок, переданий зі штучного супутника Землі, містить залежно від типу приладу від 200 до 2000 Мбіт інформації. Далі знімки коректуються від радіометричних і геометричних перешкод. Відкоректовані знімки у цифровій формі вводяться в ЕОМ для аналізу. Про широту використання знімків говорить статистика: 25-30% - фірми, 20-25% - урядові заклади: 15-20% - навчальні та наукові організації; 30-35% - за межами США [5, 22].

Обробка інформації, що подана у вигляді зображень, має безліч різновидів і практичних додатків. Цю область обробки інформації прийнято поділяти на такі види: комп'ютерна графіка, обробка зображень, розпізнавання зображень. Їх зв'язок зображено на рис.1.1.

Обробка зображень пов'язана з розв'язання таких задач, у яких вхідні і вихідні дані є зображеннями. Прикладом цього є системи передачі зображень з космічного апарата. Їх розроблювачі зустрічаються з проблемами усунення шуму і стискування даних.

Розпізнавання зображень пов'язане із розв'язанням задач, що дозволяють одержати деякий опис зображення, який поданий на вхід системи, або віднести це зображення до певного визначеного класу. Процедура розпізнавання застосовується до обраного зображення і забезпечує перетворення його в деякий абстрактний опис – набір чисел, ланцюжок символів або графів. Прикладом може бути система космічної видової розвідки. Терміном **комп'ютерна графіка** позначають генерацію або оцінку графічних об'єктів ПЕОМ, маніпулювання ними, а також установлення зв'язку між графічними об'єктами і неграфічною інформацією, що знаходиться у файлах.

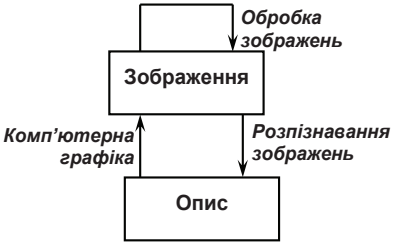


Рис. 1.1. Взаємозв'язок комп'ютерної графіки і обробки зображень, розпізнавання зображень

Комп'ютерну графіку можна розбити на області (рис. 1.2): зображувальна комп'ютерна графіка, аналіз зображень та перцептивна графіка (аналіз сцен).

Зображувальна графіка має справу із штучно створеними графічними об'єктами, що складаються, як правило, із ліній. До основних задач, що розв'язуються у зображувальній графіці, відносяться:

1. Побудова моделі (об'єкта) і генерація зображення.
2. Перетворення моделі і зображення.
3. Ідентифікація об'єкта і добування інформації.

Обробка інформації, що подана у вигляді зображень, має безліч різновидів і практичних додатків. Цю область обробки інформації прийнято поділяти на такі види: комп'ютерна графіка, обробка зображень, розпізнавання зображень. Їх зв'язок зображено на рис.1.1.

Обробка зображень пов'язана з розв'язання таких задач, у яких вхідні і вихідні дані є зображеннями. Прикладом цього є системи передачі зображень з космічного апарата. Їх розроблювачі зустрічаються з проблемами усунення шуму і стискування даних.

Розпізнавання зображень пов'язане із розв'язанням задач, що дозволяють одержати деякий опис зображення, який поданий на вхід системи, або віднести це зображення до певного визначеного класу. Процедура розпізнавання застосовується до обраного зображення і забезпечує перетворення його в деякий абстрактний опис – набір чисел, ланцюжок символів або графів. Прикладом може бути система космічної видової розвідки. Терміном **комп'ютерна графіка** позначають генерацію або оцінку графічних об'єктів ПЕОМ, маніпулювання ними, а також установлення зв'язку між графічними об'єктами і неграфічною інформацією, що знаходиться у файлах.

Рис. 1.1. Взаємозв'язок комп'ютерної графіки і обробки зображень, розпізнавання зображень

Комп'ютерну графіку можна розбити на області (рис. 1.2): зображувальна комп'ютерна графіка, аналіз зображень та перцептивна графіка (аналіз сцен).

Зображувальна графіка має справу із штучно створеними графічними об'єктами, що складаються, як правило, із ліній. До основних задач, що розв'язуються у зображувальній графіці, відносяться:

1. Побудова моделі (об'єкта) і генерація зображення.
2. Перетворення моделі і зображення.
3. Ідентифікація об'єкта і добування інформації.

Модель об'єкта - абстрактний опис графічного об'єкта, що може бути перетворений у відповідне зображення на ПЕОМ. Крім звичайної ПЕОМ у зображувальній графіці використовуються додаткові інструменти – *графопобудовники* й інтерактивні *дисплейні системи.*

Під час обробки й аналіза зображень маємо справу з фотографічними зображеннями, або більш конкретно – з дискретними поданнями таких зображень, оформлених у вигляді масивів чисел.

Зображення сканується й оцифровується, тобто у визначених точках вимірюється рівень почорніння, і результати вимірів перетворюються у числову форму. Основні задачі під час обробки зображення:

1. Підвищення якості зображення (збільшення контрастності, заглушення шумів).
2. Оцінка зображення (визначення розміру, форми і місця розташування деяких об'єктів, що є на зображенні).
3. Розпізнавання образів (виділення і класифікація властивостей).

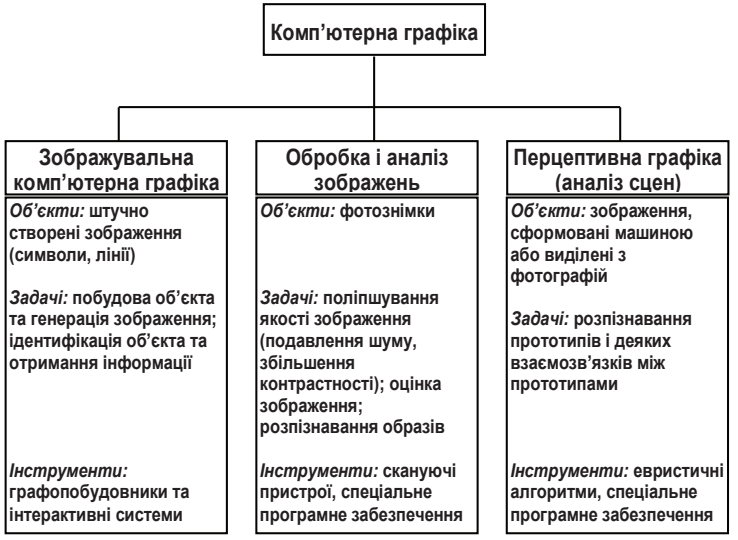


Рис. 1.2. Області комп'ютерної графіки

Модель об'єкта - абстрактний опис графічного об'єкта, що може бути перетворений у відповідне зображення на ПЕОМ. Крім звичайної ПЕОМ у зображувальній графіці використовуються додаткові інструменти – *графопобудовники* й інтерактивні *дисплейні системи.*

Під час обробки й аналіза зображень маємо справу з фотографічними зображеннями, або більш конкретно – з дискретними поданнями таких зображень, оформлених у вигляді масивів чисел.

Зображення сканується й оцифровується, тобто у визначених точках вимірюється рівень почорніння, і результати вимірів перетворюються у числову форму. Основні задачі під час обробки зображення:

1. Підвищення якості зображення (збільшення контрастності, заглушення шумів).
2. Оцінка зображення (визначення розміру, форми і місця розташування деяких об'єктів, що є на зображенні).
3. Розпізнавання образів (виділення і класифікація властивостей).

Рис. 1.2. Області комп'ютерної графіки

10

У *перцептивній* графіці (або аналізі сцен) предметом дослідження є абстрактні моделі графічних об'єктів і взаємозв'язок між ними. При цьому вирішуються задачі:

1. Розпізнавання прототипів.
2. Розпізнавання деяких взаємозв'язків між прототипами.

У комп'ютерній графіці розглядаються такі завдання:

- подання зображення в комп'ютерній графіці;
- підготовка зображення до візуалізації;
- створення зображення;
- здійснення дій із зображенням.

Під комп'ютерною графікою зазвичай розуміють автоматизацію процесів підготовки, перетворення, зберігання і відтворення графічної інформації за допомогою комп'ютера. Під графічною інформацією розуміються моделі об'єктів та їх зображення [11, 13, 24].

У випадку, якщо користувач може управляти характеристиками об'єктів, говорять про інтерактивну комп'ютерну графіку, тобто здатність комп'ютерної системи створювати графіку і вести діалог з людиною. В даний час майже будь-яку програму можна вважати системою інтерактивної комп'ютерної графіки [3].

Інтерактивна комп'ютерна графіка – це те саме використання комп'ютерів для підготовки і відтворення зображень, але при цьому користувач має можливість оперативно вносити зміни до зображення безпосередньо в процесі його відтворення, тобто передбачається можливість роботи з графікою в режимі діалогу в реальному масштабі часу.

Інтерактивна графіка є важливим розділом комп'ютерної графіки, коли користувач має можливість динамічно управляти вмістом зображення, його формою, розміром і кольором на поверхні дисплея за допомогою інтерактивних пристроїв управління.

Історично першими інтерактивними системами вважаються системи автоматизованого проектування (САПР), які з'явилися в 60-х роках. Вони є значним етапом в еволюції комп'ютерів і програмного забезпечення. У системі інтерактивної комп'ютерної графіки користувач сприймає на дисплеї зображення, що представляє деякий складний об'єкт, і може вносити зміни в опис (модель) об'єкта. Та-

У *перцептивній* графіці (або аналізі сцен) предметом дослідження є абстрактні моделі графічних об'єктів і взаємозв'язок між ними. При цьому вирішуються задачі:

1. Розпізнавання прототипів.
2. Розпізнавання деяких взаємозв'язків між прототипами.

У комп'ютерній графіці розглядаються такі завдання:

- подання зображення в комп'ютерній графіці;
- підготовка зображення до візуалізації;
- створення зображення;
- здійснення дій із зображенням.

Під комп'ютерною графікою зазвичай розуміють автоматизацію процесів підготовки, перетворення, зберігання і відтворення графічної інформації за допомогою комп'ютера. Під графічною інформацією розуміються моделі об'єктів та їх зображення [11, 13, 24].

У випадку, якщо користувач може управляти характеристиками об'єктів, говорять про інтерактивну комп'ютерну графіку, тобто здатність комп'ютерної системи створювати графіку і вести діалог з людиною. В даний час майже будь-яку програму можна вважати системою інтерактивної комп'ютерної графіки [3].

Інтерактивна комп'ютерна графіка – це те саме використання комп'ютерів для підготовки і відтворення зображень, але при цьому користувач має можливість оперативно вносити зміни до зображення безпосередньо в процесі його відтворення, тобто передбачається можливість роботи з графікою в режимі діалогу в реальному масштабі часу.

Інтерактивна графіка є важливим розділом комп'ютерної графіки, коли користувач має можливість динамічно управляти вмістом зображення, його формою, розміром і кольором на поверхні дисплея за допомогою інтерактивних пристроїв управління.

Історично першими інтерактивними системами вважаються системи автоматизованого проектування (САПР), які з'явилися в 60-х роках. Вони є значним етапом в еволюції комп'ютерів і програмного забезпечення. У системі інтерактивної комп'ютерної графіки користувач сприймає на дисплеї зображення, що представляє деякий складний об'єкт, і може вносити зміни в опис (модель) об'єкта. Та-

кими змінами можуть бути як введення і редагування окремих елементів, так і завдання числових значень для будь-яких параметрів, а також інші операції по введенню інформації на основі сприйняття зображення.

Системи типу САПР активно використовуються в багатьох областях, наприклад у машинобудуванні та електроніці. Одними з перших були створені САПР для проектування літаків, автомобілів, системи для розробки мікроелектронних інтегральних схем, архітектурні системи. Такі системи на перших порах функціонували на чималих комп'ютерах. Потім поширилося використання швидкодіючих комп'ютерів середнього класу з розвиненими графічними можливостями – графічних робочих станцій. Із зростанням потужностей персональних комп'ютерів все частіше САПР використовували на дешевих масових комп'ютерах, які зараз мають достатню швидкодію і об'єми пам'яті для вирішення багатьох завдань. Це привело до широкого поширення цих систем.

Зараз стають усе більш популярними геоінформаційні системи (ГІС). Це відносно новий для масових користувачів різновид систем інтерактивної комп'ютерної графіки. Вони акумулюють у собі методи і алгоритми багатьох наук та інформаційних технологій. Такі системи використовують останні досягнення технологій баз даних, у них закладено багато методів і алгоритмів математики, фізики, геодезії, топології, картографії, навігації, а також, комп'ютерної графіки.

Системи типу ГІС частенько вимагають значних потужностей комп'ютера як у плані роботи з базами даних, так і для візуалізації об'єктів, які знаходяться на поверхні Землі. Причому візуалізацію необхідно робити з різною мірою деталізації – як для Землі в цілому, так і у межах окремих ділянок. У даний час помітно прагнення розробників ГІС підвищити реалістичність зображень просторових об'єктів і територій.

Типовими для будь-якої ГІС є такі операції – введення і редагування об'єктів з урахуванням їх розташування на поверхні Землі, формування всіляких цифрових моделей, запис в бази даних, виконання всіляких запитів до баз даних. Важливою операцією є аналіз

кими змінами можуть бути як введення і редагування окремих елементів, так і завдання числових значень для будь-яких параметрів, а також інші операції по введенню інформації на основі сприйняття зображення.

Системи типу САПР активно використовуються в багатьох областях, наприклад у машинобудуванні та електроніці. Одними з перших були створені САПР для проектування літаків, автомобілів, системи для розробки мікроелектронних інтегральних схем, архітектурні системи. Такі системи на перших порах функціонували на чималих комп'ютерах. Потім поширилося використання швидкодіючих комп'ютерів середнього класу з розвиненими графічними можливостями – графічних робочих станцій. Із зростанням потужностей персональних комп'ютерів все частіше САПР використовували на дешевих масових комп'ютерах, які зараз мають достатню швидкодію і об'єми пам'яті для вирішення багатьох завдань. Це привело до широкого поширення цих систем.

Зараз стають усе більш популярними геоінформаційні системи (ГІС). Це відносно новий для масових користувачів різновид систем інтерактивної комп'ютерної графіки. Вони акумулюють у собі методи і алгоритми багатьох наук та інформаційних технологій. Такі системи використовують останні досягнення технологій баз даних, у них закладено багато методів і алгоритмів математики, фізики, геодезії, топології, картографії, навігації, а також, комп'ютерної графіки.

Системи типу ГІС частенько вимагають значних потужностей комп'ютера як у плані роботи з базами даних, так і для візуалізації об'єктів, які знаходяться на поверхні Землі. Причому візуалізацію необхідно робити з різною мірою деталізації – як для Землі в цілому, так і у межах окремих ділянок. У даний час помітно прагнення розробників ГІС підвищити реалістичність зображень просторових об'єктів і територій.

Типовими для будь-якої ГІС є такі операції – введення і редагування об'єктів з урахуванням їх розташування на поверхні Землі, формування всіляких цифрових моделей, запис в бази даних, виконання всіляких запитів до баз даних. Важливою операцією є аналіз

із урахуванням просторових, топологічних стосунків безлічі об'єктів, розташованих на деякій території.

Робота з комп'ютерною графікою – один із найпопулярніших напрямів використання персонального комп'ютера, причому займаються цією роботою не лише професійні художники і дизайнери. На будь-якому підприємстві час від часу виникає необхідність у подачі рекламних оголошень в газети і журнали, у випуску рекламної листівки або буклета. Інколи підприємства замовляють таку роботу спеціальним дизайнерським бюро або рекламним агентствам, але часто обходяться власними силами і доступними програмними засобами.

Таким чином, можна зробити висновок, що комп'ютерна графіка стала невід'ємною частиною нашого життя, сфери її застосування постійно збільшуються, що, в свою чергу, вимагає більш детального вивчення питань, пов'язаних з отриманням, обробкою та розробкою графічних зображень. Але перед цим давайте спробуємо розібратися с тим, як усе починалось та коли з'явилась комп'ютерна графіка.

1.2. Історія розвитку комп'ютерної графіки

При всьому розмаїтті зображень, отриманих за допомогою комп'ютера, і вражаючих ефектів, які ми можемо сьогодні спостерігати, все починалося зовсім не так легко.

Розвиток комп'ютерної графіки, особливо на її початкових етапах, у першу чергу пов'язано з розвитком технічних засобів і особливо дисплеїв.

Точкою відліку розвитку комп'ютерної графіки можна вважати 1930 рік, коли в США Володимиром Зворикіним (рис. 1.3), який працював у компанії Westinghouse, була винайдена електронно-променева трубка (ЕПТ), яка вперше дозволила отримати зображення на екрані без використання механічних частин, що рухаються [19].

Початком ери власне комп'ютерної графіки можна вважати грудень 1951 року, коли в Массачусетському технологічному інституті (МТІ) для системи протиповітряної оборони військово-морського флоту США був розроблений перший дисплей для комп'ютера "Вихрь" (рис. 1.4). Винахідником цього дисплея був інженер з МТІ Джей Форрестер.

із урахуванням просторових, топологічних стосунків безлічі об'єктів, розташованих на деякій території.

Робота з комп'ютерною графікою – один із найпопулярніших напрямів використання персонального комп'ютера, причому займаються цією роботою не лише професійні художники і дизайнери. На будь-якому підприємстві час від часу виникає необхідність у подачі рекламних оголошень в газети і журнали, у випуску рекламної листівки або буклета. Інколи підприємства замовляють таку роботу спеціальним дизайнерським бюро або рекламним агентствам, але часто обходяться власними силами і доступними програмними засобами.

Таким чином, можна зробити висновок, що комп'ютерна графіка стала невід'ємною частиною нашого життя, сфери її застосування постійно збільшуються, що, в свою чергу, вимагає більш детального вивчення питань, пов'язаних з отриманням, обробкою та розробкою графічних зображень. Але перед цим давайте спробуємо розібратися с тим, як усе починалось та коли з'явилась комп'ютерна графіка.

1.2. Історія розвитку комп'ютерної графіки

При всьому розмаїтті зображень, отриманих за допомогою комп'ютера, і вражаючих ефектів, які ми можемо сьогодні спостерігати, все починалося зовсім не так легко.

Розвиток комп'ютерної графіки, особливо на її початкових етапах, у першу чергу пов'язано з розвитком технічних засобів і особливо дисплеїв.

Точкою відліку розвитку комп'ютерної графіки можна вважати 1930 рік, коли в США Володимиром Зворикіним (рис. 1.3), який працював у компанії Westinghouse, була винайдена електронно-променева трубка (ЕПТ), яка вперше дозволила отримати зображення на екрані без використання механічних частин, що рухаються [19].

Початком ери власне комп'ютерної графіки можна вважати грудень 1951 року, коли в Массачусетському технологічному інституті (МТІ) для системи протиповітряної оборони військово-морського флоту США був розроблений перший дисплей для комп'ютера "Вихрь" (рис. 1.4). Винахідником цього дисплея був інженер з МТІ Джей Форрестер.

Рис. 1.3. Володимир Зворикін: винахідник електронно-променевої трубки

Перше використання комп'ютерної графіки пов'язують з ім'ям Дж.Уїтні. Він займався кіновиробництвом в 50-60-х роках минулого століття і вперше використовував комп'ютер для створення титрів до кінофільму. Наступним кроком у своєму розвитку комп'ютерна графіка зобов'язана Айвену Сазерленду, який у 1961 році, ще будучи студентом, створив програму малювання, названу ним Sketchpad (рис. 1.5).

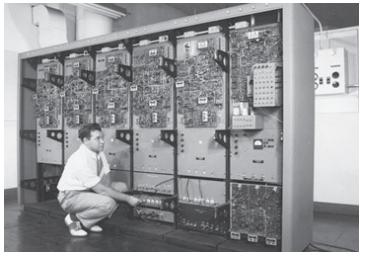


Рис. 1.4. Комп'ютер «Вихрь»

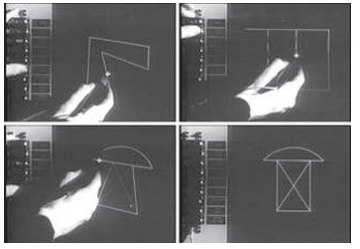


Рис. 1.5. Програма комп'ютерної графіки під назвою Sketchpad

Програма використовувала світлове перо для малювання простих фігур на екрані. Отримані картинки можна було зберігати і відновлювати. У цій програмі був розширений круг основних графічних примітивів, зокрема, окрім ліній і точок був введений прямокутник, який задавався своїми розмірами і розміщенням.

Спочатку комп'ютерна графіка була векторною, тобто зображення формувалося з тонких ліній. Ця особливість була пов'язана з технічною реалізацією комп'ютерних дисплеїв. Надалі ширше вживання отримала растрова графіка, заснована на представленні зображення на екрані у вигляді матриці однорідних елементів (пікселів).

Рис. 1.3. Володимир Зворикін: винахідник електронно-променевої трубки

Перше використання комп'ютерної графіки пов'язують з ім'ям Дж.Уїтні. Він займався кіновиробництвом в 50-60-х роках минулого століття і вперше використовував комп'ютер для створення титрів до кінофільму. Наступним кроком у своєму розвитку комп'ютерна графіка зобов'язана Айвену Сазерленду, який у 1961 році, ще будучи студентом, створив програму малювання, названу ним Sketchpad (рис. 1.5).

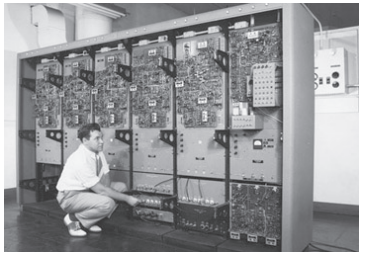


Рис. 1.4. Комп'ютер «Вихрь»

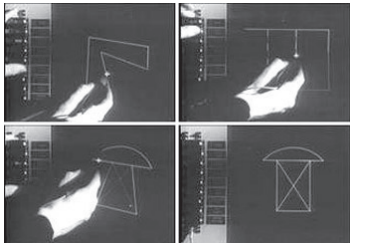


Рис. 1.5. Програма комп'ютерної графіки під назвою Sketchpad

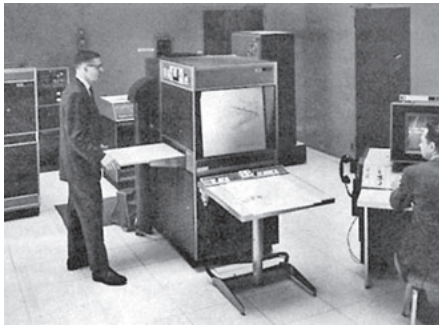
Програма використовувала світлове перо для малювання простих фігур на екрані. Отримані картинки можна було зберігати і відновлювати. У цій програмі був розширений круг основних графічних примітивів, зокрема, окрім ліній і точок був введений прямокутник, який задавався своїми розмірами і розміщенням.

Спочатку комп'ютерна графіка була векторною, тобто зображення формувалося з тонких ліній. Ця особливість була пов'язана з технічною реалізацією комп'ютерних дисплеїв. Надалі ширше вживання отримала растрова графіка, заснована на представленні зображення на екрані у вигляді матриці однорідних елементів (пікселів).

У тому ж 1961 році студент Стів Рассел створив першу комп'ютерну відеогру Spacewar ("Зоряна війна"), а науковий співробітник Bell Labs Едвард Зеджек створив анімацію "Simulation of a two-giro gravity control system".

У зв'язку з успіхами в області комп'ютерної графіки крупні корпорації почали виявляти до неї цікавість, що у свою чергу стимулювало прогрес в області її технічної підтримки.

Під керівництвом Т.Мофетта і Н.Тейлора фірма Itek розробила цифрову електронну креслярську машину. У 1964 році General Motors представила систему автоматизованого проектування DAC-1 (рис. 1.6, а), розроблену спільно з IBM.



а

б

Рис. 1.6. Засоби комп'ютерної графіки:
а - комп'ютер DAC-1; б - графічний термінал IBM-2250

У 1965 році фірма IBM випустила перший комерційний графічний термінал під назвою IBM-2250 (рис. 1.6, б).

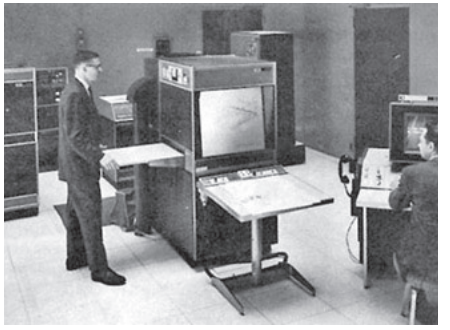
У 1968 році групою під керівництвом М. М. Константинова була створена комп'ютерна математична модель руху кішки. Машина БЕСМ-4 (рис. 1.7, а), виконуючи написану програму вирішення диференціальних рівнянь, малювала мультфільм «Кішечка» (рис.1.7, б), який для того часу був проривом. Для візуалізації використовувався алфавітно-цифровий принтер.

Університет штату Юта стає центром досліджень в області комп'ютерної графіки завдяки Д.Евансу і А.Сазерленду, які в цей

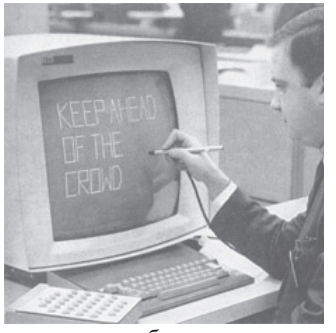
У тому ж 1961 році студент Стів Рассел створив першу комп'ютерну відеогру Spacewar ("Зоряна війна"), а науковий співробітник Bell Labs Едвард Зеджек створив анімацію "Simulation of a two-giro gravity control system".

У зв'язку з успіхами в області комп'ютерної графіки крупні корпорації почали виявляти до неї цікавість, що у свою чергу стимулювало прогрес в області її технічної підтримки.

Під керівництвом Т.Мофетта і Н.Тейлора фірма Itek розробила цифрову електронну креслярську машину. У 1964 році General Motors представила систему автоматизованого проектування DAC-1 (рис. 1.6, а), розроблену спільно з IBM.



а



б

Рис. 1.6. Засоби комп'ютерної графіки:
а - комп'ютер DAC-1; б - графічний термінал IBM-2250

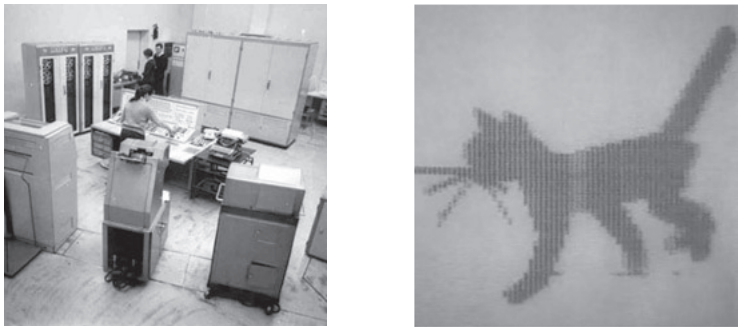
У 1965 році фірма IBM випустила перший комерційний графічний термінал під назвою IBM-2250 (рис. 1.6, б).

У 1968 році групою під керівництвом М. М. Константинова була створена комп'ютерна математична модель руху кішки. Машина БЕСМ-4 (рис. 1.7, а), виконуючи написану програму вирішення диференціальних рівнянь, малювала мультфільм «Кішечка» (рис.1.7, б), який для того часу був проривом. Для візуалізації використовувався алфавітно-цифровий принтер.

Університет штату Юта стає центром досліджень в області комп'ютерної графіки завдяки Д.Евансу і А.Сазерленду, які в цей

час були найпомітнішими фігурами в цій області. Пізніше їх круг став швидко розширюватися. Учнем Сазерленда став Е.Кетмул, майбутній творець алгоритму видалення невидимих поверхонь з використанням z-буфера (1978).

Тут же працювали Дж.Варнок, автор алгоритму видалення невидимих граней на основі розбиття області (1969) і засновник Adobe System (1982) Дж.Кларк, майбутній засновник компанії Silicon Graphics (1982). Всі ці дослідники дуже сильно просунули алгоритмічну сторону комп'ютерної графіки.



а б

Рис. 1.7. Машина БЕСМ-4 (а) та комп'ютерна математична модель руху кішки (б)

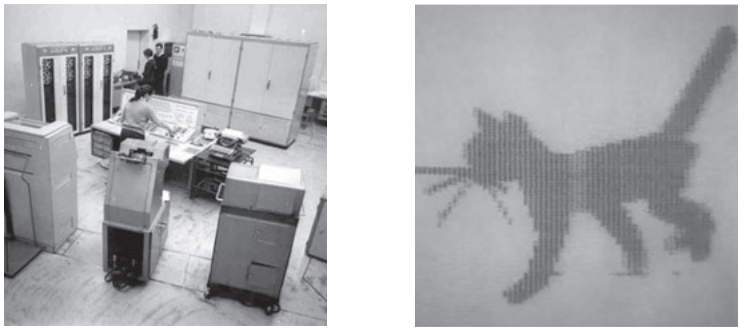
У тому ж 1971 році Гольдштейн і Нагель вперше реалізували метод трасування променів з використанням логічних операцій для формування тривимірних зображень.

У 70-ті роки стався різкий стрибок в розвитку обчислювальної техніки завдяки винаходу мікропроцесора, внаслідок чого почалася мініатюризація комп'ютерів і швидке зростання їх продуктивності. І в цей же час починає інтенсивно розвиватися індустрія комп'ютерних ігор. Одночасно комп'ютерна графіка починає широко використовуватися на телебаченні і в кіноіндустрії. Дж. Лукас створює відділення комп'ютерної графіки на Lucasfilm.

У 1977 році Commodore випустила свій PET (рис.1.8) (персональний електронний діловод), а компанія Apple створила APPLE-II

час були найпомітнішими фігурами в цій області. Пізніше їх круг став швидко розширюватися. Учнем Сазерленда став Е.Кетмул, майбутній творець алгоритму видалення невидимих поверхонь з використанням z-буфера (1978).

Тут же працювали Дж.Варнок, автор алгоритму видалення невидимих граней на основі розбиття області (1969) і засновник Adobe System (1982) Дж.Кларк, майбутній засновник компанії Silicon Graphics (1982). Всі ці дослідники дуже сильно просунули алгоритмічну сторону комп'ютерної графіки.



а б

Рис. 1.7. Машина БЕСМ-4 (а) та комп'ютерна математична модель руху кішки (б)

У тому ж 1971 році Гольдштейн і Нагель вперше реалізували метод трасування променів з використанням логічних операцій для формування тривимірних зображень.

У 70-ті роки стався різкий стрибок в розвитку обчислювальної техніки завдяки винаходу мікропроцесора, внаслідок чого почалася мініатюризація комп'ютерів і швидке зростання їх продуктивності. І в цей же час починає інтенсивно розвиватися індустрія комп'ютерних ігор. Одночасно комп'ютерна графіка починає широко використовуватися на телебаченні і в кіноіндустрії. Дж. Лукас створює відділення комп'ютерної графіки на Lucasfilm.

У 1977 році Commodore випустила свій PET (рис.1.8) (персональний електронний діловод), а компанія Apple створила APPLE-II

(рис.1.9). Поява цих пристроїв викликала змішані відчуття: графіка була жахливою, а процесори повільними, як равлики. Проте ПК стимулювали процес розробки недорогих графічних пристроїв і графічних планшетів.

В середині 70-х років минулого сторіччя графіка продовжує розвиватися у бік все більшої реалістичності зображень. Е.Кетмул у 1974 році створює перші алгоритми текстурування криволінійних поверхонь. У 1975 році з'являється метод закрашення Фонга.

У 1977 році Дж.Блин пропонує алгоритми реалістичного зображення шорстких поверхонь (мікрорельєфів) Ф.Кроу розробляє методи усунення ступінчастого ефекту при зображенні контурів (антиелайзинг). Дж.Брезенхем створює ефективні алгоритми побудови растрових образів відрізків, кіл і еліпсів.

Рис. 1.8. Персональний електронний діловод PET

Рис. 1.9. Комп'ютер APPLE-II

Рівень розвитку обчислювальної техніки до цього часу вже дозволив використовувати "жадібні" алгоритми, що вимагають великих об'ємів пам'яті, і в 1978 році Кетмул пропонує метод z-буфера, в якому використовується область пам'яті для зберігання інформації про "глибину" кожного піксела екранного зображення. У цьому ж році Сайрус і Бек розвивають алгоритми кліппірования (відсікання) ліній. А в 1979 році Кей і Грінберг вперше реалізують зображення напівпрозорої поверхні. У 1980 році Т.Уїттед розробляє загальні принципи трасування променів, що включають віддзеркалення, заломлення, затінювання і методи антиелайзинга.

(рис.1.9). Поява цих пристроїв викликала змішані відчуття: графіка була жахливою, а процесори повільними, як равлики. Проте ПК стимулювали процес розробки недорогих графічних пристроїв і графічних планшетів.

В середині 70-х років минулого сторіччя графіка продовжує розвиватися у бік все більшої реалістичності зображень. Е.Кетмул у 1974 році створює перші алгоритми текстурування криволінійних поверхонь. У 1975 році з'являється метод закрашення Фонга.

У 1977 році Дж.Блин пропонує алгоритми реалістичного зображення шорстких поверхонь (мікрорельєфів) Ф.Кроу розробляє методи усунення ступінчастого ефекту при зображенні контурів (антиелайзинг). Дж.Брезенхем створює ефективні алгоритми побудови растрових образів відрізків, кіл і еліпсів.

Рис. 1.8. Персональний електронний діловод PET

Рис. 1.9. Комп'ютер APPLE-II

Рівень розвитку обчислювальної техніки до цього часу вже дозволив використовувати "жадібні" алгоритми, що вимагають великих об'ємів пам'яті, і в 1978 році Кетмул пропонує метод z-буфера, в якому використовується область пам'яті для зберігання інформації про "глибину" кожного піксела екранного зображення. У цьому ж році Сайрус і Бек розвивають алгоритми кліппірования (відсікання) ліній. А в 1979 році Кей і Грінберг вперше реалізують зображення напівпрозорої поверхні. У 1980 році Т.Уїттед розробляє загальні принципи трасування променів, що включають віддзеркалення, заломлення, затінювання і методи антиелайзинга.

У ці роки комп’ютерна графіка вже міцно упроваджується в кіноіндустрію, розвиваються додатки до інженерних дисциплін. У 1990-ті роки у зв’язку з виникненням мережі Internet в комп’ютерної графіки з’являється ще одна сфера.

Таким чином, у процесі розвитку комп’ютерної графіки можна виділити декілька етапів.

У 1960-1970-ті роки вона формувалася як наукова дисципліна. В цей час розроблялися основні методи і алгоритми: відсікання, растрова розгортка графічних примітивів, зафарбовування узорами, реалістичне зображення просторових сцен (видалення невидимих ліній і граней, трасування променів, випромінюючі поверхні), моделювання освітленості.

У 1980-х графіка розвивається більш як прикладна дисципліна. Розробляються методи її вживання в різних областях людської діяльності. У 1990-ті роки методи комп’ютерної графіки стають основним засобом організації діалогу "людина-комп’ютер" і залишаються такими по теперішній час.

1.3. Види комп’ютерної графіки

Комп’ютерна графіка в даний час сформувалася як наука про апаратне і програмне забезпечення для різноманітних зображень від простих креслень до реалістичних образів природних об’єктів. Комп’ютерна графіка використовується майже в усіх наукових і інженерних дисциплінах для наочності і сприйняття, передачі інформації. Застосовується в медицині, рекламному бізнесі, індустрії розваг і так далі.

Кінцевим продуктом комп’ютерної графіки є зображення. Це зображення може використовуватися в різних сферах, наприклад, воно може бути технічним кресленням, ілюстрацією із зображенням деталі в посібнику з експлуатації, простою діаграмою, архітектурним видом передбачуваної конструкції.

Застосування комп’ютерної графіки

Наукова графіка. Перші комп’ютери використовувалися лише для вирішення наукових і виробничих завдань. Щоб краще зрозуміти отримані результати, проводили їх графічну обробку, будували

У ці роки комп’ютерна графіка вже міцно упроваджується в кіноіндустрію, розвиваються додатки до інженерних дисциплін. У 1990-ті роки у зв’язку з виникненням мережі Internet в комп’ютерної графіки з’являється ще одна сфера.

Таким чином, у процесі розвитку комп’ютерної графіки можна виділити декілька етапів.

У 1960-1970-ті роки вона формувалася як наукова дисципліна. В цей час розроблялися основні методи і алгоритми: відсікання, растрова розгортка графічних примітивів, зафарбовування узорами, реалістичне зображення просторових сцен (видалення невидимих ліній і граней, трасування променів, випромінюючі поверхні), моделювання освітленості.

У 1980-х графіка розвивається більш як прикладна дисципліна. Розробляються методи її вживання в різних областях людської діяльності. У 1990-ті роки методи комп’ютерної графіки стають основним засобом організації діалогу "людина-комп’ютер" і залишаються такими по теперішній час.

1.3. Види комп’ютерної графіки

Комп’ютерна графіка в даний час сформувалася як наука про апаратне і програмне забезпечення для різноманітних зображень від простих креслень до реалістичних образів природних об’єктів. Комп’ютерна графіка використовується майже в усіх наукових і інженерних дисциплінах для наочності і сприйняття, передачі інформації. Застосовується в медицині, рекламному бізнесі, індустрії розваг і так далі.

Кінцевим продуктом комп’ютерної графіки є зображення. Це зображення може використовуватися в різних сферах, наприклад, воно може бути технічним кресленням, ілюстрацією із зображенням деталі в посібнику з експлуатації, простою діаграмою, архітектурним видом передбачуваної конструкції.

Застосування комп’ютерної графіки

Наукова графіка. Перші комп’ютери використовувалися лише для вирішення наукових і виробничих завдань. Щоб краще зрозуміти отримані результати, проводили їх графічну обробку, будували

графіки, діаграми, креслення розрахованих конструкцій. Перші графіки на машині отримували в режимі символьного друку. Потім з'явилися спеціальні пристрої – графічні пристрої (плоттери) для викреслювання креслень і графіків чорнильним пером на папері. Сучасна наукова комп'ютерна графіка дає можливість проводити обчислювальні експерименти з наочним зображенням їх результатів.

Ділова графіка – область комп'ютерної графіки, призначена для наочного зображення різних показників роботи установ. Планові показники, звітна документація, статистичні відомості – ось об'єкти, для яких за допомогою ділової графіки створюються ілюстративні матеріали. Програмні засоби ділової графіки включаються до складу електронних таблиць.

Конструкторська графіка використовується в роботі інженерів-конструкторів, архітекторів, винахідників нової техніки. Цей вид комп'ютерної графіки є обов'язковим елементом САПР (систем автоматизації проектування). Засобами конструкторської графіки можна отримувати як плоскі зображення (проекції, перетини), так і просторові тривимірні зображення.

Ілюстративна графіка – це довільне малювання і креслення на екрані комп'ютера. Пакети ілюстративної графіки відносяться до прикладного програмного забезпечення спільного призначення. Прості програмні засоби ілюстративної графіки називаються графічними редакторами.

Художня і рекламна графіка стала популярною багато в чому завдяки телебаченню. За допомогою комп'ютера створюються рекламні ролики, мультфільми, комп'ютерні ігри, відеоуроки, відеопрезентації. Графічні пакети для цих цілей вимагають великих ресурсів комп'ютера по швидкодії і пам'яті.

Відмінною особливістю цих графічних пакетів є можливість створення реалістичних зображень і "рухомих картинок". Отримання малюнків тривимірних об'єктів, їх повороти, наближення, видалення, деформації пов'язано з великим об'ємом обчислень. Передача освітленості об'єкта, залежно від положення джерела світла, від розташування тіней, від фактури поверхні, вимагає розрахунків, що враховують закони оптики.

графіки, діаграми, креслення розрахованих конструкцій. Перші графіки на машині отримували в режимі символьного друку. Потім з'явилися спеціальні пристрої – графічні пристрої (плоттери) для викреслювання креслень і графіків чорнильним пером на папері. Сучасна наукова комп'ютерна графіка дає можливість проводити обчислювальні експерименти з наочним зображенням їх результатів.

Ділова графіка – область комп'ютерної графіки, призначена для наочного зображення різних показників роботи установ. Планові показники, звітна документація, статистичні відомості – ось об'єкти, для яких за допомогою ділової графіки створюються ілюстративні матеріали. Програмні засоби ділової графіки включаються до складу електронних таблиць.

Конструкторська графіка використовується в роботі інженерів-конструкторів, архітекторів, винахідників нової техніки. Цей вид комп'ютерної графіки є обов'язковим елементом САПР (систем автоматизації проектування). Засобами конструкторської графіки можна отримувати як плоскі зображення (проекції, перетини), так і просторові тривимірні зображення.

Ілюстративна графіка – це довільне малювання і креслення на екрані комп'ютера. Пакети ілюстративної графіки відносяться до прикладного програмного забезпечення спільного призначення. Прості програмні засоби ілюстративної графіки називаються графічними редакторами.

Художня і рекламна графіка стала популярною багато в чому завдяки телебаченню. За допомогою комп'ютера створюються рекламні ролики, мультфільми, комп'ютерні ігри, відеоуроки, відеопрезентації. Графічні пакети для цих цілей вимагають великих ресурсів комп'ютера по швидкодії і пам'яті.

Відмінною особливістю цих графічних пакетів є можливість створення реалістичних зображень і "рухомих картинок". Отримання малюнків тривимірних об'єктів, їх повороти, наближення, видалення, деформації пов'язано з великим об'ємом обчислень. Передача освітленості об'єкта, залежно від положення джерела світла, від розташування тіней, від фактури поверхні, вимагає розрахунків, що враховують закони оптики.

Комп'ютерна анімація – це отримання рухомих зображень на екрані дисплея. Художник створює на екрані малюнки початкового і кінцевого положення рухомих об'єктів, всі проміжні стани розраховує і малює комп'ютер, виконуючи розрахунки, що спираються на математичний опис даного виду руху. Отримані малюнки, що виводяться послідовно на екран з певною частотою, створюють ілюзію руху. Мультимедіа – це об'єднання високоякісного зображення на екрані комп'ютера із звуковим супроводом. Найбільшого поширення системи мультимедіа набули в області навчання, реклами, розваг.

Форми подання графічних даних

Розрізняють три види комп'ютерної графіки. Це растрова графіка, векторна графіка і фрактальна графіка. Вони відрізняються принципами формування зображення під час відображення на екрані монітора або під час друку на папері.

Растрова графіка і піксел

Зображення подається у вигляді великого числа дрібних точок, так званих пікселів. Кожен з них має свій колір, внаслідок чого і утворюється малюнок, аналогічно тому, як з великого числа каменів або скла створюється мозаїка (рис. 1.10).

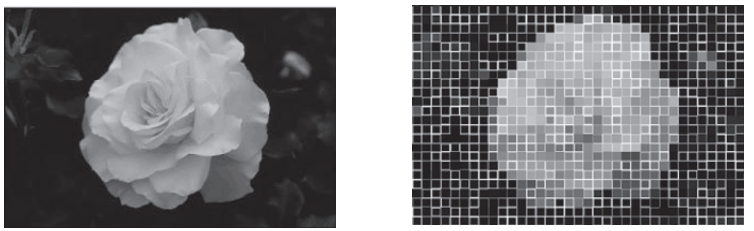


Рис. 1.10 Фотографія та мозаїка

Під час використання растрового способу в комп'ютері під кожен піксел відводиться певне число бітів, назване бітовою глибиною. Кожному кольору відповідає певний двійковий код (тобто код з нулів і одиниць). Наприклад, якщо бітова глибина дорівнює 1, тобто

Комп'ютерна анімація – це отримання рухомих зображень на екрані дисплея. Художник створює на екрані малюнки початкового і кінцевого положення рухомих об'єктів, всі проміжні стани розраховує і малює комп'ютер, виконуючи розрахунки, що спираються на математичний опис даного виду руху. Отримані малюнки, що виводяться послідовно на екран з певною частотою, створюють ілюзію руху. Мультимедіа – це об'єднання високоякісного зображення на екрані комп'ютера із звуковим супроводом. Найбільшого поширення системи мультимедіа набули в області навчання, реклами, розваг.

Форми подання графічних даних

Розрізняють три види комп'ютерної графіки. Це растрова графіка, векторна графіка і фрактальна графіка. Вони відрізняються принципами формування зображення під час відображення на екрані монітора або під час друку на папері.

Растрова графіка і піксел

Зображення подається у вигляді великого числа дрібних точок, так званих пікселів. Кожен з них має свій колір, внаслідок чого і утворюється малюнок, аналогічно тому, як з великого числа каменів або скла створюється мозаїка (рис. 1.10).

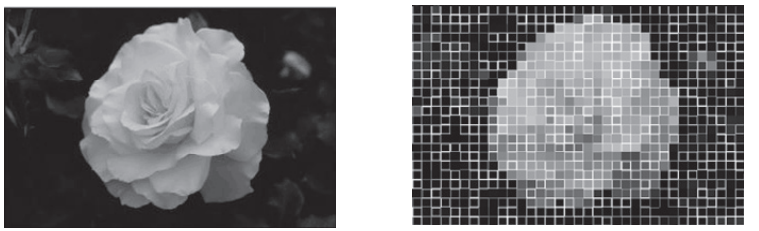


Рис. 1.10 Фотографія та мозаїка

Під час використання растрового способу в комп'ютері під кожен піксел відводиться певне число бітів, назване бітовою глибиною. Кожному кольору відповідає певний двійковий код (тобто код з нулів і одиниць). Наприклад, якщо бітова глибина дорівнює 1, тобто

під кожен піксел відводиться 1 біт, то 0 відповідає чорному кольору, 1 - білому, а зображення може бути тільки чорно-білим.

Як приклад розглянемо чорно-біле (без градацій сірого) зображення стрілки розміром 8x7 (рис. 1.11). Легко підрахувати, який інформаційний об'єм файла потрібний для зберігання цього зображення. Спільна кількість пікселів дорівнює 56. Оскільки використовується всього два кольори, то для зберігання кожного піксела необхідний 1 біт. Таким чином, файл матиме об'єм 56 бітів, або 7 байтів.

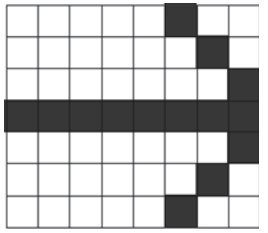


Рис. 1.11. Малюнок стрілки

Якщо бітова глибина дорівнює 2, тобто під кожен піксел відводиться 2 біти, 00 – відповідає чорному кольору, 01 – червоному, 10 – синьому, 11 – зеленому, тобто в малюнку може використовуватися чотири кольори. Зі збільшенням можливої кількості кольорів, збільшується об'єм пам'яті, необхідний для запам'ятовування зображення.

Якість растрового зображення

Якість растрового зображення залежить від розміру зображення (кількість пікселів по горизонталі і по вертикалі) і кількості кольорів, які можна задати для кожного піксела. Растрове зображення дуже чутливе до масштабування (збільшення або зменшення). Під час зменшення растрового зображення декілька сусідніх точок перетворяться в одну, тому втрачається помітність дрібних деталей зображення. Під час збільшення зображення збільшується розмір кожної крапки і з'являється ступінчастий ефект (рис. 1.12).

Рис. 1.12. Растровий малюнок та збільшений растровий малюнок

під кожен піксел відводиться 1 біт, то 0 відповідає чорному кольору, 1 - білому, а зображення може бути тільки чорно-білим.

Як приклад розглянемо чорно-біле (без градацій сірого) зображення стрілки розміром 8x7 (рис. 1.11). Легко підрахувати, який інформаційний об'єм файла потрібний для зберігання цього зображення. Спільна кількість пікселів дорівнює 56. Оскільки використовується всього два кольори, то для зберігання кожного піксела необхідний 1 біт. Таким чином, файл матиме об'єм 56 бітів, або 7 байтів.

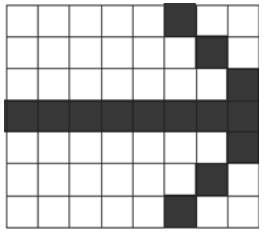


Рис. 1.11. Малюнок стрілки

Якщо бітова глибина дорівнює 2, тобто під кожен піксел відводиться 2 біти, 00 – відповідає чорному кольору, 01 – червоному, 10 – синьому, 11 – зеленому, тобто в малюнку може використовуватися чотири кольори. Зі збільшенням можливої кількості кольорів, збільшується об'єм пам'яті, необхідний для запам'ятовування зображення.

Якість растрового зображення

Якість растрового зображення залежить від розміру зображення (кількість пікселів по горизонталі і по вертикалі) і кількості кольорів, які можна задати для кожного піксела. Растрове зображення дуже чутливе до масштабування (збільшення або зменшення). Під час зменшення растрового зображення декілька сусідніх точок перетворяться в одну, тому втрачається помітність дрібних деталей зображення. Під час збільшення зображення збільшується розмір кожної крапки і з'являється ступінчастий ефект (рис. 1.12).

Рис. 1.12. Растровий малюнок та збільшений растровий малюнок

Растрові зображення мають безліч характеристик, які повинні бути організовані і фіксовані комп'ютером. Розміри зображення і розташування пікселів у ньому – це дві основні характеристики, які файл растрових зображень повинен зберегти, щоб створити картинку.

Навіть якщо зіпсована інформація про колір будь-якого піксела і будь-яких інших характеристик, комп'ютер все одно зможе відтворити версію малюнка, якщо знатиме, як розташовані всі його пікселі.

Піксел сам по собі не має ніякого розміру, він усього лише область пам'яті комп'ютера, що зберігає інформацію про колір і яскравість. Оскільки розміри зображення зберігаються окремо, пікселі запам'ятовуються один за іншим, як звичайний блок даних. Комп'ютеру не доводиться зберігати окремі позиції, він всього лише створює сітку за розмірами зображення, а потім заповнює її піксел за пікселом. Це найпростіший спосіб зберігання даного растрового зображення, але не найефективніший з погляду використання комп'ютерного часу і пам'яті. Ефективніший спосіб полягає в тому, щоб зберегти тільки кількість чорних і білих пікселів у будь-якому рядку. Цей метод стискує дані, які використовують растрові зображення. В цьому випадку вони займають менше пам'яті комп'ютера.

Роздільна здатність растра

Взагалі, розмір зображень можна розуміти по-різному. По-перше, реальний розмір картинки і по-друге, фізичний розмір даних зображення.

Будь-яке комп'ютерне растрове зображення вимірюється в пікселах. Як було сказано, пікселі - це умовні квадратики, на які розбивається реальне зображення. В цьому випадку вказується кількість пікселів у горизонтальній лінії і вертикальній. Наприклад, "растр 2048×1536 пікселів" означає, що зображення являє собою матрицю 2048 пікселів шириною і 1536 висотою. Це все добре, але нормальні люди рідко оперують одиницею виміру "піксел", вважаючи за краще використовувати доступніші міліметри або сантиметри. Для того щоб співвіднести ці 2 типи розміру, використовують одиниці типу dpi - dots per inch, тобто точка на дюйм. Вона визначає, в скільки

Растрові зображення мають безліч характеристик, які повинні бути організовані і фіксовані комп'ютером. Розміри зображення і розташування пікселів у ньому – це дві основні характеристики, які файл растрових зображень повинен зберегти, щоб створити картинку.

Навіть якщо зіпсована інформація про колір будь-якого піксела і будь-яких інших характеристик, комп'ютер все одно зможе відтворити версію малюнка, якщо знатиме, як розташовані всі його пікселі.

Піксел сам по собі не має ніякого розміру, він усього лише область пам'яті комп'ютера, що зберігає інформацію про колір і яскравість. Оскільки розміри зображення зберігаються окремо, пікселі запам'ятовуються один за іншим, як звичайний блок даних. Комп'ютеру не доводиться зберігати окремі позиції, він всього лише створює сітку за розмірами зображення, а потім заповнює її піксел за пікселом. Це найпростіший спосіб зберігання даного растрового зображення, але не найефективніший з погляду використання комп'ютерного часу і пам'яті. Ефективніший спосіб полягає в тому, щоб зберегти тільки кількість чорних і білих пікселів у будь-якому рядку. Цей метод стискує дані, які використовують растрові зображення. В цьому випадку вони займають менше пам'яті комп'ютера.

Роздільна здатність растра

Взагалі, розмір зображень можна розуміти по-різному. По-перше, реальний розмір картинки і по-друге, фізичний розмір даних зображення.

Будь-яке комп'ютерне растрове зображення вимірюється в пікселах. Як було сказано, пікселі - це умовні квадратики, на які розбивається реальне зображення. В цьому випадку вказується кількість пікселів у горизонтальній лінії і вертикальній. Наприклад, "растр 2048×1536 пікселів" означає, що зображення являє собою матрицю 2048 пікселів шириною і 1536 висотою. Це все добре, але нормальні люди рідко оперують одиницею виміру "піксел", вважаючи за краще використовувати доступніші міліметри або сантиметри. Для того щоб співвіднести ці 2 типи розміру, використовують одиниці типу dpi - dots per inch, тобто точка на дюйм. Вона визначає, в скільки

пікселів перетвориться лінія завдовжки 1 дюйм. Як правило, використовуються одиниці від 100 dpi до 2400 dpi. 100 dpi – це дуже посередня якість, непридатна для будь-якої професійної діяльності. Лазерні принтери зазвичай мають від 300 до 600 dpi (адже растрові зображення можуть не лише оцифровуватися для комп'ютера, але і підлягати зворотній операції – виводитися на традиційний носій типу паперу). Такий же дозвіл забезпечують сканери домашнього рівня. Найдосконаліші сканери і фотонабірні апарати забезпечують якість близько 2400dpi і вище.

Проте варто розібратися, що стоїть за цими dpi. Зрозуміло, що чим краща якість і більше dpi, тим більш точне зображення ми отримуємо. Але слід співвідносити необхідний дозвіл і розмір цифрового образу, що вийшов. У деяких користувачів завжди виникає спокуса встановити на сканері максимальний доступний дозвіл. Наприклад, 600 dpi. Хай ми скануємо стандартний лист формату А4. Кольорові сканери працюють у форматі RGB і видають зображення глибиною 24 біти, тобто 3 байти на піксел. А4 – це 210×297 міліметрів, тобто приблизно 8×11 дюймів. У результаті ми отримуємо матрицю (8×600) на (11×600) пікселів, тобто близько 4800×6600 пікселів. Перемноживши ці дві лінійні величини і помноживши їх на 3 (стільки займає 1 піксел в пам'яті), ми отримаємо 95040000 байтів, тобто 90,63 мегабайта. Стільки займає звичайний кольоровий лист формату А4, що сканує з пристойним дозволом. Якщо ж ми отримуємо сканувати на хорошому сканері невеликий плакат 90×60 сантиметрів, то буде потрібно дійсно потужні ресурси.

Файли растрової графіки займають велику кількість пам'яті комп'ютера. Найбільше впливають на кількість пам'яті, зайнятої растровим зображенням, три факти:

- розмір зображення (кількість пікселів);
- бітова глибина кольору;
- формат файла, що використовується для зберігання зображення.

Чим більше в зображенні пікселів, тим більше розмір файла. Чим більше бітів використовується в пікселі, тим більше буде файл. Розмір файла растрової графіки сильно залежить від формату вибраного для зберігання зображення. За інших рівних умов, таких як розміри

пікселів перетвориться лінія завдовжки 1 дюйм. Як правило, використовуються одиниці від 100 dpi до 2400 dpi. 100 dpi – це дуже посередня якість, непридатна для будь-якої професійної діяльності. Лазерні принтери зазвичай мають від 300 до 600 dpi (адже растрові зображення можуть не лише оцифровуватися для комп'ютера, але і підлягати зворотній операції – виводитися на традиційний носій типу паперу). Такий же дозвіл забезпечують сканери домашнього рівня. Найдосконаліші сканери і фотонабірні апарати забезпечують якість близько 2400dpi і вище.

Проте варто розібратися, що стоїть за цими dpi. Зрозуміло, що чим краща якість і більше dpi, тим більш точне зображення ми отримуємо. Але слід співвідносити необхідний дозвіл і розмір цифрового образу, що вийшов. У деяких користувачів завжди виникає спокуса встановити на сканері максимальний доступний дозвіл. Наприклад, 600 dpi. Хай ми скануємо стандартний лист формату А4. Кольорові сканери працюють у форматі RGB і видають зображення глибиною 24 біти, тобто 3 байти на піксел. А4 – це 210×297 міліметрів, тобто приблизно 8×11 дюймів. У результаті ми отримуємо матрицю (8×600) на (11×600) пікселів, тобто близько 4800×6600 пікселів. Перемноживши ці дві лінійні величини і помноживши їх на 3 (стільки займає 1 піксел в пам'яті), ми отримаємо 95040000 байтів, тобто 90,63 мегабайта. Стільки займає звичайний кольоровий лист формату А4, що сканує з пристойним дозволом. Якщо ж ми отримуємо сканувати на хорошому сканері невеликий плакат 90×60 сантиметрів, то буде потрібно дійсно потужні ресурси.

Файли растрової графіки займають велику кількість пам'яті комп'ютера. Найбільше впливають на кількість пам'яті, зайнятої растровим зображенням, три факти:

- розмір зображення (кількість пікселів);
- бітова глибина кольору;
- формат файла, що використовується для зберігання зображення.

Чим більше в зображенні пікселів, тим більше розмір файла. Чим більше бітів використовується в пікселі, тим більше буде файл. Розмір файла растрової графіки сильно залежить від формату вибраного для зберігання зображення. За інших рівних умов, таких як розміри

зображення і бітова глибина, істотне значення має схема стискування зображення. Наприклад, BMP файл має, як правило, великі розміри, в порівнянні з файлами PCX і GIF, які, у свою чергу, більше JPEG файла.

Багато файлів зображень володіють власними схемами стискування, вони також можуть містити додаткові дані короткого опису зображення для попереднього перегляду.

Сильні сторони растрової графіки

- Растрова графіка ефективно представляє реальні образи. Реальний світ складається з мільярдів найдрібніших об'єктів, і людське око якраз пристосоване для сприйняття величезного набору дискретних елементів, створюючи предмети. На своєму високому рівні якості зображення виглядають цілком реально, подібно до того, як виглядають фотографії порівняно з малюнками. Це вірно тільки для дуже деталізованих зображень, зазвичай отримуваних скануванням фотографій.
- Пристрої виводу, такі як лазерні принтери, для створення зображень використовують набори точок. Растрові зображення легко друкуються, тому що комп'ютерам легко управляти растровим пристроєм виводу.

Слабкі сторони растрової графіки

- Погане масштабування. При зменшенні зображення декілька сусідніх точок перетворяться в одну, тому втрачаються дрібні подробиці. При збільшенні масштабу відбувається збільшення розміру кожної точки, через що з'являється ступінчастий ефект.
- Великий розмір файла, оскільки включені дані про кожен піксел зображення.
- Споживання значних ресурсів комп'ютера при редагуванні растрових зображень.

Векторна графіка

На відміну від растра векторне зображення являє собою математичний опис. Растр – це дискретна структура, тобто завжди можна виділити певні елементи. При використанні векторного уявлення зображення являє собою базу даних описів примітивів. Тобто, у складі

зображення і бітова глибина, істотне значення має схема стискування зображення. Наприклад, BMP файл має, як правило, великі розміри, в порівнянні з файлами PCX і GIF, які, у свою чергу, більше JPEG файла.

Багато файлів зображень володіють власними схемами стискування, вони також можуть містити додаткові дані короткого опису зображення для попереднього перегляду.

Сильні сторони растрової графіки

- Растрова графіка ефективно представляє реальні образи. Реальний світ складається з мільярдів найдрібніших об'єктів, і людське око якраз пристосоване для сприйняття величезного набору дискретних елементів, створюючи предмети. На своєму високому рівні якості зображення виглядають цілком реально, подібно до того, як виглядають фотографії порівняно з малюнками. Це вірно тільки для дуже деталізованих зображень, зазвичай отримуваних скануванням фотографій.
- Пристрої виводу, такі як лазерні принтери, для створення зображень використовують набори точок. Растрові зображення легко друкуються, тому що комп'ютерам легко управляти растровим пристроєм виводу.

Слабкі сторони растрової графіки

- Погане масштабування. При зменшенні зображення декілька сусідніх точок перетворяться в одну, тому втрачаються дрібні подробиці. При збільшенні масштабу відбувається збільшення розміру кожної точки, через що з'являється ступінчастий ефект.
- Великий розмір файла, оскільки включені дані про кожен піксел зображення.
- Споживання значних ресурсів комп'ютера при редагуванні растрових зображень.

Векторна графіка

На відміну від растра векторне зображення являє собою математичний опис. Растр – це дискретна структура, тобто завжди можна виділити певні елементи. При використанні векторного уявлення зображення являє собою базу даних описів примітивів. Тобто, у складі

зображення можуть бути відрізки, кола, овали, точки, криві Безьє і так далі. А зображення вдаватиме із себе масив описів – щось типу: відрізок (20,20-100,80); коло (50,40-30); крива Без'є (20,20-50,30-100,50).

Природно, що деталі зберігання цього масиву в пам'яті залежать від конкретної програми. І найголовніша перевага векторного зображення полягає в тому, що воно є аналітично заданим і у будь-який момент ми можемо змінити будь-який з параметрів будь-якої з його складових. Крім того, над векторним зображенням дуже зручно проводити математичні, по суті, операції типу збільшення (scaling), повороту (rotation), нелінійних перетворень (deformations) і так далі.

У випадку з растровим зображенням, унаслідок його дискретності і тому втрати частки інформації, відбуватимуться необоротні втрати якості зображення. І якщо з растровим зображенням ми проведемо велику кількість операцій, то зрештою воно може стати непридатним до використання в поліграфії або інших галузях, які вимагають хорошої якості.

Приклад, що показує ефект векторної графіки при збільшенні, зображено на рис. 1.13.

а б в

Рис. 1.13. Приклад, що показує ефект векторної графіки:
а - початкове векторне зображення; б - ілюстрація, збільшена в 8 разів як векторне зображення; в - ілюстрація, збільшена в 8 разів як растрове зображення

Крім того, векторні зображення, як правило, займають менше пам'яті, ніж растрові. Адже набагато простіше сказати "Коло радіусом R і центром з координатами X, Y", чим виписувати всі піксели, на які воно розбивається при оцифровці растра.

зображення можуть бути відрізки, кола, овали, точки, криві Безьє і так далі. А зображення вдаватиме із себе масив описів – щось типу: відрізок (20,20-100,80); коло (50,40-30); крива Без'є (20,20-50,30-100,50).

Природно, що деталі зберігання цього масиву в пам'яті залежать від конкретної програми. І найголовніша перевага векторного зображення полягає в тому, що воно є аналітично заданим і у будь-який момент ми можемо змінити будь-який з параметрів будь-якої з його складових. Крім того, над векторним зображенням дуже зручно проводити математичні, по суті, операції типу збільшення (scaling), повороту (rotation), нелінійних перетворень (deformations) і так далі.

У випадку з растровим зображенням, унаслідок його дискретності і тому втрати частки інформації, відбуватимуться необоротні втрати якості зображення. І якщо з растровим зображенням ми проведемо велику кількість операцій, то зрештою воно може стати непридатним до використання в поліграфії або інших галузях, які вимагають хорошої якості.

Приклад, що показує ефект векторної графіки при збільшенні, зображено на рис. 1.13.

а б в

Рис. 1.13. Приклад, що показує ефект векторної графіки:
а - початкове векторне зображення; б - ілюстрація, збільшена в 8 разів як векторне зображення; в - ілюстрація, збільшена в 8 разів як растрове зображення

Крім того, векторні зображення, як правило, займають менше пам'яті, ніж растрові. Адже набагато простіше сказати "Коло радіусом R і центром з координатами X, Y", чим виписувати всі піксели, на які воно розбивається при оцифровці растра.

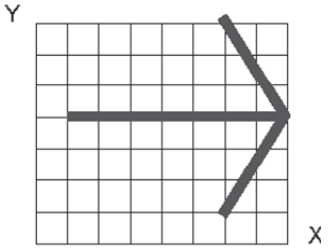


Рис. 1.14. Приклад стрілки у векторній графіці

Стрілка у векторному форматі буде задана за допомогою трьох ліній (рис. 1.14): лінія (1,4) - (8,4); лінія (6,7) - (8,4); лінія (6,1) - (8,4).

За цими даними відповідні програми побудують потрібну фігуру на екрані дисплея.

Проте растрове зображення має великі переваги при роботі з фотореалістичними об’єктами, наприклад сценами природи або фотографіями людей. Річ у тому, що наш світ по ідеї растровий. І його об’єкти важко представити у векторному, тобто математичному по суті, уявленні. Для прикладу візьмемо звичайну растрову картинку (рис. 1.15).

Рис. 1.15. Зображення фотореалістичних об’єктів

Тепер переведемо її у векторний формат за допомогою якої-небудь програми, наприклад, Adobe StreamLine. При цьому відзначимо, що розмір зображення змінився з 17 до 400 Кб. При цьому зображення прийме вигляд, зображений на рис. 1.16.

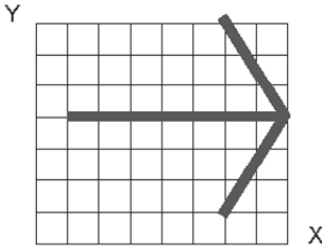


Рис. 1.14. Приклад стрілки у векторній графіці

Стрілка у векторному форматі буде задана за допомогою трьох ліній (рис. 1.14): лінія (1,4) - (8,4); лінія (6,7) - (8,4); лінія (6,1) - (8,4).

За цими даними відповідні програми побудують потрібну фігуру на екрані дисплея.

Проте растрове зображення має великі переваги при роботі з фотореалістичними об’єктами, наприклад сценами природи або фотографіями людей. Річ у тому, що наш світ по ідеї растровий. І його об’єкти важко представити у векторному, тобто математичному по суті, уявленні. Для прикладу візьмемо звичайну растрову картинку (рис. 1.15).

Рис. 1.15. Зображення фотореалістичних об’єктів

Тепер переведемо її у векторний формат за допомогою якої-небудь програми, наприклад, Adobe StreamLine. При цьому відзначимо, що розмір зображення змінився з 17 до 400 Кб. При цьому зображення прийме вигляд, зображений на рис. 1.16.

Рис. 1.16. Приклад переведення у векторний формат

Іноді це може бути корисним, але, звичайно ж, це зображення фотореалістичним вже не назвеш.

Тут ми підійшли до питання про перетворення зображення між двома форматами. Перетворення з векторного зображення в растрове особливих проблем не викликає – адже при роботі у векторному редакторі ви все одно бачите результат у растровому вигляді (монітор є растровим пристроєм). При цьому векторне зображення просто відбивається на віртуальному екрані, а потім оцифровується. Це однозначний процес.

З перетворенням з растрового зображення у векторне справа йде гірше. Річ у тому, що це процес неоднозначний, тобто тут вже з'являється елемент емпірики і нам у кожному конкретному випадку належить визначити, як краще представити вектором даний ланцюжок пікселів. Оскільки краще – поняття суб'єктивне, то і результати можуть бути дуже різними – залежно від прийнятої ідеї векторизації. Але все-таки потужні програми векторизації справляються з техногенними малюнками достатньо непогано. Ну а із звичайною картиною гірського озера результат набагато гірший.

Колір у векторній графіці

Для різних векторних форматів характерні різні колірні можливості. Прості формати, які можуть не містити взагалі ніякої інформації

Рис. 1.16. Приклад переведення у векторний формат

Іноді це може бути корисним, але, звичайно ж, це зображення фотореалістичним вже не назвеш.

Тут ми підійшли до питання про перетворення зображення між двома форматами. Перетворення з векторного зображення в растрове особливих проблем не викликає – адже при роботі у векторному редакторі ви все одно бачите результат у растровому вигляді (монітор є растровим пристроєм). При цьому векторне зображення просто відбивається на віртуальному екрані, а потім оцифровується. Це однозначний процес.

З перетворенням з растрового зображення у векторне справа йде гірше. Річ у тому, що це процес неоднозначний, тобто тут вже з'являється елемент емпірики і нам у кожному конкретному випадку належить визначити, як краще представити вектором даний ланцюжок пікселів. Оскільки краще – поняття суб'єктивне, то і результати можуть бути дуже різними – залежно від прийнятої ідеї векторизації. Але все-таки потужні програми векторизації справляються з техногенними малюнками достатньо непогано. Ну а із звичайною картиною гірського озера результат набагато гірший.

Колір у векторній графіці

Для різних векторних форматів характерні різні колірні можливості. Прості формати, які можуть не містити взагалі ніякої інформації

про колір, використовують колір за замовчанням тих пристроїв, на які вони виводяться, інші формати здатні зберігати дані про повний набір кольорів. Яку б колірну модель не застосовував би векторний формат, на розмір файла, він не впливає, окрім тих випадків, коли файл містить растрові образи.

У звичайних векторних об'єктах значення кольору відноситься до всього об'єкта в цілому. Колір об'єкта зберігається у вигляді частки його векторного опису. Деякі векторні файли можуть створити растровий ескіз зображень, що зберігаються в них. Ці растрові картинки, що інколи називаються короткими описами зображень, зазвичай є ескізами векторних малюнків у цілому. Короткий опис зображення особливо корисний у ситуаціях, коли ви не хочете відкривати весь файл, щоб подивитися, що в ньому зберігається, або коли ви не можете бачити векторний малюнок під час його використання.

Перша ситуація виникає, коли вам необхідно знайти файл за допомогою однієї з багатьох спеціально розроблених для цього програм. Для полегшення пошуку потрібного векторного файла такі програми можуть прочитувати растровий ескіз зображення та інші характеристики, наприклад, векторний формат, час створення, бітову глибину зображення і так далі. Друга ситуація виникає, коли в якому-небудь видавничому пакеті поміщається на сторінку векторний малюнок. Зображення, яке ви побачите, буде растровим ескізом справжнього векторного малюнка, для якого не можна змінити розмір, обрізати або якось інакше обробити. За ескізи зображення доводиться розплачуватися пам'яттю, оскільки ескізи – це растрова версія малюнків, а растрові дані використовують багато пам'яті комп'ютера.

Сильні сторони векторної графіки

- Векторна графіка використовує всі переваги роздільної здатності будь-якого пристрою виводу. Це дозволяє змінювати розміри векторного малюнка без утрати його якості (рис. 1.17). Векторні команди просто поступають на пристрій виводу та повідомляють, що необхідно намалювати об'єкт заданого розміру, використовуючи стільки точок, скільки можливо. Іншими словами, чим більше точок

про колір, використовують колір за замовчанням тих пристроїв, на які вони виводяться, інші формати здатні зберігати дані про повний набір кольорів. Яку б колірну модель не застосовував би векторний формат, на розмір файла, він не впливає, окрім тих випадків, коли файл містить растрові образи.

У звичайних векторних об'єктах значення кольору відноситься до всього об'єкта в цілому. Колір об'єкта зберігається у вигляді частки його векторного опису. Деякі векторні файли можуть створити растровий ескіз зображень, що зберігаються в них. Ці растрові картинки, що інколи називаються короткими описами зображень, зазвичай є ескізами векторних малюнків у цілому. Короткий опис зображення особливо корисний у ситуаціях, коли ви не хочете відкривати весь файл, щоб подивитися, що в ньому зберігається, або коли ви не можете бачити векторний малюнок під час його використання.

Перша ситуація виникає, коли вам необхідно знайти файл за допомогою однієї з багатьох спеціально розроблених для цього програм. Для полегшення пошуку потрібного векторного файла такі програми можуть прочитувати растровий ескіз зображення та інші характеристики, наприклад, векторний формат, час створення, бітову глибину зображення і так далі. Друга ситуація виникає, коли в якому-небудь видавничому пакеті поміщається на сторінку векторний малюнок. Зображення, яке ви побачите, буде растровим ескізом справжнього векторного малюнка, для якого не можна змінити розмір, обрізати або якось інакше обробити. За ескізи зображення доводиться розплачуватися пам'яттю, оскільки ескізи – це растрова версія малюнків, а растрові дані використовують багато пам'яті комп'ютера.

Сильні сторони векторної графіки

- Векторна графіка використовує всі переваги роздільної здатності будь-якого пристрою виводу. Це дозволяє змінювати розміри векторного малюнка без утрати його якості (рис. 1.17). Векторні команди просто поступають на пристрій виводу та повідомляють, що необхідно намалювати об'єкт заданого розміру, використовуючи стільки точок, скільки можливо. Іншими словами, чим більше точок

зможе використовувати пристрій виводу для створення об’єкта, тим краще він виглядатиме. Растровий формат файла точно визначає, скільки необхідно створити пікселів, і ця кількість змінюється разом з роздільною здатністю пристрою виводу. Наприклад, при збільшенні роздільної здатності розмір растрового кола зменшується, оскільки зменшується розмір точки складових піксела

Рис. 1.17. Результат збільшення векторного зображення

• Векторна графіка має ще одну важливу перевагу: тут можна редагувати окремі частки малюнка, не впливаючи на останні. Об’єкти на малюнку можуть перекриватися без жодної дії один на одного.

• Векторне зображення, що не містить растрових об’єктів, займає відносно невелике місце в пам’яті комп’ютера.

Слабкі сторони векторної графіки

• Умовність отримуваних зображень. Для побудови реалістичних зображень знадобилося б величезне число примітивів. Наприклад, розмір векторного файла, що описує фотографію, буде більшим, ніж растровий. Тому векторний формат застосовується для опису лінійних малюнків і він ідеально підходить для креслень.

• Складнощі при суцільному заповненні фігур кольором: менша кількість кольорів, менша швидкість (у порівнянні з растровими пристроями).

зможе використовувати пристрій виводу для створення об’єкта, тим краще він виглядатиме. Растровий формат файла точно визначає, скільки необхідно створити пікселів, і ця кількість змінюється разом з роздільною здатністю пристрою виводу. Наприклад, при збільшенні роздільної здатності розмір растрового кола зменшується, оскільки зменшується розмір точки складових піксела

Рис. 1.17. Результат збільшення векторного зображення

• Векторна графіка має ще одну важливу перевагу: тут можна редагувати окремі частки малюнка, не впливаючи на останні. Об’єкти на малюнку можуть перекриватися без жодної дії один на одного.

• Векторне зображення, що не містить растрових об’єктів, займає відносно невелике місце в пам’яті комп’ютера.

Слабкі сторони векторної графіки

• Умовність отримуваних зображень. Для побудови реалістичних зображень знадобилося б величезне число примітивів. Наприклад, розмір векторного файла, що описує фотографію, буде більшим, ніж растровий. Тому векторний формат застосовується для опису лінійних малюнків і він ідеально підходить для креслень.

• Складнощі при суцільному заповненні фігур кольором: менша кількість кольорів, менша швидкість (у порівнянні з растровими пристроями).

Фрактальна графіка

Фрактальна графіка, як і векторна, – обчислювана, але відрізняється від неї тим, що ніякі об'єкти в пам'яті комп'ютера не зберігаються. Зображення будується за рівнянням (або за системою рівнянь), тому нічого, окрім формули, зберігати не треба. Змінивши коефіцієнти в рівнянні, можна отримати абсолютно іншу картину[14].

Здатність фрактальної графіки моделювати образи живої природи обчислювальним шляхом часто використовують для автоматичної генерації незвичайних ілюстрацій.

Приклади фрактальних зображень наведено на рис. 1.18.

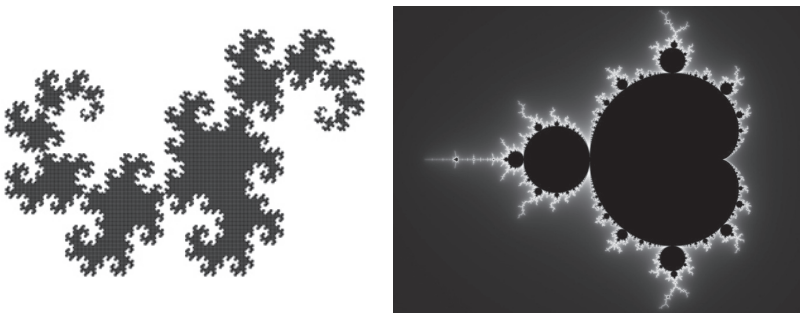


Рис. 1.18. Приклади фрактальних зображень

Фрактали дозволяють описувати цілі класи зображень, для детального опису яких потрібно відносно мало пам'яті. З іншого боку, фрактали слабо застосовуються до зображень поза цими класами. Зараз найбільш широке розповсюдження використання фракталів відмічається в області стиснення графічних зображень.

Тривимірна графіка

Окремим видом комп'ютерної графіки виступає тривимірна графіка. Тривимірна графіка оперує з об'єктами в тривимірному просторі. Зазвичай результати є плоскою картинкою, проекцією (рис. 1.19).

Фрактальна графіка

Фрактальна графіка, як і векторна, – обчислювана, але відрізняється від неї тим, що ніякі об'єкти в пам'яті комп'ютера не зберігаються. Зображення будується за рівнянням (або за системою рівнянь), тому нічого, окрім формули, зберігати не треба. Змінивши коефіцієнти в рівнянні, можна отримати абсолютно іншу картину[14].

Здатність фрактальної графіки моделювати образи живої природи обчислювальним шляхом часто використовують для автоматичної генерації незвичайних ілюстрацій.

Приклади фрактальних зображень наведено на рис. 1.18.

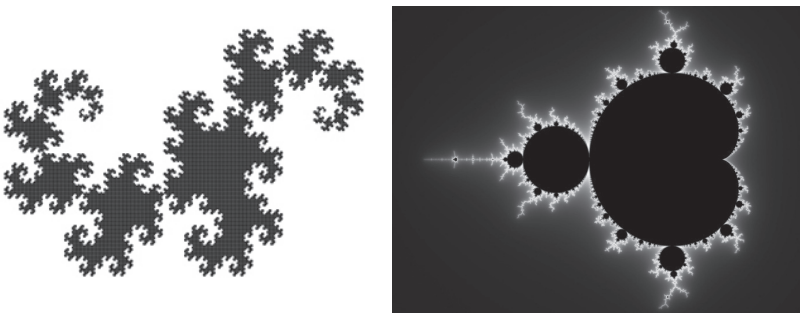


Рис. 1.18. Приклади фрактальних зображень

Фрактали дозволяють описувати цілі класи зображень, для детального опису яких потрібно відносно мало пам'яті. З іншого боку, фрактали слабо застосовуються до зображень поза цими класами. Зараз найбільш широке розповсюдження використання фракталів відмічається в області стиснення графічних зображень.

Тривимірна графіка

Окремим видом комп'ютерної графіки виступає тривимірна графіка. Тривимірна графіка оперує з об'єктами в тривимірному просторі. Зазвичай результати є плоскою картинкою, проекцією (рис. 1.19).

Тривимірна комп’ютерна графіка широко використовується в кіно, комп’ютерних іграх. У тривимірній комп’ютерній графіці всі об’єкти зазвичай представляються як набір поверхонь.

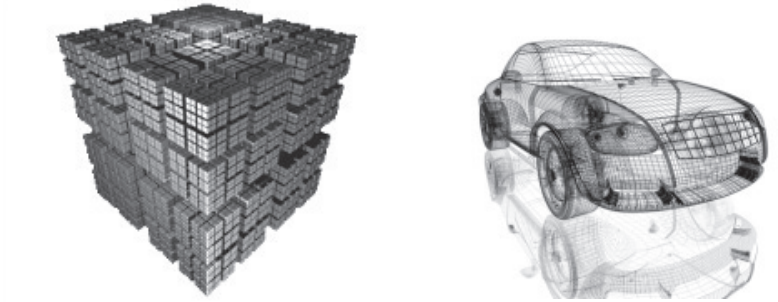


Рис. 1.19. Приклади тривимірних зображень

Мінімальну поверхню називають полігоном. Як полігон зазвичай вибирають трикутники. Всіма візуальними перетвореннями в 3d-графіці управляють матриці (див. також: афінне перетворення в лінійній алгебрі). У комп’ютерній графіці використовуються три види матриць: матриця повороту, матриця зрушення та матриця масштабування.

Будь-який полігон можна подати у вигляді набору з координат його вершин. Так, у трикутника буде 3 вершини. Координати кожної вершини є вектором (x, y, z). Помноживши вектор на відповідну матрицю, ми отримаємо новий вектор. Зробивши таке перетворення зі всіма вершинами полігону, отримаємо новий полігон, а перетворивши всі полігони, отримаємо новий об’єкт, повернутий або зсунутий відносно початкового.

Все різноманіття властивостей у комп’ютерному моделюванні зводиться до візуалізації поверхні, тобто до розрахунку коефіцієнта прозорості поверхні і кута заломлення променів світла на границі матеріалу і навколишнього простору.

Властивості поверхні описуються у створюваних масивах текстур, в яких містяться дані про міру прозорості матеріалу, коефіцієнт заломлення, колір у кожній точці, колір відблиску, його ширину та різкість і ін.

Тривимірна комп’ютерна графіка широко використовується в кіно, комп’ютерних іграх. У тривимірній комп’ютерній графіці всі об’єкти зазвичай представляються як набір поверхонь.

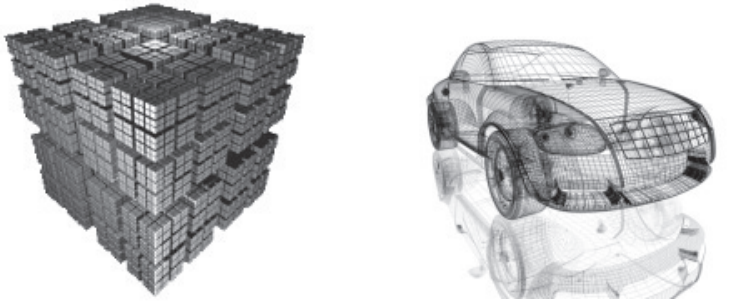


Рис. 1.19. Приклади тривимірних зображень

Мінімальну поверхню називають полігоном. Як полігон зазвичай вибирають трикутники. Всіма візуальними перетвореннями в 3d-графіці управляють матриці (див. також: афінне перетворення в лінійній алгебрі). У комп’ютерній графіці використовуються три види матриць: матриця повороту, матриця зрушення та матриця масштабування.

Будь-який полігон можна подати у вигляді набору з координат його вершин. Так, у трикутника буде 3 вершини. Координати кожної вершини є вектором (x, y, z). Помноживши вектор на відповідну матрицю, ми отримаємо новий вектор. Зробивши таке перетворення зі всіма вершинами полігону, отримаємо новий полігон, а перетворивши всі полігони, отримаємо новий об’єкт, повернутий або зсунутий відносно початкового.

Все різноманіття властивостей у комп’ютерному моделюванні зводиться до візуалізації поверхні, тобто до розрахунку коефіцієнта прозорості поверхні і кута заломлення променів світла на границі матеріалу і навколишнього простору.

Властивості поверхні описуються у створюваних масивах текстур, в яких містяться дані про міру прозорості матеріалу, коефіцієнт заломлення, колір у кожній точці, колір відблиску, його ширину та різкість і ін.

1.4. Колірні моделі

Людське око може сприймати велику кількість кольорів, у той час як монітор та принтер у змозі відтворювати лише обмежену частину цього діапазону. Причому діапазон кольорів, що відтворюються, та спосіб їх відображення для монітора та принтера теж різні. У зв'язку з необхідністю опису різних фізичних процесів відтворення кольору, були розроблені різноманітні кольорові моделі. В основі створення кольорових моделей лежить використання універсальних мов, які дозволяють реалізувати способи точного опису кольору за допомогою стандартних математичних виразів.

В сучасних комп'ютерних програмах маніпулювання з кольором здійснюється за допомогою *колірних моделей* та *режимів*. *Колірні моделі* надають засоби для концептуального та кількісного опису кольору. *Режим* – це спосіб реалізації певної колірної моделі в рамках конкретної графічної програми.

Поняття колірної моделі

Колірні моделі (color model) використовуються для математичного опису певних колірних областей спектра. Більшість комп'ютерних колірних моделей засновано на використанні трьох основних кольорів, що відповідає сприйняттю кольору людським оком. Кожному основному кольору присвоюється певне значення цифрового коду, після чого вся решта кольорів визначаються як комбінації основних кольорів.

Незважаючи на те, що колірні моделі дозволяють подати колір математично, це завжди буде вважатись недосконалим у зв'язку з нашим сприйняттям. Однак їх зручно використовувати в комп'ютерних програмах для однозначного визначення кольору. Так, якщо послати на монітор колірний сигнал R255 G000 B255, то на будь-якому моніторі повинен з'явитися один і той же колір (у даному випадку пурпуровий).

Незалежно від того, що лежить в її основі, будь-яка модель повинна відповідати таким вимогам:

– реалізовувати визначення кольору деяким стандартним способом, який не залежить від можливостей певного пристрою;

1.4. Колірні моделі

Людське око може сприймати велику кількість кольорів, у той час як монітор та принтер у змозі відтворювати лише обмежену частину цього діапазону. Причому діапазон кольорів, що відтворюються, та спосіб їх відображення для монітора та принтера теж різні. У зв'язку з необхідністю опису різних фізичних процесів відтворення кольору, були розроблені різноманітні кольорові моделі. В основі створення кольорових моделей лежить використання універсальних мов, які дозволяють реалізувати способи точного опису кольору за допомогою стандартних математичних виразів.

В сучасних комп'ютерних програмах маніпулювання з кольором здійснюється за допомогою *колірних моделей* та *режимів*. *Колірні моделі* надають засоби для концептуального та кількісного опису кольору. *Режим* – це спосіб реалізації певної колірної моделі в рамках конкретної графічної програми.

Поняття колірної моделі

Колірні моделі (color model) використовуються для математичного опису певних колірних областей спектра. Більшість комп'ютерних колірних моделей засновано на використанні трьох основних кольорів, що відповідає сприйняттю кольору людським оком. Кожному основному кольору присвоюється певне значення цифрового коду, після чого вся решта кольорів визначаються як комбінації основних кольорів.

Незважаючи на те, що колірні моделі дозволяють подати колір математично, це завжди буде вважатись недосконалим у зв'язку з нашим сприйняттям. Однак їх зручно використовувати в комп'ютерних програмах для однозначного визначення кольору. Так, якщо послати на монітор колірний сигнал R255 G000 B255, то на будь-якому моніторі повинен з'явитися один і той же колір (у даному випадку пурпуровий).

Незалежно від того, що лежить в її основі, будь-яка модель повинна відповідати таким вимогам:

– реалізовувати визначення кольору деяким стандартним способом, який не залежить від можливостей певного пристрою;

– точно задавати діапазон кольорів, що відтворюються, оскільки жодна множина кольорів не є безмежною;

– враховувати механізм сприйняття кольорів – випромінювання або відбиття.

Типи колірних моделей

Більшість графічних пакетів дозволяють оперувати широким колом колірних моделей, частина з яких створена для спеціальних цілей, а частина – для особливих типів красок.

За принципом дії колірні моделі можна умовно поділити на три класи:

- адитивні (RGB), що основані на додаванні кольорів;
- субтрактивні (CMYK), основані на відніманні кольорів;
- перцепційні (HSB, HLS), що базуються на сприйнятті.

Адитивні колірні моделі

Ці моделі відповідають сприйняттю кольорів людським оком. При парному змішуванні первинних кольорів (червоного, зеленого та синього) створюються вторинні кольори: блакитний, пурпуровий та жовтий. Слід відмітити, що первинні та вторинні кольори відносяться до базових кольорів. Базовими кольорами називають кольори, за допомогою яких можна отримати практично весь спектр видимих кольорів.

Для отримання нових кольорів за допомогою адитивного синтезу можна використовувати і різноманітні комбінації з двох основних кольорів, зміни складу яких приводять до зміни результуючого кольору. На рис. 1.20 приведено схему отримання нових кольорів на базі двох первинних шляхом використання джерел зеленого та червоного кольорів, інтенсивністю кожного з яких можна управляти за допомогою фільтра.

Можна побачити, що рівні пропорції первинних кольорів дають жовтий колір, а зниження в суміші інтенсивності зеленого кольору при незмінному червоному дозволяє отримати помаранчевий колір.

Однак таким способом неможливо отримати деякі кольори, наприклад блакитний, для створення якого потрібно наявність третього первинного кольору – синього (рис. 1.21).

– точно задавати діапазон кольорів, що відтворюються, оскільки жодна множина кольорів не є безмежною;

– враховувати механізм сприйняття кольорів – випромінювання або відбиття.

Типи колірних моделей

Більшість графічних пакетів дозволяють оперувати широким колом колірних моделей, частина з яких створена для спеціальних цілей, а частина – для особливих типів красок.

За принципом дії колірні моделі можна умовно поділити на три класи:

- адитивні (RGB), що основані на додаванні кольорів;
- субтрактивні (CMYK), основані на відніманні кольорів;
- перцепційні (HSB, HLS), що базуються на сприйнятті.

Адитивні колірні моделі

Ці моделі відповідають сприйняттю кольорів людським оком. При парному змішуванні первинних кольорів (червоного, зеленого та синього) створюються вторинні кольори: блакитний, пурпуровий та жовтий. Слід відмітити, що первинні та вторинні кольори відносяться до базових кольорів. Базовими кольорами називають кольори, за допомогою яких можна отримати практично весь спектр видимих кольорів.

Для отримання нових кольорів за допомогою адитивного синтезу можна використовувати і різноманітні комбінації з двох основних кольорів, зміни складу яких приводять до зміни результуючого кольору. На рис. 1.20 приведено схему отримання нових кольорів на базі двох первинних шляхом використання джерел зеленого та червоного кольорів, інтенсивністю кожного з яких можна управляти за допомогою фільтра.

Можна побачити, що рівні пропорції первинних кольорів дають жовтий колір, а зниження в суміші інтенсивності зеленого кольору при незмінному червоному дозволяє отримати помаранчевий колір.

Однак таким способом неможливо отримати деякі кольори, наприклад блакитний, для створення якого потрібно наявність третього первинного кольору – синього (рис. 1.21).

Математично кольорову модель RGB краще всього подавати у вигляді куба (рис. 1.22). В цьому випадку кожна його просторова точка однозначно визначається значеннями координат X, Y і Z. Якщо на осі X відкладати червону складову, на осі Y – зелену, а на осі Z – синю, то кожному кольору можна поставити точку всередині куба.

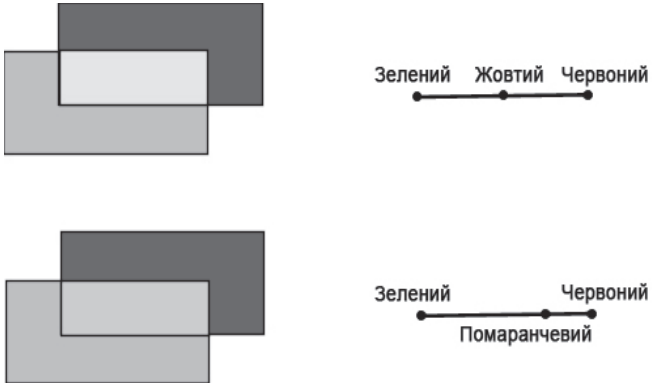


Рис. 1.20. Адитивний синтез на базі зеленого та червоного кольорів

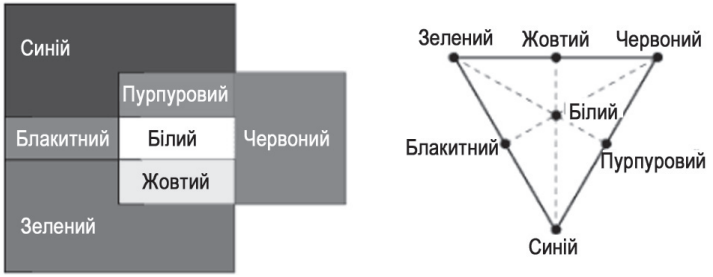


Рис. 1.21. Кольорова та колометрична схеми отримання кольорового простору RGB-моделі

Візуалізація за допомогою адитивної моделі

У графічних пакетах кольорова RGB-модель використовується для створення кольорів зображення на екрані монітора, основними елементами якого є три електронних прожектори і екран з нанесеними на

Математично кольорову модель RGB краще всього подавати у вигляді куба (рис. 1.22). В цьому випадку кожна його просторова точка однозначно визначається значеннями координат X, Y і Z. Якщо на осі X відкладати червону складову, на осі Y – зелену, а на осі Z – синю, то кожному кольору можна поставити точку всередині куба.

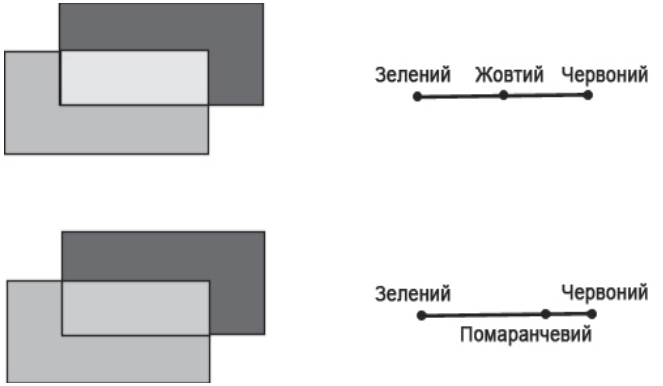


Рис. 1.20. Адитивний синтез на базі зеленого та червоного кольорів

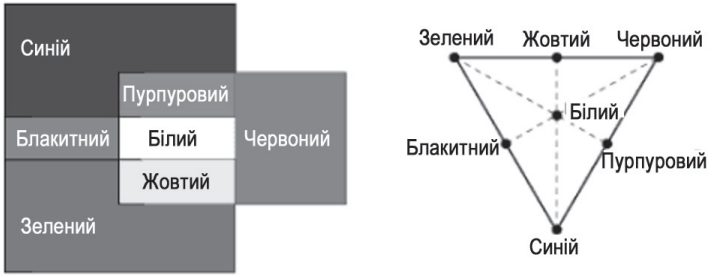


Рис. 1.21. Кольорова та колометрична схеми отримання кольорового простору RGB-моделі

Візуалізація за допомогою адитивної моделі

У графічних пакетах кольорова RGB-модель використовується для створення кольорів зображення на екрані монітора, основними елементами якого є три електронних прожектори і екран з нанесеними на

нього трьома різними люмінофорами. Ці люмінофори мають різні спектральні характеристики. Але на відміну від ока вони не поглинають, а випромінюють світло. Один люмінофор під дією падаючого на нього електронного променя випромінює червоний колір, другий – зелений, а третій – синій.

Елементарний елемент зображення, який виробляється комп'ютером, називається *пікселем* (pixel от pixture element). Якщо роздивитися білий екран включеного монітора через лупу, то можна побачити, що він складається з великої кількості окремих точок червоного, зеленого і синього кольорів (рис. 1.23, а), об'єднаних в RGB-елементи у вигляді тріад основних точок. Під час розглядання зображення на екрані з деякої відстані ці кольорові складові RGB-елементів зливаються, створюючи ілюзію результуючого кольору.

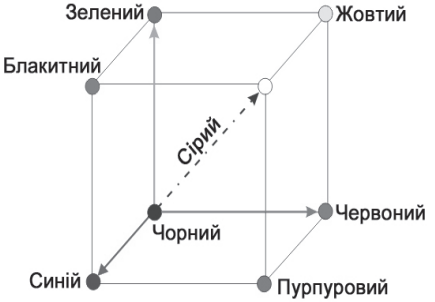
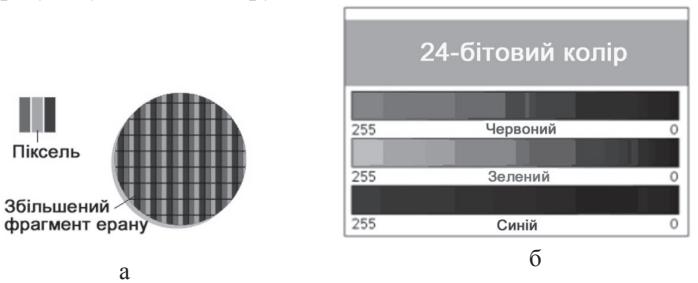


Рис. 1.22. Математична схема RGB-моделі



а б

Рис. 1.23. Колірна модель RGB: а – збільшена область екрана монітора; б – шкала інтенсивності для кожного RGB-каналу

Кожний з трьох компонентів RGB-тріади може приймати одне із 256 дискретних значень: від 0 до 255 (рис. 1.23, б).

нього трьома різними люмінофорами. Ці люмінофори мають різні спектральні характеристики. Але на відміну від ока вони не поглинають, а випромінюють світло. Один люмінофор під дією падаючого на нього електронного променя випромінює червоний колір, другий – зелений, а третій – синій.

Елементарний елемент зображення, який виробляється комп'ютером, називається *пікселем* (pixel от pixture element). Якщо роздивитися білий екран включеного монітора через лупу, то можна побачити, що він складається з великої кількості окремих точок червоного, зеленого і синього кольорів (рис. 1.23, а), об'єднаних в RGB-елементи у вигляді тріад основних точок. Під час розглядання зображення на екрані з деякої відстані ці кольорові складові RGB-елементів зливаються, створюючи ілюзію результуючого кольору.

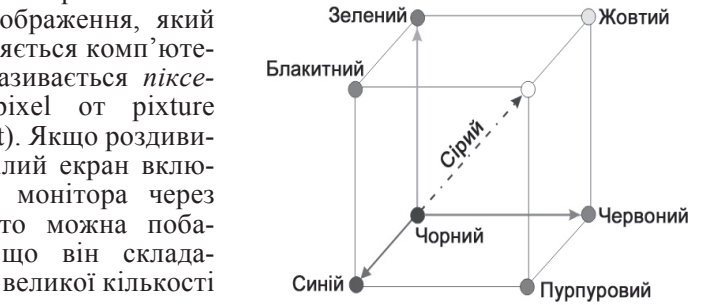
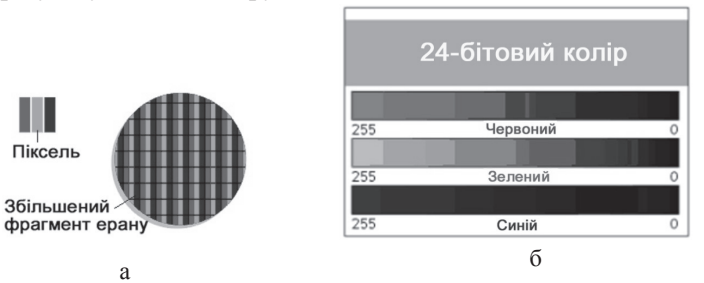


Рис. 1.22. Математична схема RGB-моделі



а б

Рис. 1.23. Колірна модель RGB: а – збільшена область екрана монітора; б – шкала інтенсивності для кожного RGB-каналу

Кожний з трьох компонентів RGB-тріади може приймати одне із 256 дискретних значень: від 0 до 255 (рис. 1.23, б).

На рис. 1.24 наведена схема отримання за допомогою адитивного синтезу шести (із 16,7 млн.) кольорів. Як уже згадувалося раніше, у випадку, коли всі три кольорові компоненти мають максимальну інтенсивність, отриманий колір здається білим. Якщо всі компоненти мають нульову інтенсивність, отриманий колір – чисто чорний.

Обмеження RGB-моделі

Незважаючи на те, що кольорова модель RGB достатньо проста і наочна, при її використанні на практиці виникає дві проблеми:

- *апаратна залежність;*
- *обмеження кольорової гами.*

Перша проблема пов'язана з тим, що колір, виникаючий у результаті змішування кольорових складових RGB-елемента, залежить від типу люмінофора. А оскільки в технології виробництва сучасних кінескопів застосовуються різні типи люмінофорів, то установка одних і тих же інтенсивностей екранних променів у випадку різних люмінофорів приведе до синтезу різного кольору. Наприклад, якщо на електронний блок монітора подати визначену трійку RGB-значень, скажемо R=98, G=127 і B=201, то не можна однозначно говорити про результат змішування. Ці значення лише задають інтенсивності збудження трьох люмінофорів одного піксела зображення. Який отримаємо при цьому колір, залежить від спектрального складу випромінюваного люмінофором світла. Тому поряд з установкою тріади значень інтенсивностей необхідно знати спектральну характеристику люмінофора.

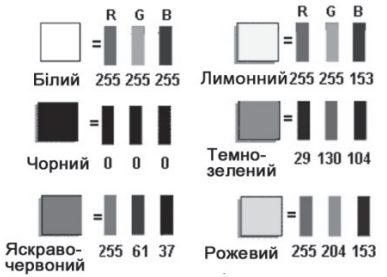


Рис. 1.24. Приклад 6 із 16,7 млн. можливих кольорів шляхом варіації інтенсивності RGB-компонент

На рис. 1.24 наведена схема отримання за допомогою адитивного синтезу шести (із 16,7 млн.) кольорів. Як уже згадувалося раніше, у випадку, коли всі три кольорові компоненти мають максимальну інтенсивність, отриманий колір здається білим. Якщо всі компоненти мають нульову інтенсивність, отриманий колір – чисто чорний.

Обмеження RGB-моделі

Незважаючи на те, що кольорова модель RGB достатньо проста і наочна, при її використанні на практиці виникає дві проблеми:

- *апаратна залежність;*
- *обмеження кольорової гами.*

Перша проблема пов'язана з тим, що колір, виникаючий у результаті змішування кольорових складових RGB-елемента, залежить від типу люмінофора. А оскільки в технології виробництва сучасних кінескопів застосовуються різні типи люмінофорів, то установка одних і тих же інтенсивностей екранних променів у випадку різних люмінофорів приведе до синтезу різного кольору. Наприклад, якщо на електронний блок монітора подати визначену трійку RGB-значень, скажемо R=98, G=127 і B=201, то не можна однозначно говорити про результат змішування. Ці значення лише задають інтенсивності збудження трьох люмінофорів одного піксела зображення. Який отримаємо при цьому колір, залежить від спектрального складу випромінюваного люмінофором світла. Тому поряд з установкою тріади значень інтенсивностей необхідно знати спектральну характеристику люмінофора.

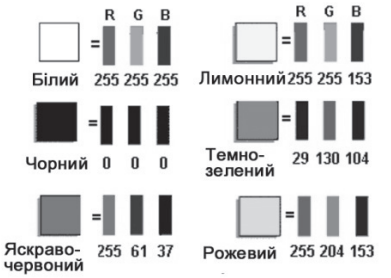


Рис. 1.24. Приклад 6 із 16,7 млн. можливих кольорів шляхом варіації інтенсивності RGB-компонент

36

Існують і інші причини, які призводять до апаратної залежності RGB-моделі навіть для моніторів, які випускаються одним і тим же виробником. Це пов'язано з тим, що в процесі експлуатації відбувається старіння люмінофора і зміни емісійних характеристик електронних прожекторів. Для усунення (або в крайньому разі мінімізації) залежності RGB-моделі від апаратних засобів використовуються різні пристрої і програми градуювання.

Кольорова гама (color gamut) – це діапазон кольорів, який може розпізнавати людина або відображати пристрій незалежно від механізму отримання кольору (випромінювання або відбивання).

Обмеження кольорової гами пояснюється тим, що за допомогою адитивного синтезу принципіально неможливо отримати всі кольори видимого спектра (це доказано теоретично!). Зокрема, деякі кольори, такі як чисто голубий або чисто жовтий, не можуть бути точно відтворені на екрані. Але, незважаючи на те, що людське око здатне розпізнавати більше кольорів, ніж монітор, RGB-моделі достатньо для створення кольорів і відтінків, необхідних для відтворення фотореалістичних зображень на екрані комп'ютера.

Субтрактивні колірні моделі

На відміну від екрана монітора, в якому відтворення кольорів основане на випромінюванні світла, друкована сторінка може тільки відбивати колір.

Субтрактивні кольори на відміну від адитивних кольорів (тієї ж RGB-моделі) отримуються відніманням другорядних кольорів із загального променя світла. В цій системі *білий* колір з'являється як результат відсутності всіх кольорів, тоді як їх присутність дає *чорний* колір (рис. 1.25).

Якщо відняти з білого три первинних кольори RGB, ми отримаємо трійку додаткових кольорів CMY (голубого (Cyan), пурпурного (Magenta) і жовтого (Yellow)). «Субтрактивний» означає, що від білого віднімаються первинні кольори.

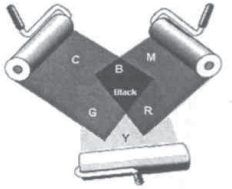


Рис. 1.25. Субтрактивна колірна модель CMY

Існують і інші причини, які призводять до апаратної залежності RGB-моделі навіть для моніторів, які випускаються одним і тим же виробником. Це пов'язано з тим, що в процесі експлуатації відбувається старіння люмінофора і зміни емісійних характеристик електронних прожекторів. Для усунення (або в крайньому разі мінімізації) залежності RGB-моделі від апаратних засобів використовуються різні пристрої і програми градуювання.

Кольорова гама (color gamut) – це діапазон кольорів, який може розпізнавати людина або відображати пристрій незалежно від механізму отримання кольору (випромінювання або відбивання).

Обмеження кольорової гами пояснюється тим, що за допомогою адитивного синтезу принципіально неможливо отримати всі кольори видимого спектра (це доказано теоретично!). Зокрема, деякі кольори, такі як чисто голубий або чисто жовтий, не можуть бути точно відтворені на екрані. Але, незважаючи на те, що людське око здатне розпізнавати більше кольорів, ніж монітор, RGB-моделі достатньо для створення кольорів і відтінків, необхідних для відтворення фотореалістичних зображень на екрані комп'ютера.

Субтрактивні колірні моделі

На відміну від екрана монітора, в якому відтворення кольорів основане на випромінюванні світла, друкована сторінка може тільки відбивати колір.

Субтрактивні кольори на відміну від адитивних кольорів (тієї ж RGB-моделі) отримуються відніманням другорядних кольорів із загального променя світла. В цій системі *білий* колір з'являється як результат відсутності всіх кольорів, тоді як їх присутність дає *чорний* колір (рис. 1.25).

Якщо відняти з білого три первинних кольори RGB, ми отримаємо трійку додаткових кольорів CMY (голубого (Cyan), пурпурного (Magenta) і жовтого (Yellow)). «Субтрактивний» означає, що від білого віднімаються первинні кольори.

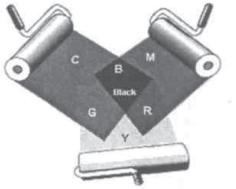


Рис. 1.25. Субтрактивна колірна модель CMY

Відношення, які пов'язують адитивні (червоний, зелений, синій) і субтрактивні (голубий, жовтий, пурпуровий) кольори:

- *зелений + синій = голубий;*
- *зелений + червоний = жовтий;*
- *червоний + синій = пурпуровий;*
- *зелений + синій + червоний = білий;*
- *голубий + жовтий + пурпуровий = чорний.*

Наприклад, якщо барвник голубий (синьо-зелений), то він поглинає із спектра компліментарний (доповнючий) йому червоний колір і відбиває голубий. Відповідно пурпуровий барвник поглинає компліментарний йому зелений колір, а жовтий барвник – синій колір.

Якщо при друкуванні накласти один на одного пурпуровий і жовтий кольори, то отримаємо червоний колір, оскільки пурпуровий барвник прибере зелену складову, а жовтий – синю складову падаючого світла. Відповідно під час друкування з накладенням усіх трьох субтрактивних кольорів результативний колір буде чорним.

На основі вищенаведеного можна сформулювати *правило корекції кольорового дисбалансу* під час кольорового друку: якщо зображення має зайвий синій відтінок, потрібно збільшити жовту складову, оскільки жовтий поглинає сині складові. Відповідно залишки зеленого кольору можна скорегувати збільшенням пурпурової складової, а залишки червоного кольору – збільшенням голубої складової.

Моделі CMY і CMYK

Існує дві найбільш розповсюджені версії субтрактивної моделі CMY і CMYK. Перша з них використовується в тому випадку, коли зображення або рисунок будуть виводитися на чорно-білому принтері, який дозволяє замінювати чорний картридж на кольоровий (color upgrade).

В її основі лежить використання трьох субтрактивних (другорядних) кольорів: голубого (Cyan), пурпурового (Magenta) і жовтого (Yellow). Теоретично при змішуванні цих кольорів на білому папері рівними порціями отримуємо *чорний* колір. Однак у реальному технологічному процесі отримання чорного кольору шляхом змішування трьох основних кольорів для паперу неефективно з таких причин:

Відношення, які пов'язують адитивні (червоний, зелений, синій) і субтрактивні (голубий, жовтий, пурпуровий) кольори:

- *зелений + синій = голубий;*
- *зелений + червоний = жовтий;*
- *червоний + синій = пурпуровий;*
- *зелений + синій + червоний = білий;*
- *голубий + жовтий + пурпуровий = чорний.*

Наприклад, якщо барвник голубий (синьо-зелений), то він поглинає із спектра компліментарний (доповнючий) йому червоний колір і відбиває голубий. Відповідно пурпуровий барвник поглинає компліментарний йому зелений колір, а жовтий барвник – синій колір.

Якщо при друкуванні накласти один на одного пурпуровий і жовтий кольори, то отримаємо червоний колір, оскільки пурпуровий барвник прибере зелену складову, а жовтий – синю складову падаючого світла. Відповідно під час друкування з накладенням усіх трьох субтрактивних кольорів результативний колір буде чорним.

На основі вищенаведеного можна сформулювати *правило корекції кольорового дисбалансу* під час кольорового друку: якщо зображення має зайвий синій відтінок, потрібно збільшити жовту складову, оскільки жовтий поглинає сині складові. Відповідно залишки зеленого кольору можна скорегувати збільшенням пурпурової складової, а залишки червоного кольору – збільшенням голубої складової.

Моделі CMY і CMYK

Існує дві найбільш розповсюджені версії субтрактивної моделі CMY і CMYK. Перша з них використовується в тому випадку, коли зображення або рисунок будуть виводитися на чорно-білому принтері, який дозволяє замінювати чорний картридж на кольоровий (color upgrade).

В її основі лежить використання трьох субтрактивних (другорядних) кольорів: голубого (Cyan), пурпурового (Magenta) і жовтого (Yellow). Теоретично при змішуванні цих кольорів на білому папері рівними порціями отримуємо *чорний* колір. Однак у реальному технологічному процесі отримання чорного кольору шляхом змішування трьох основних кольорів для паперу неефективно з таких причин:

- На практиці змішування реальних пурпурового, голубого і жовтого кольорів дає скоріше грязно-коричневий або грязно-сірий колір: тріадні фарби не дають тієї глибини і насиченості, яка досягається використанням справжнього чорного. Оскільки чистота і насиченість чорного кольору надзвичайно важлива в друкарському процесі, в цей процес був введений ще один колір.
- При виведенні дрібних чорних деталей зображення або тексту без використання чорного пігменту зростає риск непопадання (недостатньо точний збіг точок нанесення) пурпурового, голубого і жовтого кольорів. Збільшення ж точності друкуючого апарату потребує неадекватних затрат.
- Змішування 100% пурпурового, голубого і жовтого пігментів в одній точці у випадку струйного друку відповідно змочує папір, деформує його і збільшує час просушування.
- На створення чорного кольору за допомогою моделі CMY затрачується в три рази більше фарби.
- Чорний пігмент (як пігмент, як правило, використовується сажа) відповідно дешевше решти трьох.

Внаслідок перерахованих факторів під час друку використовується добавка додаткового чорного компонента кольору. Ця технологія приводить також до покращання якості тіней і сірих відтінків. Інтенсивність кожного з чотирьох компонентів кольору може змінюватися в діапазоні від 0 до 100%.

В абревіатурі CMYK використовується буква «К» (остання буква слова black) для того, щоб уникнути плутанини між black і blue. Ще один варіант трактування букви «К» як абревіатури Key color.

Кожне з чисел, яке визначає колір в CMYK, являє собою процент фарби даного кольору, який входить у склад кольорової комбінації точки растра, що виводиться на пристрій друку.

Наприклад, для отримання темно-помаранчевого кольору потрібно змішати 30% блакитної фарби, 45% пурпурової фарби, 80% жовтої і 5% чорної. Це можна позначити таким чином: (30,45,80,5). Іноді користуються таким позначенням: C30 M45 Y80 K5.

Важливо відмітити, що цифрове значення фарби в CMYK не може само по собі описати колір. Цифри – лише набір апаратних даних,

- На практиці змішування реальних пурпурового, голубого і жовтого кольорів дає скоріше грязно-коричневий або грязно-сірий колір: тріадні фарби не дають тієї глибини і насиченості, яка досягається використанням справжнього чорного. Оскільки чистота і насиченість чорного кольору надзвичайно важлива в друкарському процесі, в цей процес був введений ще один колір.
- При виведенні дрібних чорних деталей зображення або тексту без використання чорного пігменту зростає риск непопадання (недостатньо точний збіг точок нанесення) пурпурового, голубого і жовтого кольорів. Збільшення ж точності друкуючого апарату потребує неадекватних затрат.
- Змішування 100% пурпурового, голубого і жовтого пігментів в одній точці у випадку струйного друку відповідно змочує папір, деформує його і збільшує час просушування.
- На створення чорного кольору за допомогою моделі CMY затрачується в три рази більше фарби.
- Чорний пігмент (як пігмент, як правило, використовується сажа) відповідно дешевше решти трьох.

Внаслідок перерахованих факторів під час друку використовується добавка додаткового чорного компонента кольору. Ця технологія приводить також до покращання якості тіней і сірих відтінків. Інтенсивність кожного з чотирьох компонентів кольору може змінюватися в діапазоні від 0 до 100%.

В абревіатурі CMYK використовується буква «К» (остання буква слова black) для того, щоб уникнути плутанини між black і blue. Ще один варіант трактування букви «К» як абревіатури Key color.

Кожне з чисел, яке визначає колір в CMYK, являє собою процент фарби даного кольору, який входить у склад кольорової комбінації точки растра, що виводиться на пристрій друку.

Наприклад, для отримання темно-помаранчевого кольору потрібно змішати 30% блакитної фарби, 45% пурпурової фарби, 80% жовтої і 5% чорної. Це можна позначити таким чином: (30,45,80,5). Іноді користуються таким позначенням: C30 M45 Y80 K5.

Важливо відмітити, що цифрове значення фарби в CMYK не може само по собі описати колір. Цифри – лише набір апаратних даних,

які використовуються в друкарському процесі для формування зображення. На практиці реальний колір буде визначатись не тільки розміром точки растра на пристрої друку, існуючими цифрами в підготовленому до друку файлі, але і реаліями конкретного друкуючого процесу, на які можуть впливати такі фактори, як стан друкуючої машини, якість паперу, вологість повітря в цеху; умови перегляду відбитка (спектральні характеристики джерела освітлення) та інші.

Обмеження CMYK-моделі

CMYK-модель має ті ж два типи обмеження, що і RGB-модель:

- *апаратна залежність;*
- *обмежений колірний діапазон.*

В CMYK-моделі також неможливо точно передбачити результативний колір тільки на базі цифрових значень її окремих компонентів. У цьому вона є навіть більше апаратно-залежною моделлю, ніж RGB. Це пов'язано з тим, що в ній є найбільша кількість дестабілізуючих факторів, ніж в RGB-моделі.

До них у першу чергу можна віднести варіацію складу кольорових барвників, які використовуються для створення друкованих кольорів. Кольорове відчуття визначається ще і типом використовуваного паперу, способом друку, не в останню чергу, зовнішнім освітленням: ніякий об'єкт не може відбити колір, який відсутній у джерелі випромінювання.

Через те, що кольорові барвники мають гірші характеристики порівняно з люмінофорами, кольорова модель CMYK має більш вузький кольоровий діапазон порівняно з RGB-моделлю. Відповідно вона не може виробляти яскраве насичення кольору, а також ряд специфічних кольорів, таких, наприклад, як металевий або золотистий.

Про екранні кольори, які неможливо точно відтворити під час друку, говорять, що вони лежать *поза кольоровою гамою* (gamut alarm) CMYK-моделі (рис. 1.26). У більшості графічних пакетів під такими кольорами розуміються кольори, які можуть бути подані у форматі RGB або HSV, але при цьому не мають друкованих аналогів у кольоровому просторі CMYK.

які використовуються в друкарському процесі для формування зображення. На практиці реальний колір буде визначатись не тільки розміром точки растра на пристрої друку, існуючими цифрами в підготовленому до друку файлі, але і реаліями конкретного друкуючого процесу, на які можуть впливати такі фактори, як стан друкуючої машини, якість паперу, вологість повітря в цеху; умови перегляду відбитка (спектральні характеристики джерела освітлення) та інші.

Обмеження CMYK-моделі

CMYK-модель має ті ж два типи обмеження, що і RGB-модель:

- *апаратна залежність;*
- *обмежений колірний діапазон.*

В CMYK-моделі також неможливо точно передбачити результативний колір тільки на базі цифрових значень її окремих компонентів. У цьому вона є навіть більше апаратно-залежною моделлю, ніж RGB. Це пов'язано з тим, що в ній є найбільша кількість дестабілізуючих факторів, ніж в RGB-моделі.

До них у першу чергу можна віднести варіацію складу кольорових барвників, які використовуються для створення друкованих кольорів. Кольорове відчуття визначається ще і типом використовуваного паперу, способом друку, не в останню чергу, зовнішнім освітленням: ніякий об'єкт не може відбити колір, який відсутній у джерелі випромінювання.

Через те, що кольорові барвники мають гірші характеристики порівняно з люмінофорами, кольорова модель CMYK має більш вузький кольоровий діапазон порівняно з RGB-моделлю. Відповідно вона не може виробляти яскраве насичення кольору, а також ряд специфічних кольорів, таких, наприклад, як металевий або золотистий.

Про екранні кольори, які неможливо точно відтворити під час друку, говорять, що вони лежать *поза кольоровою гамою* (gamut alarm) CMYK-моделі (рис. 1.26). У більшості графічних пакетів під такими кольорами розуміються кольори, які можуть бути подані у форматі RGB або HSV, але при цьому не мають друкованих аналогів у кольоровому просторі CMYK.

Невідповідність кольорових діапазонів RGB- і CMYK-моделей створює серйозну проблему. Отримана на екрані монітора в результаті напруженої роботи красива картинка при друкуванні раптом перетворюється в пониклу і бліду подобу оригіналу.

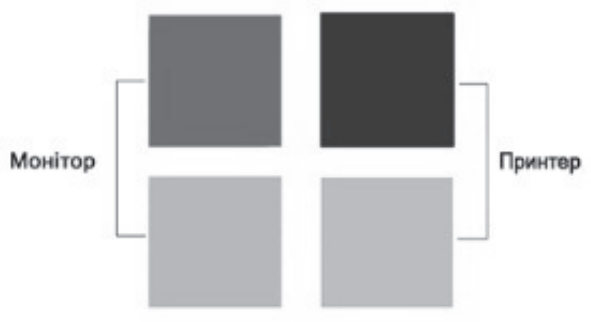


Рис. 1.26. Невідповідність кольорів, які відображаються на екрані монітора, з друкованими на принтері

Для запобігання подібної ситуації розробниками графічних програм передбачений комплекс спеціальних засобів:

– Найбільш прості основані на виявленні і корекції невідповідних кольорів відповідно в процесі редагування (*редагування зображення у форматі CMYK-моделі; використання CMYK- орієнтованих палітр; засоби індикакації, які є в програмах).*

– Більш кардинальні, призначені для розширення колірного простору *CMYK*-моделі.

– Найбільш сучасні – використання систем керування кольором – CMS (Color Management Systems).

Перцепційні колірні моделі

Для дизайнерів, художників і фотографів основним інструментом індикації і виведення кольору служить око. Цей природний «інструмент» володіє кольоровою гамою, яка набагато перевищує можливості будь-якого технічного пристрою (сканер, принтер або фотоекспонуючий пристрій виведення на плівку). Розглянуті вище кольорові системи RGB і CMYK, які використовуються для опису технічних пристроїв, є апаратно-залежними. Це означає, що

Невідповідність кольорових діапазонів RGB- і CMYK-моделей створює серйозну проблему. Отримана на екрані монітора в результаті напруженої роботи красива картинка при друкуванні раптом перетворюється в пониклу і бліду подобу оригіналу.

Рис. 1.26. Невідповідність кольорів, які відображаються на екрані монітора, з друкованими на принтері

Для запобігання подібної ситуації розробниками графічних програм передбачений комплекс спеціальних засобів:

– Найбільш прості основані на виявленні і корекції невідповідних кольорів відповідно в процесі редагування (*редагування зображення у форматі CMYK-моделі; використання CMYK- орієнтованих палітр; засоби індикакації, які є в програмах).*

– Більш кардинальні, призначені для розширення колірного простору *CMYK*-моделі.

– Найбільш сучасні – використання систем керування кольором – CMS (Color Management Systems).

Перцепційні колірні моделі

Для дизайнерів, художників і фотографів основним інструментом індикації і виведення кольору служить око. Цей природний «інструмент» володіє кольоровою гамою, яка набагато перевищує можливості будь-якого технічного пристрою (сканер, принтер або фотоекспонуючий пристрій виведення на плівку). Розглянуті вище кольорові системи RGB і CMYK, які використовуються для опису технічних пристроїв, є апаратно-залежними. Це означає, що

відтворюваний або створюваний за допомогою них колір визначається не тільки складовими моделі, але і залежить від характеристики пристрою виведення. Для усунення апаратної залежності було розроблено ряд так званих *перцепційних* (інтуїтивних) кольорових моделей. В їх основу закладено роздільне визначення яскравості і кольоровості. Такий підхід забезпечує ряд переваг:

– дозволяє використовувати колір на інтуїтивно зрозумілому рівні;

– значно спрощує проблему узгодження кольорів, оскільки після установлення значення яскравості можна зайнятися настройкою кольору.

Прототипом усіх колірних моделей, які використовують концепцію розподілення яскравості і кольоровості, є HSV-модель. До інших подібних систем відносяться HSI, HSB, HSL і YUV. Спільним для них є те, що колір задається не у виді суміші трьох основних кольорів – червоного, синього і зеленого, а визначається шляхом вказування двох компонентів: *кольоровості (кольорового тону і насиченості) і яскравості*.

Колірна модель HSB

Модель HSB (Hue — *колірний тон*, Saturation — *насиченість*, Brightness – *яскравість*) або її найближчий аналог HSL подані у більшості сучасних графічних пакетах. З усіх використовуваних у теперішній час моделей вона найбільш точно відповідає способу сприйняття кольорів людським оком і дозволяє описувати кольори інтуїтивно ясним способом. В HSB-моделі всі кольори визначаються за допомогою комбінації трьох базових параметрів (рис. 1.27):

- колірний тон (H);
- насиченість (S);
- яскравість (B)/ значення (V).

Колірний тон. Як уже відзначалось, кожне реальне джерело світла відтворює його у вигляді суміші хвиль, які мають різну довжину. Під *колірним тоном* (hue) розуміється світло з домінуючою довжиною хвилі.

відтворюваний або створюваний за допомогою них колір визначається не тільки складовими моделі, але і залежить від характеристики пристрою виведення. Для усунення апаратної залежності було розроблено ряд так званих *перцепційних* (інтуїтивних) кольорових моделей. В їх основу закладено роздільне визначення яскравості і кольоровості. Такий підхід забезпечує ряд переваг:

– дозволяє використовувати колір на інтуїтивно зрозумілому рівні;

– значно спрощує проблему узгодження кольорів, оскільки після установлення значення яскравості можна зайнятися настройкою кольору.

Прототипом усіх колірних моделей, які використовують концепцію розподілення яскравості і кольоровості, є HSV-модель. До інших подібних систем відносяться HSI, HSB, HSL і YUV. Спільним для них є те, що колір задається не у виді суміші трьох основних кольорів – червоного, синього і зеленого, а визначається шляхом вказування двох компонентів: *кольоровості (кольорового тону і насиченості) і яскравості*.

Колірна модель HSB

Модель HSB (Hue — *колірний тон*, Saturation — *насиченість*, Brightness – *яскравість*) або її найближчий аналог HSL подані у більшості сучасних графічних пакетах. З усіх використовуваних у теперішній час моделей вона найбільш точно відповідає способу сприйняття кольорів людським оком і дозволяє описувати кольори інтуїтивно ясним способом. В HSB-моделі всі кольори визначаються за допомогою комбінації трьох базових параметрів (рис. 1.27):

- колірний тон (H);
- насиченість (S);
- яскравість (B)/ значення (V).

Колірний тон. Як уже відзначалось, кожне реальне джерело світла відтворює його у вигляді суміші хвиль, які мають різну довжину. Під *колірним тоном* (hue) розуміється світло з домінуючою довжиною хвилі.

Переважно для описання колірного тону (в деяких джерелах застосовується термін *відтінок*) застосовується назва кольору, наприклад червоний, оранжевий або зелений. У традиційній інтерпретації цієї моделі кожний колірний тон займає визначене положення на периферії *колірного круга* і характеризується величиною кута в діапазоні від 0 до 360° (рис.1.28). Переважно для червоного кольору береться кут 0°, для чисто зеленого – 120° і для чисто синього – 240°.

На колірному крузі первинні кольори розташовані на рівній відстані один від одного. Другорядні кольори знаходяться між первинними. В свою чергу, кожний колір розташований проти комплементарного кольору, який його

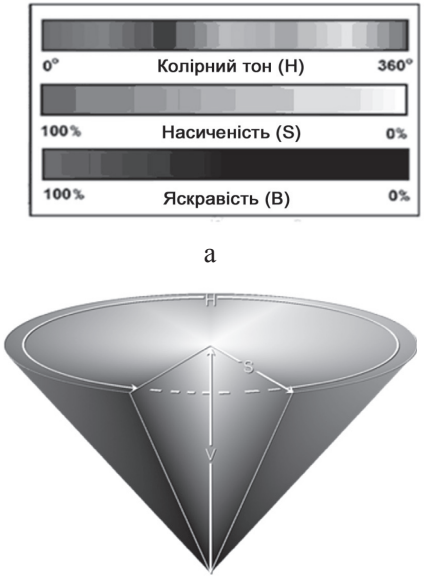


Рис. 1.27. Колірна модель HSB:
а – палітра вибору кольору за допомогою HSB-моделі;
б – структурна схема у формі конуса

доповнює, причому він находиться між кольорами, за допомогою яких був отриманий. Наприклад, додавання жовтого і голубого дає зелений.

Таким чином, на колірному колі зелений колір повинен знаходитися між жовтим і голубим. Однак само по собі поняття колірного тону не має повної інформації про кольори.

Наприклад, світло, в якому переважає компонент з довжиною хвилі близько 450 нм, буде сприйматися більшістю людей з нормальним зором (не дальтоніками!) як відтінок, який споріднюється з синім кольором (йому відповідає на колірному крузі кут 240°).

Переважно для описання колірного тону (в деяких джерелах застосовується термін *відтінок*) застосовується назва кольору, наприклад червоний, оранжевий або зелений. У традиційній інтерпретації цієї моделі кожний колірний тон займає визначене положення на периферії *колірного круга* і характеризується величиною кута в діапазоні від 0 до 360° (рис.1.28). Переважно для червоного кольору береться кут 0°, для чисто зеленого – 120° і для чисто синього – 240°.

На колірному крузі первинні кольори розташовані на рівній відстані один від одного. Другорядні кольори знаходяться між первинними. В свою чергу, кожний колір розташований проти комплементарного кольору, який його

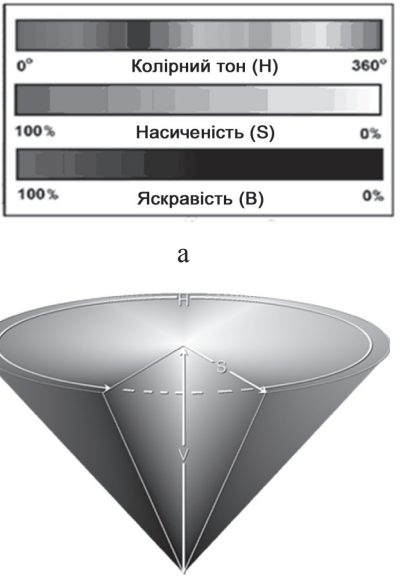


Рис. 1.27. Колірна модель HSB:
а – палітра вибору кольору за допомогою HSB-моделі;
б – структурна схема у формі конуса

доповнює, причому він находиться між кольорами, за допомогою яких був отриманий. Наприклад, додавання жовтого і голубого дає зелений.

Таким чином, на колірному колі зелений колір повинен знаходитися між жовтим і голубим. Однак само по собі поняття колірного тону не має повної інформації про кольори.

Наприклад, світло, в якому переважає компонент з довжиною хвилі близько 450 нм, буде сприйматися більшістю людей з нормальним зором (не дальтоніками!) як відтінок, який споріднюється з синім кольором (йому відповідає на колірному крузі кут 240°).

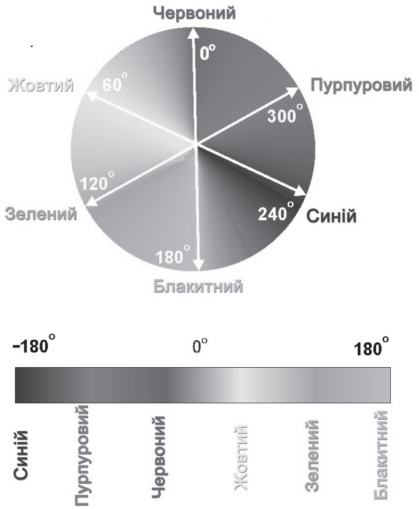


Рис. 1.28. Розміщення кольорів на колірному колі

Питання в тому, що розуміється під поняттям «синій»? Темно-синє або голубе небо, лазурне море, польова волошка і незабудка – це все приклади кольорів, у яких домінує синій колір, але, незважаючи на це, вони сприймаються нашим оком різними. Доповнюючими компонентами, які обумовлюють різницю між цими кольорами, є насиченість (saturation) і яскравість (brightness).

Насиченість характеризує частоту (інтенсивність) кольору. Він визначає співвідношення між основним, домінуючим компонентом кольору і всіма іншими довжинами хвиль (кількість сірого), які беруть участь у формуванні кольору. Кількісне значення цього параметра виражається у відсотках від 0% (сірий) до 100% (повністю насичений).

Відповідно до другого визначення, насиченість відбиває, наскільки далеко відстає даний колір від рівного з ним по яскравості білого кольору. В цьому випадку її можна виміряти числом ледь помітних переходів (градацій), які лежать між даним кольором і білим.

Чим вище значення насиченості, тим сильніше і ясніше відчувається колірний тон. Наприклад, пастельний синій колір сприймається як розмитий синій через незначний вміст у ньому чистого відтінку. Зниження насиченості приводить до того, що колір стає нейтральним, без чітко вираженого тону.

Якщо ви візьмете кольорову фотографію і понизите насиченість до 0%, то в результаті отримаєте чорно-білу фотографію (в градаціях сірого).

Прикладами кольорів з максимальною насиченістю можуть бути *спектральні* кольори, здебільшого жовтий колір, відповідний лінії

Червоний
0°
Пурпуровий
300°
Синій
240°
Блакитний
180°
Зелений
120°
Жовтий
60°
-180°
0°
180°
Синій
Пурпуровий
Червоний
Жовтий
Зелений
Блакитний

Рис. 1.28. Розміщення кольорів на колірному колі

Питання в тому, що розуміється під поняттям «синій»? Темно-синє або голубе небо, лазурне море, польова волошка і незабудка – це все приклади кольорів, у яких домінує синій колір, але, незважаючи на це, вони сприймаються нашим оком різними. Доповнюючими компонентами, які обумовлюють різницю між цими кольорами, є насиченість (saturation) і яскравість (brightness).

Насиченість характеризує частоту (інтенсивність) кольору. Він визначає співвідношення між основним, домінуючим компонентом кольору і всіма іншими довжинами хвиль (кількість сірого), які беруть участь у формуванні кольору. Кількісне значення цього параметра виражається у відсотках від 0% (сірий) до 100% (повністю насичений).

Відповідно до другого визначення, насиченість відбиває, наскільки далеко відстає даний колір від рівного з ним по яскравості білого кольору. В цьому випадку її можна виміряти числом ледь помітних переходів (градацій), які лежать між даним кольором і білим.

Чим вище значення насиченості, тим сильніше і ясніше відчувається колірний тон. Наприклад, пастельний синій колір сприймається як розмитий синій через незначний вміст у ньому чистого відтінку. Зниження насиченості приводить до того, що колір стає нейтральним, без чітко вираженого тону.

Якщо ви візьмете кольорову фотографію і понизите насиченість до 0%, то в результаті отримаєте чорно-білу фотографію (в градаціях сірого).

Прикладами кольорів з максимальною насиченістю можуть бути *спектральні* кольори, здебільшого жовтий колір, відповідний лінії

спектра натрію з довжиною хвилі 536 мкм. У той же час жовтий колір, який отриманий шляхом адитивного додавання червоного і зеленого кольорів, характеризується пониженою насиченістю. І зовсім низьку насиченість має жовте світло сонячного диску, який вміщує практично повний спектр видимих кольорів.

Прикладами «повністю» нейтральних (*ахроматичних*) кольорів можуть служити сірий, білий і чорний кольори. По мірі пересування до центру круга колір наближається до сірого, оскільки при цьому всі базові кольори змішуються в рівній пропорції.

Природні кольори мають низьку насиченість, тому дуже насичені кольори виглядають не натуральними і підкресленими.

Пересування поперек колірного круга (на відміну від руху по окружності) приводить до зменшення долі кольору, від якого ви віддаляєтесь, і зростанню долі кольору, до якого ви наближаєтесь. В результаті це приводить до пониження насиченості, яка має максимальне значення (100%) на поверхні окружності і мінімальне (0%) – в центрі круга.

Яскравість (B) кольору показує величину чорного відтінку, який добавлений до кольору. Сонячний зайчик – приклад високої інтенсивності освітлення (яскравого). В той же час тліюче вугілля – низької. Будь-які кольори і відтінки незалежно від їх колірного тону можна порівняти по яскравості, тобто визначити, який з них темніший, а який світліший.

Яскравість ніяким чином не впливає на кольоровість, але залежить від неї, настільки сильно колір буде сприйматися нашим оком. При нульовій яскравості ми нічого не бачимо, тому будь-який колір буде сприйматися як чорний. Виходячи з цього, яскравість інколи трактують подібно насиченості, тобто як величину, зворотну степеню розбавлення кольору чорним. У такому випадку при відсутності чорного ми отримуємо чистий спектральний колір, а максимальна яскравість викликає відчуття сліпучого білого кольору.

Ахроматичні кольори, тобто білі, сірі і чорні, характеризуються тільки яскравістю. Це проявляється в тому, що одні кольори темніші, а інші світліші. Величина яскравості вимірюється в процентах у діапазоні від 0% (чорний) до 100% (білий). В міру пониження проце-

спектра натрію з довжиною хвилі 536 мкм. У той же час жовтий колір, який отриманий шляхом адитивного додавання червоного і зеленого кольорів, характеризується пониженою насиченістю. І зовсім низьку насиченість має жовте світло сонячного диску, який вміщує практично повний спектр видимих кольорів.

Прикладами «повністю» нейтральних (*ахроматичних*) кольорів можуть служити сірий, білий і чорний кольори. По мірі пересування до центру круга колір наближається до сірого, оскільки при цьому всі базові кольори змішуються в рівній пропорції.

Природні кольори мають низьку насиченість, тому дуже насичені кольори виглядають не натуральними і підкресленими.

Пересування поперек колірного круга (на відміну від руху по окружності) приводить до зменшення долі кольору, від якого ви віддаляєтесь, і зростанню долі кольору, до якого ви наближаєтесь. В результаті це приводить до пониження насиченості, яка має максимальне значення (100%) на поверхні окружності і мінімальне (0%) – в центрі круга.

Яскравість (B) кольору показує величину чорного відтінку, який добавлений до кольору. Сонячний зайчик – приклад високої інтенсивності освітлення (яскравого). В той же час тліюче вугілля – низької. Будь-які кольори і відтінки незалежно від їх колірного тону можна порівняти по яскравості, тобто визначити, який з них темніший, а який світліший.

Яскравість ніяким чином не впливає на кольоровість, але залежить від неї, настільки сильно колір буде сприйматися нашим оком. При нульовій яскравості ми нічого не бачимо, тому будь-який колір буде сприйматися як чорний. Виходячи з цього, яскравість інколи трактують подібно насиченості, тобто як величину, зворотну степеню розбавлення кольору чорним. У такому випадку при відсутності чорного ми отримуємо чистий спектральний колір, а максимальна яскравість викликає відчуття сліпучого білого кольору.

Ахроматичні кольори, тобто білі, сірі і чорні, характеризуються тільки яскравістю. Це проявляється в тому, що одні кольори темніші, а інші світліші. Величина яскравості вимірюється в процентах у діапазоні від 0% (чорний) до 100% (білий). В міру пониження проце-

45

нтного вмісту яскравості колір стає темнішим, наближаючись до чорного. Даний компонент є нелінійним, що відповідає нашому сприйняттю світлих і темних кольорів.

Яскравість і колірний тон не є повністю залежними параметрами. Зміна яскравості зображення впливає на зміну колірного тону, що створює небажаний колірний відблиск (зсув) у зображенні. Так, при значному зменшенні яскравості зелені кольори синіють, сині наближаються до фіолетових, жовті – до помаранчевих, а помаранчеві – до червоних. Сильне збільшення яскравості випромінювання викликає інший ефект: червоні кольори переходять у помаранчеві, потім у жовті і, нарешті, в білі.

Комп'ютерна реалізація 3D моделі (конуса)

Відтінок подається у вигляді райдужного кільця, а насиченість і значення кольору відбирається за допомогою вписаного в це кільце трикутника (рис. 1.29).

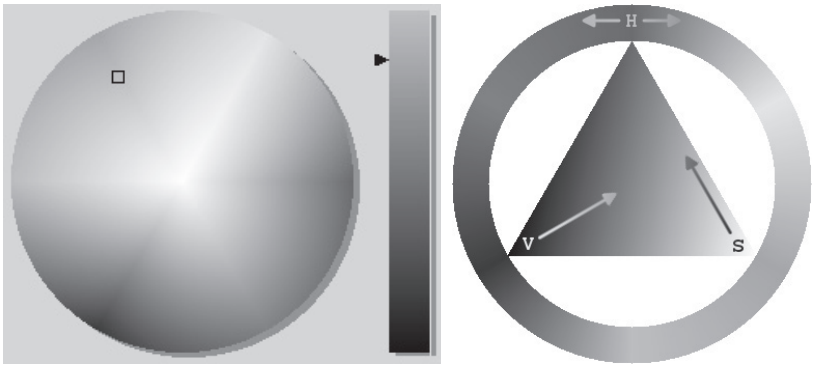


Рис. 1.29. Комп'ютерна реалізація конуса

Його вертикальна вісь, як правило, регулює насиченість, а горизонтальна дозволяє змінювати значення кольору. Таким чином, для вибору кольору потрібно спочатку вказати відтінок, а потім вибрати потрібний колір з трикутника.

нтного вмісту яскравості колір стає темнішим, наближаючись до чорного. Даний компонент є нелінійним, що відповідає нашому сприйняттю світлих і темних кольорів.

Яскравість і колірний тон не є повністю залежними параметрами. Зміна яскравості зображення впливає на зміну колірного тону, що створює небажаний колірний відблиск (зсув) у зображенні. Так, при значному зменшенні яскравості зелені кольори синіють, сині наближаються до фіолетових, жовті – до помаранчевих, а помаранчеві – до червоних. Сильне збільшення яскравості випромінювання викликає інший ефект: червоні кольори переходять у помаранчеві, потім у жовті і, нарешті, в білі.

Комп'ютерна реалізація 3D моделі (конуса)

Відтінок подається у вигляді райдужного кільця, а насиченість і значення кольору відбирається за допомогою вписаного в це кільце трикутника (рис. 1.29).

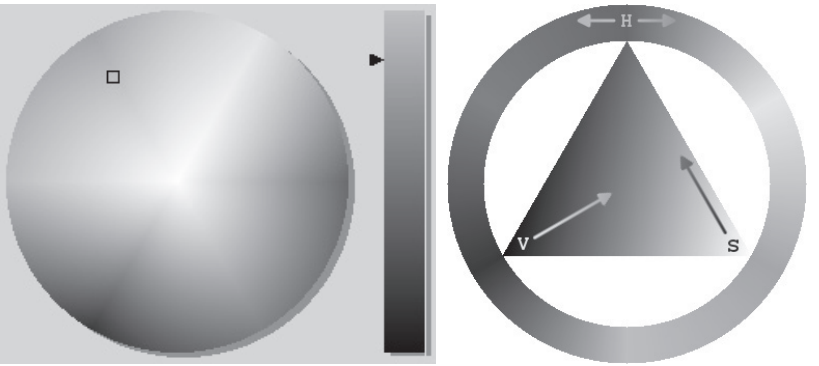


Рис. 1.29. Комп'ютерна реалізація конуса

Його вертикальна вісь, як правило, регулює насиченість, а горизонтальна дозволяє змінювати значення кольору. Таким чином, для вибору кольору потрібно спочатку вказати відтінок, а потім вибрати потрібний колір з трикутника.

Переваги і обмеження HSB-моделі

Модель HSB на відміну від моделей RGB і CMYK носить абстрактний характер. Здебільшого це зв'язано з тим, що колірний тон і насиченість кольору неможливо вимірювати безпосередньо. Будь-яка форма введення колірної інформації завжди починається з визначення червоної, зеленої і синьої складових, на базі яких потім за допомогою математичного перерахунку отримують компоненти HSB-моделі. В результаті ця колірна модель має той же колірний простір, що і RGB-модель, а значить, і належний їй недолік – обмежений колірний простір.

Разом з тим HSB-модель має порівняно з RGB- і CMYK-моделями три важливі переваги:

– Апаратна залежність. Задавання складових цієї моделі у вигляді значень колірного тону, насиченості і яскравості дозволяє однозначно визначати колір без необхідності врахування параметрів пристрою виведення.

– Модель більше відповідає природі кольору, добре підходить для сприймання людиною: колірний тон є еквівалентом довжини світла, насиченість – інтенсивності хвилі, а яскравість – кількості світла.

– Модель відрізняється більш простим і інтуїтивно зрозумілим механізмом управління кольором. Це пов'язано з тим, що колірний тон, насиченість і яскравість являють собою незалежні характеристики кольору. Наприклад, чистий червоний колір розташований на колірному крузі під кутом 0°. Якщо потрібно змістити червоний тон до оранжевого тону, то потрібно лише трохи збільшити кут, який визначає колірний тон.

Для отримання більш блідого кольору достатньо лише знизити насиченість, а для надання йому більшої яскравості – відповідно збільшити значення яскравості. Отримати такі ефекти за допомогою RGB-моделі практично неможливо, оскільки значення її колірних компонентів дуже сильно залежать один від одного.

Тому при зміні однієї з складових, наприклад червоної, це вплине не тільки на колірний тон, але одночасно і на насиченість і на яскравість.

Переваги і обмеження HSB-моделі

Модель HSB на відміну від моделей RGB і CMYK носить абстрактний характер. Здебільшого це зв'язано з тим, що колірний тон і насиченість кольору неможливо вимірювати безпосередньо. Будь-яка форма введення колірної інформації завжди починається з визначення червоної, зеленої і синьої складових, на базі яких потім за допомогою математичного перерахунку отримують компоненти HSB-моделі. В результаті ця колірна модель має той же колірний простір, що і RGB-модель, а значить, і належний їй недолік – обмежений колірний простір.

Разом з тим HSB-модель має порівняно з RGB- і CMYK-моделями три важливі переваги:

– Апаратна залежність. Задавання складових цієї моделі у вигляді значень колірного тону, насиченості і яскравості дозволяє однозначно визначати колір без необхідності врахування параметрів пристрою виведення.

– Модель більше відповідає природі кольору, добре підходить для сприймання людиною: колірний тон є еквівалентом довжини світла, насиченість – інтенсивності хвилі, а яскравість – кількості світла.

– Модель відрізняється більш простим і інтуїтивно зрозумілим механізмом управління кольором. Це пов'язано з тим, що колірний тон, насиченість і яскравість являють собою незалежні характеристики кольору. Наприклад, чистий червоний колір розташований на колірному крузі під кутом 0°. Якщо потрібно змістити червоний тон до оранжевого тону, то потрібно лише трохи збільшити кут, який визначає колірний тон.

Для отримання більш блідого кольору достатньо лише знизити насиченість, а для надання йому більшої яскравості – відповідно збільшити значення яскравості. Отримати такі ефекти за допомогою RGB-моделі практично неможливо, оскільки значення її колірних компонентів дуже сильно залежать один від одного.

Тому при зміні однієї з складових, наприклад червоної, це вплине не тільки на колірний тон, але одночасно і на насиченість і на яскравість.

Модель Lab

Для правильного відображення кольору зручно визначити стандартну модель, до якої б приводилися кольори на всіх етапах процесу. Успішною пробою створення апаратно-незалежної моделі кольору, основаної на людському сприйнятті кольору, є модель Lab (рис. 1.30).

Будь-який колір в Lab визначається яскравістю (Lightness) та двома хроматичними компонентами: параметром a, який змінюється в діапазоні від зеленого до червоного і параметром b, який змінюється від синього до жовтого. (Визначення каналів Lab основане на тому, що точка не може бути одночасно чорною і білою, одночасно червоною і зеленою, одночасно синьою і жовтою).

Рис. 1.30. Реалізація колірної моделі Lab

Яскравість у моделі Lab повністю відділена від кольору. Це робить модель Lab зручною для регулювання контрасту, різкості та інших тонових характеристик зображення. Модель Lab є триканальною. Її кольорова гама безмірно широка і відповідає видимій кольоровій гамі для стандартного спостерігача. Гама Lab включає гами всіх інших колірних моделей, які використовуються в поліграфічному процесі.

У моделі RGB колір точки та її яскравість зв’язані між собою. Наприклад, насичені сині кольори будуть дуже темними, а насичені жовті дуже світлими. Кожна точка на RGB-зображення сприймається оком більш-менш яскравою. В створенні цієї точки беруть участь всі три кольорових канали зображення (рис. 1.31).

Якби усі три кольори сприймались як однаково яскраві, то кожний вносив би в сумарну яскравість третю частину:

$$Y=R/3+G/3+B/3.$$

Модель Lab

Для правильного відображення кольору зручно визначити стандартну модель, до якої б приводилися кольори на всіх етапах процесу. Успішною пробою створення апаратно-незалежної моделі кольору, основаної на людському сприйнятті кольору, є модель Lab (рис. 1.30).

Будь-який колір в Lab визначається яскравістю (Lightness) та двома хроматичними компонентами: параметром a, який змінюється в діапазоні від зеленого до червоного і параметром b, який змінюється від синього до жовтого. (Визначення каналів Lab основане на тому, що точка не може бути одночасно чорною і білою, одночасно червоною і зеленою, одночасно синьою і жовтою).

Рис. 1.30. Реалізація колірної моделі Lab

Яскравість у моделі Lab повністю відділена від кольору. Це робить модель Lab зручною для регулювання контрасту, різкості та інших тонових характеристик зображення. Модель Lab є триканальною. Її кольорова гама безмірно широка і відповідає видимій кольоровій гамі для стандартного спостерігача. Гама Lab включає гами всіх інших колірних моделей, які використовуються в поліграфічному процесі.

У моделі RGB колір точки та її яскравість зв’язані між собою. Наприклад, насичені сині кольори будуть дуже темними, а насичені жовті дуже світлими. Кожна точка на RGB-зображення сприймається оком більш-менш яскравою. В створенні цієї точки беруть участь всі три кольорових канали зображення (рис. 1.31).

Якби усі три кольори сприймались як однаково яскраві, то кожний вносив би в сумарну яскравість третю частину:

$$Y=R/3+G/3+B/3.$$

Саме так розраховується в колірній моделі HSB, яку ми розглядали раніше. Оскільки ми вже виявили, що різні базові кольори мають різну яскравість, яка сприймається, то розрахунок не відбиває реального положення речей, тому модель HSB неможливо вважати коректною.

Для розрахунку реальної яскравості використовується наступна емпірична формула, яка враховує вклад кожного колірного каналу:
Y=0.2125R+0.7154G+0.0721B.

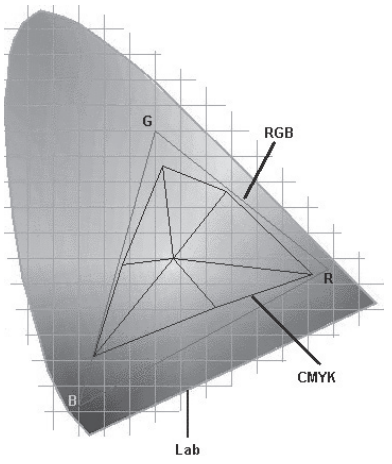


Рис. 1.31. Порівняльна характеристика колірних моделей

Безпосередньо спостерігати яскравість можна при переведенні зображення в напівтонове (Grayscale). У моделях RGB і CMYK яскравість і колір зв'язані, тобто при зміні одного параметра змінюється інший. Існує цікавий прийом, який дозволяє побачити спектр Lab-кольорів.

1. Для цього створіть нове зображення в режимі Lab, заповнене білим кольором (просто змініть модель у вказаному рядку на Lab, рис.1.32).

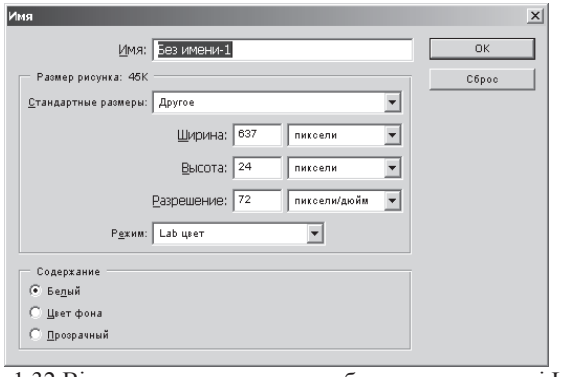


Рис. 1.32 Вікно створення нового зображення в режимі Lab

Саме так розраховується в колірній моделі HSB, яку ми розглядали раніше. Оскільки ми вже виявили, що різні базові кольори мають різну яскравість, яка сприймається, то розрахунок не відбиває реального положення речей, тому модель HSB неможливо вважати коректною.

Для розрахунку реальної яскравості використовується наступна емпірична формула, яка враховує вклад кожного колірного каналу:
Y=0.2125R+0.7154G+0.0721B.

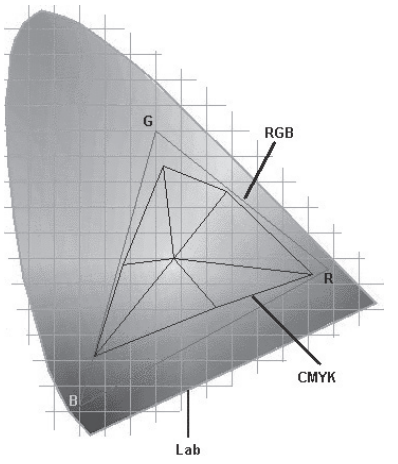


Рис. 1.31. Порівняльна характеристика колірних моделей

Безпосередньо спостерігати яскравість можна при переведенні зображення в напівтонове (Grayscale). У моделях RGB і CMYK яскравість і колір зв'язані, тобто при зміні одного параметра змінюється інший. Існує цікавий прийом, який дозволяє побачити спектр Lab-кольорів.

1. Для цього створіть нове зображення в режимі Lab, заповнене білим кольором (просто змініть модель у вказаному рядку на Lab, рис.1.32).

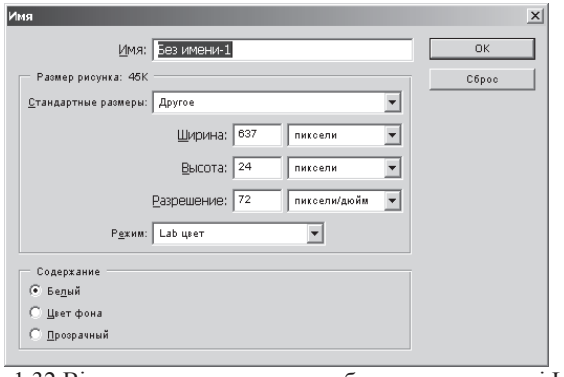


Рис. 1.32 Вікно створення нового зображення в режимі Lab

2. Відкрийте палітру Channels. Вона зображується таким чином, як показано на рис. 1.33.

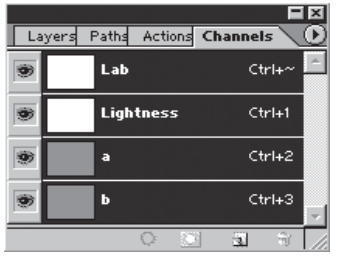


Рис. 1.33. Палітра Channels

3. Потім залийте канали *a* і *b* чорно-білим градієнтом так, як показано на наступному зображенні. Переключіться в сумісний канал Lab. Тепер ви бачите весь спектр кольорів моделі Lab максимальної яскравості, адже канал яскравості L залитий білим кольором (рис. 1.34).

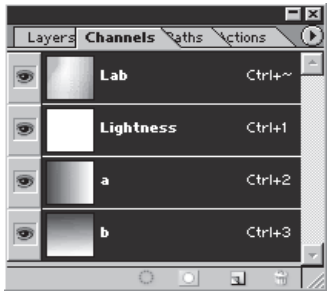


Рис. 1.34. Спектр кольорів моделі Lab

4. Як пояснити те, що ми побачили? Справа в тому, що чорний колір у каналі відповідає мінімальній яскравості цього каналу. Білий колір відповідає максимальній яскравості. Значить, в каналі *a* чорний колір відповідає зеленому зображенню, білий – червоному. Чорно-білий градієнт відобразить плавний перехід зеленого кольору в червоний. Хочете це перевірити – відключіть видимість каналу *b* і погляньте на зображення.

5. Отже, всі кольори, які знаходяться в каналі *a*, переходять один в одного по горизонталі, по вертикалі – всі кольори каналу *b*. В сумісному каналі вони змішуються у всіх можливих співвідношеннях – ми отримали спектр моделі Lab.

6. Спробуємо тепер побачити різницю у відображенні кольорів у моделях Lab і CMYK. Ця різниця досить велика, тому що модель Lab має максимальну кольорову гаму, а CMYK – мінімальну. Створіть дублікат зображення за допомогою команди Duplicate меню Image. Щоб перевести отримане зображення в колірний режим CMYK, виберіть у меню Image команду Mode > CMYK. Кольори

2. Відкрийте палітру Channels. Вона зображується таким чином, як показано на рис. 1.33.

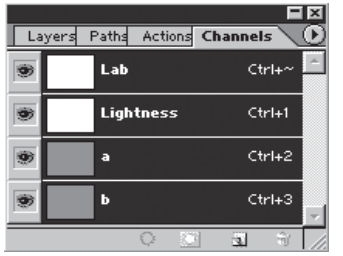


Рис. 1.33. Палітра Channels

3. Потім залийте канали *a* і *b* чорно-білим градієнтом так, як показано на наступному зображенні. Переключіться в сумісний канал Lab. Тепер ви бачите весь спектр кольорів моделі Lab максимальної яскравості, адже канал яскравості L залитий білим кольором (рис. 1.34).

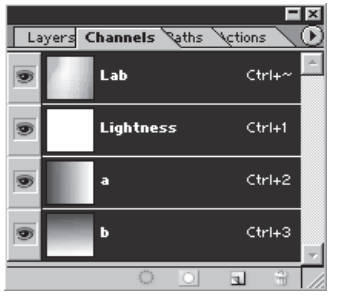


Рис. 1.34. Спектр кольорів моделі Lab

4. Як пояснити те, що ми побачили? Справа в тому, що чорний колір у каналі відповідає мінімальній яскравості цього каналу. Білий колір відповідає максимальній яскравості. Значить, в каналі *a* чорний колір відповідає зеленому зображенню, білий – червоному. Чорно-білий градієнт відобразить плавний перехід зеленого кольору в червоний. Хочете це перевірити – відключіть видимість каналу *b* і погляньте на зображення.

5. Отже, всі кольори, які знаходяться в каналі *a*, переходять один в одного по горизонталі, по вертикалі – всі кольори каналу *b*. В сумісному каналі вони змішуються у всіх можливих співвідношеннях – ми отримали спектр моделі Lab.

6. Спробуємо тепер побачити різницю у відображенні кольорів у моделях Lab і CMYK. Ця різниця досить велика, тому що модель Lab має максимальну кольорову гаму, а CMYK – мінімальну. Створіть дублікат зображення за допомогою команди Duplicate меню Image. Щоб перевести отримане зображення в колірний режим CMYK, виберіть у меню Image команду Mode > CMYK. Кольори

стали більш темними і тьмяними, хоча в моделі Lab була задана максимальна яскравість!

На відміну від RGB і CMYK, заснованих на реальних процесах, Lab являє собою чисто математичну модель. Їй важко знайти аналогічну в реальному світі. Отже, ця модель має декілька серйозних переваг:

1. Вона основана на сприйманні людиною і її кольорова гама відповідає людському оку – вона включає в себе гами RGB і CMYK і перевищує їх.

2. Lab є апаратно-незалежною моделлю.

Ці дві переваги зробили Lab стандартом при переведенні зображень з одного колірного простору в інший у процесі їх підготовки.

Цим, однак, важко пояснити, для чого про властивості Lab знати користувачу. Адже більшість з нас не хвилює, скажемо, будова файлів зображень. Однак Lab має і сугубо практичні області застосування. В цій моделі легко виконувати розповсюджені операції. В їх числі підвищення різкості, тонової корекції (підвищення контрасту, використання похибки тонових діапазонів) і видалення колірного шуму (в тому числі розмивання растра і видалення регулярної структури зображень у форматі JPEG).

Професіонали використовують цей простір навіть для створення складних масок і кардинальних змін кольорів документа. Оскільки модель має обмежену колірну гаму, переведення в ній не зв'язані з затратами. Ви можете в будь-який момент перевести зображення з RGB в Lab і назад, і при цьому його кольори не міняються.

1. 5. Апаратні засоби комп'ютерної графіки

Комп'ютер обмінюється інформацією із зовнішнім світом за допомогою периферійних пристроїв. Лише завдяки периферійним пристроям людина може взаємодіяти з комп'ютером, а також зі всіма підключеними до нього пристроями. Периферійні пристрої діляться на пристрої введення та пристрої виведення. Пристрої введення перетворять інформацію у форму, зрозумілу машині, після чого комп'ютер може її обробляти і запам'ятовувати. Пристрої виведення переводять інформацію з машинного формату в образи, зрозумілі людині. Все вищесказане відноситься і до апаратних засобів

стали більш темними і тьмяними, хоча в моделі Lab була задана максимальна яскравість!

На відміну від RGB і CMYK, заснованих на реальних процесах, Lab являє собою чисто математичну модель. Їй важко знайти аналогічну в реальному світі. Отже, ця модель має декілька серйозних переваг:

1. Вона основана на сприйманні людиною і її кольорова гама відповідає людському оку – вона включає в себе гами RGB і CMYK і перевищує їх.

2. Lab є апаратно-незалежною моделлю.

Ці дві переваги зробили Lab стандартом при переведенні зображень з одного колірного простору в інший у процесі їх підготовки.

Цим, однак, важко пояснити, для чого про властивості Lab знати користувачу. Адже більшість з нас не хвилює, скажемо, будова файлів зображень. Однак Lab має і сугубо практичні області застосування. В цій моделі легко виконувати розповсюджені операції. В їх числі підвищення різкості, тонової корекції (підвищення контрасту, використання похибки тонових діапазонів) і видалення колірного шуму (в тому числі розмивання растра і видалення регулярної структури зображень у форматі JPEG).

Професіонали використовують цей простір навіть для створення складних масок і кардинальних змін кольорів документа. Оскільки модель має обмежену колірну гаму, переведення в ній не зв'язані з затратами. Ви можете в будь-який момент перевести зображення з RGB в Lab і назад, і при цьому його кольори не міняються.

1. 5. Апаратні засоби комп'ютерної графіки

Комп'ютер обмінюється інформацією із зовнішнім світом за допомогою периферійних пристроїв. Лише завдяки периферійним пристроям людина може взаємодіяти з комп'ютером, а також зі всіма підключеними до нього пристроями. Периферійні пристрої діляться на пристрої введення та пристрої виведення. Пристрої введення перетворять інформацію у форму, зрозумілу машині, після чого комп'ютер може її обробляти і запам'ятовувати. Пристрої виведення переводять інформацію з машинного формату в образи, зрозумілі людині. Все вищесказане відноситься і до апаратних засобів

комп’ютерної графіки, які теж поділяються на дві групи – пристрої введення та виведення графічної інформації [20].

Пристрої введення

До пристроїв введення графічної інформації відносяться: сканери, дигітайзери, цифрові фотоапарати і відеокамери.

Сканери

Сканер – це пристрій введення графічних зображень у комп’ютер (рис.1.35). У сканер закладається аркуш паперу із зображенням, пристрій «прочитує» його і пересилає комп’ютеру в цифровому вигляді. Під час сканування уздовж аркуша із зображенням плавно переміщається потужна лампа і лінійка з безліччю розташованих на ній у ряд світлочутливих елементів.

Рис. 1.35. Зовнішній вигляд сканера

Зазвичай як світлочутливі елементи використовують фотодіоди. Кожен світлочутливий елемент виробляє сигнал, пропорційний яскравості відбитого світла від ділянки паперу, розташованої напроти нього. Яскравість відбитого променя міняється через те, що світлі місця сканованого зображення відображають набагато краще, ніж темні, покриті фарбою. У кольорових сканерах розташовано три групи світлочутливих елементів, що обробляють відповідно червоні, зелені і сині кольори. Таким чином, кожна точка зображення

комп’ютерної графіки, які теж поділяються на дві групи – пристрої введення та виведення графічної інформації [20].

Пристрої введення

До пристроїв введення графічної інформації відносяться: сканери, дигітайзери, цифрові фотоапарати і відеокамери.

Сканери

Сканер – це пристрій введення графічних зображень у комп’ютер (рис.1.35). У сканер закладається аркуш паперу із зображенням, пристрій «прочитує» його і пересилає комп’ютеру в цифровому вигляді. Під час сканування уздовж аркуша із зображенням плавно переміщається потужна лампа і лінійка з безліччю розташованих на ній у ряд світлочутливих елементів.

Рис. 1.35. Зовнішній вигляд сканера

Зазвичай як світлочутливі елементи використовують фотодіоди. Кожен світлочутливий елемент виробляє сигнал, пропорційний яскравості відбитого світла від ділянки паперу, розташованої напроти нього. Яскравість відбитого променя міняється через те, що світлі місця сканованого зображення відображають набагато краще, ніж темні, покриті фарбою. У кольорових сканерах розташовано три групи світлочутливих елементів, що обробляють відповідно червоні, зелені і сині кольори. Таким чином, кожна точка зображення

кодується як поєднання сигналів, що виробляються світлочутливими елементами червоної, зеленої і синьої груп. Закодований таким чином сигнал передається на контролер сканера в системний блок.

Головні характеристики сканерів - це швидкість читання, яка виражається кількістю сканованих сторінок за хвилину (pages per minute - ppm), і роздільна здатність, що виражається числом точок отримуваного зображення на дюйм оригінала (dots per inch - dpi).

Сканери поділяють на такі види:

чорно-білі сканери можуть у простому випадку розрізняти лише два значення – чорне і біле, що цілком достатньо для читання штрихових кодів (складніші сканери розрізняють градації сірого кольору);

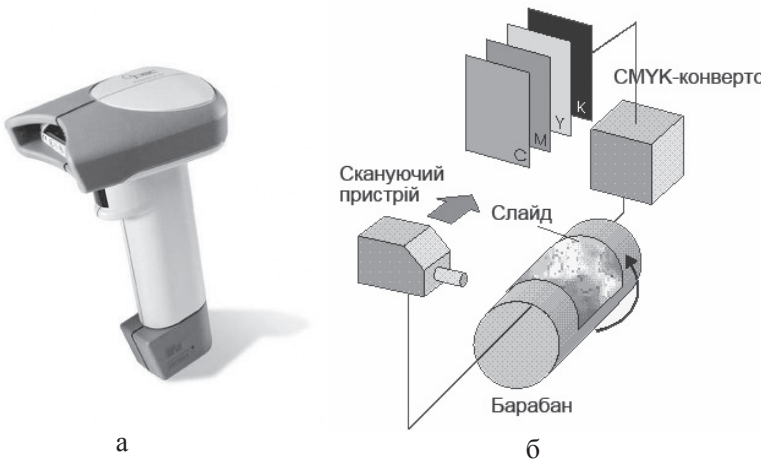


Рис. 1.36. Сканери: а – зовнішній вигляд ручного сканера;
б – принцип дії барабанного сканера

кольорові сканери працюють на принципі складання кольорів, при якому кольорове зображення виходить шляхом змішення трьох кольорів: червоного, зеленого і синього. При скануванні кольоровий оригінал освітлює не білим світлом, а послідовно червоним, зеленим і синім. Сканування здійснюється для кожного кольору окремо, отримана інформація заздалегідь обробляється і передається в

кодується як поєднання сигналів, що виробляються світлочутливими елементами червоної, зеленої і синьої груп. Закодований таким чином сигнал передається на контролер сканера в системний блок.

Головні характеристики сканерів - це швидкість читання, яка виражається кількістю сканованих сторінок за хвилину (pages per minute - ppm), і роздільна здатність, що виражається числом точок отримуваного зображення на дюйм оригінала (dots per inch - dpi).

Сканери поділяють на такі види:

чорно-білі сканери можуть у простому випадку розрізняти лише два значення – чорне і біле, що цілком достатньо для читання штрихових кодів (складніші сканери розрізняють градації сірого кольору);

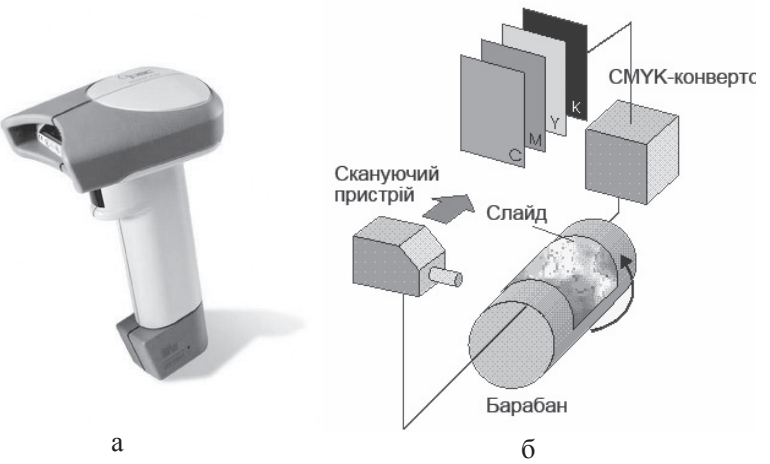


Рис. 1.36. Сканери: а – зовнішній вигляд ручного сканера;
б – принцип дії барабанного сканера

кольорові сканери працюють на принципі складання кольорів, при якому кольорове зображення виходить шляхом змішення трьох кольорів: червоного, зеленого і синього. При скануванні кольоровий оригінал освітлює не білим світлом, а послідовно червоним, зеленим і синім. Сканування здійснюється для кожного кольору окремо, отримана інформація заздалегідь обробляється і передається в

комп’ютер. У процесі сканування кольоровий оригінал освітлює білим кольором, а відбите світло потрапляє на CCD–матрицю через систему спеціальних фільтрів, розкладаючих його на три компоненти: червоний, зелений, синій, кожен з яких уловлюється своїм набором фотоелементів;

ручні сканери – відносно недорогі пристрої невеликого розміру, зручні для оперативного сканування зображень з книг і журналів. Ширина смуги сканування зазвичай не перевищує 105 мм, стандартний дозвіл 300–400 dpi. До недоліків ручного сканера можна віднести залежність якості сканування від навиків користувача і неможливість одночасного сканування відносно великих зображень;

барабанні сканери – сканований оригінал розташовується на барабані, що обертається. В даний час використовуються лише в друкарському виробництві;

листові сканери – носій із зображенням протягується уздовж лінійки, на якій розташовані CCD– елементи.

Ширина зображення, як правило, складає формат А4, а довжина обмежена можливостями використовуваного комп’ютера (чим більше зображення, тим більший розмір файла, де зберігається його цифрова копія);

а б

Рис. 1.37. Зовнішній вигляд: а – листового сканера;
б – широкоформатного сканера

планшетні сканери здійснюють сканування в автоматичному режимі. Оригінал розташовується в сканері на скляному аркуші, під

комп’ютер. У процесі сканування кольоровий оригінал освітлює білим кольором, а відбите світло потрапляє на CCD–матрицю через систему спеціальних фільтрів, розкладаючих його на три компоненти: червоний, зелений, синій, кожен з яких уловлюється своїм набором фотоелементів;

ручні сканери – відносно недорогі пристрої невеликого розміру, зручні для оперативного сканування зображень з книг і журналів. Ширина смуги сканування зазвичай не перевищує 105 мм, стандартний дозвіл 300–400 dpi. До недоліків ручного сканера можна віднести залежність якості сканування від навиків користувача і неможливість одночасного сканування відносно великих зображень;

барабанні сканери – сканований оригінал розташовується на барабані, що обертається. В даний час використовуються лише в друкарському виробництві;

листові сканери – носій із зображенням протягується уздовж лінійки, на якій розташовані CCD– елементи.

Ширина зображення, як правило, складає формат А4, а довжина обмежена можливостями використовуваного комп’ютера (чим більше зображення, тим більший розмір файла, де зберігається його цифрова копія);

а б

Рис. 1.37. Зовнішній вигляд: а – листового сканера;
б – широкоформатного сканера

планшетні сканери здійснюють сканування в автоматичному режимі. Оригінал розташовується в сканері на скляному аркуші, під

яким голівка читання з CCD - елементами сканує зображення з рівномірною швидкістю. Розміри сканованих зображень залежать від розміру сканера і можуть досягати розмірів великого креслярського аркуша (A0) (широкоформатні сканери див. рис. 1.37, б).

Спеціальна слайд-приставка дозволяє сканувати слайди і негативні плівки. Апаратний дозвіл планшетних сканерів досягає 1200dpi. Сканери підключаються до персонального комп'ютера через спеціальний контролер (для планшетних сканерів це частіше всього SCSI контролер). Сканер завжди повинен мати відповідний драйвер, оскільки лише обмежене число програмних додатків має вбудовані драйвери для спілкування з певним класом сканерів.

На практиці широке вживання знайшли також технології тривимірного сканування:

– ультразвукове сканування;
– магнітне сканування;
– лазерні сканери.

Зі всіх тривимірних технологій сканування ультразвукові системи найменш точні, найменш надійні і найбільш сприйнятливі до геометричних спотворень. Унаслідок того, що швидкість звуку залежить від повітряного тиску, температури та інших атмосферних умов, ефективність ультразвукових систем може змінюватися разом з погодою. Крім того, вони сприйнятливі до роботи різного устаткування, навіть шуму ламп денного світла.

Магнітні тривимірні цифрові перетворювачі працюють на тому ж принципі, що і «ультразвукові системи», тобто використовують магнітне поле. Вони несприйнятливі до атмосферних змін і дуже чутливі до спотворень від довколишнього металу або магнітних полів. Металеві стільці, плати, комп'ютери або інше устаткування, розміщені близько від магнітного цифрового перетворювача, спотворять дані. Крім того, такі системи не можна використовувати для оцифровування об'єктів з металевими частинами.

Лазерні сканери в 10–100 разів дорожчі, ніж системи механічного оцифровування. Системи, що використовують лазери, мають багато обмежень. Об'єкти з віддзеркалювальними або яскравими поверхнями, великі об'єкти і об'єкти з увігнутими поверхнями, які

яким голівка читання з CCD - елементами сканує зображення з рівномірною швидкістю. Розміри сканованих зображень залежать від розміру сканера і можуть досягати розмірів великого креслярського аркуша (A0) (широкоформатні сканери див. рис. 1.37, б).

Спеціальна слайд-приставка дозволяє сканувати слайди і негативні плівки. Апаратний дозвіл планшетних сканерів досягає 1200dpi. Сканери підключаються до персонального комп'ютера через спеціальний контролер (для планшетних сканерів це частіше всього SCSI контролер). Сканер завжди повинен мати відповідний драйвер, оскільки лише обмежене число програмних додатків має вбудовані драйвери для спілкування з певним класом сканерів.

На практиці широке вживання знайшли також технології тривимірного сканування:

– ультразвукове сканування;
– магнітне сканування;
– лазерні сканери.

Зі всіх тривимірних технологій сканування ультразвукові системи найменш точні, найменш надійні і найбільш сприйнятливі до геометричних спотворень. Унаслідок того, що швидкість звуку залежить від повітряного тиску, температури та інших атмосферних умов, ефективність ультразвукових систем може змінюватися разом з погодою. Крім того, вони сприйнятливі до роботи різного устаткування, навіть шуму ламп денного світла.

Магнітні тривимірні цифрові перетворювачі працюють на тому ж принципі, що і «ультразвукові системи», тобто використовують магнітне поле. Вони несприйнятливі до атмосферних змін і дуже чутливі до спотворень від довколишнього металу або магнітних полів. Металеві стільці, плати, комп'ютери або інше устаткування, розміщені близько від магнітного цифрового перетворювача, спотворять дані. Крім того, такі системи не можна використовувати для оцифровування об'єктів з металевими частинами.

Лазерні сканери в 10–100 разів дорожчі, ніж системи механічного оцифровування. Системи, що використовують лазери, мають багато обмежень. Об'єкти з віддзеркалювальними або яскравими поверхнями, великі об'єкти і об'єкти з увігнутими поверхнями, які

затінюють пряму дорогу лазерного променя, – головна проблема для лазерних систем.

Дигітайзери

Наступний тип пристроїв введення графічної інформації – це дигітайзери. Дигітайзер призначений для професійних графічних робіт. За допомогою спеціального програмного забезпечення він дозволяє перетворювати рух руки оператора у формат векторної графіки. Спочатку дигітайзер був розроблений для додатків систем автоматизованого проектування, оскільки в цьому випадку необхідно визначати і задавати точне значення координат великої кількості точок. На відміну від миші дигітайзер здатний точно визначати і обробляти абсолютні координати.

Рис. 1.38. Зовнішній вигляд дигітайзера

Дигітайзер складається із спеціального планшета, що є робочою поверхнею і виконує всілякі функції управління відповідним програмним забезпеченням, і світлового пера або, частіше, кругового курсору, що є пристроями введення інформації (рис. 1.38).

Одним із різновидів дигітайзера є графічний або малювальний планшет. Він є панеллю, під якою розташовані електромагнітні решітки (рис. 1.39). Якщо провести по його поверхні спеціальним пером, то на екрані монітора з’явиться штрих.

затінюють пряму дорогу лазерного променя, – головна проблема для лазерних систем.

Дигітайзери

Наступний тип пристроїв введення графічної інформації – це дигітайзери. Дигітайзер призначений для професійних графічних робіт. За допомогою спеціального програмного забезпечення він дозволяє перетворювати рух руки оператора у формат векторної графіки. Спочатку дигітайзер був розроблений для додатків систем автоматизованого проектування, оскільки в цьому випадку необхідно визначати і задавати точне значення координат великої кількості точок. На відміну від миші дигітайзер здатний точно визначати і обробляти абсолютні координати.

Рис. 1.38. Зовнішній вигляд дигітайзера

Дигітайзер складається із спеціального планшета, що є робочою поверхнею і виконує всілякі функції управління відповідним програмним забезпеченням, і світлового пера або, частіше, кругового курсору, що є пристроями введення інформації (рис. 1.38).

Одним із різновидів дигітайзера є графічний або малювальний планшет. Він є панеллю, під якою розташовані електромагнітні решітки (рис. 1.39). Якщо провести по його поверхні спеціальним пером, то на екрані монітора з’явиться штрих.

У планшеті реалізований принцип абсолютного позиціювання. Зображення, намальоване в лівому нижньому куті планшета, з'явиться в лівому нижньому кутку екрана монітора. Зазвичай малювальні планшети мають розміри килимка для миші, але робоча поверхня дещо менше.

Є планшети, яким притаманна чутливість до натиску, за допомогою яких, регулюючи натиск, можна отримувати на екрані лінії різної товщини. Спеціальна пластмасова плівка, що додається до планшета, дозволяє копіювати підкладені під неї зображення на паперових оригіналах. Планшети підключаються до послідовного порту персонального комп'ютера.

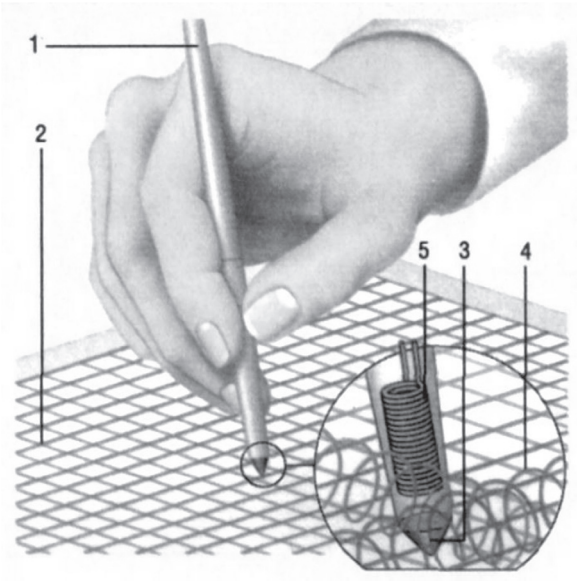


Рис. 1.39. Принцип дії графічного планшету

Графічний планшет може мати різні формати: від А2 – для професійної діяльності і менше – для простіших робіт.

У планшеті реалізований принцип абсолютного позиціювання. Зображення, намальоване в лівому нижньому куті планшета, з'явиться в лівому нижньому кутку екрана монітора. Зазвичай малювальні планшети мають розміри килимка для миші, але робоча поверхня дещо менше.

Є планшети, яким притаманна чутливість до натиску, за допомогою яких, регулюючи натиск, можна отримувати на екрані лінії різної товщини. Спеціальна пластмасова плівка, що додається до планшета, дозволяє копіювати підкладені під неї зображення на паперових оригіналах. Планшети підключаються до послідовного порту персонального комп'ютера.

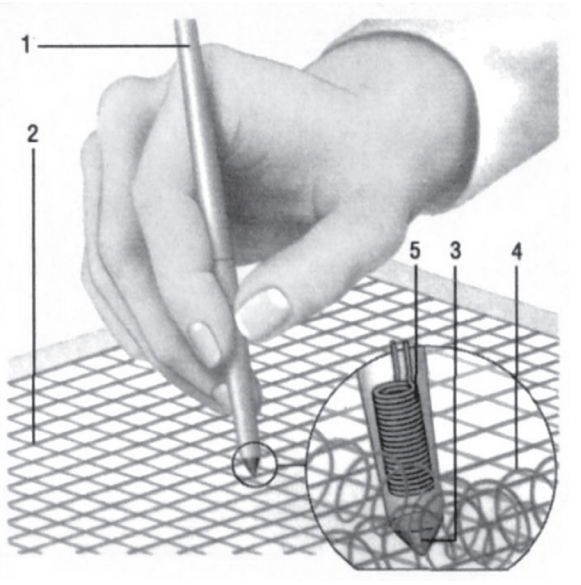


Рис. 1.39. Принцип дії графічного планшету

Графічний планшет може мати різні формати: від А2 – для професійної діяльності і менше – для простіших робіт.

Маніпулятор «миша»

Поряд з клавіатурою миша є найважливішим засобом введення інформації. У сучасних програмних продуктах, що мають складну графічну оболонку, миша є основним інструментом управління програмою.

За принципом дії миші діляться на механічні, оптико-механічні, оптичні.

Переважна кількість комп'ютерних мишок використовують оптико-механічний принцип кодування переміщення. З поверхнею столу стикається важка, покрита гумою кулька порівняно великого діаметру. Ролики, притиснуті до поверхні кульки, встановлені на перпендикулярних один одному осях з двома датчиками. Датчики, що є оптопарами (світлодіод-фотодіод), розташовуються по різні сторони дисків з прорізами. Порядок, в якому освітлюють фоточутливі елементи, визначає напрям переміщення миші, а частота імпульсів, що приходять від них, – швидкість. Хороший механічний контакт з поверхнею забезпечує спеціальний килимок.

Більш точного позиціювання курсору дозволяє добитися *оптична миша*. Для неї використовується спеціальний килимок, на поверхні якого нанесена найдрібніша сітка з перпендикулярних один одному темних і світлих смуг. Розташовані в нижній частині миші дві оптопари освітлюють килимок і по числу пересічених при русі ліній визначають величину і швидкість переміщення.

Оптичні миші не мають рухомих частин і позбавлені такого властивого оптико–механічним мишам недоліку, як переміщення курсору миші ривками через забруднення кульки. Роздільна здатність вживаного в миші пристрою прочитування координат складає 400 dpi і вище, перевершуючи аналогічні значення для механічних пристроїв.

Для оптимального функціонування миша повинна пересуватися по рівній поверхні. Покажчик миші пересувається по екрану синхронно з рухом миші по килимку. Пристроєм введення миші є кнопки (клавіші). Більшість мишей мають дві кнопки, існують також 3–кнопкові миші і менші, що мають більшу кількість кнопок.

Маніпулятор «миша»

Поряд з клавіатурою миша є найважливішим засобом введення інформації. У сучасних програмних продуктах, що мають складну графічну оболонку, миша є основним інструментом управління програмою.

За принципом дії миші діляться на механічні, оптико-механічні, оптичні.

Переважна кількість комп'ютерних мишок використовують оптико-механічний принцип кодування переміщення. З поверхнею столу стикається важка, покрита гумою кулька порівняно великого діаметру. Ролики, притиснуті до поверхні кульки, встановлені на перпендикулярних один одному осях з двома датчиками. Датчики, що є оптопарами (світлодіод-фотодіод), розташовуються по різні сторони дисків з прорізами. Порядок, в якому освітлюють фоточутливі елементи, визначає напрям переміщення миші, а частота імпульсів, що приходять від них, – швидкість. Хороший механічний контакт з поверхнею забезпечує спеціальний килимок.

Більш точного позиціювання курсору дозволяє добитися *оптична миша*. Для неї використовується спеціальний килимок, на поверхні якого нанесена найдрібніша сітка з перпендикулярних один одному темних і світлих смуг. Розташовані в нижній частині миші дві оптопари освітлюють килимок і по числу пересічених при русі ліній визначають величину і швидкість переміщення.

Оптичні миші не мають рухомих частин і позбавлені такого властивого оптико–механічним мишам недоліку, як переміщення курсору миші ривками через забруднення кульки. Роздільна здатність вживаного в миші пристрою прочитування координат складає 400 dpi і вище, перевершуючи аналогічні значення для механічних пристроїв.

Для оптимального функціонування миша повинна пересуватися по рівній поверхні. Покажчик миші пересувається по екрану синхронно з рухом миші по килимку. Пристроєм введення миші є кнопки (клавіші). Більшість мишей мають дві кнопки, існують також 3–кнопкові миші і менші, що мають більшу кількість кнопок.

Однією з важливих характеристик миші є її дозвіл, який вимірюється в dpi. Дозвіл визначає мінімальне переміщення, яке здатний відчути контролер миші. Чим більше дозвіл, тим точніше позиціює миша, тим з дрібнішими об'єктами можна працювати. Нормальний дозвіл миші лежить у діапазоні від 300 до 900 dpi. У вдосконалених мишах використовують змінний балістичний ефект швидкості, що полягає в тому, що при невеликих переміщеннях швидкість зсуву курсору, – невелика, а при значних переміщеннях – істотно збільшується. Це дозволяє ефективніше працювати в графічних пакетах, де доводиться обробляти дрібні деталі.

За принципом передачі інформації миші поділяються на послідовні (Serinl Mouse), такі, що підключаються до послідовного порту COM, та паралельні (Bus Mouse), що використовують системну шину. Bus Mouse підключається до спеціальної карти розширення, що входить до комплекту постачання миші.

Паралельні миші використовуються переважно в тих системах, де до комп'ютера потрібно підключити багато периферійних пристроїв, що особливо займають послідовні порти і де комп'ютер схильний до конфліктів переривань периферійних пристроїв (Bus Mouse не використовує переривання).

Існує декілька стандартів послідовних мишей. Найпоширенішим є стандарт MS–mouse. Альтернативними стандартами є PC–mouse, використовуваний для трикнопкових мишей фірми Genius, і рідко використовуваний PS/2. MS–mouse і сумісні з нею PC–mouse для роботи вимагають установки відповідних драйверів. Більшість програмного забезпечення для персональних комп'ютерів орієнтована на MS–mouse. Стандарт PS/2 не вимагає підключення драйверів. До основних тенденцій розвитку сучасних мишей можна віднести поступовий перехід на шину USB, а також пошуки в області ергономічних удосконалень. До них можна віднести бездротові (Cordless) миші, що працюють в радіо- або інфрачервоному діапазоні хвиль, а також миші з додатковими кнопками.

Найбільш вдалими рішеннями є наявність між двома стандартними кнопками коліщатка (миша Microsoft Intellimouse) або середньої кнопки (миші Genius Netmouse Netmouse Pro), що коливається, які використовуються для швидкої прокрутки документа під

Однією з важливих характеристик миші є її дозвіл, який вимірюється в dpi. Дозвіл визначає мінімальне переміщення, яке здатний відчути контролер миші. Чим більше дозвіл, тим точніше позиціює миша, тим з дрібнішими об'єктами можна працювати. Нормальний дозвіл миші лежить у діапазоні від 300 до 900 dpi. У вдосконалених мишах використовують змінний балістичний ефект швидкості, що полягає в тому, що при невеликих переміщеннях швидкість зсуву курсору, – невелика, а при значних переміщеннях – істотно збільшується. Це дозволяє ефективніше працювати в графічних пакетах, де доводиться обробляти дрібні деталі.

За принципом передачі інформації миші поділяються на послідовні (Serinl Mouse), такі, що підключаються до послідовного порту COM, та паралельні (Bus Mouse), що використовують системну шину. Bus Mouse підключається до спеціальної карти розширення, що входить до комплекту постачання миші.

Паралельні миші використовуються переважно в тих системах, де до комп'ютера потрібно підключити багато периферійних пристроїв, що особливо займають послідовні порти і де комп'ютер схильний до конфліктів переривань периферійних пристроїв (Bus Mouse не використовує переривання).

Існує декілька стандартів послідовних мишей. Найпоширенішим є стандарт MS–mouse. Альтернативними стандартами є PC–mouse, використовуваний для трикнопкових мишей фірми Genius, і рідко використовуваний PS/2. MS–mouse і сумісні з нею PC–mouse для роботи вимагають установки відповідних драйверів. Більшість програмного забезпечення для персональних комп'ютерів орієнтована на MS–mouse. Стандарт PS/2 не вимагає підключення драйверів. До основних тенденцій розвитку сучасних мишей можна віднести поступовий перехід на шину USB, а також пошуки в області ергономічних удосконалень. До них можна віднести бездротові (Cordless) миші, що працюють в радіо- або інфрачервоному діапазоні хвиль, а також миші з додатковими кнопками.

Найбільш вдалими рішеннями є наявність між двома стандартними кнопками коліщатка (миша Microsoft Intellimouse) або середньої кнопки (миші Genius Netmouse Netmouse Pro), що коливається, які використовуються для швидкої прокрутки документа під

Windows 95/98/nt. До найбільш відомих виробників мишей відносяться компанії Genius, Logitech, Microsoft, Mitsumi та ін.

Джойстики

Джойстик є координатним пристроєм введення інформації і найчастіше застосовується в області комп'ютерних ігор і комп'ютерних тренажерів. Джойстики бувають аналогові (зазвичай використовуються в комп'ютерних тренажерах) та цифрові (в ігрових комп'ютерах).

Аналогові джойстики забезпечують точніше управління, що дуже важливе для програмних додатків, у яких об'єкти повинні точно позиціюватися. Для зручності роботи конструкція джойстика має бути досить міцною і стійкою. Джойстик підключають до зовнішнього роз'єму карти розширення, що має відповідний порт. Для того щоб підключити джойстик до комп'ютера, потрібний ігровий порт.

Ігровий порт (або адаптер) може бути розташований на платі асинхронного послідовного адаптера, на платі мультипорта або на окремій платі. Інколи ігровий порт може бути розташований і на системній платі комп'ютера. Джойстик підключається до комп'ютера через ігровий порт. До одного ігрового порту може бути підключено два джойстики. Процедура підключення джойстика вельми проста. Все, що потрібне, – це вставити роз'єм на кінці шнура джойстика в роз'єм ігрового порту. Цей роз'єм зовні нагадує роз'єм послідовного порту, але має 15 виводів. Робота з джойстиком не вимагає підключення додаткового драйвера. Досить налагодити вашу ігрову програму на його використання.

Трекбол

Трекбол (Trackball) – це пристрій введення інформації, який можна уявити у вигляді перевернутої миші з кулькою великого розміру. Принцип дії і спосіб передачі даних трекбола такий же, як і миші. Найчастіше використовується оптико–механічний принцип реєстрації положення кульки. Підключення трекбола, як правило, здійснюється через послідовний порт.

Основними відмінностями від миші є стабільність положення за рахунок нерухомого корпусу та не потрібен майданчик для руху,

Windows 95/98/nt. До найбільш відомих виробників мишей відносяться компанії Genius, Logitech, Microsoft, Mitsumi та ін.

Джойстики

Джойстик є координатним пристроєм введення інформації і найчастіше застосовується в області комп'ютерних ігор і комп'ютерних тренажерів. Джойстики бувають аналогові (зазвичай використовуються в комп'ютерних тренажерах) та цифрові (в ігрових комп'ютерах).

Аналогові джойстики забезпечують точніше управління, що дуже важливе для програмних додатків, у яких об'єкти повинні точно позиціюватися. Для зручності роботи конструкція джойстика має бути досить міцною і стійкою. Джойстик підключають до зовнішнього роз'єму карти розширення, що має відповідний порт. Для того щоб підключити джойстик до комп'ютера, потрібний ігровий порт.

Ігровий порт (або адаптер) може бути розташований на платі асинхронного послідовного адаптера, на платі мультипорта або на окремій платі. Інколи ігровий порт може бути розташований і на системній платі комп'ютера. Джойстик підключається до комп'ютера через ігровий порт. До одного ігрового порту може бути підключено два джойстики. Процедура підключення джойстика вельми проста. Все, що потрібне, – це вставити роз'єм на кінці шнура джойстика в роз'єм ігрового порту. Цей роз'єм зовні нагадує роз'єм послідовного порту, але має 15 виводів. Робота з джойстиком не вимагає підключення додаткового драйвера. Досить налагодити вашу ігрову програму на його використання.

Трекбол

Трекбол (Trackball) – це пристрій введення інформації, який можна уявити у вигляді перевернутої миші з кулькою великого розміру. Принцип дії і спосіб передачі даних трекбола такий же, як і миші. Найчастіше використовується оптико–механічний принцип реєстрації положення кульки. Підключення трекбола, як правило, здійснюється через послідовний порт.

Основними відмінностями від миші є стабільність положення за рахунок нерухомого корпусу та не потрібен майданчик для руху,

оскільки позиція курсору розраховується по обертанню кульки. Перший пристрій подібного типу був розроблений компанією Logitech. Мініатюрні трекболи набули спочатку широкого поширення в портативних ПК. Вбудовані трекболи можуть розташовуватися в різних місцях корпусу ноутбука, зовнішні кріпляться спеціальним затиском, а до інтерфейсу підключаються кабелем.

Великого поширення в ноутбуках трекболи не набули через свій недолік поступового забруднення поверхні кулі і направляючих роликів, які буває важко очистити і, отже, повернути трекболу початкову точність. Згодом їх замінили тачпади і трекпойнти.

Тачпад і трекпойнт

Трекпойнт (Trackpoint) – координатний пристрій, що вперше з'явився в ноутбуках IBM. Це мініатюрний джойстик з шершавою вершиною діаметром 5–8 мм. Трекпойнт розташований на клавіатурі між клавішами і управляється натисненням пальця. Тачпад (Touchpad) є чутливим контактним майданчиком, рух пальця по якому викликає переміщення курсору.

У переважній більшості сучасних ноутбуків застосовується саме цей вказівний пристрій, що має не найвищий дозвіл, але має найвищу надійність через відсутність рухомих частин. Touchpad підтримує наступні протоколи: PS/2; RS–232; ADB – протокол, використовуваний комп'ютерами сімейства Apple Macintosh. У кожному з цих випадків Touchpad підтримує індустріальний стандарт «mouse» плюс власні, специфічні, розширені протоколи. Підтримка «mouse» означає, що, підключивши до комп'ютера Touchpad, ви відразу можете використовувати її як звичайну «мишку», без інсталяції її власного драйвера. Подальшим розвитком Touchpad є Touchwriter – панель Touchpad з підвищеною чутливістю, що однаково добре працює як з пальцем, так і з спеціальною ручкою і навіть з нігтем. Ця панель дозволяє вводити дані звичним для людини чином – записуючи їх ручкою. Крім того, її можна використовувати для створення графічних зображень або для підписання документів. Обидва ці пристрої передбачають наявність певного тренування для поводження з ними, проте за надійністю і малогабаритністю залишаються поза конкуренцією.

оскільки позиція курсору розраховується по обертанню кульки. Перший пристрій подібного типу був розроблений компанією Logitech. Мініатюрні трекболи набули спочатку широкого поширення в портативних ПК. Вбудовані трекболи можуть розташовуватися в різних місцях корпусу ноутбука, зовнішні кріпляться спеціальним затиском, а до інтерфейсу підключаються кабелем.

Великого поширення в ноутбуках трекболи не набули через свій недолік поступового забруднення поверхні кулі і направляючих роликів, які буває важко очистити і, отже, повернути трекболу початкову точність. Згодом їх замінили тачпади і трекпойнти.

Тачпад і трекпойнт

Трекпойнт (Trackpoint) – координатний пристрій, що вперше з'явився в ноутбуках IBM. Це мініатюрний джойстик з шершавою вершиною діаметром 5–8 мм. Трекпойнт розташований на клавіатурі між клавішами і управляється натисненням пальця. Тачпад (Touchpad) є чутливим контактним майданчиком, рух пальця по якому викликає переміщення курсору.

У переважній більшості сучасних ноутбуків застосовується саме цей вказівний пристрій, що має не найвищий дозвіл, але має найвищу надійність через відсутність рухомих частин. Touchpad підтримує наступні протоколи: PS/2; RS–232; ADB – протокол, використовуваний комп'ютерами сімейства Apple Macintosh. У кожному з цих випадків Touchpad підтримує індустріальний стандарт «mouse» плюс власні, специфічні, розширені протоколи. Підтримка «mouse» означає, що, підключивши до комп'ютера Touchpad, ви відразу можете використовувати її як звичайну «мишку», без інсталяції її власного драйвера. Подальшим розвитком Touchpad є Touchwriter – панель Touchpad з підвищеною чутливістю, що однаково добре працює як з пальцем, так і з спеціальною ручкою і навіть з нігтем. Ця панель дозволяє вводити дані звичним для людини чином – записуючи їх ручкою. Крім того, її можна використовувати для створення графічних зображень або для підписання документів. Обидва ці пристрої передбачають наявність певного тренування для поводження з ними, проте за надійністю і малогабаритністю залишаються поза конкуренцією.

Засоби діалогу для систем віртуальної реальності

У системах віртуальної реальності, на відміну від звичайних додатків комп'ютерної графіки, як правило, потрібний вивід і введення тривимірної координатної інформації як для управління положеннями об'єктів, що синтезуються, так і для визначення координат частин тіла оператора і напряму його погляду.

Спейсбол

Одним з перших з'явився пристрій спейсбол (space ball), що є конструктивним об'єднанням миші і невеликого трекбола. Миша переміщається оператором по столу і забезпечує введення двох координат. Введення третьої координати забезпечується обертанням кульки трекбола великим пальцем руки.

Для маніпулювання об'єктами в тривимірному просторі часто використовується техніка віртуальної сфери. Керований об'єкт оточується сферою (уявною). Для переміщення сфери використовується миша, а обертання сфери і укладеного в неї об'єкта забезпечується обертанням кульки трекбола.

Head Mounted Display

У системах віртуальної реальності використовуються пристрої введення-виведення у вигляді вмонтованих на голові дисплеїв (Head Mounted Display – HMD) з бінокулярним всенаправленим монітором (Binocular Omni–orientation Monitor – BOOM) із засобами відстежування положення голови (head tracking) і навіть з відстежуванням положення ока (eye tracking).

Засоби діалогу для систем віртуальної реальності

У системах віртуальної реальності, на відміну від звичайних додатків комп'ютерної графіки, як правило, потрібний вивід і введення тривимірної координатної інформації як для управління положеннями об'єктів, що синтезуються, так і для визначення координат частин тіла оператора і напряму його погляду.

Спейсбол

Одним з перших з'явився пристрій спейсбол (space ball), що є конструктивним об'єднанням миші і невеликого трекбола. Миша переміщається оператором по столу і забезпечує введення двох координат. Введення третьої координати забезпечується обертанням кульки трекбола великим пальцем руки.

Для маніпулювання об'єктами в тривимірному просторі часто використовується техніка віртуальної сфери. Керований об'єкт оточується сферою (уявною). Для переміщення сфери використовується миша, а обертання сфери і укладеного в неї об'єкта забезпечується обертанням кульки трекбола.

Head Mounted Display

У системах віртуальної реальності використовуються пристрої введення-виведення у вигляді вмонтованих на голові дисплеїв (Head Mounted Display – HMD) з бінокулярним всенаправленим монітором (Binocular Omni–orientation Monitor – BOOM) із засобами відстежування положення голови (head tracking) і навіть з відстежуванням положення ока (eye tracking).

Рис. 1.40. Head Mounted Display

Це потрібно для створення ефекту «занурення» із стереоскопічним зображенням і оперативною зміною сцени при поворотах голови і очей.

Використовувані в HMD рідкокристалічні дисплеї зазвичай невисокого дозволу (до 417×277 пікселів). Тому ведуться інтенсивні дослідження по створенню засобів відображення для систем віртуальної реальності, що володіють високим дозволом при прийнятних значеннях електромагнітних наведень. Одна з таких систем, що використовують мініатюрні монохромні прецизійні електронні трубки і рідкокристалічні затвори, забезпечує дозвіл до 2000×2000.

Цікаве рішення полягає у формуванні зображення лазером безпосередньо на сітківці, але ці пропозиції доки далекі від комерційної реалізації.

Відстежування положення голови забезпечується або механічними системами важелів, або комплектом інфрачервоних або електромагнітних датчиків.

Power Glove, Date Glove, Date Suit

Безпосереднє введення геометричної інформації про положення частин тіла з підтримкою тактильного і навіть силового зворотного зв'язку забезпечується рукавичками і костюмами даних. Дешева рукавичка даних – Power Glove, використовувана для ігор, забезпечує лише чотири рівні даних.

Рис. 1.40. Head Mounted Display

Це потрібно для створення ефекту «занурення» із стереоскопічним зображенням і оперативною зміною сцени при поворотах голови і очей.

Використовувані в HMD рідкокристалічні дисплеї зазвичай невисокого дозволу (до 417×277 пікселів). Тому ведуться інтенсивні дослідження по створенню засобів відображення для систем віртуальної реальності, що володіють високим дозволом при прийнятних значеннях електромагнітних наведень. Одна з таких систем, що використовують мініатюрні монохромні прецизійні електронні трубки і рідкокристалічні затвори, забезпечує дозвіл до 2000×2000.

Цікаве рішення полягає у формуванні зображення лазером безпосередньо на сітківці, але ці пропозиції доки далекі від комерційної реалізації.

Відстежування положення голови забезпечується або механічними системами важелів, або комплектом інфрачервоних або електромагнітних датчиків.

Power Glove, Date Glove, Date Suit

Безпосереднє введення геометричної інформації про положення частин тіла з підтримкою тактильного і навіть силового зворотного зв'язку забезпечується рукавичками і костюмами даних. Дешева рукавичка даних – Power Glove, використовувана для ігор, забезпечує лише чотири рівні даних.

У більш вдосконаленій рукавичці даних Date Glove фірми VPL (рис. 1.41) для визначення кутів згинання пальців використовуються оптичні волокна. Для забезпечення тактильного зворотного зв’язку використовуються пневматичні активатори. Були експерименти забезпечення тактильного зворотного зв’язку за рахунок вібрації п’єзокристалів.

Більш точне введення координатної інформації забезпечують системи з використанням механічного важеля екзоскелету руки (Exos Dexterous Handmaster) і з датчиками кутів згинання пальців на основі ефекту Хола.

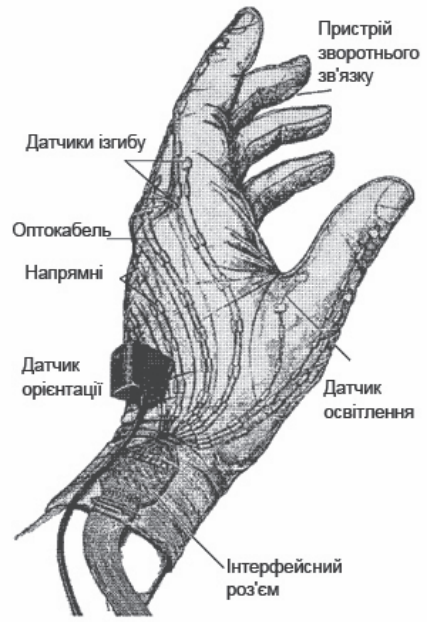


Рис. 1.41. Рукавичка даних

Системи з екзоскелетом дозволяють забезпечити і силовий зворотний зв’язок.

Простіший прилад, що використовує силовий зворотний зв’язок, був розроблений фірмою Digital і є рукояткою, подібною до рукоятки газу в мотоциклі, яка може міняти свій опір при скручуванні. Проблема в забезпеченні тактильного і силового зворотного зв’язку полягає в тому, що користувач реагує на дії і вносить зміни швидше, ніж система зможе зреагувати.

Для хорошого відчуття об’єкта система тактильного зворотного зв’язку повинна забезпечувати швидкість близько 100–300 Гц, що майже на порядок вище за звичайну швидкість перезапису візуальної інформації. Піджак даних (Date Suit) за принципом дії подібний до рукавички даних і відрізняється лише кількістю датчиків.

У більш вдосконаленій рукавичці даних Date Glove фірми VPL (рис. 1.41) для визначення кутів згинання пальців використовуються оптичні волокна. Для забезпечення тактильного зворотного зв’язку використовуються пневматичні активатори. Були експерименти забезпечення тактильного зворотного зв’язку за рахунок вібрації п’єзокристалів.

Більш точне введення координатної інформації забезпечують системи з використанням механічного важеля екзоскелету руки (Exos Dexterous Handmaster) і з датчиками кутів згинання пальців на основі ефекту Хола.

Рис. 1.41. Рукавичка даних

Системи з екзоскелетом дозволяють забезпечити і силовий зворотний зв’язок.

Простіший прилад, що використовує силовий зворотний зв’язок, був розроблений фірмою Digital і є рукояткою, подібною до рукоятки газу в мотоциклі, яка може міняти свій опір при скручуванні. Проблема в забезпеченні тактильного і силового зворотного зв’язку полягає в тому, що користувач реагує на дії і вносить зміни швидше, ніж система зможе зреагувати.

Для хорошого відчуття об’єкта система тактильного зворотного зв’язку повинна забезпечувати швидкість близько 100–300 Гц, що майже на порядок вище за звичайну швидкість перезапису візуальної інформації. Піджак даних (Date Suit) за принципом дії подібний до рукавички даних і відрізняється лише кількістю датчиків.

Цифрові фотоапарати і відеокамери

Цифрові фотоапарати і відеокамери аналогічні за принципами дії (система лінз, що проектує світло, яке потрапляє до об'єктива, на невеликий плоский майданчик) традиційним аналоговим фото- і відеокамерам відповідно.

Основна відмінність полягає в тому, що в аналогових пристроях на цьому майданчику знаходиться кадр світлочутливої плівки, а в цифрових пристроях - світлочутлива матриця ПЗС (англ. CCD) або КМОП (англ. CMOS) сенсорів. У фотоапаратах із сенсорами ПЗС формування кольорового зображення будується в спеціальному пристрої після перетворення аналогового сигналу в цифровий сигнал. У фотоапаратах із сенсорами архітектури КМОП змішання кольорів може вироблятися безпосередньо в самому сенсорі. Але й у тому, і в іншому випадку кольорове зображення будується по спеціальних математичних алгоритмах методом інтерполяції - тобто з урахуванням яскравості сусідніх елементів кожного з базових кольорів. Оскільки ці сенсори чутливі лише до яскравості, то для отримання кольорового зображення застосовують світлофільтри. Існує два підходи.

Макросвітлофільтри. Перший варіант. Для здобуття одного зображення інформація знімається з матриці сенсора 3 рази підряд, при цьому кожного разу використовується світлофільтр для одного з базисних кольорів RGB. Цей процес порівняно повільний і складний, тому така технологія застосовується лише в high-end цифрових фотокамерах при зйомці нерухомих об'єктів. Другий варіант. За допомогою призми світловий потік, що поступає, розкладається на три, що відповідають базисним кольорам RGB. Кожен з цих потоків прямує на свою ПЗС-матрицю. Тому ця технологія отримала назву 3ccd. Вона використовується у відеокамерах верхнього цінового діапазону. Для фотографії вона доки дуже дорога.

Мікросвітлофільтри. Кожен елемент ПЗС матриці має свій світлофільтр, відповідний одному з базисних кольорів RGB. Зазвичай у квадраті 2×2 розташовуються 2 зелених, 1 синій і 1 червоний елемент, так звана маска Байєра (англ. Bayer mask) (рис.1.42). Для здо-

Цифрові фотоапарати і відеокамери

Цифрові фотоапарати і відеокамери аналогічні за принципами дії (система лінз, що проектує світло, яке потрапляє до об'єктива, на невеликий плоский майданчик) традиційним аналоговим фото- і відеокамерам відповідно.

Основна відмінність полягає в тому, що в аналогових пристроях на цьому майданчику знаходиться кадр світлочутливої плівки, а в цифрових пристроях - світлочутлива матриця ПЗС (англ. CCD) або КМОП (англ. CMOS) сенсорів. У фотоапаратах із сенсорами ПЗС формування кольорового зображення будується в спеціальному пристрої після перетворення аналогового сигналу в цифровий сигнал. У фотоапаратах із сенсорами архітектури КМОП змішання кольорів може вироблятися безпосередньо в самому сенсорі. Але й у тому, і в іншому випадку кольорове зображення будується по спеціальних математичних алгоритмах методом інтерполяції - тобто з урахуванням яскравості сусідніх елементів кожного з базових кольорів. Оскільки ці сенсори чутливі лише до яскравості, то для отримання кольорового зображення застосовують світлофільтри. Існує два підходи.

Макросвітлофільтри. Перший варіант. Для здобуття одного зображення інформація знімається з матриці сенсора 3 рази підряд, при цьому кожного разу використовується світлофільтр для одного з базисних кольорів RGB. Цей процес порівняно повільний і складний, тому така технологія застосовується лише в high-end цифрових фотокамерах при зйомці нерухомих об'єктів. Другий варіант. За допомогою призми світловий потік, що поступає, розкладається на три, що відповідають базисним кольорам RGB. Кожен з цих потоків прямує на свою ПЗС-матрицю. Тому ця технологія отримала назву 3ccd. Вона використовується у відеокамерах верхнього цінового діапазону. Для фотографії вона доки дуже дорога.

Мікросвітлофільтри. Кожен елемент ПЗС матриці має свій світлофільтр, відповідний одному з базисних кольорів RGB. Зазвичай у квадраті 2×2 розташовуються 2 зелених, 1 синій і 1 червоний елемент, так звана маска Байєра (англ. Bayer mask) (рис.1.42). Для здо-

буття звичайного зображення використовуються різні алгоритми інтерполяції. Це найпоширеніша технологія як для фото-, так і для відеокамер.

Відеокамери нижнього цінового діапазону знімають відео з черезрядковою розгорткою, тобто кадр складається з двох послідовно знятих напівкадрів, що складаються лише з парних і непарних рядків зображення відповідно. При швидкому переміщенні об'єктів, що знімаються, або самої камери через помітну різницю в напівкадрах виникають неприємні артефакти. Для вирішення цієї проблеми при подальшій обробці знятого відео здійснюється deinterlacing.

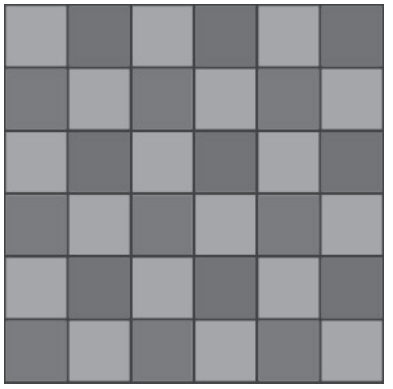


Рис. 1.42. Маска Байєра

Поняття роздільної здатності для таких пристроїв зазвичай не застосовується. Часто вживається поняття мегапіксель (Мп), що дорівнює загальній кількості пікселів на матриці сенсора, поділеному на 10242. Дозвіл, як розмір растра, варіюється від 640×480 (0,29 Мп) до 7216×5412 (39 Мп) для фотокамер і від 320×240 (0,07 Мп) до 7680×4320 (33 Мп) - для відеокамер. Глибина кольору у більшості фото- і відеокамер - стандартні 24 біта/піксель. У професійних моделей вона більша - до 48 біт/піксель.

Пристрої виводу

Монітори (класифікація, принцип дії, основні характеристики). Однією з найбільш важливих складових частин персонального комп'ютера є його відеопідсистема, що складається з монітора і відеоадаптера (зазвичай розміщуваного на системній платі). Монітор призначений для відображення на екрані текстової і графічної

буття звичайного зображення використовуються різні алгоритми інтерполяції. Це найпоширеніша технологія як для фото-, так і для відеокамер.

Відеокамери нижнього цінового діапазону знімають відео з черезрядковою розгорткою, тобто кадр складається з двох послідовно знятих напівкадрів, що складаються лише з парних і непарних рядків зображення відповідно. При швидкому переміщенні об'єктів, що знімаються, або самої камери через помітну різницю в напівкадрах виникають неприємні артефакти. Для вирішення цієї проблеми при подальшій обробці знятого відео здійснюється deinterlacing.

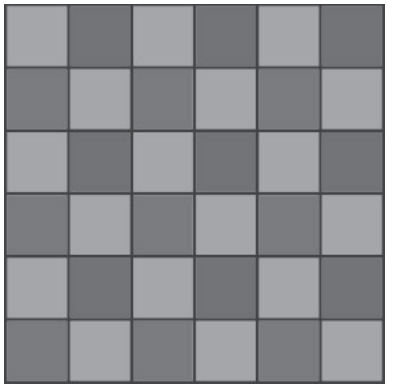


Рис. 1.42. Маска Байєра

Поняття роздільної здатності для таких пристроїв зазвичай не застосовується. Часто вживається поняття мегапіксель (Мп), що дорівнює загальній кількості пікселів на матриці сенсора, поділеному на 10242. Дозвіл, як розмір растра, варіюється від 640×480 (0,29 Мп) до 7216×5412 (39 Мп) для фотокамер і від 320×240 (0,07 Мп) до 7680×4320 (33 Мп) - для відеокамер. Глибина кольору у більшості фото- і відеокамер - стандартні 24 біта/піксель. У професійних моделей вона більша - до 48 біт/піксель.

Пристрої виводу

Монітори (класифікація, принцип дії, основні характеристики). Однією з найбільш важливих складових частин персонального комп'ютера є його відеопідсистема, що складається з монітора і відеоадаптера (зазвичай розміщуваного на системній платі). Монітор призначений для відображення на екрані текстової і графічної

інформації, що візуально сприймається користувачем персонального комп'ютера. В даний час існує велика різноманітність типів моніторів.

За режимом відображення монітори діляться на растрові дисплеї та векторні дисплеї.

У векторних дисплеях з регенерацією зображення на базі електронно-променевої трубки (ЕПТ) використовується люмінофор з дуже коротким часом післясвічення. Такі дисплеї часто називають дисплеями з довільним скануванням. Через те, що час післясвічення люмінофора малий, зображення на ЕПТ за секунду повинно багато разів перемальовуватися або регенеруватися. Мінімальна швидкість регенерації повинна складати принаймні 30 (1/с), а переважно 40–50 (1/с). Менша швидкість регенерації призводить до мерехтіння зображення.

Окрім ЕПТ, для векторного дисплея необхідний дисплейний буфер і дисплейний контролер. Дисплейний буфер – безперервна ділянка пам'яті, що містить всю інформацію, необхідну для виведення зображення на ЕПТ. Функція дисплейного контролера полягає в тому, щоб циклічно обробляти цю інформацію із швидкістю регенерації. Складність малюнка обмежується двома чинниками – розміром дисплейного буфера і швидкістю контролера.

Растровий пристрій можна розглядати як матрицю дискретних точок, кожна з яких може підсвічуватися. Неможливо, за винятком спеціальних випадків, безпосередньо намалювати відрізок прямої з однієї точки, що адресується, або піксела в матриці в іншу точку, що адресується. Відрізок можна лише апроксимувати послідовностями точок (пікселів), що близько лежать до реальної траєкторії відрізка.

Відрізок прямої з точок вийде лише в разі горизонтальних, вертикальних або розташованих під кутом 45 градусів відрізків. Всі інші відрізки виглядатимуть як послідовності сходинок. Це явище називається сходовим ефектом.

Найчастіше для графічних пристроїв з растровою ЕПТ використовується буфер кадру. Буфером кадру є велика безперервна ділянка пам'яті комп'ютера. Для кожної точки або піксела в растрі відводиться як мінімум один біт пам'яті. Ця пам'ять називається бітовою площиною. Для квадратного растра розміром 512x512 потрібний

інформації, що візуально сприймається користувачем персонального комп'ютера. В даний час існує велика різноманітність типів моніторів.

За режимом відображення монітори діляться на растрові дисплеї та векторні дисплеї.

У векторних дисплеях з регенерацією зображення на базі електронно-променевої трубки (ЕПТ) використовується люмінофор з дуже коротким часом післясвічення. Такі дисплеї часто називають дисплеями з довільним скануванням. Через те, що час післясвічення люмінофора малий, зображення на ЕПТ за секунду повинно багато разів перемальовуватися або регенеруватися. Мінімальна швидкість регенерації повинна складати принаймні 30 (1/с), а переважно 40–50 (1/с). Менша швидкість регенерації призводить до мерехтіння зображення.

Окрім ЕПТ, для векторного дисплея необхідний дисплейний буфер і дисплейний контролер. Дисплейний буфер – безперервна ділянка пам'яті, що містить всю інформацію, необхідну для виведення зображення на ЕПТ. Функція дисплейного контролера полягає в тому, щоб циклічно обробляти цю інформацію із швидкістю регенерації. Складність малюнка обмежується двома чинниками – розміром дисплейного буфера і швидкістю контролера.

Растровий пристрій можна розглядати як матрицю дискретних точок, кожна з яких може підсвічуватися. Неможливо, за винятком спеціальних випадків, безпосередньо намалювати відрізок прямої з однієї точки, що адресується, або піксела в матриці в іншу точку, що адресується. Відрізок можна лише апроксимувати послідовностями точок (пікселів), що близько лежать до реальної траєкторії відрізка.

Відрізок прямої з точок вийде лише в разі горизонтальних, вертикальних або розташованих під кутом 45 градусів відрізків. Всі інші відрізки виглядатимуть як послідовності сходинок. Це явище називається сходовим ефектом.

Найчастіше для графічних пристроїв з растровою ЕПТ використовується буфер кадру. Буфером кадру є велика безперервна ділянка пам'яті комп'ютера. Для кожної точки або піксела в растрі відводиться як мінімум один біт пам'яті. Ця пам'ять називається бітовою площиною. Для квадратного растра розміром 512x512 потрібний

218, або 262144 біта пам'яті в одній бітовій площині. Через те, що біт пам'яті має лише два стани (двійкове 0 або 1), маючи одну бітову площину, можна отримати лише чорно-біле зображення.

Бітова площина є цифровим пристроєм, тоді як растрова ЕПТ – аналоговий пристрій. Тому під час читання інформації з буфера кадру і її виводу на графічний пристрій з растровою ЕПТ повинне відбуватися перетворення з цифрової форми в аналоговий сигнал. Таке перетворення виконує цифро-аналоговий перетворювач (ЦАП).

За типом екрану монітори діляться на дисплеї на основі ЕПТ, рідкокристалічні (РК), плазмові.

Дисплеї на основі електронно-променевої трубки. Щоб зрозуміти принципи роботи растрових дисплеїв і векторних дисплеїв з регенерацією, потрібно мати уявлення про конструкцію ЕПТ і методи створення відеозображення. Катод (негативно заряджений електрод) нагрівають до тих пір, поки збуджені електрони не створять хмари, що розширюються (електрони відштовхуються один від одного, оскільки мають однаковий заряд). Ці електрони притягуються до сильно зарядженого позитивного анода. На внутрішню сторону розширеного кінця ЕПТ нанесений люмінофор. Хмара електронів за допомогою лінз фокусується у вузький, строго паралельний пучок, і промінь дає яскраву пляму в центрі ЕПТ. Промінь відхиляється або позиціює вліво або вправо від центру і (або) вище або нижче за центр за допомогою підсилювачів горизонтального і вертикального відхилення. Саме в даний момент виявляється відмінність векторних і растрових дисплеїв. У векторному дисплеї електронний промінь може бути відхилений безпосередньо з будь-якої довільної позиції в будь-яку іншу довільну позицію на екрані ЕПТ (аноді). Оскільки люмінофорне покриття нанесене на екран ЕПТ суцільним шаром, у результаті виходить майже ідеальна пряма. На відміну від цього в растровому дисплеї промінь може відхилятися лише в строго певні позиції на екрані, створюючи своєрідну мозаїку. Ця мозаїка складає відеозображення. Люмінофорне покриття на екрані растрової ЕПТ теж не безперервним, а є безліччю тісно розташованих найдрібніших точок, куди позиціюється промінь, утворюючи мозаїку.

218, або 262144 біта пам'яті в одній бітовій площині. Через те, що біт пам'яті має лише два стани (двійкове 0 або 1), маючи одну бітову площину, можна отримати лише чорно-біле зображення.

Бітова площина є цифровим пристроєм, тоді як растрова ЕПТ – аналоговий пристрій. Тому під час читання інформації з буфера кадру і її виводу на графічний пристрій з растровою ЕПТ повинне відбуватися перетворення з цифрової форми в аналоговий сигнал. Таке перетворення виконує цифро-аналоговий перетворювач (ЦАП).

За типом екрану монітори діляться на дисплеї на основі ЕПТ, рідкокристалічні (РК), плазмові.

Дисплеї на основі електронно-променевої трубки. Щоб зрозуміти принципи роботи растрових дисплеїв і векторних дисплеїв з регенерацією, потрібно мати уявлення про конструкцію ЕПТ і методи створення відеозображення. Катод (негативно заряджений електрод) нагрівають до тих пір, поки збуджені електрони не створять хмари, що розширюються (електрони відштовхуються один від одного, оскільки мають однаковий заряд). Ці електрони притягуються до сильно зарядженого позитивного анода. На внутрішню сторону розширеного кінця ЕПТ нанесений люмінофор. Хмара електронів за допомогою лінз фокусується у вузький, строго паралельний пучок, і промінь дає яскраву пляму в центрі ЕПТ. Промінь відхиляється або позиціює вліво або вправо від центру і (або) вище або нижче за центр за допомогою підсилювачів горизонтального і вертикального відхилення. Саме в даний момент виявляється відмінність векторних і растрових дисплеїв. У векторному дисплеї електронний промінь може бути відхилений безпосередньо з будь-якої довільної позиції в будь-яку іншу довільну позицію на екрані ЕПТ (аноді). Оскільки люмінофорне покриття нанесене на екран ЕПТ суцільним шаром, у результаті виходить майже ідеальна пряма. На відміну від цього в растровому дисплеї промінь може відхилятися лише в строго певні позиції на екрані, створюючи своєрідну мозаїку. Ця мозаїка складає відеозображення. Люмінофорне покриття на екрані растрової ЕПТ теж не безперервним, а є безліччю тісно розташованих найдрібніших точок, куди позиціюється промінь, утворюючи мозаїку.

Екран **рідкокристалічного дисплея** (РКД) складається з двох скляних пластин, між якими знаходиться маса, що містить рідкі кристали, які змінюють свої оптичні властивості залежно від електричного заряду, що додається. Рідкі кристали самі не світяться, тому РКД потребують підсвічування.

Основною перевагою РКД є їх габарити (екран плоский). До недоліків можна віднести недостатню швидкодію при зміні зображення на екрані, що особливо помітно при переміщенні курсору миші, а також залежність різкості і яскравості зображення від точки зору.

Рідкокристалічні дисплеї мають безперечні переваги перед конкуруючими пристроями відображення:

1. Розміри. РКД відрізняються малою глибиною і невеликою масою і тому їх зручніше переміщати і встановлювати, чим ЕПТ-монітори, в яких розмір у глибину приблизно дорівнює ширині.

2. Енергоспоживання. РКД споживає меншу потужність, чим ЕПТ-монітор з порівнянними характеристиками.

3. Зручність для користувача. У ЕПТ електронні промені при розгортці рухаються по екрану, оновлюючи зображення. Хоча в більшості випадків можна встановити таку частоту регенерації (число оновлень екрана електронними променями в секунду), що зображення виглядає стабільним, деякі користувачі все ж сприймають мерехтіння, здатне викликати швидке стомлення очей і головний біль.

На екрані РКД кожен піксел або включений, або вимкнений, так що мерехтіння відсутнє (рис. 1.43). Крім того, для ЕПТ-моніторів характерна наявність у невеликих кількостях електромагнітного випромінювання; у РК моніторах такого випромінювання немає. Недолік – висока ціна.

Ще одне досягнення, завдяки якому може статися зниження цін незабаром, – удосконалення технології панелей на кристалах (dual supertwist nematic, DSTN). Dstn-дисплєї завжди були дешевші, ніж РКпристрої на тонкоплівкових транзисторах, але дещо поступалися їм за якістю: Dstn-дисплєї не забезпечують такої контрастності і чіткості, як матриці TFT, а їх повільна реакція призводить до мерехтіння і появи паразитних (повторних) зображень на екрані, особливо при відображенні рухомих об’єктів. Проте фірма Sharp, найбільший

Екран **рідкокристалічного дисплея** (РКД) складається з двох скляних пластин, між якими знаходиться маса, що містить рідкі кристали, які змінюють свої оптичні властивості залежно від електричного заряду, що додається. Рідкі кристали самі не світяться, тому РКД потребують підсвічування.

Основною перевагою РКД є їх габарити (екран плоский). До недоліків можна віднести недостатню швидкодію при зміні зображення на екрані, що особливо помітно при переміщенні курсору миші, а також залежність різкості і яскравості зображення від точки зору.

Рідкокристалічні дисплеї мають безперечні переваги перед конкуруючими пристроями відображення:

1. Розміри. РКД відрізняються малою глибиною і невеликою масою і тому їх зручніше переміщати і встановлювати, чим ЕПТ-монітори, в яких розмір у глибину приблизно дорівнює ширині.

2. Енергоспоживання. РКД споживає меншу потужність, чим ЕПТ-монітор з порівнянними характеристиками.

3. Зручність для користувача. У ЕПТ електронні промені при розгортці рухаються по екрану, оновлюючи зображення. Хоча в більшості випадків можна встановити таку частоту регенерації (число оновлень екрана електронними променями в секунду), що зображення виглядає стабільним, деякі користувачі все ж сприймають мерехтіння, здатне викликати швидке стомлення очей і головний біль.

На екрані РКД кожен піксел або включений, або вимкнений, так що мерехтіння відсутнє (рис. 1.43). Крім того, для ЕПТ-моніторів характерна наявність у невеликих кількостях електромагнітного випромінювання; у РК моніторах такого випромінювання немає. Недолік – висока ціна.

Ще одне досягнення, завдяки якому може статися зниження цін незабаром, – удосконалення технології панелей на кристалах (dual supertwist nematic, DSTN). Dstn-дисплєї завжди були дешевші, ніж РКпристрої на тонкоплівкових транзисторах, але дещо поступалися їм за якістю: Dstn-дисплєї не забезпечують такої контрастності і чіткості, як матриці TFT, а їх повільна реакція призводить до мерехтіння і появи паразитних (повторних) зображень на екрані, особливо при відображенні рухомих об’єктів. Проте фірма Sharp, найбільший

постачальник Dstn-панелей, нещодавно провела презентацію панелі, в якій використовується розроблена нею технологія HCA (високошвидкісна адресація).

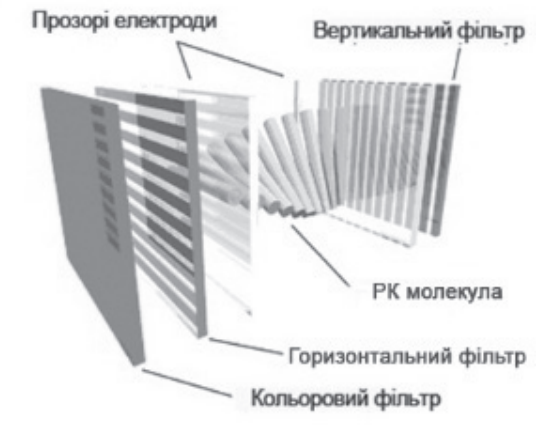


Рис. 1.43. Субпіксел кольорового РК дисплея

HCA-панелі забезпечують таку ж контрастність зображення, як TFT–матриці, і майже не поступаються їм за швидкістю реакції при відтворенні відео. Фірма Arithmos розробила процесор візуалізації для DSTN-панелей, який дозволяє ще більш поліпшити якість зображення. Таким чином, для користувачів, обмежених у засобах, DSTN-дисплей може виявитися хорошим компромісним рішенням.

У РКД кут огляду не лише малий, але і асиметричний: зазвичай він складає 45 градусів по горизонталі і +15...–30 по вертикалі. Випромінюючі дисплеї, такі як електролюмінесцентні, плазмові і на базі ЕПТ, як правило, мають конус огляду від 80 до 90 градусів по обох осях. Хоча останнім часом на ринку з'явилися моделі РКД із збільшеним кутом огляду 50–60 градусів. Представник Hitachi Тім Паттон (Tim Patton) вважає, що в традиційних РКД спостерігається залежність контрастності і кольору зображення від кута зору. Ця проблема загострювалася у міру збільшення розмірів РКД і набуття ними здатності відтворювати більше кольорів.

постачальник Dstn-панелей, нещодавно провела презентацію панелі, в якій використовується розроблена нею технологія HCA (високошвидкісна адресація).

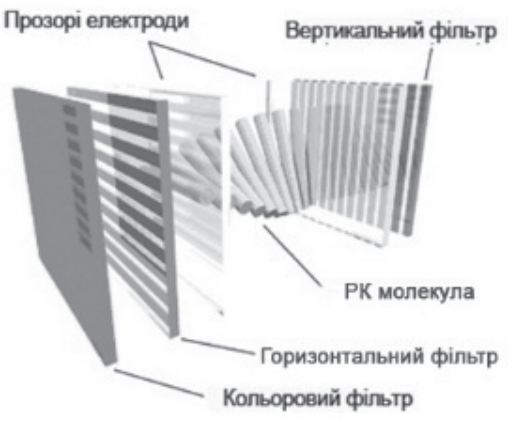


Рис. 1.43. Субпіксел кольорового РК дисплея

HCA-панелі забезпечують таку ж контрастність зображення, як TFT–матриці, і майже не поступаються їм за швидкістю реакції при відтворенні відео. Фірма Arithmos розробила процесор візуалізації для DSTN-панелей, який дозволяє ще більш поліпшити якість зображення. Таким чином, для користувачів, обмежених у засобах, DSTN-дисплей може виявитися хорошим компромісним рішенням.

У РКД кут огляду не лише малий, але і асиметричний: зазвичай він складає 45 градусів по горизонталі і +15...–30 по вертикалі. Випромінюючі дисплеї, такі як електролюмінесцентні, плазмові і на базі ЕПТ, як правило, мають конус огляду від 80 до 90 градусів по обох осях. Хоча останнім часом на ринку з'явилися моделі РКД із збільшеним кутом огляду 50–60 градусів. Представник Hitachi Тім Паттон (Tim Patton) вважає, що в традиційних РКД спостерігається залежність контрастності і кольору зображення від кута зору. Ця проблема загострювалася у міру збільшення розмірів РКД і набуття ними здатності відтворювати більше кольорів.

Hitachi при створенні свого нового дисплея SUPER TFT скористалася іншою технологією – IPS. Як відомо, у звичайних РКД молекули рідкого кристала міняють свою орієнтацію з горизонтальної на вертикальну під впливом електричного поля, а адресуючі електроди поміщаються на дві розташовані одна проти одної скляні підкладки.

У IPS (in–plane switching) – дисплеях, навпаки, відбувається чергування двох кутів у горизонтальній площині, причому обидва електроди знаходяться на одній з підкладок. У результаті кут огляду як по горизонтальній, так і по вертикальній осі досягає 70 градусів.

Плазмові дисплеї. Газоплазмові монітори складаються з двох пластин, між якими знаходиться газова суміш, що світиться під впливом електричних імпульсів. Такі монітори не мають недоліків, властивих РКД, проте їх не можна використовувати в переносних комп'ютерах з акумуляторним і батарейним живленням, оскільки вони споживають великий струм.

Розмір по діагоналі (відстань від лівого нижнього до правого верхнього кута екрану) приводиться в дюймах. Найбільш поширені монітори з діагоналлю 14". Проте працювати з монітором з діагоналлю 15" набагато зручніше, а для роботи з графічними пакетами, видавничими системами і системами автоматизованого проектування необхідні монітори з діагоналлю не менше 17".

Під час оцінки якості моніторів використовуються такі параметри як тіньова маска, дозвіл, потужність тощо.

Тіньова маска екрана. Одиницею виміру є відстань між отворами маски в міліметрах. Чим менша ця відстань і чим більше отворів, тим вища якість зображення. Цей параметр часто ототожнюють із зерном екрана монітора, проте це справедливо не у всіх випадках.

Дозвіл. Вимірюється в пікселах (точках), що поміщаються по горизонталі і вертикалі видимої частини екрана. В даний час найбільш поширені монітори з розширенням не менше 1024×768 пікселів.

Кінескоп. Найбільш поширені такі типи кінескопів: Black Trinitron, Black Matrix і Black Planar. Дані кінескопи дуже контрастні, дають відмінне зображення, проте їх люмінофор чутливий до світла, що може скоротити термін служби монітора. До того ж під час роботи з контрастним монітором швидше втомлюються очі.

Hitachi при створенні свого нового дисплея SUPER TFT скористалася іншою технологією – IPS. Як відомо, у звичайних РКД молекули рідкого кристала міняють свою орієнтацію з горизонтальної на вертикальну під впливом електричного поля, а адресуючі електроди поміщаються на дві розташовані одна проти одної скляні підкладки.

У IPS (in–plane switching) – дисплеях, навпаки, відбувається чергування двох кутів у горизонтальній площині, причому обидва електроди знаходяться на одній з підкладок. У результаті кут огляду як по горизонтальній, так і по вертикальній осі досягає 70 градусів.

Плазмові дисплеї. Газоплазмові монітори складаються з двох пластин, між якими знаходиться газова суміш, що світиться під впливом електричних імпульсів. Такі монітори не мають недоліків, властивих РКД, проте їх не можна використовувати в переносних комп'ютерах з акумуляторним і батарейним живленням, оскільки вони споживають великий струм.

Розмір по діагоналі (відстань від лівого нижнього до правого верхнього кута екрану) приводиться в дюймах. Найбільш поширені монітори з діагоналлю 14". Проте працювати з монітором з діагоналлю 15" набагато зручніше, а для роботи з графічними пакетами, видавничими системами і системами автоматизованого проектування необхідні монітори з діагоналлю не менше 17".

Під час оцінки якості моніторів використовуються такі параметри як тіньова маска, дозвіл, потужність тощо.

Тіньова маска екрана. Одиницею виміру є відстань між отворами маски в міліметрах. Чим менша ця відстань і чим більше отворів, тим вища якість зображення. Цей параметр часто ототожнюють із зерном екрана монітора, проте це справедливо не у всіх випадках.

Дозвіл. Вимірюється в пікселах (точках), що поміщаються по горизонталі і вертикалі видимої частини екрана. В даний час найбільш поширені монітори з розширенням не менше 1024×768 пікселів.

Кінескоп. Найбільш поширені такі типи кінескопів: Black Trinitron, Black Matrix і Black Planar. Дані кінескопи дуже контрастні, дають відмінне зображення, проте їх люмінофор чутливий до світла, що може скоротити термін служби монітора. До того ж під час роботи з контрастним монітором швидше втомлюються очі.

Споживана потужність. У моніторів з діагоналлю 14" споживана потужність не повинна перевищувати 60 Вт, інакше підвищується вірогідність теплового перегріву монітора, що скорочує термін його служби. В крупніших моніторів споживана потужність відповідно вища.

Покриття антивідблиску. Для дешевих моніторів використовують піскоструминну обробку поверхні екрана. При цьому якість зображення погіршується. У дорогих моніторах на поверхню екрана наноситься спеціальна хімічна речовина, що має властивості антивідблисків.

Захисні властивості монітора. В даний час поширені монітори з низьким рівнем випромінювання (LR-монітори).

За кольором монітори поділяються на кольорові та монохромні.

Частота кадрів (звичайно від 50 до 100 Гц). Всі сучасні аналогові монітори умовно можна розділити на такі типи:

– з фіксованою частотою розгортки;
– з декількома фіксованими частотами;
– багаточастотні (мультичастотні).

Мультичастотні монітори мають здатність налаштовуватися на довільні значення частот синхронізації з деякого заданого діапазону, наприклад, 30–64 кГц для рядкової і 50–100 Гц для кадрової розгортки. Розробниками моніторів даного типу є фірма NEC. У назві таких моніторів присутнє слово Multisync. Ці монітори відносяться до найбільш поширеного типу моніторів з електронно-променевою трубкою.

Відеодіапазон (звичайно від 65 до 200 Мгц). Відеосигнал:

– цифровий;
– аналоговий.

Під цифровими моніторами розуміються пристрої відображення зорової інформації на основі електронно-променевої трубки, керованої цифровими схемами. До цифрових відносяться монохромні монітори, забезпечені відеоадаптерами стандартів MDA і Hercules, кольорові Rgb–монітори, призначені для підключення до відеоадаптера стандарту EGA. Монохромні монітори здатні відображувати на екрані лише темні і світлі точки, інколи точки можуть розрізнятися

Споживана потужність. У моніторів з діагоналлю 14" споживана потужність не повинна перевищувати 60 Вт, інакше підвищується вірогідність теплового перегріву монітора, що скорочує термін його служби. В крупніших моніторів споживана потужність відповідно вища.

Покриття антивідблиску. Для дешевих моніторів використовують піскоструминну обробку поверхні екрана. При цьому якість зображення погіршується. У дорогих моніторах на поверхню екрана наноситься спеціальна хімічна речовина, що має властивості антивідблисків.

Захисні властивості монітора. В даний час поширені монітори з низьким рівнем випромінювання (LR-монітори).

За кольором монітори поділяються на кольорові та монохромні.

Частота кадрів (звичайно від 50 до 100 Гц). Всі сучасні аналогові монітори умовно можна розділити на такі типи:

– з фіксованою частотою розгортки;
– з декількома фіксованими частотами;
– багаточастотні (мультичастотні).

Мультичастотні монітори мають здатність налаштовуватися на довільні значення частот синхронізації з деякого заданого діапазону, наприклад, 30–64 кГц для рядкової і 50–100 Гц для кадрової розгортки. Розробниками моніторів даного типу є фірма NEC. У назві таких моніторів присутнє слово Multisync. Ці монітори відносяться до найбільш поширеного типу моніторів з електронно-променевою трубкою.

Відеодіапазон (звичайно від 65 до 200 Мгц). Відеосигнал:

– цифровий;
– аналоговий.

Під цифровими моніторами розуміються пристрої відображення зорової інформації на основі електронно-променевої трубки, керованої цифровими схемами. До цифрових відносяться монохромні монітори, забезпечені відеоадаптерами стандартів MDA і Hercules, кольорові Rgb–монітори, призначені для підключення до відеоадаптера стандарту EGA. Монохромні монітори здатні відображувати на екрані лише темні і світлі точки, інколи точки можуть розрізнятися

інтенсивністю. Hercules–монітори мають дозвіл до 728×348 пікселів, невеликі габарити і вагу. Блок розгортки монітора отримує синхроімпульси від відповідного відеоадаптера. Rgb–монітори здатні відображувати 16 кольорів, проте дозвіл екрана у них менший, ніж в Hercules–моніторов.

Електронно-променева трубка моніторів даного типу управляється аналоговими сигналами що поступають від відеоадаптера. Принцип роботи електронно-променевої трубки монітора такий же, як у телевізійної трубки. Аналогові монітори здатні підтримувати дозвіл стандарту VGA (640×480) пікселів і вище.

Архітектура графічної підсистеми ПК

Які бувають засоби введення і виведення графічної інформації, ми розглянули. Для повноти картини нам необхідно ще відповісти на питання: яким чином графічна інформація виводиться, наприклад, на екран монітора? За виведення графічної інформації на дисплей у ПК відповідає спеціальний набір мікросхем, що зазвичай поміщається на окрему плату, яка називається відеоплатою або відеокартою (рис. 1.44).

Основним завданням відеокарти є перетворення образу екрана, що знаходиться в пам'яті, так званого кадрового буфера (англ. frame buffer), в набір сигналів, зрозумілих дисплею. Для підключення дисплеїв до комп'ютерів використовуються стандарти, що встановлюють логічні та фізичні параметри з'єднання. В даний час два найпоширеніших з них - це аналоговий VGA і цифровий DVI. Перший використовується для підключення як аналогових по суті дисплеїв на ЕПТ, так і цифрових РК дисплеїв (аналого-цифрове перетворення в цьому випадку відбувається в самому дисплеї), другий - виключно для РК дисплеїв. На відеокарті присутня відеопам'ять, характерною особливістю якої є те, що вона двопортова – приєднана як до шини, по якій передаються дані від центрального процесора, так і до мікросхеми, що відповідає за вивід на дисплей, і вони можуть одночасно звертатися до неї. У дешевих пристроях як відеопам'ять використовується частина основної пам'яті, що значно уповільнює обробку даних на відеокарті.

інтенсивністю. Hercules–монітори мають дозвіл до 728×348 пікселів, невеликі габарити і вагу. Блок розгортки монітора отримує синхроімпульси від відповідного відеоадаптера. Rgb–монітори здатні відображувати 16 кольорів, проте дозвіл екрана у них менший, ніж в Hercules–моніторов.

Електронно-променева трубка моніторів даного типу управляється аналоговими сигналами що поступають від відеоадаптера. Принцип роботи електронно-променевої трубки монітора такий же, як у телевізійної трубки. Аналогові монітори здатні підтримувати дозвіл стандарту VGA (640×480) пікселів і вище.

Архітектура графічної підсистеми ПК

Які бувають засоби введення і виведення графічної інформації, ми розглянули. Для повноти картини нам необхідно ще відповісти на питання: яким чином графічна інформація виводиться, наприклад, на екран монітора? За виведення графічної інформації на дисплей у ПК відповідає спеціальний набір мікросхем, що зазвичай поміщається на окрему плату, яка називається відеоплатою або відеокартою (рис. 1.44).

Основним завданням відеокарти є перетворення образу екрана, що знаходиться в пам'яті, так званого кадрового буфера (англ. frame buffer), в набір сигналів, зрозумілих дисплею. Для підключення дисплеїв до комп'ютерів використовуються стандарти, що встановлюють логічні та фізичні параметри з'єднання. В даний час два найпоширеніших з них - це аналоговий VGA і цифровий DVI. Перший використовується для підключення як аналогових по суті дисплеїв на ЕПТ, так і цифрових РК дисплеїв (аналого-цифрове перетворення в цьому випадку відбувається в самому дисплеї), другий - виключно для РК дисплеїв. На відеокарті присутня відеопам'ять, характерною особливістю якої є те, що вона двопортова – приєднана як до шини, по якій передаються дані від центрального процесора, так і до мікросхеми, що відповідає за вивід на дисплей, і вони можуть одночасно звертатися до неї. У дешевих пристроях як відеопам'ять використовується частина основної пам'яті, що значно уповільнює обробку даних на відеокарті.

Обсяг пам’яті має бути достатнім для зберігання даних кадрового буфера, а бажано ще і вторинного буфера (англ. back buffer), якщо використовується технологія подвійної буферизації. Річ у тому, що якщо міняти значення пікселів прямо у момент виводу їх на екран, то при високій частоті оновлення можуть виникати артефакти, пов’язані з тим, що на екран виводиться зображення, яке ще не відмалювано до кінця. Щоб цього уникнути, при подвійній буферизації під час виведення зображення з області відеопам’яті, яка призначена кадровим буфером (званим у цьому випадку також первинним буфером (англ. front buffer)), зображення наступного кадру будується у вторинному буфері, а при показі наступного кадру ці області пам’яті міняються ролями. Ця технологія використовується для показу динамічних зображень, таких як ігри. Додаткова відеопам’ять також прискорює обробку графіки, дозволяючи тримати додаткові графічні елементи, які відображаються в кадровому буфері за допомогою бліттінга.

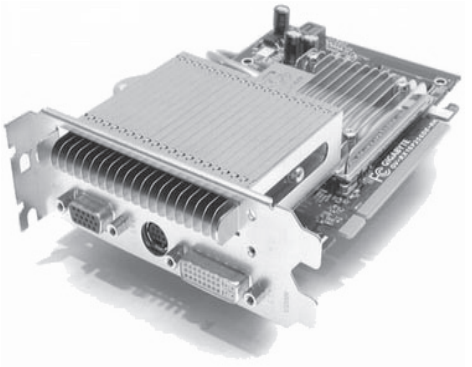


Рис. 1.44. Зовнішній вигляд відеокарти

Доступ до відеопам’яті з боку процесора може бути організований двояко – або відеопам’ять як частина адреси включається в адресний простір процесора, або для копіювання даних між основною і відеопам’яттю контролеру на відеокарті посилається спеціальна команда, і копіювання відбувається за допомогою DMA (від англ. Direct Memory Access - мікросхема, що дозволяє здійснювати передачу даних в/із оперативної пам’яті периферійним пристроям без участі центрального процесора).

Мікросхема постійно скануватиме відеопам’ять, перетворить її у форму, відповідну інтерфейсу дисплея, і формує вихідний сигнал, що передається по кабелю на дисплей (рис. 1.45).

Обсяг пам’яті має бути достатнім для зберігання даних кадрового буфера, а бажано ще і вторинного буфера (англ. back buffer), якщо використовується технологія подвійної буферизації. Річ у тому, що якщо міняти значення пікселів прямо у момент виводу їх на екран, то при високій частоті оновлення можуть виникати артефакти, пов’язані з тим, що на екран виводиться зображення, яке ще не відмалювано до кінця. Щоб цього уникнути, при подвійній буферизації під час виведення зображення з області відеопам’яті, яка призначена кадровим буфером (званим у цьому випадку також первинним буфером (англ. front buffer)), зображення наступного кадру будується у вторинному буфері, а при показі наступного кадру ці області пам’яті міняються ролями. Ця технологія використовується для показу динамічних зображень, таких як ігри. Додаткова відеопам’ять також прискорює обробку графіки, дозволяючи тримати додаткові графічні елементи, які відображаються в кадровому буфері за допомогою бліттінга.

Доступ до відеопам’яті з боку процесора може бути організований двояко – або відеопам’ять як частина адреси включається в адресний простір процесора, або для копіювання даних між основною і відеопам’яттю контролеру на відеокарті посилається спеціальна команда, і копіювання відбувається за допомогою DMA (від англ. Direct Memory Access - мікросхема, що дозволяє здійснювати передачу даних в/із оперативної пам’яті периферійним пристроям без участі центрального процесора).

Мікросхема постійно скануватиме відеопам’ять, перетворить її у форму, відповідну інтерфейсу дисплея, і формує вихідний сигнал, що передається по кабелю на дисплей (рис. 1.45).

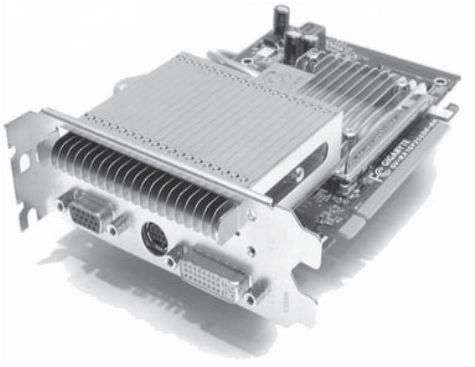


Рис. 1.44. Зовнішній вигляд відеокарти

74

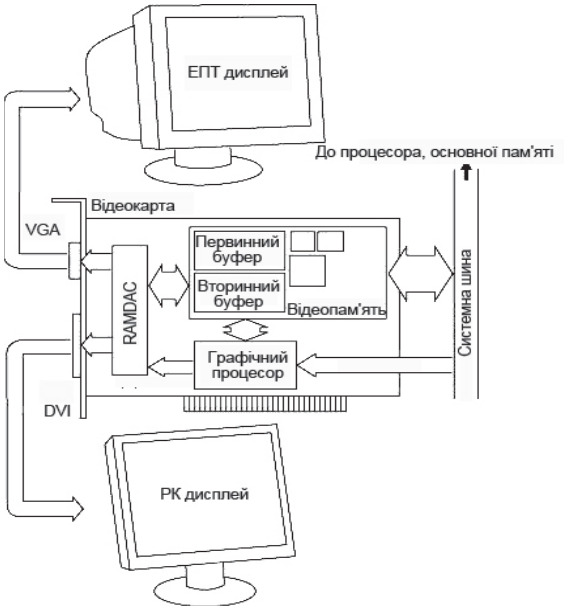


Рис. 1.45. Архітектура відеопідсистеми ПК

Якщо відеоплата оснащена аналоговим виходом, то в неї має бути вбудований цифро-аналоговий перетворювач, що називається RAMDAC - Random Access Memory Digital to Analog Converter. Оскільки інформація про піксели передається послідовно, то RAMDAC повинен мати досить високу тактову частоту, щоб дозволяти виводити зображення високого дозволу з достатньою частотою оновлення. Також на відеоплаті міститься графічний процесор, здатний швидко виконувати основні операції по роботі із зображеннями у відеопам’яті. Деякі сучасні відеокарти дозволяють підключати відразу декілька дисплеїв одночасно.

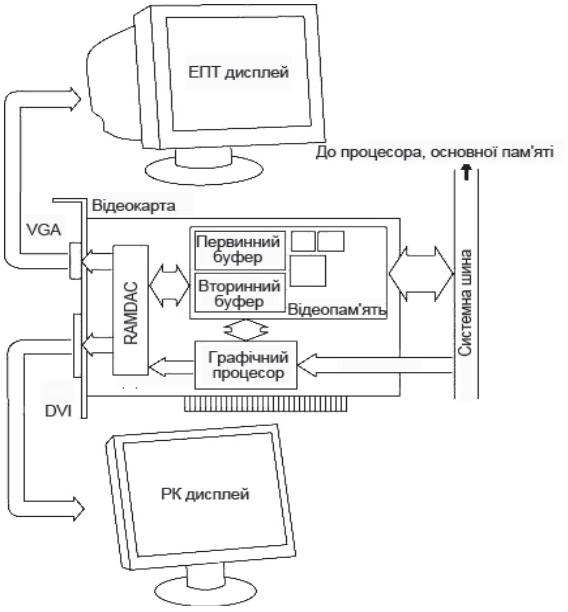


Рис. 1.45. Архітектура відеопідсистеми ПК

Якщо відеоплата оснащена аналоговим виходом, то в неї має бути вбудований цифро-аналоговий перетворювач, що називається RAMDAC - Random Access Memory Digital to Analog Converter. Оскільки інформація про піксели передається послідовно, то RAMDAC повинен мати досить високу тактову частоту, щоб дозволяти виводити зображення високого дозволу з достатньою частотою оновлення. Також на відеоплаті міститься графічний процесор, здатний швидко виконувати основні операції по роботі із зображеннями у відеопам’яті. Деякі сучасні відеокарти дозволяють підключати відразу декілька дисплеїв одночасно.

Контрольні питання

1. Що розуміють під терміном «комп'ютерна графіка»?
2. Які основні області застосування комп'ютерної графіки?
3. Наведіть приклади програмних продуктів інтерактивної комп'ютерної графіки.
4. У чому полягає актуальність вивчення дисципліни «комп'ютерна графіка»?
5. Які види комп'ютерної графіки ви знаєте? Чим вони відрізняються один від одного?
6. Чим обумовлені переваги векторної графіки в порівнянні з растровою?

Тестові завдання

Питання 1. Задачами зображувальної комп'ютерної графіки є:	А)	розпізнавання образів
	Б)	побудова об'єкта та генерація зображення
	В)	поліпшення якості зображення
Питання 2. Значна чутливість до масштабування та великий розмір файла характерно для:	А)	растрової графіки
	Б)	фрактальної графіки
	В)	векторної графіки
Питання 3. Переваги векторної графіки обумовлені тим, що:	А)	зображення являє собою математичне описання
	Б)	зображення будується по рівнянню (або за системою рівнянь), тому нічого, окрім формули, зберігати в пам'яті не потрібно
	В)	відсутня чутливість до масштабування
Питання 4. Колірна модель HSB - це модель:	А)	в якій формується зображення предмета у відбитому світлі
	Б)	значення кольору вибирається як вектор, що виходить з центру окружності
	В)	в якій формується зображення предмета в світлі, що проходить
Питання 5. Якщо Вам необхідно підготувати зображення до друку, то це більш доцільно робити з використанням колірної моделі:	А)	HSB
	Б)	CMYK
	В)	RGB
Питання 6. Дигітайзер - це пристрій:	А)	введення графічної інформації
	Б)	виведення графічної інформації
	В)	обробки графічної інформації

Контрольні питання

1. Що розуміють під терміном «комп'ютерна графіка»?
2. Які основні області застосування комп'ютерної графіки?
3. Наведіть приклади програмних продуктів інтерактивної комп'ютерної графіки.
4. У чому полягає актуальність вивчення дисципліни «комп'ютерна графіка»?
5. Які види комп'ютерної графіки ви знаєте? Чим вони відрізняються один від одного?
6. Чим обумовлені переваги векторної графіки в порівнянні з растровою?

Тестові завдання

Питання 1. Задачами зображувальної комп'ютерної графіки є:	А)	розпізнавання образів
	Б)	побудова об'єкта та генерація зображення
	В)	поліпшення якості зображення
Питання 2. Значна чутливість до масштабування та великий розмір файла характерно для:	А)	растрової графіки
	Б)	фрактальної графіки
	В)	векторної графіки
Питання 3. Переваги векторної графіки обумовлені тим, що:	А)	зображення являє собою математичне описання
	Б)	зображення будується по рівнянню (або за системою рівнянь), тому нічого, окрім формули, зберігати в пам'яті не потрібно
	В)	відсутня чутливість до масштабування
Питання 4. Колірна модель HSB - це модель:	А)	в якій формується зображення предмета у відбитому світлі
	Б)	значення кольору вибирається як вектор, що виходить з центру окружності
	В)	в якій формується зображення предмета в світлі, що проходить
Питання 5. Якщо Вам необхідно підготувати зображення до друку, то це більш доцільно робити з використанням колірної моделі:	А)	HSB
	Б)	CMYK
	В)	RGB
Питання 6. Дигітайзер - це пристрій:	А)	введення графічної інформації
	Б)	виведення графічної інформації
	В)	обробки графічної інформації

Питання 7. Менше енергоспоживання характерно для:

А) рідкокристалічного дисплея
Б) дисплея з електронно-променевою трубкою
В) плазмового дисплея

Питання 8. Наявність електромагнітного випромінювання характерно для:

А) рідкокристалічного дисплея
Б) дисплея з електронно-променевою трубкою
В) плазмового дисплея

Питання 9. Малий кут огляду характерний для:

А) дисплея з електронно-променевою трубкою
Б) плазмового дисплея
В) рідкокристалічного дисплея

Питання 10. Для підключення РК дисплея використовується:

А) VGA-стандарт
Б) DVI-стандарт
В) BGI-стандарт

Питання 11. Найбільший вплив на кількість пам’яті, займаної растровим зображенням, надають:

А) розмір зображення
Б) бітова глибина кольору
В) формат файла, що використовується для зберігання зображень
Г) усі відповіді вірні

Питання 12. Яка з колірних моделей має менший колірний охват:

А) RGB
Б) CMYK
В) HSB
Г) LAB

Питання 7. Менше енергоспоживання характерно для:

А) рідкокристалічного дисплея
Б) дисплея з електронно-променевою трубкою
В) плазмового дисплея

Питання 8. Наявність електромагнітного випромінювання характерно для:

А) рідкокристалічного дисплея
Б) дисплея з електронно-променевою трубкою
В) плазмового дисплея

Питання 9. Малий кут огляду характерний для:

А) дисплея з електронно-променевою трубкою
Б) плазмового дисплея
В) рідкокристалічного дисплея

Питання 10. Для підключення РК дисплея використовується:

А) VGA-стандарт
Б) DVI-стандарт
В) BGI-стандарт

Питання 11. Найбільший вплив на кількість пам’яті, займаної растровим зображенням, надають:

А) розмір зображення
Б) бітова глибина кольору
В) формат файла, що використовується для зберігання зображень
Г) усі відповіді вірні

Питання 12. Яка з колірних моделей має менший колірний охват:

А) RGB
Б) CMYK
В) HSB
Г) LAB

Глава 2
Математичні й алгоритмічні основи двовимірної графіки

2.1. Зображення та перетворення точок

Вивчення математичного апарату, що покладений в основу комп'ютерної графіки, доцільно розпочати з розгляду способів виведення та перетворення точок та ліній. Ці способи поряд з відповідними алгоритмами використовуються для зображення об'єктів та візуалізації графічної інформації. Можливість проведення перетворення точок та ліній є фундаментом комп'ютерної графіки [11, 13]. До нарисованого об'єкта можуть бути застосовані операції масштабування, повороту, зміщення, відображення відповідно до вимог, що висуваються в завданні.

Зображення точок

Точку зобразити на площині можна за допомогою двох координат, які визначаються матрицею $1\times 2\ [x \quad y]$ (рис. 2.1).

У тривимірному просторі використовується матриця роз-міром $1\times 3\ [x \quad y \quad z]$ (рис. 2.2).

Тобто точка може бути задана у вигляді вектора-стовпчика $\begin{bmatrix} x \\ y \end{bmatrix}$ у двовимірному просторі або у вигляді $[x \quad y \quad z]^T$ у тривимірному.

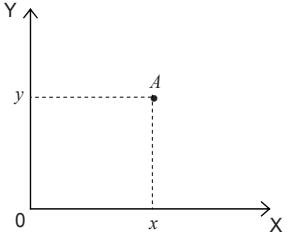


Рис. 2.1. Зображення точки на площині

Глава 2
Математичні й алгоритмічні основи двовимірної графіки

2.1. Зображення та перетворення точок

Вивчення математичного апарату, що покладений в основу комп'ютерної графіки, доцільно розпочати з розгляду способів виведення та перетворення точок та ліній. Ці способи поряд з відповідними алгоритмами використовуються для зображення об'єктів та візуалізації графічної інформації. Можливість проведення перетворення точок та ліній є фундаментом комп'ютерної графіки [11, 13]. До нарисованого об'єкта можуть бути застосовані операції масштабування, повороту, зміщення, відображення відповідно до вимог, що висуваються в завданні.

Зображення точок

Точку зобразити на площині можна за допомогою двох координат, які визначаються матрицею $1\times 2\ [x \quad y]$ (рис. 2.1).

У тривимірному просторі використовується матриця роз-міром $1\times 3\ [x \quad y \quad z]$ (рис. 2.2).

Тобто точка може бути задана у вигляді вектора-стовпчика $\begin{bmatrix} x \\ y \end{bmatrix}$ у двовимірному просторі або у вигляді $[x \quad y \quad z]^T$ у тривимірному.

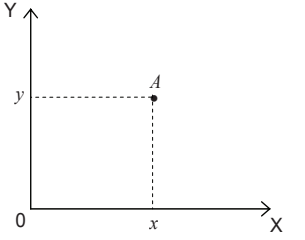


Рис. 2.1. Зображення точки на площині

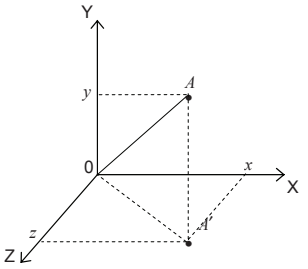


Рис. 2.2.Зображення точки у просторі

Рядок $\begin{bmatrix} x & y \end{bmatrix}$ або $\begin{bmatrix} x \\ y \end{bmatrix}$ стовпчик називають *координат-ним вектором*.

Для формування такого вектора використовується множина точок, кожна з яких визначає координатний вектор у деякій системі вимірювань.

Дана множина зберігається в комп'ютері у вигляді матриці або масиву чисел. Положенням точок можна управляти шляхом маніпулювання відповідною матрицею. Лінії, що з'єднують точки, формують відрізки, криві та картинки. Як елементи матриці можуть виступати різні величини: числа, коефіцієнти систем рівнянь тощо.

Перетворення точок

Розглянемо результат добутку матриці $\begin{bmatrix} x & y \end{bmatrix}$, що містить координати точки A(див рис. 2.1), на матрицю деякого перетворення розміром 2×2:

$$XT = \begin{bmatrix} x & y \end{bmatrix}\begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} (ax+cy)(bx+dy) \end{bmatrix} = \begin{bmatrix} x' & y' \end{bmatrix}. \qquad (2.1)$$

Даний запис означає, що початкові координати точки x та y перетворюються в x' та y', де $x' = ax + cy$, $y' = bx + dy$. Являють інтерес значення x' та y' – координати результуючої, після перетворення точки A. Розглянемо деякі спеціальні випадки.

При $a = d = 1$ та $c = b = 0$ перетворення зводиться до одиничної матриці

$$XT = \begin{bmatrix} x & y \end{bmatrix}\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} x & y \end{bmatrix} = \begin{bmatrix} x' & y' \end{bmatrix}, \qquad (2.2)$$

та координати точки A залишаються незмінними.

У випадку $d = 1$, $c = b = 0$

$$XT = \begin{bmatrix} x & y \end{bmatrix}\begin{bmatrix} a & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} ax & y \end{bmatrix} = \begin{bmatrix} x' & y' \end{bmatrix}, \qquad (2.3)$$

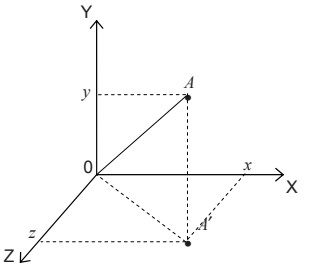


Рис. 2.2.Зображення точки у просторі

Рядок $\begin{bmatrix} x & y \end{bmatrix}$ або $\begin{bmatrix} x \\ y \end{bmatrix}$ стовпчик називають *координат-ним вектором*.

Для формування такого вектора використовується множина точок, кожна з яких визначає координатний вектор у деякій системі вимірювань.

Дана множина зберігається в комп'ютері у вигляді матриці або масиву чисел. Положенням точок можна управляти шляхом маніпулювання відповідною матрицею. Лінії, що з'єднують точки, формують відрізки, криві та картинки. Як елементи матриці можуть виступати різні величини: числа, коефіцієнти систем рівнянь тощо.

Перетворення точок

Розглянемо результат добутку матриці $\begin{bmatrix} x & y \end{bmatrix}$, що містить координати точки A(див рис. 2.1), на матрицю деякого перетворення розміром 2×2:

$$XT = \begin{bmatrix} x & y \end{bmatrix}\begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} (ax+cy)(bx+dy) \end{bmatrix} = \begin{bmatrix} x' & y' \end{bmatrix}. \qquad (2.1)$$

Даний запис означає, що початкові координати точки x та y перетворюються в x' та y', де $x' = ax + cy$, $y' = bx + dy$. Являють інтерес значення x' та y' – координати результуючої, після перетворення точки A. Розглянемо деякі спеціальні випадки.

При $a = d = 1$ та $c = b = 0$ перетворення зводиться до одиничної матриці

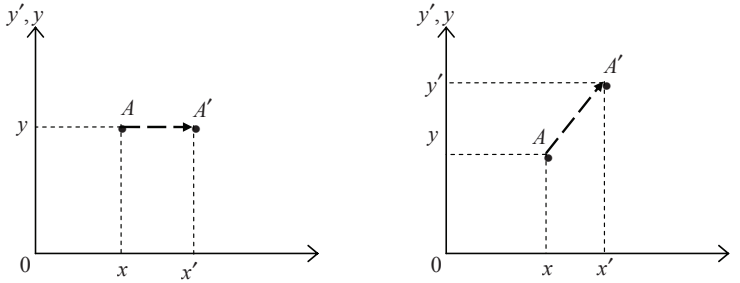
$$XT = \begin{bmatrix} x & y \end{bmatrix}\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} x & y \end{bmatrix} = \begin{bmatrix} x' & y' \end{bmatrix}, \qquad (2.2)$$

та координати точки A залишаються незмінними.

У випадку $d = 1$, $c = b = 0$

$$XT = \begin{bmatrix} x & y \end{bmatrix}\begin{bmatrix} a & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} ax & y \end{bmatrix} = \begin{bmatrix} x' & y' \end{bmatrix}, \qquad (2.3)$$

де $x' = ax$ – результат масштабування координати X. Ефект такого перетворення показаний на рис. 2.3, а.



а б

Рис. 2.3. Результати перетворень координат точки A

У випадку $c = b = 0$

$$XT = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} a & 0 \\ 0 & d \end{bmatrix} = \begin{bmatrix} ax & dy \end{bmatrix} = \begin{bmatrix} x' & y' \end{bmatrix}, \qquad (2.4)$$

перетворення викликає зміну обох координат (рис. 2.4, б).

Якщо $a \neq d$, відбувається масштабування координат по-різному. Якщо a або b від'ємне, відбувається відображення відносно координатної осі.

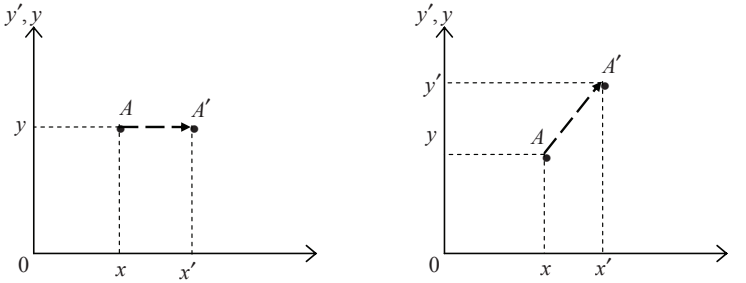
Розглянуті вище випадки враховували, що матриця перетворень діагональна. У випадку, коли $a = d = 1,\ c = 0$

$$XT = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} 1 & b \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} x & (bx + y) \end{bmatrix} = \begin{bmatrix} x' & y' \end{bmatrix}. \qquad (2.5)$$

Координата x точки A' залишається незмінною, тоді як координата y' лінійно залежить від вихідних координат. Дане перетворення називається зсувом.

Аналогічно, у випадку, коли $a = d = 1,\ b = 0$, перетворення приведе до зсуву пропорційно координаті y. Таким чином, недіагональні елементи матриці перетворення створюють ефект зсуву координат вектора точки A.

де $x' = ax$ – результат масштабування координати X. Ефект такого перетворення показаний на рис. 2.3, а.



а б

Рис. 2.3. Результати перетворень координат точки A

У випадку $c = b = 0$

$$XT = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} a & 0 \\ 0 & d \end{bmatrix} = \begin{bmatrix} ax & dy \end{bmatrix} = \begin{bmatrix} x' & y' \end{bmatrix}, \qquad (2.4)$$

перетворення викликає зміну обох координат (рис. 2.4, б).

Якщо $a \neq d$, відбувається масштабування координат по-різному. Якщо a або b від'ємне, відбувається відображення відносно координатної осі.

Розглянуті вище випадки враховували, що матриця перетворень діагональна. У випадку, коли $a = d = 1,\ c = 0$

$$XT = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} 1 & b \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} x & (bx + y) \end{bmatrix} = \begin{bmatrix} x' & y' \end{bmatrix}. \qquad (2.5)$$

Координата x точки A' залишається незмінною, тоді як координата y' лінійно залежить від вихідних координат. Дане перетворення називається зсувом.

Аналогічно, у випадку, коли $a = d = 1,\ b = 0$, перетворення приведе до зсуву пропорційно координаті y. Таким чином, недіагональні елементи матриці перетворення створюють ефект зсуву координат вектора точки A.

Перед тим як закінчити з перетворенням точок, розглянемо дію загального перетворення, заданого виразом (2.1), для випадку, коли початковий вектор лежить у точці початку координат, тобто

$$\begin{bmatrix}0 & 0\end{bmatrix}\begin{bmatrix}a & b\\ c & d\end{bmatrix} = \begin{bmatrix}0 & 0\end{bmatrix} = \begin{bmatrix}x' & y'\end{bmatrix}. \qquad (2.6)$$

З виразу (2.6) видно, що початок координат інваріантний (незалежний) відносно перетворення загального виду. Це обмеження усувається при використанні *однорідних координат*.

2.2. Зображення та перетворення ліній

Пряма лінія може бути задана двома векторами положення, що визначають координати її двох точок. Розташування та напрямок лінії, що з'єднує дві точки, може змінюватися залежно від розташування векторів. На рис. 2.4 зображена пряма лінія, що з'єднує дві точки A та B з координатами $A = \begin{bmatrix}0 & 1\end{bmatrix}$ та $B = \begin{bmatrix}2 & 3\end{bmatrix}$ відповідно.

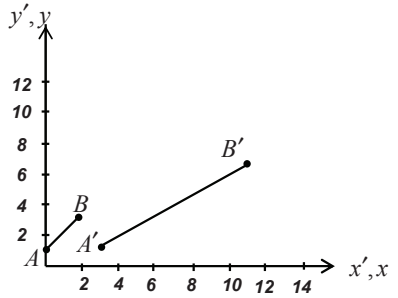


Рис. 2.4.Перетворення ліній

Розглянемо матрицю перетворення

$$T = \begin{bmatrix}1 & 2\\ 3 & 1\end{bmatrix},$$

яка, відповідно до сказано-го, приводить до зсуву зображення. Перетворення векторів A та B за допомогою матриці T дає нове розташування векторів A' та B'.

$$AT = \begin{bmatrix}0 & 1\end{bmatrix}\begin{bmatrix}1 & 2\\ 3 & 1\end{bmatrix} = \begin{bmatrix}3 & 1\end{bmatrix} = A'$$
$$BT = \begin{bmatrix}2 & 3\end{bmatrix}\begin{bmatrix}1 & 2\\ 3 & 1\end{bmatrix} = \begin{bmatrix}11 & 7\end{bmatrix} = B'. \qquad (2.7)$$

Таким чином, результуючі координати для точки $A' - x' = 3$ та $y' = 1$. Відповідно, координати точки $B' - x' = 11$ та $y' = 7$. У більш компактному вигляді відрізок AB може бути поданий матрицею розміром 2×2:

Перед тим як закінчити з перетворенням точок, розглянемо дію загального перетворення, заданого виразом (2.1), для випадку, коли початковий вектор лежить у точці початку координат, тобто

$$\begin{bmatrix}0 & 0\end{bmatrix}\begin{bmatrix}a & b\\ c & d\end{bmatrix} = \begin{bmatrix}0 & 0\end{bmatrix} = \begin{bmatrix}x' & y'\end{bmatrix}. \qquad (2.6)$$

З виразу (2.6) видно, що початок координат інваріантний (незалежний) відносно перетворення загального виду. Це обмеження усувається при використанні *однорідних координат*.

2.2. Зображення та перетворення ліній

Пряма лінія може бути задана двома векторами положення, що визначають координати її двох точок. Розташування та напрямок лінії, що з'єднує дві точки, може змінюватися залежно від розташування векторів. На рис. 2.4 зображена пряма лінія, що з'єднує дві точки A та B з координатами $A = \begin{bmatrix}0 & 1\end{bmatrix}$ та $B = \begin{bmatrix}2 & 3\end{bmatrix}$ відповідно.

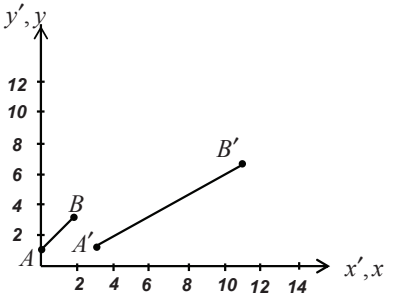


Рис. 2.4.Перетворення ліній

Розглянемо матрицю перетворення

$$T = \begin{bmatrix}1 & 2\\ 3 & 1\end{bmatrix},$$

яка, відповідно до сказано-го, приводить до зсуву зображення. Перетворення векторів A та B за допомогою матриці T дає нове розташування векторів A' та B'.

$$AT = \begin{bmatrix}0 & 1\end{bmatrix}\begin{bmatrix}1 & 2\\ 3 & 1\end{bmatrix} = \begin{bmatrix}3 & 1\end{bmatrix} = A'$$
$$BT = \begin{bmatrix}2 & 3\end{bmatrix}\begin{bmatrix}1 & 2\\ 3 & 1\end{bmatrix} = \begin{bmatrix}11 & 7\end{bmatrix} = B'. \qquad (2.7)$$

Таким чином, результуючі координати для точки $A' - x' = 3$ та $y' = 1$. Відповідно, координати точки $B' - x' = 11$ та $y' = 7$. У більш компактному вигляді відрізок AB може бути поданий матрицею розміром 2×2:

$$L=\begin{bmatrix}0 & 1\\ 2 & 3\end{bmatrix}.$$

Добуток матриць відповідно дорівнює:

$$LT=\begin{bmatrix}0 & 1\\ 2 & 3\end{bmatrix}\begin{bmatrix}1 & 2\\ 3 & 1\end{bmatrix}=\begin{bmatrix}3 & 1\\ 11 & 7\end{bmatrix}=L'. \qquad (2.8)$$

Результат перетворення відрізка наведено на рис. 2.4. Із рисунка видно, що перетворення приводить до збільшення довжини відрізка та зміни його напрямку.

Додаткове вивчення можливостей матриці перетворення показує, що, крім зсуву, можливі ще операції повороту, відображення та масштабування. Розглянемо кожне з цих перетворень окремо.

Поворот

Розглянемо трикутник ABC (рис. 2.5) та за допомогою наступного перетворення повернемо його на 90° проти годинникової стрілки відносно початку координат

$$T=\begin{bmatrix}0 & 1\\ -1 & 0\end{bmatrix}.$$

Якщо використовувати матрицю розміром (3×2), яка складається з координат x та y вершин трикутника, то можна записати

$$\begin{bmatrix}3 & -1\\ 4 & 1\\ 2 & 1\end{bmatrix}\begin{bmatrix}0 & 1\\ -1 & 0\end{bmatrix}=\begin{bmatrix}1 & 3\\ -1 & 4\\ -1 & 2\end{bmatrix},$$

що є координатами результуючого трикутника $A'B'C'$. Поворот на 180° відносно початку координат досягається шляхом такого перетворення:

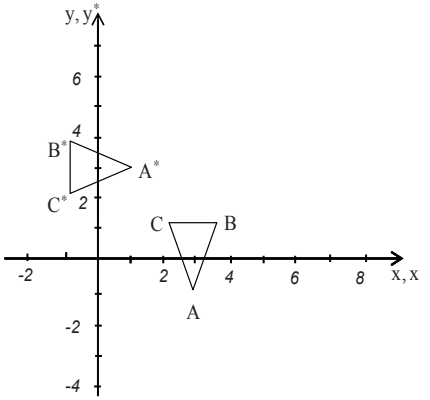


Рис. 2.5. Поворот

$$L=\begin{bmatrix}0 & 1\\ 2 & 3\end{bmatrix}.$$

Добуток матриць відповідно дорівнює:

$$LT=\begin{bmatrix}0 & 1\\ 2 & 3\end{bmatrix}\begin{bmatrix}1 & 2\\ 3 & 1\end{bmatrix}=\begin{bmatrix}3 & 1\\ 11 & 7\end{bmatrix}=L'. \qquad (2.8)$$

Результат перетворення відрізка наведено на рис. 2.4. Із рисунка видно, що перетворення приводить до збільшення довжини відрізка та зміни його напрямку.

Додаткове вивчення можливостей матриці перетворення показує, що, крім зсуву, можливі ще операції повороту, відображення та масштабування. Розглянемо кожне з цих перетворень окремо.

Поворот

Розглянемо трикутник ABC (рис. 2.5) та за допомогою наступного перетворення повернемо його на 90° проти годинникової стрілки відносно початку координат

$$T=\begin{bmatrix}0 & 1\\ -1 & 0\end{bmatrix}.$$

Якщо використовувати матрицю розміром (3×2), яка складається з координат x та y вершин трикутника, то можна записати

$$\begin{bmatrix}3 & -1\\ 4 & 1\\ 2 & 1\end{bmatrix}\begin{bmatrix}0 & 1\\ -1 & 0\end{bmatrix}=\begin{bmatrix}1 & 3\\ -1 & 4\\ -1 & 2\end{bmatrix},$$

що є координатами результуючого трикутника $A'B'C'$. Поворот на 180° відносно початку координат досягається шляхом такого перетворення:

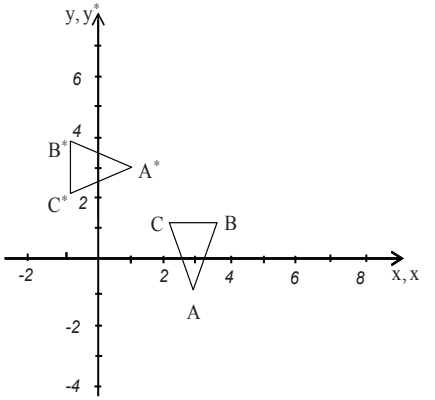


Рис. 2.5. Поворот

$$T = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix},$$

а на 270° відносно початку координат – перетворенням

$$T = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}.$$

Відповідно, що матриця перетворення

$$T = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

відповідає повороту навколо початку координат на 0° або 360°. У цих прикладах не зустрічалось ні масштабування, ні відображення.

У наведених прикладах здійснюється перетворення у спеціальних випадках повороту початку координат на кути 0°, 90°, 180° та 270°. Як здійснити поворот навколо початку координат на довільний кут θ? Для відповіді на це питання розглянемо вектор положення від початку координат до точки P(рис. 2.6). Позначимо r – довжину вектора, а φ – кут між вектором та віссю x. Вектор положення обертається навколо початку координат на кут θ та потрапляє в точку P^*. Записавши вектори положень для P та P^*, отримаємо:

$$P = \begin{bmatrix} x & y \end{bmatrix} = \begin{bmatrix} r\cos\varphi & r\sin\varphi \end{bmatrix}$$

та

$$P^* = \begin{bmatrix} x^* & y^* \end{bmatrix} = \begin{bmatrix} r\cos(\varphi+\theta) & r\sin(\varphi+\theta) \end{bmatrix}.$$

Використовуючи вираз для косинусу суми кутів, перепишемо вираз для P^* таким чином:

$$P^* = \begin{bmatrix} x^* & y^* \end{bmatrix} = \begin{bmatrix} r(\cos\varphi\cos\theta - \sin\varphi\sin\theta) & r(\cos\varphi\sin\theta + \sin\varphi\cos\theta) \end{bmatrix}$$

.

Використовуючи координати x та y, можна переписати P^* як

$$P^* = \begin{bmatrix} x^* & y^* \end{bmatrix} = \begin{bmatrix} x\cos\theta - y\sin\theta & x\sin\theta + y\cos\theta \end{bmatrix}.$$

Таким чином, перетворена точка має координати

$$x^* = x\cos\theta - y\sin\theta, \qquad (2.9)$$

$$y^* = x\sin\theta + y\cos\theta, \qquad (2.10)$$

або в матричному вигляді

$$T = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix},$$

а на 270° відносно початку координат – перетворенням

$$T = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}.$$

Відповідно, що матриця перетворення

$$T = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

відповідає повороту навколо початку координат на 0° або 360°. У цих прикладах не зустрічалось ні масштабування, ні відображення.

У наведених прикладах здійснюється перетворення у спеціальних випадках повороту початку координат на кути 0°, 90°, 180° та 270°. Як здійснити поворот навколо початку координат на довільний кут θ? Для відповіді на це питання розглянемо вектор положення від початку координат до точки P(рис. 2.6). Позначимо r – довжину вектора, а φ – кут між вектором та віссю x. Вектор положення обертається навколо початку координат на кут θ та потрапляє в точку P^*. Записавши вектори положень для P та P^*, отримаємо:

$$P = \begin{bmatrix} x & y \end{bmatrix} = \begin{bmatrix} r\cos\varphi & r\sin\varphi \end{bmatrix}$$

та

$$P^* = \begin{bmatrix} x^* & y^* \end{bmatrix} = \begin{bmatrix} r\cos(\varphi+\theta) & r\sin(\varphi+\theta) \end{bmatrix}.$$

Використовуючи вираз для косинусу суми кутів, перепишемо вираз для P^* таким чином:

$$P^* = \begin{bmatrix} x^* & y^* \end{bmatrix} = \begin{bmatrix} r(\cos\varphi\cos\theta - \sin\varphi\sin\theta) & r(\cos\varphi\sin\theta + \sin\varphi\cos\theta) \end{bmatrix}$$

.

Використовуючи координати x та y, можна переписати P^* як

$$P^* = \begin{bmatrix} x^* & y^* \end{bmatrix} = \begin{bmatrix} x\cos\theta - y\sin\theta & x\sin\theta + y\cos\theta \end{bmatrix}.$$

Таким чином, перетворена точка має координати

$$x^* = x\cos\theta - y\sin\theta, \qquad (2.9)$$

$$y^* = x\sin\theta + y\cos\theta, \qquad (2.10)$$

або в матричному вигляді

$$X^* = XT = \begin{bmatrix} x^* & y^* \end{bmatrix} = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}. \qquad (2.11)$$

Таким чином, перетворення повороту навколо точки початку координат на довільний кут θ задається матрицею

$$T = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}. \qquad (2.12)$$

Повороти є позитивними, якщо вони здійснюються проти годинникової стрілки відносно точки повороту (рис.2.6).

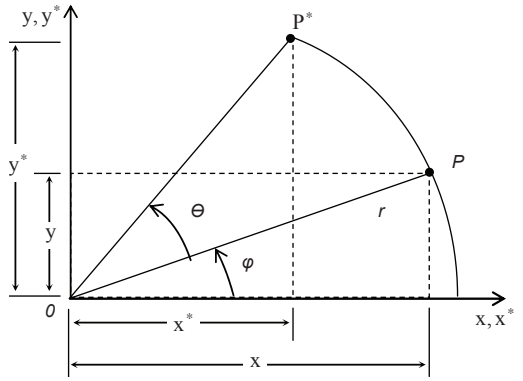


Рис. 2.6. Поворот координатного вектора

Визначник загальної матриці повороту має вигляд

$$\det T = \cos^2\theta + \sin^2\theta = 1.$$

У загальному випадку перетворення за матрицею з детермінантом, що дорівнює 1, приводять до повного повороту.

Відображення

У той час як повний поворот на площині xy зазвичай здійснюється в двовимірному просторі відносно нормалі до площини, відображення являє собою той же поворот на кут 180° у тривимірному просторі та зворотно на площину відносно осі, що лежить у площині xy. На рис. 2.7 наведено приклади двох відображень на площині

$$X^* = XT = \begin{bmatrix} x^* & y^* \end{bmatrix} = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}. \qquad (2.11)$$

Таким чином, перетворення повороту навколо точки початку координат на довільний кут θ задається матрицею

$$T = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}. \qquad (2.12)$$

Повороти є позитивними, якщо вони здійснюються проти годинникової стрілки відносно точки повороту (рис.2.6).

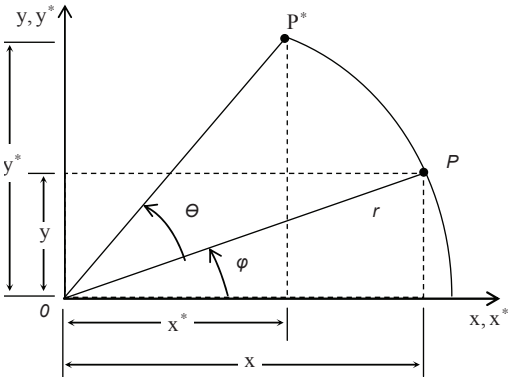


Рис. 2.6. Поворот координатного вектора

Визначник загальної матриці повороту має вигляд

$$\det T = \cos^2\theta + \sin^2\theta = 1.$$

У загальному випадку перетворення за матрицею з детермінантом, що дорівнює 1, приводять до повного повороту.

Відображення

У той час як повний поворот на площині xy зазвичай здійснюється в двовимірному просторі відносно нормалі до площини, відображення являє собою той же поворот на кут 180° у тривимірному просторі та зворотно на площину відносно осі, що лежить у площині xy. На рис. 2.7 наведено приклади двох відображень на площині

трикутника DEF. Відображення відносно прямої $y=0$ (вісь x) отримано з використанням матриці

$$T=\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}. \qquad (2.13)$$

У цьому випадку нові вершини трикутника $D^*E^*F^*$ будуть визначатись перетворенням

$$\begin{bmatrix}8 & 1\\ 7 & 3\\ 6 & 2\end{bmatrix}\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}=\begin{bmatrix}8 & -1\\ 7 & -3\\ 6 & -2\end{bmatrix}.$$

Подібним чином відображення відносно осі y при $x=0$ буде мати вигляд

$$T=\begin{bmatrix}-1 & 0\\ 0 & 1\end{bmatrix}. \qquad (2.14)$$

Відображення відносно прямої $y=x$ здійснюється за допомогою матриці

$$T=\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix} \qquad (2.15)$$

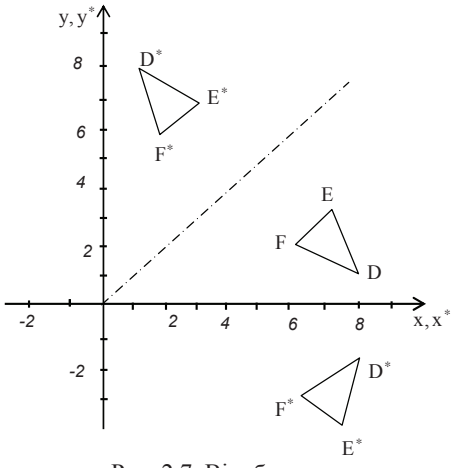


Рис. 2.7. Відображення

Виконавши перетворення, отримаємо координати вершин трикутника $D^+E^+F^+$

$$\begin{bmatrix}8 & 1\\ 7 & 3\\ 6 & 2\end{bmatrix}\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}=\begin{bmatrix}1 & 8\\ 3 & 7\\ 2 & 6\end{bmatrix}$$

Відображення відносно осі x задається відповідно матрицею

$$T=\begin{bmatrix}0 & -1\\ -1 & 0\end{bmatrix}$$

(2.16)

трикутника DEF. Відображення відносно прямої $y=0$ (вісь x) отримано з використанням матриці

$$T=\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}. \qquad (2.13)$$

У цьому випадку нові вершини трикутника $D^*E^*F^*$ будуть визначатись перетворенням

$$\begin{bmatrix}8 & 1\\ 7 & 3\\ 6 & 2\end{bmatrix}\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}=\begin{bmatrix}8 & -1\\ 7 & -3\\ 6 & -2\end{bmatrix}.$$

Подібним чином відображення відносно осі y при $x=0$ буде мати вигляд

$$T=\begin{bmatrix}-1 & 0\\ 0 & 1\end{bmatrix}. \qquad (2.14)$$

Відображення відносно прямої $y=x$ здійснюється за допомогою матриці

$$T=\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix} \qquad (2.15)$$

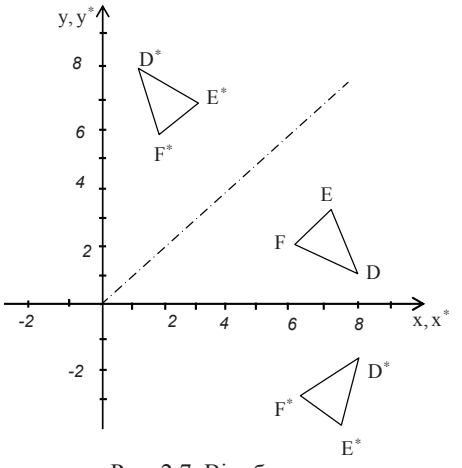


Рис. 2.7. Відображення

Виконавши перетворення, отримаємо координати вершин трикутника $D^+E^+F^+$

$$\begin{bmatrix}8 & 1\\ 7 & 3\\ 6 & 2\end{bmatrix}\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}=\begin{bmatrix}1 & 8\\ 3 & 7\\ 2 & 6\end{bmatrix}$$

Відображення відносно осі x задається відповідно матрицею

$$T=\begin{bmatrix}0 & -1\\ -1 & 0\end{bmatrix}$$

(2.16)

У кожній з матриць (2.13 – 2.16) визначник дорівнює -1. У загальному випадку, якщо визначник матриці перетворення дорівнює -1, то перетворення дає повне відображення.

Масштабування

Величина масштабування визначається значенням елементів вихідної діагональної матриці. Якщо матриця

$$T = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \qquad (2.17)$$

використовується як оператор впливу на вершини трикутника, то має місце «двократне» розширення або рівномірне масштабування відносно точки початку координат. Якщо значення елементів матриці перетворення не однакові, то трикутник спотворюється, що зображено на рис. 2.8. Трикутник ABC, перетворений за допомогою матриці (2.17), переходить у пропорційно збільшений трикутник $A^*B^*C^*$.

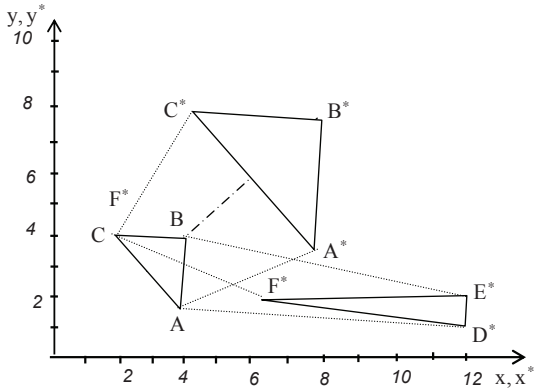


Рис. 2.8. Пропорційне та непропорційне масштабування

Той же трикутник, але перетворений за допомогою матриці

У кожній з матриць (2.13 – 2.16) визначник дорівнює -1. У загальному випадку, якщо визначник матриці перетворення дорівнює -1, то перетворення дає повне відображення.

Масштабування

Величина масштабування визначається значенням елементів вихідної діагональної матриці. Якщо матриця

$$T = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \qquad (2.17)$$

використовується як оператор впливу на вершини трикутника, то має місце «двократне» розширення або рівномірне масштабування відносно точки початку координат. Якщо значення елементів матриці перетворення не однакові, то трикутник спотворюється, що зображено на рис. 2.8. Трикутник ABC, перетворений за допомогою матриці (2.17), переходить у пропорційно збільшений трикутник $A^*B^*C^*$.

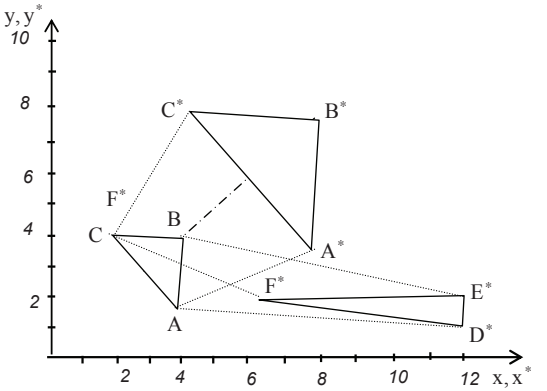


Рис. 2.8. Пропорційне та непропорційне масштабування

Той же трикутник, але перетворений за допомогою матриці

$$T = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & 3 \end{bmatrix},$$

переходить у трикутник $D^*E^*F^*$, що має спотворення, викликане різними коефіцієнтами масштабування.

У загальному випадку при матриці

$$T = \begin{bmatrix} a & b \\ c & d \end{bmatrix}, \qquad (2.18)$$

у якій $a = d$, $b = c = 0$, виконується пропорційне масштабування; якщо $a \neq d$, $b = c = 0$, то масштабування буде непропорційним. У першому випадку для $a = d > 1$ відбувається розширення, тобто збільшення зображення. Якщо $a = d < 1$, то відбувається рівномірне стиснення (зменшення).

Комбіноване масштабування

За допомогою матричних операцій над координатними векторами, що визначають вершини фігур, можна управляти формою та положенням поверхні. Однак для отримання бажаної орієнтації може бути необхідно більше одного перетворення. Оскільки операція добутку матриць не комутативна, то важливий порядок виконання перетворень.

Для ілюстрації ефекту некомутативності операцій добутку матриць розглянемо перетворення повороту та відображення координатного вектора $\begin{bmatrix} x & y \end{bmatrix}$. Якщо вслід за поворотом на 90° (за допомогою T_1) відбувається відображення відносно прямої $y = -x$ (за допомогою T_2), то ці два послідовних перетворення дають

$$X' = XT_1 = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} = \begin{bmatrix} -y & x \end{bmatrix}$$

та потім

$$X^* = X'T_2 = \begin{bmatrix} -y & x \end{bmatrix} \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} = \begin{bmatrix} -x & y \end{bmatrix}.$$

З іншого боку, якщо поворот відбувається після відображення, то отримаємо такий результат:

$$T = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & 3 \end{bmatrix},$$

переходить у трикутник $D^*E^*F^*$, що має спотворення, викликане різними коефіцієнтами масштабування.

У загальному випадку при матриці

$$T = \begin{bmatrix} a & b \\ c & d \end{bmatrix}, \qquad (2.18)$$

у якій $a = d$, $b = c = 0$, виконується пропорційне масштабування; якщо $a \neq d$, $b = c = 0$, то масштабування буде непропорційним. У першому випадку для $a = d > 1$ відбувається розширення, тобто збільшення зображення. Якщо $a = d < 1$, то відбувається рівномірне стиснення (зменшення).

Комбіноване масштабування

За допомогою матричних операцій над координатними векторами, що визначають вершини фігур, можна управляти формою та положенням поверхні. Однак для отримання бажаної орієнтації може бути необхідно більше одного перетворення. Оскільки операція добутку матриць не комутативна, то важливий порядок виконання перетворень.

Для ілюстрації ефекту некомутативності операцій добутку матриць розглянемо перетворення повороту та відображення координатного вектора $\begin{bmatrix} x & y \end{bmatrix}$. Якщо вслід за поворотом на 90° (за допомогою T_1) відбувається відображення відносно прямої $y = -x$ (за допомогою T_2), то ці два послідовних перетворення дають

$$X' = XT_1 = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} = \begin{bmatrix} -y & x \end{bmatrix}$$

та потім

$$X^* = X'T_2 = \begin{bmatrix} -y & x \end{bmatrix} \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} = \begin{bmatrix} -x & y \end{bmatrix}.$$

З іншого боку, якщо поворот відбувається після відображення, то отримаємо такий результат:

$$X' = XT_2 = [x \quad y]\begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} = [-y \quad -x]$$

та

$$X^* = X'T_1 = [-y \quad -x]\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} = [x \quad -y].$$

Обидва результати різні, що підтверджує важливість порядку застосування матричних перетворень.

2.3. Однорідні координати й матричне подання двовимірних перетворень

У випадку послідовного виконання будь-якої комбінації операцій повороту й масштабування результат легко можна записати у вигляді добутку матриць відповідних перетворень. Це буде матриця результуючого повороту й масштабування. Очевидно, що зручніше застосовувати результуючу матрицю замість того, щоб кожного разу заново обчислювати добуток матриць. Однак таким способом не можна одержати результуючу матрицю перетворення, якщо серед послідовності перетворень присутній хоча б один перенос.

Математично перенос можна описати за допомогою вектора переносу $\overline{AB}$. На рис. 2.9 точка A перенесена в точку B.Нехай $\overline{R}$ радіус-вектор, що відповідає вектору переносу $\overline{AB}$. Тоді перехід із точки A в точку B буде відповідати векторному запису $\overline{B} = \overline{A} + \overline{R}$. Звідси одержуємо, що для переносу точки в нове положення необхідно додати до її координат деякі числа, які являють собою координати вектора переносу:

$$\overline{B} = \overline{A} + \overline{R} = \left[A_x + R_x, A_y + R_y, A_z + R_z\right].$$

Матричний добуток у комп'ютерній графіці також називають композицією. Було б зручніше мати математичний апарат, що дозволяє включати в композиції перетворень всі три вищезазначені операції. При цьому вийшов би значний виграш у швидкості обчислень. Однорідні координати і є цей математичний апарат.

$$X' = XT_2 = [x \quad y]\begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} = [-y \quad -x]$$

та

$$X^* = X'T_1 = [-y \quad -x]\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} = [x \quad -y].$$

Обидва результати різні, що підтверджує важливість порядку застосування матричних перетворень.

2.3. Однорідні координати й матричне подання двовимірних перетворень

У випадку послідовного виконання будь-якої комбінації операцій повороту й масштабування результат легко можна записати у вигляді добутку матриць відповідних перетворень. Це буде матриця результуючого повороту й масштабування. Очевидно, що зручніше застосовувати результуючу матрицю замість того, щоб кожного разу заново обчислювати добуток матриць. Однак таким способом не можна одержати результуючу матрицю перетворення, якщо серед послідовності перетворень присутній хоча б один перенос.

Математично перенос можна описати за допомогою вектора переносу $\overline{AB}$. На рис. 2.9 точка A перенесена в точку B.Нехай $\overline{R}$ радіус-вектор, що відповідає вектору переносу $\overline{AB}$. Тоді перехід із точки A в точку B буде відповідати векторному запису $\overline{B} = \overline{A} + \overline{R}$. Звідси одержуємо, що для переносу точки в нове положення необхідно додати до її координат деякі числа, які являють собою координати вектора переносу:

$$\overline{B} = \overline{A} + \overline{R} = \left[A_x + R_x, A_y + R_y, A_z + R_z\right].$$

Матричний добуток у комп'ютерній графіці також називають композицією. Було б зручніше мати математичний апарат, що дозволяє включати в композиції перетворень всі три вищезазначені операції. При цьому вийшов би значний виграш у швидкості обчислень. Однорідні координати і є цей математичний апарат.

Двовимірний вектор *(x,y)* в однорідних координатах записується у вигляді *(wx,wy,w)*, де *w≠0*. Число *w* називається масштабним множником.

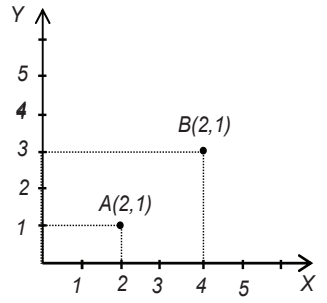


Рис. 2.9. Операція переносу або трансляції точки А в точку В

Для того щоб з вектора, записаного в однорідних координатах, одержати вектор у звичайних координатах, необхідно розділити перші дві координати на третю:

$$\left(\frac{wx}{w},\ \frac{wy}{w},\ \frac{w}{w}\right)\rightarrow(x,\ \ y,\ \ 1).$$

У загальному випадку здійснюється перехід від n-мір-ного простору до *(n+1)*-мірного. Це перетворення не єдине. Зворотне перетворення називається проекцією однорідних координат. Розглянемо деякі властивості однорідних координат.

Деякі точки, невизначені в n-вимірному просторі, стають цілком визначеними при переході до однорідних координат. Наприклад, однорідний вектор (0,0,1,0) у тривимірному просторі відповідає нескінченно вилученій точці $z=\infty$.

Оскільки в однорідних координатах цю точку можна подати у вигляді (0,0,1,ε), при ε→0, то в тривимірному просторі це відповідає точці (0,0,1/ε).

Приклад

Розглянемо точку тривимірного простору *(a,b,c)*. Якщо представити цю точку як однорідне подання точки двовимірного простору, то її координати будуть *(a/c,b/c)*. Порівнюючи ці координати із іншим видом формул, виведених для центральної перспективної проекції, легко помітити, що двовимірне подання точки з координатами *(a,b,c)* виглядає як її проекція на площину *z=1*, як показано на рис. 2.10.

Аналогічно, розглядаючи застосування однорідних координат для векторів тривимірного простору, можна подати тривимірний

Двовимірний вектор *(x,y)* в однорідних координатах записується у вигляді *(wx,wy,w)*, де *w≠0*. Число *w* називається масштабним множником.

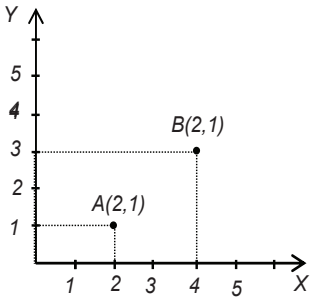


Рис. 2.9. Операція переносу або трансляції точки А в точку В

Для того щоб з вектора, записаного в однорідних координатах, одержати вектор у звичайних координатах, необхідно розділити перші дві координати на третю:

$$\left(\frac{wx}{w},\ \frac{wy}{w},\ \frac{w}{w}\right)\rightarrow(x,\ \ y,\ \ 1).$$

У загальному випадку здійснюється перехід від n-мір-ного простору до *(n+1)*-мірного. Це перетворення не єдине. Зворотне перетворення називається проекцією однорідних координат. Розглянемо деякі властивості однорідних координат.

Деякі точки, невизначені в n-вимірному просторі, стають цілком визначеними при переході до однорідних координат. Наприклад, однорідний вектор (0,0,1,0) у тривимірному просторі відповідає нескінченно вилученій точці $z=\infty$.

Оскільки в однорідних координатах цю точку можна подати у вигляді (0,0,1,ε), при ε→0, то в тривимірному просторі це відповідає точці (0,0,1/ε).

Приклад

Розглянемо точку тривимірного простору *(a,b,c)*. Якщо представити цю точку як однорідне подання точки двовимірного простору, то її координати будуть *(a/c,b/c)*. Порівнюючи ці координати із іншим видом формул, виведених для центральної перспективної проекції, легко помітити, що двовимірне подання точки з координатами *(a,b,c)* виглядає як її проекція на площину *z=1*, як показано на рис. 2.10.

Аналогічно, розглядаючи застосування однорідних координат для векторів тривимірного простору, можна подати тривимірний

простір як проекцію чотиривимірного простору на гіперплощину *w=1*, якщо *(x,y,z)→(wx,wy,wz)=(x,y,z,1)*. В однорідних координатах перетворення центральної перспективи визначається матричною операцією. Ця матриця запису-ється у вигляді

$$\begin{bmatrix} k & 0 & 0 & 0 \\ 0 & k & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & k \end{bmatrix} = P$$

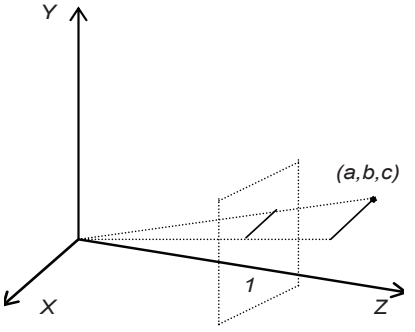


Рис. 2.10. Проекція точки (a,b,c) на площину z=1

Покажемо, що ця матриця визначає перетворення точки об'єкта, заданої в однорідних координатах, у точку перс-пективної проекції (також в однорідних координатах).

Нехай *p=(x,y,z)* – точка в тривимірному просторі. Її однорідне подання v=*(wx,wy,wz,w)*. Помножимо v на *P*:

$$vP = \begin{bmatrix} wkx, & wky, & 0 & w(z+k) \end{bmatrix} = \begin{bmatrix} \dfrac{kx}{(z+k)}, & \dfrac{ky}{(z+k)}, & 0, & 1 \end{bmatrix},$$

це в точності повторює формули, виведені для центральної перспективи.

Тоді точки двовимірного простору будуть описуватися трьохелементними векторами-рядками, тому й матриці перетворень, на які буде множитися вектор точки, будуть мати розміри (3×3). Запишемо матричне перетворення операції переносу для однорідних координат:

$$\begin{bmatrix} x', y', 1 \end{bmatrix} = \begin{bmatrix} x, y, 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ D_x & D_y & 1 \end{bmatrix} \quad \text{або} \quad p' = p \cdot T(D_x, D_y),$$

де $T(D_x, D_y) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ D_x & D_y & 1 \end{bmatrix}$

простір як проекцію чотиривимірного простору на гіперплощину *w=1*, якщо *(x,y,z)→(wx,wy,wz)=(x,y,z,1)*. В однорідних координатах перетворення центральної перспективи визначається матричною операцією. Ця матриця запису-ється у вигляді

$$\begin{bmatrix} k & 0 & 0 & 0 \\ 0 & k & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & k \end{bmatrix} = P$

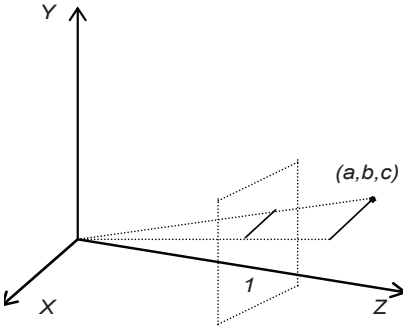


Рис. 2.10. Проекція точки (a,b,c) на площину z=1

Покажемо, що ця матриця визначає перетворення точки об'єкта, заданої в однорідних координатах, у точку перс-пективної проекції (також в однорідних координатах).

Нехай *p=(x,y,z)* – точка в тривимірному просторі. Її однорідне подання v=*(wx,wy,wz,w)*. Помножимо v на *P*:

$$vP = \begin{bmatrix} wkx, & wky, & 0 & w(z+k) \end{bmatrix} = \begin{bmatrix} \dfrac{kx}{(z+k)}, & \dfrac{ky}{(z+k)}, & 0, & 1 \end{bmatrix},$$

це в точності повторює формули, виведені для центральної перспективи.

Тоді точки двовимірного простору будуть описуватися трьохелементними векторами-рядками, тому й матриці перетворень, на які буде множитися вектор точки, будуть мати розміри (3×3). Запишемо матричне перетворення операції переносу для однорідних координат:

$$\begin{bmatrix} x', y', 1 \end{bmatrix} = \begin{bmatrix} x, y, 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ D_x & D_y & 1 \end{bmatrix} \quad \text{або} \quad p' = p \cdot T(D_x, D_y),$$

де $T(D_x, D_y) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ D_x & D_y & 1 \end{bmatrix}$

При послідовному переносі точки p в точку p' й потім у точку p'' компонента сумарного вектора переносу є сумою відповідних компонентів послідовних векторів переносу. Розглянемо, які будуть елементи матриці сумарного переносу. Нехай $p'=pT(D_x,D_y)$, $p''=p'T(D'_x,D'_y)$. Підставивши перше рівняння в друге, одержуємо $p''=pT(D_x,D_y)T(D'_x,D'_y)$.

Матричний добуток, тобто сумарний перенос, дорівнює добутку відповідних матриць переносу.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ D_x & D_y & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ D'_x & D'_y & 1 \end{bmatrix}=\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ D_x+D'_x & D_y+D'_y & 1 \end{bmatrix}. \qquad (2.19)$$

Запишемо матричний вид операції масштабування.

$$[x', y', 1]=[x, y, 1]\cdot\begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Визначимо матрицю масштабування:

$$S(S_x, S_y)=\begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Так само, як послідовні переноси є адитивними, покажемо, що послідовні масштабування будуть мультиплікативними.

$$S(S_x, S_y)\cdot S(S'_x, S'_y)=\begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} S'_x & 0 & 0 \\ 0 & S'_y & 0 \\ 0 & 0 & 1 \end{bmatrix}=\begin{bmatrix} S_x\cdot S'_x & 0 & 0 \\ 0 & S_y\cdot S'_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Для операції повороту матричний вид буде таким:

$$[x', y', 1]=[x, y, 1]\cdot\begin{bmatrix} Cos\alpha & Sin\alpha & 0 \\ -Sin\alpha & Cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Визначимо матрицю повороту:

$$R(\alpha)=\begin{bmatrix} Cos\alpha & Sin\alpha & 0 \\ -Sin\alpha & Cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

При послідовному переносі точки p в точку p' й потім у точку p'' компонента сумарного вектора переносу є сумою відповідних компонентів послідовних векторів переносу. Розглянемо, які будуть елементи матриці сумарного переносу. Нехай $p'=pT(D_x,D_y)$, $p''=p'T(D'_x,D'_y)$. Підставивши перше рівняння в друге, одержуємо $p''=pT(D_x,D_y)T(D'_x,D'_y)$.

Матричний добуток, тобто сумарний перенос, дорівнює добутку відповідних матриць переносу.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ D_x & D_y & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ D'_x & D'_y & 1 \end{bmatrix}=\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ D_x+D'_x & D_y+D'_y & 1 \end{bmatrix}. \qquad (2.19)$$

Запишемо матричний вид операції масштабування.

$$[x', y', 1]=[x, y, 1]\cdot\begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Визначимо матрицю масштабування:

$$S(S_x, S_y)=\begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Так само, як послідовні переноси є адитивними, покажемо, що послідовні масштабування будуть мультиплікативними.

$$S(S_x, S_y)\cdot S(S'_x, S'_y)=\begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} S'_x & 0 & 0 \\ 0 & S'_y & 0 \\ 0 & 0 & 1 \end{bmatrix}=\begin{bmatrix} S_x\cdot S'_x & 0 & 0 \\ 0 & S_y\cdot S'_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Для операції повороту матричний вид буде таким:

$$[x', y', 1]=[x, y, 1]\cdot\begin{bmatrix} Cos\alpha & Sin\alpha & 0 \\ -Sin\alpha & Cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Визначимо матрицю повороту:

$$R(\alpha)=\begin{bmatrix} Cos\alpha & Sin\alpha & 0 \\ -Sin\alpha & Cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Аналогічно двом попереднім випадкам, покажемо, що матриця повороту залишається такою ж при послідовних поворотах

$$R(\alpha)R(\beta)=\begin{bmatrix} Cos\alpha & Sin\alpha & 0 \\ -Sin\alpha & Cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} Cos\beta & Sin\beta & 0 \\ -Sin\beta & Cos\beta & 0 \\ 0 & 0 & 1 \end{bmatrix}=$$

$$=\begin{bmatrix} Cos(\alpha+\beta) & Sin(\alpha+\beta) & 0 \\ -Sin(\alpha+\beta) & Cos(\alpha+\beta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Таким чином, доведено, що два, а значить, і будь-яку кількість послідовних поворотів можна записати у вигляді однієї матриці сумарного повороту. Також легко помітити, що будь-яка послідовність операцій, що включає в себе перенос, масштабування й поворот в однорідних координатах, може бути подана однією матрицею, що є добутком матриць даних операцій.

Розглянемо, яким чином за допомогою композиції матричних перетворень можна одержати одне загальне результуюче перетворення. Для цього будемо використовувати матриці T, S і R. З обчислювальної точки зору набагато простіше й швидше застосовувати матрицю вже готового перетворення замість того, щоб застосовувати їх послідовно одну за одною. Для прикладу розглянемо задачу повороту об'єкта на площині щодо деякої довільної точки p_0. Поки ми вміємо повертати об'єкти тільки навколо початку координат. Але можна представити цю задачу як послідовність кроків, на кожному з яких буде застосовуватися тільки елементарна операція: перенос, масштабування або поворот.

Послідовність елементарних перетворень (рис. 2.11) полягає в наступному:

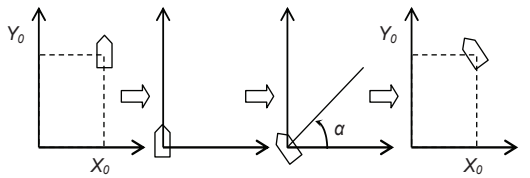


Рис. 2.11. Послідовність перетворень при повороті об'єкта навколо точки $p_0=(x_0, y_0)$ на кут α

Аналогічно двом попереднім випадкам, покажемо, що матриця повороту залишається такою ж при послідовних поворотах

$$R(\alpha)R(\beta)=\begin{bmatrix} Cos\alpha & Sin\alpha & 0 \\ -Sin\alpha & Cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} Cos\beta & Sin\beta & 0 \\ -Sin\beta & Cos\beta & 0 \\ 0 & 0 & 1 \end{bmatrix}=$$

$$=\begin{bmatrix} Cos(\alpha+\beta) & Sin(\alpha+\beta) & 0 \\ -Sin(\alpha+\beta) & Cos(\alpha+\beta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Таким чином, доведено, що два, а значить, і будь-яку кількість послідовних поворотів можна записати у вигляді однієї матриці сумарного повороту. Також легко помітити, що будь-яка послідовність операцій, що включає в себе перенос, масштабування й поворот в однорідних координатах, може бути подана однією матрицею, що є добутком матриць даних операцій.

Розглянемо, яким чином за допомогою композиції матричних перетворень можна одержати одне загальне результуюче перетворення. Для цього будемо використовувати матриці T, S і R. З обчислювальної точки зору набагато простіше й швидше застосовувати матрицю вже готового перетворення замість того, щоб застосовувати їх послідовно одну за одною. Для прикладу розглянемо задачу повороту об'єкта на площині щодо деякої довільної точки p_0. Поки ми вміємо повертати об'єкти тільки навколо початку координат. Але можна представити цю задачу як послідовність кроків, на кожному з яких буде застосовуватися тільки елементарна операція: перенос, масштабування або поворот.

Послідовність елементарних перетворень (рис. 2.11) полягає в наступному:

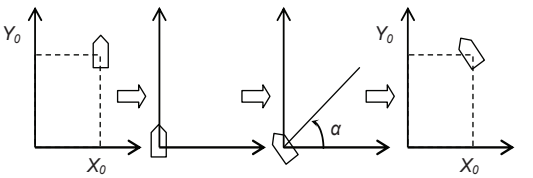


Рис. 2.11. Послідовність перетворень при повороті об'єкта навколо точки $p_0=(x_0, y_0)$ на кут α

2. Поворот на заданий кут.

3. Перенос, при якому точка з початку координат повертається в початкове положення p_0.

Точка $p_0=(x_0, y_0)$. Перший перенос здійснюється за допомогою вектора $[-x_0, -y_0]$, а зворотний перенос – за допомогою вектора $[x_0, y_0]$.

2.4. Ефективність обчислень

Розглянемо проблему прискорення обчислень в одній із найбільш трудомістких операцій комп'ютерної графіки - операції повороту точки відносно початку координат. Як було показано раніше, для її виконання необхідно зробити 4 операції множення, 2 операції додавання, а також обчислити значення синуса й косинуса кута повороту. Нагадаємо вид формул повороту:

$$x' = x \cdot Cos\alpha - y \cdot Sin\alpha$$
$$y' = x \cdot Sin\alpha + y \cdot Cos\alpha$$

Одним з найпопулярніших способів прискорення операції повороту є відмова від обчислення синуса й косинуса кута під час виконання програми і використання заздалегідь їх підрахованих значень, які занесені в спеціальну таблицю. Наприклад, у цій таблиці можуть зберігатися значення синусів і косинусів кутів повороту із кроком в 1 градус. Тоді ціла кількість градусів кута повороту може служити як індекс при добуванні відповідних значень синусів і косинусів з таблиці. Такий прийом називається табличним поворотом.

Додатковим способом прискорення операції повороту є зменшення кількості операцій множення. Розглянемо виведення формули О.Б'юнемана з використанням тангенса половинного кута, в якій поворот точки навколо початку координат виробляється за 3 операції множення й 3 операції додавання. Через те, що на багатьох мікропроцесорах операції множення виконуються довше, чим операції додавання, економія часу досягається за рахунок зменшення операцій множення. Виведення формули отримаємо з геометричних побудов, як показано на рис.2.12.

Виведемо координати x і y через x' і y'. На осі Ox відкладемо відрізок OS, такий що $OS=x'$. Тоді $x=OS-QS=x'-QS$. Тут відрізок QS є горизонтальною проекцією відрізка PT, де $PT=PU+UT$, $PU=y'$,

2. Поворот на заданий кут.

3. Перенос, при якому точка з початку координат повертається в початкове положення p_0.

Точка $p_0=(x_0, y_0)$. Перший перенос здійснюється за допомогою вектора $[-x_0, -y_0]$, а зворотний перенос – за допомогою вектора $[x_0, y_0]$.

2.4. Ефективність обчислень

Розглянемо проблему прискорення обчислень в одній із найбільш трудомістких операцій комп'ютерної графіки - операції повороту точки відносно початку координат. Як було показано раніше, для її виконання необхідно зробити 4 операції множення, 2 операції додавання, а також обчислити значення синуса й косинуса кута повороту. Нагадаємо вид формул повороту:

$$x' = x \cdot Cos\alpha - y \cdot Sin\alpha$$
$$y' = x \cdot Sin\alpha + y \cdot Cos\alpha$$

Одним з найпопулярніших способів прискорення операції повороту є відмова від обчислення синуса й косинуса кута під час виконання програми і використання заздалегідь їх підрахованих значень, які занесені в спеціальну таблицю. Наприклад, у цій таблиці можуть зберігатися значення синусів і косинусів кутів повороту із кроком в 1 градус. Тоді ціла кількість градусів кута повороту може служити як індекс при добуванні відповідних значень синусів і косинусів з таблиці. Такий прийом називається табличним поворотом.

Додатковим способом прискорення операції повороту є зменшення кількості операцій множення. Розглянемо виведення формули О.Б'юнемана з використанням тангенса половинного кута, в якій поворот точки навколо початку координат виробляється за 3 операції множення й 3 операції додавання. Через те, що на багатьох мікропроцесорах операції множення виконуються довше, чим операції додавання, економія часу досягається за рахунок зменшення операцій множення. Виведення формули отримаємо з геометричних побудов, як показано на рис.2.12.

Виведемо координати x і y через x' і y'. На осі Ox відкладемо відрізок OS, такий що $OS=x'$. Тоді $x=OS-QS=x'-QS$. Тут відрізок QS є горизонтальною проекцією відрізка PT, де $PT=PU+UT$, $PU=y'$,

$UT=x'tg(\theta/2) \Rightarrow x=x'-PT*Sin\theta$, де $PT=y'+x'tg(\theta/2)$. Тепер, знаючи x, можна виразити y у вигляді суми довжин відрізків QV і VP. Враховуючи, що довжини відрізків PV і PT рівні як радіуси окружності із центром у точці P, значення $y=x\cdot tg(\theta/2)+PT$. Позначимо $PT=T^*$, звідси виходить, що

$$T^*=y'+x'\cdot tg(\theta/2), x=x'-T^*Sin\theta, y=x'\cdot tg(\theta/2)+T^*.$$

Останні три рівності називаються формулою Б'юнемана.

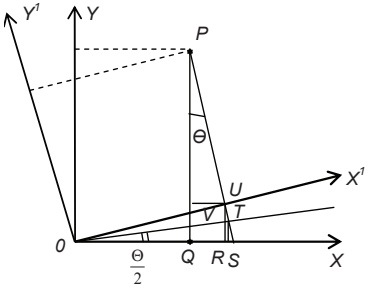


Рис. 2.12. Виведення формули О. Б'юнемана

Контрольні питання

1. Назвіть основні афінні геометричні перетворення на площині, сформулюйте їх особливості.
2. Яку роль відіграють матричний запис та однорідні координати в процесі застосування афінних геометричних перетворень у комп'ютерній графіці?
3. Застосуйте до квадрата такі перетворення: зсув вздовж осі Ox; масштабування вздовж осі Oy; обертання відносно осі Ox.
4. Наведіть класифікацію геометричних перетворень, що найчастіше використовуються в комп'ютерній графіці, сформулюйте їх особливості.

Тестові завдання

Питання 1. Матриця $T=\begin{vmatrix} 0 & 1 \\ -1 & 0 \end{vmatrix}$ здійснює:

А) відображення
Б) поворот на 180 градусів відносно початку координат
В) поворот на 90 градусів проти годинникової стрілки

$UT=x'tg(\theta/2) \Rightarrow x=x'-PT*Sin\theta$, де $PT=y'+x'tg(\theta/2)$. Тепер, знаючи x, можна виразити y у вигляді суми довжин відрізків QV і VP. Враховуючи, що довжини відрізків PV і PT рівні як радіуси окружності із центром у точці P, значення $y=x\cdot tg(\theta/2)+PT$. Позначимо $PT=T^*$, звідси виходить, що

$T^*=y'+x'\cdot tg(\theta/2), x=x'-T^*Sin\theta, y=x'\cdot tg(\theta/2)+T^*.$

Останні три рівності називаються формулою Б'юнемана.

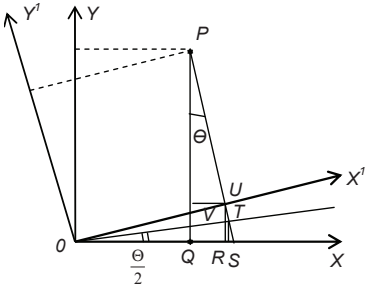


Рис. 2.12. Виведення формули О. Б'юнемана

Контрольні питання

1. Назвіть основні афінні геометричні перетворення на площині, сформулюйте їх особливості.
2. Яку роль відіграють матричний запис та однорідні координати в процесі застосування афінних геометричних перетворень у комп'ютерній графіці?
3. Застосуйте до квадрата такі перетворення: зсув вздовж осі Ox; масштабування вздовж осі Oy; обертання відносно осі Ox.
4. Наведіть класифікацію геометричних перетворень, що найчастіше використовуються в комп'ютерній графіці, сформулюйте їх особливості.

Тестові завдання

Питання 1. Матриця $T=\begin{vmatrix} 0 & 1 \\ -1 & 0 \end{vmatrix}$ здійснює:

А) відображення
Б) поворот на 180 градусів відносно початку координат
В) поворот на 90 градусів проти годинникової стрілки

Питання 2. Перетворення повороту навколо точки початку координат на довільний кут θ задається матрицею:

А) $T = \begin{vmatrix} \sin\theta & -\cos\theta \\ -\cos\theta & \sin\theta \end{vmatrix}$

Б) $T = \begin{vmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{vmatrix}$

В) $T = \begin{vmatrix} \sin\theta & \cos\theta \\ -\cos\theta & \sin\theta \end{vmatrix}$

Питання 3. Формула Б'юнемана використовується для:

А) розрахунку та виведення на екран плоских кривих
Б) прискорення операцій повороту
В) масштабування
Г) відповіді Б) та В)

Питання 4. Матриця $T = \begin{vmatrix} 2 & 0 \\ 0 & 2 \end{vmatrix}$ використовується для:

А) зменшення
Б) розширення
В) відображення
Г) зсуву

Питання 5. Координати точки, зображеної на екрані, залишаться незмінними, якщо матриця перетворень має вигляд:

А) $T = \begin{vmatrix} 0 & 0 \\ 0 & 1 \end{vmatrix}$

Б) $T = \begin{vmatrix} 1 & 1 \\ 1 & 1 \end{vmatrix}$

В) $T = \begin{vmatrix} 1 & 0 \\ 0 & 1 \end{vmatrix}$

Питання 6. У випадку, коли матриця перетворення має вигляд $T = \begin{vmatrix} a & 0 \\ 0 & d \end{vmatrix}$:

А) координати точки залишаються незмінними
Б) отримаємо відображення точки відносно початку координат
В) перетворення викликає зміну обох координат

Питання 7. Якщо в матриці $T = \begin{vmatrix} a & 0 \\ 0 & d \end{vmatrix}$ коефіцієнти a або d від'ємні:

А) відбувається відображення відносно координатної осі
Б) відбувається відображення відносно початку координат
В) координати точки залишаються незмінними

Питання 8. Недіагональні елементи матриці перетворення створюють ефект

А) повороту
Б) зсуву
В) відображення

Питання 9.У загальному випадку перетворення за матрицею з детермінантом, що дорівнює 1, призводять

А) до відображення
Б) до повного повороту
В) не змінюють координати точки

Питання 2. Перетворення повороту навколо точки початку координат на довільний кут θ задається матрицею:

А) $T = \begin{vmatrix} \sin\theta & -\cos\theta \\ -\cos\theta & \sin\theta \end{vmatrix}$

Б) $T = \begin{vmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{vmatrix}$

В) $T = \begin{vmatrix} \sin\theta & \cos\theta \\ -\cos\theta & \sin\theta \end{vmatrix}$

Питання 3. Формула Б'юнемана використовується для:

А) розрахунку та виведення на екран плоских кривих
Б) прискорення операцій повороту
В) масштабування
Г) відповіді Б) та В)

Питання 4. Матриця $T = \begin{vmatrix} 2 & 0 \\ 0 & 2 \end{vmatrix}$ використовується для:

А) зменшення
Б) розширення
В) відображення
Г) зсуву

Питання 5. Координати точки, зображеної на екрані, залишаться незмінними, якщо матриця перетворень має вигляд:

А) $T = \begin{vmatrix} 0 & 0 \\ 0 & 1 \end{vmatrix}$

Б) $T = \begin{vmatrix} 1 & 1 \\ 1 & 1 \end{vmatrix}$

В) $T = \begin{vmatrix} 1 & 0 \\ 0 & 1 \end{vmatrix}$

Питання 6. У випадку, коли матриця перетворення має вигляд $T = \begin{vmatrix} a & 0 \\ 0 & d \end{vmatrix}$:

А) координати точки залишаються незмінними
Б) отримаємо відображення точки відносно початку координат
В) перетворення викликає зміну обох координат

Питання 7. Якщо в матриці $T = \begin{vmatrix} a & 0 \\ 0 & d \end{vmatrix}$ коефіцієнти a або d від'ємні:

А) відбувається відображення відносно координатної осі
Б) відбувається відображення відносно початку координат
В) координати точки залишаються незмінними

Питання 8. Недіагональні елементи матриці перетворення створюють ефект

А) повороту
Б) зсуву
В) відображення

Питання 9.У загальному випадку перетворення за матрицею з детермінантом, що дорівнює 1, призводять

А) до відображення
Б) до повного повороту
В) не змінюють координати точки

95

Глава 3
Математичні й алгоритмічні основи тривимірної графіки

3.1. Просторові перетворення

Здатність відображати просторовий об'єкт є основою для розуміння форми цього об'єкта. Крім того, у багатьох випадках для цього важлива здатність обертати, переносити й будувати різні види проекцій об'єкта. Щоб зрозуміти його форму, ми відразу починаємо обертати об'єкт, відсувати на відстань витягнутої руки, пересувати нагору й униз, вперед та назад і т. д. Щоб зробити те ж саме за допомогою комп'ютера, ми повинні поширити наш попередній двовимірний аналіз на три виміри [11].

З попередньої глави зрозуміло, що при розв'язанні задач побудови композицій геометричних перетворень, у комп'ютерній графіці важливу роль відіграє математичний апарат матричних перетворень та однорідних координат. Таким чином, точку в тривимірному просторі $[x\ y\ z]$ можна зобразити чотиривимірним вектором

$$[x' \quad y' \quad z' \quad h] = [x \quad y \quad z \quad 1]T,$$

де T є матрицею деякого перетворення. Як і раніше, перетворення однорідних координат у звичайні задається формулою

$$[x* \quad y* \quad z* \quad h] = \left[\frac{x'}{h} \quad \frac{y'}{h} \quad \frac{z'}{h} \quad 1\right]. \qquad (3.1)$$

Узагальнену матрицю перетворення розмірності 4×4 для тримірних однорідних координат можна подати так:

Глава 3
Математичні й алгоритмічні основи тривимірної графіки

3.1. Просторові перетворення

Здатність відображати просторовий об'єкт є основою для розуміння форми цього об'єкта. Крім того, у багатьох випадках для цього важлива здатність обертати, переносити й будувати різні види проекцій об'єкта. Щоб зрозуміти його форму, ми відразу починаємо обертати об'єкт, відсувати на відстань витягнутої руки, пересувати нагору й униз, вперед та назад і т. д. Щоб зробити те ж саме за допомогою комп'ютера, ми повинні поширити наш попередній двовимірний аналіз на три виміри [11].

З попередньої глави зрозуміло, що при розв'язанні задач побудови композицій геометричних перетворень, у комп'ютерній графіці важливу роль відіграє математичний апарат матричних перетворень та однорідних координат. Таким чином, точку в тривимірному просторі $[x\ y\ z]$ можна зобразити чотиривимірним вектором

$$[x' \quad y' \quad z' \quad h] = [x \quad y \quad z \quad 1]T,$$

де T є матрицею деякого перетворення. Як і раніше, перетворення однорідних координат у звичайні задається формулою

$$[x* \quad y* \quad z* \quad h] = \left[\frac{x'}{h} \quad \frac{y'}{h} \quad \frac{z'}{h} \quad 1\right]. \qquad (3.1)$$

Узагальнену матрицю перетворення розмірності 4×4 для тримірних однорідних координат можна подати так:

$$T = \begin{bmatrix} a & b & c & p \\ d & e & f & q \\ q & i & j & r \\ l & m & n & s \end{bmatrix}. \qquad (3.2)$$

Матрицю перетворення 4×4 з (3.2) можна розділити на чотири окремі частини:

$$T = \begin{bmatrix} & & 3 \\ 3\times 3 & & \times \\ & & 1 \\ 1\times 3 & & 1\times 1 \end{bmatrix} \qquad (3.3)$$

Верхня ліва (3×3) – підматриця задає лінійне перетворення у формі масштабування, зрушення, відображення і обертання. Ліва нижня (1×3)– підматриця задає переміщення, а права верхня (3×1) – підматриця перспективного перетворення. Остання права нижня (1×1) – підматриця задає спільне масштабування. Спільне перетворення, отримане після застосування цієї (4×4) – матриці до однорідного вектора і обчислення звичайних координат, називається *білінійним* перетворенням. Дане перетворення здійснює комбінацію зсуву, локального масштабування, обертання, відображення, переміщення, перспективного перетворення і масштабування.

Тривимірне масштабування

Діагональні елементи (4×4) - матриці узагальненого перетворення задають локальне і загальне масштабування. Для ілюстрації цього роздивимось перетворення

$$XT = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & e & 0 & 0 \\ 0 & 0 & j & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \qquad (3.4)$$

$$= \begin{bmatrix} ax & ey & jz & 1 \end{bmatrix} = \begin{bmatrix} x* & y* & z* & 1 \end{bmatrix}$$

яке показує дію локального масштабування. Нижче наводиться приклад.

$$T = \begin{bmatrix} a & b & c & p \\ d & e & f & q \\ q & i & j & r \\ l & m & n & s \end{bmatrix}. \qquad (3.2)$$

Матрицю перетворення 4×4 з (3.2) можна розділити на чотири окремі частини:

$$T = \begin{bmatrix} & & 3 \\ 3\times 3 & & \times \\ & & 1 \\ 1\times 3 & & 1\times 1 \end{bmatrix} \qquad (3.3)$$

Верхня ліва (3×3) – підматриця задає лінійне перетворення у формі масштабування, зрушення, відображення і обертання. Ліва нижня (1×3)– підматриця задає переміщення, а права верхня (3×1) – підматриця перспективного перетворення. Остання права нижня (1×1) – підматриця задає спільне масштабування. Спільне перетворення, отримане після застосування цієї (4×4) – матриці до однорідного вектора і обчислення звичайних координат, називається *білінійним* перетворенням. Дане перетворення здійснює комбінацію зсуву, локального масштабування, обертання, відображення, переміщення, перспективного перетворення і масштабування.

Тривимірне масштабування

Діагональні елементи (4×4) - матриці узагальненого перетворення задають локальне і загальне масштабування. Для ілюстрації цього роздивимось перетворення

$$XT = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & e & 0 & 0 \\ 0 & 0 & j & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \qquad (3.4)$$

$$= \begin{bmatrix} ax & ey & jz & 1 \end{bmatrix} = \begin{bmatrix} x* & y* & z* & 1 \end{bmatrix}$$

яке показує дію локального масштабування. Нижче наводиться приклад.

Приклад. Локальне масштабування.

Роздивимося прямокутний паралелепіпед, який показаний на рис.3.1, з наведеними нижче однорідними координатами вершин:

$$X = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 2 & 0 & 1 & 1 \\ 2 & 3 & 1 & 1 \\ 0 & 3 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 2 & 0 & 0 & 1 \\ 2 & 3 & 0 & 1 \\ 0 & 3 & 0 & 1 \end{bmatrix}.$$

Щоб отримати одиничний куб з паралелепіпеда за допомогою локального масштабування, необхідно масштабувати множники 1/2, 1/3, 1 вздовж осей *x*, *y*, *z* відповідно. Перетворення локального масштабування задається матрицею

$$T = \begin{bmatrix} 1/2 & 0 & 0 & 0 \\ 0 & 1/3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Результуючий куб має такі однорідні координати вершин:

$$X = XT = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 2 & 0 & 1 & 1 \\ 2 & 3 & 1 & 1 \\ 0 & 3 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 2 & 0 & 0 & 1 \\ 2 & 3 & 0 & 1 \\ 0 & 3 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1/2 & 0 & 0 & 0 \\ 0 & 1/3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}.$$

Відмітимо, що однорідний координатний множник *h* дорівнює одиниці для кожної з перетворених вершин. Результат масштабування показаний на рис.3.1,в.

Приклад. Локальне масштабування.

Роздивимося прямокутний паралелепіпед, який показаний на рис.3.1, з наведеними нижче однорідними координатами вершин:

$$X = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 2 & 0 & 1 & 1 \\ 2 & 3 & 1 & 1 \\ 0 & 3 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 2 & 0 & 0 & 1 \\ 2 & 3 & 0 & 1 \\ 0 & 3 & 0 & 1 \end{bmatrix}.$$

Щоб отримати одиничний куб з паралелепіпеда за допомогою локального масштабування, необхідно масштабувати множники 1/2, 1/3, 1 вздовж осей *x*, *y*, *z* відповідно. Перетворення локального масштабування задається матрицею

$$T = \begin{bmatrix} 1/2 & 0 & 0 & 0 \\ 0 & 1/3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Результуючий куб має такі однорідні координати вершин:

$$X = XT = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 2 & 0 & 1 & 1 \\ 2 & 3 & 1 & 1 \\ 0 & 3 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 2 & 0 & 0 & 1 \\ 2 & 3 & 0 & 1 \\ 0 & 3 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1/2 & 0 & 0 & 0 \\ 0 & 1/3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}.$$

Відмітимо, що однорідний координатний множник *h* дорівнює одиниці для кожної з перетворених вершин. Результат масштабування показаний на рис.3.1,в.

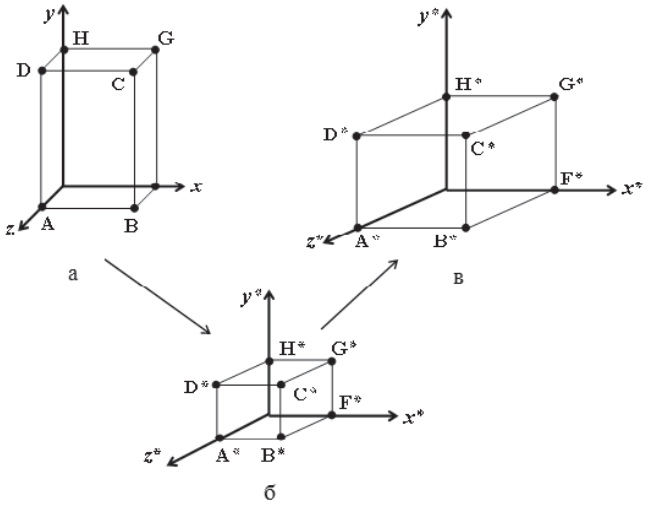


Рис. 3.1. Тривимірне масштабування:
а – початковий паралелепіпед; б – отримання одиничного кубу;
в – результат масштабування

Загальне масштабування можна здійснити, скориставшись четвертим діагональним елементом, тобто

$$XT = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & s \end{bmatrix} = \begin{bmatrix} x' & y' & z' & s \end{bmatrix}.$$

Звичайні або фізичні координати мають вигляд

$$\begin{bmatrix} x* & y* & z* & 1 \end{bmatrix} = \begin{bmatrix} \frac{z'}{s} & \frac{y'}{s} & \frac{z'}{s} & 1 \end{bmatrix}.$$

Тривимірні зрушення

Недіагональні елементи у верхній лівій 3×3- підматриці узагальненої матриці перетворення розміром 4×4 задають зрушення в трьох вимірах, тобто

Рис. 3.1. Тривимірне масштабування:
а – початковий паралелепіпед; б – отримання одиничного кубу;
в – результат масштабування

Загальне масштабування можна здійснити, скориставшись четвертим діагональним елементом, тобто

$$XT = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & s \end{bmatrix} = \begin{bmatrix} x' & y' & z' & s \end{bmatrix}.$$

Звичайні або фізичні координати мають вигляд

$$\begin{bmatrix} x* & y* & z* & 1 \end{bmatrix} = \begin{bmatrix} \frac{z'}{s} & \frac{y'}{s} & \frac{z'}{s} & 1 \end{bmatrix}.$$

Тривимірні зрушення

Недіагональні елементи у верхній лівій 3×3- підматриці узагальненої матриці перетворення розміром 4×4 задають зрушення в трьох вимірах, тобто

$$XT = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} 1 & b & c & 0 \\ d & 1 & f & 0 \\ q & i & 1 & 0 \\ 0 & 0 & 0 & s \end{bmatrix} =$$

$$= \begin{bmatrix} bx + yd + qz & bx + y + iz & cx + fy + z & 1 \end{bmatrix} \quad (3.5)$$

Тривимірне обертання

Перед тим як перейти до тривимірного обертання навколо довільної осі, роздивимось обертання навколо кожної координатної осі. Під час обертання навколо осі x залишаються незмінними x-координати координатного вектора.

Фактично обертання відбувається в площинах, які перпендикулярні відповідно осям y і z. Перетворення координатного вектора в кожній з цих площин задається вказаною в главі 2 матрицею двовимірного обертання. Ця матриця і незмінність координати x під час обертання навколо осі x дозволяє записати матрицю розміром 4×4, що використовується для перетворення однорідних координат у випадку повороту на кут Θ:

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\Theta & \sin\Theta & 0 \\ 0 & -\sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (3.6)$$

Обертання вважається позитивним за правилом правої руки, тобто за часовою стрілкою, якщо дивитися з початку координат у позитивному напрямку осі обертання. На рис. 3.2,а зображений паралелепіпед, що був отриманий поворотом на -90° навколо осі x паралелепіпеда з рис. 3.2,б. Аналогічно матриця перетворення для обертання навколо осі z на кут ψ має вигляд

$$T = \begin{bmatrix} \cos\psi & \sin\psi & 0 & 0 \\ -\sin\psi & \cos\psi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (3.7)$$

Під час обертання на кут φ навколо осі y перетворення має вигляд

$$XT = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} 1 & b & c & 0 \\ d & 1 & f & 0 \\ q & i & 1 & 0 \\ 0 & 0 & 0 & s \end{bmatrix} =$$

$$= \begin{bmatrix} bx + yd + qz & bx + y + iz & cx + fy + z & 1 \end{bmatrix} \quad (3.5)$$

Тривимірне обертання

Перед тим як перейти до тривимірного обертання навколо довільної осі, роздивимось обертання навколо кожної координатної осі. Під час обертання навколо осі x залишаються незмінними x-координати координатного вектора.

Фактично обертання відбувається в площинах, які перпендикулярні відповідно осям y і z. Перетворення координатного вектора в кожній з цих площин задається вказаною в главі 2 матрицею двовимірного обертання. Ця матриця і незмінність координати x під час обертання навколо осі x дозволяє записати матрицю розміром 4×4, що використовується для перетворення однорідних координат у випадку повороту на кут Θ:

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\Theta & \sin\Theta & 0 \\ 0 & -\sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (3.6)$$

Обертання вважається позитивним за правилом правої руки, тобто за часовою стрілкою, якщо дивитися з початку координат у позитивному напрямку осі обертання. На рис. 3.2,а зображений паралелепіпед, що був отриманий поворотом на -90° навколо осі x паралелепіпеда з рис. 3.2,б. Аналогічно матриця перетворення для обертання навколо осі z на кут ψ має вигляд

$$T = \begin{bmatrix} \cos\psi & \sin\psi & 0 & 0 \\ -\sin\psi & \cos\psi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (3.7)$$

Під час обертання на кут φ навколо осі y перетворення має вигляд

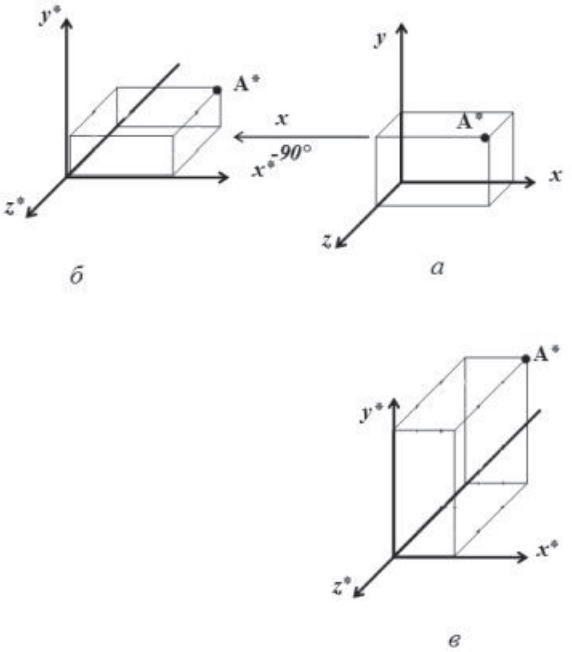


Рис. 3.2. Тривимірне масштабування:
а – початковий стан; б – обертання навколо осі x;
в – обертання навколо осі z

$$T = \begin{bmatrix} \cos\phi & 0 & -\sin\phi & 0 \\ 0 & 1 & 0 & 0 \\ \sin\phi & 0 & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \qquad (3.8)$$

Помітимо, що в (3.8) знаки у синусів протилежні знакам цих членів у рівності (3.6) і (3.7). Це необхідно для того, щоб виконувалась угода про позитивний напрямок за правилом правої руки.

Із рівностей (3.6) – (3.8) виходить, що детермінант кожної з матриць перетворень дорівнює +1, що і необхідно для правильного обертання.

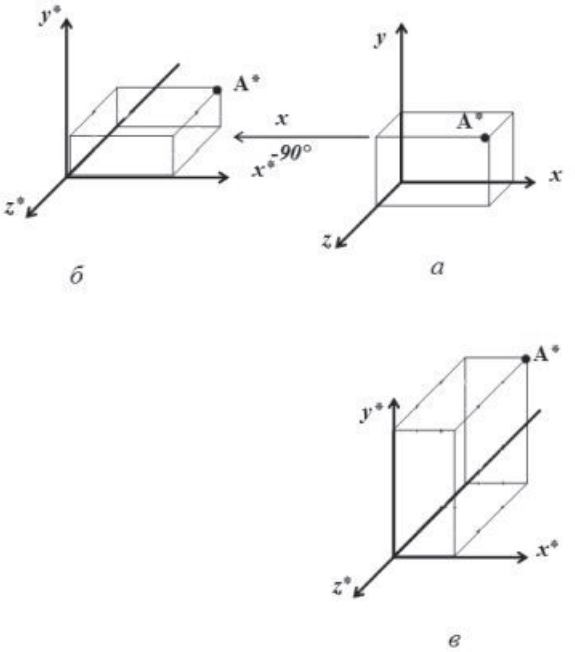


Рис. 3.2. Тривимірне масштабування:
а – початковий стан; б – обертання навколо осі x;
в – обертання навколо осі z

$$T = \begin{bmatrix} \cos\phi & 0 & -\sin\phi & 0 \\ 0 & 1 & 0 & 0 \\ \sin\phi & 0 & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \qquad (3.8)$$

Помітимо, що в (3.8) знаки у синусів протилежні знакам цих членів у рівності (3.6) і (3.7). Це необхідно для того, щоб виконувалась угода про позитивний напрямок за правилом правої руки.

Із рівностей (3.6) – (3.8) виходить, що детермінант кожної з матриць перетворень дорівнює +1, що і необхідно для правильного обертання.

101

Тривимірне відображення

Деякі орієнтації тривимірного об'єкта не можна отримати одними обертаннями, потрібно перетворювати відображення. У тривимірному просторі відображення відбувається відносно площини. За аналогією з двомірним відображенням, тривимірне відображення відносно площини еквівалентно обертанню навколо осі у тривимірному просторі в чотиривимірний простір і зворотно у вихідний тривимірний простір. Для повного відображення детермінант матриці дорівнює -1.

У випадку відображення відносно площини xy змінюються лише значення z-координати координатного вектору об'єкта. Таким чином, матриця перетворення для відображення відносно площини xy дорівнює

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (3.9)$$

На рис. 3.3 зображено відображення одиничного кубу відносно площини xy. У випадку відображення відносно площини yz матриця перетворення має вигляд

$$T = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \qquad (3.10)$$

а для відображення відносно площини xz дорівнює

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \qquad (3.11)$$

Тобто можна побачити, що при здійсненні процедури відображення площина, відносно якої здійснюється перетворення, обирається за рахунок зміни діоганальних елементів одиничної матриці.

Тривимірне відображення

Деякі орієнтації тривимірного об'єкта не можна отримати одними обертаннями, потрібно перетворювати відображення. У тривимірному просторі відображення відбувається відносно площини. За аналогією з двомірним відображенням, тривимірне відображення відносно площини еквівалентно обертанню навколо осі у тривимірному просторі в чотиривимірний простір і зворотно у вихідний тривимірний простір. Для повного відображення детермінант матриці дорівнює -1.

У випадку відображення відносно площини xy змінюються лише значення z-координати координатного вектору об'єкта. Таким чином, матриця перетворення для відображення відносно площини xy дорівнює

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (3.9)$$

На рис. 3.3 зображено відображення одиничного кубу відносно площини xy. У випадку відображення відносно площини yz матриця перетворення має вигляд

$$T = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \qquad (3.10)$$

а для відображення відносно площини xz дорівнює

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \qquad (3.11)$$

Тобто можна побачити, що при здійсненні процедури відображення площина, відносно якої здійснюється перетворення, обирається за рахунок зміни діоганальних елементів одиничної матриці.

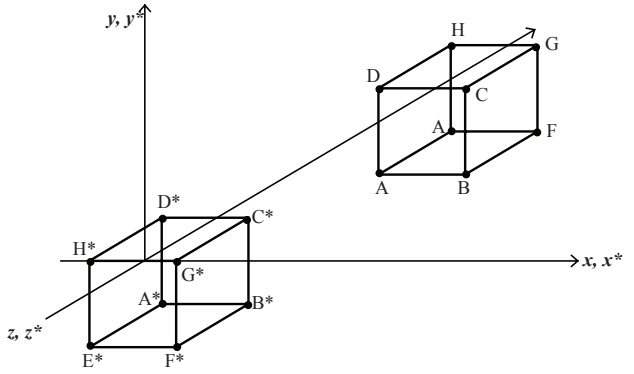


Рис. 3.3. Тривимірне відображення відносно площини xy

Просторове перенесення

Матриця просторового перенесення має вигляд:

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ l & m & n & 1 \end{bmatrix} \qquad (3.12)$$

Пересувні однорідні координати отримуються за допомогою перетворення

$$\begin{bmatrix} x' & y' & z' & h \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ l & m & n & 1 \end{bmatrix}.$$

Виконавши множення, отримаємо

$$\begin{bmatrix} x' & y' & z' & h \end{bmatrix} = \begin{bmatrix} (x+l) & (y+m) & (z+n) & 1 \end{bmatrix}. \qquad (3.13)$$

З цього випливає, що перетворювані фізичні координати дорівнюють

$$x^* = x + l,$$
$$y^* = y + m,$$
$$z^* = z + n.$$

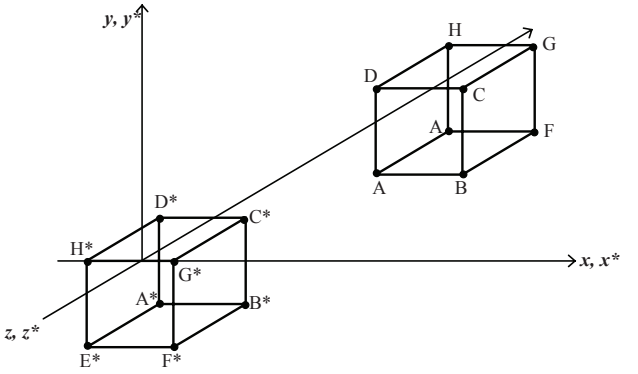


Рис. 3.3. Тривимірне відображення відносно площини xy

Просторове перенесення

Матриця просторового перенесення має вигляд:

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ l & m & n & 1 \end{bmatrix} \qquad (3.12)$$

Пересувні однорідні координати отримуються за допомогою перетворення

$$\begin{bmatrix} x' & y' & z' & h \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ l & m & n & 1 \end{bmatrix}.$$

Виконавши множення, отримаємо

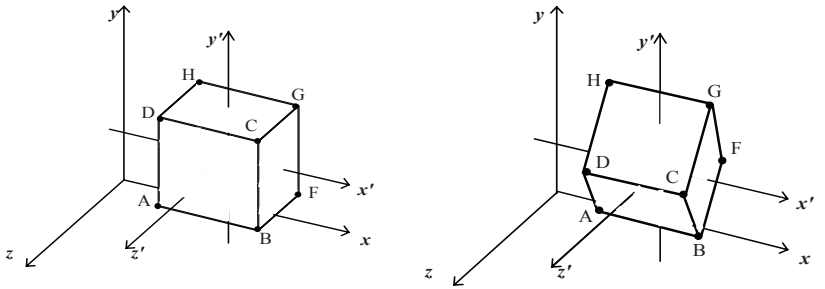
$$\begin{bmatrix} x' & y' & z' & h \end{bmatrix} = \begin{bmatrix} (x+l) & (y+m) & (z+n) & 1 \end{bmatrix}. \qquad (3.13)$$

З цього випливає, що перетворювані фізичні координати дорівнюють

$$x^* = x + l,$$
$$y^* = y + m,$$
$$z^* = z + n.$$

Обертання навколо осі, паралельній координатній осі

Перетворення, що задаються рівняннями (3.6) – (3.8), описують обертання навколо координатних осей x, y та z. Однак доволі часто буває необхідно обертати об'єкт навколо осі, що не збігається з цими трьома. Розглянемо частковий випадок для осі, паралельній одній з координатних осей x, y або z. На рис. 3.4, а зображено фігуру в локальній системі осей $x'y'z'$, які паралельні фіксованій глобальній системі xyz. На рис. 3.4, б – результат повороту на 30 градусів, що відбувається відносно осі x'.



а б

Рис. 3.4. Поворот навколо осі, яка паралельна одній з координатних осей: а – початкове положення; б – поворот на 30 градусів відносно осі x'

Обертання фігури навколо будь-якої з локальних осей виконуються за допомогою такої процедури:

– перемістити фігуру так, щоб локальна вісь збігалась із координатною;

– повернути навколо вказаної осі;

– перемістити перетворену фігуру у вихідне положення.

Математично це можна описати таким чином:

$$X^* = XT_r R_x T_r^{-1},$$

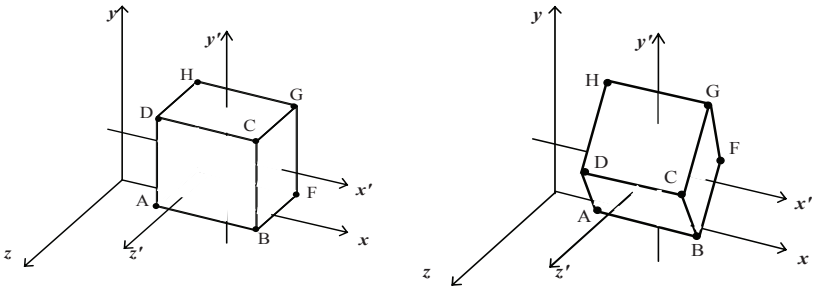
де

X^* – перетворювана фігура;

X – вихідна фігура;

Обертання навколо осі, паралельній координатній осі

Перетворення, що задаються рівняннями (3.6) – (3.8), описують обертання навколо координатних осей x, y та z. Однак доволі часто буває необхідно обертати об'єкт навколо осі, що не збігається з цими трьома. Розглянемо частковий випадок для осі, паралельній одній з координатних осей x, y або z. На рис. 3.4, а зображено фігуру в локальній системі осей $x'y'z'$, які паралельні фіксованій глобальній системі xyz. На рис. 3.4, б – результат повороту на 30 градусів, що відбувається відносно осі x'.



а б

Рис. 3.4. Поворот навколо осі, яка паралельна одній з координатних осей: а – початкове положення; б – поворот на 30 градусів відносно осі x'

Обертання фігури навколо будь-якої з локальних осей виконуються за допомогою такої процедури:

– перемістити фігуру так, щоб локальна вісь збігалась із координатною;

– повернути навколо вказаної осі;

– перемістити перетворену фігуру у вихідне положення.

Математично це можна описати таким чином:

$$X^* = XT_r R_x T_r^{-1},$$

де

X^* – перетворювана фігура;

X – вихідна фігура;

T_r – матриця переміщення;

R_x – відповідна матриця повороту;

T_r^{-1} – матриця, зворотна до матриці переміщення.

Для того щоб здійснити декілька поворотів у локальній системі координат, паралельній осям глобальної системи координат, потрібно сумістити початки глобальної та локальної систем [13].

Обертання навколо довільної осі у просторі

Узагальнений випадок повороту навколо довільної осі у просторі зустрічається доволі часто, наприклад у робототехніці, мультиплікації, моделюванні тощо. Йдучи за логікою попереднього обговорення, поворот навколо довільної осі у просторі виконується за допомогою переносу і простих поворотів навколо координатних осей. Оскільки метод повороту навколо координатної осі відомий, то основна ідея полягає в тому, щоб сумістити довільну вісь обертання з однією із координатних осей.

Припустимо, що довільна вісь у просторі проходить через точку $(x_0, \quad y_0, \quad z_0)$ з напрямним вектором $(c_x, \quad c_y, \quad c_z)$. Поворот навколо осі на деякий кут δ виконується за таким правилом:

– виконати перенесення так, щоб точка $(x_0, \quad y_0, \quad z_0)$ знаходилась на початку системи координат;

– виконати відповідні повороти так, щоб вісь обертання збігалась з віссю z (вибір осі довільний);

– виконати поворот на кут δ навколо осі z;

– виконати зворотне перетворення;

– виконати зворотне перенесення.

У загальному випадку для того, щоб довільна вісь, що проходить через початок координат, збігалась з однією із координатних осей, необхідно зробити два послідовних повороти навколо двох інших координатних осей.

Для суміщення довільної осі обертання з віссю z спочатку виконаємо поворот навколо осі x, а після цього навколо осі y. Щоб визначити кут повороту α навколо осі x, що використовується для переведення довільної осі у площину xz, зробимо проекцію на площину yz напрямного одиничного вектора цієї осі (рис.3.5, а).

T_r – матриця переміщення;

R_x – відповідна матриця повороту;

T_r^{-1} – матриця, зворотна до матриці переміщення.

Для того щоб здійснити декілька поворотів у локальній системі координат, паралельній осям глобальної системи координат, потрібно сумістити початки глобальної та локальної систем [13].

Обертання навколо довільної осі у просторі

Узагальнений випадок повороту навколо довільної осі у просторі зустрічається доволі часто, наприклад у робототехніці, мультиплікації, моделюванні тощо. Йдучи за логікою попереднього обговорення, поворот навколо довільної осі у просторі виконується за допомогою переносу і простих поворотів навколо координатних осей. Оскільки метод повороту навколо координатної осі відомий, то основна ідея полягає в тому, щоб сумістити довільну вісь обертання з однією із координатних осей.

Припустимо, що довільна вісь у просторі проходить через точку $(x_0, \quad y_0, \quad z_0)$ з напрямним вектором $(c_x, \quad c_y, \quad c_z)$. Поворот навколо осі на деякий кут δ виконується за таким правилом:

– виконати перенесення так, щоб точка $(x_0, \quad y_0, \quad z_0)$ знаходилась на початку системи координат;

– виконати відповідні повороти так, щоб вісь обертання збігалась з віссю z (вибір осі довільний);

– виконати поворот на кут δ навколо осі z;

– виконати зворотне перетворення;

– виконати зворотне перенесення.

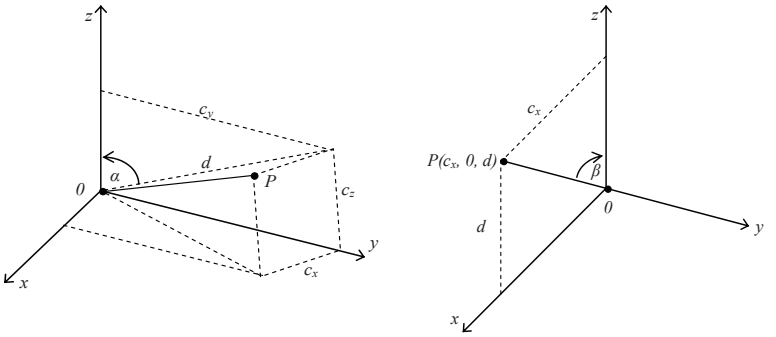
У загальному випадку для того, щоб довільна вісь, що проходить через початок координат, збігалась з однією із координатних осей, необхідно зробити два послідовних повороти навколо двох інших координатних осей.

Для суміщення довільної осі обертання з віссю z спочатку виконаємо поворот навколо осі x, а після цього навколо осі y. Щоб визначити кут повороту α навколо осі x, що використовується для переведення довільної осі у площину xz, зробимо проекцію на площину yz напрямного одиничного вектора цієї осі (рис.3.5, а).

Компоненти y та z спроектованого вектора дорівнюють c_y та c_z компонентам одиничного напрямного вектора осі обертання. З рис.3.5,а випливає, що

$$d = \sqrt{c_y^2 + c_z^2}, \tag{3.14}$$

$$\cos\alpha = \frac{c_z}{d}, \sin\alpha = \frac{c_y}{d}. \tag{3.15}$$



а б

Рис. 3.5. Повороти, необхідні для суміщення з віссю z одиничного вектора OP: а – поворот навколо x; б – поворот навколо y

Після переведення у площину xz за допомогою повороту навколо осі x z-компонента одиничного вектора дорівнює d, а x - компонентта дорівнює c_x, тобто x- компоненті напрямного вектора, як це показано на рис. 3.5, б. Довжина одиничного вектора дорівнює 1.

Таким чином, кут повороту β навколо осі y, необхідний для суміщення довільної осі з віссю z, дорівнює

$$\cos\beta = d, \sin\beta = c_x. \tag{3.16}$$

У цьому випадку повне перетворення можна подати у вигляді

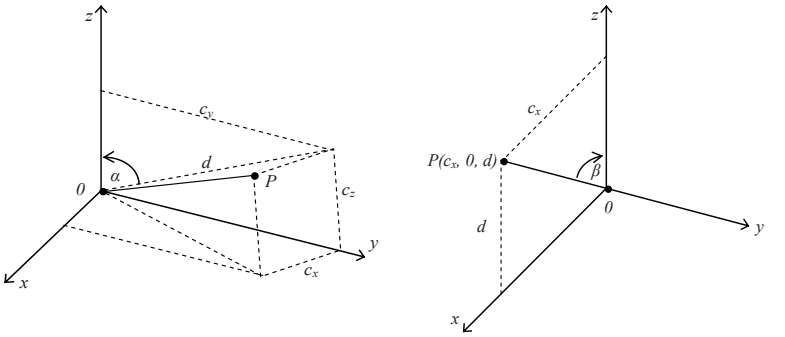
$$M = TR_xR_yR_\delta R_y^{-1}R_x^{-1}T^{-1}, \tag{3.17}$$

де матриця перенесення дорівнює

Компоненти y та z спроектованого вектора дорівнюють c_y та c_z компонентам одиничного напрямного вектора осі обертання. З рис.3.5,а випливає, що

$$d = \sqrt{c_y^2 + c_z^2}, \tag{3.14}$$

$$\cos\alpha = \frac{c_z}{d}, \sin\alpha = \frac{c_y}{d}. \tag{3.15}$$



а б

Рис. 3.5. Повороти, необхідні для суміщення з віссю z одиничного вектора OP: а – поворот навколо x; б – поворот навколо y

Після переведення у площину xz за допомогою повороту навколо осі x z-компонента одиничного вектора дорівнює d, а x - компонентта дорівнює c_x, тобто x- компоненті напрямного вектора, як це показано на рис. 3.5, б. Довжина одиничного вектора дорівнює 1.

Таким чином, кут повороту β навколо осі y, необхідний для суміщення довільної осі з віссю z, дорівнює

$$\cos\beta = d, \sin\beta = c_x. \tag{3.16}$$

У цьому випадку повне перетворення можна подати у вигляді

$$M = TR_xR_yR_\delta R_y^{-1}R_x^{-1}T^{-1}, \tag{3.17}$$

де матриця перенесення дорівнює

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x_0 & -y_0 & -z_0 & 1 \end{bmatrix}. \qquad (3.18)$$

Матриця перетворення повороту навколо осі x

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c_x/d & c_y/d & 0 \\ 0 & -c_y/d & c_z/d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (3.19)$$

та навколо осі y

$$R_y = \begin{bmatrix} \cos(-\beta) & 0 & -\sin(-\beta) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(-\beta) & 0 & \cos(-\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} d & 0 & c_x & 0 \\ 0 & 1 & 0 & 0 \\ -c_x & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \qquad (3.20)$$

Обертання навколо довільної осі задається матрицею повороту навколо осі z

$$R_\delta = \begin{bmatrix} -\cos\delta & \sin\delta & 0 & 0 \\ -\sin\delta & \cos\delta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \qquad (3.21)$$

На практиці кути β та α не обчислюються явним чином. Елементи матриць поворотів R_x та R_y у виразі (3.17) можна отримати з рівнянь (3.14) – (3.16) за рахунок виконання двох операцій ділення та витягування квадратного кореня. Хоч дані результати були розроблені для довільної осі у першому квадранті, їх можна застосовувати для всіх квадрантів.

Якщо компоненти напрямного вектора довільної осі невідомі, то, знаючи другу точку(x_1, y_1, z_1) на осі, їх можна визначити, нормалізувавши вектор, який з'єднує першу та другу точку. Більш точніше вектор осі з (x_0, y_0, z_0) у вектор (x_1, y_1, z_1) дорівнює

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x_0 & -y_0 & -z_0 & 1 \end{bmatrix}. \qquad (3.18)$$

Матриця перетворення повороту навколо осі x

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c_x/d & c_y/d & 0 \\ 0 & -c_y/d & c_z/d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (3.19)$$

та навколо осі y

$$R_y = \begin{bmatrix} \cos(-\beta) & 0 & -\sin(-\beta) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(-\beta) & 0 & \cos(-\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} d & 0 & c_x & 0 \\ 0 & 1 & 0 & 0 \\ -c_x & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \qquad (3.20)$$

Обертання навколо довільної осі задається матрицею повороту навколо осі z

$$R_\delta = \begin{bmatrix} -\cos\delta & \sin\delta & 0 & 0 \\ -\sin\delta & \cos\delta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \qquad (3.21)$$

На практиці кути β та α не обчислюються явним чином. Елементи матриць поворотів R_x та R_y у виразі (3.17) можна отримати з рівнянь (3.14) – (3.16) за рахунок виконання двох операцій ділення та витягування квадратного кореня. Хоч дані результати були розроблені для довільної осі у першому квадранті, їх можна застосовувати для всіх квадрантів.

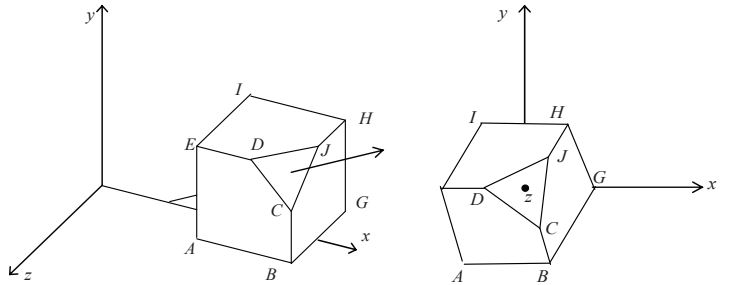
Якщо компоненти напрямного вектора довільної осі невідомі, то, знаючи другу точку(x_1, y_1, z_1) на осі, їх можна визначити, нормалізувавши вектор, який з'єднує першу та другу точку. Більш точніше вектор осі з (x_0, y_0, z_0) у вектор (x_1, y_1, z_1) дорівнює

$$V = [(x_1 - x_0)(y_1 - y_0)(z_1 - z_0)].$$

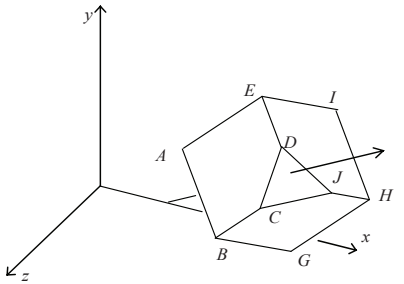
Нормалізація дає компоненти напрямного вектора:

$$[c_x \quad c_y \quad c_z] = \frac{[(x_1 - x_0)(y_1 - y_0)(z_1 - z_0)]}{\left[(x_1 - x_0)^2 + (y_1 - y_0)^2 + (z_1 - z_0)^2\right]^{\frac{1}{2}}}. \qquad (3.22)$$

Для більшого розуміння цього методу розглянемо приклад.



а б



в

Рис. 3.6. Поворот навколо довільної осі:
а – початкове положення; б – проміжний результат;
в – перетворений об’єкт

Приклад 3.1. Поворот навколо довільної осі.

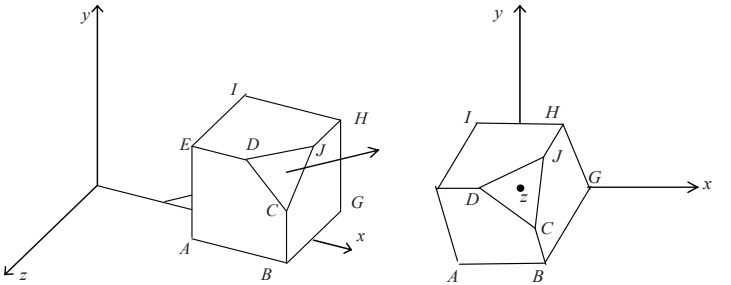
Розглянемо куб з одним відсіченим кутом (рис. 3.6). Координатні вектори вершин куба дорівнюють

$$V = [(x_1 - x_0)(y_1 - y_0)(z_1 - z_0)].$$

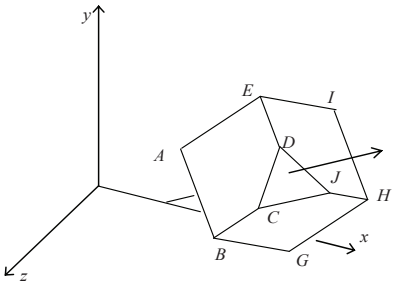
Нормалізація дає компоненти напрямного вектора:

$$[c_x \quad c_y \quad c_z] = \frac{[(x_1 - x_0)(y_1 - y_0)(z_1 - z_0)]}{\left[(x_1 - x_0)^2 + (y_1 - y_0)^2 + (z_1 - z_0)^2\right]^{\frac{1}{2}}}. \qquad (3.22)$$

Для більшого розуміння цього методу розглянемо приклад.



а б



в

Рис. 3.6. Поворот навколо довільної осі:
а – початкове положення; б – проміжний результат;
в – перетворений об’єкт

Приклад 3.1. Поворот навколо довільної осі.

Розглянемо куб з одним відсіченим кутом (рис. 3.6). Координатні вектори вершин куба дорівнюють

$$X = \begin{bmatrix} 2 & 1 & 2 & 1 \\ 3 & 1 & 2 & 1 \\ 3 & 1{,}5 & 2 & 1 \\ 2{,}5 & 2 & 2 & 1 \\ 2 & 2 & 2 & 1 \\ 2 & 1 & 1 & 1 \\ 3 & 1 & 1 & 1 \\ 3 & 2 & 1 & 1 \\ 2 & 2 & 1 & 1 \\ 3 & 2 & 1{,}5 & 1 \end{bmatrix} \begin{matrix} A \\ B \\ C \\ D \\ E \\ F \\ G \\ H \\ I \\ J \end{matrix}.$$

Куб необхідно повернути на -45° навколо осі, що проходить через точку F та протилежний кут на діагоналі. Вісь спрямована з точки F у протилежний кут та проходить через центр кутової грані.

Спочатку визначимо компоненти напрямного одиничного вектора осі обертання. Враховуючи, що кут, відрізаний трикутником CDJ, також лежить на осі, з виразу (3.22) випливає, що

$$\begin{bmatrix} c_x & c_y & c_z \end{bmatrix} = \frac{\left[(3-2)(2-1)(2-1)\right]}{\left[(3-2)^2 + (2-1)^2 + (2-1)^2\right]^{\frac{1}{2}}} =$$

$$= \begin{bmatrix} 1/\sqrt{3} & 1/\sqrt{3} & 1/\sqrt{3} \end{bmatrix}.$$

За допомогою решти рівнянь (3.14) – (3.21) отримаємо

$$d = \sqrt{2/3},\ \alpha = 45^\circ,\ \beta = 35{,}26^\circ;$$

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -2 & -1 & -1 & 1 \end{bmatrix},\ R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & -1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix};$$

$$X = \begin{bmatrix} 2 & 1 & 2 & 1 \\ 3 & 1 & 2 & 1 \\ 3 & 1{,}5 & 2 & 1 \\ 2{,}5 & 2 & 2 & 1 \\ 2 & 2 & 2 & 1 \\ 2 & 1 & 1 & 1 \\ 3 & 1 & 1 & 1 \\ 3 & 2 & 1 & 1 \\ 2 & 2 & 1 & 1 \\ 3 & 2 & 1{,}5 & 1 \end{bmatrix} \begin{matrix} A \\ B \\ C \\ D \\ E \\ F \\ G \\ H \\ I \\ J \end{matrix}.$$

Куб необхідно повернути на -45° навколо осі, що проходить через точку F та протилежний кут на діагоналі. Вісь спрямована з точки F у протилежний кут та проходить через центр кутової грані.

Спочатку визначимо компоненти напрямного одиничного вектора осі обертання. Враховуючи, що кут, відрізаний трикутником CDJ, також лежить на осі, з виразу (3.22) випливає, що

$$\begin{bmatrix} c_x & c_y & c_z \end{bmatrix} = \frac{\left[(3-2)(2-1)(2-1)\right]}{\left[(3-2)^2 + (2-1)^2 + (2-1)^2\right]^{\frac{1}{2}}} =$$

$$= \begin{bmatrix} 1/\sqrt{3} & 1/\sqrt{3} & 1/\sqrt{3} \end{bmatrix}.$$

За допомогою решти рівнянь (3.14) – (3.21) отримаємо

$$d = \sqrt{2/3},\ \alpha = 45^\circ,\ \beta = 35{,}26^\circ;$$

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -2 & -1 & -1 & 1 \end{bmatrix},\ R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & -1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix};$$

$$R_y = \begin{bmatrix} 2/\sqrt{6} & 0 & 1/\sqrt{3} & 0 \\ 0 & 1 & 0 & 0 \\ -1/\sqrt{3} & 0 & 2/\sqrt{6} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix};$$

$$M = TR_xR_y = \begin{bmatrix} 2/\sqrt{6} & 0 & 1/\sqrt{3} & 0 \\ -1/\sqrt{6} & 1/\sqrt{2} & 1/\sqrt{3} & 0 \\ -1/\sqrt{6} & -1/\sqrt{2} & 1/\sqrt{3} & 0 \\ -2/\sqrt{6} & 0 & -4/\sqrt{3} & 1 \end{bmatrix}.$$

Проміжний результат перетворення зображено на рис. 3.6, б. Відмітимо, що точка F дорівнює $(0,0,0)$. Обертання навколо довільної осі тепер еквівалентно обертанню навколо осі z. У зв'язку з цим

$$R_\delta = \begin{bmatrix} \sqrt{2}/2 & -\sqrt{2}/2 & 0 & 0 \\ \sqrt{2}/2 & \sqrt{2}/2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Перетворений об'єкт повертається у свій вихідний стан у просторі за допомогою перетворення

$$M^{-1} = R_y^{-1}R_x^{-1}T^{-1} = \begin{bmatrix} 2/\sqrt{6} & -1/\sqrt{6} & -1/\sqrt{6} & 0 \\ 0 & 1/\sqrt{2} & -1/\sqrt{2} & 0 \\ 1/\sqrt{3} & 1/\sqrt{3} & 1/\sqrt{3} & 0 \\ 2 & 1 & 1 & 1 \end{bmatrix}.$$

Цей результат можна отримати або оберненням та об'єднанням матриць, що входять до складу матриці M, або шляхом обернення самої цієї матриці.

$$R_y = \begin{bmatrix} 2/\sqrt{6} & 0 & 1/\sqrt{3} & 0 \\ 0 & 1 & 0 & 0 \\ -1/\sqrt{3} & 0 & 2/\sqrt{6} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix};$$

$$M = TR_xR_y = \begin{bmatrix} 2/\sqrt{6} & 0 & 1/\sqrt{3} & 0 \\ -1/\sqrt{6} & 1/\sqrt{2} & 1/\sqrt{3} & 0 \\ -1/\sqrt{6} & -1/\sqrt{2} & 1/\sqrt{3} & 0 \\ -2/\sqrt{6} & 0 & -4/\sqrt{3} & 1 \end{bmatrix}.$$

Проміжний результат перетворення зображено на рис. 3.6, б. Відмітимо, що точка F дорівнює $(0,0,0)$. Обертання навколо довільної осі тепер еквівалентно обертанню навколо осі z. У зв'язку з цим

$$R_\delta = \begin{bmatrix} \sqrt{2}/2 & -\sqrt{2}/2 & 0 & 0 \\ \sqrt{2}/2 & \sqrt{2}/2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Перетворений об'єкт повертається у свій вихідний стан у просторі за допомогою перетворення

$$M^{-1} = R_y^{-1}R_x^{-1}T^{-1} = \begin{bmatrix} 2/\sqrt{6} & -1/\sqrt{6} & -1/\sqrt{6} & 0 \\ 0 & 1/\sqrt{2} & -1/\sqrt{2} & 0 \\ 1/\sqrt{3} & 1/\sqrt{3} & 1/\sqrt{3} & 0 \\ 2 & 1 & 1 & 1 \end{bmatrix}.$$

Цей результат можна отримати або оберненням та об'єднанням матриць, що входять до складу матриці M, або шляхом обернення самої цієї матриці.

3.2. Показ тривимірних зображень на двовимірній площині

Розглянемо проблему показу тривимірних зображень на двовимірній площині. Для цього необхідно мати певні математичні моделі. У цих моделях повинні враховуватися різні фактори, що впливають на візуальне сприйняття людиною реальних образів. Спосіб переходу від тривимірних об'єктів до їхніх зображень на площині будемо називати проекцією. Далі розглядаються різні види проекцій.

Афінна та перспективна (нарисна) геометрія

Афінне перетворення є комбінацією лінійних перетворень, наприклад повороту та наступного перенесення. Для афінного перетворення останній стовпчик в узагальненій матриці дорівнює $[0 \quad 0 \quad 0 \quad 1]^T$. Афінні перетворення створюють корисну підмножину білінійних перетворень, оскільки добуток двох афінних перетворень також є афінним перетворенням.

Рис. 3.7. Ієрархія плоских геометричних проекцій

3.2. Показ тривимірних зображень на двовимірній площині

Розглянемо проблему показу тривимірних зображень на двовимірній площині. Для цього необхідно мати певні математичні моделі. У цих моделях повинні враховуватися різні фактори, що впливають на візуальне сприйняття людиною реальних образів. Спосіб переходу від тривимірних об'єктів до їхніх зображень на площині будемо називати проекцією. Далі розглядаються різні види проекцій.

Афінна та перспективна (нарисна) геометрія

Афінне перетворення є комбінацією лінійних перетворень, наприклад повороту та наступного перенесення. Для афінного перетворення останній стовпчик в узагальненій матриці дорівнює $[0 \quad 0 \quad 0 \quad 1]^T$. Афінні перетворення створюють корисну підмножину білінійних перетворень, оскільки добуток двох афінних перетворень також є афінним перетворенням.

Рис. 3.7. Ієрархія плоских геометричних проекцій

Ця властивість дозволяє скомбінувати загальне перетворення множини точок відносно довільної системи координат при зберіганні значення одиниці для однорідної координати h.

Хоч евклідова геометрія і є основою для графічних методів креслення, архітектори доволі часто використовують перспективні види для створення більш реалістичних зображень.

На рис. 3.7 зображена ієрархія плоских геометричних проекцій. Плоскі геометричні проекції об'єктів створюються за рахунок перетинання прямих, які називають проекторами, з площиною, яку називають площиною проекції. ***Проектори*** – це прямі, які проходять через довільну точку, що називається центром проекції, і кожну точку об'єкта. Якщо центр проекції розміщений у кінцевій точці тривимірного простору, створюється ***перспективна проекція***. Якщо центр розміщений у нескінченності, то всі проектори паралельні і результатом є ***паралельна проекція***.

Найпростішою з паралельних проекцій є ***ортографічна проекція***, що використовується в інженерних кресленнях. У цьому випадку точно зображуються правильні (істинні) розміри та форма однієї плоскої грані об'єкта.

Одна ортографічна проекція не може дати уявлення про загальну тривимірну форму об'єкта. Це обмеження можна подолати за допомогою ***аксонометричних проекцій***. Аксонометрична проекція створюється маніпулюванням об'єктом за допомогою поворотів та переміщень таким чином, щоб було видно три сусідніх грані. Результат потім проецирується з центром проекції, розміщеним у нескінченності, на одну із координатних осей, зазвичай на площину $z = 0$.

Для того щоб побачити на площині монітора тривимірне зображення, потрібно вміти задати спосіб відображення тривимірних точок у двовимірні. Зробити це можна по-різному. У загальному випадку проекції перетворять точки, задані в системі координат розмірністю n, у точки системи координат розмірністю меншою, чим n. У нашому випадку точки тривимірного простору перетворяться в точки двовимірного простору. Проекції будуються за допомогою променів, що проектують, або проекторів, які виходять із точки, що називається центром проекції. Проектори проходять через площину, що назива-

Ця властивість дозволяє скомбінувати загальне перетворення множини точок відносно довільної системи координат при зберіганні значення одиниці для однорідної координати h.

Хоч евклідова геометрія і є основою для графічних методів креслення, архітектори доволі часто використовують перспективні види для створення більш реалістичних зображень.

На рис. 3.7 зображена ієрархія плоских геометричних проекцій. Плоскі геометричні проекції об'єктів створюються за рахунок перетинання прямих, які називають проекторами, з площиною, яку називають площиною проекції. ***Проектори*** – це прямі, які проходять через довільну точку, що називається центром проекції, і кожну точку об'єкта. Якщо центр проекції розміщений у кінцевій точці тривимірного простору, створюється ***перспективна проекція***. Якщо центр розміщений у нескінченності, то всі проектори паралельні і результатом є ***паралельна проекція***.

Найпростішою з паралельних проекцій є ***ортографічна проекція***, що використовується в інженерних кресленнях. У цьому випадку точно зображуються правильні (істинні) розміри та форма однієї плоскої грані об'єкта.

Одна ортографічна проекція не може дати уявлення про загальну тривимірну форму об'єкта. Це обмеження можна подолати за допомогою ***аксонометричних проекцій***. Аксонометрична проекція створюється маніпулюванням об'єктом за допомогою поворотів та переміщень таким чином, щоб було видно три сусідніх грані. Результат потім проецирується з центром проекції, розміщеним у нескінченності, на одну із координатних осей, зазвичай на площину $z = 0$.

Для того щоб побачити на площині монітора тривимірне зображення, потрібно вміти задати спосіб відображення тривимірних точок у двовимірні. Зробити це можна по-різному. У загальному випадку проекції перетворять точки, задані в системі координат розмірністю n, у точки системи координат розмірністю меншою, чим n. У нашому випадку точки тривимірного простору перетворяться в точки двовимірного простору. Проекції будуються за допомогою променів, що проектують, або проекторів, які виходять із точки, що називається центром проекції. Проектори проходять через площину, що назива-

ється проекційною або картинною площиною, та потім проходять через кожну точку тривимірного об'єкта й утворять тим самим проекцію.

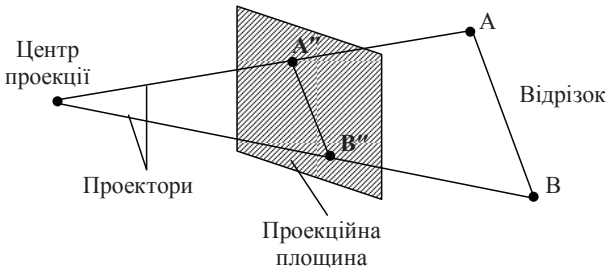


Рис. 3.8. Центральна проекція

Тип проектування на плоску, а не скривлену поверхню, де як проектори використовуються прямі, а не викривлені лінії, називається плоскою геометричною проекцією. Плоскі геометричні проекції діляться на два види: центральні й паралельні. Якщо центр проекції перебуває на кінцевій відстані від проекційної площини, то проекція - центральна. Якщо ж центр проекції вилучений на нескінченність, то проекція - паралельна.

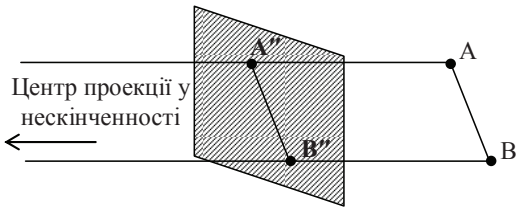


Рис. 3.9. Паралельна проекція

Точкою сходження називається точка перетинання центральних проекцій будь-якої сукупності паралельних прямих, які не паралельні проекційній площині. Існує нескінченна множина точок сходження. Точка сходження називається головною, якщо сукупність прямих паралельна одній з координатних осей. Залежно від того, скільки координатних осей перетинає проекційну площину, розрізняють одно-, дво- і триточкові проекції.

ється проекційною або картинною площиною, та потім проходять через кожну точку тривимірного об'єкта й утворять тим самим проекцію.

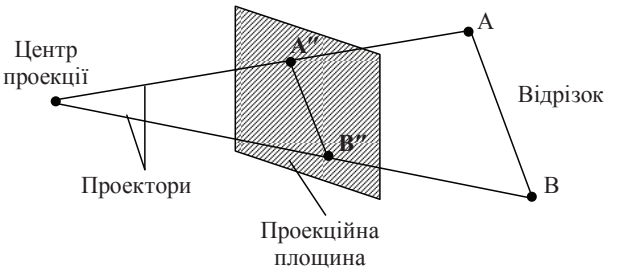


Рис. 3.8. Центральна проекція

Тип проектування на плоску, а не скривлену поверхню, де як проектори використовуються прямі, а не викривлені лінії, називається плоскою геометричною проекцією. Плоскі геометричні проекції діляться на два види: центральні й паралельні. Якщо центр проекції перебуває на кінцевій відстані від проекційної площини, то проекція - центральна. Якщо ж центр проекції вилучений на нескінченність, то проекція - паралельна.

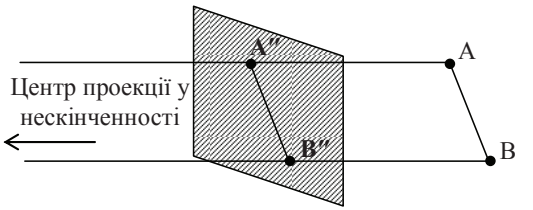


Рис. 3.9. Паралельна проекція

Точкою сходження називається точка перетинання центральних проекцій будь-якої сукупності паралельних прямих, які не паралельні проекційній площині. Існує нескінченна множина точок сходження. Точка сходження називається головною, якщо сукупність прямих паралельна одній з координатних осей. Залежно від того, скільки координатних осей перетинає проекційну площину, розрізняють одно-, дво- і триточкові проекції.

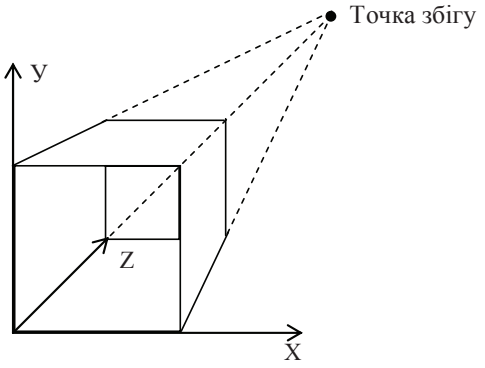


Рис. 3.10. Одноточкова проекція

Найпростішою є паралельна прямокутна проекція. У ній спільно зображаються види зверху, спереду й збоку. Ці проекції часто використовуються в кресленні. Залежно від співвідношення між напрямками проектування й нормаллю до проекційної площини паралельні проекції розділяються на ортографічні або ортогональні, в яких ці напрямки збігаються, і косокутні, в яких вони не збігаються. Залежно від положення осей системи координат об'єкта щодо проекційної площини ортографічні проекції діляться на аксонометричні й ізометричні. В ізометричних проекціях осі системи координат становлять однакові кути із проекційною площиною. В аксонометричних проекціях ці кути різні. Центральна перспективна проекція приводить до візуального ефекту, подібному тому, який дає зорова система людини. При цьому спостерігається ефект перспективного укорочування, коли розмір проекції об'єкта змінюється обернено пропорційно відстані від центра проекції до об'єкта. У паралельних проекціях відсутнє перспективне укорочування, за рахунок чого зображення виходить менш реалістичним і паралельні прямі завжди залишаються паралельними.

Розглянемо більш детально центральну перспективну проекцію з математичної точки зору. Для одержання формул центральної перспективної проекції розташуємо осі системи координат, проекційну площину й центр проекції, як показано на рис. 3.11.

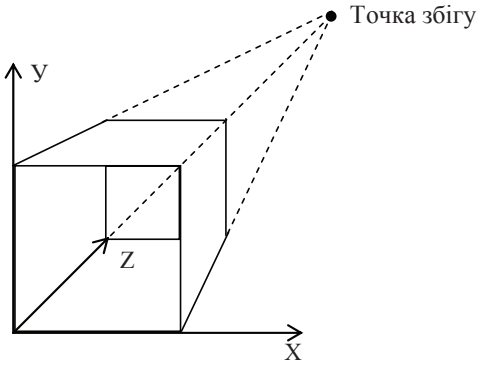


Рис. 3.10. Одноточкова проекція

Найпростішою є паралельна прямокутна проекція. У ній спільно зображаються види зверху, спереду й збоку. Ці проекції часто використовуються в кресленні. Залежно від співвідношення між напрямками проектування й нормаллю до проекційної площини паралельні проекції розділяються на ортографічні або ортогональні, в яких ці напрямки збігаються, і косокутні, в яких вони не збігаються. Залежно від положення осей системи координат об'єкта щодо проекційної площини ортографічні проекції діляться на аксонометричні й ізометричні. В ізометричних проекціях осі системи координат становлять однакові кути із проекційною площиною. В аксонометричних проекціях ці кути різні. Центральна перспективна проекція приводить до візуального ефекту, подібному тому, який дає зорова система людини. При цьому спостерігається ефект перспективного укорочування, коли розмір проекції об'єкта змінюється обернено пропорційно відстані від центра проекції до об'єкта. У паралельних проекціях відсутнє перспективне укорочування, за рахунок чого зображення виходить менш реалістичним і паралельні прямі завжди залишаються паралельними.

Розглянемо більш детально центральну перспективну проекцію з математичної точки зору. Для одержання формул центральної перспективної проекції розташуємо осі системи координат, проекційну площину й центр проекції, як показано на рис. 3.11.

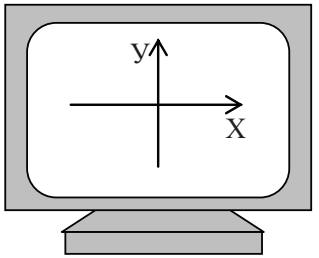


Рис. 3.11. Розташування осей координат на екрані

Будемо імітувати на екрані те, що начебто реально перебуває в просторі за ним. Помітимо, що вийшла лівостороння система координат. Будемо вважати, що площина екрана монітора збігається із проекційною площиною. Перш ніж переходити до обчислень варто зробити одне важливе зауваження. Оскільки поверхня будь-якого тривимірного об'єкта містить нескінченне число точок, то необхідно задати спосіб опису поверхні об'єкта кінцевим числом точок для подання в комп'ютері. А саме, будемо використовувати лінійну апроксимацію об'єктів у тривимірному просторі за допомогою відрізків прямих і плоских багатокутників. При цьому відрізки прямих після перспективного перетворення переходять у відрізки прямих на проекційній площині. Доказ цього досить простий і тут не приводиться. Ця важлива властивість центральної перспективи дозволяє проектувати, тобто робити обчислення, тільки для кінцевих точок відрізків, а потім з'єднувати проекції точок лініями вже на проекційній площині.

Точка A проектується на екран як A'. Відстань від спостерігача до проекційної площини дорівнює k. Необхідно визначити координати точки A' на екрані. Позначимо їх x_e і y_e. З подібності трикутників A_yA_zN і y_eON знаходимо, що

$$\frac{y}{z+k}=\frac{y_e}{k},\Rightarrow y_e=\frac{ky}{z+k}, \qquad (3.23)$$

аналогічно для x: $x_e=\frac{kx}{z+k}$.

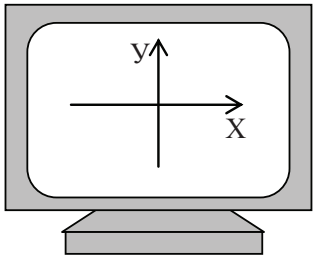


Рис. 3.11. Розташування осей координат на екрані

Будемо імітувати на екрані те, що начебто реально перебуває в просторі за ним. Помітимо, що вийшла лівостороння система координат. Будемо вважати, що площина екрана монітора збігається із проекційною площиною. Перш ніж переходити до обчислень варто зробити одне важливе зауваження. Оскільки поверхня будь-якого тривимірного об'єкта містить нескінченне число точок, то необхідно задати спосіб опису поверхні об'єкта кінцевим числом точок для подання в комп'ютері. А саме, будемо використовувати лінійну апроксимацію об'єктів у тривимірному просторі за допомогою відрізків прямих і плоских багатокутників. При цьому відрізки прямих після перспективного перетворення переходять у відрізки прямих на проекційній площині. Доказ цього досить простий і тут не приводиться. Ця важлива властивість центральної перспективи дозволяє проектувати, тобто робити обчислення, тільки для кінцевих точок відрізків, а потім з'єднувати проекції точок лініями вже на проекційній площині.

Точка A проектується на екран як A'. Відстань від спостерігача до проекційної площини дорівнює k. Необхідно визначити координати точки A' на екрані. Позначимо їх x_e і y_e. З подібності трикутників A_yA_zN і y_eON знаходимо, що

$$\frac{y}{z+k}=\frac{y_e}{k},\Rightarrow y_e=\frac{ky}{z+k}, \qquad (3.23)$$

аналогічно для x: $x_e=\frac{kx}{z+k}$.

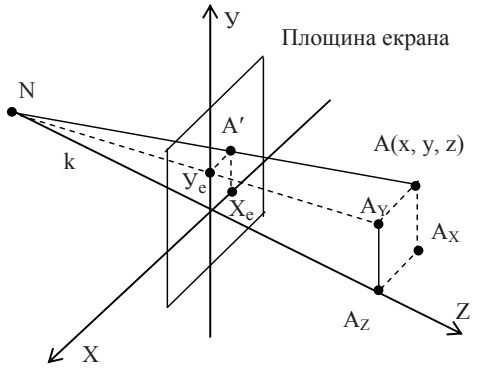


Рис. 3.12. Виведення формул центральної перспективної проекції

Нагадаємо, що k – це відстань, а спостерігач перебуває в точці $N=(0,0,-k)$.

Якщо точку спостереження помістити в початок координат, а проекційну площину на відстань a, як показано на рис.3.13, то формули для x_e й y_e приймуть вигляд:

$$x_e = \frac{kx}{z}, \quad y_e = \frac{ky}{z}. \tag{3.24}$$

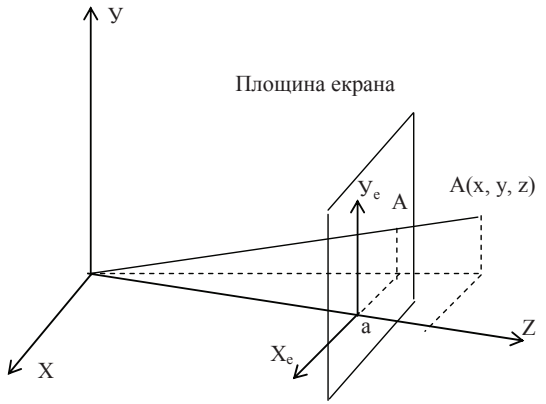


Рис. 3.13. Інший спосіб обчислення координат точок у центральній перспективній проекції

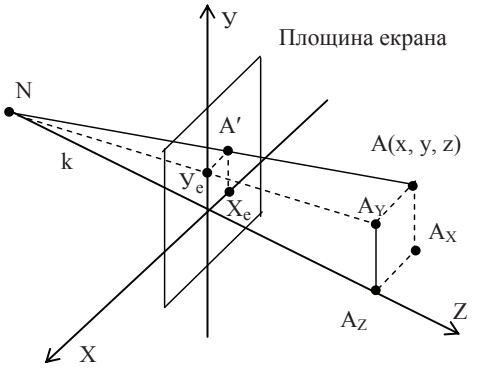


Рис. 3.12. Виведення формул центральної перспективної проекції

Нагадаємо, що k – це відстань, а спостерігач перебуває в точці $N=(0,0,-k)$.

Якщо точку спостереження помістити в початок координат, а проекційну площину на відстань a, як показано на рис.3.13, то формули для x_e й y_e приймуть вигляд:

$$x_e = \frac{kx}{z}, \quad y_e = \frac{ky}{z}. \tag{3.24}$$

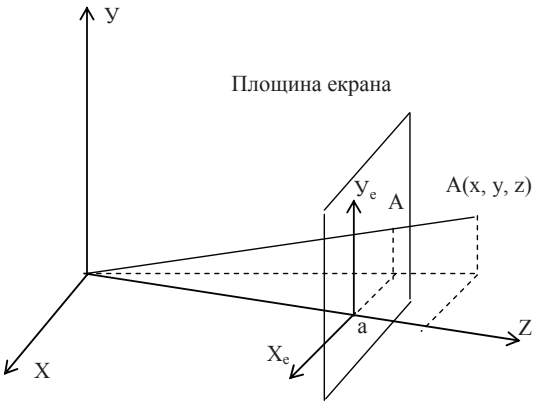


Рис. 3.13. Інший спосіб обчислення координат точок у центральній перспективній проекції

Вираз (3.23) більш зручніший і за необхідності дає можливість простіше наближати або віддаляти спостерігача від проекційної площини. Вираз (3.24) вимагає менше часу для обчислень за рахунок відсутності операції додавання.

Розглянемо далі деякі фактори, що впливають на сприйняття людиною тривимірності. Одним із простих способів подання тривимірних об'єктів є так звані «відрізкові» зображення. Криві лінії при цьому апроксимуються відрізками прямих. Це найбільш швидкий і простий спосіб зображення.

Для посилення ефекту тривимірної глибини в зображеннях об'єктів, що подаються відрізками, видаляють невидимі лінії. Лінії або їхні частини, закриті поверхнями об'єкта, не зображуються. Для цього застосовується спеціальний алгоритм, що вимагає вже більших обчислень. Передача глибини може здійснюватися зміною рівня яскравості. Об'єкти, які перебувають ближче до спостерігача, зображуються яскравіше, ніж ті, які розташовані далі від нього. Рух об'єктів також дає додатковий ефект глибини.

Наприклад, обертання об'єктів навколо вертикальної осі дозволяє відрізнити точки, що перебувають на різній відстані від осі, за рахунок розходження лінійної швидкості обертання точок. Це так званий кінетичний або динамічний ефект глибини.

Більш тонко тривимірність об'єктів може бути представлена за рахунок розходжень відбивних здатностей поверхонь, їхнього рельєфу й текстури, а також розрахунку тіней, що відкидають поверхні об'єкта.

Одним з рідко використовуваних, але найбільш ефективних способів досягнення ефекту тривимірності є стереоскопія. При цьому окремо для правого й лівого ока спостерігача формуються зображення, які незначно відрізняються одне від одного, подібно тому, як це відбувається в реальності. Це викликає так званий бінокулярний ефект, який полягає в тому, що наш мозок зливає два окремих образи в один, який інтерпретується як тривимірний. Ці два роздільних зображення називаються стереопарою.

Технічно цей метод реалізується, наприклад, за допомогою окулярів зі спеціальним поляризованим склом. На екран монітора по черзі виводяться зображення для лівого і правого ока. А скло окулярів стає

Вираз (3.23) більш зручніший і за необхідності дає можливість простіше наближати або віддаляти спостерігача від проекційної площини. Вираз (3.24) вимагає менше часу для обчислень за рахунок відсутності операції додавання.

Розглянемо далі деякі фактори, що впливають на сприйняття людиною тривимірності. Одним із простих способів подання тривимірних об'єктів є так звані «відрізкові» зображення. Криві лінії при цьому апроксимуються відрізками прямих. Це найбільш швидкий і простий спосіб зображення.

Для посилення ефекту тривимірної глибини в зображеннях об'єктів, що подаються відрізками, видаляють невидимі лінії. Лінії або їхні частини, закриті поверхнями об'єкта, не зображуються. Для цього застосовується спеціальний алгоритм, що вимагає вже більших обчислень. Передача глибини може здійснюватися зміною рівня яскравості. Об'єкти, які перебувають ближче до спостерігача, зображуються яскравіше, ніж ті, які розташовані далі від нього. Рух об'єктів також дає додатковий ефект глибини.

Наприклад, обертання об'єктів навколо вертикальної осі дозволяє відрізнити точки, що перебувають на різній відстані від осі, за рахунок розходження лінійної швидкості обертання точок. Це так званий кінетичний або динамічний ефект глибини.

Більш тонко тривимірність об'єктів може бути представлена за рахунок розходжень відбивних здатностей поверхонь, їхнього рельєфу й текстури, а також розрахунку тіней, що відкидають поверхні об'єкта.

Одним з рідко використовуваних, але найбільш ефективних способів досягнення ефекту тривимірності є стереоскопія. При цьому окремо для правого й лівого ока спостерігача формуються зображення, які незначно відрізняються одне від одного, подібно тому, як це відбувається в реальності. Це викликає так званий бінокулярний ефект, який полягає в тому, що наш мозок зливає два окремих образи в один, який інтерпретується як тривимірний. Ці два роздільних зображення називаються стереопарою.

Технічно цей метод реалізується, наприклад, за допомогою окулярів зі спеціальним поляризованим склом. На екран монітора по черзі виводяться зображення для лівого і правого ока. А скло окулярів стає

по черзі, відповідно, прозорим або непрозорим. При досить частій зміні зображень зміни станів прозорості й непрозорості не відчувається. Оскільки при зміні положення голови центр проекції залишається на місці, то створюється псевдотривимірний ефект. Синхронізація зміни кадрів на екрані й поляризації лінз окулярів відбувається за допомогою спеціальних датчиків, розташованих на окулярах і моніторі.

Рис. 3.14. Бінокулярний ефект, стереоскопія

3.3. Перетворення, пов'язані із системою координат

Необхідно навчитися управляти зображенням на екрані, вносити зміни в його положення, форму, орієнтацію, розмір. Для цих цілей існують спеціальні геометричні перетворення, які дозволяють змінювати ці характеристики об'єктів у просторі. Представимо задачу створення комп'ютерного імітатора польотів на військовому літаку. Об'єкти на землі, як і сам літак, змінюють своє положення: обертається антена локатора, рухається танк. При цьому спостерігач бачить цю картину з певної точки в просторі в обраному напрямку. Необхідно описати ці складні перетворення математично.

Уведемо три види систем координат. Перша з них – світова система координат – задається осями $X_M Y_M Z_M$. Ми розміщуємо її в деякій точці, і вона залишається нерухомою завжди. Друга – система координат спостерігача. Цю систему назвемо $X_N Y_N Z_N$. Вона визначає положення спостерігача в просторі й задає напрямок погляду. І третя – система координат об'єкта. У нашому випадку їх дві: системи координат локатора й система координат танка. Ці системи також можуть переміщатися й змінювати своє положення в просторі щодо світової системи координат.

Координати точок об'єктів задаються в системах координат об'єктів, кожна з яких, у свою чергу, прив'язана до світової системи координат. Система координат спостерігача також переміщається щодо

по черзі, відповідно, прозорим або непрозорим. При досить частій зміні зображень зміни станів прозорості й непрозорості не відчувається. Оскільки при зміні положення голови центр проекції залишається на місці, то створюється псевдотривимірний ефект. Синхронізація зміни кадрів на екрані й поляризації лінз окулярів відбувається за допомогою спеціальних датчиків, розташованих на окулярах і моніторі.

Рис. 3.14. Бінокулярний ефект, стереоскопія

3.3. Перетворення, пов'язані із системою координат

Необхідно навчитися управляти зображенням на екрані, вносити зміни в його положення, форму, орієнтацію, розмір. Для цих цілей існують спеціальні геометричні перетворення, які дозволяють змінювати ці характеристики об'єктів у просторі. Представимо задачу створення комп'ютерного імітатора польотів на військовому літаку. Об'єкти на землі, як і сам літак, змінюють своє положення: обертається антена локатора, рухається танк. При цьому спостерігач бачить цю картину з певної точки в просторі в обраному напрямку. Необхідно описати ці складні перетворення математично.

Уведемо три види систем координат. Перша з них – світова система координат – задається осями $X_M Y_M Z_M$. Ми розміщуємо її в деякій точці, і вона залишається нерухомою завжди. Друга – система координат спостерігача. Цю систему назвемо $X_N Y_N Z_N$. Вона визначає положення спостерігача в просторі й задає напрямок погляду. І третя – система координат об'єкта. У нашому випадку їх дві: системи координат локатора й система координат танка. Ці системи також можуть переміщатися й змінювати своє положення в просторі щодо світової системи координат.

Координати точок об'єктів задаються в системах координат об'єктів, кожна з яких, у свою чергу, прив'язана до світової системи координат. Система координат спостерігача також переміщається щодо

світової системи координат. Тепер стає зрозуміло, що для того, щоб побачити тривимірний об'єкт на екрані монітора, *треба реалізувати такі кроки*.

Крок 1. Перетворити координати об'єкта, задані у власній системі координат, у світові координати.

Крок 2. Перетворити координати об'єкта, задані вже у світовій системі координат, у систему координат спостерігача.

Крок 3. Спроектувати отримані координати на проекційну площину в системі координат спостерігача.

Відзначимо певну подвійність вражень, що виникають при взаємних переміщеннях систем координат щодо одна одної. Уявимо собі, що ми спостерігаємо куб у просторі. Нехай тепер цей куб почне обертатися навколо, наприклад, вертикальної осі. Ми побачимо, що куб обертається. Але той же самий ефект ми одержимо, якщо самі почнемо облітати навколо куба й розглядати його з різних сторін. Візуальний ефект залишається тим же самим, хоча в першому випадку наша система координат залишається нерухомою, а в другому - обертається по орбіті. Цей ефект можна використати при виведенні формул руху в просторі.

3.4. Алгоритми растрової графіки

Растром називається прямокутна сітка точок, що формують зображення на екрані комп'ютера. Кожна точка растра характеризується двома параметрами: своїм положенням на екрані і своїм кольором, якщо монітор кольоровий, або ступенем яскравості, якщо монітор чорно-білий. Оскільки растрові зображення складаються із множини дискретних точок, то для роботи з ними необхідні спеціальні алгоритми. Малювання відрізка прямої лінії – одна з найпростіших задач растрової графіки. Зміст її полягає в обчисленні координат пікселів, що перебувають поблизу безперервних відрізків, розміщених на двовимірній растровій сітці.

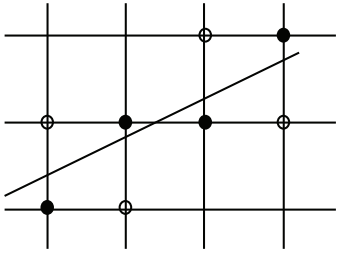


Рис. 3.15. Растеризація відрізка прямої лінії

світової системи координат. Тепер стає зрозуміло, що для того, щоб побачити тривимірний об'єкт на екрані монітора, *треба реалізувати такі кроки*.

Крок 1. Перетворити координати об'єкта, задані у власній системі координат, у світові координати.

Крок 2. Перетворити координати об'єкта, задані вже у світовій системі координат, у систему координат спостерігача.

Крок 3. Спроектувати отримані координати на проекційну площину в системі координат спостерігача.

Відзначимо певну подвійність вражень, що виникають при взаємних переміщеннях систем координат щодо одна одної. Уявимо собі, що ми спостерігаємо куб у просторі. Нехай тепер цей куб почне обертатися навколо, наприклад, вертикальної осі. Ми побачимо, що куб обертається. Але той же самий ефект ми одержимо, якщо самі почнемо облітати навколо куба й розглядати його з різних сторін. Візуальний ефект залишається тим же самим, хоча в першому випадку наша система координат залишається нерухомою, а в другому - обертається по орбіті. Цей ефект можна використати при виведенні формул руху в просторі.

3.4. Алгоритми растрової графіки

Растром називається прямокутна сітка точок, що формують зображення на екрані комп'ютера. Кожна точка растра характеризується двома параметрами: своїм положенням на екрані і своїм кольором, якщо монітор кольоровий, або ступенем яскравості, якщо монітор чорно-білий. Оскільки растрові зображення складаються із множини дискретних точок, то для роботи з ними необхідні спеціальні алгоритми. Малювання відрізка прямої лінії – одна з найпростіших задач растрової графіки. Зміст її полягає в обчисленні координат пікселів, що перебувають поблизу безперервних відрізків, розміщених на двовимірній растровій сітці.

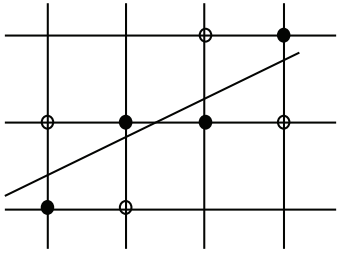


Рис. 3.15. Растеризація відрізка прямої лінії

119

Термін “піксел” утворений від англійського pixel (picture element– елемент зображення) – тобто точка на екрані. Будемо вважати, що пікселі мають цілочисельні координати. На перший погляд здається, що ця задача має просте рішення. Нехай кінцеві точки відрізка мають цілочисельні координати і рівняння прямої, що містить відрізок: $y=kx+b$. Не порушуючи спільності, будемо також уважати, що тангенс кута нахилу прямої лежить у межах від 0 до 1. Тоді для зображення відрізка на растрі досить для всіх цілих x, що належать відрізку, виводити на екран точки з координатами $(x, Round(y))$. Однак у цьому методі присутня операція множення kx. Хотілося б мати алгоритм без частого використання операції множення дійсних чисел. Позбутися від операції множення можна в такий спосіб. Оскільки $k=\Delta y/\Delta x$, то один крок по цілочисельній сітці на осі x буде відповідати $\Delta x=1$. Звідси одержуємо, що y буде збільшуватися на величину k. Ітераційна послідовність виглядає в такий спосіб:

$$x_{i+1}=x_i+1,\ y_{i+1}=y_i+k\ .$$

Коли $k>1$, то крок по x буде призводити до кроку по $y>1$, тому x і y варто поміняти місцями, надаючи y одиничний приріст, а x буде збільшуватися на $\Delta x=\Delta y/k=1/k$ одиниць. Цей алгоритм не обходиться без операцій з дійсними числами. Найбільш витончене розв’язання задачі растрового розгорнення відрізків прямих було знайдено Брезенхемом. В його алгоритмі взагалі не використовуються операції з дійсними числами, у тому числі операції множення й ділення.

Для виведення формул алгоритму Брезенхема розглянемо рис. 3.16.

Нехай початок відрізка має координати (x_1,x_2), а кінець (x_2,y_2). Позначимо $dx=(x_2-x_1)$, $dy=(y_2-y_1)$. Не порушуючи спільності, будемо вважати, що початок відрізка збігається з початком координат, і пряма має вигляд $y=\frac{dx}{dy}x$, де $\frac{dx}{dy}\in[0,1]$. Вважаємо що початкова точка

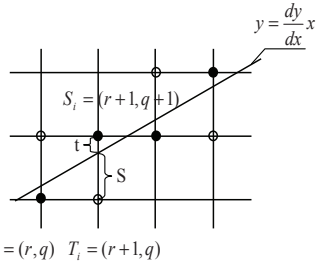


Рис. 3.16. Побудова відрізків прямих за методом Брезенхема

Термін “піксел” утворений від англійського pixel (picture element– елемент зображення) – тобто точка на екрані. Будемо вважати, що пікселі мають цілочисельні координати. На перший погляд здається, що ця задача має просте рішення. Нехай кінцеві точки відрізка мають цілочисельні координати і рівняння прямої, що містить відрізок: $y=kx+b$. Не порушуючи спільності, будемо також уважати, що тангенс кута нахилу прямої лежить у межах від 0 до 1. Тоді для зображення відрізка на растрі досить для всіх цілих x, що належать відрізку, виводити на екран точки з координатами $(x, Round(y))$. Однак у цьому методі присутня операція множення kx. Хотілося б мати алгоритм без частого використання операції множення дійсних чисел. Позбутися від операції множення можна в такий спосіб. Оскільки $k=\Delta y/\Delta x$, то один крок по цілочисельній сітці на осі x буде відповідати $\Delta x=1$. Звідси одержуємо, що y буде збільшуватися на величину k. Ітераційна послідовність виглядає в такий спосіб:

$$x_{i+1}=x_i+1,\ y_{i+1}=y_i+k\ .$$

Коли $k>1$, то крок по x буде призводити до кроку по $y>1$, тому x і y варто поміняти місцями, надаючи y одиничний приріст, а x буде збільшуватися на $\Delta x=\Delta y/k=1/k$ одиниць. Цей алгоритм не обходиться без операцій з дійсними числами. Найбільш витончене розв’язання задачі растрового розгорнення відрізків прямих було знайдено Брезенхемом. В його алгоритмі взагалі не використовуються операції з дійсними числами, у тому числі операції множення й ділення.

Для виведення формул алгоритму Брезенхема розглянемо рис. 3.16.

Нехай початок відрізка має координати (x_1,x_2), а кінець (x_2,y_2). Позначимо $dx=(x_2-x_1)$, $dy=(y_2-y_1)$. Не порушуючи спільності, будемо вважати, що початок відрізка збігається з початком координат, і пряма має вигляд $y=\frac{dx}{dy}x$, де $\frac{dx}{dy}\in[0,1]$. Вважаємо що початкова точка

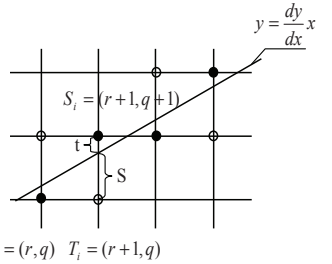


Рис. 3.16. Побудова відрізків прямих за методом Брезенхема

120

перебуває ліворуч. Нехай на *(i-1)*-му кроці поточною точкою відрізка є $P_{i-1}=(r,q)$. Вибір наступної точки S_i або T_i залежить від знака різниці *(s-t)*. Якщо $(s-t)<0$, то $P_i=T_i=(r+1,q)$ і тоді $x_{i+1}=x_i+1$, $y_{i+1}=y_i$, якщо $(s-t)\geq 0$, то $P_i=S_i=(r+1,q+1)$, тоді $x_{i+1}=x_i+1$, $y_{i+1}=y_i$.

$$s=\frac{dy}{dx}(r+1)-q\,,\ t=q+1-\frac{dy}{dx}(r+1)$$

$$s-t=2\cdot\frac{dy}{dx}(r+1)-2q-1\Rightarrow dx(s-t)=2(r\cdot dy-qdx)+2\cdot dy-dx\,.$$

Оскільки знак $dx(s-t)$ збігається зі знаком різниці *(s-t)*, то будемо перевіряти знак вираження $d_i=dx(s-t)$. Тому що $r=x_{i-1}$ і $q=y_{i-1}$, то $d_{x+1}=d_i-2dy(y_i-y_{i-1})$.

Нехай на попередньому кроці $d_i<0$, тоді $(y_i-y_{i-1})=0$ і $d_{i+1}=d_i+2dy$. Якщо ж на попередньому кроці $d_i\geq 0$, то $(y_i-y_{i-1})=1$ і $d_{i+1}=d_i+2(dy-dx)$.

Залишилося довідатися, як обчислити d_1.

При $i=1$: $(x_0,y_0)=(0,0)\Rightarrow d_1=2dy-dx$.

Перед тим, як досліджувати методи одержання зображень більш складних, ніж відрізки прямих, розглянемо проблему, що незримо є присутньою у більшості задач комп'ютерної графіки. Ця проблема відсікання зображення по деякій границі, наприклад по границі екрана, або, у загальному випадку, деякого прямокутного вікна. Розглянемо цю задачу стосовно до відрізків прямих. Деякі з них повністю лежать усередині області екрана, інші цілком поза нею, а деякі перетинають границю екрана. Правильне відображення відрізків означає знаходження точок перетинання їх із границею екрана й малювання тільки тих їхніх частин, які попадають на екран. Один з очевидних способів відсікання відрізків полягає у визначенні точок перетинання прямої, на якій розміщений відрізок, з кожною із чотирьох прямих, на яких лежать границі вікна, й у перевірці, чи не лежить хоча б одна точка перетинання на границі. У цьому випадку для кожної пари сторона-відрізок необхідно вирішувати систему із двох рівнянь, використовуючи операції множення й ділення. При цьому зручно параметричне завдання прямих:

$$x=x1+t(x2-x1)$$
$$y=y1+t(y2-y1)$$

Для $t\in[0,1]$ ці рівняння визначають точки, що перебувають між (x_1,y_1) і (x_2,y_2). Спеціальної перевірки вимагає випадок, коли відрізок

перебуває ліворуч. Нехай на *(i-1)*-му кроці поточною точкою відрізка є $P_{i-1}=(r,q)$. Вибір наступної точки S_i або T_i залежить від знака різниці *(s-t)*. Якщо $(s-t)<0$, то $P_i=T_i=(r+1,q)$ і тоді $x_{i+1}=x_i+1$, $y_{i+1}=y_i$, якщо $(s-t)\geq 0$, то $P_i=S_i=(r+1,q+1)$, тоді $x_{i+1}=x_i+1$, $y_{i+1}=y_i$.

$$s=\frac{dy}{dx}(r+1)-q\,,\ t=q+1-\frac{dy}{dx}(r+1)$$

$$s-t=2\cdot\frac{dy}{dx}(r+1)-2q-1\Rightarrow dx(s-t)=2(r\cdot dy-qdx)+2\cdot dy-dx\,.$$

Оскільки знак $dx(s-t)$ збігається зі знаком різниці *(s-t)*, то будемо перевіряти знак вираження $d_i=dx(s-t)$. Тому що $r=x_{i-1}$ і $q=y_{i-1}$, то $d_{x+1}=d_i-2dy(y_i-y_{i-1})$.

Нехай на попередньому кроці $d_i<0$, тоді $(y_i-y_{i-1})=0$ і $d_{i+1}=d_i+2dy$. Якщо ж на попередньому кроці $d_i\geq 0$, то $(y_i-y_{i-1})=1$ і $d_{i+1}=d_i+2(dy-dx)$.

Залишилося довідатися, як обчислити d_1.

При $i=1$: $(x_0,y_0)=(0,0)\Rightarrow d_1=2dy-dx$.

Перед тим, як досліджувати методи одержання зображень більш складних, ніж відрізки прямих, розглянемо проблему, що незримо є присутньою у більшості задач комп'ютерної графіки. Ця проблема відсікання зображення по деякій границі, наприклад по границі екрана, або, у загальному випадку, деякого прямокутного вікна. Розглянемо цю задачу стосовно до відрізків прямих. Деякі з них повністю лежать усередині області екрана, інші цілком поза нею, а деякі перетинають границю екрана. Правильне відображення відрізків означає знаходження точок перетинання їх із границею екрана й малювання тільки тих їхніх частин, які попадають на екран. Один з очевидних способів відсікання відрізків полягає у визначенні точок перетинання прямої, на якій розміщений відрізок, з кожною із чотирьох прямих, на яких лежать границі вікна, й у перевірці, чи не лежить хоча б одна точка перетинання на границі. У цьому випадку для кожної пари сторона-відрізок необхідно вирішувати систему із двох рівнянь, використовуючи операції множення й ділення. При цьому зручно параметричне завдання прямих:

$$x=x1+t(x2-x1)$$
$$y=y1+t(y2-y1)$$

Для $t\in[0,1]$ ці рівняння визначають точки, що перебувають між (x_1,y_1) і (x_2,y_2). Спеціальної перевірки вимагає випадок, коли відрізок

паралельний стороні вікна. Нехай координата x точки перетинання знайдена, тоді

$$t = \frac{x - x1}{x2 - x1} \Rightarrow y = y1 + \frac{x - x1}{x2 - x1}(y2 - y1).$$

Розглянемо алгоритм Коена-Сазерленда для відсікання відрізків прямих. Цей алгоритм дозволяє легко визначати знаходження відрізка повністю усередині або повністю зовні вікна і якщо так, то його можна малювати або не малювати, не піклуючись про відсікання по границі вікна.

Для роботи алгоритму вся площина, в якій лежить вікно, розбивається на дев'ять підобластей або квадрантів, як показано на рис.3.17.

1001	1000	1010
0001	0000	0010
0101	0100	0110

Рис. 3.17. Розбивка на підобласті в методі Коена-Сазерленда

Вікну відповідає область, позначена кодом 0000. Кінцевим точкам відрізка приписується 4-бітний код “поза/усередині” залежно від знаходження відрізка у відповідній підобласті. Кожному біту привласнюється значення 1 відповідно до такого правила:

біт 1 - точка перебуває вище вікна;

біт 2 - точка перебуває нижче вікна;

біт 3 - точка перебуває праворуч від вікна;

біт 4 - точка перебуває ліворуч від вікна;

Інакше, біту привласнюється нульове значення. Значення цих бітів для кінцевих точок відрізків легко визначити по знаках відповідних різниць: $(y_{max}-y)$ – для 1-го біта, $(y-y_{max})$ – для 2-го біта, $(x_{max}-x)$ – для 3-го біта й $(x-x_{min})$ – для 4-го біта. Відрізок рисується без відсікання, тобто приймається цілком, якщо обидва коди дорівнюють 0000, або [кодР1]АБО[кодР2]=0000 , де АБО – бінарна операція. Відрізок відкидається без обчислень якщо обидва його кінці перебувають вище, нижче, правіше або лівіше вікна.

У цих випадках відповідні біти в обох кодах дорівнюють 1 і це легко визначити, помноживши ці коди по бінарній операції І. Якщо

паралельний стороні вікна. Нехай координата x точки перетинання знайдена, тоді

$$t = \frac{x - x1}{x2 - x1} \Rightarrow y = y1 + \frac{x - x1}{x2 - x1}(y2 - y1).$$

Розглянемо алгоритм Коена-Сазерленда для відсікання відрізків прямих. Цей алгоритм дозволяє легко визначати знаходження відрізка повністю усередині або повністю зовні вікна і якщо так, то його можна малювати або не малювати, не піклуючись про відсікання по границі вікна.

Для роботи алгоритму вся площина, в якій лежить вікно, розбивається на дев'ять підобластей або квадрантів, як показано на рис.3.17.

1001	1000	1010
0001	0000	0010
0101	0100	0110

Рис. 3.17. Розбивка на підобласті в методі Коена-Сазерленда

Вікну відповідає область, позначена кодом 0000. Кінцевим точкам відрізка приписується 4-бітний код “поза/усередині” залежно від знаходження відрізка у відповідній підобласті. Кожному біту привласнюється значення 1 відповідно до такого правила:

біт 1 - точка перебуває вище вікна;

біт 2 - точка перебуває нижче вікна;

біт 3 - точка перебуває праворуч від вікна;

біт 4 - точка перебуває ліворуч від вікна;

Інакше, біту привласнюється нульове значення. Значення цих бітів для кінцевих точок відрізків легко визначити по знаках відповідних різниць: $(y_{max}-y)$ – для 1-го біта, $(y-y_{max})$ – для 2-го біта, $(x_{max}-x)$ – для 3-го біта й $(x-x_{min})$ – для 4-го біта. Відрізок рисується без відсікання, тобто приймається цілком, якщо обидва коди дорівнюють 0000, або [кодР1]АБО[кодР2]=0000 , де АБО – бінарна операція. Відрізок відкидається без обчислень якщо обидва його кінці перебувають вище, нижче, правіше або лівіше вікна.

У цих випадках відповідні біти в обох кодах дорівнюють 1 і це легко визначити, помноживши ці коди по бінарній операції І. Якщо

результат операції I дорівнює 0000, то відрізок не можна і не прийняти і не відкинути, тому що він може перетинатися з вікном. У цьому випадку застосовується послідовний поділ відрізка, так що на кожному кроці кінцева точка відрізка з ненульовим кодом «поза/усередині» заміняється на точку, що лежить на стороні вікна або на прямій відрізка. При цьому порядок перебору сторін вікна не має значення.

3.5. Нормуючі перетворення видимого об'єкта

Щоб розглянути задачі нормуючих перетворень видимих об’єктів, задамо центральну перспективну проекцію із центром проекції на початку координат, як показано на рис. 3.18. Для реальних обчислень необхідно також визначити значення мінімальної і максимальної площин, що відтинають, по z координаті $z = z_{\min}$ і $z = z_{\max}$, відповідно.

Границі екрана або вікна виведення задають чотири відсікаючих площини зверху, знизу, праворуч і ліворуч. Таким чином, зображення, одержуване за допомогою нашої проекції, може перебувати тільки всередині усіченої піраміди, утвореної згаданими площинами, причому об'єкти поза цією пірамідою не проектуються на екран, тобто є невидимими для спостерігача. Видимим об'ємом називається замкнута область простору, об'єкти всередині якої проектуються на екран. У випадку центральної перспективної проекції видимим об'ємом є усічена піраміда.

Однією з важливих задач комп'ютерної графіки є знаходження ефективного способу відсікання тривимірних об'єктів по границі видимого об'єму й видалення невидимих ребер і граней.

Наприклад, у випадку центральної перспективи, для розв’язання задачі відсікання довелося б для кожної грані або ребра знаходити точки перетинання із площинами усіченої піраміди, що в загальному випадку вимагало б значних обчислень. Рішення полягає в перетворенні видимого об'єму до виду, в якому обчислення проводилися б значно простіше. У загальному випадку ідея полягає в тому, щоб звести перетворення центральної перспективи математично до виду паралельної проекції, у якій, мабуть, операція взяття проекції зводиться до простого відкидання точок координати z.

результат операції I дорівнює 0000, то відрізок не можна і не прийняти і не відкинути, тому що він може перетинатися з вікном. У цьому випадку застосовується послідовний поділ відрізка, так що на кожному кроці кінцева точка відрізка з ненульовим кодом «поза/усередині» заміняється на точку, що лежить на стороні вікна або на прямій відрізка. При цьому порядок перебору сторін вікна не має значення.

3.5. Нормуючі перетворення видимого об'єкта

Щоб розглянути задачі нормуючих перетворень видимих об’єктів, задамо центральну перспективну проекцію із центром проекції на початку координат, як показано на рис. 3.18. Для реальних обчислень необхідно також визначити значення мінімальної і максимальної площин, що відтинають, по z координаті $z = z_{\min}$ і $z = z_{\max}$, відповідно.

Границі екрана або вікна виведення задають чотири відсікаючих площини зверху, знизу, праворуч і ліворуч. Таким чином, зображення, одержуване за допомогою нашої проекції, може перебувати тільки всередині усіченої піраміди, утвореної згаданими площинами, причому об'єкти поза цією пірамідою не проектуються на екран, тобто є невидимими для спостерігача. Видимим об'ємом називається замкнута область простору, об'єкти всередині якої проектуються на екран. У випадку центральної перспективної проекції видимим об'ємом є усічена піраміда.

Однією з важливих задач комп'ютерної графіки є знаходження ефективного способу відсікання тривимірних об'єктів по границі видимого об'єму й видалення невидимих ребер і граней.

Наприклад, у випадку центральної перспективи, для розв’язання задачі відсікання довелося б для кожної грані або ребра знаходити точки перетинання із площинами усіченої піраміди, що в загальному випадку вимагало б значних обчислень. Рішення полягає в перетворенні видимого об'єму до виду, в якому обчислення проводилися б значно простіше. У загальному випадку ідея полягає в тому, щоб звести перетворення центральної перспективи математично до виду паралельної проекції, у якій, мабуть, операція взяття проекції зводиться до простого відкидання точок координати z.

Будемо вирішувати задачу у два етапи. На початку приведемо видимий об'єм до нормованого виду. При цьому значення z_{max}=1, а границі по осях *x* і *y* лежать у діапазоні [-1,1], як показано на рис.3.19.

Нормуючим перетворенням у цьому випадку буде операція масштабування, що для довільної *X* точки виражається у вигляді

$$X' = X \cdot S\left(1/x_{\max}, 1/y_{\max}, 1/z_{\max}\right),$$

де $\frac{x_{\max} \cdot a}{z_{\max}} = x_{екр} \Rightarrow x_{екр} = \frac{z_{\max} x_{\max}}{a}$ і, відповідно,

$$y_{екр} = \frac{z_{\max} \cdot y_{\max}}{a}.$$

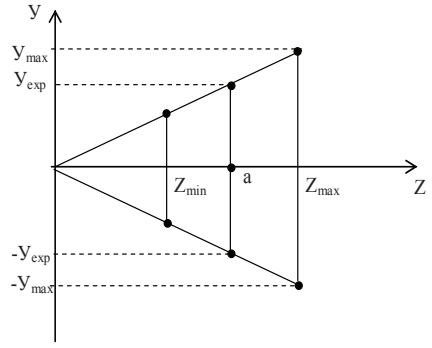


Рис 3.18. Видимий об'єкт, вид збоку

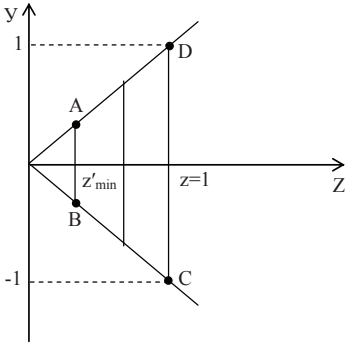


Рис. 3.19. Нормований видимий об'єкт

Нормований видимий об'єм дозволяє з більшою легкістю вирішувати задачу відсікання по границі. А саме, у цьому випадку може застосовуватися модифікований варіант алгоритму Коена-Сазарленда, в якому замість 4-бітових використовуються 6-бітові коди поза/усередині для опису знаходження точки у відповідній області простору.

Рівняння бічних граней видимого об'єму сильно спрощуються, наприклад, для правої площини, що відтинає, рівняння запишеться, як $z = x$, а для лівобічної $z = -x$ і т. ін. Тоді для деякої точки *(x,y,z)* умова встановлення біта в одиницю буде такою:

Будемо вирішувати задачу у два етапи. На початку приведемо видимий об'єм до нормованого виду. При цьому значення z_{max}=1, а границі по осях *x* і *y* лежать у діапазоні [-1,1], як показано на рис.3.19.

Нормуючим перетворенням у цьому випадку буде операція масштабування, що для довільної *X* точки виражається у вигляді

$$X' = X \cdot S\left(1/x_{\max}, 1/y_{\max}, 1/z_{\max}\right),$$

де $\frac{x_{\max} \cdot a}{z_{\max}} = x_{екр} \Rightarrow x_{екр} = \frac{z_{\max} x_{\max}}{a}$ і, відповідно,

$$y_{екр} = \frac{z_{\max} \cdot y_{\max}}{a}.$$

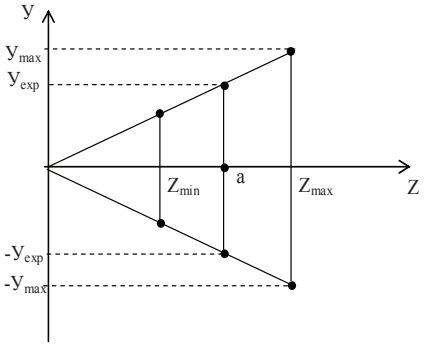


Рис 3.18. Видимий об'єкт, вид збоку

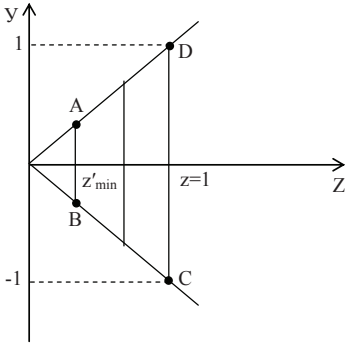


Рис. 3.19. Нормований видимий об'єкт

Нормований видимий об'єм дозволяє з більшою легкістю вирішувати задачу відсікання по границі. А саме, у цьому випадку може застосовуватися модифікований варіант алгоритму Коена-Сазарленда, в якому замість 4-бітових використовуються 6-бітові коди поза/усередині для опису знаходження точки у відповідній області простору.

Рівняння бічних граней видимого об'єму сильно спрощуються, наприклад, для правої площини, що відтинає, рівняння запишеться, як $z = x$, а для лівобічної $z = -x$ і т. ін. Тоді для деякої точки *(x,y,z)* умова встановлення біта в одиницю буде такою:

1-й біт: $y>z$;
2-й біт: $z<-z$;
3-й біт: $x>z$;
4-й біт: $x<-z$;
5-й біт: $z>1$;
6-й біт: $z<z'_{min}$.

Для ефективного розв'язання задачі видалення невидимих ребер/граней перетворимо нормований видимий об'єм до канонічного виду, як показано на рис.3.20.

Це досягається за допомогою матриці

$$M_p = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \dfrac{1}{1-z'_{\min}} & 1 \\ 0 & 0 & \dfrac{-z'_{\min}}{1-z'_{\min}} & 0 \end{bmatrix}$$

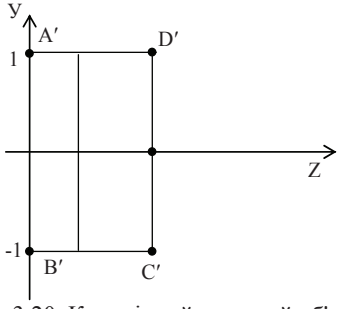


Рис. 3.20. Канонічний видимий об'єкт

Після застосування матриці M_p нормований видимий об'єм стає прямокутним паралелепіпедом, що дозволяє перейти від центральної перспективної до паралельної проекції.

Легко перевірити, що, як показано на рис. 3.18, 3.19, $D'=DM_p$, $A'=AM_p$, $B'=BM_p$, $C'=CM_p$, а також, наприклад,

$[z'_{min}, z'_{min}, z'_{min}, 1] \rightarrow [1,1,0,1]$.

Отже, нормуючі перетворення видимого об'єкта можуть виробляться за два кроки.

1 крок - перетворення до нормованого видимого об'єкта й відсікання за тривимірним алгоритмом Коена-Сазерленда.

1-й біт: $y>z$;
2-й біт: $z<-z$;
3-й біт: $x>z$;
4-й біт: $x<-z$;
5-й біт: $z>1$;
6-й біт: $z<z'_{min}$.

Для ефективного розв'язання задачі видалення невидимих ребер/граней перетворимо нормований видимий об'єм до канонічного виду, як показано на рис.3.20.

Це досягається за допомогою матриці

$$M_p = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \dfrac{1}{1-z'_{\min}} & 1 \\ 0 & 0 & \dfrac{-z'_{\min}}{1-z'_{\min}} & 0 \end{bmatrix}$$

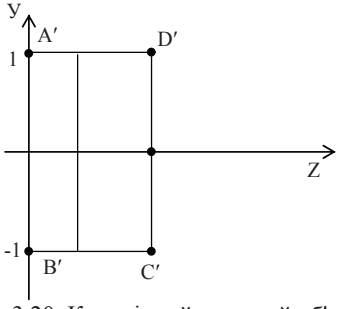


Рис. 3.20. Канонічний видимий об'єкт

Після застосування матриці M_p нормований видимий об'єм стає прямокутним паралелепіпедом, що дозволяє перейти від центральної перспективної до паралельної проекції.

Легко перевірити, що, як показано на рис. 3.18, 3.19, $D'=DM_p$, $A'=AM_p$, $B'=BM_p$, $C'=CM_p$, а також, наприклад,

$[z'_{min}, z'_{min}, z'_{min}, 1] \rightarrow [1,1,0,1]$.

Отже, нормуючі перетворення видимого об'єкта можуть виробляться за два кроки.

1 крок - перетворення до нормованого видимого об'єкта й відсікання за тривимірним алгоритмом Коена-Сазерленда.

2 крок - перетворення до прямокутного паралелепіпеда за допомогою матриці M_p й видалення схованих поверхонь за умови рівності координат *x* і *y*.

3.6. Алгоритми видалення невидимих ребер і граней

Алгоритми видалення невидимих граней можуть бути умовно поділені на два класи залежно від принципів, закладених для їх реалізації. Перший клас – це алгоритми, що працюють у просторі об'єкта. Це означає, що для визначення видимості даної грані порівнюється її взаємне розташування з усіма іншими гранями в тривимірній сцені. Нехай N – кількість граней у тривимірній сцені. Для побудови тривимірної сцени в цьому випадку необхідно зрівняти положення кожної грані із тими, які залишилися, що вимагає близько N^2 операцій. Наприклад, нехай кількість граней у тривимірній сцені N=1000, тоді час роботи алгоритмів цього класу дорівнює 1,000,000 операцій.

Другий клас це – алгоритми, що працюють у просторі зображення, засновані на знаходженні точки найближчої грані, яку перетинає промінь зору, що проходить через задану точку на растрі. Оскільки число точок на растровому екрані фіксоване, то алгоритми цього класу менш чутливі до збільшення кількості об'єктів у тривимірній сцені. Нехай n - число точок на растровому екрані. Тоді кількість операцій, необхідних для побудови тривимірної сцени, буде близько $n \times N$. Наприклад, для екранного дозволу 320×200 точок, n=64000, тоді кількість операцій для N=1000 граней буде приблизно 64,000,000. Вибір класу алгоритму може залежати від особливостей конкретної задачі, а також від способів реалізації алгоритму.

Розглянемо алгоритм видалення невидимих граней з використанням z-буфера, що є одним з найбільш часто використовуваних у сучасних додатках комп'ютерної графіки. Він працює в просторі зображення й застосовується в таких популярних графічних бібліотеках, як OpenGL і Direct3D.

Алгоритм працює в паралельній проекції. Нехай розміри вікна висновку або екрана становлять X точок завширшки й Y точок у висоту. Як z-буфер заведемо двовимірний прямокутний масив чисел, що за розмірністю збігається з вікном висновку або екрана, тобто $X \times Y$. В

2 крок - перетворення до прямокутного паралелепіпеда за допомогою матриці M_p й видалення схованих поверхонь за умови рівності координат *x* і *y*.

3.6. Алгоритми видалення невидимих ребер і граней

Алгоритми видалення невидимих граней можуть бути умовно поділені на два класи залежно від принципів, закладених для їх реалізації. Перший клас – це алгоритми, що працюють у просторі об'єкта. Це означає, що для визначення видимості даної грані порівнюється її взаємне розташування з усіма іншими гранями в тривимірній сцені. Нехай N – кількість граней у тривимірній сцені. Для побудови тривимірної сцени в цьому випадку необхідно зрівняти положення кожної грані із тими, які залишилися, що вимагає близько N^2 операцій. Наприклад, нехай кількість граней у тривимірній сцені N=1000, тоді час роботи алгоритмів цього класу дорівнює 1,000,000 операцій.

Другий клас це – алгоритми, що працюють у просторі зображення, засновані на знаходженні точки найближчої грані, яку перетинає промінь зору, що проходить через задану точку на растрі. Оскільки число точок на растровому екрані фіксоване, то алгоритми цього класу менш чутливі до збільшення кількості об'єктів у тривимірній сцені. Нехай n - число точок на растровому екрані. Тоді кількість операцій, необхідних для побудови тривимірної сцени, буде близько $n \times N$. Наприклад, для екранного дозволу 320×200 точок, n=64000, тоді кількість операцій для N=1000 граней буде приблизно 64,000,000. Вибір класу алгоритму може залежати від особливостей конкретної задачі, а також від способів реалізації алгоритму.

Розглянемо алгоритм видалення невидимих граней з використанням z-буфера, що є одним з найбільш часто використовуваних у сучасних додатках комп'ютерної графіки. Він працює в просторі зображення й застосовується в таких популярних графічних бібліотеках, як OpenGL і Direct3D.

Алгоритм працює в паралельній проекції. Нехай розміри вікна висновку або екрана становлять X точок завширшки й Y точок у висоту. Як z-буфер заведемо двовимірний прямокутний масив чисел, що за розмірністю збігається з вікном висновку або екрана, тобто $X \times Y$. В

z-буфері будуть зберігатися поточні значення z-координат кожного піксела.

На початку роботи алгоритму в z-буфер заносяться значення, що відповідають нескінченності. Кожна грань тривимірного об'єкта, подана у вигляді багатокутника, перетвориться в растрову форму. При розкладанні в растр для кожної точки багатокутника обчислюється значення її z-координати. Якщо z-координата виявилася менша, ніж поточне значення в z-буфері, то в z-буфер заноситься z-координата точки і на екрані рисується точка кольорами поточного багатокутника. Після розкладання в растр всіх багатокутників зображення тривимірної сцени побудовано.

Розглянемо спосіб прискореного обчислення z-координат під час розкладання багатокутників у растр. Запишемо рівняння площини, утвореної багатокутником у просторі: $Ax+By+Cz=0$.

Виразимо z-координату точки: $z = \frac{-D-Ax-By}{C} = f(x,y)$.

Нехай $z_0=f(x_0,y_0)$. Знайдемо z-координату для сусідньої точки $z_1=f(x_0+\Delta x,y_0)=\frac{-D-A(x_0+\Delta x)-By_0}{C}=\frac{-D-Ax_0-By_0}{C}-\frac{A\Delta x}{C}=z_0-\frac{A\Delta x}{C}$.

Для сусіднього піксела на екрані $\Delta x=1$, тоді $\frac{A\Delta x}{C}=Const$, звідси виходить, що $z_1=z_0-Const$.

Таким чином, обчислення z-координати сусіднього піксела зводиться до однієї операції вирахування. Розглянемо далі алгоритм видалення невидимих граней методом сортування по глибині (автори: Ньюэлл, Санча). Частина цього методу працює в просторі об'єкта, а частина в просторі зображення. Введемо визначення просторової оболонки.

Просторовою оболонкою тривимірного об'єкта називається мінімальний прямокутний паралелепіпед, що цілком містить усередині себе даний об'єкт. Аналогічно можна визначити двовимірні й одномірну просторові оболонки.

Метод складається із трьох основних кроків:

Крок 1. Упорядкування всіх багатокутників відповідно до їх найбільших z-координат.

z-буфері будуть зберігатися поточні значення z-координат кожного піксела.

На початку роботи алгоритму в z-буфер заносяться значення, що відповідають нескінченності. Кожна грань тривимірного об'єкта, подана у вигляді багатокутника, перетвориться в растрову форму. При розкладанні в растр для кожної точки багатокутника обчислюється значення її z-координати. Якщо z-координата виявилася менша, ніж поточне значення в z-буфері, то в z-буфер заноситься z-координата точки і на екрані рисується точка кольорами поточного багатокутника. Після розкладання в растр всіх багатокутників зображення тривимірної сцени побудовано.

Розглянемо спосіб прискореного обчислення z-координат під час розкладання багатокутників у растр. Запишемо рівняння площини, утвореної багатокутником у просторі: $Ax+By+Cz=0$.

Виразимо z-координату точки: $z = \frac{-D-Ax-By}{C} = f(x,y)$.

Нехай $z_0=f(x_0,y_0)$. Знайдемо z-координату для сусідньої точки $z_1=f(x_0+\Delta x,y_0)=\frac{-D-A(x_0+\Delta x)-By_0}{C}=\frac{-D-Ax_0-By_0}{C}-\frac{A\Delta x}{C}=z_0-\frac{A\Delta x}{C}$.

Для сусіднього піксела на екрані $\Delta x=1$, тоді $\frac{A\Delta x}{C}=Const$, звідси виходить, що $z_1=z_0-Const$.

Таким чином, обчислення z-координати сусіднього піксела зводиться до однієї операції вирахування. Розглянемо далі алгоритм видалення невидимих граней методом сортування по глибині (автори: Ньюэлл, Санча). Частина цього методу працює в просторі об'єкта, а частина в просторі зображення. Введемо визначення просторової оболонки.

Просторовою оболонкою тривимірного об'єкта називається мінімальний прямокутний паралелепіпед, що цілком містить усередині себе даний об'єкт. Аналогічно можна визначити двовимірні й одномірну просторові оболонки.

Метод складається із трьох основних кроків:

Крок 1. Упорядкування всіх багатокутників відповідно до їх найбільших z-координат.

127

Крок 2. Дозвіл всіх невизначеностей, які виникають при перекритті z-оболонок багатокутників.

Крок 3. Перетворення кожного з багатокутників у растрову форму, вироблене в порядку зменшення їх найбільшої z-координати.

Найближчі багатокутники перетворяться в растрову форму останніми й закривають більш віддалені багатокутники, тому що зображаються поверх попередніх. Реалізація пунктів 1 і 3 досить очевидна. Розглянемо докладніше пункт 2.

Нехай багатокутник P після впорядкування перебуває наприкінці списку, тобто є найбільш вилученим. Всі багатокутники Q, чиї оболонки перекриваються з z-оболонкою P, повинні проходити перевірку за п'ятьома тестами (кроками).

Якщо на деякому кроці отримана позитивна відповідь, то P відразу перетвориться в растрову форму.

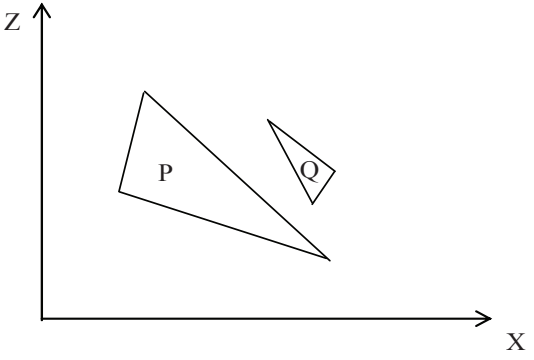


Рис. 3.21. Z-оболонки трикутників P і Q не перетинаються

П'ять тестів:

– x-оболонки багатокутників не перекриваються, тому самі багатокутники теж не перекриваються;

– y-оболонки багатокутників не перекриваються, тому самі багатокутники теж не перекриваються;

– P повністю розташований з тієї сторони від площини Q, що далі від точки зору (цей тест дає позитивну відповідь, як показано на рис.3.22, а);

Крок 2. Дозвіл всіх невизначеностей, які виникають при перекритті z-оболонок багатокутників.

Крок 3. Перетворення кожного з багатокутників у растрову форму, вироблене в порядку зменшення їх найбільшої z-координати.

Найближчі багатокутники перетворяться в растрову форму останніми й закривають більш віддалені багатокутники, тому що зображаються поверх попередніх. Реалізація пунктів 1 і 3 досить очевидна. Розглянемо докладніше пункт 2.

Нехай багатокутник P після впорядкування перебуває наприкінці списку, тобто є найбільш вилученим. Всі багатокутники Q, чиї оболонки перекриваються з z-оболонкою P, повинні проходити перевірку за п'ятьома тестами (кроками).

Якщо на деякому кроці отримана позитивна відповідь, то P відразу перетвориться в растрову форму.

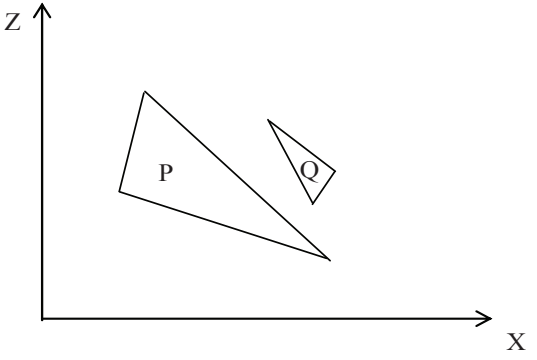


Рис. 3.21. Z-оболонки трикутників P і Q не перетинаються

П'ять тестів:

– x-оболонки багатокутників не перекриваються, тому самі багатокутники теж не перекриваються;

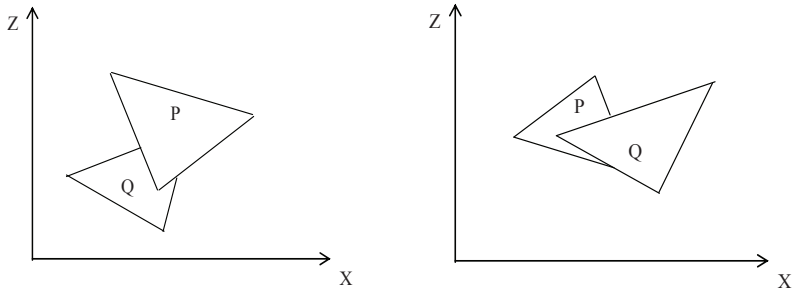
– y-оболонки багатокутників не перекриваються, тому самі багатокутники теж не перекриваються;

– P повністю розташований з тієї сторони від площини Q, що далі від точки зору (цей тест дає позитивну відповідь, як показано на рис.3.22, а);

– Q повністю розташований з тієї сторони від площини P, що ближче до точки зору (цей тест дає позитивну відповідь, як показано на рис.3.22,б).

Проекції багатокутників на площині *xOy*, тобто на екрані, не перекриваються (це визначається порівнянням ребер одного багатокутника з ребрами іншого).

Якщо у всіх п'яти тестах отримана негативна відповідь, то P дійсно закриває Q. Тоді міняємо P і Q у списку місцями. У випадку, як показано на рис. 3.23, алгоритм зациклюється.



а б

Рис. 3.22. Взаємні розташування трикутників у просторі:
а – трикутник P на передньому плані; б – трикутник Q на передньому плані

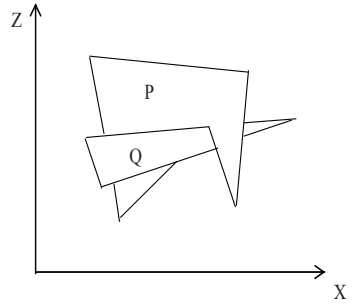


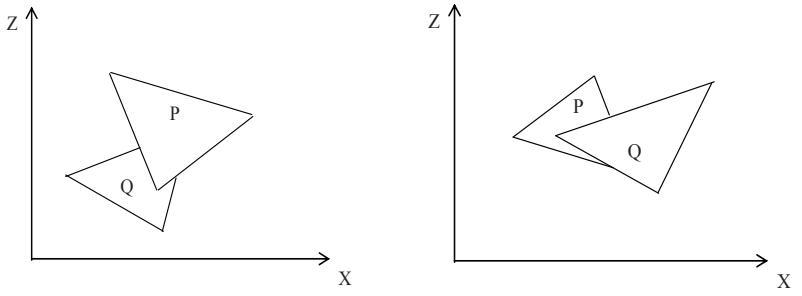
Рис. 3.23. Розміщення об’єктів, при яких алгоритм зациклюється

Для запобігання зациклювання вводиться обмеження: багатокутник, переміщений у кінець списку (тобто позначений), не може бути повторно переміщений. Замість цього багатокутник P або Q розділяється площиною іншого на два нових багатокутники. Ці два нових багатокутники включаються у відповідні місця впорядкованого списку, і алгоритм продовжує роботу.

– Q повністю розташований з тієї сторони від площини P, що ближче до точки зору (цей тест дає позитивну відповідь, як показано на рис.3.22,б).

Проекції багатокутників на площині *xOy*, тобто на екрані, не перекриваються (це визначається порівнянням ребер одного багатокутника з ребрами іншого).

Якщо у всіх п'яти тестах отримана негативна відповідь, то P дійсно закриває Q. Тоді міняємо P і Q у списку місцями. У випадку, як показано на рис. 3.23, алгоритм зациклюється.



а б

Рис. 3.22. Взаємні розташування трикутників у просторі:
а – трикутник P на передньому плані; б – трикутник Q на передньому плані

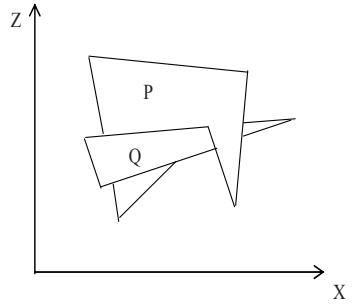


Рис. 3.23. Розміщення об’єктів, при яких алгоритм зациклюється

Для запобігання зациклювання вводиться обмеження: багатокутник, переміщений у кінець списку (тобто позначений), не може бути повторно переміщений. Замість цього багатокутник P або Q розділяється площиною іншого на два нових багатокутники. Ці два нових багатокутники включаються у відповідні місця впорядкованого списку, і алгоритм продовжує роботу.

На відміну від універсальних алгоритмів вузькоспеціалізований алгоритм видалення невидимих граней випуклих тіл дозволяє робити обчислення набагато швидше. Він працює для центральної перспективної проекції. Розглянемо роботу цього алгоритму на прикладі, як зображено на рис. 3.24.

Нехай спостерігач перебуває в точці A. Виберемо точку B, що свідомо є внутрішньою для опуклої фігури, у цьому випадку призми. Виберемо деяку грань, про яку ми хочемо довідатися, видима вона із точки A, або не видима. Побудуємо площину, в якій лежить обрана грань. Знайдемо точку перетинання площині й прямої, що утворена відрізком AB. Якщо точка перетинання прямої і площині лежить усередині відрізка AB, то робимо висновок, що дана грань видима. Якщо точка перетинання перебуває поза відрізком AB, то грань невидима. У випадку, коли пряма й площина паралельні, вважаємо що грань невидима.

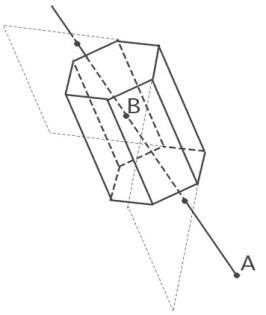


Рис. 3.24. Перетинання прямої AB із площинами граней призми

3.7. Моделі розрахунку освітленості граней тривимірних об'єктів

Основною характеристикою світла в комп'ютерній графіці є яскравість. Оскільки яскравість є суб'єктивним поняттям, заснованим на людському сприйнятті світла, то для чисельних розрахунків застосовується термін інтенсивність, що відповідає яскравості і є енергетичною характеристикою світлової хвилі. У розрахунках інтенсивність звичайно приймає значення від 0 до 1. При цьому інтенсивність дорівнює нулю при повній відсутності світла, а значення 1 відповідає максимальній яскравості.

У комп'ютерній графіці для розрахунку освітленості граней об'єктів найчастіше застосовується трикомпонентна колірна модель “Червоний, Зелений, Синій”, що в англійському варіанті записується RGB (Red, Green, Blue). Ця модель дозволяє задавати будь-які кольори у вигляді трьох компонентів інтенсивностей базових кольорів:

На відміну від універсальних алгоритмів вузькоспеціалізований алгоритм видалення невидимих граней випуклих тіл дозволяє робити обчислення набагато швидше. Він працює для центральної перспективної проекції. Розглянемо роботу цього алгоритму на прикладі, як зображено на рис. 3.24.

Нехай спостерігач перебуває в точці A. Виберемо точку B, що свідомо є внутрішньою для опуклої фігури, у цьому випадку призми. Виберемо деяку грань, про яку ми хочемо довідатися, видима вона із точки A, або не видима. Побудуємо площину, в якій лежить обрана грань. Знайдемо точку перетинання площині й прямої, що утворена відрізком AB. Якщо точка перетинання прямої і площині лежить усередині відрізка AB, то робимо висновок, що дана грань видима. Якщо точка перетинання перебуває поза відрізком AB, то грань невидима. У випадку, коли пряма й площина паралельні, вважаємо що грань невидима.

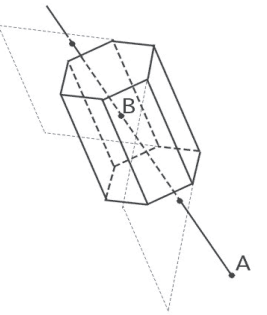


Рис. 3.24. Перетинання прямої AB із площинами граней призми

3.7. Моделі розрахунку освітленості граней тривимірних об'єктів

Основною характеристикою світла в комп'ютерній графіці є яскравість. Оскільки яскравість є суб'єктивним поняттям, заснованим на людському сприйнятті світла, то для чисельних розрахунків застосовується термін інтенсивність, що відповідає яскравості і є енергетичною характеристикою світлової хвилі. У розрахунках інтенсивність звичайно приймає значення від 0 до 1. При цьому інтенсивність дорівнює нулю при повній відсутності світла, а значення 1 відповідає максимальній яскравості.

У комп'ютерній графіці для розрахунку освітленості граней об'єктів найчастіше застосовується трикомпонентна колірна модель “Червоний, Зелений, Синій”, що в англійському варіанті записується RGB (Red, Green, Blue). Ця модель дозволяє задавати будь-які кольори у вигляді трьох компонентів інтенсивностей базових кольорів:

червоного, зеленого й синього. Інтенсивність відбитого світла точок просторових об'єктів обчислюють окремо для кожної з трьох складових колірних компонентів, а потім поєднують у результуючу трійку кольорів. Далі будемо вважати, що приклади розрахунку інтенсивностей відбитого світла застосовуються до кожного з трьох базових кольорів.

Під час розрахунку освітленості граней застосовують такі типи висвітлення й відбиття світла від поверхонь: *розсіюване*; *дифузійне*; *дзеркальне*.

Інтенсивність висвітлення граней тривимірних об'єктів розсіяним світлом уважається постійною в будь-якій точці простору. Вона обумовлена множинними відбиттями світла від усіх об'єктів у просторі. При висвітленні тривимірного об'єкта розсіяним світлом інтенсивність відбитого світла обчислюється як $I_a=I_p \cdot k_a$, де I_p- інтенсивність падаючого світла, $k_a \in [0,1]$ - коефіцієнт розсіюваного відбиття, залежить від відбиваючих властивостей, матеріалу грані.

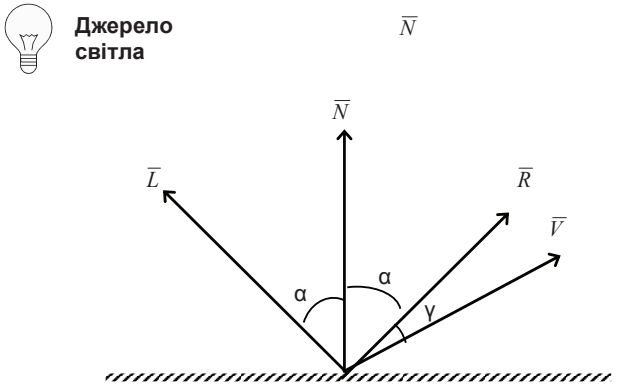


Рис. 3.25. Розрахунок інтенсивності відбитого світла

Для розрахунку інтенсивності дифузійного відбиття світла може застосовуватися закон косинусів Ламберта: $I_d=I_p \cdot k_d \cdot Cos(\alpha)$, де α- кут падіння, розраховується як кут між напрямком на джерело світла й нормаллю до поверхні. Нехай напрямок на джерело світла зображено одиничним вектором $\overline{L}$, а $\overline{N}$ – одиничний вектор нормалі. Тоді

червоного, зеленого й синього. Інтенсивність відбитого світла точок просторових об'єктів обчислюють окремо для кожної з трьох складових колірних компонентів, а потім поєднують у результуючу трійку кольорів. Далі будемо вважати, що приклади розрахунку інтенсивностей відбитого світла застосовуються до кожного з трьох базових кольорів.

Під час розрахунку освітленості граней застосовують такі типи висвітлення й відбиття світла від поверхонь: *розсіюване*; *дифузійне*; *дзеркальне*.

Інтенсивність висвітлення граней тривимірних об'єктів розсіяним світлом уважається постійною в будь-якій точці простору. Вона обумовлена множинними відбиттями світла від усіх об'єктів у просторі. При висвітленні тривимірного об'єкта розсіяним світлом інтенсивність відбитого світла обчислюється як $I_a=I_p \cdot k_a$, де I_p- інтенсивність падаючого світла, $k_a \in [0,1]$ - коефіцієнт розсіюваного відбиття, залежить від відбиваючих властивостей, матеріалу грані.

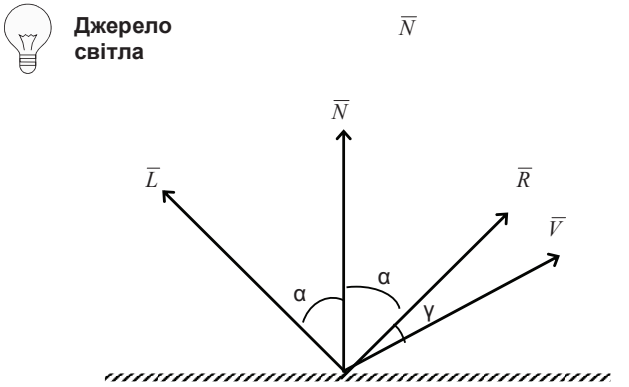


Рис. 3.25. Розрахунок інтенсивності відбитого світла

Для розрахунку інтенсивності дифузійного відбиття світла може застосовуватися закон косинусів Ламберта: $I_d=I_p \cdot k_d \cdot Cos(\alpha)$, де α- кут падіння, розраховується як кут між напрямком на джерело світла й нормаллю до поверхні. Нехай напрямок на джерело світла зображено одиничним вектором $\overline{L}$, а $\overline{N}$ – одиничний вектор нормалі. Тоді

$Cos\alpha = (\overline{L}, \overline{N})$ – скалярний добуток векторів, $I_d = I_p \cdot k_d \cdot (\overline{L}, \overline{N})$, де k_d– коефіцієнт дифузійного відбиття.

Обчислення дзеркально відбитого світла здійснюється також за допомогою різних емпіричних моделей, які дозволяють ураховувати реальну шорсткість поверхонь. Наприклад, у моделі, запропонованої Фонгом, інтенсивність дзеркально відбитого світла розраховується залежно від ступеня відхилення від середнього значення вектора дзеркально відбитого проміння світла. Нехай $\overline{R}$ – вектор дзеркально відбитого проміння світла, а $\overline{V}$ – вектор, що визначає напрямок на спостерігача. Тоді інтенсивність дзеркально відбитого світла по моделі Фонга розраховується так: $I_m=I_p \cdot k_m \cdot Cos^n(\gamma)$, де γ – кут між векторами $\overline{R}$ і $\overline{V}$. Константа n – може приймати значення від 1 до приблизно 200, залежно від відбиваючої здатності матеріалу. Більшим значенням n відповідає більший ступінь "гладкості" або "дзеркальності" поверхні. Якщо вектори $\overline{R}$ і $\overline{V}$ - нормовані, то формула перетвориться до вигляду: $I_m = I_p \cdot k_m \cdot (\overline{R}, \overline{V})^n$.

Інтенсивність відбитого світла зменшується обернено пропорційно квадрату відстані від джерела до спостерігача. Тому можна записати формулу розрахунку інтенсивності відбитого променя світла для трьох складових: розсіюваного, дифузійного й дзеркального відбиття з урахуванням відстані:

$$I = I_p k_a + \frac{I_p}{R + r^2}\left(k_d \cdot (\overline{L}, \overline{N}) + k_m (\overline{R}, \overline{V})^n\right),$$

де *r*- відстань від точки відбиття до спостерігача, а *R≥1* - деяка константа. Іноді для прискорення обчислень беруть не другий, а перший ступінь відстані *r*.

У системах комп'ютерної візуалізації також ураховуються такі властивості матеріалів відбиваючих поверхонь, як прозорість, переломлення й світіння. Ступінь прозорості матеріалу грані може описуватися за допомогою константи, що приймає значення від нуля до одиниці, причому значення 1 відповідає повній непрозорості матеріалу грані.

$Cos\alpha = (\overline{L}, \overline{N})$ – скалярний добуток векторів, $I_d = I_p \cdot k_d \cdot (\overline{L}, \overline{N})$, де k_d– коефіцієнт дифузійного відбиття.

Обчислення дзеркально відбитого світла здійснюється також за допомогою різних емпіричних моделей, які дозволяють ураховувати реальну шорсткість поверхонь. Наприклад, у моделі, запропонованої Фонгом, інтенсивність дзеркально відбитого світла розраховується залежно від ступеня відхилення від середнього значення вектора дзеркально відбитого проміння світла. Нехай $\overline{R}$ – вектор дзеркально відбитого проміння світла, а $\overline{V}$ – вектор, що визначає напрямок на спостерігача. Тоді інтенсивність дзеркально відбитого світла по моделі Фонга розраховується так: $I_m=I_p \cdot k_m \cdot Cos^n(\gamma)$, де γ – кут між векторами $\overline{R}$ і $\overline{V}$. Константа n – може приймати значення від 1 до приблизно 200, залежно від відбиваючої здатності матеріалу. Більшим значенням n відповідає більший ступінь "гладкості" або "дзеркальності" поверхні. Якщо вектори $\overline{R}$ і $\overline{V}$ - нормовані, то формула перетвориться до вигляду: $I_m = I_p \cdot k_m \cdot (\overline{R}, \overline{V})^n$.

Інтенсивність відбитого світла зменшується обернено пропорційно квадрату відстані від джерела до спостерігача. Тому можна записати формулу розрахунку інтенсивності відбитого променя світла для трьох складових: розсіюваного, дифузійного й дзеркального відбиття з урахуванням відстані:

$$I = I_p k_a + \frac{I_p}{R + r^2}\left(k_d \cdot (\overline{L}, \overline{N}) + k_m (\overline{R}, \overline{V})^n\right),$$

де *r*- відстань від точки відбиття до спостерігача, а *R≥1* - деяка константа. Іноді для прискорення обчислень беруть не другий, а перший ступінь відстані *r*.

У системах комп'ютерної візуалізації також ураховуються такі властивості матеріалів відбиваючих поверхонь, як прозорість, переломлення й світіння. Ступінь прозорості матеріалу грані може описуватися за допомогою константи, що приймає значення від нуля до одиниці, причому значення 1 відповідає повній непрозорості матеріалу грані.

3.8. Кубічні сплайни

Розглянемо задачу проведення гладких кривих по заданих граничних точках, або задачу інтерполяції. Оскільки через дві точки можна провести множину гладких кривих, то для розв'язання цієї задачі необхідно обмежити клас функцій, які будуть визначати шукану криву. Математичними сплайнами називають функції, використовувані для апроксимації кривих. Важливою їхньою властивістю є простота обчислень. На практиці часто використовують сплайни виду поліномів третього ступеня. З їхньою допомогою досить зручно проводити криві, які інтуїтивно відповідають людському суб'єктивному поняттю гладкості. Термін "сплайн" походить від англійського spline - що означає гнучку смужку сталі, яку застосовували креслярі для проведення плавних кривих, наприклад, для побудови обводів кораблів або літаків.

Розглянемо спочатку сплайнову функцію для побудови графіка функції однієї змінної (рис. 3.26). Нехай на площині задана послідовність точок $\{x_i,y_i\}$, $i=0...m$, причому $x_0<x_1<...<x_{m-1}<x_m$. Визначимо шукану функцію $y=S(x)$, причому поставимо дві умови:

1. Функція повинна проходити через всі задані точки: $S(x_i)=y_i$, $i=0...m$.

2. Функція повинна бути двічі безупинно диференційованою, тобто мати безперервну другу похідну на всьому відрізку $[x_0,x_m]$.

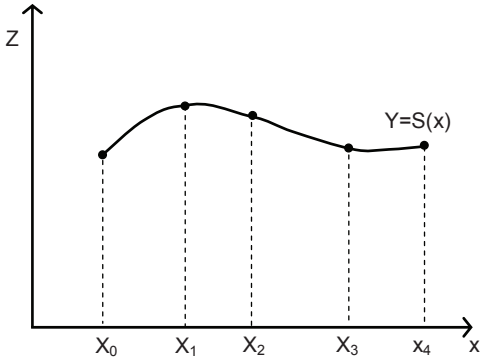


Рис. 3.26. Сплайнова функція

3.8. Кубічні сплайни

Розглянемо задачу проведення гладких кривих по заданих граничних точках, або задачу інтерполяції. Оскільки через дві точки можна провести множину гладких кривих, то для розв'язання цієї задачі необхідно обмежити клас функцій, які будуть визначати шукану криву. Математичними сплайнами називають функції, використовувані для апроксимації кривих. Важливою їхньою властивістю є простота обчислень. На практиці часто використовують сплайни виду поліномів третього ступеня. З їхньою допомогою досить зручно проводити криві, які інтуїтивно відповідають людському суб'єктивному поняттю гладкості. Термін "сплайн" походить від англійського spline - що означає гнучку смужку сталі, яку застосовували креслярі для проведення плавних кривих, наприклад, для побудови обводів кораблів або літаків.

Розглянемо спочатку сплайнову функцію для побудови графіка функції однієї змінної (рис. 3.26). Нехай на площині задана послідовність точок $\{x_i,y_i\}$, $i=0...m$, причому $x_0<x_1<...<x_{m-1}<x_m$. Визначимо шукану функцію $y=S(x)$, причому поставимо дві умови:

1. Функція повинна проходити через всі задані точки: $S(x_i)=y_i$, $i=0...m$.

2. Функція повинна бути двічі безупинно диференційованою, тобто мати безперервну другу похідну на всьому відрізку $[x_0,x_m]$.

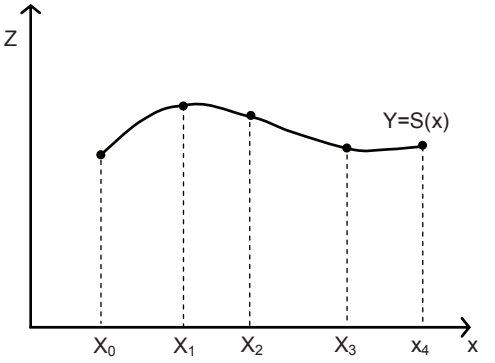


Рис. 3.26. Сплайнова функція

133

На кожному з відрізків $[x_i,x_{i+1}]$, $i=0...m-1$ будемо шукати нашу функцію у вигляді полінома третього ступеня

$$S_i(x)=\sum_{j=0}^{3}a_{ij}(x-x_i)^j.$$

Задача побудови полінома зводиться до знаходження коефіцієнтів a_{ij}. Оскільки для кожного з відрізків $[x_i,x_{i+1}]$ необхідно знайти 4 коефіцієнти a_{ij}, то всього кількість шуканих коефіцієнтів буде $4m$. Для знаходження всіх коефіцієнтів визначимо відповідну кількість рівнянь. Перші $(m-1)$ рівнянь одержуємо з умов збігу значень функції у внутрішніх вузлах x_i, $i=1...m-1$. Наступні $2(m-1)$ рівнянь одержуємо аналогічно з умов збігу значень перших і других похідних у внутрішніх вузлах. Разом з першою умовою одержуємо $m-1+m+1+m-1+m+1=4m-2$ рівнянь. Відсутні два рівняння можна одержати завданням значень перших похідних у кінцевих точках відрізка $[x_0,x_m]$. Так можуть бути задані граничні умови.

Перейдемо до більш складного випадку – завданню кривих у тривимірному просторі. У випадку функціонального завдання кривої

$$\begin{cases}y=f(x)\\z=f(x)\end{cases}$$

можливі багатозначності у випадку самоперетинань і незручності при значеннях похідних рівних ∞. Через це будемо шукати функцію в параметричному вигляді. Нехай t - незалежний параметр, такий що $0\le t\le 1$. Кубічним параметричним сплайном назвемо таку систему рівнянь:

$$\begin{cases}x(t)=a_xt^3+b_xt^2+c_xt+d_x\\y(t)=a_yt^3+b_yt^2+c_yt+d_y\\z(t)=a_zt^3+b_zt^2+c_zt+d_z\end{cases}$$

Координати точок на кривій описуються вектором $(x(t),y(t),z(t))$, а три похідні задають координати відповідного дотичного вектора в точці. Наприклад, для координати x

$$\frac{dx}{dt}=3a_xt^2+2b_xt+c_x.$$

На кожному з відрізків $[x_i,x_{i+1}]$, $i=0...m-1$ будемо шукати нашу функцію у вигляді полінома третього ступеня

$$S_i(x)=\sum_{j=0}^{3}a_{ij}(x-x_i)^j.$$

Задача побудови полінома зводиться до знаходження коефіцієнтів a_{ij}. Оскільки для кожного з відрізків $[x_i,x_{i+1}]$ необхідно знайти 4 коефіцієнти a_{ij}, то всього кількість шуканих коефіцієнтів буде $4m$. Для знаходження всіх коефіцієнтів визначимо відповідну кількість рівнянь. Перші $(m-1)$ рівнянь одержуємо з умов збігу значень функції у внутрішніх вузлах x_i, $i=1...m-1$. Наступні $2(m-1)$ рівнянь одержуємо аналогічно з умов збігу значень перших і других похідних у внутрішніх вузлах. Разом з першою умовою одержуємо $m-1+m+1+m-1+m+1=4m-2$ рівнянь. Відсутні два рівняння можна одержати завданням значень перших похідних у кінцевих точках відрізка $[x_0,x_m]$. Так можуть бути задані граничні умови.

Перейдемо до більш складного випадку – завданню кривих у тривимірному просторі. У випадку функціонального завдання кривої

$$\begin{cases}y=f(x)\\z=f(x)\end{cases}$$

можливі багатозначності у випадку самоперетинань і незручності при значеннях похідних рівних ∞. Через це будемо шукати функцію в параметричному вигляді. Нехай t - незалежний параметр, такий що $0\le t\le 1$. Кубічним параметричним сплайном назвемо таку систему рівнянь:

$$\begin{cases}x(t)=a_xt^3+b_xt^2+c_xt+d_x\\y(t)=a_yt^3+b_yt^2+c_yt+d_y\\z(t)=a_zt^3+b_zt^2+c_zt+d_z\end{cases}$$

Координати точок на кривій описуються вектором $(x(t),y(t),z(t))$, а три похідні задають координати відповідного дотичного вектора в точці. Наприклад, для координати x

$$\frac{dx}{dt}=3a_xt^2+2b_xt+c_x.$$

Одним із способів завдання параметричного кубічного сплайна є визначення координат початкової і кінцевої точок, а також дотичних векторів у них. Такий спосіб завдання називається формою Ерміта. Позначимо кінцеві точки P_1іP_4, а дотичні вектори в них R_1іR_4. Індекси обрані в такий спосіб з урахуванням подальшого викладу.

Будемо вирішувати задачу знаходження четвірки коефіцієнтів a_x, b_x,c_x,d_x, тому що для двох рівнянь, що залишилися, коефіцієнти знаходяться аналогічно. Запишемо умову для побудови сплайна:

$$x(0)=P_{1x},\ x(1)=P_{4x},\ x'(0)=R_{1x},\ x'(1)=R_{4x}\ . \qquad (3.25)$$

Перепишемо вираз для x у векторному виді:

$$x(t)=[t^3,t^2,t,1]\begin{bmatrix}a\\b\\c\\d\end{bmatrix}_x .$$

Позначимо вектор-рядок $T=[t^3,t^2,t,1]$ і вектор-стовпець коефіцієнтів

$$C_x=\begin{bmatrix}a\\b\\c\\d\end{bmatrix}_x ,\ \text{тоді}\ x(t)=TC_x\ .$$

З виразу (3.25) видно, що $x(0)=P_{1x}=[0,0,0,1]C_x$, $x(1)=P_{4x}=[1,1,1,1]C_x$. Для дотичних $x'(t)=[3t^2,2t,1,0]C_x,\Rightarrow$ $x'(0)=[0,0,1,0]C_x$,$x'(1)=[3,2,1,0]C_x$. Звідси одержуємо векторно-матричне рівняння:

$$\begin{bmatrix}P_{1x}\\P_{4x}\\R_{1x}\\R_{4x}\end{bmatrix}=\begin{bmatrix}0&0&0&1\\1&1&1&1\\0&0&1&0\\3&2&1&0\end{bmatrix}C_x$$

Ця система вирішується відносно C_x знаходженням зворотної матриці розміром 4×4.

Одним із способів завдання параметричного кубічного сплайна є визначення координат початкової і кінцевої точок, а також дотичних векторів у них. Такий спосіб завдання називається формою Ерміта. Позначимо кінцеві точки P_1іP_4, а дотичні вектори в них R_1іR_4. Індекси обрані в такий спосіб з урахуванням подальшого викладу.

Будемо вирішувати задачу знаходження четвірки коефіцієнтів a_x, b_x,c_x,d_x, тому що для двох рівнянь, що залишилися, коефіцієнти знаходяться аналогічно. Запишемо умову для побудови сплайна:

$$x(0)=P_{1x},\ x(1)=P_{4x},\ x'(0)=R_{1x},\ x'(1)=R_{4x}\ . \qquad (3.25)$$

Перепишемо вираз для x у векторному виді:

$$x(t)=[t^3,t^2,t,1]\begin{bmatrix}a\\b\\c\\d\end{bmatrix}_x .$$

Позначимо вектор-рядок $T=[t^3,t^2,t,1]$ і вектор-стовпець коефіцієнтів

$$C_x=\begin{bmatrix}a\\b\\c\\d\end{bmatrix}_x ,\ \text{тоді}\ x(t)=TC_x\ .$$

З виразу (3.25) видно, що $x(0)=P_{1x}=[0,0,0,1]C_x$, $x(1)=P_{4x}=[1,1,1,1]C_x$. Для дотичних $x'(t)=[3t^2,2t,1,0]C_x,\Rightarrow$ $x'(0)=[0,0,1,0]C_x$,$x'(1)=[3,2,1,0]C_x$. Звідси одержуємо векторно-матричне рівняння:

$$\begin{bmatrix}P_{1x}\\P_{4x}\\R_{1x}\\R_{4x}\end{bmatrix}=\begin{bmatrix}0&0&0&1\\1&1&1&1\\0&0&1&0\\3&2&1&0\end{bmatrix}C_x$$

Ця система вирішується відносно C_x знаходженням зворотної матриці розміром 4×4.

$$C_x = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_{1x} \\ P_{4x} \\ R_{1x} \\ R_{4x} \end{bmatrix} = M_h G_{hx}$$

Тут M_h - ермітова матриця, G_h - геометричний вектор Ерміта. Підставимо вираз C_x для знаходження $x(t)$: $x(t)=TM_hG_{hx}$. Аналогічно для інших координат: $y(t)=TM_hG_{hy}$, $z(t)=TM_hG_{hz}$.

Випишемо в явному виді формули для обчислення координат точок сплайна. Враховуючи що $TM_h=[(2t^3-3t^2+1),(-2t^3+3t^2),(t^3-2t^2+t),(t^3-t^2)]$, та домножуючи праву частину рівняння на G_{hx}, одержуємо:

$$x(t) = TM_h G_{hx} =$$
$$= P_{1x}\left(2t^3 - 3t^2 + 1\right) + P_{4x}\left(-2t^3 + 3t^2\right) + R_{1x}\left(t^3 - 2t^2 + t\right) + R_{4x}\left(t^3 - t^2\right).$$

Чотири функції в дужках називаються функціями сполучення.

Форму кривої, заданої у формі Ерміта, легко змінювати, якщо враховувати, що напрямок дотичного вектора задає початковий напрямок, а модуль дотичного вектора задає ступінь витягнутості кривої у напрямку цього вектора, як показано на рис. 3.27.

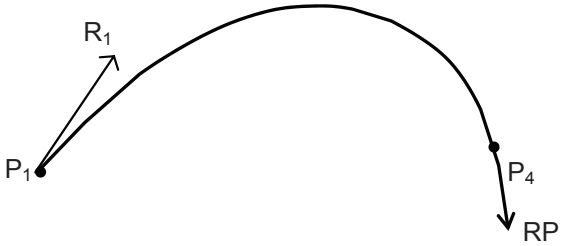


Рис. 3.27. Параметричний сплайн у формі Ерміта

Витягнутість кривої вправо забезпечується тим, що $|R_1|>|R_4|$. Розглянемо форму Без'є, що відрізняється від форми Ерміта способом завдання граничних умов, а саме, замість векторів R_1іR_4 вводяться точки (і відповідні їм радіус-вектори) P_2іP_3, як показано на рис. 3.28, так що виконуються умови: $P'(0)=R_1=(P_2-P_1)$ і $P'(1)=R_4=3(P_4-P_3)$.

$$C_x = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_{1x} \\ P_{4x} \\ R_{1x} \\ R_{4x} \end{bmatrix} = M_h G_{hx}$$

Тут M_h - ермітова матриця, G_h - геометричний вектор Ерміта. Підставимо вираз C_x для знаходження $x(t)$: $x(t)=TM_hG_{hx}$. Аналогічно для інших координат: $y(t)=TM_hG_{hy}$, $z(t)=TM_hG_{hz}$.

Випишемо в явному виді формули для обчислення координат точок сплайна. Враховуючи що $TM_h=[(2t^3-3t^2+1),(-2t^3+3t^2),(t^3-2t^2+t),(t^3-t^2)]$, та домножуючи праву частину рівняння на G_{hx}, одержуємо:

$$x(t) = TM_h G_{hx} =$$
$$= P_{1x}\left(2t^3 - 3t^2 + 1\right) + P_{4x}\left(-2t^3 + 3t^2\right) + R_{1x}\left(t^3 - 2t^2 + t\right) + R_{4x}\left(t^3 - t^2\right).$$

Чотири функції в дужках називаються функціями сполучення.

Форму кривої, заданої у формі Ерміта, легко змінювати, якщо враховувати, що напрямок дотичного вектора задає початковий напрямок, а модуль дотичного вектора задає ступінь витягнутості кривої у напрямку цього вектора, як показано на рис. 3.27.

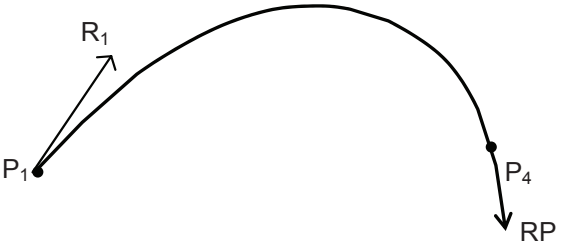


Рис. 3.27. Параметричний сплайн у формі Ерміта

Витягнутість кривої вправо забезпечується тим, що $|R_1|>|R_4|$. Розглянемо форму Без'є, що відрізняється від форми Ерміта способом завдання граничних умов, а саме, замість векторів R_1іR_4 вводяться точки (і відповідні їм радіус-вектори) P_2іP_3, як показано на рис. 3.28, так що виконуються умови: $P'(0)=R_1=(P_2-P_1)$ і $P'(1)=R_4=3(P_4-P_3)$.

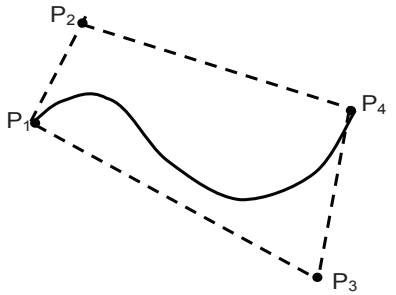


Рис. 3.28. Параметричний сплайн у формі Без'є

Перехід від форми Ерміта до форми Без'є здійснюється перетворенням:

$$G_h = \begin{bmatrix} P_1 \\ P_4 \\ R_1 \\ R_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix} = M_{hb} G_b$$

де G_b - геометричний вектор Без'є. Підставляючи це у вираз для $x(t)$, отримаємо

$$x(t) = TM_h G_{hx} = TM_h M_{hb} G_{bx} = \left(1 - t^3\right) P_1 + 3t(t-1)^2 P_2 + 3t^2(1-t) P_3 + t^3 P_4 .$$

Корисною властивістю сплайнів у формі Без'є є те, що крива завжди лежить усередині опуклої оболонки, утвореної чотирикутником $(P_1P_2P_3P_4)$. Цю властивість можна довести, користуючись тим, що коефіцієнти приймають значення від 0 до 1 і їх сума дорівнює одиниці.

Помітимо, що матриця виду

$$M_h M_{hb} = M_b = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

– називається матрицею Без'є.

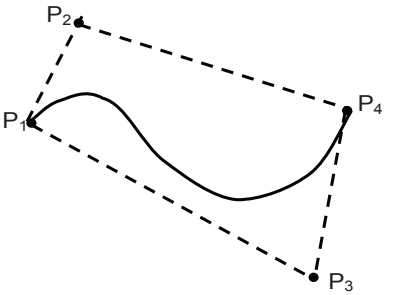


Рис. 3.28. Параметричний сплайн у формі Без'є

Перехід від форми Ерміта до форми Без'є здійснюється перетворенням:

$$G_h = \begin{bmatrix} P_1 \\ P_4 \\ R_1 \\ R_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix} = M_{hb} G_b$$

де G_b - геометричний вектор Без'є. Підставляючи це у вираз для $x(t)$, отримаємо

$$x(t) = TM_h G_{hx} = TM_h M_{hb} G_{bx} = \left(1 - t^3\right) P_1 + 3t(t-1)^2 P_2 + 3t^2(1-t) P_3 + t^3 P_4 .$$

Корисною властивістю сплайнів у формі Без'є є те, що крива завжди лежить усередині опуклої оболонки, утвореної чотирикутником $(P_1P_2P_3P_4)$. Цю властивість можна довести, користуючись тим, що коефіцієнти приймають значення від 0 до 1 і їх сума дорівнює одиниці.

Помітимо, що матриця виду

$$M_h M_{hb} = M_b = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

– називається матрицею Без'є.

Контрольні питання

1. Охарактеризуйте практичну цінність задачі моделювання реалістичної освітленості зображень та сформулюйте сутність найвідоміших методів розв'язування цієї задачі.
2. Виконайте порівняльний аналіз методів висвітлення й відбиття світла від поверхонь: розсіюваного, дифузійного, дзеркального.
3. Застосуйте метод зворотного трасування променів до спрощеної 3D-сцени, що складається з одного точкового джерела світла та однієї дзеркальної плаского поверхні, розташованих одне відносно одної довільним чином.
4. Яку роль відіграє метод геометричного векторно-параметричного моделювання в автоматизованому проектуванні криволінійних обводів?
5. Для яких класів кривих та поверхонь найчастіше застосовують векторно-параметричне моделювання?
6. Дайте визначення поняттям апроксимації, інтерполяції, згладжування, геометричного сплайну та сплайн-функції. Охарактеризуйте їх практичне призначення.
7. Що являє собою крива Без'є та які властивості вона має?
8. В чому полягає основний недолік кривих Без'є?

Тестові завдання

Питання 1. В матриці перетворень

$$T = \begin{bmatrix} & & 3 \\ & 3\times 3 & \times \\ & & 1 \\ 1\times 3 & & 1\times 1 \end{bmatrix}$$

підматриця (3×3):

А) задає переміщення

Б) призначена для перспективного перетворення

В) задає лінійне перетворення у формі масштабування, зрушення, відображення і обертання

Питання 2. В матриці перетворень

$$T = \begin{bmatrix} & & 3 \\ & 3\times 3 & \times \\ & & 1 \\ 1\times 3 & & 1\times 1 \end{bmatrix}$$

підматриця (3×1)задає:

А) перспективне перетворення

Б) переміщення

В) лінійне перетворення у формі масштабування, зрушення, відображення і обертання

Питання 3. Матриця

$$T = \begin{bmatrix} \cos\psi & \sin\psi & 0 & 0 \\ -\sin\psi & \cos\psi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

призначена для обертання навколо:

А) осі z на кут ψ

Б) осі y на кут ψ

В) осі x на кут ψ

Г) початку координат на кут ψ

Контрольні питання

1. Охарактеризуйте практичну цінність задачі моделювання реалістичної освітленості зображень та сформулюйте сутність найвідоміших методів розв'язування цієї задачі.
2. Виконайте порівняльний аналіз методів висвітлення й відбиття світла від поверхонь: розсіюваного, дифузійного, дзеркального.
3. Застосуйте метод зворотного трасування променів до спрощеної 3D-сцени, що складається з одного точкового джерела світла та однієї дзеркальної плаского поверхні, розташованих одне відносно одної довільним чином.
4. Яку роль відіграє метод геометричного векторно-параметричного моделювання в автоматизованому проектуванні криволінійних обводів?
5. Для яких класів кривих та поверхонь найчастіше застосовують векторно-параметричне моделювання?
6. Дайте визначення поняттям апроксимації, інтерполяції, згладжування, геометричного сплайну та сплайн-функції. Охарактеризуйте їх практичне призначення.
7. Що являє собою крива Без'є та які властивості вона має?
8. В чому полягає основний недолік кривих Без'є?

Тестові завдання

Питання 1. В матриці перетворень

$$T = \begin{bmatrix} & & 3 \\ & 3\times 3 & \times \\ & & 1 \\ 1\times 3 & & 1\times 1 \end{bmatrix}$$

підматриця (3×3):

А) задає переміщення

Б) призначена для перспективного перетворення

В) задає лінійне перетворення у формі масштабування, зрушення, відображення і обертання

Питання 2. В матриці перетворень

$$T = \begin{bmatrix} & & 3 \\ & 3\times 3 & \times \\ & & 1 \\ 1\times 3 & & 1\times 1 \end{bmatrix}$$

підматриця (3×1)задає:

А) перспективне перетворення

Б) переміщення

В) лінійне перетворення у формі масштабування, зрушення, відображення і обертання

Питання 3. Матриця

$$T = \begin{bmatrix} \cos\psi & \sin\psi & 0 & 0 \\ -\sin\psi & \cos\psi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

призначена для обертання навколо:

А) осі z на кут ψ

Б) осі y на кут ψ

В) осі x на кут ψ

Г) початку координат на кут ψ

Питання 4. Матриця

$$T = \begin{bmatrix} \cos\phi & 0 & -\sin\phi & 0 \\ 0 & 1 & 0 & 0 \\ \sin\phi & 0 & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

призначена для обертання навколо:

А) осі z на кут ψ
Б) осі y на кут ψ
В) осі x на кут ψ
Г) початку координат на кут ψ

Питання 5. Алгоритм Брезенхема застосовується для:

А) відсікання відрізків прямих
Б) розв'язування задачі растрового розгорнення відрізків прямих
В) для видалення невидимих граней

Питання 6. Алгоритм Коена-Сазерленда призначений для:

А) відсікання відрізків прямих
Б) розв'язування задачі растрового розгорнення відрізків прямих
В) видалення невидимих граней

Питання 7. Метод z-буфера використовується для:

А) відсікання відрізків прямих
Б) розв'язування задачі растрового розгорнення відрізків прямих
В) видалення невидимих граней

Питання 8. Математичними сплайнами називають функції, що використовуються для:

А) розрахунку освітлення
Б) апроксимації кривих
В) приховування невидимих граней

Питання 9. Основний недолік кривих Без`є полягає в тому, що

А) крива завжди лежить усередині опуклої оболонки
Б) неможливо вносити локальні зміни у форму кривої
В) для них характерні достатньо складні математичні розрахунки

Питання 10. Обов'язковою умовою для кривих Без`є є те, що крива не повинна проходити

А) як мінімум через дві будь-які опорні точки
Б) через опорні точки
В) через першу та кінцеву опорні точки

Питання 4. Матриця

$$T = \begin{bmatrix} \cos\phi & 0 & -\sin\phi & 0 \\ 0 & 1 & 0 & 0 \\ \sin\phi & 0 & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

призначена для обертання навколо:

А) осі z на кут ψ
Б) осі y на кут ψ
В) осі x на кут ψ
Г) початку координат на кут ψ

Питання 5. Алгоритм Брезенхема застосовується для:

А) відсікання відрізків прямих
Б) розв'язування задачі растрового розгорнення відрізків прямих
В) для видалення невидимих граней

Питання 6. Алгоритм Коена-Сазерленда призначений для:

А) відсікання відрізків прямих
Б) розв'язування задачі растрового розгорнення відрізків прямих
В) видалення невидимих граней

Питання 7. Метод z-буфера використовується для:

А) відсікання відрізків прямих
Б) розв'язування задачі растрового розгорнення відрізків прямих
В) видалення невидимих граней

Питання 8. Математичними сплайнами називають функції, що використовуються для:

А) розрахунку освітлення
Б) апроксимації кривих
В) приховування невидимих граней

Питання 9. Основний недолік кривих Без`є полягає в тому, що

А) крива завжди лежить усередині опуклої оболонки
Б) неможливо вносити локальні зміни у форму кривої
В) для них характерні достатньо складні математичні розрахунки

Питання 10. Обов'язковою умовою для кривих Без`є є те, що крива не повинна проходити

А) як мінімум через дві будь-які опорні точки
Б) через опорні точки
В) через першу та кінцеву опорні точки

Глава 4
Формати графічних файлів та алгоритми стиснення

Практично кожна прикладна програма створює і зберігає деякі види графічних даних. Навіть найпростіші текстові редактори дозволяють створювати лінії за допомогою символів ASCII або керуючих послідовностей терміналу. Широко поширені в останні роки програми, засновані на GUI (Graphic User Interface - графічний інтерфейс користувача), сьогодні повинні підтримувати змішані формати, щоб можна було включати растрові дані у текстові документи. Програми управління базами даних, що дозволяють працювати із зображеннями, теж уміють зберігати в одному файлі і текст, і растрові дані. Крім того, графічні файли - важливий "транспортний засіб", що забезпечує обмін візуальними даними між програмами та комп'ютерними системами.

Тому в рамках комп’ютерної графіки стрімко розвивається відносно нова область – алгоритми архівації зображень [6, 14, 18]. Поява цієї області обумовлена тим, що зображення – це своєрідний тип даних, для якого характерні три особливості:

1. Зображення (як і відео) займає набагато більше місця в пам’яті, ніж текст. Ця особливість зображень визначає актуальність алгоритмів архівації графіки.

2. Другою особливістю є те, що людський зір під час аналізу зображень оперує контурами, загальним переходом кольорів та практично нечутливий до незначних змін у зображенні. Це дозволило створити спеціальні алгоритми стиснення, які орієнтовані лише на зображення.

Глава 4
Формати графічних файлів та алгоритми стиснення

Практично кожна прикладна програма створює і зберігає деякі види графічних даних. Навіть найпростіші текстові редактори дозволяють створювати лінії за допомогою символів ASCII або керуючих послідовностей терміналу. Широко поширені в останні роки програми, засновані на GUI (Graphic User Interface - графічний інтерфейс користувача), сьогодні повинні підтримувати змішані формати, щоб можна було включати растрові дані у текстові документи. Програми управління базами даних, що дозволяють працювати із зображеннями, теж уміють зберігати в одному файлі і текст, і растрові дані. Крім того, графічні файли - важливий "транспортний засіб", що забезпечує обмін візуальними даними між програмами та комп'ютерними системами.

Тому в рамках комп’ютерної графіки стрімко розвивається відносно нова область – алгоритми архівації зображень [6, 14, 18]. Поява цієї області обумовлена тим, що зображення – це своєрідний тип даних, для якого характерні три особливості:

1. Зображення (як і відео) займає набагато більше місця в пам’яті, ніж текст. Ця особливість зображень визначає актуальність алгоритмів архівації графіки.

2. Другою особливістю є те, що людський зір під час аналізу зображень оперує контурами, загальним переходом кольорів та практично нечутливий до незначних змін у зображенні. Це дозволило створити спеціальні алгоритми стиснення, які орієнтовані лише на зображення.

3. Зображення, на відміну, від тексту, має надмірність у двох вимірах. Тобто, як правило, сусідні точки як по горизонталі, так і по вертикалі у зображенні близькі за кольором. Таким чином, при створенні алгоритму компресії графіки ми можемо використати особливості структури зображення.

Залежно від алгоритму, що застосовується для стиснення, існує велика кількість різноманітних форматів графічних файлів. Саме цьому і присвячено матеріал даної глави.

Форматом файла можна назвати сукупність методів, правил подання й розміщення даних [3]. Відповідно, ***формат графічних файлів*** – це набір методів, правил, призначених для подання, зберігання, обробки й розповсюдження зображень, наданих у цифровій формі.

4.1. Типи графічних форматів

Існує кілька різних типів графічних форматів, кожен з яких зберігає дані певним способом. У даний час найбільш широко використовуються растровий, векторний і метафайловий формати. Існують, однак, і інші типи форматів - формати сцени, анімації, мультимедіа, гібридні, гіпертекстові, гіпермедіа, об'ємні, мова моделювання віртуальної реальності (VRML), аудіоформати, формати шрифтів, мова опису сторінки (PDL) [3, 6].

Растрові формати

Растрові формати використовуються для зберігання растрових даних. Файли цього типу особливо добре підходять для зберігання реальних зображень, наприклад фотографій та відеозйомки. Растрові файли, по суті справи, містять точну попіксельно карту зображення. Програма візуалізації реконструює це зображення на поверхні відображення пристрою виводу.

Найбільш поширені растрові формати – це BMP, DIB, RLE, TIFF, PCX, PSD, JPEG, PNG, LWF, TGA, MrSID, RAW, IFF, CAM, CLP, CPT, CUR, DCM, DCX, IMG, FIF, KDC, LBM, KDC, PBM, PIC, PGM, RAS, RBS, PCD, PIC, РГХ, SUN, STING, XPM і ін.

Векторні формати

Файли векторного формату особливо корисні для зберігання лінійних елементів (ліній і багатокутників), а також елементів, які мо-

3. Зображення, на відміну, від тексту, має надмірність у двох вимірах. Тобто, як правило, сусідні точки як по горизонталі, так і по вертикалі у зображенні близькі за кольором. Таким чином, при створенні алгоритму компресії графіки ми можемо використати особливості структури зображення.

Залежно від алгоритму, що застосовується для стиснення, існує велика кількість різноманітних форматів графічних файлів. Саме цьому і присвячено матеріал даної глави.

Форматом файла можна назвати сукупність методів, правил подання й розміщення даних [3]. Відповідно, ***формат графічних файлів*** – це набір методів, правил, призначених для подання, зберігання, обробки й розповсюдження зображень, наданих у цифровій формі.

4.1. Типи графічних форматів

Існує кілька різних типів графічних форматів, кожен з яких зберігає дані певним способом. У даний час найбільш широко використовуються растровий, векторний і метафайловий формати. Існують, однак, і інші типи форматів - формати сцени, анімації, мультимедіа, гібридні, гіпертекстові, гіпермедіа, об'ємні, мова моделювання віртуальної реальності (VRML), аудіоформати, формати шрифтів, мова опису сторінки (PDL) [3, 6].

Растрові формати

Растрові формати використовуються для зберігання растрових даних. Файли цього типу особливо добре підходять для зберігання реальних зображень, наприклад фотографій та відеозйомки. Растрові файли, по суті справи, містять точну попіксельно карту зображення. Програма візуалізації реконструює це зображення на поверхні відображення пристрою виводу.

Найбільш поширені растрові формати – це BMP, DIB, RLE, TIFF, PCX, PSD, JPEG, PNG, LWF, TGA, MrSID, RAW, IFF, CAM, CLP, CPT, CUR, DCM, DCX, IMG, FIF, KDC, LBM, KDC, PBM, PIC, PGM, RAS, RBS, PCD, PIC, РГХ, SUN, STING, XPM і ін.

Векторні формати

Файли векторного формату особливо корисні для зберігання лінійних елементів (ліній і багатокутників), а також елементів, які мо-

жна розкласти на прості геометричні об'єкти (наприклад, текст). Векторні файли містять не піксельні значення, а математичні описи елементів зображень. За математичними описами графічних форм (ліній, кривих, сплайнів) програма візуалізації будує зображення.

Векторні файли структурно більш прості, ніж більшість растрових файлів, і зазвичай організовані у вигляді потоків даних.

Приклади найбільш поширених векторних форматів – AutoCAD DXF, Microsoft SYLK, Shockwave Flash і ін..

Метафайлові формати

Метафайли можуть зберігати і растрові, і векторні дані. Найпростіші метафайли нагадують файли векторного формату; вони містять мову або синтаксис для визначення елементів векторних даних, але можуть включати і растрове подання зображення. Метафайли часто використовуються для транспортування растрових і векторних даних між апаратними платформами, а також для переміщення зображень між програмними платформами.

Найбільш поширені метафайлові формати -CGM, PDF, EMF, WMF, графіка Excel, файли Adobe Table Editor, PLT- та HPGL-графіка, OLE-об'єкти, DXF-графіка, рисунки Lotus PIC та інші.

Формати сцени

Файли формату сцени (іноді звані файлами опису сцени) були розроблені для зберігання стислого зображення (або сцени). Векторні файли містять описи частин зображення, а файли сцени містять інструкції, що дозволяють програмі візуалізації відновити зображення цілком. На практиці іноді важко визначити, маємо ми справу з векторним форматом або з форматом сцени.

Формати анімації

Формати анімації з'явилися порівняно нещодавно. Вони створюються за тим же принципом, який ви використовували у своїх дитячих іграх з "рухомими" картинками. Якщо швидко відображати одне зображення за іншим, то створюється враження, що об'єкти цього зображення рухаються. Найпримітивніші з форматів анімації зберігають зображення в цілому, дозволяючи відображати їх просто в циклі одне за одним. Трохи більше ускладнені формати зберігають лише одне зображення і декілька колірних таблиць для даного зображення.

жна розкласти на прості геометричні об'єкти (наприклад, текст). Векторні файли містять не піксельні значення, а математичні описи елементів зображень. За математичними описами графічних форм (ліній, кривих, сплайнів) програма візуалізації будує зображення.

Векторні файли структурно більш прості, ніж більшість растрових файлів, і зазвичай організовані у вигляді потоків даних.

Приклади найбільш поширених векторних форматів – AutoCAD DXF, Microsoft SYLK, Shockwave Flash і ін..

Метафайлові формати

Метафайли можуть зберігати і растрові, і векторні дані. Найпростіші метафайли нагадують файли векторного формату; вони містять мову або синтаксис для визначення елементів векторних даних, але можуть включати і растрове подання зображення. Метафайли часто використовуються для транспортування растрових і векторних даних між апаратними платформами, а також для переміщення зображень між програмними платформами.

Найбільш поширені метафайлові формати -CGM, PDF, EMF, WMF, графіка Excel, файли Adobe Table Editor, PLT- та HPGL-графіка, OLE-об'єкти, DXF-графіка, рисунки Lotus PIC та інші.

Формати сцени

Файли формату сцени (іноді звані файлами опису сцени) були розроблені для зберігання стислого зображення (або сцени). Векторні файли містять описи частин зображення, а файли сцени містять інструкції, що дозволяють програмі візуалізації відновити зображення цілком. На практиці іноді важко визначити, маємо ми справу з векторним форматом або з форматом сцени.

Формати анімації

Формати анімації з'явилися порівняно нещодавно. Вони створюються за тим же принципом, який ви використовували у своїх дитячих іграх з "рухомими" картинками. Якщо швидко відображати одне зображення за іншим, то створюється враження, що об'єкти цього зображення рухаються. Найпримітивніші з форматів анімації зберігають зображення в цілому, дозволяючи відображати їх просто в циклі одне за одним. Трохи більше ускладнені формати зберігають лише одне зображення і декілька колірних таблиць для даного зображення.

Після завантаження нової колірної таблиці колір зображення змінюється і створюється ілюзія руху об'єктів. Ще більш складні формати анімації зберігають лише різницю між двома послідовно відображуваними зображеннями (званими фреймами) і змінюють тільки ті пікселі, які змінюються при відображенні даного фрейму. Відображення зі швидкістю 10-15 кадрів в секунду типове для анімації мультиплікаційного виду. У відеоанімації для створення ілюзії плавного руху необхідно відображати 20 і більше фреймів за секунду. Прикладами форматів анімації можуть служити TDDD і TTDDD.

Мультимедіа-формати

Мультимедіа-формати відносно нові, але набувають все більшого значення. Вони призначені для зберігання даних різних типів у одному файлі. Ці формати зазвичай дозволяють об'єднувати графічну, звукову і відеоінформацію. Прикладами можуть служити добре відомі формати RIFF фірми Microsoft, QuickTime фірми Apple, MPEG і FLI фірми Autodesk, а в найближчому майбутньому очікується поява нових форматів.

Змішані формати

В даний час широко досліджуються можливості об'єднання неструктурованого тексту і растрових даних (змішаний текст), а також інтеграції інформації, об'єднаної в записі, і растрових даних (змішана база даних). Ми припускаємо, що незабаром з'являться змішані формати, придатні для ефективного зберігання графічних даних.

Гіпертекст і гіпермедіа

Гіпертекст – це система, що забезпечує нелінійний доступ до інформації. Більшість книг побудовані за лінійним принципом: вони мають початок, закінчення і певну схему розміщення тексту. Гіпертекст дозволяє створювати документи з одним або декількома початками, з одним, декількома закінченнями або взагалі без нього, а також з безліччю гіпертекстових зв'язків, які допомагають читачеві "перестрибувати" у будь-яке місце документа.

Гіпертекстові мови не є форматами графічних файлів, як GIF або DXF. Це, швидше, мови програмування на зразок PostScript або C. Вони спеціально призначені для послідовної передачі потоків даних, тобто потік гіпертекстової інформації можна декодувати у міру

Після завантаження нової колірної таблиці колір зображення змінюється і створюється ілюзія руху об'єктів. Ще більш складні формати анімації зберігають лише різницю між двома послідовно відображуваними зображеннями (званими фреймами) і змінюють тільки ті пікселі, які змінюються при відображенні даного фрейму. Відображення зі швидкістю 10-15 кадрів в секунду типове для анімації мультиплікаційного виду. У відеоанімації для створення ілюзії плавного руху необхідно відображати 20 і більше фреймів за секунду. Прикладами форматів анімації можуть служити TDDD і TTDDD.

Мультимедіа-формати

Мультимедіа-формати відносно нові, але набувають все більшого значення. Вони призначені для зберігання даних різних типів у одному файлі. Ці формати зазвичай дозволяють об'єднувати графічну, звукову і відеоінформацію. Прикладами можуть служити добре відомі формати RIFF фірми Microsoft, QuickTime фірми Apple, MPEG і FLI фірми Autodesk, а в найближчому майбутньому очікується поява нових форматів.

Змішані формати

В даний час широко досліджуються можливості об'єднання неструктурованого тексту і растрових даних (змішаний текст), а також інтеграції інформації, об'єднаної в записі, і растрових даних (змішана база даних). Ми припускаємо, що незабаром з'являться змішані формати, придатні для ефективного зберігання графічних даних.

Гіпертекст і гіпермедіа

Гіпертекст – це система, що забезпечує нелінійний доступ до інформації. Більшість книг побудовані за лінійним принципом: вони мають початок, закінчення і певну схему розміщення тексту. Гіпертекст дозволяє створювати документи з одним або декількома початками, з одним, декількома закінченнями або взагалі без нього, а також з безліччю гіпертекстових зв'язків, які допомагають читачеві "перестрибувати" у будь-яке місце документа.

Гіпертекстові мови не є форматами графічних файлів, як GIF або DXF. Це, швидше, мови програмування на зразок PostScript або C. Вони спеціально призначені для послідовної передачі потоків даних, тобто потік гіпертекстової інформації можна декодувати у міру

отримання даних. Щоб переглянути гіпертекстовий документ, не потрібно чекати його повного завантаження.

Термін гіпермедіа позначає сплав гіпертексту і мультимедіа. Сучасні гіпертекстові мови і мережні протоколи підтримують найрізноманітніші засоби, включаючи текст і шрифти, нерухому і динамічну графіку, аудіо-, відео-і об'ємні дані. Гіпертекст забезпечує створення структури, яка дозволяє користувачеві комп'ютера організовувати мультимедіа дані, показувати їх та інтерактивно переміщатися по них.

Гіпертекстові і гіпермедійні системи, наприклад World Wide Web, зберігають великі інформаційні ресурси у вигляді файлів GIF, JPEG, PostScript, MPEG і AVI. Використовуються і багато інших форматів.

Тривимірні формати

У тривимірних файлах даних зберігається опис форми і кольору об'ємних моделей уявних і реальних об'єктів. Об'ємні моделі зазвичай конструюються з багатокутників і гладких поверхонь, об'єднаних з описами відповідних елементів: кольори, текстури, віддзеркалення і так далі, за допомогою яких програма візуалізації реконструює об'єкт. Моделі перетворюються в сцени з джерелами світла і камерами, тому об'єкти в тривимірних файлах часто називають елементами сцени.

Програми візуалізації, які користуються тривимірними даними, - це, як правило, програми моделювання та анімації (наприклад, Lightwave фірми NewTek і 3D Studio фірми Autodesk). Вони дозволяють коригувати зовнішній вигляд візуалізованого зображення, змінюючи і доповнюючи систему освітлення, текстуру елементів сцени та їх відносне розташування. Крім того, вони дають можливість користувачеві "оживляти" елементи сцени, тобто приписують їм рух. Після цього програма створює ряд растрових файлів (або кадрів), які, якщо взяти їх по порядку, збираються у фільм.

Важливо зрозуміти, що векторні дані є двовимірними. Це означає, що програма-творець цих даних не намагалася моделювати об'ємне зображення і передавати перспективу. До векторних даних відносяться креслення САПР і більшість ілюстративних вставок,

отримання даних. Щоб переглянути гіпертекстовий документ, не потрібно чекати його повного завантаження.

Термін гіпермедіа позначає сплав гіпертексту і мультимедіа. Сучасні гіпертекстові мови і мережні протоколи підтримують найрізноманітніші засоби, включаючи текст і шрифти, нерухому і динамічну графіку, аудіо-, відео-і об'ємні дані. Гіпертекст забезпечує створення структури, яка дозволяє користувачеві комп'ютера організовувати мультимедіа дані, показувати їх та інтерактивно переміщатися по них.

Гіпертекстові і гіпермедійні системи, наприклад World Wide Web, зберігають великі інформаційні ресурси у вигляді файлів GIF, JPEG, PostScript, MPEG і AVI. Використовуються і багато інших форматів.

Тривимірні формати

У тривимірних файлах даних зберігається опис форми і кольору об'ємних моделей уявних і реальних об'єктів. Об'ємні моделі зазвичай конструюються з багатокутників і гладких поверхонь, об'єднаних з описами відповідних елементів: кольори, текстури, віддзеркалення і так далі, за допомогою яких програма візуалізації реконструює об'єкт. Моделі перетворюються в сцени з джерелами світла і камерами, тому об'єкти в тривимірних файлах часто називають елементами сцени.

Програми візуалізації, які користуються тривимірними даними, - це, як правило, програми моделювання та анімації (наприклад, Lightwave фірми NewTek і 3D Studio фірми Autodesk). Вони дозволяють коригувати зовнішній вигляд візуалізованого зображення, змінюючи і доповнюючи систему освітлення, текстуру елементів сцени та їх відносне розташування. Крім того, вони дають можливість користувачеві "оживляти" елементи сцени, тобто приписують їм рух. Після цього програма створює ряд растрових файлів (або кадрів), які, якщо взяти їх по порядку, збираються у фільм.

Важливо зрозуміти, що векторні дані є двовимірними. Це означає, що програма-творець цих даних не намагалася моделювати об'ємне зображення і передавати перспективу. До векторних даних відносяться креслення САПР і більшість ілюстративних вставок,

призначених для настільних видавничих систем. На ринку існує деяка плутанина щодо поняття «об'ємна візуалізація». Ситуація ускладнюється тим, що тривимірні дані зараз підтримуються рядом форматів, які раніше служили тільки для зберігання двовимірних векторних даних. Приклад - формат DXF фірми Autodesk. Формати типу DXF іноді називають розширеними векторними форматами.

Формати мови моделювання віртуальної реальності (VRML)

VRML можна розглядати як гібрид об'ємної графіки і HTML. Формат VRML v.1.0 – це, по суті, файловий формат Silicon Graphics Inventor, в який додані засоби, що забезпечують з'єднання з URL у системі World Wide Web.

VRML кодує тривимірні дані у формат, придатний для обміну через Internet відповідно до протоколу Hypertext Transfer Protocol (HTTP). Отримані з Web-сервера VRML-дані відображаються в Web-браузері, який підтримує інтерпретатор мови VRML.

Формати аудіофайлів

Аудіоінформація зазвичай зберігається на магнітній стрічці у вигляді аналогових даних. Записам аудіоданих на такі носії, як компакт-диск (CD-ROM) і жорсткий диск, передує їх кодування за допомогою дискретизації, подібно до того, як це робиться під час запису цифрових відеоданих. Після кодування аудіодані можна записувати на диск як необроблений потік цифрових даних або, що зустрічається частіше, зберігати у форматі аудіофайла.

Формати аудіофайлів за своєю концепцією ідентичні форматам графічних файлів, тільки зберігається у них інформація, яка призначена не для очей, а для вух. Більшість форматів містять простий заголовок, який описує збережені у файлі аудіодані. Найчастіше в заголовку вказується кількість відліків за секунду, кількість каналів та кількість бітів на відлік. Ця інформація в першому наближенні відповідає даним про кількість відліків на піксел, кількість колірних площин і кількість бітів на відлік, що містяться в заголовках графічних файлів.

У форматах аудіофайлів застосовуються різні методи стиснення даних. Для 8-бітових графічних і звукових аудіоданих зазвичай використовується кодування за алгоритмом Хаффмена. А ось для 16-

призначених для настільних видавничих систем. На ринку існує деяка плутанина щодо поняття «об'ємна візуалізація». Ситуація ускладнюється тим, що тривимірні дані зараз підтримуються рядом форматів, які раніше служили тільки для зберігання двовимірних векторних даних. Приклад - формат DXF фірми Autodesk. Формати типу DXF іноді називають розширеними векторними форматами.

Формати мови моделювання віртуальної реальності (VRML)

VRML можна розглядати як гібрид об'ємної графіки і HTML. Формат VRML v.1.0 – це, по суті, файловий формат Silicon Graphics Inventor, в який додані засоби, що забезпечують з'єднання з URL у системі World Wide Web.

VRML кодує тривимірні дані у формат, придатний для обміну через Internet відповідно до протоколу Hypertext Transfer Protocol (HTTP). Отримані з Web-сервера VRML-дані відображаються в Web-браузері, який підтримує інтерпретатор мови VRML.

Формати аудіофайлів

Аудіоінформація зазвичай зберігається на магнітній стрічці у вигляді аналогових даних. Записам аудіоданих на такі носії, як компакт-диск (CD-ROM) і жорсткий диск, передує їх кодування за допомогою дискретизації, подібно до того, як це робиться під час запису цифрових відеоданих. Після кодування аудіодані можна записувати на диск як необроблений потік цифрових даних або, що зустрічається частіше, зберігати у форматі аудіофайла.

Формати аудіофайлів за своєю концепцією ідентичні форматам графічних файлів, тільки зберігається у них інформація, яка призначена не для очей, а для вух. Більшість форматів містять простий заголовок, який описує збережені у файлі аудіодані. Найчастіше в заголовку вказується кількість відліків за секунду, кількість каналів та кількість бітів на відлік. Ця інформація в першому наближенні відповідає даним про кількість відліків на піксел, кількість колірних площин і кількість бітів на відлік, що містяться в заголовках графічних файлів.

У форматах аудіофайлів застосовуються різні методи стиснення даних. Для 8-бітових графічних і звукових аудіоданих зазвичай використовується кодування за алгоритмом Хаффмена. А ось для 16-

бітових аудіоданих необхідні адаптовані спеціально для цих цілей алгоритми.

Формати шрифтів

Такі файли містять описи наборів буквено-цифрових знаків і символів у компактному, зручному для доступу форматі. З файлів шрифтів можна довільно вибирати дані, пов'язані з окремими знаками. У цьому сенсі вони являють собою бази даних про знаки і символи і тому іноді використовуються для зберігання графічних даних, хоч подібні дані за своєю природою не є буквено-цифровими або символьними. Файли шрифтів можуть мати (а можуть і не мати) загальні заголовки, а деякі файли підтримують навіть підзаголовки для кожного знака. У будь-якому випадку для того, щоб вибрати окремі знаки без читання і аналізу всього файла, потрібно знати початок даних про знаки, обсяг даних по кожному знаку і порядок, у якому ці знаки зберігаються.

Деякі файли шрифтів підтримують стиснення, а багато - шифрування даних. Історично склалися три основні типи файлів шрифтів: растрові, штрихові і сплайнові контурні [3].

Формати мов опису сторінки

Мови опису сторінки (PDL – Page Description Language) - це справжні машинні мови, що використовуються для опису компонування, шрифтів і графіки друкованих та відображуваних сторінок. PDL призначені для передачі інформації на пристрої друку (наприклад, принтери) і на пристрої відображення (такі, як екрани графічного інтерфейсу користувача GUI). Особливість цих мов полягає в тому, що коди PDL залежать від апаратних засобів. Типовий файл PostScript містить детальну інформацію про пристрій виведення. Файл PostScript з кодом для чотириколірного документа формату A4 може бути надрукований або відображений тільки на пристрої, який має засоби обробки цієї метрики.

А ось мови розмітки не містять інформації про пристрій виведення. Вони основані на тому, що пристрій, що візуалізує код на мові розмітки, зможе адаптуватися до переданих командам форматування. Програма візуалізації сама вибирає шрифти, кольори і ме-

бітових аудіоданих необхідні адаптовані спеціально для цих цілей алгоритми.

Формати шрифтів

Такі файли містять описи наборів буквено-цифрових знаків і символів у компактному, зручному для доступу форматі. З файлів шрифтів можна довільно вибирати дані, пов'язані з окремими знаками. У цьому сенсі вони являють собою бази даних про знаки і символи і тому іноді використовуються для зберігання графічних даних, хоч подібні дані за своєю природою не є буквено-цифровими або символьними. Файли шрифтів можуть мати (а можуть і не мати) загальні заголовки, а деякі файли підтримують навіть підзаголовки для кожного знака. У будь-якому випадку для того, щоб вибрати окремі знаки без читання і аналізу всього файла, потрібно знати початок даних про знаки, обсяг даних по кожному знаку і порядок, у якому ці знаки зберігаються.

Деякі файли шрифтів підтримують стиснення, а багато - шифрування даних. Історично склалися три основні типи файлів шрифтів: растрові, штрихові і сплайнові контурні [3].

Формати мов опису сторінки

Мови опису сторінки (PDL – Page Description Language) - це справжні машинні мови, що використовуються для опису компонування, шрифтів і графіки друкованих та відображуваних сторінок. PDL призначені для передачі інформації на пристрої друку (наприклад, принтери) і на пристрої відображення (такі, як екрани графічного інтерфейсу користувача GUI). Особливість цих мов полягає в тому, що коди PDL залежать від апаратних засобів. Типовий файл PostScript містить детальну інформацію про пристрій виведення. Файл PostScript з кодом для чотириколірного документа формату A4 може бути надрукований або відображений тільки на пристрої, який має засоби обробки цієї метрики.

А ось мови розмітки не містять інформації про пристрій виведення. Вони основані на тому, що пристрій, що візуалізує код на мові розмітки, зможе адаптуватися до переданих командам форматування. Програма візуалізації сама вибирає шрифти, кольори і ме-

тод відображення графічних даних. Мова розмітки дає тільки інформацію та відомості про її структуру. Мови опису сторінки істотно відрізняються від набагато більш простих аналізаторів, що застосовуються для читання графічних форматів.

4.2. Елементи графічного файла

Графічні файли складаються з послідовностей даних або структур даних, званих файловими елементами або елементами даних. Ці елементи підрозділяються на три категорії: поля, теги і потоки.

Поле – це структура даних у графічному файлі, що має фіксований розмір. Фіксоване поле може мати не тільки фіксований розмір, а й фіксовану позицію у файлі. Для визначення місця розташування поля задають або абсолютний зсув від орієнтиру у файлі, наприклад від початку або кінця файла, або відносне зміщення від будь-яких інших даних. Розмір поля може бути зазначений в специфікації формату або визначений за іншою інформацією.

Теги – це структура даних, розмір і позиція якої змінюються від файла до файла. Аналогічно полю, позиція тегів задається або абсолютним зміщенням щодо відомого орієнтира у файлі, або відносним зміщенням від іншого файлового елемента. Теги можуть містити в собі інші теги або набори пов'язаних полів.

Поля і теги спроектовані таким чином, щоб допомогти програмі одержати швидкий доступ до потрібних даних. Якщо позиція у файлі відома, то програма може отримати доступ до неї безпосередньо, без попереднього читання проміжних даних.

Файл, в якому дані організовані у вигляді *потоку*, не дає таких можливостей і повинен читатися послідовно. Ми припускаємо, що потік дозволяє підтримувати блоки даних змінної довжини, які є елементами потоку і повинні враховуватися програмою при читанні файла. Хоч початок і кінець потоку можуть бути відомі і задані, місце розташування блоків даних (за винятком першого) невідоме і визначається в процесі читання.

Файли, що складаються з елементів одного типу, – велика рідкість. Найчастіше застосовуються комбінації двох і більше елементів даних. Наприклад, формати TIFF і TGA використовують і теги, і фіксовані поля, а файли формату GIF - фіксовані поля і потоки.

тод відображення графічних даних. Мова розмітки дає тільки інформацію та відомості про її структуру. Мови опису сторінки істотно відрізняються від набагато більш простих аналізаторів, що застосовуються для читання графічних форматів.

4.2. Елементи графічного файла

Графічні файли складаються з послідовностей даних або структур даних, званих файловими елементами або елементами даних. Ці елементи підрозділяються на три категорії: поля, теги і потоки.

Поле – це структура даних у графічному файлі, що має фіксований розмір. Фіксоване поле може мати не тільки фіксований розмір, а й фіксовану позицію у файлі. Для визначення місця розташування поля задають або абсолютний зсув від орієнтиру у файлі, наприклад від початку або кінця файла, або відносне зміщення від будь-яких інших даних. Розмір поля може бути зазначений в специфікації формату або визначений за іншою інформацією.

Теги – це структура даних, розмір і позиція якої змінюються від файла до файла. Аналогічно полю, позиція тегів задається або абсолютним зміщенням щодо відомого орієнтира у файлі, або відносним зміщенням від іншого файлового елемента. Теги можуть містити в собі інші теги або набори пов'язаних полів.

Поля і теги спроектовані таким чином, щоб допомогти програмі одержати швидкий доступ до потрібних даних. Якщо позиція у файлі відома, то програма може отримати доступ до неї безпосередньо, без попереднього читання проміжних даних.

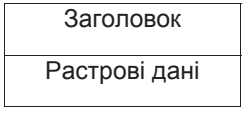
Файл, в якому дані організовані у вигляді *потоку*, не дає таких можливостей і повинен читатися послідовно. Ми припускаємо, що потік дозволяє підтримувати блоки даних змінної довжини, які є елементами потоку і повинні враховуватися програмою при читанні файла. Хоч початок і кінець потоку можуть бути відомі і задані, місце розташування блоків даних (за винятком першого) невідоме і визначається в процесі читання.

Файли, що складаються з елементів одного типу, – велика рідкість. Найчастіше застосовуються комбінації двох і більше елементів даних. Наприклад, формати TIFF і TGA використовують і теги, і фіксовані поля, а файли формату GIF - фіксовані поля і потоки.

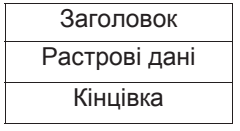
Організація даних у вигляді фіксованих полів дозволяє читати їх значно швидше, ніж дані, організовані у вигляді тегів або потоків. Однак файли, що містять в основному фіксовані поля, менш зручні при додаванні або видаленні даних. Формати, що складаються з фіксованих полів, складно розширити. Однак файли, що містять в основному потокові дані, не підтримують довільний доступ і, отже, не можуть бути використані для швидкого пошуку.

4.3. Растрові формати

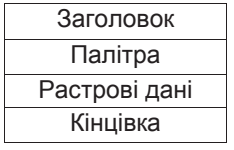
Растрові файли, відрізняючись один від одного деталями, мають загальну структуру. Растрові файли містять заголовок, растрові дані та іншу інформацію, в тому числі і про колірну палітру. З незрозумілої причини тривають розробки програм, що використовують так звані неструктуровані формати. Такі файли містять лише дані зображення, не залишаючи нам ніяких шансів розібратися в їх структурі. Про те, як організовані дані в цих файлах, мають уявлення тільки їх розробник і програма візуалізації, яка їх використовує. Основними компонентами простого растрового файла є:



Якщо файл не містить даних зображення, то повинен бути поданий тільки заголовок. Додаткову інформацію, яка не поміщається у заголовку, розміщують у кінцівці файла:



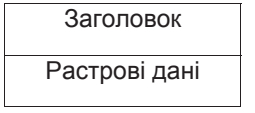
Якщо застосовується палітра, то її можна зберегти в заголовку файла, але зручніше розмістити її в середині файла, після заголовка:



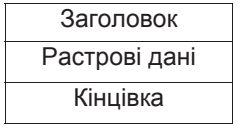
Організація даних у вигляді фіксованих полів дозволяє читати їх значно швидше, ніж дані, організовані у вигляді тегів або потоків. Однак файли, що містять в основному фіксовані поля, менш зручні при додаванні або видаленні даних. Формати, що складаються з фіксованих полів, складно розширити. Однак файли, що містять в основному потокові дані, не підтримують довільний доступ і, отже, не можуть бути використані для швидкого пошуку.

4.3. Растрові формати

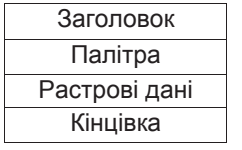
Растрові файли, відрізняючись один від одного деталями, мають загальну структуру. Растрові файли містять заголовок, растрові дані та іншу інформацію, в тому числі і про колірну палітру. З незрозумілої причини тривають розробки програм, що використовують так звані неструктуровані формати. Такі файли містять лише дані зображення, не залишаючи нам ніяких шансів розібратися в їх структурі. Про те, як організовані дані в цих файлах, мають уявлення тільки їх розробник і програма візуалізації, яка їх використовує. Основними компонентами простого растрового файла є:



Якщо файл не містить даних зображення, то повинен бути поданий тільки заголовок. Додаткову інформацію, яка не поміщається у заголовку, розміщують у кінцівці файла:



Якщо застосовується палітра, то її можна зберегти в заголовку файла, але зручніше розмістити її в середині файла, після заголовка:



Таблиці рядків розгортки і таблиці кольорової корекції можуть розташовуватися після заголовка як перед даними зображення, так і після них:

Заголовок
Палітра
Таблиця рядків розгортки
Таблиця колірної корекції
Растрові дані
Таблиця колірної корекції
Кінцівка

Заголовок – це розділ двійкових або символьних (у форматі ASCII) даних. Зазвичай він розташовується на початку файла і зберігає загальну інформацію про растрові дані, які в цьому файлі містяться. Всі структуровані растрові файли мають заголовки, структура і вміст яких визначаються конкретним форматом. Звичайно заголовок растрового файла складається з фіксованих полів. Жодне з цих полів не є обов'язковим, але певний набір полів типовий для більшості популярних на сьогоднішній день форматів. Нижче наведена інформація, що зазвичай міститься в заголовку:

Ідентифікатор файла
Версія файла
Кількість рядків у зображенні
Кількість пікселів у рядку
Кількість бітів у пікселі
Кількість кольорових площин
Тип стиснення
X-координата початку зображення
Y-координата початку зображення
Текстовий опис
Простір, що не використовується

Звичайно заголовок починається з певного унікального значення, що називається *ідентифікатор формату файла*, файловий ID або ID-значення. Ідентифікатор дозволяє програмі визначити формат графічного файла, з яким вона працює.

Як ідентифікатор може використовуватися послідовність символів ASCII (наприклад, BM або GIF), або дво- або чотирибайтове

Таблиці рядків розгортки і таблиці кольорової корекції можуть розташовуватися після заголовка як перед даними зображення, так і після них:

Заголовок
Палітра
Таблиця рядків розгортки
Таблиця колірної корекції
Растрові дані
Таблиця колірної корекції
Кінцівка

Заголовок – це розділ двійкових або символьних (у форматі ASCII) даних. Зазвичай він розташовується на початку файла і зберігає загальну інформацію про растрові дані, які в цьому файлі містяться. Всі структуровані растрові файли мають заголовки, структура і вміст яких визначаються конкретним форматом. Звичайно заголовок растрового файла складається з фіксованих полів. Жодне з цих полів не є обов'язковим, але певний набір полів типовий для більшості популярних на сьогоднішній день форматів. Нижче наведена інформація, що зазвичай міститься в заголовку:

Ідентифікатор файла
Версія файла
Кількість рядків у зображенні
Кількість пікселів у рядку
Кількість бітів у пікселі
Кількість кольорових площин
Тип стиснення
X-координата початку зображення
Y-координата початку зображення
Текстовий опис
Простір, що не використовується

Звичайно заголовок починається з певного унікального значення, що називається *ідентифікатор формату файла*, файловий ID або ID-значення. Ідентифікатор дозволяє програмі визначити формат графічного файла, з яким вона працює.

Як ідентифікатор може використовуватися послідовність символів ASCII (наприклад, BM або GIF), або дво- або чотирибайтове

слово (наприклад, 4242h або 596aa695h), або довільна послідовність даних, зрозуміла лише розробнику формату. Передбачається, що ідентифікатор повинен бути унікальним навіть для форматів, що використовуються на різних платформах, але, як ви побачите далі, ця умова далеко не завжди дотримується. Як правило, якщо значення, прочитане з певного місця у файлі, збігається з очікуваним ідентифікаційним значенням, то програма, що читає заголовок файла, припускає, що їй відомий даний формат.

Після ідентифікатора в заголовку файла зазвичай розташовується поле *версії файла*. Природно, версії одного і того ж формату можуть мати різні характеристики (розмір заголовка, підтримувані дані тощо). Тому після ідентифікації формату за допомогою ID-значення програма зазвичай перевіряє номер версії, щоб визначити, чи зможе вона обробити дані, що містяться в цьому файлі.

Після полів ідентифікатора і версії файла йдуть декілька полів, що описують саме зображення. Як правило, растри фізично або логічно організовані в рядки пікселів. Поле *кількість рядків* у растровому зображенні також називають довжиною зображення, висотою зображення або кількістю рядків розгортки, воно містить значення, що визначає кількість рядків у реальних растрових даних. Кількість пікселів у рядку, що також називається шириною зображення або шириною рядка розгортки, визначає кількість пікселів, збережених у кожному рядку.

Кількість бітів на піксел визначає розмір даних, необхідних для опису кожного піксела в кольоровій площині. Іноді в цьому полі насправді зберігається інформація про кількість байтів на піксел. Це поле характеризує піксельну глибину. Найбільш поширеною помилкою при розробці програм читання форматів є неправильна інтерпретація програмістами значення цього поля.

Якщо з метою зменшення обсягу файла формат підтримує який-небудь вид кодування, то в заголовок повинно бути включено поле *тип стиснення* (тип компресії). Деякі формати підтримують декілька алгоритмів компресії, включаючи необроблені і не стиснені дані. Часто модифікації форматів – це головним чином доповнені або змінені схеми стиснення. Проблема стиснення даних постійно привертає увагу розробників, тому досить часто з'являються нові, більш досконалі алгоритми компресії.

слово (наприклад, 4242h або 596aa695h), або довільна послідовність даних, зрозуміла лише розробнику формату. Передбачається, що ідентифікатор повинен бути унікальним навіть для форматів, що використовуються на різних платформах, але, як ви побачите далі, ця умова далеко не завжди дотримується. Як правило, якщо значення, прочитане з певного місця у файлі, збігається з очікуваним ідентифікаційним значенням, то програма, що читає заголовок файла, припускає, що їй відомий даний формат.

Після ідентифікатора в заголовку файла зазвичай розташовується поле *версії файла*. Природно, версії одного і того ж формату можуть мати різні характеристики (розмір заголовка, підтримувані дані тощо). Тому після ідентифікації формату за допомогою ID-значення програма зазвичай перевіряє номер версії, щоб визначити, чи зможе вона обробити дані, що містяться в цьому файлі.

Після полів ідентифікатора і версії файла йдуть декілька полів, що описують саме зображення. Як правило, растри фізично або логічно організовані в рядки пікселів. Поле *кількість рядків* у растровому зображенні також називають довжиною зображення, висотою зображення або кількістю рядків розгортки, воно містить значення, що визначає кількість рядків у реальних растрових даних. Кількість пікселів у рядку, що також називається шириною зображення або шириною рядка розгортки, визначає кількість пікселів, збережених у кожному рядку.

Кількість бітів на піксел визначає розмір даних, необхідних для опису кожного піксела в кольоровій площині. Іноді в цьому полі насправді зберігається інформація про кількість байтів на піксел. Це поле характеризує піксельну глибину. Найбільш поширеною помилкою при розробці програм читання форматів є неправильна інтерпретація програмістами значення цього поля.

Якщо з метою зменшення обсягу файла формат підтримує який-небудь вид кодування, то в заголовок повинно бути включено поле *тип стиснення* (тип компресії). Деякі формати підтримують декілька алгоритмів компресії, включаючи необроблені і не стиснені дані. Часто модифікації форматів – це головним чином доповнені або змінені схеми стиснення. Проблема стиснення даних постійно привертає увагу розробників, тому досить часто з'являються нові, більш досконалі алгоритми компресії.

Поля *X -координата початку зображення* і *Y -координата початку зображення* визначають координати точки початку зображення на пристрої виведення. Найчастіше вони мають значення (0, 0), що дозволяє поєднувати початок зображення з точкою відліку системи координат пристрою. Якщо ж застосовуються інші координати, то при візуалізації зображення почне відтворюватися з іншої точки.

Іноді в заголовок включають поле текстового опису растра. Це поле являє собою коментар, який містить довільні символьні (у форматі ASCII) дані, наприклад назва зображення, ім'я файла, ім'я автора зображення та / або ім'я програми, що використовується для його створення. Щоб забезпечити можливість переносити інформацію заголовка на інші платформи, це поле містить тільки 7-бітові дані у форматі ASCII.

У кінці заголовка можуть розташовуватися *невикористовувані поля*, що іноді називається набиванням, заповнювачами зарезервованого простору або зарезервованими полями. Зарезервовані поля не містять даних, не описуються і не структуруються. Їх розміри і місце розташування в заголовку відомі. Якщо виникне необхідність розширити формат, то відомості про нові дані заносяться в зарезервований простір. Таким чином зберігається сумісність з програмами, що підтримують старі версії цього формату, а проблеми, пов'язані з появою нових версій, зводяться до мінімуму. Заголовок формату має фіксовану довжину і структуру. У процесі модифікації формату його структура ускладнюється, при цьому нові поля додаються в зарезервовані області заголовка без зміни його розмірів. Часто заголовок спеціально розширюється зарезервованими полями до загальної довжини 128, 256 або 512 байтів. Це пов'язано з необхідністю враховувати розміри буферів введення / виведення для підвищення продуктивності системи. Наявність таких полів свідчить про те, що автор формату забув про проблеми продуктивності та кешування файлів.

Приклад 4.1. Заголовок формату Microsoft Windows Bitmap, версія 1.x

```
// Структура заголовка формату MS Windows Bitmap Format
// BYTE - це однобайтове беззнакове символьне значення
```


Поля *X -координата початку зображення* і *Y -координата початку зображення* визначають координати точки початку зображення на пристрої виведення. Найчастіше вони мають значення (0, 0), що дозволяє поєднувати початок зображення з точкою відліку системи координат пристрою. Якщо ж застосовуються інші координати, то при візуалізації зображення почне відтворюватися з іншої точки.

Іноді в заголовок включають поле текстового опису растра. Це поле являє собою коментар, який містить довільні символьні (у форматі ASCII) дані, наприклад назва зображення, ім'я файла, ім'я автора зображення та / або ім'я програми, що використовується для його створення. Щоб забезпечити можливість переносити інформацію заголовка на інші платформи, це поле містить тільки 7-бітові дані у форматі ASCII.

У кінці заголовка можуть розташовуватися *невикористовувані поля*, що іноді називається набиванням, заповнювачами зарезервованого простору або зарезервованими полями. Зарезервовані поля не містять даних, не описуються і не структуруються. Їх розміри і місце розташування в заголовку відомі. Якщо виникне необхідність розширити формат, то відомості про нові дані заносяться в зарезервований простір. Таким чином зберігається сумісність з програмами, що підтримують старі версії цього формату, а проблеми, пов'язані з появою нових версій, зводяться до мінімуму. Заголовок формату має фіксовану довжину і структуру. У процесі модифікації формату його структура ускладнюється, при цьому нові поля додаються в зарезервовані області заголовка без зміни його розмірів. Часто заголовок спеціально розширюється зарезервованими полями до загальної довжини 128, 256 або 512 байтів. Це пов'язано з необхідністю враховувати розміри буферів введення / виведення для підвищення продуктивності системи. Наявність таких полів свідчить про те, що автор формату забув про проблеми продуктивності та кешування файлів.

Приклад 4.1. Заголовок формату Microsoft Windows Bitmap, версія 1.x

```
// Структура заголовка формату MS Windows Bitmap Format
// BYTE - це однобайтове беззнакове символьне значення
```

```
// WORD - це коротке беззнакове ціле (16 бітів)
//
Typedef struct_WinHeaderlx
{
//
// Тип Ім'я Зсув Коментар
WORD Type / * OOh Ідентифікатор типу файла (завжди 0) * /
WORD Width; / * 02h Ширина растра в пікселах * /
WORD Height / * 04h Висота растра в пікселах * /
WORD Width; / * 06h Ширина растра в байтах * /
BYTE Planes / * 08h Кількість кольорових площин * /
BYTE BitsPixel; / * 09h Кількість бітів на піксел * /
} OLDWINHEAD;
```

Як видно з коментарів, цей заголовок містить ідентифікатор типу файла, а також інформацію про ширину і висоту зображення, ширину одного рядка зображення (у байтах), кількість колірних площин і кількість бітів на піксел. Це необхідний мінімум інформації, яка дозволяє описати растрове зображення таким чином, щоб воно могло бути прочитане і візуалізоване на довільно обраному обладнанні.

Зверніть увагу на те, що заголовок Windows Bitmap не містить інформації про кольори і стиснення даних зображення. Розширений формат зображень повинен передбачати можливості опису кольорової інформації та хоча б простої схеми стиснення.

Растрові дані зазвичай займають більшу частину растрового файла. У більшості форматів растрових файлів растрові дані розташовуються безпосередньо після заголовка, але можуть розміщуватися і в будь-якому іншому місці файла. Разом з ними можуть зберігатися палітра і інші дані. У такому випадку в заголовку в полі зсуву даних зображення (або в документації) вказується місце розташування початку даних зображення у файлі.

Структура растрових даних у більшості форматів досить проста. Растрові дані складаються з піксельних значень. На пристрої виводу піксели зазвичай виводяться у вигляді рядків розгорнення по всій ширині поверхні відображення, і цей факт, як правило, визначає порядок розташування даних у файлі. Тому інформація про


```
// WORD - це коротке беззнакове ціле (16 бітів)
//
Typedef struct_WinHeaderlx
{
//
// Тип Ім'я Зсув Коментар
WORD Type / * OOh Ідентифікатор типу файла (завжди 0) * /
WORD Width; / * 02h Ширина растра в пікселах * /
WORD Height / * 04h Висота растра в пікселах * /
WORD Width; / * 06h Ширина растра в байтах * /
BYTE Planes / * 08h Кількість кольорових площин * /
BYTE BitsPixel; / * 09h Кількість бітів на піксел * /
} OLDWINHEAD;
```

Як видно з коментарів, цей заголовок містить ідентифікатор типу файла, а також інформацію про ширину і висоту зображення, ширину одного рядка зображення (у байтах), кількість колірних площин і кількість бітів на піксел. Це необхідний мінімум інформації, яка дозволяє описати растрове зображення таким чином, щоб воно могло бути прочитане і візуалізоване на довільно обраному обладнанні.

Зверніть увагу на те, що заголовок Windows Bitmap не містить інформації про кольори і стиснення даних зображення. Розширений формат зображень повинен передбачати можливості опису кольорової інформації та хоча б простої схеми стиснення.

Растрові дані зазвичай займають більшу частину растрового файла. У більшості форматів растрових файлів растрові дані розташовуються безпосередньо після заголовка, але можуть розміщуватися і в будь-якому іншому місці файла. Разом з ними можуть зберігатися палітра і інші дані. У такому випадку в заголовку в полі зсуву даних зображення (або в документації) вказується місце розташування початку даних зображення у файлі.

Структура растрових даних у більшості форматів досить проста. Растрові дані складаються з піксельних значень. На пристрої виводу піксели зазвичай виводяться у вигляді рядків розгорнення по всій ширині поверхні відображення, і цей факт, як правило, визначає порядок розташування даних у файлі. Тому інформація про

те, на який пристрій виводу орієнтувався розробник формату, можливо, допоможе вам "вирахувати" точний порядок розташування даних.

Рядки розгортки об'єднують піксельні дані в двомірну сітку, що дозволяє нам розглядати розташування кожного піксела растра в заданих логічних координатах. Растр може бути поданий і у вигляді послідовності значень, які логічно відображають у файлі растрові дані, відповідні картинці на поверхні відображення пристрою виведення. Реальні растрові дані зазвичай складають більшу частину будь-якого растрового файла.

Кінцівка, що іноді називається хвостом, являє собою структуру даних, яка часто доповнює основний заголовок, але розташовується в кінці файла. Зазвичай кінцівку додають у тих випадках, коли файловий формат модифікувався (у нього були включені нові типи даних), а розширити або змінити структуру заголовка неможливо. Як правило, кінцівка додається для того, щоб зберегти сумісність формату з його попередніми версіями.

Назва цього елемента структури однозначно вказує на те, що він розташовується після даних зображення змінної довжини. Отже, кінцівка ніколи не має постійного зміщення від початку файла (виключаючи зображення незмінного розміру). Тому зміщення кінцівки, як правило, задається щодо кінця файла.

Зсув кінцівки може бути зазначено в інформації заголовка за умови, що в ньому є вільний простір. Кінцівка, як і заголовок, може містити поле ідентифікатора або спеціальне число, використовуване програмою візуалізації для того, щоб відрізнити її від інших структурних елементів растрового файла.

Крім заголовків, кінцівок і палітр, растрові файли можуть містити додаткові структури даних, які використовуються програмою візуалізації при різних маніпуляціях з даними зображення.

Растрові формати один від одного відрізняються такими властивостями: колірними моделями (глава 1), методами ущільнення, максимальним розміром зображення, який вони можуть забезпечити, шарами різних типів, наявністю Alpha-каналу, можливістю здійснювати анімацію, наявністю черезрядкового розгорнення тощо. Деякі характеристики популярних растрових форматів наведено у табл. 4.1.

те, на який пристрій виводу орієнтувався розробник формату, можливо, допоможе вам "вирахувати" точний порядок розташування даних.

Рядки розгортки об'єднують піксельні дані в двомірну сітку, що дозволяє нам розглядати розташування кожного піксела растра в заданих логічних координатах. Растр може бути поданий і у вигляді послідовності значень, які логічно відображають у файлі растрові дані, відповідні картинці на поверхні відображення пристрою виведення. Реальні растрові дані зазвичай складають більшу частину будь-якого растрового файла.

Кінцівка, що іноді називається хвостом, являє собою структуру даних, яка часто доповнює основний заголовок, але розташовується в кінці файла. Зазвичай кінцівку додають у тих випадках, коли файловий формат модифікувався (у нього були включені нові типи даних), а розширити або змінити структуру заголовка неможливо. Як правило, кінцівка додається для того, щоб зберегти сумісність формату з його попередніми версіями.

Назва цього елемента структури однозначно вказує на те, що він розташовується після даних зображення змінної довжини. Отже, кінцівка ніколи не має постійного зміщення від початку файла (виключаючи зображення незмінного розміру). Тому зміщення кінцівки, як правило, задається щодо кінця файла.

Зсув кінцівки може бути зазначено в інформації заголовка за умови, що в ньому є вільний простір. Кінцівка, як і заголовок, може містити поле ідентифікатора або спеціальне число, використовуване програмою візуалізації для того, щоб відрізнити її від інших структурних елементів растрового файла.

Крім заголовків, кінцівок і палітр, растрові файли можуть містити додаткові структури даних, які використовуються програмою візуалізації при різних маніпуляціях з даними зображення.

Растрові формати один від одного відрізняються такими властивостями: колірними моделями (глава 1), методами ущільнення, максимальним розміром зображення, який вони можуть забезпечити, шарами різних типів, наявністю Alpha-каналу, можливістю здійснювати анімацію, наявністю черезрядкового розгорнення тощо. Деякі характеристики популярних растрових форматів наведено у табл. 4.1.

Таблиця 4.1

Характеристики популярних растрових форматів

Формат	Фірма-розробник	Максимальна кількість кольорів (бітів на піксел)	Максимальний розмір зображення	Метод ущільнення	Запис декількох зображень
BMP	Microsoft	16'777'216 (24 біти)	65535×65535	RLE (використовується рідко)	–
GIF	CompuServe	256 (8 бітів)	65535×65535	LZW	+
JPEG	Joint Photographic Experts Group	16'777'216 (24 біти)	65535×65535	JPEG	–
PCX	Z-Soft	16'777'216 (24-біти)	65535×65535	RLE	–
PNG	W3C	281'474'976710'656 (48 бітів)	214748364×2147483647	Deflate	–
TGA	Truevision	4'294'967'296 (32 біти)	65535×65535	RLE	–
TIFF	Aldus Corporation	16'777'216 (24-біти)	усього 4'294'967'295	LZW, RLE, JPEG, та інші	+

Формат BMP

Це один з перших растрових форматів. Формат відрізняє дуже великий обсяг файла, оскільки дані записуються по кожному пікселу окремо. Він являє собою надзвичайно просту структуру і служить для опису та візуалізації невеликих зображень-піктограм (icons), широко застосовується у графічних інтерфейсах Windows, а також використовується в мультимедійних презентаціях.

Існує кілька різновидів цього формату. Нам найбільше знайомий варіант з розширенням *. bmp.

Формат Bitmap32

Це порівняно новий формат, створений на базі формату BMP, від якого відрізняється тим, що дані про одну точку зберігаються не в 24, а в 32 бітах. Додаткові 8 бітів використовуються для Alpha-каналу. Формат поки що не одержав поширення, але після появи Windows XP, де Alpha-канал був узаконений на рівні ядра системи, формат отримав гарні перспективи на майбутнє.

Таблиця 4.1

Характеристики популярних растрових форматів

Формат	Фірма-розробник	Максимальна кількість кольорів (бітів на піксел)	Максимальний розмір зображення	Метод ущільнення	Запис декількох зображень
BMP	Microsoft	16'777'216 (24 біти)	65535×65535	RLE (використовується рідко)	–
GIF	CompuServe	256 (8 бітів)	65535×65535	LZW	+
JPEG	Joint Photographic Experts Group	16'777'216 (24 біти)	65535×65535	JPEG	–
PCX	Z-Soft	16'777'216 (24-біти)	65535×65535	RLE	–
PNG	W3C	281'474'976710'656 (48 бітів)	214748364×2147483647	Deflate	–
TGA	Truevision	4'294'967'296 (32 біти)	65535×65535	RLE	–
TIFF	Aldus Corporation	16'777'216 (24-біти)	усього 4'294'967'295	LZW, RLE, JPEG, та інші	+

Формат BMP

Це один з перших растрових форматів. Формат відрізняє дуже великий обсяг файла, оскільки дані записуються по кожному пікселу окремо. Він являє собою надзвичайно просту структуру і служить для опису та візуалізації невеликих зображень-піктограм (icons), широко застосовується у графічних інтерфейсах Windows, а також використовується в мультимедійних презентаціях.

Існує кілька різновидів цього формату. Нам найбільше знайомий варіант з розширенням *. bmp.

Формат Bitmap32

Це порівняно новий формат, створений на базі формату BMP, від якого відрізняється тим, що дані про одну точку зберігаються не в 24, а в 32 бітах. Додаткові 8 бітів використовуються для Alpha-каналу. Формат поки що не одержав поширення, але після появи Windows XP, де Alpha-канал був узаконений на рівні ядра системи, формат отримав гарні перспективи на майбутнє.

Отже, цей формат зручний насамперед через збереження додаткової складової прозорості, яка зберігається всередині файла з текстурою.

Формат PCX

Формат ***PCX*** запропонований компанією Z-Soft у програмі Paintbrush, він може бути використаний на платформі Macintosh, хоча був написаний для PC. Цей формат застосовувався багатьма компаніями, що спеціалізуються в області програмного забезпечення. Він зручний для зберігання зображень типу ділової графіки (креслення, діаграми, схеми тощо). Підтримуються колірні формати 1, 4, 8 та 24 біти на піксел. До недоліків PCX слід віднести непристосованість до запису фотографій, а також наявність численних версій.

У форматі PCX використаний один із варіантів алгоритму ущільнення RLE (від англійського *Run Length Encoding*— групове кодування).

Формат TGA

Для підтримки своїх відеокарт фірмою Truevision був розроблений формат *Target Image File (TGA).* Кілька компаній час від часу перекупили права на цей формат одна в одної. З файлами TGA працювали адаптери Targa, True Vista й інші. Формат дозволяє зберігати зображення з глибиною кольору до 32 бітів і при цьому швидко читається й розпаковується. Має спеціальну опцію "Bottom-up orientation", тобто завантаження файла не "зверху вниз", а "знизу нагору".

У TGA використовується алгоритм ущільнення RLE, який відрізняється від варіанта алгоритму, що використовується у форматі PCX. Інший варіант алгоритму RLE має більший максимальний ступінь компресії для деяких растрів і не так сильно збільшує в розмірах вихідний файл у найгіршому випадку.

Формат GIF

Формат *GIF {Graphics Interchange Format)* — популярний растровий формат. У форматі GIF можна призначити один чи більше кольорів прозорими, вони стануть невидимими у браузерах Інтернету й у деяких інших програмах.

Отже, цей формат зручний насамперед через збереження додаткової складової прозорості, яка зберігається всередині файла з текстурою.

Формат PCX

Формат ***PCX*** запропонований компанією Z-Soft у програмі Paintbrush, він може бути використаний на платформі Macintosh, хоча був написаний для PC. Цей формат застосовувався багатьма компаніями, що спеціалізуються в області програмного забезпечення. Він зручний для зберігання зображень типу ділової графіки (креслення, діаграми, схеми тощо). Підтримуються колірні формати 1, 4, 8 та 24 біти на піксел. До недоліків PCX слід віднести непристосованість до запису фотографій, а також наявність численних версій.

У форматі PCX використаний один із варіантів алгоритму ущільнення RLE (від англійського *Run Length Encoding*— групове кодування).

Формат TGA

Для підтримки своїх відеокарт фірмою Truevision був розроблений формат *Target Image File (TGA).* Кілька компаній час від часу перекупили права на цей формат одна в одної. З файлами TGA працювали адаптери Targa, True Vista й інші. Формат дозволяє зберігати зображення з глибиною кольору до 32 бітів і при цьому швидко читається й розпаковується. Має спеціальну опцію "Bottom-up orientation", тобто завантаження файла не "зверху вниз", а "знизу нагору".

У TGA використовується алгоритм ущільнення RLE, який відрізняється від варіанта алгоритму, що використовується у форматі PCX. Інший варіант алгоритму RLE має більший максимальний ступінь компресії для деяких растрів і не так сильно збільшує в розмірах вихідний файл у найгіршому випадку.

Формат GIF

Формат *GIF {Graphics Interchange Format)* — популярний растровий формат. У форматі GIF можна призначити один чи більше кольорів прозорими, вони стануть невидимими у браузерах Інтернету й у деяких інших програмах.

Ще одна корисна якість цього формату — це підтримка анімаційних зображень. Файл GIF може містити не одне, а декілька растрових зображень, які браузери можуть довантажувати одне за одним із зазначеною у файлі частотою. Так досягається ілюзія руху (GIF-анімація).

Формат GIF добре підходить для створення невеликих і досить простих анімаційних фрагментів. GIF-файли не потребують значного місця для їхнього зберігання. Щоб створити анімацію, необхідно спочатку створити кожний окремий фрейм, наприклад, за допомогою пакетів Adobe або Corel. Потім необхідно компілювати ряд окремих фреймів у єдиний GIF-файл.

У форматі GIF для ущільнення растрів використовується алгоритм LZW.

Формат PNG

Формат ***PNG***– це новий графічний стандарт, створений для заміни формату GIF. Він розроблений спеціальним комітетом, очолюваним Томасом Бутеллем. Абревіатура PNG (вимовляється як "пінг") означає Portable Network Graphics. Як видно з назви, цей формат призначений спеціально для передачі зображень по мережах. Затверджено стандарт PNG і надруковано у 1996 році [3].

Цей формат не захищений патентами, не вимагає ліцензування й фінансових відрахувань і тому має широко розповсюджуватися – саме через патентування й жорстке ліцензування алгоритму LZW і виник PNG.

Спільні риси GIF та PNG:

- використання методів компресії без утрат;
- підтримка індексованих кольорів до 8 бітів на піксел;
- маска прозорості (Alpha-канал);
- забезпечення прогресуючого показу;
- окрім зображення, файл може містити також і текст.

Відмінності PNG від GIF:

- більша максимальна глибина кольору – до 48 бітів на піксел для зображень типу TraeColor, а для градацій сірого – до 16 бітів на піксел;
- повний Alpha-канал до 16 бітів на піксел;

Ще одна корисна якість цього формату — це підтримка анімаційних зображень. Файл GIF може містити не одне, а декілька растрових зображень, які браузери можуть довантажувати одне за одним із зазначеною у файлі частотою. Так досягається ілюзія руху (GIF-анімація).

Формат GIF добре підходить для створення невеликих і досить простих анімаційних фрагментів. GIF-файли не потребують значного місця для їхнього зберігання. Щоб створити анімацію, необхідно спочатку створити кожний окремий фрейм, наприклад, за допомогою пакетів Adobe або Corel. Потім необхідно компілювати ряд окремих фреймів у єдиний GIF-файл.

У форматі GIF для ущільнення растрів використовується алгоритм LZW.

Формат PNG

Формат ***PNG***– це новий графічний стандарт, створений для заміни формату GIF. Він розроблений спеціальним комітетом, очолюваним Томасом Бутеллем. Абревіатура PNG (вимовляється як "пінг") означає Portable Network Graphics. Як видно з назви, цей формат призначений спеціально для передачі зображень по мережах. Затверджено стандарт PNG і надруковано у 1996 році [3].

Цей формат не захищений патентами, не вимагає ліцензування й фінансових відрахувань і тому має широко розповсюджуватися – саме через патентування й жорстке ліцензування алгоритму LZW і виник PNG.

Спільні риси GIF та PNG:

- використання методів компресії без утрат;
- підтримка індексованих кольорів до 8 бітів на піксел;
- маска прозорості (Alpha-канал);
- забезпечення прогресуючого показу;
- окрім зображення, файл може містити також і текст.

Відмінності PNG від GIF:

- більша максимальна глибина кольору – до 48 бітів на піксел для зображень типу TraeColor, а для градацій сірого – до 16 бітів на піксел;
- повний Alpha-канал до 16 бітів на піксел;

• запис у файл гама-корекції;
• ефективне розпізнавання пошкоджень даних;
• у файл PNG базового формату записується тільки одне зображення – немає підтримки анімації, як у GIF (у наступних версіях PNG планується це змінити).

Формат JPEG

Розвиток технічної бази машинної графіки й засобів мультимедіа викликали швидке зростання розмірів графічних файлів. Для пошуку кращого способу запису великих обсягів графічної інформації при ***Міжнародному Комітеті зі стандартизації (ISO)*** була створена дослідницька група *Joint Photographic Experts Group (JPEG).* Робота групи завершилася пропозиціями по створенню файлового формату з високим ступенем ущільнення даних на основі алгоритму JPEG. У 1991 році з'явилися офіційні стандарти для JPEG – 10918-1, 2 [3, 6, 12].

Офіційно ім'я "JPEG" (вимовляється як "джейпег") вказує на алгоритм, метод ущільнення, а не на формат файла. Для уніфікації файлових форматів, що використовують метод JPEG, були розповсюджені рекомендації, які названо як JFIF (JPEG File Interchange Format) [3]. Отже, можливо, більшість файлів JPEG правильніше називати JFIF. Файли цього формату мають розширення .JPEG, JPG, JFIF. Крім того, метод ущільнення JPEG використовується в інших форматах файлів, наприклад, TIFF, Kodak Photo CD, QuickTime та інших.

Можна сказати, що метод JPEG відкрив шлях масовому розповсюдженню й використанню ефективних технологій роботи з повноколірними зображеннями. Це пояснюється не тільки позитивними рисами запропонованого алгоритму і споживчими якостями відповідних технологій, але й відкритістю стандарту, відсутністю патентних обмежень та підтримкою для розробників програм.

Наприклад, для підтримки програмістів, які розроблюють програми запису та читання файлів JPEG, групою Independent JPEG Group була створена бібліотека IJG, яка написана мовою C для майже всіх існуючих комп'ютерних платформ. Розповсюджується ця бібліотека безплатно [20].

• запис у файл гама-корекції;
• ефективне розпізнавання пошкоджень даних;
• у файл PNG базового формату записується тільки одне зображення – немає підтримки анімації, як у GIF (у наступних версіях PNG планується це змінити).

Формат JPEG

Розвиток технічної бази машинної графіки й засобів мультимедіа викликали швидке зростання розмірів графічних файлів. Для пошуку кращого способу запису великих обсягів графічної інформації при ***Міжнародному Комітеті зі стандартизації (ISO)*** була створена дослідницька група *Joint Photographic Experts Group (JPEG).* Робота групи завершилася пропозиціями по створенню файлового формату з високим ступенем ущільнення даних на основі алгоритму JPEG. У 1991 році з'явилися офіційні стандарти для JPEG – 10918-1, 2 [3, 6, 12].

Офіційно ім'я "JPEG" (вимовляється як "джейпег") вказує на алгоритм, метод ущільнення, а не на формат файла. Для уніфікації файлових форматів, що використовують метод JPEG, були розповсюджені рекомендації, які названо як JFIF (JPEG File Interchange Format) [3]. Отже, можливо, більшість файлів JPEG правильніше називати JFIF. Файли цього формату мають розширення .JPEG, JPG, JFIF. Крім того, метод ущільнення JPEG використовується в інших форматах файлів, наприклад, TIFF, Kodak Photo CD, QuickTime та інших.

Можна сказати, що метод JPEG відкрив шлях масовому розповсюдженню й використанню ефективних технологій роботи з повноколірними зображеннями. Це пояснюється не тільки позитивними рисами запропонованого алгоритму і споживчими якостями відповідних технологій, але й відкритістю стандарту, відсутністю патентних обмежень та підтримкою для розробників програм.

Наприклад, для підтримки програмістів, які розроблюють програми запису та читання файлів JPEG, групою Independent JPEG Group була створена бібліотека IJG, яка написана мовою C для майже всіх існуючих комп'ютерних платформ. Розповсюджується ця бібліотека безплатно [20].

Формат JPEG 2000

Новий стандарт ***JPEG* 2000** [20], у розробці якого взяли участь Міжнародна організація по стандартизації (International Organization for Standardization), Міжнародний союз телекомунікацій (International Telecommunications Union), компанії Agfa, Canon, Fujifilm, Hewlett Packard, Kodak, LuraTech, Motorola, Ricoh, Sony й інші, створювався як нова система кодування зображень із різними характеристиками (природні, наукові, медичні, текстові, модельовані тощо). Основною відмінністю JPEG2000 від попередньої версії цього формату є використання рекурсивного wavelet- перетворення замість дискретного косинусного перетворення.

Перетворення типу wavelet було відоме в математиці досить давно. Для реалізації такого перетворення в системах ущільнення необхідно вирішувати питання цифрової фільтрації та пірамідального подання інформації. Вагомий внесок у 90-х роках минулого сторіччя зробила *Інгрід Добеші* [22]. Відтоді значно активізувалися роботи в цьому напрямку, зокрема, розробка алгоритмів ущільнення зображень.

Формат TIFF

Формат ***TIFF*** *(Tagged Image File Format)* запропонований фірмою Aldus Corporation. Він був розроблений для зберігання сканованих зображень із високою роздільною здатністю. Висока роздільна здатність обумовлює великий об'єм файлів, що ускладнює їхню обробку. Основна ідея формату TIFF — це підтримка швидкого доступу до окремих фрагментів зображення. Це дозволяє редагувати зображення окремими частинами.

Позитивною рисою формату TIFF є його гнучкість. Він може зберігати декілька зображень. Можуть використовуватися різноманітні колірні моделі. Підтримується багато методів ущільнення — LZW, Deflate, JPEG та інші.

Отже, цей формат можна вважати стандартом обміну графічними даними. Однак насиченість можливостей обумовлює проблеми для розробників програм – важко урахувати всі можливості. Трапляється, що файл, записаний одною програмою, не читається іншими. Для вирішення цієї проблеми у стандарті TIFF версії 6.0 визначено підмножину Baseline TIFF, яку повинні підтримувати всі програми [22].

Формат JPEG 2000

Новий стандарт ***JPEG* 2000** [20], у розробці якого взяли участь Міжнародна організація по стандартизації (International Organization for Standardization), Міжнародний союз телекомунікацій (International Telecommunications Union), компанії Agfa, Canon, Fujifilm, Hewlett Packard, Kodak, LuraTech, Motorola, Ricoh, Sony й інші, створювався як нова система кодування зображень із різними характеристиками (природні, наукові, медичні, текстові, модельовані тощо). Основною відмінністю JPEG2000 від попередньої версії цього формату є використання рекурсивного wavelet- перетворення замість дискретного косинусного перетворення.

Перетворення типу wavelet було відоме в математиці досить давно. Для реалізації такого перетворення в системах ущільнення необхідно вирішувати питання цифрової фільтрації та пірамідального подання інформації. Вагомий внесок у 90-х роках минулого сторіччя зробила *Інгрід Добеші* [22]. Відтоді значно активізувалися роботи в цьому напрямку, зокрема, розробка алгоритмів ущільнення зображень.

Формат TIFF

Формат ***TIFF*** *(Tagged Image File Format)* запропонований фірмою Aldus Corporation. Він був розроблений для зберігання сканованих зображень із високою роздільною здатністю. Висока роздільна здатність обумовлює великий об'єм файлів, що ускладнює їхню обробку. Основна ідея формату TIFF — це підтримка швидкого доступу до окремих фрагментів зображення. Це дозволяє редагувати зображення окремими частинами.

Позитивною рисою формату TIFF є його гнучкість. Він може зберігати декілька зображень. Можуть використовуватися різноманітні колірні моделі. Підтримується багато методів ущільнення — LZW, Deflate, JPEG та інші.

Отже, цей формат можна вважати стандартом обміну графічними даними. Однак насиченість можливостей обумовлює проблеми для розробників програм – важко урахувати всі можливості. Трапляється, що файл, записаний одною програмою, не читається іншими. Для вирішення цієї проблеми у стандарті TIFF версії 6.0 визначено підмножину Baseline TIFF, яку повинні підтримувати всі програми [22].

158

Формат DjVu

DjVu (вимовляється "дежавю") — це комплект технологій ущільнення, формату файла й програмної платформи, що розроблявся з 1996 у фірмі AT&T Labs спеціально з метою розміщення в Інтернеті сканованих документів (книг, журналів, документації, зображень із високою роздільністю тощо).

Це відносно новий формат, що використовує хвильовий *(wavelet)* алгоритм ущільнення. Кінцевий результат ущільнення порівняний за своєю якістю з оригінальною сканованою версією. При багаторазовому його збільшенні не з'являється растрова мозаїка.

DjVu зберігає зображення, використовуючи три шари.

- Перший шар містить маску (бітовий масив), що вказує, яка точка зображення відповідає передньому плану (текст, малюнки, підписи) і яка – фону (текстура паперу, фотографії). Цей шар кодується алгоритмом без втрат.
- Другий шар містить інформацію про колір фону, використовуючи кодування, засноване на хвильовому ущільненні.
- Третій рівень містить інформацію про передній план.

Одна з основних технологій DjVu — здатність відокремити фон зображення й передній план. Традиційні методи ущільнення зображення придатні для простих фотографій, але вони значно погіршують різкі переходи кольору між суміжними контрастними областями. Відокремлюючи текст від фону, DjVu може зберегти текст із високою роздільністю (у такий спосіб зберігаючи гострі грані й максимізуючи чіткість), у той самий час ущільнюючи фон з більш низькою роздільною здатністю (використовуючи методику на основі хвильового алгоритму).

Розмір документів настільки малий, що вони можуть відправлятися електронною поштою як вкладення.

Формат LWF

Група розроблювачів, яка займається обробкою супутникових фотографій, створила у компанії LuraTech формат *LWF {Lura Tech Wavelet format)*. У форматі LWF використовується алгоритм ущільнення з утратами (на основі хвильового алгоритму), який часто дає кращі за JPEG результати. Формат LWF відрізняється тим, що при ущільненні можна заздалегідь установити розмір майбутнього файла.

Формат DjVu

DjVu (вимовляється "дежавю") — це комплект технологій ущільнення, формату файла й програмної платформи, що розроблявся з 1996 у фірмі AT&T Labs спеціально з метою розміщення в Інтернеті сканованих документів (книг, журналів, документації, зображень із високою роздільністю тощо).

Це відносно новий формат, що використовує хвильовий *(wavelet)* алгоритм ущільнення. Кінцевий результат ущільнення порівняний за своєю якістю з оригінальною сканованою версією. При багаторазовому його збільшенні не з'являється растрова мозаїка.

DjVu зберігає зображення, використовуючи три шари.

- Перший шар містить маску (бітовий масив), що вказує, яка точка зображення відповідає передньому плану (текст, малюнки, підписи) і яка – фону (текстура паперу, фотографії). Цей шар кодується алгоритмом без втрат.
- Другий шар містить інформацію про колір фону, використовуючи кодування, засноване на хвильовому ущільненні.
- Третій рівень містить інформацію про передній план.

Одна з основних технологій DjVu — здатність відокремити фон зображення й передній план. Традиційні методи ущільнення зображення придатні для простих фотографій, але вони значно погіршують різкі переходи кольору між суміжними контрастними областями. Відокремлюючи текст від фону, DjVu може зберегти текст із високою роздільністю (у такий спосіб зберігаючи гострі грані й максимізуючи чіткість), у той самий час ущільнюючи фон з більш низькою роздільною здатністю (використовуючи методику на основі хвильового алгоритму).

Розмір документів настільки малий, що вони можуть відправлятися електронною поштою як вкладення.

Формат LWF

Група розроблювачів, яка займається обробкою супутникових фотографій, створила у компанії LuraTech формат *LWF {Lura Tech Wavelet format)*. У форматі LWF використовується алгоритм ущільнення з утратами (на основі хвильового алгоритму), який часто дає кращі за JPEG результати. Формат LWF відрізняється тим, що при ущільненні можна заздалегідь установити розмір майбутнього файла.

Програма Lura Wave Smart Compress v2.2 (Shareware) може забезпечити високий ступінь ущільнення при досить високій якості. При ущільненні, наприклад у 60 разів, втрати менші, ніж при такому ж ущільненні в JPEG. Зображення практично не втрачає деталізації, а набуває дещо розмитого вигляду із підкресленими контурами, тобто якість сприйняття майже не погіршується. При збільшенні коефіцієнта ущільнення не проступає "мозаїка" квадратів, як у JPEG. Практично з'являється можливість інтерактивної роботи із сервером. Програма існує для всіх ОС. Цей формат використовують рідко, хоча LWF іноді ущільнює майже в 1,5 раза більше за інші формати [4, 22].

4.4. Векторні формати

Растровий файл містить точну попіксельну карту зображення, яке відтворюється програмою візуалізації на поверхні відображення пристрою виведення. Програми візуалізації рідко беруть до уваги будь-які структурні елементи растрових форматів, крім точок, рядків розгортки, смуг і фрагментів – частин зображення, створених без урахування його змісту

Векторні файли, навпаки, містять математичні описи всіх елементів зображення, що використовуються програмою візуалізації для конструювання кінцевого зображення. Таким чином, можна сказати, що векторні файли будуються не з піксельних значень, а з описів елементів зображення, або об'єктів.

Вектор - це відрізок прямої, заданий початковою точкою, напрямком і довжиною. Однак визначення вектора може бути більш складним і включати дані про тип лінії, кривої або сплайну. Прямі і криві лінії можуть застосовуватися для побудови геометричних фігур, таких як кола та прямокутники, які, у свою чергу, можуть використовуватися для створення більш складних об'ємних фігур – сфер, кубів і багатогранників.

Векторні формати відрізняються один від одного в більшій мірі, ніж растрові, тому що кожен з них проектувався для конкретних цілей Якщо концептуально формати, що підтримують 1-бітові та 24-бітові растрові дані, відрізняються незначно, то розходження між векторними форматами, які використовуються програмами САПР , та форматами, які використовуються для обміну загальними даними,

Програма Lura Wave Smart Compress v2.2 (Shareware) може забезпечити високий ступінь ущільнення при досить високій якості. При ущільненні, наприклад у 60 разів, втрати менші, ніж при такому ж ущільненні в JPEG. Зображення практично не втрачає деталізації, а набуває дещо розмитого вигляду із підкресленими контурами, тобто якість сприйняття майже не погіршується. При збільшенні коефіцієнта ущільнення не проступає "мозаїка" квадратів, як у JPEG. Практично з'являється можливість інтерактивної роботи із сервером. Програма існує для всіх ОС. Цей формат використовують рідко, хоча LWF іноді ущільнює майже в 1,5 раза більше за інші формати [4, 22].

4.4. Векторні формати

Растровий файл містить точну попіксельну карту зображення, яке відтворюється програмою візуалізації на поверхні відображення пристрою виведення. Програми візуалізації рідко беруть до уваги будь-які структурні елементи растрових форматів, крім точок, рядків розгортки, смуг і фрагментів – частин зображення, створених без урахування його змісту

Векторні файли, навпаки, містять математичні описи всіх елементів зображення, що використовуються програмою візуалізації для конструювання кінцевого зображення. Таким чином, можна сказати, що векторні файли будуються не з піксельних значень, а з описів елементів зображення, або об'єктів.

Вектор - це відрізок прямої, заданий початковою точкою, напрямком і довжиною. Однак визначення вектора може бути більш складним і включати дані про тип лінії, кривої або сплайну. Прямі і криві лінії можуть застосовуватися для побудови геометричних фігур, таких як кола та прямокутники, які, у свою чергу, можуть використовуватися для створення більш складних об'ємних фігур – сфер, кубів і багатогранників.

Векторні формати відрізняються один від одного в більшій мірі, ніж растрові, тому що кожен з них проектувався для конкретних цілей Якщо концептуально формати, що підтримують 1-бітові та 24-бітові растрові дані, відрізняються незначно, то розходження між векторними форматами, які використовуються програмами САПР , та форматами, які використовуються для обміну загальними даними,

будуть досить істотними. Отже, узагальнити векторні формати тим же способом, що і растрові, – завдання непросте.

З іншого боку, більшість пристроїв виведення підтримують сітку з пікселів, кожен з яких адресується окремо, так, наче поверхня відображення являла б собою папір у клітинку. Завдяки цьому програма завжди може знайти спосіб намалювати елементи зображення у векторному форматі.

Незважаючи на те, що векторні файли значно відрізняються один від одного, більшість з них, подібно растровим, має певну базову структуру: заголовок, розділ даних і маркер кінця файла. Така структура дозволяє коректно зберігати векторні дані та інтерпретувати їх при візуалізації. Загальна інформація, що описує структуру файла, звичайно міститься в заголовку, хоча іноді для цих цілей може використовуватися і кінцівка.

Структура векторних файлів, як правило, простіша, ніж у більшості растрових; часто вони організовані у вигляді потоків даних. Велика частина інформації, що міститься у файлі, це дані про зображення.

Формат DXF

Формат ***DXF*** *(Drawing Exchange Format)* розроблено фірмою AutoDesk у 1982 році для обміну кресленнями та іншими графічними документами в середовищі AutoCAD. Незважаючи на вік цього формату та його недоліки, DXF зараз підтримується багатьма програмами.

Головним недоліком формату DXF можна вважати великий обсяг файлів. У середовищі системи AutoCAD для роботи з документами використовується більш компактний формат – DWG, однак він є внутрішнім форматом, його не «розуміють» інші програми.

Формат MIF-MID

Тут ми дещо відхилимося від тематики "класичної комп'ютерної графіки" й розглянемо приклад векторного формату, який використовується в геоінформаційних системах (ГІС). Ці системи описують просторові об'єкти сукупністю метричних та атрибутивних (семантичних) даних. Формат MIF-MID є найпопулярнішим векторним форматом обміну даними для ГІС. Він розроблений фірмою MapInfo

будуть досить істотними. Отже, узагальнити векторні формати тим же способом, що і растрові, – завдання непросте.

З іншого боку, більшість пристроїв виведення підтримують сітку з пікселів, кожен з яких адресується окремо, так, наче поверхня відображення являла б собою папір у клітинку. Завдяки цьому програма завжди може знайти спосіб намалювати елементи зображення у векторному форматі.

Незважаючи на те, що векторні файли значно відрізняються один від одного, більшість з них, подібно растровим, має певну базову структуру: заголовок, розділ даних і маркер кінця файла. Така структура дозволяє коректно зберігати векторні дані та інтерпретувати їх при візуалізації. Загальна інформація, що описує структуру файла, звичайно міститься в заголовку, хоча іноді для цих цілей може використовуватися і кінцівка.

Структура векторних файлів, як правило, простіша, ніж у більшості растрових; часто вони організовані у вигляді потоків даних. Велика частина інформації, що міститься у файлі, це дані про зображення.

Формат DXF

Формат ***DXF*** *(Drawing Exchange Format)* розроблено фірмою AutoDesk у 1982 році для обміну кресленнями та іншими графічними документами в середовищі AutoCAD. Незважаючи на вік цього формату та його недоліки, DXF зараз підтримується багатьма програмами.

Головним недоліком формату DXF можна вважати великий обсяг файлів. У середовищі системи AutoCAD для роботи з документами використовується більш компактний формат – DWG, однак він є внутрішнім форматом, його не «розуміють» інші програми.

Формат MIF-MID

Тут ми дещо відхилимося від тематики "класичної комп'ютерної графіки" й розглянемо приклад векторного формату, який використовується в геоінформаційних системах (ГІС). Ці системи описують просторові об'єкти сукупністю метричних та атрибутивних (семантичних) даних. Формат MIF-MID є найпопулярнішим векторним форматом обміну даними для ГІС. Він розроблений фірмою MapInfo

для власної ГІС, однак зараз використовується майже усіма ГІС як формат експорту-імпорту. Опис просторових об'єктів у цьому форматі складається з двох файлів – *.MIF та *.MID. Файл із розширенням MIF містить загальний опис та координати вузлових точок об'єктів. Об'єкти можуть бути точковими, лінійними або площинними. Графічні примітиви: Arc, Ellipse, Line, Pline, Point, Rect, Region, Roundrect та Text. Більш детальну інформацію про структуру растрових та векторних файлів дивись у [20, 22].

4.5. Метафайли й інші формати

Під час сумісної роботи декількох програм часто виникає потреба в обміні різними графічними даними, як растровими, так і векторними. Для підтримки обміну такими даними використовуються спеціальні графічні формати – метафайли. Метафайли можуть зберігати інформацію про растрові та/або векторні зображення й про команди візуалізації. Як приклади метафайлів можна назвати файли CGM, EPS, PICT, WMF/EMF, графіка Excel, файли Adobe Table Editor, PLT- і HPGL-графіка, OLE-об'єкти, зображення Lotus PIC та інші зображення, вставлені командою PasteSpecial. У числі прикладів використання метафайлів в Інтернеті в першу чергу варто назвати формат PDF, який запропонований і активно просувається фірмою Adobe.

Слід зазначити, що важко відокремлювати метафайли та векторні формати. Вважаємо головною ознакою метафайла те, що графічний опис об'єктів складається з *команд* для певного графічного пристрою (принтера, драйвера GDI тощо). Можна сказати, що, мабуть, будь-який векторний формат – це метафайл. Наприклад, формат DXF містить команди(хоча команди не для пристрою, а для програми, що інтерпретує ці команди). Крім того, оскільки векторний формат містить опис зображення як список графічних примітивів, то важливою є послідовність накладання цих примітивів – знову можна сказати, що векторний формат — це не *список примітивів,* а послідовність *команд.* Будь-яка спроба догматично класифікувати реальні формати, які зазвичай містять багато різноманітних можливостей, вважається сумнівною. Для класифікації можна скористатися й таким принципом – якщо розробник формату називає це метафайлом, то, напевне, так воно й є.

для власної ГІС, однак зараз використовується майже усіма ГІС як формат експорту-імпорту. Опис просторових об'єктів у цьому форматі складається з двох файлів – *.MIF та *.MID. Файл із розширенням MIF містить загальний опис та координати вузлових точок об'єктів. Об'єкти можуть бути точковими, лінійними або площинними. Графічні примітиви: Arc, Ellipse, Line, Pline, Point, Rect, Region, Roundrect та Text. Більш детальну інформацію про структуру растрових та векторних файлів дивись у [20, 22].

4.5. Метафайли й інші формати

Під час сумісної роботи декількох програм часто виникає потреба в обміні різними графічними даними, як растровими, так і векторними. Для підтримки обміну такими даними використовуються спеціальні графічні формати – метафайли. Метафайли можуть зберігати інформацію про растрові та/або векторні зображення й про команди візуалізації. Як приклади метафайлів можна назвати файли CGM, EPS, PICT, WMF/EMF, графіка Excel, файли Adobe Table Editor, PLT- і HPGL-графіка, OLE-об'єкти, зображення Lotus PIC та інші зображення, вставлені командою PasteSpecial. У числі прикладів використання метафайлів в Інтернеті в першу чергу варто назвати формат PDF, який запропонований і активно просувається фірмою Adobe.

Слід зазначити, що важко відокремлювати метафайли та векторні формати. Вважаємо головною ознакою метафайла те, що графічний опис об'єктів складається з *команд* для певного графічного пристрою (принтера, драйвера GDI тощо). Можна сказати, що, мабуть, будь-який векторний формат – це метафайл. Наприклад, формат DXF містить команди(хоча команди не для пристрою, а для програми, що інтерпретує ці команди). Крім того, оскільки векторний формат містить опис зображення як список графічних примітивів, то важливою є послідовність накладання цих примітивів – знову можна сказати, що векторний формат — це не *список примітивів,* а послідовність *команд.* Будь-яка спроба догматично класифікувати реальні формати, які зазвичай містять багато різноманітних можливостей, вважається сумнівною. Для класифікації можна скористатися й таким принципом – якщо розробник формату називає це метафайлом, то, напевне, так воно й є.

Формат WMF

Формат *WMF (Windows Metafile)* був розроблений компанією Microsoft для 16-розрядних версій Windows. Метафайл являє собою послідовність команд GDI з усіх основних категорій графічних примітивів 16-розрядного інтерфейсу Windows GDI. У цей формат конвертуються векторні зображення при переносі з програми в програму через clipboard (буфер обміну).

Формат WMF відрізняється великою сумісністю для PC, його розуміють і деякі програми для Macintosh. Можливості метафайла Windows значною мірою обмежені через залежність від пристроїв і програм. Метафайл не має інформації про розмір зображення, початкову роздільність чи стан палітри пристрою, на якому записувалось це зображення. Існують й інші обмеження, що накладаються GDI [3, 4].

Формат EMF

У 32-розрядних версіях Windows, починаючи з Windows NT 3/5.1, компанія Microsoft використовує 32-розрядний формат метафайлів, що називається розширеним форматом метафайлів *(Enhanced Metafile, EMF)*. Порівняно з WMF, формат EMF підтримує 32-розрядну систему координат та нові 32-розрядні функції GDI, містить заголовок із геометричними даними та палітрою й навіть забезпечує деяку підтримку OpenGL. Формат EMF, що порівняно з форматом WMF менше залежить від пристрою й підтримує нові функції GDI, отримує все більшу популярність [3].

Формат PICT

На платформі Macintosh аналогічну формату WMF роль відіграє формат *PICT*. Формат PICT був розроблений компанією Apple Computer у 1984 спочатку для програми MacDraw. Багато програм для PC також розуміють цей формат. PICT-файли в об'єктно-орієнтованому форматі складаються з окремих графічних об'єктів (ліній, дуг, овалів чи прямокутників), кожний з яких можна незалежно редагувати й масштабувати. У PICT-файлах можуть також зберігатися растрові зображення. Файли PICT кодуються командами QuickDraw. Файли PICT в основному використовуються для обміну

Формат WMF

Формат *WMF (Windows Metafile)* був розроблений компанією Microsoft для 16-розрядних версій Windows. Метафайл являє собою послідовність команд GDI з усіх основних категорій графічних примітивів 16-розрядного інтерфейсу Windows GDI. У цей формат конвертуються векторні зображення при переносі з програми в програму через clipboard (буфер обміну).

Формат WMF відрізняється великою сумісністю для PC, його розуміють і деякі програми для Macintosh. Можливості метафайла Windows значною мірою обмежені через залежність від пристроїв і програм. Метафайл не має інформації про розмір зображення, початкову роздільність чи стан палітри пристрою, на якому записувалось це зображення. Існують й інші обмеження, що накладаються GDI [3, 4].

Формат EMF

У 32-розрядних версіях Windows, починаючи з Windows NT 3/5.1, компанія Microsoft використовує 32-розрядний формат метафайлів, що називається розширеним форматом метафайлів *(Enhanced Metafile, EMF)*. Порівняно з WMF, формат EMF підтримує 32-розрядну систему координат та нові 32-розрядні функції GDI, містить заголовок із геометричними даними та палітрою й навіть забезпечує деяку підтримку OpenGL. Формат EMF, що порівняно з форматом WMF менше залежить від пристрою й підтримує нові функції GDI, отримує все більшу популярність [3].

Формат PICT

На платформі Macintosh аналогічну формату WMF роль відіграє формат *PICT*. Формат PICT був розроблений компанією Apple Computer у 1984 спочатку для програми MacDraw. Багато програм для PC також розуміють цей формат. PICT-файли в об'єктно-орієнтованому форматі складаються з окремих графічних об'єктів (ліній, дуг, овалів чи прямокутників), кожний з яких можна незалежно редагувати й масштабувати. У PICT-файлах можуть також зберігатися растрові зображення. Файли PICT кодуються командами QuickDraw. Файли PICT в основному використовуються для обміну

графікою між програмами на Macintosh, також підтримуються на платформі Windows за допомогою QuickTime for Windows.

Формат PostScript

PostScript – мова опису сторінок (мова керування лазерними принтерами) фірми Adobe. Була створена у 80-х роках для реалізації принципу WYSIWYG (What You Seeis What You Get). Файли цього формату мають розширення .PS чи .PRN. Вони утворюються за допомогою функції PrinttoFile графічних програм при використанні драйвера PostScript-принтера, тобто є по суті програмою з командами на виконання для вивідного пристрою.

Файли формату PostScript містять у собі сам документ (те, що ви бачите на сторінках), усі зв'язані файли (растрові й векторні), використані шрифти, а також іншу інформацію для пристрою: плати кольороподілу, додаткові плати, лініатуру растра та форму растрової точки для кожної плати тощо. Дані в PostScript-файлі, як правило, записуються у двійковому кодуванні (Binery), яке приблизно вдвічі економніше за кодування ASCII. Але під час передачі файлів через мережі, під час крос-платформного обміну, під час друку через послідовні кабелі використовується й ASCII. Це викликано можливістю спотворення двійкового кодування у старих комп'ютерах.

Отже, якщо файл створений правильно, не має значення, на якій платформі він робився і які шрифти були використані. Найкоректніші PS-файли створюють програми Adobe. Сказане відноситься до форматів, заснованих мовою PostScript, а саме: EPS і PDF [22].

Формат EPS

Претендент на звання стандартного – формат *EPS (Encapsulated PostScript)* — призначений для передачі векторної й растрової графіки у видавничі системи. Спочатку EPS розроблявся як векторний формат, пізніше з'явився растровий варіант — Photoshop EPS. Формат EPS використовує спрощену версію мови PostScript: не може містити в одному файлі більше, ніж одну сторінку, не зберігає ряд установок для принтера тощо [3].

Основна перевага EPS – його надійність й універсальність. Майже всі програми, що працюють із графікою, можуть писати й

графікою між програмами на Macintosh, також підтримуються на платформі Windows за допомогою QuickTime for Windows.

Формат PostScript

PostScript – мова опису сторінок (мова керування лазерними принтерами) фірми Adobe. Була створена у 80-х роках для реалізації принципу WYSIWYG (What You Seeis What You Get). Файли цього формату мають розширення .PS чи .PRN. Вони утворюються за допомогою функції PrinttoFile графічних програм при використанні драйвера PostScript-принтера, тобто є по суті програмою з командами на виконання для вивідного пристрою.

Файли формату PostScript містять у собі сам документ (те, що ви бачите на сторінках), усі зв'язані файли (растрові й векторні), використані шрифти, а також іншу інформацію для пристрою: плати кольороподілу, додаткові плати, лініатуру растра та форму растрової точки для кожної плати тощо. Дані в PostScript-файлі, як правило, записуються у двійковому кодуванні (Binery), яке приблизно вдвічі економніше за кодування ASCII. Але під час передачі файлів через мережі, під час крос-платформного обміну, під час друку через послідовні кабелі використовується й ASCII. Це викликано можливістю спотворення двійкового кодування у старих комп'ютерах.

Отже, якщо файл створений правильно, не має значення, на якій платформі він робився і які шрифти були використані. Найкоректніші PS-файли створюють програми Adobe. Сказане відноситься до форматів, заснованих мовою PostScript, а саме: EPS і PDF [22].

Формат EPS

Претендент на звання стандартного – формат *EPS (Encapsulated PostScript)* — призначений для передачі векторної й растрової графіки у видавничі системи. Спочатку EPS розроблявся як векторний формат, пізніше з'явився растровий варіант — Photoshop EPS. Формат EPS використовує спрощену версію мови PostScript: не може містити в одному файлі більше, ніж одну сторінку, не зберігає ряд установок для принтера тощо [3].

Основна перевага EPS – його надійність й універсальність. Майже всі програми, що працюють із графікою, можуть писати й

читати EPS-файли. Формат EPS найбільш придатний для обміну зображеннями між різними програмами (наприклад, Flash і FreeHand). Цей формат забезпечує точне перетворення кривих, стилю ліній та заливок. У ньому, наприклад, можна записати векторний контур, що буде обмежувати растрове зображення, – це дозволяє одержати не тільки прямокутне зображення, а також колоподібне чи будь-якої іншої форми.

Слід зазначити, що EPS-файли, створені різними програмами, відкриваються по-різному, а часом і зовсім не відкриваються через декілька причин. По-перше, через існування щонайменше трьох версій мови PostScript, що використовується в цьому форматі. По-друге, через можливість зберігання зображення у файлі у двох копіях: основній і додатковій (imageheader, preview) – *ескізи* (із цієї причини растрове зображення, записане у форматі EPS буде мати більший розмір, ніж PCX або BMP).

"Рідна" програма для формату EPS — Adobe Illustrator, є версія 7.0 відразу для трьох платформ: PC, Macintosh і Silicon Graphics. Програма Adobe Illustrator має ще один формат — AI, однак він не отримав такої широкої підтримки, як EPS.

Формат PSD

Формат *PSD розроблений для програми* Adobe Photoshop. Він підтримує всі типи зображень від монохромної графіки до кольорового СМУК, дозволяє записувати растрове зображення з багатьма шарами, додатковими колірними каналами й іншою інформацією. При роботі з програмами фірми Adobe зображення можна зберегти у форматі PSD. Однак цей формат невідомий програмам верстки, для роботи з ними необхідно зробити спрощену копію файла в іншому форматі. Починаючи з версії 3.0, Photoshop записує такі файли з компресією, що не позначається на якості зображення при помітному зменшенні розміру. Під час роботи з версією Photoshop 4.0 файли стають ще меншими.

Формат PSD сприймається обмеженою кількістю графічних пакетів. Зазвичай формат PSD використовується, якщо необхідно зберегти створені користувачем альфа-канали чи завершувати шари, що можуть знадобитися пізніше під час редагування зображення.

читати EPS-файли. Формат EPS найбільш придатний для обміну зображеннями між різними програмами (наприклад, Flash і FreeHand). Цей формат забезпечує точне перетворення кривих, стилю ліній та заливок. У ньому, наприклад, можна записати векторний контур, що буде обмежувати растрове зображення, – це дозволяє одержати не тільки прямокутне зображення, а також колоподібне чи будь-якої іншої форми.

Слід зазначити, що EPS-файли, створені різними програмами, відкриваються по-різному, а часом і зовсім не відкриваються через декілька причин. По-перше, через існування щонайменше трьох версій мови PostScript, що використовується в цьому форматі. По-друге, через можливість зберігання зображення у файлі у двох копіях: основній і додатковій (imageheader, preview) – *ескізи* (із цієї причини растрове зображення, записане у форматі EPS буде мати більший розмір, ніж PCX або BMP).

"Рідна" програма для формату EPS — Adobe Illustrator, є версія 7.0 відразу для трьох платформ: PC, Macintosh і Silicon Graphics. Програма Adobe Illustrator має ще один формат — AI, однак він не отримав такої широкої підтримки, як EPS.

Формат PSD

Формат *PSD розроблений для програми* Adobe Photoshop. Він підтримує всі типи зображень від монохромної графіки до кольорового СМУК, дозволяє записувати растрове зображення з багатьма шарами, додатковими колірними каналами й іншою інформацією. При роботі з програмами фірми Adobe зображення можна зберегти у форматі PSD. Однак цей формат невідомий програмам верстки, для роботи з ними необхідно зробити спрощену копію файла в іншому форматі. Починаючи з версії 3.0, Photoshop записує такі файли з компресією, що не позначається на якості зображення при помітному зменшенні розміру. Під час роботи з версією Photoshop 4.0 файли стають ще меншими.

Формат PSD сприймається обмеженою кількістю графічних пакетів. Зазвичай формат PSD використовується, якщо необхідно зберегти створені користувачем альфа-канали чи завершувати шари, що можуть знадобитися пізніше під час редагування зображення.

Формат CDR

Формат ***CDR*** використовується програмою Corel Draw, яка на сьогоднішній день є однією з найпопулярніших серед програм, що дозволяють рисувати векторні рисунки. Формат CDR дозволяє записувати векторну й растрову графіку, текст. У процесі випуску нових версій формат зазнав серйозних змін. Через це виникло кілька прикрих незручностей, наприклад, погана сумісність файлів. Але, починаючи з восьмої версії, цей формат не викликає у користувачів будь-яких серйозних нарікань.

У формату CDR є незаперечна перевага – можливість читати ушкоджені файли. Програма, знайшовши ушкоджений об'єкт, просто пропускає його.

Однією з властивостей векторних форматів є відтворення масштабованих зображень об'єктів без погіршення роздільної здатності під час подальшого друку або виводу на інші пристрої. Файли Corel DRAW мають робоче поле до 45х45 метрів.

Деякі програми (FreeHand, Illustrator, PageMaker серед них) можуть імпортувати файли Corel DRAW, однак цей формат вважається закритим та не документованим.

Формати FH7, FH5

Програма FreeHand кілька разів переходила з рук у руки й на сьогоднішній день права на неї належать фірмі Macromedia. Як і Adobe Illustrator та Corel Draw, Free-Hand працює з векторними й растровими зображеннями. Вона має свій формат *–FH7* (останній символ у розширені файла вказує на номер версії програми). Щоб передати готове зображення іншій програмі, звичайно, доводиться записати його в більш сумісному форматі, наприклад, EPS. [3, 4, 20, 22].

3D - формати

VRML (мова моделювання віртуальної реальності – Virtual Reality Modeling Language) — графічний формат, що базується на підмножині OpenInventor фірми Silicon Graphics. Він призначений для опису тривимірних зображень й обміну ними в мережі World Wide Web.

Формат CDR

Формат ***CDR*** використовується програмою Corel Draw, яка на сьогоднішній день є однією з найпопулярніших серед програм, що дозволяють рисувати векторні рисунки. Формат CDR дозволяє записувати векторну й растрову графіку, текст. У процесі випуску нових версій формат зазнав серйозних змін. Через це виникло кілька прикрих незручностей, наприклад, погана сумісність файлів. Але, починаючи з восьмої версії, цей формат не викликає у користувачів будь-яких серйозних нарікань.

У формату CDR є незаперечна перевага – можливість читати ушкоджені файли. Програма, знайшовши ушкоджений об'єкт, просто пропускає його.

Однією з властивостей векторних форматів є відтворення масштабованих зображень об'єктів без погіршення роздільної здатності під час подальшого друку або виводу на інші пристрої. Файли Corel DRAW мають робоче поле до 45х45 метрів.

Деякі програми (FreeHand, Illustrator, PageMaker серед них) можуть імпортувати файли Corel DRAW, однак цей формат вважається закритим та не документованим.

Формати FH7, FH5

Програма FreeHand кілька разів переходила з рук у руки й на сьогоднішній день права на неї належать фірмі Macromedia. Як і Adobe Illustrator та Corel Draw, Free-Hand працює з векторними й растровими зображеннями. Вона має свій формат *–FH7* (останній символ у розширені файла вказує на номер версії програми). Щоб передати готове зображення іншій програмі, звичайно, доводиться записати його в більш сумісному форматі, наприклад, EPS. [3, 4, 20, 22].

3D - формати

VRML (мова моделювання віртуальної реальності – Virtual Reality Modeling Language) — графічний формат, що базується на підмножині OpenInventor фірми Silicon Graphics. Він призначений для опису тривимірних зображень й обміну ними в мережі World Wide Web.

166

Мова VRML, що була розроблена GavinBell, RickCarey, MarkPesce і TonyParisi, стала першою мовою тривимірного моделювання для Web. У 1995 році була створена група VAG (VRML Authoring Group) і з'явилася остаточна редакція специфікації VRML 1.0. У 1997 році технологію підтримали у своїх браузерах як Microsoft, так і Netscape. ISO схвалила другу версією специфікації як міжнародний стандарт VRML 97 (або ISO/IEC 14772-1).

VRML-файл має розширення WRL. Він використовує формат ASCII і являє собою звичайний текстовий файл із списком об'єктів, які названі вузлами (nodes). До вузлів VRML 2.0, зокрема, відносяться 3D-геометрія, властивості світла, що створюється за допомогою VRML, файли зображень формату JPEG, відеофайли формату MPEG, звукові файли формату MIDI, текстові документи формату HTML.

Основні ідеї мови VRML активно використовуються в сучасних засобах візуалізації тривимірних сцен.

Формат 3DS

Це один із найпоширеніших форматів для 3D-графіки. Файли формату *3DS* були стандартними файлами програми 3D Studio, ще коли вона працювала під DOS. У 3D Studio MAX з'явився інший формат збереження — MAX, але для розробки ігор цей новий формат виявився незручним. Натомість формат 3DS виявився придатним для цієї мети: крім самих тривимірних моделей (які являють собою каркасні сітки), він зберігає їхнє положення у світових координатах, координати текстур, кольори вершин, ключові кадри анімації, дані про властивості матеріалів і навіть атмосферні ефекти. Це практично готовий формат для збереження моделей і цілих карт (тільки скриптові команди доводиться зберігати окремо). Підкреслимо, що під час збереження 3DS-файлів можна вказати, щоб координати текстур зберігалися разом із моделлю. Після цієї операції накладення текстури відбувається якісніше. Формат 3DS є зручним і практичним для будь-яких видів ігрових моделей. Він широко використовується для обміну даними між системами тривимірного моделювань.

Мова VRML, що була розроблена GavinBell, RickCarey, MarkPesce і TonyParisi, стала першою мовою тривимірного моделювання для Web. У 1995 році була створена група VAG (VRML Authoring Group) і з'явилася остаточна редакція специфікації VRML 1.0. У 1997 році технологію підтримали у своїх браузерах як Microsoft, так і Netscape. ISO схвалила другу версією специфікації як міжнародний стандарт VRML 97 (або ISO/IEC 14772-1).

VRML-файл має розширення WRL. Він використовує формат ASCII і являє собою звичайний текстовий файл із списком об'єктів, які названі вузлами (nodes). До вузлів VRML 2.0, зокрема, відносяться 3D-геометрія, властивості світла, що створюється за допомогою VRML, файли зображень формату JPEG, відеофайли формату MPEG, звукові файли формату MIDI, текстові документи формату HTML.

Основні ідеї мови VRML активно використовуються в сучасних засобах візуалізації тривимірних сцен.

Формат 3DS

Це один із найпоширеніших форматів для 3D-графіки. Файли формату *3DS* були стандартними файлами програми 3D Studio, ще коли вона працювала під DOS. У 3D Studio MAX з'явився інший формат збереження — MAX, але для розробки ігор цей новий формат виявився незручним. Натомість формат 3DS виявився придатним для цієї мети: крім самих тривимірних моделей (які являють собою каркасні сітки), він зберігає їхнє положення у світових координатах, координати текстур, кольори вершин, ключові кадри анімації, дані про властивості матеріалів і навіть атмосферні ефекти. Це практично готовий формат для збереження моделей і цілих карт (тільки скриптові команди доводиться зберігати окремо). Підкреслимо, що під час збереження 3DS-файлів можна вказати, щоб координати текстур зберігалися разом із моделлю. Після цієї операції накладення текстури відбувається якісніше. Формат 3DS є зручним і практичним для будь-яких видів ігрових моделей. Він широко використовується для обміну даними між системами тривимірного моделювань.

Формати MDL, MD2 і MD3

Ці формати призначені для збереження анімаційних моделей, особливо анімації людей. Усі три формати відкриті, ними може скористатися будь-який бажаючий. У них зберігаються моделі й дані про немов би "кістякову" анімацію. Анімації моделей зберігаються в цих форматах у тривимірних кадрах, причому кожному кадру відповідає своя повноцінна модель. Розмір файлів при цьому зростає пропорційно кількості кадрів анімації й може виявитися завеликим для серйозних ігор, втім, формати MDL, MD2 і MD3 можна досить сильно ущільнити.

Формат SMD

У файлах формату SMD зберігається "чиста" кістякова анімація. Це приводить до зменшення розмірів файлів і поліпшення якості анімації. Система збереження файлів має деякі особливості. По-перше, необхідний ключовий SMD-файл. У ньому зберігається сама модель без анімації. Його необхідно завантажувати в першу чергу. По-друге, необхідні анімаційні дані про модель, які зберігаються в інших файлах (один файл на одну анімацію). Якщо необхідна висока якість анімації, використовують цей формат.

Формати мультимедіа

Через особливості відеоінформації у цифровому відеозаписі ущільнення без утрат саме по собі майже не застосовується. Використання цих методів ущільнення (подібних до методів. що використовуються в архіваторах типу WinZIP) дозволяє зменшити розмір файла не більше, ніж на 2/3, хоча, звісно, при цьому не відбувається погіршення якості зображення.

Ущільнення з утратою якості є основним методом зменшення розміру відеофайлів. Такі алгоритми дозволяють визначити ту частину інформації, що глядач, імовірніше всього, не помітить під час перегляду фільму, й видалити її з файла. Основними форматами цифрового відео, що використовують ущільнення з утратами, на сьогоднішній день є Apple QuickTime, AVI, IntelIndeo, MJPEG, MPEG-1, MPEG-2, і MPEG-4.

Формати MDL, MD2 і MD3

Ці формати призначені для збереження анімаційних моделей, особливо анімації людей. Усі три формати відкриті, ними може скористатися будь-який бажаючий. У них зберігаються моделі й дані про немов би "кістякову" анімацію. Анімації моделей зберігаються в цих форматах у тривимірних кадрах, причому кожному кадру відповідає своя повноцінна модель. Розмір файлів при цьому зростає пропорційно кількості кадрів анімації й може виявитися завеликим для серйозних ігор, втім, формати MDL, MD2 і MD3 можна досить сильно ущільнити.

Формат SMD

У файлах формату SMD зберігається "чиста" кістякова анімація. Це приводить до зменшення розмірів файлів і поліпшення якості анімації. Система збереження файлів має деякі особливості. По-перше, необхідний ключовий SMD-файл. У ньому зберігається сама модель без анімації. Його необхідно завантажувати в першу чергу. По-друге, необхідні анімаційні дані про модель, які зберігаються в інших файлах (один файл на одну анімацію). Якщо необхідна висока якість анімації, використовують цей формат.

Формати мультимедіа

Через особливості відеоінформації у цифровому відеозаписі ущільнення без утрат саме по собі майже не застосовується. Використання цих методів ущільнення (подібних до методів. що використовуються в архіваторах типу WinZIP) дозволяє зменшити розмір файла не більше, ніж на 2/3, хоча, звісно, при цьому не відбувається погіршення якості зображення.

Ущільнення з утратою якості є основним методом зменшення розміру відеофайлів. Такі алгоритми дозволяють визначити ту частину інформації, що глядач, імовірніше всього, не помітить під час перегляду фільму, й видалити її з файла. Основними форматами цифрового відео, що використовують ущільнення з утратами, на сьогоднішній день є Apple QuickTime, AVI, IntelIndeo, MJPEG, MPEG-1, MPEG-2, і MPEG-4.

Формат AVI

AVI (Audio Video Interleaved – чергування аудіо й відео) – формат файлів, уведений фірмою Microsoft для використання систем роботи з відеозображеннями в середовищі Windows. У цьому форматі сектори відеоданих чергуються із секторами звукових даних таким чином, щоб відеоплеєр міг підтримувати мінімальну буферизацію даних.

AVI є спеціальним випадком формату *RIFF (Resource Interchange File Format).* RTFF – універсальний формат для обміну даними мультимедіа, спільно розроблений Microsor IBM. Фактично, RIFF – це аналог формату *IFF,* створеного Electronic Arts у 1984 році.

За структурою файл AVI-формату являє собою варіант файла формату RIFF. Файл цього формату складається з блоків (chunks), що, у свою чергу, можуть містити інші вкладені блоки. Якщо найбільш "поверхневий" блок містить ідентифікатор формату "avi", це означає, що ми маємо справу з *.avi-файлом.

Запис у форматі AVI може здійснюватися без або з ущільненням. Зазвичай використовується Motion JPEG. Також підтримуються формати компресії: Microsoft RLE (тільки 8-бітний колір), Microsoft Video 1 (тільки 8- і 16-бітний колір), Indeo, Cinepak Editable MPEG (використовує тільки I-кадри)[3]. Останнім часом усе популярнішим стає формат компресії за алгоритмом Div.

Дані у форматі AVI можна експортувати в різні формати.

Незважаючи на значне поширення форматів MPEG, формат AVI залишається популярним для аудіо- відеоданих на PC. Значний відсоток існуючих систем захоплення кадрів і нелінійного монтажу мають справу з форматом AVI.

У зв'язку з великою кількістю обмежень базового стандарту AVI, консорціумом Open Digital Media було розроблене розширення формату AVI –Open DML AVI,з урахуванням особливостей, необхідних для професійного виробництва відео. Це розширення включає підтримку полів (не тільки кадрів), розміри файлів можуть бути більшими за 1 Гб, є часовий код і багато інших особливостей. Microsoft включила підтримку Open DML AVI у DirectShow 5.1. Це

Формат AVI

AVI (Audio Video Interleaved – чергування аудіо й відео) – формат файлів, уведений фірмою Microsoft для використання систем роботи з відеозображеннями в середовищі Windows. У цьому форматі сектори відеоданих чергуються із секторами звукових даних таким чином, щоб відеоплеєр міг підтримувати мінімальну буферизацію даних.

AVI є спеціальним випадком формату *RIFF (Resource Interchange File Format).* RTFF – універсальний формат для обміну даними мультимедіа, спільно розроблений Microsor IBM. Фактично, RIFF – це аналог формату *IFF,* створеного Electronic Arts у 1984 році.

За структурою файл AVI-формату являє собою варіант файла формату RIFF. Файл цього формату складається з блоків (chunks), що, у свою чергу, можуть містити інші вкладені блоки. Якщо найбільш "поверхневий" блок містить ідентифікатор формату "avi", це означає, що ми маємо справу з *.avi-файлом.

Запис у форматі AVI може здійснюватися без або з ущільненням. Зазвичай використовується Motion JPEG. Також підтримуються формати компресії: Microsoft RLE (тільки 8-бітний колір), Microsoft Video 1 (тільки 8- і 16-бітний колір), Indeo, Cinepak Editable MPEG (використовує тільки I-кадри)[3]. Останнім часом усе популярнішим стає формат компресії за алгоритмом Div.

Дані у форматі AVI можна експортувати в різні формати.

Незважаючи на значне поширення форматів MPEG, формат AVI залишається популярним для аудіо- відеоданих на PC. Значний відсоток існуючих систем захоплення кадрів і нелінійного монтажу мають справу з форматом AVI.

У зв'язку з великою кількістю обмежень базового стандарту AVI, консорціумом Open Digital Media було розроблене розширення формату AVI –Open DML AVI,з урахуванням особливостей, необхідних для професійного виробництва відео. Це розширення включає підтримку полів (не тільки кадрів), розміри файлів можуть бути більшими за 1 Гб, є часовий код і багато інших особливостей. Microsoft включила підтримку Open DML AVI у DirectShow 5.1. Це

розширення також використовується в різних професійних програмах для виробництва відео на PC, зокрема Digi Suite(Matrox) (за матеріалами [3,22]).

Формат QuickTime

Розроблена фірмою Apple система QuickTime складається з трьох частин:

- формат файлів QuickTime Moviefile,
- QuickTime Media Abstraction Layer,
- QuickTime mediaservices.

Media Abstraction Layer– це рівень, що визначає способи взаємодії між програмами та апаратурою комп'ютера й набором функцій QuickTime, тобто він дозволяє розробникам програмного та апаратного забезпечення отримати доступ до можливостей QuickTime, зокрема, визначає, як апаратура комп'ютера може прискорювати роботу якоїсь окремої частини системи QuickTime й т. ін.

QuickTime mediaservices – бібліотека вбудованих медіафункцій QuickTime, що забезпечує програмний інтерфейс.

Фундамент усієї системи – формат файлів QuickTime, що називається *QuickTime Movie.* Формат може містити практично будь-які медіадані – відео- та аудіодоріжки, послідовності MIDI-повідомлень (стандартні MIDI-файли SMF), текст для титрів, графіка, анімацію тощо. Структура *movie* визначає номер та тип кожної доріжки й надає інформацію про фільм у цілому. *Доріжки* задають тривалість, послідовність та походження даних. Нарешті, *носії* містять реальні дані.

Формат QuickTime Movie дозволяє зберігати всю можливу інформацію не тільки різного типу медіа доріжок (різного типу аудіо-*і відеоінформацію), але й інформацію про медіакомпозиції* (набір медіадоріжок різного типу) у цілому. Це можуть бути, наприклад, описи просторового розташування джерел звуку на кожній доріжці (панорама) і т. д.

Крім того, QuickTime Movie– це формат на базі об'єктів. Він дає користувачам можливість маніпулювати даними, наприклад, імпортувати з мультимедіафайла вміст доріжок різного типу (відео, звук і т.д.) для окремої обробки, а потім вставити її назад. Можна, напри-

розширення також використовується в різних професійних програмах для виробництва відео на PC, зокрема Digi Suite(Matrox) (за матеріалами [3,22]).

Формат QuickTime

Розроблена фірмою Apple система QuickTime складається з трьох частин:

- формат файлів QuickTime Moviefile,
- QuickTime Media Abstraction Layer,
- QuickTime mediaservices.

Media Abstraction Layer– це рівень, що визначає способи взаємодії між програмами та апаратурою комп'ютера й набором функцій QuickTime, тобто він дозволяє розробникам програмного та апаратного забезпечення отримати доступ до можливостей QuickTime, зокрема, визначає, як апаратура комп'ютера може прискорювати роботу якоїсь окремої частини системи QuickTime й т. ін.

QuickTime mediaservices – бібліотека вбудованих медіафункцій QuickTime, що забезпечує програмний інтерфейс.

Фундамент усієї системи – формат файлів QuickTime, що називається *QuickTime Movie.* Формат може містити практично будь-які медіадані – відео- та аудіодоріжки, послідовності MIDI-повідомлень (стандартні MIDI-файли SMF), текст для титрів, графіка, анімацію тощо. Структура *movie* визначає номер та тип кожної доріжки й надає інформацію про фільм у цілому. *Доріжки* задають тривалість, послідовність та походження даних. Нарешті, *носії* містять реальні дані.

Формат QuickTime Movie дозволяє зберігати всю можливу інформацію не тільки різного типу медіа доріжок (різного типу аудіо-*і відеоінформацію), але й інформацію про медіакомпозиції* (набір медіадоріжок різного типу) у цілому. Це можуть бути, наприклад, описи просторового розташування джерел звуку на кожній доріжці (панорама) і т. д.

Крім того, QuickTime Movie– це формат на базі об'єктів. Він дає користувачам можливість маніпулювати даними, наприклад, імпортувати з мультимедіафайла вміст доріжок різного типу (відео, звук і т.д.) для окремої обробки, а потім вставити її назад. Можна, напри-

клад, збільшити або стиснути відеовставку (videoinsert) чи скорегувати її положення на своєму екрані. Можна або замінити вміст доріжок різного типу на інший, або додати до існуючої доріжки нові.

QuickTime надає можливість працювати практично з будь-яким типом відеокомпресії за допомогою Image Compression Manager.

Формат QuickTime може використовуватися в різних середовищах передачі даних, від звичайних телефонних модемів зі швидкістю 14.4біт/с до супутникової трансляції й волоконно-оптичних мереж із високою (до 160 Мбіт/с) пропускною здатністю.

Формати MPEG

Стандарт ***MPEG*** визначає ряд різних форматів. Концепція ущільнення відеозображень у форматі MPEG така:

• визначити, яка саме інформація в потоці повторюється хоча б у плині якогось відрізка часу, й вжити заходів до запобігання дублювання цієї інформації;

• використати особливості людського зору при виборі відповідних методів ущільнення та їхньої точності.

Відомо, що пікова зорова роздільна здатність досягається в ділянці сітківки, так званій центральній ямці, що охоплює всього лише два градуси кута зору. В міру віддалення від центральної ямки щільність фоторецепторів знижується майже за експонентним законом. Це означає, що для глядача, який дивиться у центр кіно-екрана, роздільність у периферійних зонах зображення майже цілком утрачає значення (з другого боку, периферійний зір характеризується високою сприйнятливістю до коливань яскравості) [3]. Крім того, здатність людини розрізняти дрібні колірні елементи в зображенні, що рухається, значно програє його здатності розрізняти зміни яскравості – тому зміни кольорів можуть кодуватися значно грубіше, ніж зміни яскравості.

Пояснимо, як може реалізуватися концепція ущільнення відеозображень на прикладі наступної сцени: переміщення автомобіля на фоні лісу. Фон при цьому можна вважати незмінним, оскільки основна увага глядача зазвичай звернена на рухомий предмет на передньому плані, навіть якщо є якісь дрібні зміни — глядач навряд чи зверне увагу чи встигне їх помітити.

клад, збільшити або стиснути відеовставку (videoinsert) чи скорегувати її положення на своєму екрані. Можна або замінити вміст доріжок різного типу на інший, або додати до існуючої доріжки нові.

QuickTime надає можливість працювати практично з будь-яким типом відеокомпресії за допомогою Image Compression Manager.

Формат QuickTime може використовуватися в різних середовищах передачі даних, від звичайних телефонних модемів зі швидкістю 14.4біт/с до супутникової трансляції й волоконно-оптичних мереж із високою (до 160 Мбіт/с) пропускною здатністю.

Формати MPEG

Стандарт ***MPEG*** визначає ряд різних форматів. Концепція ущільнення відеозображень у форматі MPEG така:

• визначити, яка саме інформація в потоці повторюється хоча б у плині якогось відрізка часу, й вжити заходів до запобігання дублювання цієї інформації;

• використати особливості людського зору при виборі відповідних методів ущільнення та їхньої точності.

Відомо, що пікова зорова роздільна здатність досягається в ділянці сітківки, так званій центральній ямці, що охоплює всього лише два градуси кута зору. В міру віддалення від центральної ямки щільність фоторецепторів знижується майже за експонентним законом. Це означає, що для глядача, який дивиться у центр кіно-екрана, роздільність у периферійних зонах зображення майже цілком утрачає значення (з другого боку, периферійний зір характеризується високою сприйнятливістю до коливань яскравості) [3]. Крім того, здатність людини розрізняти дрібні колірні елементи в зображенні, що рухається, значно програє його здатності розрізняти зміни яскравості – тому зміни кольорів можуть кодуватися значно грубіше, ніж зміни яскравості.

Пояснимо, як може реалізуватися концепція ущільнення відеозображень на прикладі наступної сцени: переміщення автомобіля на фоні лісу. Фон при цьому можна вважати незмінним, оскільки основна увага глядача зазвичай звернена на рухомий предмет на передньому плані, навіть якщо є якісь дрібні зміни — глядач навряд чи зверне увагу чи встигне їх помітити.

Отже, можна розбити кадр на дві складові частини – фон, що зберігається один раз, а потім підставляється при відтворенні всіх кадрів, і область, де рухається автомобіль, – її записують окремо для кожного кадру. Переміщення автомобіля може задаватися двома параметрами — вектором руху й кутом повороту навколо своєї осі.

Завдяки реалізації цієї концепції MPEG-компресор ущільнює відеопотік у десятки й навіть сотні разів без значної утрати якості зображення.

Слід зазначити, що однозначно оцінити якість кодування якимись приладами неможливо. Головним критерієм є те, як людина сприйме ущільнену інформацію. Тому правила ущільнення відеоданих при MPEG-кодуванні здійснюється на основі *моделі сприйняття людиною відеозображень (HVS — Human Visual Sense)* [3].

Надмірність зображення згідно з HVS підрозділяється на три основні види:

• невидимі людським оком ділянки зображення. Це, наприклад, місця гасіння по вертикалі й горизонталі. Видалення цієї інформації ніяк не позначається на зображенні;

• статистично надмірні ділянки зображення. Розрізняють просторову (ділянки зображення, на яких суміжні піксели практично однакові) й часову (незмінювані в часі ділянки зображення) надмірність;

• надмірні за кольором і яскравістю ділянки зображення. Враховується обмежена чутливість людини до невеликих змін яскравості й особливо кольору фрагментів зображення.

Редагувати MPEG-відео дуже важко через неможливість точної до кадру нарізки фрагментів.

Формат MPEG-1

Незважаючи на всі помітні недоліки цього формату, MPEG-1 є одним із найбільш масових форматів відеоущільнення.

Треба сказати, що можливості MPEG-1 не обмежені тією низькою роздільністю, яку ви бачили при перегляді VIDEO-CD. Перші CD-ROM програвачі були одношвидкісними, тому максимальна *швидкість передачі потоку даних (bitstream)* у форматі MPEG-1 обмежена 150 Кб/с, що відповідає одній швидкості CD-ROM. У самому форматі була закладена можливість ущільнення й відтворення

Отже, можна розбити кадр на дві складові частини – фон, що зберігається один раз, а потім підставляється при відтворенні всіх кадрів, і область, де рухається автомобіль, – її записують окремо для кожного кадру. Переміщення автомобіля може задаватися двома параметрами — вектором руху й кутом повороту навколо своєї осі.

Завдяки реалізації цієї концепції MPEG-компресор ущільнює відеопотік у десятки й навіть сотні разів без значної утрати якості зображення.

Слід зазначити, що однозначно оцінити якість кодування якимись приладами неможливо. Головним критерієм є те, як людина сприйме ущільнену інформацію. Тому правила ущільнення відеоданих при MPEG-кодуванні здійснюється на основі *моделі сприйняття людиною відеозображень (HVS — Human Visual Sense)* [3].

Надмірність зображення згідно з HVS підрозділяється на три основні види:

• невидимі людським оком ділянки зображення. Це, наприклад, місця гасіння по вертикалі й горизонталі. Видалення цієї інформації ніяк не позначається на зображенні;

• статистично надмірні ділянки зображення. Розрізняють просторову (ділянки зображення, на яких суміжні піксели практично однакові) й часову (незмінювані в часі ділянки зображення) надмірність;

• надмірні за кольором і яскравістю ділянки зображення. Враховується обмежена чутливість людини до невеликих змін яскравості й особливо кольору фрагментів зображення.

Редагувати MPEG-відео дуже важко через неможливість точної до кадру нарізки фрагментів.

Формат MPEG-1

Незважаючи на всі помітні недоліки цього формату, MPEG-1 є одним із найбільш масових форматів відеоущільнення.

Треба сказати, що можливості MPEG-1 не обмежені тією низькою роздільністю, яку ви бачили при перегляді VIDEO-CD. Перші CD-ROM програвачі були одношвидкісними, тому максимальна *швидкість передачі потоку даних (bitstream)* у форматі MPEG-1 обмежена 150 Кб/с, що відповідає одній швидкості CD-ROM. У самому форматі була закладена можливість ущільнення й відтворення

відеоінформації з роздільністю до 4095x4095 і частотою зміни кадрів до 60 Гц. Але через те, що потік передачі даних був обмежений 150 Кб/с, так звана *Constrained Parameters Bitstream – **CPB** – зафіксована ширина потоку даних)*, розроблювачі формату були змушені використовувати роздільність кадру, оптимізовану під даний CPB [3].

Формат MPEG-2

Формат MPEG-2 виник у результаті переробки MPEG-1 під запити компаній, що займалися супутниковим телебаченням і нелінійним цифровим відеомонтажем.

Найцінніша перевага MPEG-кодування – особливо зручна для передачі різними мережами можливість гнучкого настроювання якості зображення залежно від пропускної здатності мережі. Це зробило MPEG-2 фактичним стандартом для прийому/передачі цифрового телебачення різними мережами.

Формат MPEG-4

Популярний у даний час формат для передачі всіх видів мультимедійних даних: графіки, відео, анімації, звуку і т. д. Основна особливість– гнучкість, що дозволяє працювати з даному форматі MPEG-4 навіть малопотужним клієнтам (таким, як кишенькові ПК).

Ще раз зазначимо, що MPEG-4 є стандартом, формат файлів MPEG-4 має назву MP4, його розробка ґрунтувалася на алгоритмі Apple QuickTime. Ряд структур цього алгоритму зустрічається й у MPEG-4. Це, наприклад, треки для завдання варіанту передачі потокових медіа даних мережею, причому в MPEG-4 для одного файла можуть бути задані декілька варіантів передачі в різні програмні засоби. На відміну від QuickTime, MPEG-4 забезпечує стійке відтворення всіх типів потокових медіаданих, тобто не тільки аудіо- й відеоданих, але й змішаного контента *(контент* — цифрове подання медіаданих).

Стандарт MPEG-4 задає принципи роботи з контентом для трьох областей: інтерактивного мультимедіа (зокрема, це оптичні диски й інформація, розповсюджувана через мережу), графічних додатків (синтетичного контента) і цифрового телебачення – DTV; фактично

відеоінформації з роздільністю до 4095x4095 і частотою зміни кадрів до 60 Гц. Але через те, що потік передачі даних був обмежений 150 Кб/с, так звана *Constrained Parameters Bitstream – **CPB** – зафіксована ширина потоку даних)*, розроблювачі формату були змушені використовувати роздільність кадру, оптимізовану під даний CPB [3].

Формат MPEG-2

Формат MPEG-2 виник у результаті переробки MPEG-1 під запити компаній, що займалися супутниковим телебаченням і нелінійним цифровим відеомонтажем.

Найцінніша перевага MPEG-кодування – особливо зручна для передачі різними мережами можливість гнучкого настроювання якості зображення залежно від пропускної здатності мережі. Це зробило MPEG-2 фактичним стандартом для прийому/передачі цифрового телебачення різними мережами.

Формат MPEG-4

Популярний у даний час формат для передачі всіх видів мультимедійних даних: графіки, відео, анімації, звуку і т. д. Основна особливість– гнучкість, що дозволяє працювати з даному форматі MPEG-4 навіть малопотужним клієнтам (таким, як кишенькові ПК).

Ще раз зазначимо, що MPEG-4 є стандартом, формат файлів MPEG-4 має назву MP4, його розробка ґрунтувалася на алгоритмі Apple QuickTime. Ряд структур цього алгоритму зустрічається й у MPEG-4. Це, наприклад, треки для завдання варіанту передачі потокових медіа даних мережею, причому в MPEG-4 для одного файла можуть бути задані декілька варіантів передачі в різні програмні засоби. На відміну від QuickTime, MPEG-4 забезпечує стійке відтворення всіх типів потокових медіаданих, тобто не тільки аудіо- й відеоданих, але й змішаного контента *(контент* — цифрове подання медіаданих).

Стандарт MPEG-4 задає принципи роботи з контентом для трьох областей: інтерактивного мультимедіа (зокрема, це оптичні диски й інформація, розповсюджувана через мережу), графічних додатків (синтетичного контента) і цифрового телебачення – DTV; фактично

даний формат задає правила організації середовища, причому середовища об'єктно-орієнтованого. Він має справу не просто з потоками й масивами медіа даних, а з медіаоб'єктами (ключове поняття стандарту).

Як медіаоб'єкти можуть виступати тексти, графіка, синтезовані мовні фрагменти, тривимірні моделі (у тому числі анімація обличчя), потокові аудіо та відео тощо. Ці медіаоб'єкти допускають відображення на екрані зв'язаної з ними в сценарії медіаінформації будь-якого виду. MPEG-4 представляє користувачам гнучкі засоби роботи з мультимедійним контентом. Формат дозволяє робити прив'язку взаємного розташування аудіо- і відеоданих, природних і синтезованих комп'ютером 2D- і 3D-об'єктів, їх взаємну синхронізацію, а також вказувати їх інтерактивну взаємодію з користувачем.

Наявність у форматі MPEG-4 протоколу для підтримки фронтальної анімації у медіафайлах дозволяє анімувати в реальному часі тривимірні моделі обличчя, їх сполучення з аудіошаблонами чи мовним відтворенням тексту, отриманим за допомогою синтезатора робить можливим абсолютно синхронне озвучування. Але слід зазначити, що в MPEG-4 пропонується тільки протокол для керування тривимірними моделями.

Значне нововведення при компресії відео в стандарті MPEG-4 полягає в наступному. Якщо попередні формати поділяли зображення на прямокутники, кодер MPEG-4 при обробці зображень оперує об'єктами з довільною формою. Наприклад, людина, що рухається кімнатою, буде сприйнята як окремий об'єкт, що перемішується щодо нерухомого об’єкта – тильного плану.

Завдяки підтримці альфа-каналів, закладеної у кодер MPEG-4, можна в реальному часі накладати відео на тильний план. Такий прийом є корисним, наприклад, при сегментації (виділенні на базовому зображенні елементів переднього й тильного планів). Сегментація значно підвищує якість зображення при заданій швидкості передачі. Наприклад, у випадку сюжету, в якому людина періодично з'являється на якомусь фоні, звичайний кодер зображення тильного плану повинен передавати декілька разів. А якщо створити кодер відповідно до MPEG-4, то можна здійснити сегментацію і зберегти в пам'яті кодера інформацію про зображення тильного плану, тоді повторне пересилання даних не потрібне.

даний формат задає правила організації середовища, причому середовища об'єктно-орієнтованого. Він має справу не просто з потоками й масивами медіа даних, а з медіаоб'єктами (ключове поняття стандарту).

Як медіаоб'єкти можуть виступати тексти, графіка, синтезовані мовні фрагменти, тривимірні моделі (у тому числі анімація обличчя), потокові аудіо та відео тощо. Ці медіаоб'єкти допускають відображення на екрані зв'язаної з ними в сценарії медіаінформації будь-якого виду. MPEG-4 представляє користувачам гнучкі засоби роботи з мультимедійним контентом. Формат дозволяє робити прив'язку взаємного розташування аудіо- і відеоданих, природних і синтезованих комп'ютером 2D- і 3D-об'єктів, їх взаємну синхронізацію, а також вказувати їх інтерактивну взаємодію з користувачем.

Наявність у форматі MPEG-4 протоколу для підтримки фронтальної анімації у медіафайлах дозволяє анімувати в реальному часі тривимірні моделі обличчя, їх сполучення з аудіошаблонами чи мовним відтворенням тексту, отриманим за допомогою синтезатора робить можливим абсолютно синхронне озвучування. Але слід зазначити, що в MPEG-4 пропонується тільки протокол для керування тривимірними моделями.

Значне нововведення при компресії відео в стандарті MPEG-4 полягає в наступному. Якщо попередні формати поділяли зображення на прямокутники, кодер MPEG-4 при обробці зображень оперує об'єктами з довільною формою. Наприклад, людина, що рухається кімнатою, буде сприйнята як окремий об'єкт, що перемішується щодо нерухомого об’єкта – тильного плану.

Завдяки підтримці альфа-каналів, закладеної у кодер MPEG-4, можна в реальному часі накладати відео на тильний план. Такий прийом є корисним, наприклад, при сегментації (виділенні на базовому зображенні елементів переднього й тильного планів). Сегментація значно підвищує якість зображення при заданій швидкості передачі. Наприклад, у випадку сюжету, в якому людина періодично з'являється на якомусь фоні, звичайний кодер зображення тильного плану повинен передавати декілька разів. А якщо створити кодер відповідно до MPEG-4, то можна здійснити сегментацію і зберегти в пам'яті кодера інформацію про зображення тильного плану, тоді повторне пересилання даних не потрібне.

Що стосується використання MPEG-4 в Інтернеті, то в порівнянні з MPEG-1 поліпшена якість передачі відеоданих при швидкостях обміну від 20 Кб/с до 1000 Кб/с. MPEG-4 забезпечує повну підтримку контента з черезрядковим розгорненням і роздільністю до 4096 x 4096, а також швидкість передачі даних у діапазоні від 5 Кб/с до 10 Мб/с (версія 1).

На відміну від інших форматів, для яких необхідна повна підтримка всіх умов, MPEG-4 доступний у різних варіантах. Більшість інструментів, створених з урахуванням MPEG-4, підтримують тільки функції версії 1.

Підсистеми стандарту розбиті на кілька профілів. Кожен профіль підтримує ряд функціональних пакетів, орієнтованих тільки на визначену область застосування. Для забезпечення сумісності в другій версії подані не тільки додаткові профілі, але й збережені всі старі.

У першу версію MPEG-4 включені дев'ять відео- і чотири аудіопрофілі, а в другу додані ще сім відео- і чотири аудіопрофілі. Пропоновані профілі охоплюють весь спектр функцій відтворення: від простого відео з єдиним аудіопотоком (Simple Visual Profile) до повного розгортання медіафайлів, які містять різноманітні елементи, що забезпечується розширеним кодеком другої версії (Advanced Coding Profile). Стандартним для настільних комп'ютерів є визнаний MainVisual Profile [3, 20].

Формат MPEG-7

Офіційна назва стандарту MPEG-7 — "Інтерфейс для опису мультимедійного контента" Multimedia Content Description Interface). MPEG-7 не є новим технічним рішенням, основна увага приділяється метаданим, індексації, організації роботи програм. Зокрема, MPEG-7 є універсальним способом для ведення каталогу й пошуку відео.

Формат MPEG-21

Розробка MPEG-21 — це довгостроковий проект, що називається "Система мультимедійних засобів" (Multimedia Framework). На перших етапах планувалося провести розширення, уніфікацію й об'єднання MPEG-4 і MPEG-7 у єдину узагальнюючу структуру. Малось

Що стосується використання MPEG-4 в Інтернеті, то в порівнянні з MPEG-1 поліпшена якість передачі відеоданих при швидкостях обміну від 20 Кб/с до 1000 Кб/с. MPEG-4 забезпечує повну підтримку контента з черезрядковим розгорненням і роздільністю до 4096 x 4096, а також швидкість передачі даних у діапазоні від 5 Кб/с до 10 Мб/с (версія 1).

На відміну від інших форматів, для яких необхідна повна підтримка всіх умов, MPEG-4 доступний у різних варіантах. Більшість інструментів, створених з урахуванням MPEG-4, підтримують тільки функції версії 1.

Підсистеми стандарту розбиті на кілька профілів. Кожен профіль підтримує ряд функціональних пакетів, орієнтованих тільки на визначену область застосування. Для забезпечення сумісності в другій версії подані не тільки додаткові профілі, але й збережені всі старі.

У першу версію MPEG-4 включені дев'ять відео- і чотири аудіопрофілі, а в другу додані ще сім відео- і чотири аудіопрофілі. Пропоновані профілі охоплюють весь спектр функцій відтворення: від простого відео з єдиним аудіопотоком (Simple Visual Profile) до повного розгортання медіафайлів, які містять різноманітні елементи, що забезпечується розширеним кодеком другої версії (Advanced Coding Profile). Стандартним для настільних комп'ютерів є визнаний MainVisual Profile [3, 20].

Формат MPEG-7

Офіційна назва стандарту MPEG-7 — "Інтерфейс для опису мультимедійного контента" Multimedia Content Description Interface). MPEG-7 не є новим технічним рішенням, основна увага приділяється метаданим, індексації, організації роботи програм. Зокрема, MPEG-7 є універсальним способом для ведення каталогу й пошуку відео.

Формат MPEG-21

Розробка MPEG-21 — це довгостроковий проект, що називається "Система мультимедійних засобів" (Multimedia Framework). На перших етапах планувалося провести розширення, уніфікацію й об'єднання MPEG-4 і MPEG-7 у єдину узагальнюючу структуру. Малось

на увазі, що вона буде забезпечувати глибоку підтримку керування правами й платіжними системами, а також якістю наданих послуг.

Подальший розвиток форматів подання відеоінформації може бути пов'язаний з використанням психофізіологічних моделей візуального сприйняття, які досліджуються в даний час не тільки в наукових установах, але й у компаніях-розробниках відеоустаткування. Ці дослідження основані на ідеї, що ущільнене відеозображення повинно не стільки апроксимувати первинний матеріал, скільки найбільш адекватним способом збуджувати специфічні нервові вузли зорового тракту.

Ефективність алгоритмів відеокомпресії може бути підвищена також за рахунок подальшого дослідження й урахування особливостей людського зору, зокрема, нерівномірної щільності розподілу фоторецепторів тощо.

4.6. Алгоритми стиснення даних

Графічна інформація, як будь-які дані, що зберігаються або передається по каналах зв’язку, має значну надмірність. Використання алгоритмів стиснення інформації дозволяє підвищити в кілька разів швидкість обміну по каналах зв’язку та значно економити обсяги пам’яті.

На сьогоднішній день існує багато підходів до стиснення даних і алгоритмів, які їх реалізують [17, 18, 22].

Стиснення – це процес зменшення фізичного розміру блока даних. Існує декілька способів стиснення. Ми будемо розрізняти фізичне і логічне стиснення, симетричне й асиметричне, стиснення з утратами і без утрат. Найбільш поширені методи (або алгоритми) стиснення:

• Упаковка пікселів фактично не є методом стиснення даних, але дозволяє ефективно записувати їх у послідовно розташовані байти пам'яті. Цей метод застосовується у форматі Macintosh PICT та інших форматах, що дають можливість записувати декілька одного, двох або чотирьох бітових пікселів в один байт пам'яті або дискового простору.

• Групове кодування (RLE) є алгоритмом стиснення, що застосовуються у таких растрових форматах, як BMP, TIFF і PCX для зменшення обсягу надлишкових графічних даних.

на увазі, що вона буде забезпечувати глибоку підтримку керування правами й платіжними системами, а також якістю наданих послуг.

Подальший розвиток форматів подання відеоінформації може бути пов'язаний з використанням психофізіологічних моделей візуального сприйняття, які досліджуються в даний час не тільки в наукових установах, але й у компаніях-розробниках відеоустаткування. Ці дослідження основані на ідеї, що ущільнене відеозображення повинно не стільки апроксимувати первинний матеріал, скільки найбільш адекватним способом збуджувати специфічні нервові вузли зорового тракту.

Ефективність алгоритмів відеокомпресії може бути підвищена також за рахунок подальшого дослідження й урахування особливостей людського зору, зокрема, нерівномірної щільності розподілу фоторецепторів тощо.

4.6. Алгоритми стиснення даних

Графічна інформація, як будь-які дані, що зберігаються або передається по каналах зв’язку, має значну надмірність. Використання алгоритмів стиснення інформації дозволяє підвищити в кілька разів швидкість обміну по каналах зв’язку та значно економити обсяги пам’яті.

На сьогоднішній день існує багато підходів до стиснення даних і алгоритмів, які їх реалізують [17, 18, 22].

Стиснення – це процес зменшення фізичного розміру блока даних. Існує декілька способів стиснення. Ми будемо розрізняти фізичне і логічне стиснення, симетричне й асиметричне, стиснення з утратами і без утрат. Найбільш поширені методи (або алгоритми) стиснення:

• Упаковка пікселів фактично не є методом стиснення даних, але дозволяє ефективно записувати їх у послідовно розташовані байти пам'яті. Цей метод застосовується у форматі Macintosh PICT та інших форматах, що дають можливість записувати декілька одного, двох або чотирьох бітових пікселів в один байт пам'яті або дискового простору.

• Групове кодування (RLE) є алгоритмом стиснення, що застосовуються у таких растрових форматах, як BMP, TIFF і PCX для зменшення обсягу надлишкових графічних даних.

• Алгоритм Lempel-Ziv-Welch (LZW) застосовується у форматах GIF і TIFF, а також включений на стандарт стиснення даних для деяких модемів і є частиною PostScript Level 2.

• Кодування CCITT - форма стиснення даних, застосовувана для факсимільної передачі й стандартизована Міжнародним консультативним комітетом по телеграфії і телефонії (CCITT). Стандарт базується на схемі ключового стиснення, запропонованої Девідом Хаффманом, і широко відомий як кодування за алгоритм Хаффмана.

• Алгоритм, розроблений об'єднаною експертною групою по фотографії (JPEG), - набір методів стиснення, що використовуються в основному для обробки зображень з плавним переходом тону і для мультимедіа. Базова реалізація JPEG застосовує схему кодування за алгоритмом дискретних косинусних перетворень (DCT).

• Алгоритм, розроблений об'єднаною експертною групою з дворівневим зображенням (JBIG), - метод (стиснення даних дворівневих (двоколірних) зображень, який покликаний замінити алгоритми стиснення MR (Modified READ) і MMR (Modified Modified READ), використовувані CCITT Group 3 і Group 4 .

• ART - патентований алгоритм стиснення, розроблений фірмою Johnson-Grace, який у майбутньому можна буде адаптувати для підтримки аудіо, анімації і повномасштабного відео.

• Фрактальне стиснення - математичний процес, використовуваний для кодування растрів, що містять реальне зображення, в сукупність математичних даних, які описують фрактальні (тобто схожі, повторювані) властивості зображення. Надалі під час опису кожного формату графічного файла вказуються вживані в ньому алгоритми стиснення.

Перш ніж безпосередньо почати розглядати сутність алгоритмів, хотілося б зробити застереження. Один і той же алгоритм часто можна реалізувати різними способами. Багато відомих алгоритмів, таких як RLE, LZW або JPEG, мають десятки різних реалізацій.

Крім того, у алгоритмів буває кілька явних параметрів, варіюючи які, можна змінювати характеристики процесів архівації та розархівації. У разі конкретної реалізації ці параметри фіксуються, виходячи з найбільш імовірних характеристик вхідних зображень, вимог на економію пам'яті, вимог на час архівації і т.д.

• Алгоритм Lempel-Ziv-Welch (LZW) застосовується у форматах GIF і TIFF, а також включений на стандарт стиснення даних для деяких модемів і є частиною PostScript Level 2.

• Кодування CCITT - форма стиснення даних, застосовувана для факсимільної передачі й стандартизована Міжнародним консультативним комітетом по телеграфії і телефонії (CCITT). Стандарт базується на схемі ключового стиснення, запропонованої Девідом Хаффманом, і широко відомий як кодування за алгоритм Хаффмана.

• Алгоритм, розроблений об'єднаною експертною групою по фотографії (JPEG), - набір методів стиснення, що використовуються в основному для обробки зображень з плавним переходом тону і для мультимедіа. Базова реалізація JPEG застосовує схему кодування за алгоритмом дискретних косинусних перетворень (DCT).

• Алгоритм, розроблений об'єднаною експертною групою з дворівневим зображенням (JBIG), - метод (стиснення даних дворівневих (двоколірних) зображень, який покликаний замінити алгоритми стиснення MR (Modified READ) і MMR (Modified Modified READ), використовувані CCITT Group 3 і Group 4 .

• ART - патентований алгоритм стиснення, розроблений фірмою Johnson-Grace, який у майбутньому можна буде адаптувати для підтримки аудіо, анімації і повномасштабного відео.

• Фрактальне стиснення - математичний процес, використовуваний для кодування растрів, що містять реальне зображення, в сукупність математичних даних, які описують фрактальні (тобто схожі, повторювані) властивості зображення. Надалі під час опису кожного формату графічного файла вказуються вживані в ньому алгоритми стиснення.

Перш ніж безпосередньо почати розглядати сутність алгоритмів, хотілося б зробити застереження. Один і той же алгоритм часто можна реалізувати різними способами. Багато відомих алгоритмів, таких як RLE, LZW або JPEG, мають десятки різних реалізацій.

Крім того, у алгоритмів буває кілька явних параметрів, варіюючи які, можна змінювати характеристики процесів архівації та розархівації. У разі конкретної реалізації ці параметри фіксуються, виходячи з найбільш імовірних характеристик вхідних зображень, вимог на економію пам'яті, вимог на час архівації і т.д.

Алгоритм RLE

Даний алгоритм незвичайно простий в реалізації. Групове кодування від англійського Run Length Encoding (RLE) – один із найстаріших і найпростіших алгоритмів архівації графіки. Зображення в ньому (як і в декількох алгоритмах, описаних нижче) витягується в ланцюжок байт по рядках растру. Само стискування в RLE відбувається за рахунок того, що в початковому зображенні зустрічаються ланцюжки однакових байтів. Заміна їх на пари <лічильник повторень, значення>зменшує надмірність даних.

Алгоритм декомпресії при цьому виглядає так:

```
    Initialization(...);
do {
   byte = ImageFile.ReadNextByte();
   if(є лічильником(byte)) {
      counter = Low6bits(byte)+1;
      value = ImageFile.ReadNextByte();
      for(i=1 tocounter)
         DecompressedFile.WriteByte(value)
      }
   else {
      DecompressedFile.WriteByte(byte)
} while(ImageFile.EOF());
```

У даному алгоритмі ознакою лічильника (counter) служать одиниці в двох верхніх бітах прочитаного файла:

Відповідно 6 бітів, що залишилися, витрачаються на лічильник, який може набувати значень від 1 до 64. Рядок з 64 байтів, що повторюються, ми перетворюємо на два байти, тобто стискуватимемо в 32 рази.

Рис. 4.1. Пояснення до алгоритму RLE

Алгоритм RLE

Даний алгоритм незвичайно простий в реалізації. Групове кодування від англійського Run Length Encoding (RLE) – один із найстаріших і найпростіших алгоритмів архівації графіки. Зображення в ньому (як і в декількох алгоритмах, описаних нижче) витягується в ланцюжок байт по рядках растру. Само стискування в RLE відбувається за рахунок того, що в початковому зображенні зустрічаються ланцюжки однакових байтів. Заміна їх на пари <лічильник повторень, значення>зменшує надмірність даних.

Алгоритм декомпресії при цьому виглядає так:

```
    Initialization(...);
do {
   byte = ImageFile.ReadNextByte();
   if(є лічильником(byte)) {
      counter = Low6bits(byte)+1;
      value = ImageFile.ReadNextByte();
      for(i=1 tocounter)
         DecompressedFile.WriteByte(value)
      }
   else {
      DecompressedFile.WriteByte(byte)
} while(ImageFile.EOF());
```

У даному алгоритмі ознакою лічильника (counter) служать одиниці в двох верхніх бітах прочитаного файла:

Відповідно 6 бітів, що залишилися, витрачаються на лічильник, який може набувати значень від 1 до 64. Рядок з 64 байтів, що повторюються, ми перетворюємо на два байти, тобто стискуватимемо в 32 рази.

Рис. 4.1. Пояснення до алгоритму RLE

Алгоритм розрахований на ділову графіку - зображення з великими областями кольору, що повторюється. Ситуація, коли файл збільшується, для цього простого алгоритму не така вже рідка. Її можна легко отримати, застосовуючи групове кодування до оброблених кольорових фотографій.

Для збільшення зображення в два рази його треба застосувати до зображення, в якому значення всіх пікселів більше двійкового 11000000 і підряд не повторюються.

Алгоритм LZW

Назву алгоритм отримав по перших буквах прізвищ його розробників - Lempel, Ziv і Welch. Стиснення в ньому, на відміну від RLE, здійснюється вже за рахунок однакових ланцюжків байтів.

Існує досить велике сімейство LZ-подібних алгоритмів, що розрізняються, наприклад, методом пошуку ланцюжків, що повторюються. Один з достатньо простих варіантів цього алгоритму, наприклад, передбачає, що у вхідному потоці йде або пара <лічильник, зсув щодо поточної позиції>, або просто <лічильник>байтів, що «пропускаються», і самі значення байтів (як у другому варіанті алгоритму RLE). При розархівуванні для пари <лічильник, зсув> копіюються <лічильник> байт з вихідного масиву, отриманого в результаті розархівування, на <зсув> байт, отриманий раніше, а <лічильник> (тобто число рівне лічильнику) значень байт, що «пропускаються», просто копіюються у вихідний масив з вхідного потоку. Даний алгоритм є несиметричним за часом, оскільки вимагає повного перебору буфера при пошуку однакових підрядків. У результаті нам складно задати великий буфер через різке зростання часу компресії. Проте потенційна побудова алгоритму, в якому на <лічильник>і на <зсув> буде виділено по 2 байти (старший біт старшого байта лічильника - ознака повтору рядка / копіювання потоку), дасть нам можливість стискувати підрядки, що все повторюються, розміром до 32Кб у буфері розміром 64Кб.

Алгоритм розрахований на ділову графіку - зображення з великими областями кольору, що повторюється. Ситуація, коли файл збільшується, для цього простого алгоритму не така вже рідка. Її можна легко отримати, застосовуючи групове кодування до оброблених кольорових фотографій.

Для збільшення зображення в два рази його треба застосувати до зображення, в якому значення всіх пікселів більше двійкового 11000000 і підряд не повторюються.

Алгоритм LZW

Назву алгоритм отримав по перших буквах прізвищ його розробників - Lempel, Ziv і Welch. Стиснення в ньому, на відміну від RLE, здійснюється вже за рахунок однакових ланцюжків байтів.

Існує досить велике сімейство LZ-подібних алгоритмів, що розрізняються, наприклад, методом пошуку ланцюжків, що повторюються. Один з достатньо простих варіантів цього алгоритму, наприклад, передбачає, що у вхідному потоці йде або пара <лічильник, зсув щодо поточної позиції>, або просто <лічильник>байтів, що «пропускаються», і самі значення байтів (як у другому варіанті алгоритму RLE). При розархівуванні для пари <лічильник, зсув> копіюються <лічильник> байт з вихідного масиву, отриманого в результаті розархівування, на <зсув> байт, отриманий раніше, а <лічильник> (тобто число рівне лічильнику) значень байт, що «пропускаються», просто копіюються у вихідний масив з вхідного потоку. Даний алгоритм є несиметричним за часом, оскільки вимагає повного перебору буфера при пошуку однакових підрядків. У результаті нам складно задати великий буфер через різке зростання часу компресії. Проте потенційна побудова алгоритму, в якому на <лічильник>і на <зсув> буде виділено по 2 байти (старший біт старшого байта лічильника - ознака повтору рядка / копіювання потоку), дасть нам можливість стискувати підрядки, що все повторюються, розміром до 32Кб у буфері розміром 64Кб.

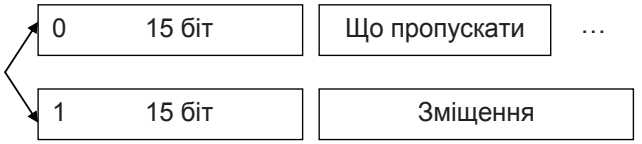


Рис. 4.2. Пояснення до алгоритму LZ

При цьому ми отримаємо збільшення розміру файла у гіршому разі на 32770/32768 (у двох байтах записано, що потрібно переписати у вихідний потік наступні 215 байтів), що зовсім непогано. До переваг LZ можна віднести надзвичайну простоту алгоритму декомпресії.

Варіант алгоритму, що розглядається нами нижче, використовуватиме дерево для уявлення і зберігання ланцюжків.

Вочевидь, що це достатньо сильне обмеження на вигляд ланцюжків, і далеко не всі однакові підланцюжки в нашому зображенні будуть використані при стискуванні. Проте в запропонованому алгоритмі вигідно стискувати навіть ланцюжки, що складаються з 2 байтів.

Процес стискування виглядає досить просто. Ми прочитуємо послідовно символи вхідного потоку і перевіряємо, чи є в створеній нами таблиці рядків такий рядок.

```
InitTable();
CompressedFile.WriteCode(Clearcode);
CurStr=пустий рядок;
while(не ImageFile.EOF()){ //Поки не кінець файла
   C=ImageFile.ReadNextByte();
   if(CurStr+C є в таблиці)
      CurStr=CurStr+C;// Приклеїти символ до рядка
   else {
      code=CodeForString(CurStr);//code-не байт!
      CompressedFile.WriteCode(code);
      AddStringToTable (CurStr+C);
      CurStr=C; // Рядок з одного символу
   }
}
```

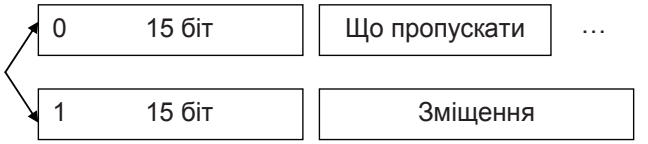



Рис. 4.2. Пояснення до алгоритму LZ

При цьому ми отримаємо збільшення розміру файла у гіршому разі на 32770/32768 (у двох байтах записано, що потрібно переписати у вихідний потік наступні 215 байтів), що зовсім непогано. До переваг LZ можна віднести надзвичайну простоту алгоритму декомпресії.

Варіант алгоритму, що розглядається нами нижче, використовуватиме дерево для уявлення і зберігання ланцюжків.

Вочевидь, що це достатньо сильне обмеження на вигляд ланцюжків, і далеко не всі однакові підланцюжки в нашому зображенні будуть використані при стискуванні. Проте в запропонованому алгоритмі вигідно стискувати навіть ланцюжки, що складаються з 2 байтів.

Процес стискування виглядає досить просто. Ми прочитуємо послідовно символи вхідного потоку і перевіряємо, чи є в створеній нами таблиці рядків такий рядок.

```
InitTable();
CompressedFile.WriteCode(Clearcode);
CurStr=пустий рядок;
while(не ImageFile.EOF()){ //Поки не кінець файла
   C=ImageFile.ReadNextByte();
   if(CurStr+C є в таблиці)
      CurStr=CurStr+C;// Приклеїти символ до рядка
   else {
      code=CodeForString(CurStr);//code-не байт!
      CompressedFile.WriteCode(code);
      AddStringToTable (CurStr+C);
      CurStr=C; // Рядок з одного символу
   }
}
```

```
code=CodeForString(CurStr);
CompressedFile.WriteCode(code);
CompressedFile.WriteCode(CodeEndOfInformation);
```

Якщо рядок є, то ми прочитуємо наступний символ, а якщо рядка немає, то ми заносимо в потік код для попереднього знайденого рядка, заносимо рядок в таблицю і починаємо пошук знову.

Функція InitTable() очищає таблицю і поміщає в неї всі рядки одиничної довжини.

Як мовилося вище, функція InitTable() ініціалізувала таблицю рядків так, щоб вона містила всі можливі рядки, що складаються з одного символу. Наприклад, якщо ми стискуємо байтові дані, то таких рядків у таблиці буде 256 ('0', '1' ..., '255'). Для коду очищення (ClearCode) і коду кінця інформації (Code EndOf Information) зарезервовані значення 256 і 257. У даному варіанті алгоритму використовується 12-бітовий код, і, відповідно, під коди для рядків нам залишаються значення від 258 до 4095. Рядки, що додаються, записуються в таблицю послідовно, при цьому індекс рядка в таблиці стає її кодом.

Функція ReadNextByte() читає символ з файла. Функція WriteCode() записує код (не рівний за розміром байту) у вихідний файл. Функція AddStringToTable() додає новий рядок у таблицю, приписуючи їй код. Крім того, в даній функції відбувається обробка ситуації переповнювання таблиці.

В цьому випадку в потік записується код попереднього знайденого рядка і код очищення, після чого таблиця очищається функцією InitTable(). Функція CodeForString() знаходить рядок у таблиці і видає код цього рядка.

Приклад 4.2.

Хай ми стискуємо послідовність 45, 55, 55, 151, 55, 55, 55. Тоді, згідно з викладеним вище алгоритмом, ми помістимо у вихідний потік спочатку код очищення <256>, потім додамо до початку порожнього рядка '45' і перевіримо, чи є рядок '45' у таблиці. Оскільки ми під час ініціалізації занесли в таблицю всі рядки з одного символу, то рядок '45' є в таблиці.

Далі ми читаємо наступний символ 55 з вхідного потоку і перевіряємо, чи є рядок '45, 55' в таблиці. Такого рядка в таблиці поки немає. Ми заносимо в таблицю рядок '45, 55' (з першим вільним кодом 258) і записуємо в потік код <45>. Можна коротко подати архівацію так:


```
code=CodeForString(CurStr);
CompressedFile.WriteCode(code);
CompressedFile.WriteCode(CodeEndOfInformation);
```

Якщо рядок є, то ми прочитуємо наступний символ, а якщо рядка немає, то ми заносимо в потік код для попереднього знайденого рядка, заносимо рядок в таблицю і починаємо пошук знову.

Функція InitTable() очищає таблицю і поміщає в неї всі рядки одиничної довжини.

Як мовилося вище, функція InitTable() ініціалізувала таблицю рядків так, щоб вона містила всі можливі рядки, що складаються з одного символу. Наприклад, якщо ми стискуємо байтові дані, то таких рядків у таблиці буде 256 ('0', '1' ..., '255'). Для коду очищення (ClearCode) і коду кінця інформації (Code EndOf Information) зарезервовані значення 256 і 257. У даному варіанті алгоритму використовується 12-бітовий код, і, відповідно, під коди для рядків нам залишаються значення від 258 до 4095. Рядки, що додаються, записуються в таблицю послідовно, при цьому індекс рядка в таблиці стає її кодом.

Функція ReadNextByte() читає символ з файла. Функція WriteCode() записує код (не рівний за розміром байту) у вихідний файл. Функція AddStringToTable() додає новий рядок у таблицю, приписуючи їй код. Крім того, в даній функції відбувається обробка ситуації переповнювання таблиці.

В цьому випадку в потік записується код попереднього знайденого рядка і код очищення, після чого таблиця очищається функцією InitTable(). Функція CodeForString() знаходить рядок у таблиці і видає код цього рядка.

Приклад 4.2.

Хай ми стискуємо послідовність 45, 55, 55, 151, 55, 55, 55. Тоді, згідно з викладеним вище алгоритмом, ми помістимо у вихідний потік спочатку код очищення <256>, потім додамо до початку порожнього рядка '45' і перевіримо, чи є рядок '45' у таблиці. Оскільки ми під час ініціалізації занесли в таблицю всі рядки з одного символу, то рядок '45' є в таблиці.

Далі ми читаємо наступний символ 55 з вхідного потоку і перевіряємо, чи є рядок '45, 55' в таблиці. Такого рядка в таблиці поки немає. Ми заносимо в таблицю рядок '45, 55' (з першим вільним кодом 258) і записуємо в потік код <45>. Можна коротко подати архівацію так:

- '45' - є в таблиці;
- '45, 55' - немає. Додаємо в таблицю <258>'45, 55'. У потік: <45>;
- '55, 55' - немає. У таблицю: <259>'55, 55'. У потік: <55>;
- '55, 151' - немає. У таблицю: <260>'55, 151'. У потік: <55>;
- '151, 55' - немає. У таблицю: <261>'151, 55'. У потік: <151>;
- '55, 55' - є в таблиці;
- '55, 55, 55' - немає. У таблицю: '55, 55, 55' <262>. У потік: <259>;

Послідовність кодів для даного прикладу, що потрапляють у вихідний потік: <256>, <45>, <55>, <55>, <151>, <259>.

Особливість LZW полягає в тому, що для декомпресії нам не треба зберігати таблицю рядків у файл для розпаковування. Алгоритм побудований таким чином, що ми в змозі відновити таблицю рядків, користуючись тільки потоком коду.

Ми знаємо, що для кожного коду треба додавати в таблицю рядок, що складається з уже присутнього там рядка і символу, з якого починається наступний рядок в потоці.

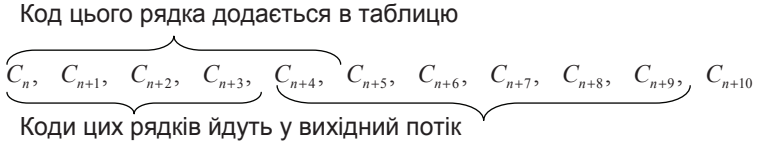


Рис. 4.3. Пояснення методу LZW

Алгоритм декомпресії, що здійснює цю операцію, виглядає таким чином:

```
code=File.ReadCode();
while(code != Codeendofinformation){
  if(code = Clearcode) {
    InitTable();
    code=File.ReadCode();
    if(code = Codeendofinformation)
      {закінчити роботу};
    ImageFile.WriteString(StrFromTable(code));
    old_code=code;    }
  else {
    if(InTable(code)) {
      ImageFile.WriteString(FromTable(code));
      AddStringToTable(StrFromTable(old_code)+
```


- '45' - є в таблиці;
- '45, 55' - немає. Додаємо в таблицю <258>'45, 55'. У потік: <45>;
- '55, 55' - немає. У таблицю: <259>'55, 55'. У потік: <55>;
- '55, 151' - немає. У таблицю: <260>'55, 151'. У потік: <55>;
- '151, 55' - немає. У таблицю: <261>'151, 55'. У потік: <151>;
- '55, 55' - є в таблиці;
- '55, 55, 55' - немає. У таблицю: '55, 55, 55' <262>. У потік: <259>;

Послідовність кодів для даного прикладу, що потрапляють у вихідний потік: <256>, <45>, <55>, <55>, <151>, <259>.

Особливість LZW полягає в тому, що для декомпресії нам не треба зберігати таблицю рядків у файл для розпаковування. Алгоритм побудований таким чином, що ми в змозі відновити таблицю рядків, користуючись тільки потоком коду.

Ми знаємо, що для кожного коду треба додавати в таблицю рядок, що складається з уже присутнього там рядка і символу, з якого починається наступний рядок в потоці.

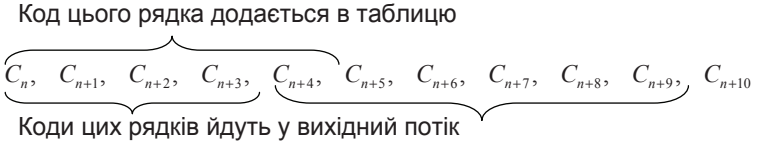


Рис. 4.3. Пояснення методу LZW

Алгоритм декомпресії, що здійснює цю операцію, виглядає таким чином:

```
code=File.ReadCode();
while(code != Codeendofinformation){
  if(code = Clearcode) {
    InitTable();
    code=File.ReadCode();
    if(code = Codeendofinformation)
      {закінчити роботу};
    ImageFile.WriteString(StrFromTable(code));
    old_code=code;    }
  else {
    if(InTable(code)) {
      ImageFile.WriteString(FromTable(code));
      AddStringToTable(StrFromTable(old_code)+
```

```
    FirstChar(StrFromTable(code)));
    old_code=code; }
   }
  else {
    OutString=StrFromTable(old_code)+
    FirstChar(StrFromTable(old_code));
    ImageFile.WriteString(OutString);
    AddStringToTable(OutString);
    old_code=code; } }
```

Тут функція ReadCode() читає черговий код з файла. Функція InitTable() виконує ті ж дії, що і при компресії, тобто очищає таблицю і заносить в неї всі рядки з одного символу. Функція FirstChar() видає нам перший символ рядка. Функція StrFromTable() видає рядок з таблиці за кодом. Функція AddStringToTable() додає новий рядок у таблицю (привласнюючи їй перший вільний код). Функція WriteString() записує рядок у файл.

Алгоритм JPEG

JPEG - один з найновіших і достатньо потужних алгоритмів. Практично він є стандартом де-факто для повнокольорових зображень [3]. Оперує алгоритм областями 8x8, на яких яскравість і колір змінюються порівняно плавно. Внаслідок цього в разі розкладання матриці такої області в подвійний рядок по косинусах (4.1), значущими виявляються тільки перші коефіцієнти. Таким чином, стискування в JPEG здійснюється за рахунок плавності зміни кольорів у зображенні.

В цілому алгоритм заснований на дискретному косинусоїдальному перетворенні (надалі ДКП), що застосовується до матриці зображення для отримання деякої нової матриці коефіцієнтів. Для отримання початкового зображення застосовується зворотне перетворення.

ДКП розкладає зображення по амплітудах деяких частот. Таким чином, при перетворенні ми отримуємо матрицю, в якій багато коефіцієнтів або близькі, або дорівнюють нулю. Крім того, завдяки недосконалості людського зору, можна апроксимувати коефіцієнти грубіше без помітної втрати якості зображення.


```
    FirstChar(StrFromTable(code)));
    old_code=code; }
   }
  else {
    OutString=StrFromTable(old_code)+
    FirstChar(StrFromTable(old_code));
    ImageFile.WriteString(OutString);
    AddStringToTable(OutString);
    old_code=code; } }
```

Тут функція ReadCode() читає черговий код з файла. Функція InitTable() виконує ті ж дії, що і при компресії, тобто очищає таблицю і заносить в неї всі рядки з одного символу. Функція FirstChar() видає нам перший символ рядка. Функція StrFromTable() видає рядок з таблиці за кодом. Функція AddStringToTable() додає новий рядок у таблицю (привласнюючи їй перший вільний код). Функція WriteString() записує рядок у файл.

Алгоритм JPEG

JPEG - один з найновіших і достатньо потужних алгоритмів. Практично він є стандартом де-факто для повнокольорових зображень [3]. Оперує алгоритм областями 8x8, на яких яскравість і колір змінюються порівняно плавно. Внаслідок цього в разі розкладання матриці такої області в подвійний рядок по косинусах (4.1), значущими виявляються тільки перші коефіцієнти. Таким чином, стискування в JPEG здійснюється за рахунок плавності зміни кольорів у зображенні.

В цілому алгоритм заснований на дискретному косинусоїдальному перетворенні (надалі ДКП), що застосовується до матриці зображення для отримання деякої нової матриці коефіцієнтів. Для отримання початкового зображення застосовується зворотне перетворення.

ДКП розкладає зображення по амплітудах деяких частот. Таким чином, при перетворенні ми отримуємо матрицю, в якій багато коефіцієнтів або близькі, або дорівнюють нулю. Крім того, завдяки недосконалості людського зору, можна апроксимувати коефіцієнти грубіше без помітної втрати якості зображення.

Для цього використовується квантування коефіцієнтів (quantization). У найпростішому випадку - це арифметичне побітове зрушення вправо. При цьому перетворенні втрачається частка інформації, але можуть досягатися великі коефіцієнти стискування.

Припустимо ми стискуємо 24-бітове зображення.

Крок 1.

Переводимо зображення з колірного простору RGB з компонентами, що відповідають за червону (Red), зелену (Green) і синю (Blue) складові кольору точки, у колірний простір YCrCb (інколи називають YUV).

В ньому Y - складова яскравості, а Cr, Cb - компоненти, що відповідають за колір (хроматичний червоний і хроматичний синій)[3]. За рахунок того, що людське око менш чутливе до кольору, чим до яскравості, з'являється можливість архівувати масиви для Cr і Cb компонент з великими втратами і, відповідно, великими коефіцієнтами стискування. Подібне перетворення вже давно використовується в телебаченні. На сигнали, що відповідають за колір, там виділяється вужча смуга частот.

Спрощено перетворення з колірного простору RGB в колірний простір YCrCb можна показати за допомогою матриці переходу

$$\begin{vmatrix} Y \\ C_b \\ C_r \end{vmatrix} = \begin{vmatrix} 0{,}299 & 0{,}587 & 0{,}114 \\ 0{,}5 & -0{,}4187 & -0{,}0813 \\ 0{,}1687 & -0{,}3313 & 0{,}5 \end{vmatrix} * \begin{vmatrix} R \\ G \\ B \end{vmatrix} + \begin{vmatrix} 0 \\ 128 \\ 128 \end{vmatrix} \qquad (4.1)$$

Зворотне перетворення здійснюється множенням вектора YUV на зворотну матрицю.

$$\begin{vmatrix} R \\ G \\ B \end{vmatrix} = \begin{vmatrix} 1 & 1 & 1{,}402 \\ 1 & -0{,}34414 & -0{,}71414 \\ 1 & 1{,}772 & 0 \end{vmatrix} * \left(\begin{vmatrix} Y \\ C_b \\ C_r \end{vmatrix} - \begin{vmatrix} 0 \\ 128 \\ 128 \end{vmatrix} \right) \qquad (4.2)$$

Крок 2.

Зображення розділяється на блоки розміром 8x8 пікселів, і кожний такий блок обробляється двовимірним *дискретним косинусним перетворенням (ДКП) — Discrete Cosine Transform (DCT).* Це перетворення має такий вигляд:

Для цього використовується квантування коефіцієнтів (quantization). У найпростішому випадку - це арифметичне побітове зрушення вправо. При цьому перетворенні втрачається частка інформації, але можуть досягатися великі коефіцієнти стискування.

Припустимо ми стискуємо 24-бітове зображення.

Крок 1.

Переводимо зображення з колірного простору RGB з компонентами, що відповідають за червону (Red), зелену (Green) і синю (Blue) складові кольору точки, у колірний простір YCrCb (інколи називають YUV).

В ньому Y - складова яскравості, а Cr, Cb - компоненти, що відповідають за колір (хроматичний червоний і хроматичний синій)[3]. За рахунок того, що людське око менш чутливе до кольору, чим до яскравості, з'являється можливість архівувати масиви для Cr і Cb компонент з великими втратами і, відповідно, великими коефіцієнтами стискування. Подібне перетворення вже давно використовується в телебаченні. На сигнали, що відповідають за колір, там виділяється вужча смуга частот.

Спрощено перетворення з колірного простору RGB в колірний простір YCrCb можна показати за допомогою матриці переходу

$$\begin{vmatrix} Y \\ C_b \\ C_r \end{vmatrix} = \begin{vmatrix} 0{,}299 & 0{,}587 & 0{,}114 \\ 0{,}5 & -0{,}4187 & -0{,}0813 \\ 0{,}1687 & -0{,}3313 & 0{,}5 \end{vmatrix} * \begin{vmatrix} R \\ G \\ B \end{vmatrix} + \begin{vmatrix} 0 \\ 128 \\ 128 \end{vmatrix} \qquad (4.1)$$

Зворотне перетворення здійснюється множенням вектора YUV на зворотну матрицю.

$$\begin{vmatrix} R \\ G \\ B \end{vmatrix} = \begin{vmatrix} 1 & 1 & 1{,}402 \\ 1 & -0{,}34414 & -0{,}71414 \\ 1 & 1{,}772 & 0 \end{vmatrix} * \left(\begin{vmatrix} Y \\ C_b \\ C_r \end{vmatrix} - \begin{vmatrix} 0 \\ 128 \\ 128 \end{vmatrix} \right) \qquad (4.2)$$

Крок 2.

Зображення розділяється на блоки розміром 8x8 пікселів, і кожний такий блок обробляється двовимірним *дискретним косинусним перетворенням (ДКП) — Discrete Cosine Transform (DCT).* Це перетворення має такий вигляд:

$$F(u,v)=\frac{1}{4}C(u)C(v)\sum_{x=0}^{7}\sum_{y=0}^{7}f(x,y)\cos\frac{(2x+1)u\pi}{16}\cos\frac{(2y+1)v\pi}{16}. \quad (4.3)$$

де: $C(u)$, $C(v) = \frac{1}{\sqrt{2}}$ для u, $v = 0$. C(u), C(v) = 1 в інших випадках.

Перетворення виконується окремо для компонент *Y, Cb* та *Cr,* причому для *Y* обробляється весь блок 8x8, а для *Cb* та *Cr* робиться проріджування – береться кожний другий піксел. Що дає ДКП? Воно перетворює просторове розподілення в частотне (подібно перетворенню Фур'є). У результаті отримуємо блоки також розмірами 8x8, однак кожний елемент *F(u,* v) тут є частотним коефіцієнтом спектра, причому коефіцієнти для нижніх частот розташовуються в лівому верхньому куті, а для високих частот – у правому нижньому куті. Оскільки основна енергія в спектрі – у нижніх частотах, то максимальне числове значення буде в лівому верхньому куті.

Перетворення ДКП саме по собі не призводить до втрат інформації, однак якщо елементи спектра *(F(u,* v)) записуються не як дробові, а округлюються до цілих, то це вже дасть деяку помилку при декодуванні.

Крок 3.

Квантування. Кожний елемент блока 8x8 після ДКП ділитиметься на відповідний елемент матриці квантування

$$F_q(u,v)=\left[\frac{F(u,v)}{Q(u,v)}\right],$$

де *Q(u,* v) — матриця квантування, елементами якої є числа від 1 до 255. Після ділення виконується округлення до найближчого цілого числа.

У стандарті JPEG є рекомендовані таблиці квантування (рис. 4.4), окремі для *Y* та *Cb, Cr* .

Для *Y*: Для *Cb, Cr:*

$$F(u,v)=\frac{1}{4}C(u)C(v)\sum_{x=0}^{7}\sum_{y=0}^{7}f(x,y)\cos\frac{(2x+1)u\pi}{16}\cos\frac{(2y+1)v\pi}{16}. \quad (4.3)$$

де: $C(u)$, $C(v) = \frac{1}{\sqrt{2}}$ для u, $v = 0$. C(u), C(v) = 1 в інших випадках.

Перетворення виконується окремо для компонент *Y, Cb* та *Cr,* причому для *Y* обробляється весь блок 8x8, а для *Cb* та *Cr* робиться проріджування – береться кожний другий піксел. Що дає ДКП? Воно перетворює просторове розподілення в частотне (подібно перетворенню Фур'є). У результаті отримуємо блоки також розмірами 8x8, однак кожний елемент *F(u,* v) тут є частотним коефіцієнтом спектра, причому коефіцієнти для нижніх частот розташовуються в лівому верхньому куті, а для високих частот – у правому нижньому куті. Оскільки основна енергія в спектрі – у нижніх частотах, то максимальне числове значення буде в лівому верхньому куті.

Перетворення ДКП саме по собі не призводить до втрат інформації, однак якщо елементи спектра *(F(u,* v)) записуються не як дробові, а округлюються до цілих, то це вже дасть деяку помилку при декодуванні.

Крок 3.

Квантування. Кожний елемент блока 8x8 після ДКП ділитиметься на відповідний елемент матриці квантування

$$F_q(u,v)=\left[\frac{F(u,v)}{Q(u,v)}\right],$$

де *Q(u,* v) — матриця квантування, елементами якої є числа від 1 до 255. Після ділення виконується округлення до найближчого цілого числа.

У стандарті JPEG є рекомендовані таблиці квантування (рис. 4.4), окремі для *Y* та *Cb, Cr* .

Для *Y*: Для *Cb, Cr:*

16	12	14	14	18	24	49	72
11	12	13	17	22	35	64	92
10	14	16	22	37	55	78	95
16	19	24	29	56	64	87	98
24	26	40	51	68	81	103	112
40	58	57	87	109	104	121	100
51	60	69	80	103	113	120	103
61	55	56	62	77	92	101	99

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

Рис. 4.4. Таблиці квантування

Зверніть увагу на те, що значення коефіцієнтів квантування є меншими в лівій верхній частині таблиць порівняно з коефіцієнтами у правій нижній частині. Фактично таблиця квантування визначає цифровий фільтр, що ослаблює верхні частоти.

Вказані в стандарті таблиці не є обов'язковими – будь-яка програма запису файлів JPEG може використовувати власні таблиці (зрозуміло, що тоді ці таблиці повинні бути записані у файл, інакше неможливе декодування).

Квантування — це основний фактор ущільнення згідно з методом JPEG. У результаті ділення й округлення більшість елементів блока 8x8 дорівнюватимуть нулю, що сприяє компактному запису блока. Чим більшим є значення таблиці квантування, тим вірогідніше перетворення результату поділу в нуль. Однак це призводить до втрат інформації — чим більше значущих елементів перетворюється в нулі, тим менше результат декодування буде схожий на первісне зображення.

Крок 4.

Елементи блока 8x8 записуються у вигляді одномірного вектора байтів. Елементи обираються "зигзагом", як наведено на рис. 4.5.

Крок 5.

16	12	14	14	18	24	49	72
11	12	13	17	22	35	64	92
10	14	16	22	37	55	78	95
16	19	24	29	56	64	87	98
24	26	40	51	68	81	103	112
40	58	57	87	109	104	121	100
51	60	69	80	103	113	120	103
61	55	56	62	77	92	101	99

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

Рис. 4.4. Таблиці квантування

Зверніть увагу на те, що значення коефіцієнтів квантування є меншими в лівій верхній частині таблиць порівняно з коефіцієнтами у правій нижній частині. Фактично таблиця квантування визначає цифровий фільтр, що ослаблює верхні частоти.

Вказані в стандарті таблиці не є обов'язковими – будь-яка програма запису файлів JPEG може використовувати власні таблиці (зрозуміло, що тоді ці таблиці повинні бути записані у файл, інакше неможливе декодування).

Квантування — це основний фактор ущільнення згідно з методом JPEG. У результаті ділення й округлення більшість елементів блока 8x8 дорівнюватимуть нулю, що сприяє компактному запису блока. Чим більшим є значення таблиці квантування, тим вірогідніше перетворення результату поділу в нуль. Однак це призводить до втрат інформації — чим більше значущих елементів перетворюється в нулі, тим менше результат декодування буде схожий на первісне зображення.

Крок 4.

Елементи блока 8x8 записуються у вигляді одномірного вектора байтів. Елементи обираються "зигзагом", як наведено на рис. 4.5.

Крок 5.

Кодування вектора. Спочатку вектор кодується методом RLE. Утворюється код у вигляді таких пар: (лічильник, значення). Тут лічильник означає кількість нулів, які записуються попереду значення. Потім виконується кодування пар методом Хаффмана. У файл JPEG, окрім ущільнених кодів, записується також таблиця Хаффмана для забезпечення подальшого декодування.

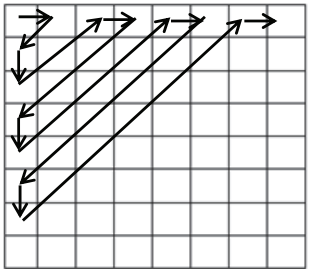


Рис. 4.5. Формування одномірного вектора

Метод ущільнення Хаффмана, розроблений у 1952 році, використовується як складова частина в ряді схем ущільнення, таких як LZ. Згідно з цим методом спочатку для кожного символу обчислюється вірогідність його появи. Потім символу приписується бітовий код, довжина якого залежить від цієї вірогідності. Символи, які частіше зустрічаються, отримують коротший бітовий код, а символи, які зустрічаються рідше, – довший код [14].

Тепер розглянемо процес JPEG-декодування. Виконується у зворотному порядку.

Крок 1.

Декодування RLE та Хаффмана для кодів елементів блоків 8×8.

Крок 2.

Кожний елемент блоків 8×8 компонент YCbCr множимо на елемент відповідної матриці квантування.

Крок 3.

Виконуємо *зворотне дискретне косинусне перетворення (ЗДКП) — Inverse Disrete Cosine Transform (IDCT):*

$$f(x,y)=\frac{1}{4}\sum_{x=0}^{7}\sum_{y=0}^{7}C(u)C(v)F(u,v)\cos\frac{(2x+1)u\pi}{16}\cos\frac{(2y+1)v\pi}{16},$$

де *C(u), C(v)* такі самі, як для прямого дискретне косинусне перетворення.

Крок 4.

Перетворюємо YCbCr у RGB за виразом (4.2).Конвеєр операцій, використовуваний в алгоритмі JPEG зображено на рис. 4.6.

Кодування вектора. Спочатку вектор кодується методом RLE. Утворюється код у вигляді таких пар: (лічильник, значення). Тут лічильник означає кількість нулів, які записуються попереду значення. Потім виконується кодування пар методом Хаффмана. У файл JPEG, окрім ущільнених кодів, записується також таблиця Хаффмана для забезпечення подальшого декодування.

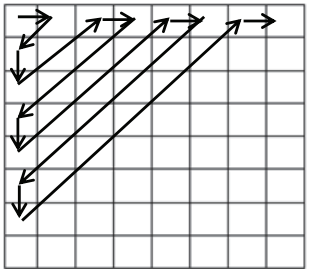


Рис. 4.5. Формування одномірного вектора

Метод ущільнення Хаффмана, розроблений у 1952 році, використовується як складова частина в ряді схем ущільнення, таких як LZ. Згідно з цим методом спочатку для кожного символу обчислюється вірогідність його появи. Потім символу приписується бітовий код, довжина якого залежить від цієї вірогідності. Символи, які частіше зустрічаються, отримують коротший бітовий код, а символи, які зустрічаються рідше, – довший код [14].

Тепер розглянемо процес JPEG-декодування. Виконується у зворотному порядку.

Крок 1.

Декодування RLE та Хаффмана для кодів елементів блоків 8×8.

Крок 2.

Кожний елемент блоків 8×8 компонент YCbCr множимо на елемент відповідної матриці квантування.

Крок 3.

Виконуємо *зворотне дискретне косинусне перетворення (ЗДКП) — Inverse Disrete Cosine Transform (IDCT):*

$$f(x,y)=\frac{1}{4}\sum_{x=0}^{7}\sum_{y=0}^{7}C(u)C(v)F(u,v)\cos\frac{(2x+1)u\pi}{16}\cos\frac{(2y+1)v\pi}{16},$$

де *C(u), C(v)* такі самі, як для прямого дискретне косинусне перетворення.

Крок 4.

Перетворюємо YCbCr у RGB за виразом (4.2).Конвеєр операцій, використовуваний в алгоритмі JPEG зображено на рис. 4.6.

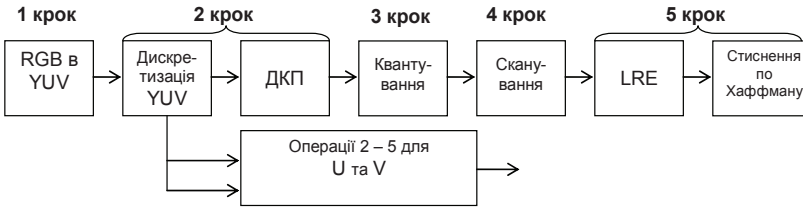


Рис. 4.6.Конвеєр операцій, використовуваний в алгоритмі JPEG

Змінювати ступінь ущільнення для формату JPEG досить просто. Достатньо задавати число, на яке помножуються коефіцієнти таблиць квантування при кодуванні та декодуванні. Наприклад, якщо коефіцієнти таблиць квантування поділити на два, то ущільнення зменшиться, а якість зображення буде значно вища.

Істотні позитивні сторони алгоритму:

1. Задається ступінь стискування.
2. Вихідне кольорове зображення може мати 24 біти на точку.

Негативні сторони алгоритму:

1. При підвищенні ступеня стискування зображення розпадається на окремі квадрати (8x8). Це пов'язано з тим, що відбуваються великі втрати в низьких частотах при квантуванні, і відновити початкові дані стає неможливо.
2. Виявляється ефект Гіббса - ореоли по кордонах різких переходів кольорів.

Фрактальний алгоритм

Фрактальне кодування – це математичний процес, що застосовується для кодування растрів, які містять реальне зображення, в сукупність математичних даних, що описують фрактальні властивості зображення. Фрактальне кодування основане на тому, що всі реальні та більшість штучних об’єктів містять надлишкову інформацію у вигляді однакових рисунків, що повторюються, які називаються фракталами [14, 17].

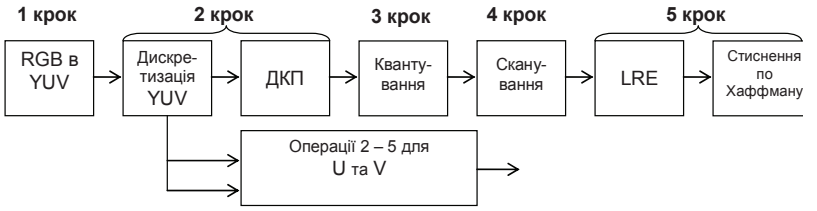


Рис. 4.6.Конвеєр операцій, використовуваний в алгоритмі JPEG

Змінювати ступінь ущільнення для формату JPEG досить просто. Достатньо задавати число, на яке помножуються коефіцієнти таблиць квантування при кодуванні та декодуванні. Наприклад, якщо коефіцієнти таблиць квантування поділити на два, то ущільнення зменшиться, а якість зображення буде значно вища.

Істотні позитивні сторони алгоритму:

1. Задається ступінь стискування.
2. Вихідне кольорове зображення може мати 24 біти на точку.

Негативні сторони алгоритму:

1. При підвищенні ступеня стискування зображення розпадається на окремі квадрати (8x8). Це пов'язано з тим, що відбуваються великі втрати в низьких частотах при квантуванні, і відновити початкові дані стає неможливо.
2. Виявляється ефект Гіббса - ореоли по кордонах різких переходів кольорів.

Фрактальний алгоритм

Фрактальне кодування – це математичний процес, що застосовується для кодування растрів, які містять реальне зображення, в сукупність математичних даних, що описують фрактальні властивості зображення. Фрактальне кодування основане на тому, що всі реальні та більшість штучних об’єктів містять надлишкову інформацію у вигляді однакових рисунків, що повторюються, які називаються фракталами [14, 17].

Якщо ми зробимо копію невеликої частини зображення та порівняємо її з іншими частинами зображення, то можна побачити декілька областей, які майже збігаються з обраною копією. Змінюючи розміри та орієнтацію цієї копії та описавши ці процеси математично, можна отримати фрактальні коди, що описують зображення.

Фрактальна архівація заснована на тому, що ми подаємо зображення в компактнішій формі - за допомогою коефіцієнтів системи інтегрованих функцій (Iterated Function System - далі по тексту як IFS).

Перш ніж розглядати сам процес архівації, розберемо, як IFS будує зображення, тобто процес декомпресії. Строго кажучи, IFS є набором тривимірних афінних перетворень, в нашому випадку тих, що переводять одне зображення в інше. Перетворенню підлягають точки у тривимірному просторі (x-координата, y-координата, яскравість).

Найнаочніше цей процес продемонстрував Барнслі у своїй книзі «Fractal Image Compression». Там введено поняття Машини, що Фотокопіює, яка складається з екрана, на якому змальована початкова картинка, і системи лінз, що проектують зображення на інший екран (рис. 4.7):

- Лінзи можуть проектувати частку зображення довільної форми в будь-яке інше місце нового зображення.
- Області, в які проектуються зображення, не перетинаються.
- Лінза може міняти яскравість і зменшувати контрастність.
- Лінза може дзеркально відображати і повертати свій фрагмент зображення.
- Лінза повинна масштабувати (зменшувати) свій фрагмент зображення.

Якщо ми зробимо копію невеликої частини зображення та порівняємо її з іншими частинами зображення, то можна побачити декілька областей, які майже збігаються з обраною копією. Змінюючи розміри та орієнтацію цієї копії та описавши ці процеси математично, можна отримати фрактальні коди, що описують зображення.

Фрактальна архівація заснована на тому, що ми подаємо зображення в компактнішій формі - за допомогою коефіцієнтів системи інтегрованих функцій (Iterated Function System - далі по тексту як IFS).

Перш ніж розглядати сам процес архівації, розберемо, як IFS будує зображення, тобто процес декомпресії. Строго кажучи, IFS є набором тривимірних афінних перетворень, в нашому випадку тих, що переводять одне зображення в інше. Перетворенню підлягають точки у тривимірному просторі (x-координата, y-координата, яскравість).

Найнаочніше цей процес продемонстрував Барнслі у своїй книзі «Fractal Image Compression». Там введено поняття Машини, що Фотокопіює, яка складається з екрана, на якому змальована початкова картинка, і системи лінз, що проектують зображення на інший екран (рис. 4.7):

- Лінзи можуть проектувати частку зображення довільної форми в будь-яке інше місце нового зображення.
- Області, в які проектуються зображення, не перетинаються.
- Лінза може міняти яскравість і зменшувати контрастність.
- Лінза може дзеркально відображати і повертати свій фрагмент зображення.
- Лінза повинна масштабувати (зменшувати) свій фрагмент зображення.

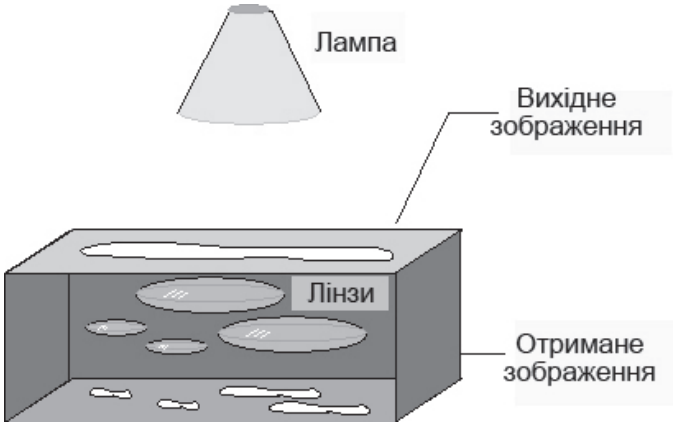


Рис. 4.7. До принципу фрактального стиснення

Розставляючи лінзи і міняючи їх характеристики, ми можемо управляти зображенням, що отримується. Одна ітерація роботи Машини полягає в тому, що по початковому зображенню за допомогою проектування будується нове, після чого нове береться як початкове. Зрештою в процесі ітерацій ми отримаємо зображення, яке перестане змінюватися. Воно залежатиме тільки від розташування і характеристик лінз, і не залежатиме від початкової картинки.

Це зображення називається 'Нерухомою точкою' або *аттрактором* даної IFS. Відповідна теорія гарантує наявність рівно однієї нерухомої точки для кожної IFS.

Оскільки відображення лінз є таким, що стискує, кожна лінза в явному вигляді задає самоподібні області в нашому зображенні. Завдяки самоподібності ми отримуємо складну структуру зображення при будь-якому збільшенні. Таким чином, інтуїтивно зрозуміло, що система інтегрованих функцій задає фрактал (нестрого - самоподібний математичний об'єкт).

Найбільш відомо два зображення, отриманих за допомогою IFS: «трикутник Серпінського» і «папоротник Барнслі». «Трикутник Серпінського» задається трьома, а «папоротник Барнслі» чотирма афінними перетвореннями (в нашій термінології, «лінзами»).

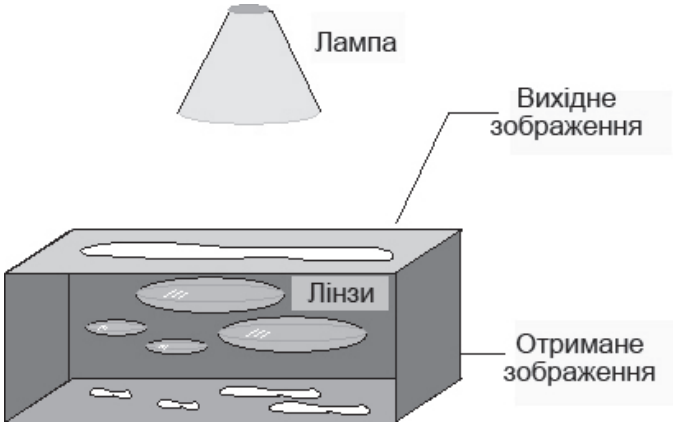


Рис. 4.7. До принципу фрактального стиснення

Розставляючи лінзи і міняючи їх характеристики, ми можемо управляти зображенням, що отримується. Одна ітерація роботи Машини полягає в тому, що по початковому зображенню за допомогою проектування будується нове, після чого нове береться як початкове. Зрештою в процесі ітерацій ми отримаємо зображення, яке перестане змінюватися. Воно залежатиме тільки від розташування і характеристик лінз, і не залежатиме від початкової картинки.

Це зображення називається 'Нерухомою точкою' або *аттрактором* даної IFS. Відповідна теорія гарантує наявність рівно однієї нерухомої точки для кожної IFS.

Оскільки відображення лінз є таким, що стискує, кожна лінза в явному вигляді задає самоподібні області в нашому зображенні. Завдяки самоподібності ми отримуємо складну структуру зображення при будь-якому збільшенні. Таким чином, інтуїтивно зрозуміло, що система інтегрованих функцій задає фрактал (нестрого - самоподібний математичний об'єкт).

Найбільш відомо два зображення, отриманих за допомогою IFS: «трикутник Серпінського» і «папоротник Барнслі». «Трикутник Серпінського» задається трьома, а «папоротник Барнслі» чотирма афінними перетвореннями (в нашій термінології, «лінзами»).

Кожне перетворення кодується буквально ліченими байтами, тоді як зображення, побудоване з їх допомогою, може займати і декілька мегабайтів. З вищесказаного стає зрозуміло, як працює архіватор і чому йому вимагається так багато часу. Фактично, фрактальна компресія – це пошук самоподібних областей у зображенні і визначення для них параметрів афінних перетворень.

У процесі перетворення звичайних растрових зображень у фрактальні дані відразу ж реалізуються дві величезні переваги. Перша – це можливість масштабувати фрактальне зображення без введення артефактів і втрати деталей, як це характерно для растрових зображень. Цей процес "фрактального панорамування" не залежить від дозволу вихідного растрового зображення, і масштаб обмежується тільки об'ємом вільної пам'яті комп'ютера.

Рис. 4.8. Зовнішній вигляд папоротника Барнслі

Друга перевага полягає в тому, що розмір фізичних даних, що використовуються для запису фрактальних кодів, значно менше розміру вихідних растрових даних. Іноді фрактальне зображення більш ніж у 100 разів менше від свого растрового оригіналу. Саме цей аспект фрактальної технології, що називається фрактальним стисненням, викликав найбільший інтерес у сфері формування та відтворення комп'ютерних зображень.

Фрактальне стиснення супроводжується втратами. Процес порівняння фракталів не передбачає пошуку точної їх відповідності. Замість цього шукається "найкраща" відповідність на підставі параметрів стиснення (часу кодування, якості зображення, розміру

Кожне перетворення кодується буквально ліченими байтами, тоді як зображення, побудоване з їх допомогою, може займати і декілька мегабайтів. З вищесказаного стає зрозуміло, як працює архіватор і чому йому вимагається так багато часу. Фактично, фрактальна компресія – це пошук самоподібних областей у зображенні і визначення для них параметрів афінних перетворень.

У процесі перетворення звичайних растрових зображень у фрактальні дані відразу ж реалізуються дві величезні переваги. Перша – це можливість масштабувати фрактальне зображення без введення артефактів і втрати деталей, як це характерно для растрових зображень. Цей процес "фрактального панорамування" не залежить від дозволу вихідного растрового зображення, і масштаб обмежується тільки об'ємом вільної пам'яті комп'ютера.

Рис. 4.8. Зовнішній вигляд папоротника Барнслі

Друга перевага полягає в тому, що розмір фізичних даних, що використовуються для запису фрактальних кодів, значно менше розміру вихідних растрових даних. Іноді фрактальне зображення більш ніж у 100 разів менше від свого растрового оригіналу. Саме цей аспект фрактальної технології, що називається фрактальним стисненням, викликав найбільший інтерес у сфері формування та відтворення комп'ютерних зображень.

Фрактальне стиснення супроводжується втратами. Процес порівняння фракталів не передбачає пошуку точної їх відповідності. Замість цього шукається "найкраща" відповідність на підставі параметрів стиснення (часу кодування, якості зображення, розміру

виведення). Процесом кодування, однак, можна управляти, доводячи його до стану, в якому зображення "візуально не має втрат". Тобто ви не повинні помітити втрати даних.

Фрактальне стиснення відрізняється від інших методів стиску з втратами (наприклад, від JPEG). Метод JPEG дозволяє стискати, відкидаючи ті дані зображення, які не важливі для сприйняття людським оком. Отримані в результаті дані надалі обробляються з допомогою методу стиснення без втрат. Для досягнення високого ступеня стиснення необхідно відкидати більше даних, що призводить до погіршення якості зображення і появи дискретних (блокових) ділянок.

Фрактальні зображення не будуються на базі растра, а кодування не порівнюється з візуальними характеристиками людського ока. Навпаки, растрові дані відкидаються, якщо необхідно створити найкращий фрактальний рисунок. Високий ступінь стиснення досягається шляхом виконання більшого обсягу перетворень і обчислень, що може погіршити якість зображення, проте завдяки фрактальним компонентам викривлення не такі помітні.

Більшість інших методів стиснення із втратами за своїм характером симетричні. Це означає, що вони засновані на використанні конкретної послідовності операцій, які при розпакуванні виконуються в зворотному порядку. Крім того, на стиснення і розпакування потрібно приблизно однаковий час. Фрактальне стиснення - процес асиметричний. Стиснення триває набагато довше, ніж розпакування. Ця характеристика переносить "фокус корисності" фрактально стиснених даних на додатки, в яких дані зображень безупинно розпаковуються, але ніколи не стискуються. Очевидно, що фрактальну компресію доцільно застосовувати в базах даних зображень та додатках з інформаційною базою на CD-ROM.

На ступінь фрактального стиснення помітний вплив чинять вміст і дозвіл вихідного растра. Зображення з високим вмістом фрактальних елементів (наприклад, портрети, пейзажі і складні текстури) характеризуються більш високим ступенем стиснення, ніж зображення з низьким вмістом таких елементів (такі, як графіки, схеми, текст і прості текстури). Зображення з високою роздільною здатністю можна стискати з високим ступенем стиснення, не впливаючи

виведення). Процесом кодування, однак, можна управляти, доводячи його до стану, в якому зображення "візуально не має втрат". Тобто ви не повинні помітити втрати даних.

Фрактальне стиснення відрізняється від інших методів стиску з втратами (наприклад, від JPEG). Метод JPEG дозволяє стискати, відкидаючи ті дані зображення, які не важливі для сприйняття людським оком. Отримані в результаті дані надалі обробляються з допомогою методу стиснення без втрат. Для досягнення високого ступеня стиснення необхідно відкидати більше даних, що призводить до погіршення якості зображення і появи дискретних (блокових) ділянок.

Фрактальні зображення не будуються на базі растра, а кодування не порівнюється з візуальними характеристиками людського ока. Навпаки, растрові дані відкидаються, якщо необхідно створити найкращий фрактальний рисунок. Високий ступінь стиснення досягається шляхом виконання більшого обсягу перетворень і обчислень, що може погіршити якість зображення, проте завдяки фрактальним компонентам викривлення не такі помітні.

Більшість інших методів стиснення із втратами за своїм характером симетричні. Це означає, що вони засновані на використанні конкретної послідовності операцій, які при розпакуванні виконуються в зворотному порядку. Крім того, на стиснення і розпакування потрібно приблизно однаковий час. Фрактальне стиснення - процес асиметричний. Стиснення триває набагато довше, ніж розпакування. Ця характеристика переносить "фокус корисності" фрактально стиснених даних на додатки, в яких дані зображень безупинно розпаковуються, але ніколи не стискуються. Очевидно, що фрактальну компресію доцільно застосовувати в базах даних зображень та додатках з інформаційною базою на CD-ROM.

На ступінь фрактального стиснення помітний вплив чинять вміст і дозвіл вихідного растра. Зображення з високим вмістом фрактальних елементів (наприклад, портрети, пейзажі і складні текстури) характеризуються більш високим ступенем стиснення, ніж зображення з низьким вмістом таких елементів (такі, як графіки, схеми, текст і прості текстури). Зображення з високою роздільною здатністю можна стискати з високим ступенем стиснення, не впливаючи

на їх якість. Для збереження високої якості зображень з низьким дозволом підсумковий ступінь стиску повинен бути набагато нижче. Відзначимо, що зображення з більшою бітовою глибиною (наприклад, 24-бітові truecolor-зображення) стискаються більш ефективно, ніж зображення з меншою кількістю бітів на піксел (зокрема, 8-бітові напівтонові). Процес фрактального стиснення не призначений для громадського користування. Є багато патентів на методи стиснення даних на базі фрактальних перетворень. Крім того, сам процес, використовуваний деякими фрактальними пакетами, включаючи Fractal Transform Майкла Барнслі, вважається патентованим. Переважна більшість досліджень в області фрактальної компресії зараз направлена на зменшення часу архівації, необхідного для отримання якісного зображення.

Контрольні питання

1. Наведіть класифікацію типів графічних файлів. Чим обумовлено велике різноманіття графічних форматів?
2. В чому полягає необхідність застосування алгоритмів стиснення?
3. Наведіть класифікацію способів та алгоритмів стиснення.
4. Дайте характеристику алгоритмів стиснення, що застосовуються у форматі JPEG.
5. Складіть схему алгоритму декомпресії, що реалізує метод LZW.
6. Перерахуйте основні етапи перетворень у форматі JPEG. Які основні переваги та недоліки цього формату?
7. Що таке фрактал? Наведіть приклади використання фракталів.
8. В чому полягає сутність алгоритмів фрактального стиснення? Класи яких графічних зображень недоцільно стискувати за допомогою цих алгоритмів?

Тестові завдання

Питання 1. Поле графічного файла -це:	А)	структура даних у графічному файлі, що має фіксований розмір
	Б)	структура даних, розмір і позиція якої змінюються від файла до файла
	В)	обидві відповіді вірні
Питання 2. Растрові формати відрізняються один від одного	А)	колірними моделями, методами ущільнення, максимальним розміром зображення, який вони можуть забезпечити

на їх якість. Для збереження високої якості зображень з низьким дозволом підсумковий ступінь стиску повинен бути набагато нижче. Відзначимо, що зображення з більшою бітовою глибиною (наприклад, 24-бітові truecolor-зображення) стискаються більш ефективно, ніж зображення з меншою кількістю бітів на піксел (зокрема, 8-бітові напівтонові). Процес фрактального стиснення не призначений для громадського користування. Є багато патентів на методи стиснення даних на базі фрактальних перетворень. Крім того, сам процес, використовуваний деякими фрактальними пакетами, включаючи Fractal Transform Майкла Барнслі, вважається патентованим. Переважна більшість досліджень в області фрактальної компресії зараз направлена на зменшення часу архівації, необхідного для отримання якісного зображення.

Контрольні питання

1. Наведіть класифікацію типів графічних файлів. Чим обумовлено велике різноманіття графічних форматів?
2. В чому полягає необхідність застосування алгоритмів стиснення?
3. Наведіть класифікацію способів та алгоритмів стиснення.
4. Дайте характеристику алгоритмів стиснення, що застосовуються у форматі JPEG.
5. Складіть схему алгоритму декомпресії, що реалізує метод LZW.
6. Перерахуйте основні етапи перетворень у форматі JPEG. Які основні переваги та недоліки цього формату?
7. Що таке фрактал? Наведіть приклади використання фракталів.
8. В чому полягає сутність алгоритмів фрактального стиснення? Класи яких графічних зображень недоцільно стискувати за допомогою цих алгоритмів?

Тестові завдання

Питання 1. Поле графічного файла -це:	А)	структура даних у графічному файлі, що має фіксований розмір
	Б)	структура даних, розмір і позиція якої змінюються від файла до файла
	В)	обидві відповіді вірні
Питання 2. Растрові формати відрізняються один від одного	А)	колірними моделями, методами ущільнення, максимальним розміром зображення, який вони можуть забезпечити

Б) шарами різних типів, наявністю Alpha-каналу, можливістю здійснювати анімацію, наявністю черезрядкового розгорнення
В) обидві відповіді вірні

Питання 3. Wavelet- перетворення застосовуються у форматі:
А) GIF
Б) PSD
В) JPEG2000

Питання 4. В якому з наведених форматів застосовуються різні методи (алгоритми) для кодування переднього та заднього плану зображення:
А) LWF
Б) JPEG2000
В) DjVu
Г) DXF

Питання 5. Метафайли знайшли використання для:
А) підвищення роздільної здатності растрових зображень
Б) обміну різними (як растровими, так і векторними) графічними даними
В) підвищення роздільної здатності векторних зображень

Питання 6. До метафайлів відносять такі графічні формати:
А) GIF, PSD, JPEG2000, TIFF, DjVu
Б) TIFF, DjVu, , EPS, PICT, WMF, VRML, SMD
В) CGM, EPS, PICT, WMF/EMF

Питання 7. Алгоритм Lempel-Ziv-Welch (LZW) застосовується у форматах:
А) GIF, TIFF
Б) BMP, TIFF і PCX
В) CGM, EPS, PICT

Питання 8. Дискретне косинус-синусне перетворення застосовується у форматі:
А) JPEG
Б) TIFF
В) JPEG2000

Питання 9. Фрактальне стиснення – це:
А) симетричний процес
Б) асиметричний процес
В) обидві відповіді вірні

Питання 10. Одним із основних недоліків фрактального стиснення є:
А) великий час архівації
Б) великі обсяги зайнятої пам’яті
В) низький ступінь стиснення

Б) шарами різних типів, наявністю Alpha-каналу, можливістю здійснювати анімацію, наявністю черезрядкового розгорнення
В) обидві відповіді вірні

Питання 3. Wavelet- перетворення застосовуються у форматі:
А) GIF
Б) PSD
В) JPEG2000

Питання 4. В якому з наведених форматів застосовуються різні методи (алгоритми) для кодування переднього та заднього плану зображення:
А) LWF
Б) JPEG2000
В) DjVu
Г) DXF

Питання 5. Метафайли знайшли використання для:
А) підвищення роздільної здатності растрових зображень
Б) обміну різними (як растровими, так і векторними) графічними даними
В) підвищення роздільної здатності векторних зображень

Питання 6. До метафайлів відносять такі графічні формати:
А) GIF, PSD, JPEG2000, TIFF, DjVu
Б) TIFF, DjVu, , EPS, PICT, WMF, VRML, SMD
В) CGM, EPS, PICT, WMF/EMF

Питання 7. Алгоритм Lempel-Ziv-Welch (LZW) застосовується у форматах:
А) GIF, TIFF
Б) BMP, TIFF і PCX
В) CGM, EPS, PICT

Питання 8. Дискретне косинус-синусне перетворення застосовується у форматі:
А) JPEG
Б) TIFF
В) JPEG2000

Питання 9. Фрактальне стиснення – це:
А) симетричний процес
Б) асиметричний процес
В) обидві відповіді вірні

Питання 10. Одним із основних недоліків фрактального стиснення є:
А) великий час архівації
Б) великі обсяги зайнятої пам’яті
В) низький ступінь стиснення

Розділ 2
ІНТЕРАКТИВНА КОМП'ЮТЕРНА ГРАФІКА

- Розробка графічних програм для Windows
- Графічна бібліотека OpenGL
- Характеристика основних можливостей пакету растрової графіки
- Характеристика основних можливостей пакету векторної графіки

Розділ 2
ІНТЕРАКТИВНА КОМП'ЮТЕРНА ГРАФІКА

- Розробка графічних програм для Windows
- Графічна бібліотека OpenGL
- Характеристика основних можливостей пакету растрової графіки
- Характеристика основних можливостей пакету векторної графіки

Глава 5
Розробка графічних програм для Windows

5.1. Використання API функцій для побудови графічних зображень

Borland C++ Builder як середовище візуального програмування має певний набір компонентів і класів, що дозволяє не лише малювати графічні примітиви, але і працювати із зображеннями, шрифтами і графічними файлами. Принципи роботи з графічними зображеннями в Borland C++ Builder засновані на загальних принципах створення графічних об'єктів у операційній системі Windows. Саме операційна система Windows надає програмістові засоби ***інтерфейсу графічних пристроїв*** (Graphics Device Interface, або скорочено GDI) [3, 16].

З GDI тісно пов'язано поняття "Device Context" (DC) - ***контексту графічного пристрою*** це те, на чому виконується зображення. DC - універсальний пристрій виведення, для якого можна використовувати однакові функції GDI для екрана, принтера та інших пристроїв виведення. Це можливо тому, що і сама операційна система Windows є не залежною від пристроїв системою (device independent).

З боку Windows це забезпечується бібліотекою GDI32.dll (C:\Windows\system), а з боку пристрою - драйвером цього пристрою. Сполучною ланкою між програмою та пристроєм і є контекст пристрою. Програма для виведення зображення повинна отримати Handle контексту пристрою, тобто Handle структури, що містить набір характеристик цього пристрою. У цей набір входять бітові карти для зображень, пера, кисті, шрифти і так далі.

Глава 5
Розробка графічних програм для Windows

5.1. Використання API функцій для побудови графічних зображень

Borland C++ Builder як середовище візуального програмування має певний набір компонентів і класів, що дозволяє не лише малювати графічні примітиви, але і працювати із зображеннями, шрифтами і графічними файлами. Принципи роботи з графічними зображеннями в Borland C++ Builder засновані на загальних принципах створення графічних об'єктів у операційній системі Windows. Саме операційна система Windows надає програмістові засоби ***інтерфейсу графічних пристроїв*** (Graphics Device Interface, або скорочено GDI) [3, 16].

З GDI тісно пов'язано поняття "Device Context" (DC) - ***контексту графічного пристрою*** це те, на чому виконується зображення. DC - універсальний пристрій виведення, для якого можна використовувати однакові функції GDI для екрана, принтера та інших пристроїв виведення. Це можливо тому, що і сама операційна система Windows є не залежною від пристроїв системою (device independent).

З боку Windows це забезпечується бібліотекою GDI32.dll (C:\Windows\system), а з боку пристрою - драйвером цього пристрою. Сполучною ланкою між програмою та пристроєм і є контекст пристрою. Програма для виведення зображення повинна отримати Handle контексту пристрою, тобто Handle структури, що містить набір характеристик цього пристрою. У цей набір входять бітові карти для зображень, пера, кисті, шрифти і так далі.

Програма звертається до контексту пристрою через функції GDI. У Windows підтримуються такі типи контекстів пристроїв:

- контекст дисплея;
- контекст принтера;
- контекст в пам'яті (моделює в пам'яті пристрій виводу);
- інформаційний контекст (служить для отримання даних від пристроїв).

Коли програмі потрібний контекст пристрою, вона отримує його вже заповненим значеннями за замовчанням – так званий поточний об'єкт. Контекст пристрою можна змінити. Програма може створити новий об'єкт і зробити його поточним. Заміщений об'єкт з пам'яті не видаляється, і його можна відновити. Змінити значення характеристики можна тільки через заміну об'єкта.

Використовуючи функції GDI, можна оперувати з графічними об'єктами і отримувати графічні зображення. У той же час робота з функціями GDI вимагала від програміста виконання безлічі допоміжних та рутинних дій і великої уваги, що привело спочатку (у старому добром Borland C++ без добавки Builder) до появи OWL (Object Windows Library). У OWL функції GDI інкапсульовані і переобтяжені в класах (багато функцій інкапсульовано як вбудовані – inline функції, які безпосередньо переводяться в еквівалентні оператори WinApi.

Засоби BorlandC++Builder використовують усі можливості GDI і OWL, але на більш високому рівні, і тим самим ще більше спрощують працю програміста. У той же час пряме звернення додатків до окремих функцій Windows GDI також не виключається.

Узагальнюючи, можна сказати –Borland C++ Builder має можливість і прямого виклику функції GDI з використанням контексту пристрою і може використовувати власний інтерфейс для роботи з графікою через властивості своїх компонент.

Графічні об'єкти - це об'єкти, за допомогою яких здійснюється вивід на екран зображень. Їх використання в додатках для створення графічних зображень можливо на різних рівнях.

Графічних об'єктів не так вже і багато:

- *Точка*. Зображення можна малювати точками. Найбільш простий метод, оскільки для нього досить виклику лиш однієї

Програма звертається до контексту пристрою через функції GDI. У Windows підтримуються такі типи контекстів пристроїв:

- контекст дисплея;
- контекст принтера;
- контекст в пам'яті (моделює в пам'яті пристрій виводу);
- інформаційний контекст (служить для отримання даних від пристроїв).

Коли програмі потрібний контекст пристрою, вона отримує його вже заповненим значеннями за замовчанням – так званий поточний об'єкт. Контекст пристрою можна змінити. Програма може створити новий об'єкт і зробити його поточним. Заміщений об'єкт з пам'яті не видаляється, і його можна відновити. Змінити значення характеристики можна тільки через заміну об'єкта.

Використовуючи функції GDI, можна оперувати з графічними об'єктами і отримувати графічні зображення. У той же час робота з функціями GDI вимагала від програміста виконання безлічі допоміжних та рутинних дій і великої уваги, що привело спочатку (у старому добром Borland C++ без добавки Builder) до появи OWL (Object Windows Library). У OWL функції GDI інкапсульовані і переобтяжені в класах (багато функцій інкапсульовано як вбудовані – inline функції, які безпосередньо переводяться в еквівалентні оператори WinApi.

Засоби BorlandC++Builder використовують усі можливості GDI і OWL, але на більш високому рівні, і тим самим ще більше спрощують працю програміста. У той же час пряме звернення додатків до окремих функцій Windows GDI також не виключається.

Узагальнюючи, можна сказати –Borland C++ Builder має можливість і прямого виклику функції GDI з використанням контексту пристрою і може використовувати власний інтерфейс для роботи з графікою через властивості своїх компонент.

Графічні об'єкти - це об'єкти, за допомогою яких здійснюється вивід на екран зображень. Їх використання в додатках для створення графічних зображень можливо на різних рівнях.

Графічних об'єктів не так вже і багато:

- *Точка*. Зображення можна малювати точками. Найбільш простий метод, оскільки для нього досить виклику лиш однієї

функції SetPixel() (так як і для читання GetPixel()). Взагалі, точка не є графічним об'єктом, оскільки вона не існує як об'єкт і розглядається тут лише як одна з можливостей отримання зображення.

- *Перо*. Зображення можна малювати пером (як олівцем на аркуші паперу). Перо володіє товщиною, кольором і типом лінії (суцільна, переривчаста, точкова і т. п.). Перед використанням перо створюється як об'єкт.
- *Кисть*. Зумовлений системою або створений програмістом набір пікселів, який як єдине ціле може бути відображений на екрані монітора. Кисті використовуються у функціях для заповнення внутрішніх областей замкнутих фігур і фонів вікон (аналогічно суцільному зафарбовуванню). Перед використанням кисть створюється як об'єкт.
- *Растрові зображення*. Набір байтів, що містить значення кольорів і інформацію про координати для відображення на екрані пікселів, що в сукупності становлять зображення. Це картинки, фони, графічні елементи (кнопки, меню, іконки) і т. п. Перед використанням растрові зображення можуть бути створені програмно або використовуватися як заздалегідь створені і такі, що зберігаються у файлах, ресурсах і так далі.
- *Шрифти*. Це або набір пікселів (растрове зображення) для матричних шрифтів, або набір кривих, що описують контур букв, які відображаються, для векторних шрифтів (наприклад шрифти TrueType). Перед використанням шрифти можуть бути створені програмно або використовуються заздалегідь підготовлені і встановлені в системі шрифти.

Отримання зображення перемальовуванням пікселів

Для зміни контексту пристрою використовується функція SetPixel():

```
WINGDIAPI COLORREF WINAPI SetPixel(HDC hdc, int x, int y,
                                   COLORREF color).
```

Перший аргумент Handle – це контекст пристрою (в усіх наступних прикладах далі цей параметр не пояснюватиметься).

Аргументи x та y – координати піксела, що малюється;

функції SetPixel() (так як і для читання GetPixel()). Взагалі, точка не є графічним об'єктом, оскільки вона не існує як об'єкт і розглядається тут лише як одна з можливостей отримання зображення.

- *Перо*. Зображення можна малювати пером (як олівцем на аркуші паперу). Перо володіє товщиною, кольором і типом лінії (суцільна, переривчаста, точкова і т. п.). Перед використанням перо створюється як об'єкт.
- *Кисть*. Зумовлений системою або створений програмістом набір пікселів, який як єдине ціле може бути відображений на екрані монітора. Кисті використовуються у функціях для заповнення внутрішніх областей замкнутих фігур і фонів вікон (аналогічно суцільному зафарбовуванню). Перед використанням кисть створюється як об'єкт.
- *Растрові зображення*. Набір байтів, що містить значення кольорів і інформацію про координати для відображення на екрані пікселів, що в сукупності становлять зображення. Це картинки, фони, графічні елементи (кнопки, меню, іконки) і т. п. Перед використанням растрові зображення можуть бути створені програмно або використовуватися як заздалегідь створені і такі, що зберігаються у файлах, ресурсах і так далі.
- *Шрифти*. Це або набір пікселів (растрове зображення) для матричних шрифтів, або набір кривих, що описують контур букв, які відображаються, для векторних шрифтів (наприклад шрифти TrueType). Перед використанням шрифти можуть бути створені програмно або використовуються заздалегідь підготовлені і встановлені в системі шрифти.

Отримання зображення перемальовуванням пікселів

Для зміни контексту пристрою використовується функція SetPixel():

```
WINGDIAPI COLORREF WINAPI SetPixel(HDC hdc, int x, int y,
                                   COLORREF color).
```

Перший аргумент Handle – це контекст пристрою (в усіх наступних прикладах далі цей параметр не пояснюватиметься).

Аргументи x та y – координати піксела, що малюється;

color – колір піксела як сукупність трьох кольорів. Кожна з цих мікроточок може мати значення, що відповідає інтенсивності світіння від 0 до 255 (максимальна яскравість). Таким чином можуть бути визначені майже 17 мільйонів кольорів. Можна задати колір як 16-ричне число:00xx00bbggrr

Макрос RGB, повертаючий колір піксела, визначений як:

```
##define RGB(r, g, b) ((COLORREF) (((BYTE)(r)
   | ((WORD) ((BYTE)(g)) <8 ))
              | (((DWORD) (BYTE) (P)) <16 )))
```

Приклад використання функції SetPixel(). У прикладі точками малюється зелена лінія завдовжки 100 пікселів.

```
HDC  hDc = GetDC(Handle);
for(int i=0; i < 100;i++)
SetPixel(hDc, 100,100, RGB(0,255,0)) :
ReleaseDC(Handle, hDc);
```

Перо перед використанням створюється за допомогою функції CreatePen. Ця функція створює логічне перо, яке задає вказаний стиль, ширину і колір пера.

```
HPEN CreatePen
(
intPenStyle,        // стиль пера
intnWidth,          // ширина
 COLORREF  crColor     // колір );
```

Стиль пера задають константи:

PS_SOLID– суцільна лінія.
PS_DASH– штрихова лінія.
PS_DOT– пунктирна лінія.
PS_DASHDOT– штрих пунктирна лінія.
PS_DASHDOTDOT– риски, що чергуються, і подвійні точки.
PS_NULL– невидиме перо.
PS_INSIDEFRAME– перо для ліній, що виводяться усередині рамки закритих форм ((наприклад, Ellipse, Rectangle, RoundRect, Pie, і Chord функції).

color – колір піксела як сукупність трьох кольорів. Кожна з цих мікроточок може мати значення, що відповідає інтенсивності світіння від 0 до 255 (максимальна яскравість). Таким чином можуть бути визначені майже 17 мільйонів кольорів. Можна задати колір як 16-ричне число:00xx00bbggrr

Макрос RGB, повертаючий колір піксела, визначений як:

##define RGB(r, g, b) ((COLORREF) (((BYTE)(r)
| ((WORD) ((BYTE)(g)) <8))
| (((DWORD) (BYTE) (P)) <16)))

Приклад використання функції SetPixel(). У прикладі точками малюється зелена лінія завдовжки 100 пікселів.

HDC hDc = GetDC(Handle);
for(int i=0; i < 100;i++)
SetPixel(hDc, 100,100, RGB(0,255,0)) :
ReleaseDC(Handle, hDc);

Перо перед використанням створюється за допомогою функції CreatePen. Ця функція створює логічне перо, яке задає вказаний стиль, ширину і колір пера.

HPEN CreatePen
(
intPenStyle, // стиль пера
intnWidth, // ширина
COLORREF crColor // колір);

Стиль пера задають константи:
PS_SOLID– суцільна лінія.
PS_DASH– штрихова лінія.
PS_DOT– пунктирна лінія.
PS_DASHDOT– штрих пунктирна лінія.
PS_DASHDOTDOT– риски, що чергуються, і подвійні точки.
PS_NULL– невидиме перо.
PS_INSIDEFRAME– перо для ліній, що виводяться усередині рамки закритих форм ((наприклад, Ellipse, Rectangle, RoundRect, Pie, і Chord функції).

Перо може бути створене також функцією CreatePenIndirect().

```
HPEN CreatePenIndirect
(
  CONST LOGPEN *lplgpn  // покажчик на структуру LOGPEN
);
```

Функція SelectObject вибирає об’єкт (в даному випадку перо) у вказаний контекст пристрою. Новий об'єкт замінює попередній об'єкт того ж самого типу і повертає HDC замінюваного об'єкта.

```
HGDIOBJ SelectObject
(
  HDC hdc,        // дескриптор контексту пристрою
  HGDIOBJ hgdiobj  // дескриптор об'єкту який вибирается
);
```

Для малювання пером використовуються такі функції.

MoveTo(HDC hdc, int x1, int y1, LPPOINT lpPoint). Переміщає точку початку малювання лінії у вказані координати. LPPOINT lpPoint - адреса старої поточної позиції.

LineTo(int x2, int y2). Малює лінію починаючи з поточної позиції, заданої функцією MoveTo, до вказаних координат.

Rectangle(HDC hdc, int x1, int y1, int x2, int y2). Малює прямокутник, розмір якого визначається координатами верхнього (x1, y1) і нижнього (x2, y2) кута. Використовується поточне перо, а для заповнення поточна кисть.

Ellipse(HDC hdc, int x1, int y1, int x2, int y2). Малює еліпс, вписаний у прямокутник, розмір якого визначається координатами верхнього (x1, y1) і нижнього (x2, y2) кута. Еліпс заповнений білим кольором і обведений лінією пера контексту пристрою. Використовується поточне перо, а для заповнення поточна кисть.

RoundRect(HDC hdc, int x1, int y1, int x2, int y2, int x3, int y3). Малює прямокутник із закругленими краями, розмір якого визначається координатами верхнього (x1, y1), нижнього (x2, y2) кута і координатами скруглення (x3, y3). Використовується поточне перо, а для заповнення поточна кисть.

Arc(HDC hdc, int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4). Малює еліптичну дугу, логічно обмежену прямокутником, розмір якого визначається координатами верхнього (x1, y1) і нижнього (x2, y2) кута. Безпосередньо дуга визначається додатковими

Перо може бути створене також функцією CreatePenIndirect().

```
HPEN CreatePenIndirect
(
  CONST LOGPEN *lplgpn  // покажчик на структуру LOGPEN
);
```

Функція SelectObject вибирає об’єкт (в даному випадку перо) у вказаний контекст пристрою. Новий об'єкт замінює попередній об'єкт того ж самого типу і повертає HDC замінюваного об'єкта.

```
HGDIOBJ SelectObject
(
  HDC hdc,        // дескриптор контексту пристрою
  HGDIOBJ hgdiobj  // дескриптор об'єкту який вибирается
);
```

Для малювання пером використовуються такі функції.

MoveTo(HDC hdc, int x1, int y1, LPPOINT lpPoint). Переміщає точку початку малювання лінії у вказані координати. LPPOINT lpPoint - адреса старої поточної позиції.

LineTo(int x2, int y2). Малює лінію починаючи з поточної позиції, заданої функцією MoveTo, до вказаних координат.

Rectangle(HDC hdc, int x1, int y1, int x2, int y2). Малює прямокутник, розмір якого визначається координатами верхнього (x1, y1) і нижнього (x2, y2) кута. Використовується поточне перо, а для заповнення поточна кисть.

Ellipse(HDC hdc, int x1, int y1, int x2, int y2). Малює еліпс, вписаний у прямокутник, розмір якого визначається координатами верхнього (x1, y1) і нижнього (x2, y2) кута. Еліпс заповнений білим кольором і обведений лінією пера контексту пристрою. Використовується поточне перо, а для заповнення поточна кисть.

RoundRect(HDC hdc, int x1, int y1, int x2, int y2, int x3, int y3). Малює прямокутник із закругленими краями, розмір якого визначається координатами верхнього (x1, y1), нижнього (x2, y2) кута і координатами скруглення (x3, y3). Використовується поточне перо, а для заповнення поточна кисть.

Arc(HDC hdc, int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4). Малює еліптичну дугу, логічно обмежену прямокутником, розмір якого визначається координатами верхнього (x1, y1) і нижнього (x2, y2) кута. Безпосередньо дуга визначається додатковими

двома точками (x3, y3, x4, y4). Перша – початок дуги знаходиться на перетині еліпса, частиною якого є дуга, і прямої, що проходить через центр прямокутника і точку початку дуги. Друга – кінець дуги визначається аналогічно.

ArcTo(HDC hdc, int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4). Повністю аналогічна функції ArcTo, за винятком того, що запам'ятовується як поточна позиція пера, остання точка дуги.

Pie(HDC hdc, int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4). Малює сектор еліпса, що вписується в прямокутник, розмір якого визначається координатами верхнього (x1, y1) і нижнього (x2, y2) кута, а координати початкової і кінцевої точок (x3, y3, x4, y4) аналогічно функції Arc(). Використовується поточне перо, а для заповнення поточна кисть.

Chord(HDC hdc, int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4). Малює сегмент еліпса (область, обмежену перетином еліпса і лінії).

Polyline(HDC hdc, CONST POINT *lppt, intcPoints). Функція малює ламану лінію по масиву точок, на який вказує lppt і число точок з цього масиву рівне cPoints. Відрізки прямих малюються поточним пером. Фігури, утворені сегментами, не зафарбовуються.

Polygon(HDC hdc, CONST POINT *lppt, intcPoints). Функція малює багатокутник, що складається з двох або більше вершин, пов'язаних прямими лініями по масиву точок, на який вказує lppt і число точок з цього масиву рівне cPoints. Використовується поточне перо і заповнення поточною кистю.

PolyPolyline(HDC hdc, CONST POINT *lppt, CONST DWORD *lpdwPolyPoints, DWORD cCount); lppt – покажчик на масив структур типу POINT. Кожна структура в масиві ідентифікує точку в логічному просторі. lpdwPolyPoints – вказує на масив змінних, що визначають число точок у масиві lppt. Значення кожного елемента має бути більше або дорівнювати двом. Параметр cCount– визначає кількість елементів у масиві lpdwPolyPo. Відрізки прямих малюються поточним пером. Фігури, утворені сегментами, не зафарбовуються.

PolyBezier(HDC hdc, CONST POINT *lppt, DWORD cPoints). Виводить одну або більше Bezier-сплайнів (кривих Без'є). Ці криві

двома точками (x3, y3, x4, y4). Перша – початок дуги знаходиться на перетині еліпса, частиною якого є дуга, і прямої, що проходить через центр прямокутника і точку початку дуги. Друга – кінець дуги визначається аналогічно.

ArcTo(HDC hdc, int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4). Повністю аналогічна функції ArcTo, за винятком того, що запам'ятовується як поточна позиція пера, остання точка дуги.

Pie(HDC hdc, int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4). Малює сектор еліпса, що вписується в прямокутник, розмір якого визначається координатами верхнього (x1, y1) і нижнього (x2, y2) кута, а координати початкової і кінцевої точок (x3, y3, x4, y4) аналогічно функції Arc(). Використовується поточне перо, а для заповнення поточна кисть.

Chord(HDC hdc, int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4). Малює сегмент еліпса (область, обмежену перетином еліпса і лінії).

Polyline(HDC hdc, CONST POINT *lppt, intcPoints). Функція малює ламану лінію по масиву точок, на який вказує lppt і число точок з цього масиву рівне cPoints. Відрізки прямих малюються поточним пером. Фігури, утворені сегментами, не зафарбовуються.

Polygon(HDC hdc, CONST POINT *lppt, intcPoints). Функція малює багатокутник, що складається з двох або більше вершин, пов'язаних прямими лініями по масиву точок, на який вказує lppt і число точок з цього масиву рівне cPoints. Використовується поточне перо і заповнення поточною кистю.

PolyPolyline(HDC hdc, CONST POINT *lppt, CONST DWORD *lpdwPolyPoints, DWORD cCount); lppt – покажчик на масив структур типу POINT. Кожна структура в масиві ідентифікує точку в логічному просторі. lpdwPolyPoints – вказує на масив змінних, що визначають число точок у масиві lppt. Значення кожного елемента має бути більше або дорівнювати двом. Параметр cCount– визначає кількість елементів у масиві lpdwPolyPo. Відрізки прямих малюються поточним пером. Фігури, утворені сегментами, не зафарбовуються.

PolyBezier(HDC hdc, CONST POINT *lppt, DWORD cPoints). Виводить одну або більше Bezier-сплайнів (кривих Без'є). Ці криві

задаються початком, кінцем лінії і проміжними точками, що визначають вигин. У масиві точок перша і четверта використовується як кінцеві, друга і третя як проміжні.

PolyDraw(const POINT* lpPoints, const BYTE* lpTypes, intnCount). Функція складає безліч сегментів рядка і Кривих Без'є. lpPoints - покажчик на масив структур даних POINT, який містить крайові точки для кожного сегмента лінії і контрольних точок для кожного Bezier-сплайна. Аргумент lpTypes - покажчик на масив, який визначає, як кожна точка в lpPoints масиві використовується.

Наступний приклад показує використання деяких описаних функцій (рис. 5.1).

```
void__fastcall
TForm1::Button1Click(TObject *Sender)
{
  HPEN       hPen, hPenOld;
 // //Отримуємо контекст вікна додатка
  HDC        hDc = GetDC(Handle);
 // //Створюємо перо суцільне, завтовшки 2, червоне
hPen=CreatePen(PS_SOLID, 2, RGB(255,0,0));
 // //Вибираємо перо в контекст пристрою і запам'ятовуємо старе
hPenOld = SelectObject(hDc, hPen);

 // //Малюємо лінію
MoveToEx(hDc, 0,0, NULL);
LineTo(hDc, Width, Height);
 // //Малюємо  прямокутник
Rectangle(hDc, 50,50,100,100);
 // //Малюємо  прямокутник з кутами, що округляють
RoundRect(hDc, 150,150,400,400,50,50);
 // //Малюємо эліпс
Ellipse(hDc, 100,100,300,300);
 // //Малюємо  сектор эліпса
Pie(hDc, 0,0,200,200,0,0,0,100);
 // //Малюємо дугу
Arc(hDc, 0,0,250,250,125,0,50,0);
 // //Малюємо  сегмент, замкнутий дугою
AngleArc(hDc, 100,150,50,0,180);
 // //Малюємо  сегмент эліпса
Chord(hDc, 0,0,200,200,0,0,10,200);
```


задаються початком, кінцем лінії і проміжними точками, що визначають вигин. У масиві точок перша і четверта використовується як кінцеві, друга і третя як проміжні.

PolyDraw(const POINT* lpPoints, const BYTE* lpTypes, intnCount). Функція складає безліч сегментів рядка і Кривих Без'є. lpPoints - покажчик на масив структур даних POINT, який містить крайові точки для кожного сегмента лінії і контрольних точок для кожного Bezier-сплайна. Аргумент lpTypes - покажчик на масив, який визначає, як кожна точка в lpPoints масиві використовується.

Наступний приклад показує використання деяких описаних функцій (рис. 5.1).

```
void__fastcall
TForm1::Button1Click(TObject *Sender)
{
  HPEN       hPen, hPenOld;
 // //Отримуємо контекст вікна додатка
  HDC        hDc = GetDC(Handle);
 // //Створюємо перо суцільне, завтовшки 2, червоне
hPen=CreatePen(PS_SOLID, 2, RGB(255,0,0));
 // //Вибираємо перо в контекст пристрою і запам'ятовуємо старе
hPenOld = SelectObject(hDc, hPen);

 // //Малюємо лінію
MoveToEx(hDc, 0,0, NULL);
LineTo(hDc, Width, Height);
 // //Малюємо  прямокутник
Rectangle(hDc, 50,50,100,100);
 // //Малюємо  прямокутник з кутами, що округляють
RoundRect(hDc, 150,150,400,400,50,50);
 // //Малюємо эліпс
Ellipse(hDc, 100,100,300,300);
 // //Малюємо  сектор эліпса
Pie(hDc, 0,0,200,200,0,0,0,100);
 // //Малюємо дугу
Arc(hDc, 0,0,250,250,125,0,50,0);
 // //Малюємо  сегмент, замкнутий дугою
AngleArc(hDc, 100,150,50,0,180);
 // //Малюємо  сегмент эліпса
Chord(hDc, 0,0,200,200,0,0,10,200);
```

```
 // //Малюємо ламану лінію
TPointtPoint[3];
tPoint[0] = Point(100,500);
tPoint[1] = Point(150,400);
tPoint[2] = Point(100,300);
Polyline(hDc, tPoint, 3);

 // //Малюємо  криві Без'є
TPointtPoints[7];
tPoints[0]=TPoint(0,0);
tPoints[1]=TPoint(800,30);
tPoints[2]=TPoint(0,40);
tPoints[3]=TPoint(550,400);
tPoints[4]=TPoint(350,200);
tPoints[5]=TPoint(550,400);
tPoints[6]=TPoint(0,500);
PolyBezier(hDc, tPoints, 6);

 // //Повертаємо усе на свої місця
SelectObject(hDc, hPenOld);
DeleteObject(hPen);
DeleteObject(hPenOld);
ReleaseDC(Handle, hDc);
DeleteDC(hDc);
}
```

Для того щоб малювати не у формі додатка, а на екрані дисплея, досить у попередньому прикладі отримати контекст екрана монітора.

```
HDC   hDc=CreateDC("DISPLAY", NULL, NULL, NULL).
```

Кисті, їх створення і застосування

Кисті використовуються в Windows в основному для заливки внутрішніх областей.

Щоб використовувати кисть, її необхідно створити за допомогою таких функцій, як CreateSolidBrush, CreateDIBPatternBrush, CreateHatchBrush, CreatePatternBrush і потім, як і для пера, вибрати їх у контексті відображення, використовуючи функцію SelectObject. Після застосування засобів малювання їх можна видалити за допомогою функції DeleteObject.


```
 // //Малюємо ламану лінію
TPointtPoint[3];
tPoint[0] = Point(100,500);
tPoint[1] = Point(150,400);
tPoint[2] = Point(100,300);
Polyline(hDc, tPoint, 3);

 // //Малюємо  криві Без'є
TPointtPoints[7];
tPoints[0]=TPoint(0,0);
tPoints[1]=TPoint(800,30);
tPoints[2]=TPoint(0,40);
tPoints[3]=TPoint(550,400);
tPoints[4]=TPoint(350,200);
tPoints[5]=TPoint(550,400);
tPoints[6]=TPoint(0,500);
PolyBezier(hDc, tPoints, 6);

 // //Повертаємо усе на свої місця
SelectObject(hDc, hPenOld);
DeleteObject(hPen);
DeleteObject(hPenOld);
ReleaseDC(Handle, hDc);
DeleteDC(hDc);
}
```

Для того щоб малювати не у формі додатка, а на екрані дисплея, досить у попередньому прикладі отримати контекст екрана монітора.

```
HDC   hDc=CreateDC("DISPLAY", NULL, NULL, NULL).
```

Кисті, їх створення і застосування

Кисті використовуються в Windows в основному для заливки внутрішніх областей.

Щоб використовувати кисть, її необхідно створити за допомогою таких функцій, як CreateSolidBrush, CreateDIBPatternBrush, CreateHatchBrush, CreatePatternBrush і потім, як і для пера, вибрати їх у контексті відображення, використовуючи функцію SelectObject. Після застосування засобів малювання їх можна видалити за допомогою функції DeleteObject.

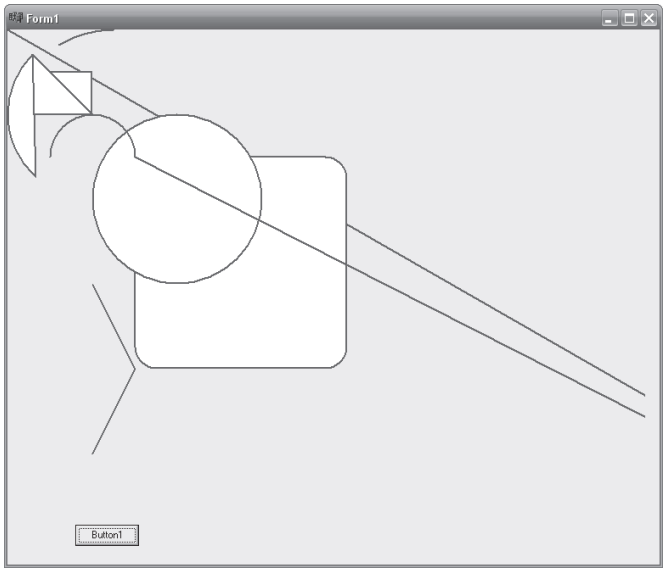


Рис. 5.1. Приклад використання функцій малювання

Основні функції для створення кисті такі:

Функція CreateSolidBrush створює логічну кисть, яка має вказаний колір.

```
HBRUSH CreateSolidBrush
(
  COLORREF crColor // колір кисті
).
```

Функція CreateHatchBrush створює логічну кисть, яка має вказаний зразок штрихування і колір.

```
HBRUSH CreateHatchBrush
(
IntfnStyle,    // стиль штрихування
  COLORREF clrref // колір
);
```

FnStyleвизначає стиль штрихування кисті.

Функція CreatePatternBrushстворює логічну кисть з вказаним растровим зображенням.

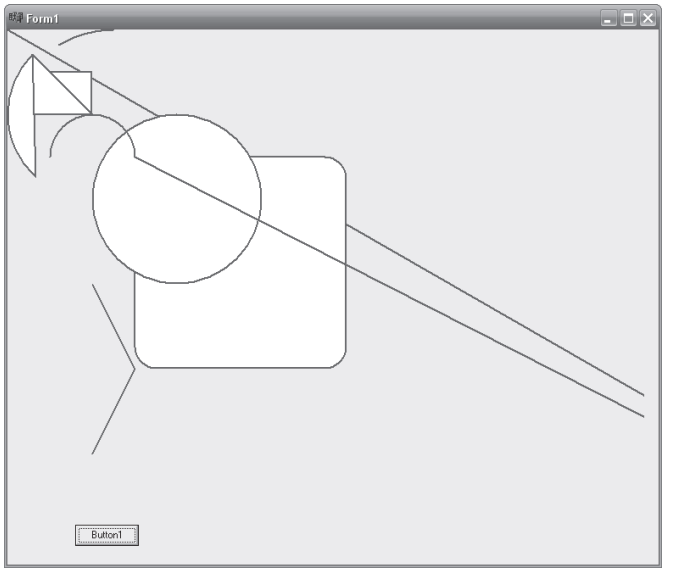


Рис. 5.1. Приклад використання функцій малювання

Основні функції для створення кисті такі:

Функція CreateSolidBrush створює логічну кисть, яка має вказаний колір.

```
HBRUSH CreateSolidBrush
(
  COLORREF crColor // колір кисті
).
```

Функція CreateHatchBrush створює логічну кисть, яка має вказаний зразок штрихування і колір.

```
HBRUSH CreateHatchBrush
(
IntfnStyle,    // стиль штрихування
  COLORREF clrref // колір
);
```

FnStyleвизначає стиль штрихування кисті.

Функція CreatePatternBrushстворює логічну кисть з вказаним растровим зображенням.

```
HBRUSH CreatePatternBrush
(
  HBITMAP hbmp // Handle зображення
).
```

Приклад 5.1. Робота з кистями.

Використання кисті із створеним растровим зображенням(рис. 5.2):

```
void__fastcall TForm1::Button1Click(TObject *Sender)
{
HDC    hDc;
 HBRUSH hBrush,hBrushOld;
unsignedshortusrgMassPix[225];
if(GetDeviceCaps(CreateDC("DISPLAY",NULL,NULL,NULL),BITS
PIXEL) == 24)
 {
for(int i=0; i < 75; i++)
  {
usrgMassPix[i*3]=255;
usrgMassPix[i*3+1]=0;
usrgMassPix[i*3+2]=0;
  }
 }
else
for(int i=0; i < 225; i++) usrgMassPix[i] =random(255);
 HBITMAP
hBmp =
CreateBitmap(10,10,GetDeviceCaps(CreateDC("DISPLAY",
                              NULL,NULL,NULL),PLANES),
GetDeviceCaps(CreateDC("DISPLAY",NULL,NULL,NULL),BITSP
IXEL),&usrgMassPix[0]);
hDc=GetDC(Handle);
hBrush = CreatePatternBrush(hBmp);
hBrushOld=SelectObject(hDc,hBrush);
Ellipse(hDc,200,200,600,400);
SelectObject(hDc,hBrushOld);
DeleteObject(hBmp);
DeleteObject(hBrush);
DeleteObject(hBrushOld);
ReleaseDC(Handle,hDc);
DeleteDC(hDc);
}
```



```
HBRUSH CreatePatternBrush
(
  HBITMAP hbmp // Handle зображення
).
```

Приклад 5.1. Робота з кистями.

Використання кисті із створеним растровим зображенням(рис. 5.2):

```
void__fastcall TForm1::Button1Click(TObject *Sender)
{
HDC    hDc;
 HBRUSH hBrush,hBrushOld;
unsignedshortusrgMassPix[225];
if(GetDeviceCaps(CreateDC("DISPLAY",NULL,NULL,NULL),BITS
PIXEL) == 24)
 {
for(int i=0; i < 75; i++)
  {
usrgMassPix[i*3]=255;
usrgMassPix[i*3+1]=0;
usrgMassPix[i*3+2]=0;
  }
 }
else
for(int i=0; i < 225; i++) usrgMassPix[i] =random(255);
 HBITMAP
hBmp =
CreateBitmap(10,10,GetDeviceCaps(CreateDC("DISPLAY",
                              NULL,NULL,NULL),PLANES),
GetDeviceCaps(CreateDC("DISPLAY",NULL,NULL,NULL),BITSP
IXEL),&usrgMassPix[0]);
hDc=GetDC(Handle);
hBrush = CreatePatternBrush(hBmp);
hBrushOld=SelectObject(hDc,hBrush);
Ellipse(hDc,200,200,600,400);
SelectObject(hDc,hBrushOld);
DeleteObject(hBmp);
DeleteObject(hBrush);
DeleteObject(hBrushOld);
ReleaseDC(Handle,hDc);
DeleteDC(hDc);
}
```

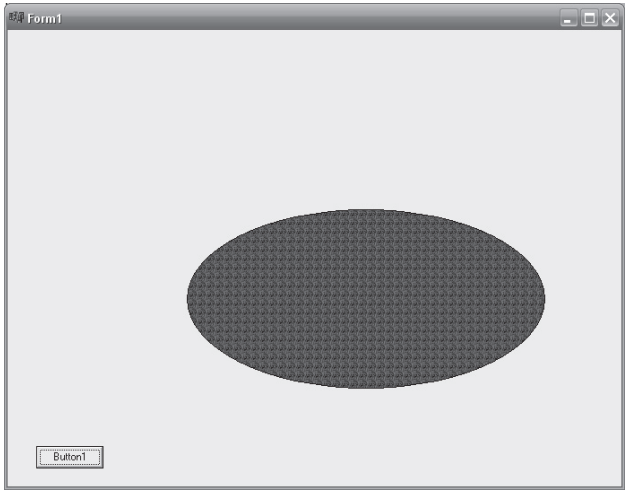


Рис. 5.2. Приклад роботи з кистями

У цьому прикладі використано не лише перо і кисть, але і об'єкт DC - растрове зображення.

5.2. Робота з графікою з використанням класів, властивостей і функцій компонент Borland C++ Builder

У середовищі C++Builder існує три роди об'єктів, які мають відношення до графіки.

Канва – надає бітову карту поверхні вікна додатка, компоненти, принтера і тому подібне, яка може бути використана для виведення графіки. Канва не самостійний об'єкт, вона завжди є властивістю якогось іншого графічного об'єкта.

Графіка – подає растрове зображення деякого файла або ресурсу (бітового образу, піктограми або метафайла).

C++Builder визначає похідні від базового класу TGraphic об'єктні класи:

– TBitmap,
– Ticon,
– TMetafile.

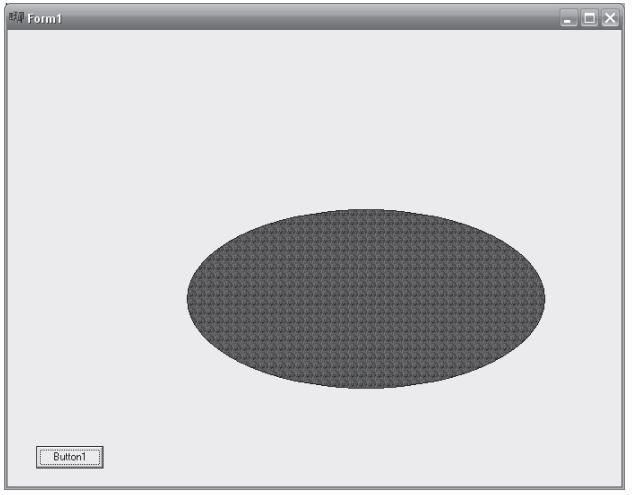


Рис. 5.2. Приклад роботи з кистями

У цьому прикладі використано не лише перо і кисть, але і об'єкт DC - растрове зображення.

5.2. Робота з графікою з використанням класів, властивостей і функцій компонент Borland C++ Builder

У середовищі C++Builder існує три роди об'єктів, які мають відношення до графіки.

Канва – надає бітову карту поверхні вікна додатка, компоненти, принтера і тому подібне, яка може бути використана для виведення графіки. Канва не самостійний об'єкт, вона завжди є властивістю якогось іншого графічного об'єкта.

Графіка – подає растрове зображення деякого файла або ресурсу (бітового образу, піктограми або метафайла).

C++Builder визначає похідні від базового класу TGraphic об'єктні класи:

– TBitmap,
– Ticon,
– TMetafile.

Малюнок (TPicture) – є контейнером для графіки, який може містити будь-які класи графічних об'єктів. Таким чином, контейнерний клас TPicture може містити бітовий образ, піктограму, метафайл або деякий інший графічний тип, визначений користувачем, а додаток може стандартно звертатися до всіх об'єктів контейнера за допомогою об'єкта TPicture.

Відмітимо, що графічні об'єкти Windows взаємозв'язані. Так, – об'єкт TPicture завжди містить деяку графіку, якій, у свою чергу, може знадобитися для відображення канва, а єдиний стандартний графічний клас канви – це TBitmap.

Як відзначалося вище, BorlandC++ Builder інкапсулює функції Windows GDI на різних рівнях. Найбільш завершеним є інтерфейс, що надається властивістю Canvas (канва), об'єктного класу канви, його графічних компонент.

Використання канви знімає з програміста турботу про виведення зображень, про ініціалізацію контексту пристрою і його звільнення. Наявності вкладених властивостей (характеристик пера, кисті, шрифтів, растрових зображень) також не вимагає стеження за станами ресурсів – основне завдання – це визначення характеристик для цих графічних об'єктів і грамотне їх використання.

Об'єктний клас канви

Інкапсульовані і переобтяжені функції GDI і WinApi об'єктного класу канви багато авторів відносять до трьох різних рівнів. У цій умовній класифікації функції високого рівня забезпечують можливість малювання ліній, фігур і тексту.

Визначення властивостей і методів маніпулювання графічними примітивами канви віднесені до середнього рівня. Нижній рівень забезпечує доступ до самих функцій Windows GDI. Класифікація не безперечна, але вона дозволяє орієнтуватися в досить великій кількості властивостей і методів канви.

Приклад 5.2.Використання функцій канви для малювання примітивів.

Основи малювання примітивів розглянуті вище при розгляді функцій GDI, тут, на прикладах, не повторюючи вже описані раніше прийоми виведення примітивів, показані лише деякі особливості їх відображення з використанням функцій канви.

Малюнок (TPicture) – є контейнером для графіки, який може містити будь-які класи графічних об'єктів. Таким чином, контейнерний клас TPicture може містити бітовий образ, піктограму, метафайл або деякий інший графічний тип, визначений користувачем, а додаток може стандартно звертатися до всіх об'єктів контейнера за допомогою об'єкта TPicture.

Відмітимо, що графічні об'єкти Windows взаємозв'язані. Так, – об'єкт TPicture завжди містить деяку графіку, якій, у свою чергу, може знадобитися для відображення канва, а єдиний стандартний графічний клас канви – це TBitmap.

Як відзначалося вище, BorlandC++ Builder інкапсулює функції Windows GDI на різних рівнях. Найбільш завершеним є інтерфейс, що надається властивістю Canvas (канва), об'єктного класу канви, його графічних компонент.

Використання канви знімає з програміста турботу про виведення зображень, про ініціалізацію контексту пристрою і його звільнення. Наявності вкладених властивостей (характеристик пера, кисті, шрифтів, растрових зображень) також не вимагає стеження за станами ресурсів – основне завдання – це визначення характеристик для цих графічних об'єктів і грамотне їх використання.

Об'єктний клас канви

Інкапсульовані і переобтяжені функції GDI і WinApi об'єктного класу канви багато авторів відносять до трьох різних рівнів. У цій умовній класифікації функції високого рівня забезпечують можливість малювання ліній, фігур і тексту.

Визначення властивостей і методів маніпулювання графічними примітивами канви віднесені до середнього рівня. Нижній рівень забезпечує доступ до самих функцій Windows GDI. Класифікація не безперечна, але вона дозволяє орієнтуватися в досить великій кількості властивостей і методів канви.

Приклад 5.2.Використання функцій канви для малювання примітивів.

Основи малювання примітивів розглянуті вище при розгляді функцій GDI, тут, на прикладах, не повторюючи вже описані раніше прийоми виведення примітивів, показані лише деякі особливості їх відображення з використанням функцій канви.

Таблиця 5.1

Класифікація функцій GDI і WinApi

Рівень	Метод (Функція)	Властивості	Дія
Високий	MoveTo	PenPos	Визначає поточну позицію пера
	LineTo	PenPos	Малює пряму до заданої точки
	Rectangle		Малює прямокутник
	Ellipse		Малює еліпс
	Arc		Малює дугу
	Polyline		Малює ламану лінію
	PolyBezier		Малює криву Без'є
	Chord		Малює сектор
	DrawFocusRect		Малює прямокутник
	FrameRect		Виводить рамку навколо прямокутника
	Pie		Виводить сектор круга
	TextOut		Виводить текстовий рядок
	TextHeight		Задає висоту текстового рядка
	TextWidth		Задає ширину для виведення текстового рядка
	TextRect		Виведення тексту усередині прямокутника
	FillRect		Заливка вказаного прямокутника кольором і текстурою поточної кисті
	FloodFill		Заливка області канви (довільної форми) заданим кольором
Середній		Pen	Використовується для установки кольору, стилю, ширини і режиму пера
		Brush	Використовується для установки кольору і текстури при заливці графічних фігур і фону канви.
		Font	Використовується для установки шрифту заданого кольору, розміру і стилю
		Pixels	Використовується для читання і запису кольору заданого пікселя канви

Таблиця 5.1

Класифікація функцій GDI і WinApi

Рівень	Метод (Функція)	Властивості	Дія
Високий	MoveTo	PenPos	Визначає поточну позицію пера
	LineTo	PenPos	Малює пряму до заданої точки
	Rectangle		Малює прямокутник
	Ellipse		Малює еліпс
	Arc		Малює дугу
	Polyline		Малює ламану лінію
	PolyBezier		Малює криву Без'є
	Chord		Малює сектор
	DrawFocusRect		Малює прямокутник
	FrameRect		Виводить рамку навколо прямокутника
	Pie		Виводить сектор круга
	TextOut		Виводить текстовий рядок
	TextHeight		Задає висоту текстового рядка
	TextWidth		Задає ширину для виведення текстового рядка
	TextRect		Виведення тексту усередині прямокутника
	FillRect		Заливка вказаного прямокутника кольором і текстурою поточної кисті
	FloodFill		Заливка області канви (довільної форми) заданим кольором
Середній		Pen	Використовується для установки кольору, стилю, ширини і режиму пера
		Brush	Використовується для установки кольору і текстури при заливці графічних фігур і фону канви.
		Font	Використовується для установки шрифту заданого кольору, розміру і стилю
		Pixels	Використовується для читання і запису кольору заданого пікселя канви

208

	CopyRect	CopyMode	Копіює прямокутну область канви в режимі CopyMode
Рівень	Метод(Функція)	Властивості	Дія
	BrushCopy		Копіює прямокутну область канви із заміною кольору
	Draw		Малює бітовий образ, піктограму, метафайл у заданому місці канви
	StretchDraw		Малює бітовий образ, піктограму або метафайл так, щоб цілком заповнити заданий прямокутник
Низький		Handle	Використовується як параметр під час виклику функцій Windows GDI

Найпростіший приклад відноситься до малювання ліній, крім того, показано, як можна задавати параметри пера:

```
void__fastcall
TForm1::Button1Click(TObject *Sender)
{
 // //Задаємо колір пера
Canvas–>Pen–>Color=(TColor) RGB(255,0,0);
 // //Задаємо ширину  пера
Canvas–>Pen–>Width=5;
 // //Можна перемістити перо у вихідну точку так
Canvas–>MoveTo(100,200);
 // //Чи тпереместить перо так
TPointtPoint;
  tPoint.x=100;
  tPoint.y=200;
Canvas–>PenPos=tPoint;
 // //І малюємо лінію від вихідної точки 100,200 до конкечной 0,50
Canvas–>LineTo(0,50);
 // //Звільняти і відновлювати нічого не потрібно
 // //проте канва запам'ятовує встановлені параметри
}
```

Приклад малювання дуги і секторів :

```
void__fastcall
TForm1::Button1Click(TObject *Sender)
{
```


	CopyRect	CopyMode	Копіює прямокутну область канви в режимі CopyMode
Рівень	Метод(Функція)	Властивості	Дія
	BrushCopy		Копіює прямокутну область канви із заміною кольору
	Draw		Малює бітовий образ, піктограму, метафайл у заданому місці канви
	StretchDraw		Малює бітовий образ, піктограму або метафайл так, щоб цілком заповнити заданий прямокутник
Низький		Handle	Використовується як параметр під час виклику функцій Windows GDI

Найпростіший приклад відноситься до малювання ліній, крім того, показано, як можна задавати параметри пера:

```
void__fastcall
TForm1::Button1Click(TObject *Sender)
{
 // //Задаємо колір пера
Canvas–>Pen–>Color=(TColor) RGB(255,0,0);
 // //Задаємо ширину  пера
Canvas–>Pen–>Width=5;
 // //Можна перемістити перо у вихідну точку так
Canvas–>MoveTo(100,200);
 // //Чи тпереместить перо так
TPointtPoint;
  tPoint.x=100;
  tPoint.y=200;
Canvas–>PenPos=tPoint;
 // //І малюємо лінію від вихідної точки 100,200 до конкечной 0,50
Canvas–>LineTo(0,50);
 // //Звільняти і відновлювати нічого не потрібно
 // //проте канва запам'ятовує встановлені параметри
}
Приклад малювання дуги і секторів :
void__fastcall
TForm1::Button1Click(TObject *Sender)
{
```

```
 // //Стиль кисті
Canvas–>Brush–>Style=bsHorizontal;
 // //Колір кисті
Canvas–>Pen–>Color = clBlue;
 // //Малюємо дугу
Canvas–>Arc(0,0,500,500,250,0,50,0);
 // //Малюємо сектор змінюючи стиль взаємодії кольору пера і полотна
Canvas–>Pen–>Mode=pmWhite;
Canvas–>Chord(0,0,250,250,250,125,0,0);
 // //Звільняти і відновлювати нічого не потрібно
 // //Але пам'ятаємо, що канва запам'ятала встановлені параметри
}
```

Приклад малювання ломаних ліній:

```
void__fastcall
TForm1::Button1Click(TObject *Sender)
{
TPointtPoints[6];
Canvas–>Pen–>Color = clRed;
Canvas–>Pen–>Width=3;
tPoints[0].x = 40;
tPoints[0].y = 10;
tPoints[1].x = 20;
tPoints[1].y = 60;
tPoints[2].x = 70;
tPoints[2].y = 30;
tPoints[3].x = 10;
tPoints[3].y = 30;
tPoints[4].x = 60;
tPoints[4].y = 60;
tPoints[5].x = 40;
tPoints[5].y = 10;
Canvas–>Polyline(tPoints, 5);
}
```

Приклад малювання кривих Без'є:

```
void__fastcall
TForm1::Button1Click(TObject *Sender)
{
```



```
 // //Стиль кисті
Canvas–>Brush–>Style=bsHorizontal;
 // //Колір кисті
Canvas–>Pen–>Color = clBlue;
 // //Малюємо дугу
Canvas–>Arc(0,0,500,500,250,0,50,0);
 // //Малюємо сектор змінюючи стиль взаємодії кольору пера і полотна
Canvas–>Pen–>Mode=pmWhite;
Canvas–>Chord(0,0,250,250,250,125,0,0);
 // //Звільняти і відновлювати нічого не потрібно
 // //Але пам'ятаємо, що канва запам'ятала встановлені параметри
}
```

Приклад малювання ломаних ліній:

```
void__fastcall
TForm1::Button1Click(TObject *Sender)
{
TPointtPoints[6];
Canvas–>Pen–>Color = clRed;
Canvas–>Pen–>Width=3;
tPoints[0].x = 40;
tPoints[0].y = 10;
tPoints[1].x = 20;
tPoints[1].y = 60;
tPoints[2].x = 70;
tPoints[2].y = 30;
tPoints[3].x = 10;
tPoints[3].y = 30;
tPoints[4].x = 60;
tPoints[4].y = 60;
tPoints[5].x = 40;
tPoints[5].y = 10;
Canvas–>Polyline(tPoints, 5);
}
```

Приклад малювання кривих Без'є:

```
void__fastcall
TForm1::Button1Click(TObject *Sender)
{
```

```
TPointtPoints[7];
tPoints[0]=TPoint(0,0);
tPoints[1]=TPoint(800,30);
tPoints[2]=TPoint(0,40);
tPoints[3]=TPoint(550,400);
tPoints[4]=TPoint(350,200);
tPoints[5]=TPoint(550,400);
tPoints[6]=TPoint(0,500);
Canvas–>PolyBezier(tPoints, 6);
}
```

Приклад відображення прямокутників і еліпсів різними способами і використання кистей :

```
void__fastcall
TForm1::Button1Click(TObject *Sender)
{
 // //Задаємо колір пера
Canvas–>Pen–>Color=(TColor) RGB(0,255,0);
 // //Задаємо ширину  пера
Canvas–>Pen–>Width=1;
 // //Стиль пера – пунктир
Canvas–>Pen–>Style=psDot;
 // //Стиль виведення замкнутої фігури, що залежить від кольору
пера
 // //і канви ( замінює своїми можливостями) функцію SetBkMode()
 // //Тут прозорий фон
Canvas–>Pen–>Mode=pmCopy;
 // //Малюємо прямокутник по точках
Canvas–>Rectangle(0,0,100,100);
 // //Тут аналог GDI непрозорого фону
Canvas–>Pen–>Mode=pmWhite;
 // //Малюємо еліпс, вписаний у прямокутник
Canvas–>Ellipse(250,250,350,550);
 // //Створюємо перо функцією CreatePen() 1 – товщина пера
  HPEN hPen=CreatePen(PS_DASHDOTDOT, 1, RGB(255,0,0));
 // //Встановлюємо це перо як поточне
Canvas–>Pen–>Handle=hPen;
 // //Змінюємо стиль виводу
Canvas–>Pen–>Mode=pmCopy;
 // //Стиль кисті - вертикальне штрихування
```



```
TPointtPoints[7];
tPoints[0]=TPoint(0,0);
tPoints[1]=TPoint(800,30);
tPoints[2]=TPoint(0,40);
tPoints[3]=TPoint(550,400);
tPoints[4]=TPoint(350,200);
tPoints[5]=TPoint(550,400);
tPoints[6]=TPoint(0,500);
Canvas–>PolyBezier(tPoints, 6);
}
```

Приклад відображення прямокутників і еліпсів різними способами і використання кистей :

```
void__fastcall
TForm1::Button1Click(TObject *Sender)
{
 // //Задаємо колір пера
Canvas–>Pen–>Color=(TColor) RGB(0,255,0);
 // //Задаємо ширину  пера
Canvas–>Pen–>Width=1;
 // //Стиль пера – пунктир
Canvas–>Pen–>Style=psDot;
 // //Стиль виведення замкнутої фігури, що залежить від кольору
пера
 // //і канви ( замінює своїми можливостями) функцію SetBkMode()
 // //Тут прозорий фон
Canvas–>Pen–>Mode=pmCopy;
 // //Малюємо прямокутник по точках
Canvas–>Rectangle(0,0,100,100);
 // //Тут аналог GDI непрозорого фону
Canvas–>Pen–>Mode=pmWhite;
 // //Малюємо еліпс, вписаний у прямокутник
Canvas–>Ellipse(250,250,350,550);
 // //Створюємо перо функцією CreatePen() 1 – товщина пера
  HPEN hPen=CreatePen(PS_DASHDOTDOT, 1, RGB(255,0,0));
 // //Встановлюємо це перо як поточне
Canvas–>Pen–>Handle=hPen;
 // //Змінюємо стиль виводу
Canvas–>Pen–>Mode=pmCopy;
 // //Стиль кисті - вертикальне штрихування
```

```
Canvas–>Brush–>Style=bsVertical;
 // //Можна визначити прозорість
SetBkMode(Canvas–>Handle, OPAQUE);
 // //Малюємо прямокутник із закругленими краями
Canvas–>RoundRect(100,100,200,200,50,50);
 // //Задаємо координати
TRecttRect;        //Координати точок
  tRect.Left=100;     //Ліва
  tRect.Right=500;    //Права
  tRect.Top=250;      //Верхня
  tRect.Bottom=450;   //Нижня
 // //Використовуємо кисть для зафрбовування об'єкта
Canvas–>Brush–>Color=(TColor) RGB(0,0,255);
Canvas–>Rectangle(tRect);
Canvas–>Brush–>Color=(TColor) RGB(0,255,255);
 // //Стиль кисті
Canvas–>Brush–>Style=bsCross;
 // //Чи по  координатах
Canvas–>RoundRect(100,300,250,450,50,50);
}
```

Використання кисті для заливки фігур:

```
void__fastcall
TForm1::Button1Click(TObject *Sender)
{
Canvas–>Brush–>Color=(TColor) RGB(0,255,255);
TRecttRect(0,0,100,100);
Canvas–>FillRect(tRect);
 // //Малюємо прямокутник по точках
Canvas–>Rectangle(0,0,100,100);
 // //Звільняти і відновлювати нічого не потрібно
}
```

І цікавий ефект заповнення канви кольором кисті :

```
void__fastcall
TForm1::Button1Click(TObject *Sender)
{
 // //Параметр fsBorder -  заповнити усю область
 // //кольором кисті, до краю канви
Canvas–>Brush–>Color=(TColor) RGB(255,0,255);
 // //Вихідна точка в центрі канви, в даному випадку
```



```
Canvas–>Brush–>Style=bsVertical;
 // //Можна визначити прозорість
SetBkMode(Canvas–>Handle, OPAQUE);
 // //Малюємо прямокутник із закругленими краями
Canvas–>RoundRect(100,100,200,200,50,50);
 // //Задаємо координати
TRecttRect;        //Координати точок
  tRect.Left=100;     //Ліва
  tRect.Right=500;    //Права
  tRect.Top=250;      //Верхня
  tRect.Bottom=450;   //Нижня
 // //Використовуємо кисть для зафрбовування об'єкта
Canvas–>Brush–>Color=(TColor) RGB(0,0,255);
Canvas–>Rectangle(tRect);
Canvas–>Brush–>Color=(TColor) RGB(0,255,255);
 // //Стиль кисті
Canvas–>Brush–>Style=bsCross;
 // //Чи по  координатах
Canvas–>RoundRect(100,300,250,450,50,50);
}
```

Використання кисті для заливки фігур:

```
void__fastcall
TForm1::Button1Click(TObject *Sender)
{
Canvas–>Brush–>Color=(TColor) RGB(0,255,255);
TRecttRect(0,0,100,100);
Canvas–>FillRect(tRect);
 // //Малюємо прямокутник по точках
Canvas–>Rectangle(0,0,100,100);
 // //Звільняти і відновлювати нічого не потрібно
}
```

І цікавий ефект заповнення канви кольором кисті :

```
void__fastcall
TForm1::Button1Click(TObject *Sender)
{
 // //Параметр fsBorder -  заповнити усю область
 // //кольором кисті, до краю канви
Canvas–>Brush–>Color=(TColor) RGB(255,0,255);
 // //Вихідна точка в центрі канви, в даному випадку
```

```
 // //вікна додатка
Canvas–>FloodFill(Width/2, Height/2, NULL, fsBorder);
 // //Міняємо колір і повторюємо
Canvas–>Brush–>Color=(TColor) RGB(255,0,0);
Canvas–>FloodFill(Width/2, Height/2, NULL, fsBorder);
}
```

Малювання рамки навколо прямокутника:

```
void__fastcall
TForm1::Button1Click(TObject *Sender)
{
Canvas–>Brush–>Color=(TColor) RGB(125,0,255);
TRecttRect;         //Координати точок
  tRect.Left=100;       //Ліва
  tRect.Right=500;      //Права
  tRect.Top=250;        //Верхня
  tRect.Bottom=450;     //Нижня
Canvas–>FrameRect(tRect);
}
```

З прикладів видно, що (з урахуванням деякої модифікації параметрів функцій) малювання графічних примітивів аналогічно інтерфейсу GDI. Проте, оскільки не вимагається стежити за станом контексту і його графічних об'єктів, написання коду значно спрощується.

Контрольні питання

1. У чому полягає необхідність використання функцій WinAPI?
2. Дайте характеристику контексту графічного пристрою. Які типи контекстів пристроїв підтримуються у Windows?
3. Наведіть приклади API-функцій, укажіть їх призначення.
4. Наведіть приклади функцій для побудови графічних примітивів з урахуванням кількості аргументів.
5. Що таке канва? Для чого використовується ця властивість?

Тестові завдання

Питання 1. Сполучною ланкою між програмою побудови графічного зображення та пристроєм виведення його на екран є:

А) бітова карта
Б) контекст пристрою
В) інтерфейс графічного пристрою

А) контекст дисплея та принтера


```
 // //вікна додатка
Canvas–>FloodFill(Width/2, Height/2, NULL, fsBorder);
 // //Міняємо колір і повторюємо
Canvas–>Brush–>Color=(TColor) RGB(255,0,0);
Canvas–>FloodFill(Width/2, Height/2, NULL, fsBorder);
}
```

Малювання рамки навколо прямокутника:

```
void__fastcall
TForm1::Button1Click(TObject *Sender)
{
Canvas–>Brush–>Color=(TColor) RGB(125,0,255);
TRecttRect;         //Координати точок
  tRect.Left=100;       //Ліва
  tRect.Right=500;      //Права
  tRect.Top=250;        //Верхня
  tRect.Bottom=450;     //Нижня
Canvas–>FrameRect(tRect);
}
```

З прикладів видно, що (з урахуванням деякої модифікації параметрів функцій) малювання графічних примітивів аналогічно інтерфейсу GDI. Проте, оскільки не вимагається стежити за станом контексту і його графічних об'єктів, написання коду значно спрощується.

Контрольні питання

1. У чому полягає необхідність використання функцій WinAPI?
2. Дайте характеристику контексту графічного пристрою. Які типи контекстів пристроїв підтримуються у Windows?
3. Наведіть приклади API-функцій, укажіть їх призначення.
4. Наведіть приклади функцій для побудови графічних примітивів з урахуванням кількості аргументів.
5. Що таке канва? Для чого використовується ця властивість?

Тестові завдання

Питання 1. Сполучною ланкою між програмою побудови графічного зображення та пристроєм виведення його на екран є:

А) бітова карта
Б) контекст пристрою
В) інтерфейс графічного пристрою

А) контекст дисплея та принтера

Питання 2. У Windows підтримуються наступні типи контекстів пристроїв:
Б) контекст пам'яті та інформаційний контекст
В) обидві відповіді вірні

Питання 3. Функція LineTo(intx2, inty2) призначена для:
А) малювання лінії з координатами (x1,y1,x2,y2)
Б) малювання лінії починаючи з поточної позиції (x2, y2) до вказаних координат за допомогою функції MoveTo()
В) малювання лінії починаючи з поточної позиції до вказаних координат
Г) всі відповіді вірні

Питання 4. Функція Polyline() малює:
А) ламану лінію по масиву точок
Б) багатокутник, що складається з трьох або більше вершин
В) сектор еліпса, що вписується в багатокутник

Питання 5. Функція Arc() малює:
А) еліпс, вписаний у прямокутник, розмір якого визначається координатами верхнього і нижнього кута
Б) еліптичну дугу, логічно обмежену прямокутником, розмір якого визначається координатами верхнього і нижнього кута
В) прямокутник із закругленими краями, розмір якого визначається координатами верхнього, нижнього кута і координатами скруглення

Питання 6. Функція Rectangle()малює:
А) прямокутник, розмір якого визначається координатами верхнього і нижнього кута
Б) прямокутник із закругленими краями, розмір якого визначається координатами верхнього, нижнього кута і координатами округлення
В) сегмент еліпса

Питання 7. Функція Pie() малює:
А) сектор еліпса, що вписується в багатокутник
Б) еліптичну дугу, логічно обмежену прямокутником, розмір якого визначається координатами верхнього і нижнього кута

Питання 2. У Windows підтримуються наступні типи контекстів пристроїв:
Б) контекст пам'яті та інформаційний контекст
В) обидві відповіді вірні

Питання 3. Функція LineTo(intx2, inty2) призначена для:
А) малювання лінії з координатами (x1,y1,x2,y2)
Б) малювання лінії починаючи з поточної позиції (x2, y2) до вказаних координат за допомогою функції MoveTo()
В) малювання лінії починаючи з поточної позиції до вказаних координат
Г) всі відповіді вірні

Питання 4. Функція Polyline() малює:
А) ламану лінію по масиву точок
Б) багатокутник, що складається з трьох або більше вершин
В) сектор еліпса, що вписується в багатокутник

Питання 5. Функція Arc() малює:
А) еліпс, вписаний у прямокутник, розмір якого визначається координатами верхнього і нижнього кута
Б) еліптичну дугу, логічно обмежену прямокутником, розмір якого визначається координатами верхнього і нижнього кута
В) прямокутник із закругленими краями, розмір якого визначається координатами верхнього, нижнього кута і координатами скруглення

Питання 6. Функція Rectangle()малює:
А) прямокутник, розмір якого визначається координатами верхнього і нижнього кута
Б) прямокутник із закругленими краями, розмір якого визначається координатами верхнього, нижнього кута і координатами округлення
В) сегмент еліпса

Питання 7. Функція Pie() малює:
А) сектор еліпса, що вписується в багатокутник
Б) еліптичну дугу, логічно обмежену прямокутником, розмір якого визначається координатами верхнього і нижнього кута

В) еліпс, вписаний в прямокутник, розмір якого визначається координатами верхнього і нижнього кута

Питання 8. Функція Chord() малює:

А) еліпс, вписаний в прямокутник, розмір якого визначається координатами верхнього і нижнього кута
Б) сектор еліпса, що вписується в багатокутник
В) сегмент еліпса

Питання 9. Об'єкт *канва* надає

А) контекст графічного пристрою
Б) надає бітову карту поверхні вікна додатка
В) обидві відповіді вірні

Питання 10. Функції побудови графічних зображень знаходяться у бібліотеці:

А) graphiks.h
Б) stdlib.h
В) graphics.h

В) еліпс, вписаний в прямокутник, розмір якого визначається координатами верхнього і нижнього кута

Питання 8. Функція Chord() малює:

А) еліпс, вписаний в прямокутник, розмір якого визначається координатами верхнього і нижнього кута
Б) сектор еліпса, що вписується в багатокутник
В) сегмент еліпса

Питання 9. Об'єкт *канва* надає

А) контекст графічного пристрою
Б) надає бітову карту поверхні вікна додатка
В) обидві відповіді вірні

Питання 10. Функції побудови графічних зображень знаходяться у бібліотеці:

А) graphiks.h
Б) stdlib.h
В) graphics.h

Глава 6
Графічна бібліотека OpenGL

6.1. Загальні відомості про OpenGL

OpenGL є одним із найбільш популярних прикладних програмних інтерфейсів (API – Application Programming Interface) для розробки додатків в області двовимірної й тривимірної графіки.

Стандарт OpenGL (Open Graphics Library – відкрита графічна бібліотека) був розроблений і затверджений в 1992 році провідними фірмами в області розробки програмного забезпечення як ефективний апаратно-незалежний інтерфейс, придатний для реалізації на різних платформах. Основою стандарту стала бібліотека IRIS GL, розроблена фірмою Silicon Graphics Inc. Бібліотека нараховує близько 120 різних команд, які програміст використовує для завдання об'єктів і операцій, необхідних для написання інтерактивних графічних додатків. На сьогоднішній день графічна система OpenGL підтримується більшістю виробників апаратних і програмних платформ. Ця система доступна тим, хто працює в середовищі Windows, користувачам комп'ютерів Apple. Вільно розповсюджувані коди системи Mesa (пакет API на базі OpenGL) можна компілювати в більшості операційних систем, у тому числі в Linux.

Характерними рисами OpenGL, які забезпечили поширення й розвиток цього графічного стандарту, є[2,15]:

- ***Стабільність.*** Доповнення й зміни в стандарті реалізуються таким чином, щоб зберегти сумісність із розробленим раніше програмним забезпеченням.

Глава 6
Графічна бібліотека OpenGL

6.1. Загальні відомості про OpenGL

OpenGL є одним із найбільш популярних прикладних програмних інтерфейсів (API – Application Programming Interface) для розробки додатків в області двовимірної й тривимірної графіки.

Стандарт OpenGL (Open Graphics Library – відкрита графічна бібліотека) був розроблений і затверджений в 1992 році провідними фірмами в області розробки програмного забезпечення як ефективний апаратно-незалежний інтерфейс, придатний для реалізації на різних платформах. Основою стандарту стала бібліотека IRIS GL, розроблена фірмою Silicon Graphics Inc. Бібліотека нараховує близько 120 різних команд, які програміст використовує для завдання об'єктів і операцій, необхідних для написання інтерактивних графічних додатків. На сьогоднішній день графічна система OpenGL підтримується більшістю виробників апаратних і програмних платформ. Ця система доступна тим, хто працює в середовищі Windows, користувачам комп'ютерів Apple. Вільно розповсюджувані коди системи Mesa (пакет API на базі OpenGL) можна компілювати в більшості операційних систем, у тому числі в Linux.

Характерними рисами OpenGL, які забезпечили поширення й розвиток цього графічного стандарту, є[2,15]:

- ***Стабільність.*** Доповнення й зміни в стандарті реалізуються таким чином, щоб зберегти сумісність із розробленим раніше програмним забезпеченням.

- ***Надійність і переносимість.*** Додатки, що використовують OpenGL, гарантують однаковий візуальний результат незалежно від типу операційної системи, що використовується, й організації відображення інформації. Крім того, ці додатки можуть виконуватися як на персональних комп'ютерах, так і на робочих станціях і суперкомп'ютерах.
- ***Легкість застосування.*** Стандарт OpenGL має продуману структуру й інтуїтивно зрозумілий інтерфейс, який дозволяє з меншими витратами створювати ефективні додатки, що містять менше рядків коду, ніж при використанні інших графічних бібліотек. Необхідні функції для забезпечення сумісності з різним устаткуванням реалізовані на рівні бібліотеки й значно спрощують розробку додатків.

Наявність гарного базового пакета для роботи із тривимірними додатками спрощує розуміння тих, хто навчається, ключових тем курсу комп'ютерної графіки - моделювання тривимірних об'єктів, зафарбовування, накладання текстури, анімацію тощо.

Основні можливості

Всі функції OpenGL можна розділити на п'ять категорій:

- ***Функції опису примітивів*** визначають об'єкти нижнього рівня ієрархії (примітиви), які може відображати графічна підсистема. В OpenGL як примітиви виступають точки, лінії, багатокутники.
- ***Функції опису джерел світла*** служать для опису положення й параметрів джерел світла, розташованих у тривимірній сцені.
- ***Функції задавання атрибутів***. За допомогою задавання атрибутів програміст визначає, як будуть виглядати на екрані відображувані об'єкти. Інакше кажучи, якщо за допомогою примітивів визначається, що з'явиться на екрані, то атрибути визначають ***спосіб*** виведення на екран. Як атрибути OpenGL дозволяє задавати кольори, характеристики матеріалу, текстури, параметри освітлення.
- ***Функції візуалізації*** дозволять задати положення спостерігача у віртуальному просторі, параметри об'єктива камери. Знаючи ці параметри, система зможе не тільки правильно побудувати

- ***Надійність і переносимість.*** Додатки, що використовують OpenGL, гарантують однаковий візуальний результат незалежно від типу операційної системи, що використовується, й організації відображення інформації. Крім того, ці додатки можуть виконуватися як на персональних комп'ютерах, так і на робочих станціях і суперкомп'ютерах.
- ***Легкість застосування.*** Стандарт OpenGL має продуману структуру й інтуїтивно зрозумілий інтерфейс, який дозволяє з меншими витратами створювати ефективні додатки, що містять менше рядків коду, ніж при використанні інших графічних бібліотек. Необхідні функції для забезпечення сумісності з різним устаткуванням реалізовані на рівні бібліотеки й значно спрощують розробку додатків.

Наявність гарного базового пакета для роботи із тривимірними додатками спрощує розуміння тих, хто навчається, ключових тем курсу комп'ютерної графіки - моделювання тривимірних об'єктів, зафарбовування, накладання текстури, анімацію тощо.

Основні можливості

Всі функції OpenGL можна розділити на п'ять категорій:

- ***Функції опису примітивів*** визначають об'єкти нижнього рівня ієрархії (примітиви), які може відображати графічна підсистема. В OpenGL як примітиви виступають точки, лінії, багатокутники.
- ***Функції опису джерел світла*** служать для опису положення й параметрів джерел світла, розташованих у тривимірній сцені.
- ***Функції задавання атрибутів***. За допомогою задавання атрибутів програміст визначає, як будуть виглядати на екрані відображувані об'єкти. Інакше кажучи, якщо за допомогою примітивів визначається, що з'явиться на екрані, то атрибути визначають ***спосіб*** виведення на екран. Як атрибути OpenGL дозволяє задавати кольори, характеристики матеріалу, текстури, параметри освітлення.
- ***Функції візуалізації*** дозволять задати положення спостерігача у віртуальному просторі, параметри об'єктива камери. Знаючи ці параметри, система зможе не тільки правильно побудувати

зображення, але й відітнути об'єкти, які опинилися поза полем зору.

- Набір ***функцій геометричних перетворень*** дозволяє програмістові виконувати різні перетворення об'єктів – поворот, перенесення, масштабування, відображення.

При цьому OpenGL може виконувати додаткові операції, такі як використання сплайнів для побудови ліній і поверхонь, видалення невидимих фрагментів зображень, робота із зображеннями на рівні пікселів і т.д.

Інтерфейс OpenGL

OpenGL складається з набору бібліотек. Всі базові функції зберігаються в основній бібліотеці, для позначення якої надалі ми будемо використовути абревіатуру ***GL***. Крім основної, OpenGL містить у собі кілька додаткових бібліотек.

Перша з них – ***бібліотека утилітGL*(*GLU – GL Utility*)**. Всі функції цієї бібліотеки визначені через базові функції GL. До складу GLU увійшла реалізація більш складних функцій, таких як набір популярних геометричних примітивів (куб, куля, циліндр, диск), функції побудови сплайнів, реалізація додаткових операцій над матрицями й т.п.

OpenGL не містить у собі ніяких спеціальних команд для роботи з вікнами або введення інформації від користувача. Тому були створені спеціальні окремі бібліотеки для забезпечення часто використовуваних функцій взаємодії з користувачем і для відображення інформації за допомогою віконної підсистеми. Найбільш використовуваною є бібліотека GLUT (GL Utility Toolkit). Формально GLUT не входить в OpenGL, але de facto включається майже в усі його дистрибутиви й має реалізації для різних платформ GLUT, тобто надає тільки мінімально необхідний набір функцій для створення OpenGL-додатка. Функціонально аналогічна бібліотека GLX менш популярна. Надалі в цьому посібнику як основний буде розглядатися GLUT.

Крім того, функції, специфічні для конкретної віконної підсистеми, звичайно входять у її прикладний програмний інтерфейс. Так, функції, що підтримують виконання OpenGL, є в складі Win32 API.

зображення, але й відітнути об'єкти, які опинилися поза полем зору.

- Набір ***функцій геометричних перетворень*** дозволяє програмістові виконувати різні перетворення об'єктів – поворот, перенесення, масштабування, відображення.

При цьому OpenGL може виконувати додаткові операції, такі як використання сплайнів для побудови ліній і поверхонь, видалення невидимих фрагментів зображень, робота із зображеннями на рівні пікселів і т.д.

Інтерфейс OpenGL

OpenGL складається з набору бібліотек. Всі базові функції зберігаються в основній бібліотеці, для позначення якої надалі ми будемо використовути абревіатуру ***GL***. Крім основної, OpenGL містить у собі кілька додаткових бібліотек.

Перша з них – ***бібліотека утилітGL*(*GLU – GL Utility*)**. Всі функції цієї бібліотеки визначені через базові функції GL. До складу GLU увійшла реалізація більш складних функцій, таких як набір популярних геометричних примітивів (куб, куля, циліндр, диск), функції побудови сплайнів, реалізація додаткових операцій над матрицями й т.п.

OpenGL не містить у собі ніяких спеціальних команд для роботи з вікнами або введення інформації від користувача. Тому були створені спеціальні окремі бібліотеки для забезпечення часто використовуваних функцій взаємодії з користувачем і для відображення інформації за допомогою віконної підсистеми. Найбільш використовуваною є бібліотека GLUT (GL Utility Toolkit). Формально GLUT не входить в OpenGL, але de facto включається майже в усі його дистрибутиви й має реалізації для різних платформ GLUT, тобто надає тільки мінімально необхідний набір функцій для створення OpenGL-додатка. Функціонально аналогічна бібліотека GLX менш популярна. Надалі в цьому посібнику як основний буде розглядатися GLUT.

Крім того, функції, специфічні для конкретної віконної підсистеми, звичайно входять у її прикладний програмний інтерфейс. Так, функції, що підтримують виконання OpenGL, є в складі Win32 API.

На рис.6.1 схематично зображена організація системи бібліотек у версії, що працює під управлінням системи Windows. Аналогічна організація використовується й в інших версіях OpenGL.

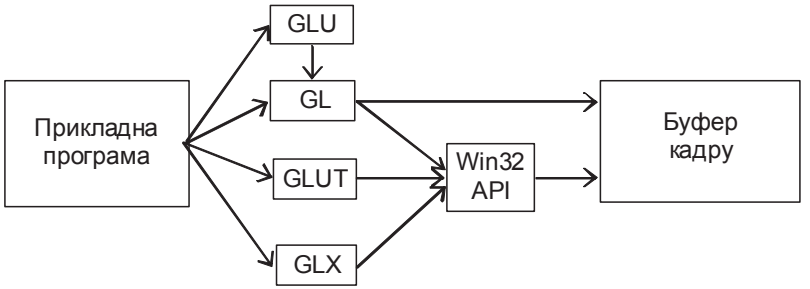


Рис. 6.1. Організація бібліотеки OpenGL

Архітектура OpenGL

Функції OpenGL реалізовані в моделі клієнт-сервер. Додаток виступає в ролі клієнта – він виробляє команди, а сервер OpenGL інтерпретує й виконує їх. Сам сервер може перебувати як на комп'ютері, де перебуває клієнт (наприклад, у вигляді бібліотеки, що завантажує динамічно, - DLL), так і на іншому (при цьому може бути використаний спеціальний протокол передачі даних між машинами).

GL обробляє й малює в буфері кадру графічні примітиви з урахуванням деякого числа обраних режимів. Кожний примітив – це точка, відрізок, багатокутник і т.д. Кожний режим може бути змінений незалежно від інших. Визначення примітивів, вибір режимів та інших операцій описуються за допомогою команд у формі викликів функцій прикладної бібліотеки.

Примітиви визначаються набором з однієї або більше вершин (vertex). Вершина визначає точку, кінець відрізка або кут багатокутника. З кожною вершиною асоціюються деякі дані (координати, кольори, нормаль, текстурні координати й т.д.), що називаються атрибутами. У переважній більшості випадків кожна вершина обробляється незалежно від інших.

На рис.6.1 схематично зображена організація системи бібліотек у версії, що працює під управлінням системи Windows. Аналогічна організація використовується й в інших версіях OpenGL.

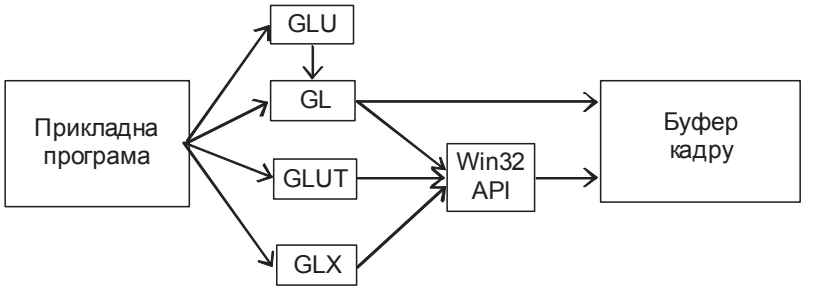


Рис. 6.1. Організація бібліотеки OpenGL

Архітектура OpenGL

Функції OpenGL реалізовані в моделі клієнт-сервер. Додаток виступає в ролі клієнта – він виробляє команди, а сервер OpenGL інтерпретує й виконує їх. Сам сервер може перебувати як на комп'ютері, де перебуває клієнт (наприклад, у вигляді бібліотеки, що завантажує динамічно, - DLL), так і на іншому (при цьому може бути використаний спеціальний протокол передачі даних між машинами).

GL обробляє й малює в буфері кадру графічні примітиви з урахуванням деякого числа обраних режимів. Кожний примітив – це точка, відрізок, багатокутник і т.д. Кожний режим може бути змінений незалежно від інших. Визначення примітивів, вибір режимів та інших операцій описуються за допомогою команд у формі викликів функцій прикладної бібліотеки.

Примітиви визначаються набором з однієї або більше вершин (vertex). Вершина визначає точку, кінець відрізка або кут багатокутника. З кожною вершиною асоціюються деякі дані (координати, кольори, нормаль, текстурні координати й т.д.), що називаються атрибутами. У переважній більшості випадків кожна вершина обробляється незалежно від інших.

З погляду архітектури, графічна система OpenGL є конвеєром, що складається з декількох послідовних етапів обробки графічних даних.

Команди OpenGL завжди обробляються в тому порядку, в якому вони надходять, хоча можуть відбуватися затримки перед тим, як появиться ефект від їхнього виконання. У більшості випадків OpenGL надає безпосередній інтерфейс, тобто визначення об'єкта викликає його візуалізацію в буфері кадру.

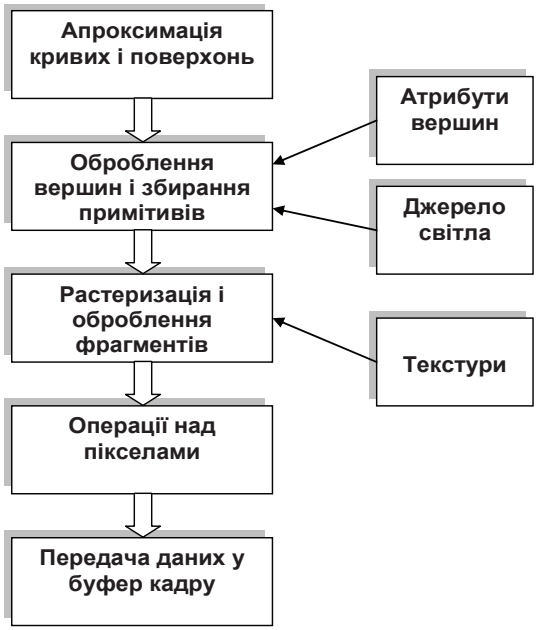


Рис. 6.2. Функціонування конвеєра OpenGL

З погляду розроблювачів, OpenGL - це набір команд, які управляють використанням графічних пристроїв. Якщо апаратура складається тільки з адресованого буфера кадру, тоді OpenGL повинен бути реалізований повністю з використанням ресурсів центрального процесора. Звичайно графічна апаратура надає різні рівні приско-

З погляду архітектури, графічна система OpenGL є конвеєром, що складається з декількох послідовних етапів обробки графічних даних.

Команди OpenGL завжди обробляються в тому порядку, в якому вони надходять, хоча можуть відбуватися затримки перед тим, як появиться ефект від їхнього виконання. У більшості випадків OpenGL надає безпосередній інтерфейс, тобто визначення об'єкта викликає його візуалізацію в буфері кадру.

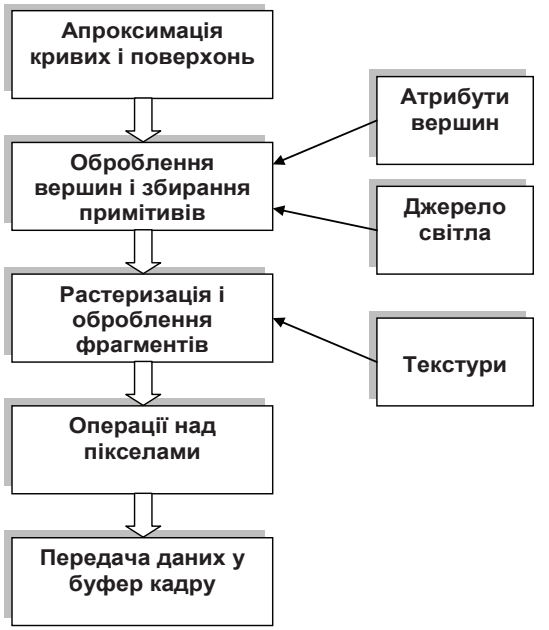


Рис. 6.2. Функціонування конвеєра OpenGL

З погляду розроблювачів, OpenGL - це набір команд, які управляють використанням графічних пристроїв. Якщо апаратура складається тільки з адресованого буфера кадру, тоді OpenGL повинен бути реалізований повністю з використанням ресурсів центрального процесора. Звичайно графічна апаратура надає різні рівні приско-

рення: від апаратної реалізації виводу ліній і багатокутників до витончених графічних процесорів з підтримкою різних операцій над геометричними даними.

OpenGL є прошарком між апаратурою й користувальницьким рівнем, що дозволяє надавати єдиний інтерфейс на різних платформах, використовуючи можливості апаратної підтримки.

Крім того, OpenGL можна розглядати як кінцевий автомат, стан якого визначається безліччю значень спеціальних змінних і значеннями поточної нормалі, кольору, координат текстури й інших атрибутів і ознак. Вся ця інформація буде використана при надходженні в графічну систему координат вершини для побудови фігури, в яку вона входить. Зміна станів відбувається за допомогою команд, які оформляються як виклики функцій.

Синтаксис команд

Визначення команд GL перебувають у файлі gl.h, для виклику якого потрібно написати

```
#include <gl/gl.h>
```

Для роботи з бібліотекою GLU потрібно аналогічно викликати файл glu.h. Версії цих бібліотек, як правило, включаються в дистрибутиви систем програмування, таких як Microsoft Visual C++ або Borland C++.

На відміну від стандартних бібліотек, пакет GLUT потрібно інсталювати і підключати окремо [15].

Всі команди (процедури й функції) бібліотеки GL починаються із префікса gl, всі константи - із префікса GL_. Відповідні команди й константи бібліотек GLU і GLUT аналогічно мають префікси glu (GLU_) і glut (GLUT_). OpenGL була розроблена для мови C, а не для C++, тому замість перевантажених поліморфних функцій у цій бібліотеці реалізовано набір функцій зі схожими іменами, в яких умовно позначено кількість параметрів функцій. Імена функцій в OpenGL обрані згідно з правилом:

```
ім'я функції[число параметрів][тип параметрів][покажчик]
```


рення: від апаратної реалізації виводу ліній і багатокутників до витончених графічних процесорів з підтримкою різних операцій над геометричними даними.

OpenGL є прошарком між апаратурою й користувальницьким рівнем, що дозволяє надавати єдиний інтерфейс на різних платформах, використовуючи можливості апаратної підтримки.

Крім того, OpenGL можна розглядати як кінцевий автомат, стан якого визначається безліччю значень спеціальних змінних і значеннями поточної нормалі, кольору, координат текстури й інших атрибутів і ознак. Вся ця інформація буде використана при надходженні в графічну систему координат вершини для побудови фігури, в яку вона входить. Зміна станів відбувається за допомогою команд, які оформляються як виклики функцій.

Синтаксис команд

Визначення команд GL перебувають у файлі gl.h, для виклику якого потрібно написати

```
#include <gl/gl.h>
```

Для роботи з бібліотекою GLU потрібно аналогічно викликати файл glu.h. Версії цих бібліотек, як правило, включаються в дистрибутиви систем програмування, таких як Microsoft Visual C++ або Borland C++.

На відміну від стандартних бібліотек, пакет GLUT потрібно інсталювати і підключати окремо [15].

Всі команди (процедури й функції) бібліотеки GL починаються із префікса gl, всі константи - із префікса GL_. Відповідні команди й константи бібліотек GLU і GLUT аналогічно мають префікси glu (GLU_) і glut (GLUT_). OpenGL була розроблена для мови C, а не для C++, тому замість перевантажених поліморфних функцій у цій бібліотеці реалізовано набір функцій зі схожими іменами, в яких умовно позначено кількість параметрів функцій. Імена функцій в OpenGL обрані згідно з правилом:

```
ім'я функції[число параметрів][тип параметрів][покажчик]
```

Тип параметрів позначається буквами англійського алфавіту:

'b'– байт зі знаком (charили GLbyte)
's'– коротке ціле (shortили GLshort)
'i' – ціле (intили GLint)
'f'– дійсне (floatили GLfloat)
'd'– дійсні з подвійною точністю(doubleили GLdouble)
'ub'– беззнаковий байт (unsignedcharили GLubyte)
'us'– беззнакове коротке ціле (unsignedshortили GLushort)
'ui'– беззнакове цілое (unsignedintили GLuint)
'v'– масив з n параметрів вказаного типу.

Наявність третього символу показує, що як параметри функції використовується покажчик на масив значень. Для скалярних форм він відсутній.

Символи у квадратних дужках у деяких назвах не використовуються. Наприклад, команда `glVertex2i()` описана в бібліотеці GL, і використовує як параметри два цілих числа, а команда `glColor3fv()` використовує як параметр покажчик на масив із трьох дійсних чисел.

Приклад 6.1. Типова програма, що використовує OpenGL.

Програма починається з визначення вікна, у якому буде відбуватися відображення. Потім створюється контекст (клієнт) OpenGL і асоціюється із цим вікном. Далі програміст може вільно використовувати команди й операції OpenGL API.

Нижче наведений текст невеликої програми, написаної з використанням бібліотеки GLUT - своєрідний аналог класичного прикладу “Hello, C++!”.

Усе, що робить ця програма - малює в центрі вікна червоний квадрат. Проте навіть на цьому простому прикладі можна зрозуміти принципи програмування за допомогою OpenGL.

```
#include <stdlib.h>
/* підключаємо бібліотеку GLUT */
#include <gl/glut.h>
/* початкова ширина й висота вікна */
GLint Width = 512, Height = 512;
/* розмір куба */
const int CubeSize = 200;
/* ця функція керує всім виведенням на екран */
```


Тип параметрів позначається буквами англійського алфавіту:

'b'– байт зі знаком (charили GLbyte)
's'– коротке ціле (shortили GLshort)
'i' – ціле (intили GLint)
'f'– дійсне (floatили GLfloat)
'd'– дійсні з подвійною точністю(doubleили GLdouble)
'ub'– беззнаковий байт (unsignedcharили GLubyte)
'us'– беззнакове коротке ціле (unsignedshortили GLushort)
'ui'– беззнакове цілое (unsignedintили GLuint)
'v'– масив з n параметрів вказаного типу.

Наявність третього символу показує, що як параметри функції використовується покажчик на масив значень. Для скалярних форм він відсутній.

Символи у квадратних дужках у деяких назвах не використовуються. Наприклад, команда `glVertex2i()` описана в бібліотеці GL, і використовує як параметри два цілих числа, а команда `glColor3fv()` використовує як параметр покажчик на масив із трьох дійсних чисел.

Приклад 6.1. Типова програма, що використовує OpenGL.

Програма починається з визначення вікна, у якому буде відбуватися відображення. Потім створюється контекст (клієнт) OpenGL і асоціюється із цим вікном. Далі програміст може вільно використовувати команди й операції OpenGL API.

Нижче наведений текст невеликої програми, написаної з використанням бібліотеки GLUT - своєрідний аналог класичного прикладу “Hello, C++!”.

Усе, що робить ця програма - малює в центрі вікна червоний квадрат. Проте навіть на цьому простому прикладі можна зрозуміти принципи програмування за допомогою OpenGL.

```
#include <stdlib.h>
/* підключаємо бібліотеку GLUT */
#include <gl/glut.h>
/* початкова ширина й висота вікна */
GLint Width = 512, Height = 512;
/* розмір куба */
const int CubeSize = 200;
/* ця функція керує всім виведенням на екран */
```

```
void Display(void)
{
    int left, right, top, bottom;
    left  = (Width - CubeSize) / 2;
    right = left + CubeSize;
    bottom = (Height - CubeSize) / 2;
    top = bottom + CubeSize;
    glClearColor(0, 0, 0, 1);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3ub(255,0,0);
    glBegin(GL_QUADS);
      glVertex2f(left,bottom);
      glVertex2f(left,top);
      glVertex2f(right,top);
      glVertex2f(right,bottom);
    glEnd();
    glFinish();
}
/* Функція викликається при зміні розмірів вікна */
void Reshape(GLint w, GLint h)
{
    Width = w;
    Height = h;
    /* встановлюємо розміри області відображення */
    glViewport(0, 0, w, h);
    /* ортографічна проекція */
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0, w, 0, h, -1.0, 1.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
/* Функція обробляє повідомлення від клавіатури */
void
Keyboard( unsigned char key, int x, int y )
{
#define ESCAPE '\033'

    if( key == ESCAPE )
        exit(0);
}
/* Головний цикл додатка */
main(int argc, char *argv[])
```



```
void Display(void)
{
    int left, right, top, bottom;
    left  = (Width - CubeSize) / 2;
    right = left + CubeSize;
    bottom = (Height - CubeSize) / 2;
    top = bottom + CubeSize;
    glClearColor(0, 0, 0, 1);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3ub(255,0,0);
    glBegin(GL_QUADS);
      glVertex2f(left,bottom);
      glVertex2f(left,top);
      glVertex2f(right,top);
      glVertex2f(right,bottom);
    glEnd();
    glFinish();
}
/* Функція викликається при зміні розмірів вікна */
void Reshape(GLint w, GLint h)
{
    Width = w;
    Height = h;
    /* встановлюємо розміри області відображення */
    glViewport(0, 0, w, h);
    /* ортографічна проекція */
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0, w, 0, h, -1.0, 1.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
/* Функція обробляє повідомлення від клавіатури */
void
Keyboard( unsigned char key, int x, int y )
{
#define ESCAPE '\033'

    if( key == ESCAPE )
        exit(0);
}
/* Головний цикл додатка */
main(int argc, char *argv[])
```

```
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB);
    glutInitWindowSize(Width, Height);
    glutCreateWindow("Red square example");
    glutDisplayFunc(Display);
    glutReshapeFunc(Reshape);
    glutKeyboardFunc(Keyboard);
    glutMainLoop();
}
```

Це повністю завершена програма, що повинна компілюватися й працювати на будь-якій системі, що підтримує OpenGL і GLUT.

Бібліотека GLUT підтримує взаємодію з користувачем за допомогою так званих функцій зі зворотним викликом (*callback function*). Якщо користувач зсунув мишу, натиснув на кнопку клавіатури або змінив розміри вікна, відбувається подія й викликається відповідна функція користувача – оброблювач подій (функція зі зворотним викликом).

Розглянемо більш детально наведений приклад.Функція `main` складається із трьох частин - ініціалізації вікна, у якому буде малювати OpenGL, настроювання функцій зі зворотним викликом і головним циклом обробки подій.

Ініціалізація вікна складається з настроювання відповідних буферів кадру, початкового положення й розмірів вікна, а також заголовка вікна.

Функція `glutInit(&argc, argv)` робить початкову ініціалізацію самої бібліотеки GLUT.

Команда `glutInitDisplayMode(GLUT_RGB)` – ініціалізує буфер кадру й настроює повнокольоровий режим RGB.

`glutInitWindowSize(Width, Height)` використовується для задавання початкових розмірів вікна.

Нарешті, `glutCreateWindow("Red square example")` задає заголовок вікна й візуалізує саме вікно на екрані.

Потім команди

```
glutDisplayFunc(Display);
glutReshapeFunc(Reshape);
glutKeyboardFunc(Keyboard);
```

реєструють функції `Display()`, `Reshape()` і `Keyboard()` як функції, які будуть викликані, відповідно, при перемальовуванні вікна, зміні розмірів вікна, натисканні клавіші на клавіатурі.


```
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB);
    glutInitWindowSize(Width, Height);
    glutCreateWindow("Red square example");
    glutDisplayFunc(Display);
    glutReshapeFunc(Reshape);
    glutKeyboardFunc(Keyboard);
    glutMainLoop();
}
```

Це повністю завершена програма, що повинна компілюватися й працювати на будь-якій системі, що підтримує OpenGL і GLUT.

Бібліотека GLUT підтримує взаємодію з користувачем за допомогою так званих функцій зі зворотним викликом (*callback function*). Якщо користувач зсунув мишу, натиснув на кнопку клавіатури або змінив розміри вікна, відбувається подія й викликається відповідна функція користувача – оброблювач подій (функція зі зворотним викликом).

Розглянемо більш детально наведений приклад.Функція `main` складається із трьох частин - ініціалізації вікна, у якому буде малювати OpenGL, настроювання функцій зі зворотним викликом і головним циклом обробки подій.

Ініціалізація вікна складається з настроювання відповідних буферів кадру, початкового положення й розмірів вікна, а також заголовка вікна.

Функція `glutInit(&argc, argv)` робить початкову ініціалізацію самої бібліотеки GLUT.

Команда `glutInitDisplayMode(GLUT_RGB)` – ініціалізує буфер кадру й настроює повнокольоровий режим RGB.

`glutInitWindowSize(Width, Height)` використовується для задавання початкових розмірів вікна.

Нарешті, `glutCreateWindow("Red square example")` задає заголовок вікна й візуалізує саме вікно на екрані.

Потім команди

```
glutDisplayFunc(Display);
glutReshapeFunc(Reshape);
glutKeyboardFunc(Keyboard);
```

реєструють функції `Display()`, `Reshape()` і `Keyboard()` як функції, які будуть викликані, відповідно, при перемальовуванні вікна, зміні розмірів вікна, натисканні клавіші на клавіатурі.

Контроль усіх подій і виклик потрібних функцій відбувається усередині нескінченного циклу у функції `glutMainLoop()`.

Відмітимо, що бібліотека GLUT не входить до складу OpenGL, а є лише прикладним прошарком між OpenGL і віконною підсистемою, надаючи мінімальний інтерфейс. OpenGL-додаток для конкретної платформи може бути написаний з використанням специфічних API функцій (Win32, X Window і т.д.), які, як правило надають більш широкі можливості.

Всі виклики команд OpenGL відбуваються в оброблювачах подій. Звернемо увагу на функцію Display, у якій зосереджений код, безпосередньо відповідальний за малювання на екрані.

Наступна послідовність команд із функції Display

```
glClearColor(0, 0, 0, 1);
glClear(GL_COLOR_BUFFER_BIT);
glColor3ub(255,0,0);
glBegin(GL_QUADS);
  glVertex2f(left,bottom);
  glVertex2f(left,top);
  glVertex2f(right,top);
  glVertex2f(right,bottom);
glEnd();
```

очищає вікно й виводить на екран квадрат, задаючи координати чотирьох кутових вершин і кольори.

6.2. Створення та перетворення графічних об'єктів

Процес відновлення зображення

Як правило, завданням програми, що використовує OpenGL, є обробка тривимірної сцени й інтерактивне відображення в буфері кадру. Сцена складається з набору тривимірних об'єктів, джерел світла й віртуальної камери, що визначає поточне положення спостерігача.

Звичайний додаток OpenGL у нескінченному циклі викликає функцію відновлення зображення у вікні. У цій функції й зосереджені виклики основних команд OpenGL. Якщо використовується бібліотека GLUT, то це буде функція зі зворотним викликом, зареєстрована за допомогою виклику `glutDisplayFunc()`.GLUT викликає цю функцію, коли операційна система інформує додаток про те, що вміст вікна необхідно перемалювати (наприклад, якщо вікно було перекрито іншим).

Контроль усіх подій і виклик потрібних функцій відбувається усередині нескінченного циклу у функції `glutMainLoop()`.

Відмітимо, що бібліотека GLUT не входить до складу OpenGL, а є лише прикладним прошарком між OpenGL і віконною підсистемою, надаючи мінімальний інтерфейс. OpenGL-додаток для конкретної платформи може бути написаний з використанням специфічних API функцій (Win32, X Window і т.д.), які, як правило надають більш широкі можливості.

Всі виклики команд OpenGL відбуваються в оброблювачах подій. Звернемо увагу на функцію Display, у якій зосереджений код, безпосередньо відповідальний за малювання на екрані.

Наступна послідовність команд із функції Display

```
glClearColor(0, 0, 0, 1);
glClear(GL_COLOR_BUFFER_BIT);
glColor3ub(255,0,0);
glBegin(GL_QUADS);
  glVertex2f(left,bottom);
  glVertex2f(left,top);
  glVertex2f(right,top);
  glVertex2f(right,bottom);
glEnd();
```

очищає вікно й виводить на екран квадрат, задаючи координати чотирьох кутових вершин і кольори.

6.2. Створення та перетворення графічних об'єктів

Процес відновлення зображення

Як правило, завданням програми, що використовує OpenGL, є обробка тривимірної сцени й інтерактивне відображення в буфері кадру. Сцена складається з набору тривимірних об'єктів, джерел світла й віртуальної камери, що визначає поточне положення спостерігача.

Звичайний додаток OpenGL у нескінченному циклі викликає функцію відновлення зображення у вікні. У цій функції й зосереджені виклики основних команд OpenGL. Якщо використовується бібліотека GLUT, то це буде функція зі зворотним викликом, зареєстрована за допомогою виклику `glutDisplayFunc()`.GLUT викликає цю функцію, коли операційна система інформує додаток про те, що вміст вікна необхідно перемалювати (наприклад, якщо вікно було перекрито іншим).

Створюване зображення може бути як статичним, так і анімаційним, тобто залежати від яких-небудь параметрів, що змінюються згодом. У цьому випадку краще викликати функцію відновлення самостійно. Наприклад, за допомогою команди `glutPostRedisplay()`.

Приступимо, нарешті, до того, чим займається типова функція відновлення зображення. Як правило, вона складається із трьох кроків:

– очищення буферів OpenGL;
– встановлення положення спостерігача;
– перетворення й малювання геометричних об'єктів.

Очищення буферів проводиться за допомогою команди:

```
  void glClearColor ( clampfr, clampfg, clampfb,
clampfa )
  void glClear(bitfield buf)
```

Команда `glClearColor` встановлює колір, яким буде заповнений буфер кадру. Перші три параметри команди задають R,G і B компоненти кольору і повинні належати відрізку [0,1]. Четвертий параметр задає так звану альфа-компоненту. Як правило, він дорівнює 1. За замовчуванням колір - чорний (0,0,0,1).

Команда `glClear` очищає буфери, а параметр *buf* визначає комбінацію констант, що відповідає буферам, які потрібно очистити. Типова програма викликає команду

```
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
```

для очищення буферів кольору й глибини.

Встановлення положення спостерігача й перетворення тривимірних об'єктів (поворот, зрушення й т.д.) контролюються за допомогою задавання матриць перетворення. Перетворення об'єктів і настроювання положення віртуальної камери описані далі.

Зараз зосередимося на тому, як передати в OpenGL опис об'єктів, що обробляються. Кожний об'єкт є набором примітивів OpenGL.

Створюване зображення може бути як статичним, так і анімаційним, тобто залежати від яких-небудь параметрів, що змінюються згодом. У цьому випадку краще викликати функцію відновлення самостійно. Наприклад, за допомогою команди `glutPostRedisplay()`.

Приступимо, нарешті, до того, чим займається типова функція відновлення зображення. Як правило, вона складається із трьох кроків:

– очищення буферів OpenGL;
– встановлення положення спостерігача;
– перетворення й малювання геометричних об'єктів.

Очищення буферів проводиться за допомогою команди:

```
  void glClearColor ( clampfr, clampfg, clampfb,
clampfa )
  void glClear(bitfield buf)
```

Команда `glClearColor` встановлює колір, яким буде заповнений буфер кадру. Перші три параметри команди задають R,G і B компоненти кольору і повинні належати відрізку [0,1]. Четвертий параметр задає так звану альфа-компоненту. Як правило, він дорівнює 1. За замовчуванням колір - чорний (0,0,0,1).

Команда `glClear` очищає буфери, а параметр *buf* визначає комбінацію констант, що відповідає буферам, які потрібно очистити. Типова програма викликає команду

```
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
```

для очищення буферів кольору й глибини.

Встановлення положення спостерігача й перетворення тривимірних об'єктів (поворот, зрушення й т.д.) контролюються за допомогою задавання матриць перетворення. Перетворення об'єктів і настроювання положення віртуальної камери описані далі.

Зараз зосередимося на тому, як передати в OpenGL опис об'єктів, що обробляються. Кожний об'єкт є набором примітивів OpenGL.

Вершини й примітиви

Вершина є атомарним графічним примітивом OpenGL і визначає точку, кінець відрізка, кут багатокутника й т.д. Всі інші примітиви формуються за допомогою задавання вершин, що входять у даний примітив. Наприклад, відрізок визначається двома вершинами, що є його кінцями.

З кожною вершиною асоціюються її *атрибути*. У число основних атрибутів входять положення вершини в просторі, колір вершини й вектор нормалі.

Положення вершини в просторі

Положення вершини визначається її координатами у дво-, три- або чотиривимірному просторі (однорідні координати). Це реалізується за допомогою декількох варіантів команди `glVertex*`:

```
void glVertex[2 3 4][s i f d] (type coords)
void glVertex[2 3 4][s i f d]v (type *coords)
```

Кожна команда задає чотири координати вершини: x, y, z, w. Команда `glVertex2*` отримує значення x і y. Координата z у такому випадку встановлюється за замовчуванням і дорівнює 0, координата w – 1. `Vertex3*` отримує координати x, y, z і заносить у координату w значення 1. `Vertex4*` дозволяє задати всі чотири координати.

Для асоціації з вершинами кольорів, нормалей і текстурних координат використовуються поточні значення відповідних даних, що відповідає організації OpenGL як кінцевого результату. Ці значення можуть бути змінені в будь-який момент за допомогою виклику відповідних команд.

Колір вершини

Для задавання поточного кольору вершини використовуються такі функції:

```
void glColor[3 4][b s i f] (GLtype components)
void glColor[3 4][b s i f]v (GLtype components)
```

Перші три параметри задають R, G, B компоненти кольору, а останній параметр визначає коефіцієнт непрозорості (так званий альфа-компонент). Якщо в назві команди зазначений тип 'f' (float),

Вершини й примітиви

Вершина є атомарним графічним примітивом OpenGL і визначає точку, кінець відрізка, кут багатокутника й т.д. Всі інші примітиви формуються за допомогою задавання вершин, що входять у даний примітив. Наприклад, відрізок визначається двома вершинами, що є його кінцями.

З кожною вершиною асоціюються її *атрибути*. У число основних атрибутів входять положення вершини в просторі, колір вершини й вектор нормалі.

Положення вершини в просторі

Положення вершини визначається її координатами у дво-, три- або чотиривимірному просторі (однорідні координати). Це реалізується за допомогою декількох варіантів команди `glVertex*`:

```
void glVertex[2 3 4][s i f d] (type coords)
void glVertex[2 3 4][s i f d]v (type *coords)
```

Кожна команда задає чотири координати вершини: x, y, z, w. Команда `glVertex2*` отримує значення x і y. Координата z у такому випадку встановлюється за замовчуванням і дорівнює 0, координата w – 1. `Vertex3*` отримує координати x, y, z і заносить у координату w значення 1. `Vertex4*` дозволяє задати всі чотири координати.

Для асоціації з вершинами кольорів, нормалей і текстурних координат використовуються поточні значення відповідних даних, що відповідає організації OpenGL як кінцевого результату. Ці значення можуть бути змінені в будь-який момент за допомогою виклику відповідних команд.

Колір вершини

Для задавання поточного кольору вершини використовуються такі функції:

```
void glColor[3 4][b s i f] (GLtype components)
void glColor[3 4][b s i f]v (GLtype components)
```

Перші три параметри задають R, G, B компоненти кольору, а останній параметр визначає коефіцієнт непрозорості (так званий альфа-компонент). Якщо в назві команди зазначений тип 'f' (float),

то значення всіх параметрів повинні належати відрізку [0,1], при цьому за замовчуванням значення альфа-компоненти встановлюється рівним 1.0, що відповідає повній непрозорості. Тип 'ub' (unsigned byte) має на увазі, що значення повинні лежати у відрізку [0, 255].

Вершинам можна призначати різні кольори, і, якщо включено відповідний режим, то буде проводитися лінійна інтерполяція кольорів по поверхні примітива.

Для керування режимом інтерполяції використовується команда

```
void glShadeModel (GLenum  mode),
```

виклик якої з параметром **GL_SMOOTH** включає інтерполяцію (встановлення за замовчуванням), а з **GL_FLAT** – відключає.

Нормаль

Визначити нормаль у вершині можна, використовуючи команди

```
void glNormal3[b s i f d] (type coords)
void glNormal3[b s i f d]v (type coords)
```

Для правильного розрахунку освітлення необхідно, щоб вектор нормалі мав одиничну довжину.

Командою `glEnable(GL_NORMALIZE)` можна викликати спеціальний режим, при якому задані нормалі, що будуть нормуватися автоматично.

Режим автоматичної нормалізації повинен бути включений, якщо додаток використовує модельні перетворення розтягання/стиснення, тому що в цьому випадку довжина нормалей змінюється при множенні на модельно-видову матрицю.

Однак застосування цього режиму зменшує швидкість роботи механізму візуалізації OpenGL, тому що нормалізація векторів має помітну обчислювальну складність (узяття квадратного кореня й т.п.). Тому краще відразу задавати одиничні нормалі.

Відзначимо, що команди

```
void glEnable (GLenum mode)
void glDisable (GLenum mode)
```


то значення всіх параметрів повинні належати відрізку [0,1], при цьому за замовчуванням значення альфа-компоненти встановлюється рівним 1.0, що відповідає повній непрозорості. Тип 'ub' (unsigned byte) має на увазі, що значення повинні лежати у відрізку [0, 255].

Вершинам можна призначати різні кольори, і, якщо включено відповідний режим, то буде проводитися лінійна інтерполяція кольорів по поверхні примітива.

Для керування режимом інтерполяції використовується команда

```
void glShadeModel (GLenum  mode),
```

виклик якої з параметром **GL_SMOOTH** включає інтерполяцію (встановлення за замовчуванням), а з **GL_FLAT** – відключає.

Нормаль

Визначити нормаль у вершині можна, використовуючи команди

```
void glNormal3[b s i f d] (type coords)
void glNormal3[b s i f d]v (type coords)
```

Для правильного розрахунку освітлення необхідно, щоб вектор нормалі мав одиничну довжину.

Командою `glEnable(GL_NORMALIZE)` можна викликати спеціальний режим, при якому задані нормалі, що будуть нормуватися автоматично.

Режим автоматичної нормалізації повинен бути включений, якщо додаток використовує модельні перетворення розтягання/стиснення, тому що в цьому випадку довжина нормалей змінюється при множенні на модельно-видову матрицю.

Однак застосування цього режиму зменшує швидкість роботи механізму візуалізації OpenGL, тому що нормалізація векторів має помітну обчислювальну складність (узяття квадратного кореня й т.п.). Тому краще відразу задавати одиничні нормалі.

Відзначимо, що команди

```
void glEnable (GLenum mode)
void glDisable (GLenum mode)
```

роблять включення й відключення того або іншого режиму роботи конвеєра OpenGL. Ці команди застосовуються досить часто, і їхні можливі параметри будуть розглядатися в кожному конкретному випадку.

Операторні дужки glBegin / glEnd

Ми розглянули задавання атрибутів однієї вершини. Однак, щоб задати атрибути графічного примітива, одних координат вершин недостатньо. Ці вершини треба об'єднати в одне ціле, визначивши необхідні властивості. Для цього в OpenGL використовуються так звані операторні дужки, що є викликами спеціальних команд OpenGL Визначення примітива або послідовності примітивів відбувається між викликами команд

```
void glBegin (GLenum mode);
void glEnd (void);
```

Параметр *mode* визначає тип примітива, що задається усередині й може приймати такі значення:

GL_POINTS	кожна вершина задає координати деякої точки.
GL_LINES	кожна окрема пара вершин визначає відрізок; якщо задано непарне число вершин, то остання вершина ігнорується.
GL_LINE_STRIP	кожна наступна вершина задає відрізок разом з попередньою.
GL_LINE_LOOP	відмінність від попереднього примітива тільки в тому, що останній відрізок визначається останньою й першою вершинами, утворюючи замкнуту ламану.
GL_TRIANGLES	кожні окремі три вершини визначають трикутник; якщо задано не кратне трьом число вершин, то останні вершини ігноруються.
GL_TRIANGLE_STRIP	кожна наступна вершина задає трикутник разом із двома попередніми.

роблять включення й відключення того або іншого режиму роботи конвеєра OpenGL. Ці команди застосовуються досить часто, і їхні можливі параметри будуть розглядатися в кожному конкретному випадку.

Операторні дужки glBegin / glEnd

Ми розглянули задавання атрибутів однієї вершини. Однак, щоб задати атрибути графічного примітива, одних координат вершин недостатньо. Ці вершини треба об'єднати в одне ціле, визначивши необхідні властивості. Для цього в OpenGL використовуються так звані операторні дужки, що є викликами спеціальних команд OpenGL Визначення примітива або послідовності примітивів відбувається між викликами команд

```
void glBegin (GLenum mode);
void glEnd (void);
```

Параметр *mode* визначає тип примітива, що задається усередині й може приймати такі значення:

GL_POINTS	кожна вершина задає координати деякої точки.
GL_LINES	кожна окрема пара вершин визначає відрізок; якщо задано непарне число вершин, то остання вершина ігнорується.
GL_LINE_STRIP	кожна наступна вершина задає відрізок разом з попередньою.
GL_LINE_LOOP	відмінність від попереднього примітива тільки в тому, що останній відрізок визначається останньою й першою вершинами, утворюючи замкнуту ламану.
GL_TRIANGLES	кожні окремі три вершини визначають трикутник; якщо задано не кратне трьом число вершин, то останні вершини ігноруються.
GL_TRIANGLE_STRIP	кожна наступна вершина задає трикутник разом із двома попередніми.

GL_TRIANGLE_FAN	трикутники задаються першою вершиною й кожною наступною парою вершин (пари не перетинаються).
GL_QUADS	кожна окрема четвірка вершин визначає чотирикутник; якщо задано не кратне чотирьом число вершин, то останні вершини ігноруються.
GL_QUAD_STRIP	чотирикутник з номером n визначається вершинами з номерами 2n-1, 2n, 2n+2, 2n+1.
GL_POLYGON	послідовно задаються вершини опуклого багатокутника.

Наприклад, щоб намалювати трикутник з різними кольорами у вершинах, досить написати:

```
GLfloat BlueCol[3] = {0,0,1};

glBegin(GL_TRIANGLES);
    glColor3f(1.0, 0.0, 0.0);/* червоний */
    glVertex3f(0.0, 0.0, 0.0);
    glColor3ub(0,255,0); /* зелений */
    glVertex3f(1.0, 0.0, 0.0);
    glColor3fv(BlueCol); /* синій */
    glVertex3f(1.0, 1.0, 0.0);
glEnd();
```

Як правило, різні типи примітивів мають різну швидкість візуалізації на різних платформах. Для збільшення продуктивності переважно використовують примітиви, що вимагають меншої кількості інформації для передачі на сервер, такі примітиви як **GL_TRIANGLE_STRIP**, **GL_QUAD_STRIP.**

Окрім задавання самих багатокутників, можна визначити метод їхнього відображення на екрані.Однак спочатку треба визначити поняття лицьових і зворотних граней.

Під *гранню* розуміється одна із сторін багатокутника, і за замовчуванням лицьовою вважається та сторона, вершини якої обходяться проти годинникової стрілки. Напрямок обходу вершин лицьових граней можна змінити викликом команди

GL_TRIANGLE_FAN	трикутники задаються першою вершиною й кожною наступною парою вершин (пари не перетинаються).
GL_QUADS	кожна окрема четвірка вершин визначає чотирикутник; якщо задано не кратне чотирьом число вершин, то останні вершини ігноруються.
GL_QUAD_STRIP	чотирикутник з номером n визначається вершинами з номерами 2n-1, 2n, 2n+2, 2n+1.
GL_POLYGON	послідовно задаються вершини опуклого багатокутника.

Наприклад, щоб намалювати трикутник з різними кольорами у вершинах, досить написати:

```
GLfloat BlueCol[3] = {0,0,1};

glBegin(GL_TRIANGLES);
    glColor3f(1.0, 0.0, 0.0);/* червоний */
    glVertex3f(0.0, 0.0, 0.0);
    glColor3ub(0,255,0); /* зелений */
    glVertex3f(1.0, 0.0, 0.0);
    glColor3fv(BlueCol); /* синій */
    glVertex3f(1.0, 1.0, 0.0);
glEnd();
```

Як правило, різні типи примітивів мають різну швидкість візуалізації на різних платформах. Для збільшення продуктивності переважно використовують примітиви, що вимагають меншої кількості інформації для передачі на сервер, такі примітиви як **GL_TRIANGLE_STRIP**, **GL_QUAD_STRIP.**

Окрім задавання самих багатокутників, можна визначити метод їхнього відображення на екрані.Однак спочатку треба визначити поняття лицьових і зворотних граней.

Під *гранню* розуміється одна із сторін багатокутника, і за замовчуванням лицьовою вважається та сторона, вершини якої обходяться проти годинникової стрілки. Напрямок обходу вершин лицьових граней можна змінити викликом команди

```
void glFrontFace (GLenum mode)
```

зі значенням параметра *mode* рівним **GL_CW** (clockwise), а повернути значення за замовчуванням можна, указавши **GL_CCW** (counter-clockwise).

Щоб змінити метод відображення багатокутника, використовується команда

```
void glPolygonMode (GLenum face, Glenum mode).
```

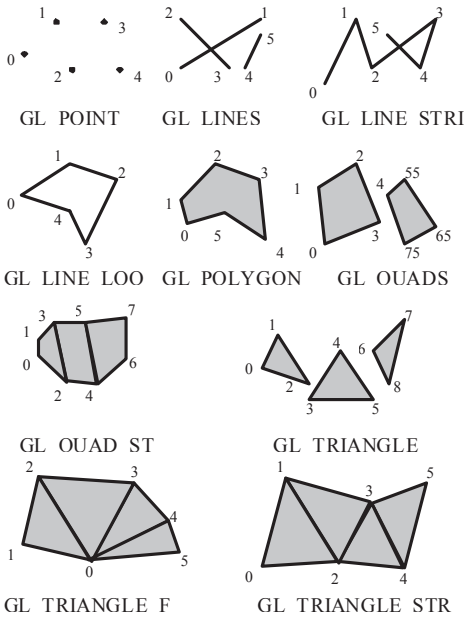


Рис. 6.3. Примітиви OpenGL

Параметр *mode* визначає, як будуть відображатися багатокутники, а параметр *face* встановлює тип багатокутників, до яких буде застосовуватися ця команда й може приймати такі значення:

GL_FRONT	для лицьових граней
GL_BACK	для зворотних граней


```
void glFrontFace (GLenum mode)
```

зі значенням параметра *mode* рівним **GL_CW** (clockwise), а повернути значення за замовчуванням можна, указавши **GL_CCW** (counter-clockwise).

Щоб змінити метод відображення багатокутника, використовується команда

```
void glPolygonMode (GLenum face, Glenum mode).
```

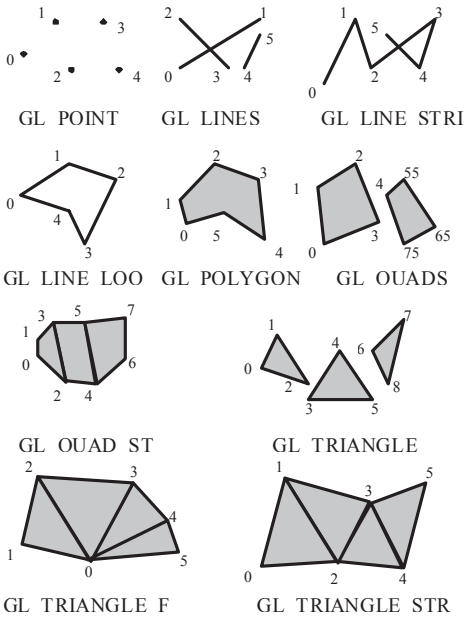



Рис. 6.3. Примітиви OpenGL

Параметр *mode* визначає, як будуть відображатися багатокутники, а параметр *face* встановлює тип багатокутників, до яких буде застосовуватися ця команда й може приймати такі значення:

GL_FRONT	для лицьових граней
GL_BACK	для зворотних граней

GL_FRONT_AND_BACK для всіх граней

Параметр *mode* може бути таким:

GL_POINT	відображення тільки вершин багатокутників.
GL_LINE	багатокутники будуть зображатися набором відрізків.
GL_FILL	багатокутники будуть зафарбовуватися поточними кольорами з урахуванням освітлення, і цей режим установлений за замовчуванням.

Також можна вказувати, який тип граней відображати на екрані. Для цього спочатку треба встановити відповідний режим викликом команди**glEnable** (GL_CULL_FACE), а потім вибрати тип відображуваних граней за допомогою команди

```
void glCullFace (GLenum mode)
```

Виклик параметра **GL_FRONT** приводить до видалення із зображення всіх лицьових граней, а параметра **GL_BACK** – зворотних (встановлення за замовчуванням).

Крім розглянутих стандартних примітивів у бібліотеках GLU і GLUT описані більш складні фігури, такі як сфера, циліндр, диск (в GLU) і сфера, куб, конус, тор, тетраедр, додекаедр, ікосаедр, октаедр і чайник (в GLUT).

Наприклад, щоб намалювати сферу або циліндр, треба спочатку створити об'єкт спеціального типу GLUquadricObj за допомогою команди

```
GLUquadricObj* gluNewQuadric(void);
```

а потім викликати відповідну команду:

```
void gluSphere (GLUquadricObj * qobj, GLdouble
radius,GLint slices, GLint stacks)

void gluCylinder (GLUquadricObj * qobj,
GLdouble baseRadius,
GLdouble topRadius,
```


GL_FRONT_AND_BACK для всіх граней

Параметр *mode* може бути таким:

GL_POINT	відображення тільки вершин багатокутників.
GL_LINE	багатокутники будуть зображатися набором відрізків.
GL_FILL	багатокутники будуть зафарбовуватися поточними кольорами з урахуванням освітлення, і цей режим установлений за замовчуванням.

Також можна вказувати, який тип граней відображати на екрані. Для цього спочатку треба встановити відповідний режим викликом команди**glEnable** (GL_CULL_FACE), а потім вибрати тип відображуваних граней за допомогою команди

```
void glCullFace (GLenum mode)
```

Виклик параметра **GL_FRONT** приводить до видалення із зображення всіх лицьових граней, а параметра **GL_BACK** – зворотних (встановлення за замовчуванням).

Крім розглянутих стандартних примітивів у бібліотеках GLU і GLUT описані більш складні фігури, такі як сфера, циліндр, диск (в GLU) і сфера, куб, конус, тор, тетраедр, додекаедр, ікосаедр, октаедр і чайник (в GLUT).

Наприклад, щоб намалювати сферу або циліндр, треба спочатку створити об'єкт спеціального типу GLUquadricObj за допомогою команди

```
GLUquadricObj* gluNewQuadric(void);
```

а потім викликати відповідну команду:

```
void gluSphere (GLUquadricObj * qobj, GLdouble
radius,GLint slices, GLint stacks)

void gluCylinder (GLUquadricObj * qobj,
GLdouble baseRadius,
GLdouble topRadius,
```

```
GLdouble height, GLint slices,
GLint stacks),
```
де параметр *slices* задає число розбивок навколо осі z, а *stacks* – уздовж осі z.

Дисплейні списки

Якщо ми кілька разів звертаємося до однієї й тієї ж групи команд, то їх можна об'єднати в так званий дисплейний список (display list) і викликати за необхідності. Для того щоб створити новий дисплейний список, треба помістити всі команди, які повинні до нього ввійти, між наступними операторними дужками:

```
void glNewList (GLuint list, GLenum mode)
void glEndList()
```

Для розрізнення списків використовуються цілі позитивні числа, що задають при створенні списку значенням параметра *list*, а параметр *mode* визначає режим обробки команд, що входять у список:

GL_COMPILE	команди записуються в список без виконання
GL_COMPILE_AND_EXECUTE	команди спочатку виконуються, а потім записуються в список

Після того, як список створений, його можна викликати командою

```
void glCallList (GLuint list),
```

вказавши в параметрі *list* ідентифікатор потрібного списку. Щоб викликати відразу кілька списків, можна скористатися командою

```
void glCallLists(GLsizei n, GLenum type, const GLvoid
                 *lists)
```

яка викликає *n* списки з ідентифікаторами з масиву *lists*, тип елементів якого вказується в параметрі *type*. Це можуть бути типи

GLdouble height, GLint slices,
GLint stacks),
де параметр *slices* задає число розбивок навколо осі z, а *stacks* – уздовж осі z.

Дисплейні списки

Якщо ми кілька разів звертаємося до однієї й тієї ж групи команд, то їх можна об'єднати в так званий дисплейний список (display list) і викликати за необхідності. Для того щоб створити новий дисплейний список, треба помістити всі команди, які повинні до нього ввійти, між наступними операторними дужками:

void glNewList (GLuint list, GLenum mode)
void glEndList()

Для розрізнення списків використовуються цілі позитивні числа, що задають при створенні списку значенням параметра *list*, а параметр *mode* визначає режим обробки команд, що входять у список:

GL_COMPILE	команди записуються в список без виконання
GL_COMPILE_AND_EXECUTE	команди спочатку виконуються, а потім записуються в список

Після того, як список створений, його можна викликати командою

void glCallList (GLuint list),

вказавши в параметрі *list* ідентифікатор потрібного списку. Щоб викликати відразу кілька списків, можна скористатися командою

void glCallLists(GLsizei n, GLenum type, const GLvoid
*lists)

яка викликає *n* списки з ідентифікаторами з масиву *lists*, тип елементів якого вказується в параметрі *type*. Це можуть бути типи

GL_BYTE, **GL_UNSIGNED_BYTE**, **GL_SHORT**, **GL_INT**, **GL_UNSIGNED_INT** і деякі інші. Для видалення списків використовується команда

```
void glDeleteLists(GLint list, GLsizei range),
```

яка видаляє списки з ідентифікаторами ID з діапазону *list* ≤ ID ≤ *list*+*range*-1.

Наприклад:

```
glNewList(1, GL_COMPILE);
  glBegin(GL_TRIANGLES);
    glVertex3f(1.0f, 1.0f, 1.0f);
    glVertex3f(10.0f, 1.0f, 1.0f);
    glVertex3f(10.0f, 10.0f, 1.0f);
  glEnd();
glEndList()

  …
glCallList(1);
```

Дисплейні списки в оптимальному, скомпільованому вигляді зберігаються в пам'яті сервера, що дозволяє малювати примітиви в такій формі максимально швидко. У той же час великі обсяги даних займають багато пам'яті, що тягне, у свою чергу, падіння продуктивності. Такі великі обсяги (більше декількох десятків тисяч примітивів) краще малювати за допомогою масивів вершин.

Масиви вершин

Якщо вершин багато, то, щоб не викликати для кожної команду `glVertex*()`, зручно поєднувати вершини в масиви, використовуючи команду

```
void glVertexPointer(GLint size, GLenum type,
                     GLsizei stride, void* ptr),
```

яка визначає спосіб зберігання й координати вершин. При цьому *size* визначає число координат вершини (може дорівнювати 2, 3, 4), type визначає тип даних (може дорівнювати **GL_SHORT**, **GL_INT**,

GL_BYTE, **GL_UNSIGNED_BYTE**, **GL_SHORT**, **GL_INT**, **GL_UNSIGNED_INT** і деякі інші. Для видалення списків використовується команда

```
void glDeleteLists(GLint list, GLsizei range),
```

яка видаляє списки з ідентифікаторами ID з діапазону *list* ≤ ID ≤ *list*+*range*-1.

Наприклад:

```
glNewList(1, GL_COMPILE);
  glBegin(GL_TRIANGLES);
    glVertex3f(1.0f, 1.0f, 1.0f);
    glVertex3f(10.0f, 1.0f, 1.0f);
    glVertex3f(10.0f, 10.0f, 1.0f);
  glEnd();
glEndList()

  …
glCallList(1);
```

Дисплейні списки в оптимальному, скомпільованому вигляді зберігаються в пам'яті сервера, що дозволяє малювати примітиви в такій формі максимально швидко. У той же час великі обсяги даних займають багато пам'яті, що тягне, у свою чергу, падіння продуктивності. Такі великі обсяги (більше декількох десятків тисяч примітивів) краще малювати за допомогою масивів вершин.

Масиви вершин

Якщо вершин багато, то, щоб не викликати для кожної команду `glVertex*()`, зручно поєднувати вершини в масиви, використовуючи команду

```
void glVertexPointer(GLint size, GLenum type,
                     GLsizei stride, void* ptr),
```

яка визначає спосіб зберігання й координати вершин. При цьому *size* визначає число координат вершини (може дорівнювати 2, 3, 4), type визначає тип даних (може дорівнювати **GL_SHORT**, **GL_INT**,

GL_FLOAT, **GL_DOUBLE**). Іноді зручно зберігати в одному масиві інші атрибути вершини, тоді параметр *stride* задає зсув від координати однієї вершини до координати наступної; якщо *stride* дорівнює нулю, це значить, що координати розташовані послідовно. У параметрі *ptr* вказується адреса, де перебувають дані.

Аналогічно можна визначити масив нормалей, кольорів і деяких інших атрибутів вершини, використовуючи команди

```
void glNormalPointer ( GLenum type, GLsizei stride,
                       void  *pointer )
void glColorPointer ( GLint size, GLenum type,
                      GLsizei stride, void *pointer )
```

Для того щоб ці масиви можна було використати надалі, треба викликати команду

```
void  glEnableClientState (GLenum array)
```

з параметрами **GL_VERTEX_ARRAY**, **GL_NORMAL_ARRAY**, **GL_COLOR_ARRAY** відповідно. Після закінчення роботи з масивом бажано викликати команду

```
void glDisableClientState (GLenum array)
```

з відповідним значенням параметра *array*. Для відображення вмісту масивів використовується команда

```
void glArrayElement (GLint index),
```

яка передає OpenGL атрибути вершини, використовуючи елементи масиву з номером *index*. Це аналогічно послідовному застосуванню команд виду `glColor*(...)`, `glNormal*(...)`, `glVertex*(...)` з відповідними параметрами. Однак зазвичай викликається команда

```
void glDrawArrays (GLenum mode, GLint first,
                   GLsizei count),
```

що малює *count* примітивів, обумовлених параметром *mode*, використовуючи елементи із масивів з індексами від *first* до *first*+*count*-1. Це еквівалентно виклику послідовності команд `glArrayElement()` з відповідними індексами.

GL_FLOAT, **GL_DOUBLE**). Іноді зручно зберігати в одному масиві інші атрибути вершини, тоді параметр *stride* задає зсув від координати однієї вершини до координати наступної; якщо *stride* дорівнює нулю, це значить, що координати розташовані послідовно. У параметрі *ptr* вказується адреса, де перебувають дані.

Аналогічно можна визначити масив нормалей, кольорів і деяких інших атрибутів вершини, використовуючи команди

```
void glNormalPointer ( GLenum type, GLsizei stride,
                       void  *pointer )
void glColorPointer ( GLint size, GLenum type,
                      GLsizei stride, void *pointer )
```

Для того щоб ці масиви можна було використати надалі, треба викликати команду

```
void  glEnableClientState (GLenum array)
```

з параметрами **GL_VERTEX_ARRAY**, **GL_NORMAL_ARRAY**, **GL_COLOR_ARRAY** відповідно. Після закінчення роботи з масивом бажано викликати команду

```
void glDisableClientState (GLenum array)
```

з відповідним значенням параметра *array*. Для відображення вмісту масивів використовується команда

```
void glArrayElement (GLint index),
```

яка передає OpenGL атрибути вершини, використовуючи елементи масиву з номером *index*. Це аналогічно послідовному застосуванню команд виду `glColor*(...)`, `glNormal*(...)`, `glVertex*(...)` з відповідними параметрами. Однак зазвичай викликається команда

```
void glDrawArrays (GLenum mode, GLint first,
                   GLsizei count),
```

що малює *count* примітивів, обумовлених параметром *mode*, використовуючи елементи із масивів з індексами від *first* до *first*+*count*-1. Це еквівалентно виклику послідовності команд `glArrayElement()` з відповідними індексами.

Важливо відзначити, що використання масивів вершин дозволяє оптимізувати передачу даних на сервер OpenGL і, як наслідок, підвищити швидкість малювання тривимірної сцени. Такий метод визначення примітивів є одним з найшвидших і добре підходить для візуалізації великих обсягів даних.

Перетворення об'єктів

В OpenGL використовуються як основні три системи координат: лівостороння, правобічна й віконна. Перші дві системи є тривимірними й відрізняються одна від іншої напрямком осі z: у правобічній вона спрямована на спостерігача, у лівосторонній - у глибину екрана. Вісь x спрямована вправо щодо спостерігача, вісь y – відповідно нагору.

Лівостороння система використовується для задавання значень параметрам команди `gluPerspective()`, `glOrtho()`, які будуть розглянуті далі. Правобічна система координат використовується у всіх інших випадках. Відображення тривимірної інформації відбувається у двовимірну *віконну* систему координат.

Отже, OpenGL дозволяє шляхом маніпуляцій з матрицями моделювати як праву, так і ліву системи координат. Але на даному етапі краще піти простим шляхом і запам'ятати: основною системою координат OpenGL є правобічна система.

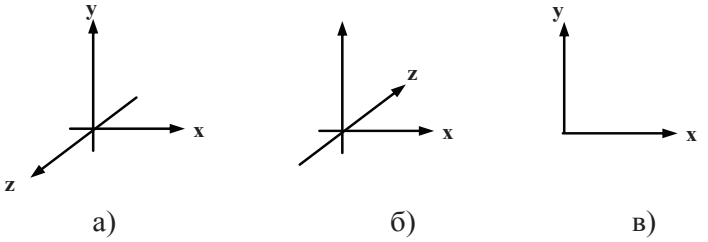


Рис. 6.4. Системи координат в OpenGL:
а – правобічна система; б – лівобічна система; в – віконна система.

Робота з матрицями

Для задавання різних перетворень об'єктів сцени в OpenGL використовуються операції над матрицями, при цьому розрізняють три

Важливо відзначити, що використання масивів вершин дозволяє оптимізувати передачу даних на сервер OpenGL і, як наслідок, підвищити швидкість малювання тривимірної сцени. Такий метод визначення примітивів є одним з найшвидших і добре підходить для візуалізації великих обсягів даних.

Перетворення об'єктів

В OpenGL використовуються як основні три системи координат: лівостороння, правобічна й віконна. Перші дві системи є тривимірними й відрізняються одна від іншої напрямком осі z: у правобічній вона спрямована на спостерігача, у лівосторонній - у глибину екрана. Вісь x спрямована вправо щодо спостерігача, вісь y – відповідно нагору.

Лівостороння система використовується для задавання значень параметрам команди `gluPerspective()`, `glOrtho()`, які будуть розглянуті далі. Правобічна система координат використовується у всіх інших випадках. Відображення тривимірної інформації відбувається у двовимірну *віконну* систему координат.

Отже, OpenGL дозволяє шляхом маніпуляцій з матрицями моделювати як праву, так і ліву системи координат. Але на даному етапі краще піти простим шляхом і запам'ятати: основною системою координат OpenGL є правобічна система.

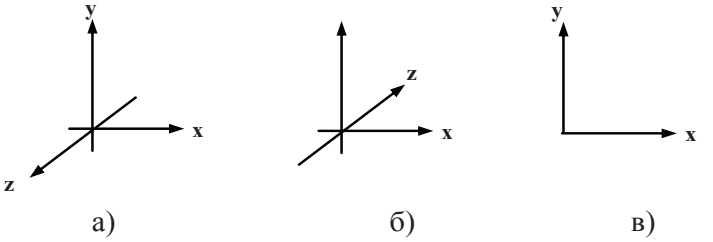


Рис. 6.4. Системи координат в OpenGL:
а – правобічна система; б – лівобічна система; в – віконна система.

Робота з матрицями

Для задавання різних перетворень об'єктів сцени в OpenGL використовуються операції над матрицями, при цьому розрізняють три

типи матриць: модельно-видова, матриця проекцій і матриця текстури. Всі вони мають розмір 4x4. Видова матриця визначає перетворення об'єкта у світових координатах, такі як паралельний перенос, зміна масштабу й поворот. Матриця проекцій визначає, як будуть проектуватися тривимірні об'єкти на площину екрана (у віконні координати), а матриця текстури визначає накладення текстури на об'єкт.

Множення координат на матриці відбувається в момент виклику відповідної команди OpenGL, що визначає координату (як правило, це команда `glVertex*`).

Для того щоб вибрати, яку матрицю треба змінити, використовується команда:

```
void glMatrixMode (GLenum mode),
```

виклик якої зі значенням параметра *mode*, що дорівнює **GL_MODELVIEW**, **GL_PROJECTION**, або **GL_TEXTURE,** включає режим роботи з модельно-видовою матрицею, матрицею проекцій або матрицею текстури відповідно. Для виклику команд, що задають матриці того або іншого типу, необхідно спочатку встановити відповідний режим.

Для визначення елементів матриці поточного типу викликається команда

```
void glLoadMatrix[f d] (GLtype *m),
```

де *m* вказує на масив з 16 елементів типу float або double відповідно до назви команди, при цьому спочатку в ньому повинен бути записаний перший стовпець матриці, потім другий, третій і четвертий. Ще раз зверніть увагу: у масиві *m* матриця записана *по стовпцях*.

Команда

```
void  glLoadIdentity (void)
```

заміняє поточну матрицю на одиничну.

Часто буває необхідно зберегти вміст поточної матриці для подальшого використання, для чого застосовуються команди

```
void glPushMatrix (void)
```


типи матриць: модельно-видова, матриця проекцій і матриця текстури. Всі вони мають розмір 4x4. Видова матриця визначає перетворення об'єкта у світових координатах, такі як паралельний перенос, зміна масштабу й поворот. Матриця проекцій визначає, як будуть проектуватися тривимірні об'єкти на площину екрана (у віконні координати), а матриця текстури визначає накладення текстури на об'єкт.

Множення координат на матриці відбувається в момент виклику відповідної команди OpenGL, що визначає координату (як правило, це команда `glVertex*`).

Для того щоб вибрати, яку матрицю треба змінити, використовується команда:

```
void glMatrixMode (GLenum mode),
```

виклик якої зі значенням параметра *mode*, що дорівнює **GL_MODELVIEW**, **GL_PROJECTION**, або **GL_TEXTURE,** включає режим роботи з модельно-видовою матрицею, матрицею проекцій або матрицею текстури відповідно. Для виклику команд, що задають матриці того або іншого типу, необхідно спочатку встановити відповідний режим.

Для визначення елементів матриці поточного типу викликається команда

```
void glLoadMatrix[f d] (GLtype *m),
```

де *m* вказує на масив з 16 елементів типу float або double відповідно до назви команди, при цьому спочатку в ньому повинен бути записаний перший стовпець матриці, потім другий, третій і четвертий. Ще раз зверніть увагу: у масиві *m* матриця записана *по стовпцях*.

Команда

```
void  glLoadIdentity (void)
```

заміняє поточну матрицю на одиничну.

Часто буває необхідно зберегти вміст поточної матриці для подальшого використання, для чого застосовуються команди

```
void glPushMatrix (void)
```

```
void glPopMatrix (void)
```

Вони записують і відновлюють поточну матрицю зі стека, причому для кожного типу матриць стек свій. Для модельно-видових матриць його глибина дорівнює як мінімум 32, для інших - як мінімум 2.

Для множення поточної матриці на іншу матрицю використовується команда

```
void glMultMatrix[f d] (GLtype *m),
```

де параметр *m* повинен задавати матрицю розміром 4x4.

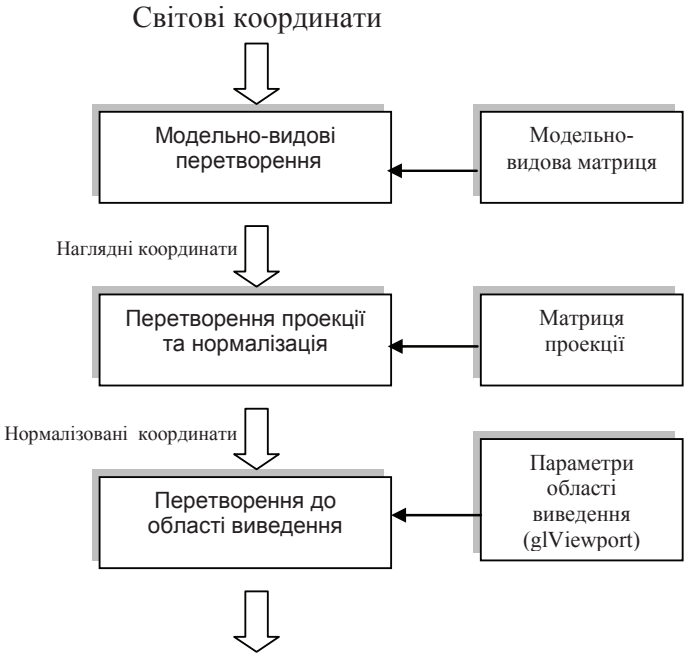


Рис. 6.5. Перетворення координат в OpenGL

Якщо позначити поточну матрицю за M, передану матрицю за T, то в результаті виконання команди `glMultMatrix` поточною стає матриця M * T. Однак звичайно для зміни матриці того або іншого


```
void glPopMatrix (void)
```

Вони записують і відновлюють поточну матрицю зі стека, причому для кожного типу матриць стек свій. Для модельно-видових матриць його глибина дорівнює як мінімум 32, для інших - як мінімум 2.

Для множення поточної матриці на іншу матрицю використовується команда

```
void glMultMatrix[f d] (GLtype *m),
```

де параметр *m* повинен задавати матрицю розміром 4x4.

Рис. 6.5. Перетворення координат в OpenGL

Якщо позначити поточну матрицю за M, передану матрицю за T, то в результаті виконання команди `glMultMatrix` поточною стає матриця M * T. Однак звичайно для зміни матриці того або іншого

типу зручно використовути спеціальні команди, які за значеннями своїх параметрів створюють потрібну матрицю й множать її на поточну.

У цілому, для відображення тривимірних об'єктів сцени у вікно додатка використовується послідовність, показана на рис.6.5.

Всі перетворення об'єктів і камери в OpenGL виводяться за допомогою множення векторів координат на матриці. Причому множення відбувається на *поточну матрицю* в момент визначення координати командою `glVertex*` і деякими іншими.

Модельно-видові перетворення

До модельно-видових перетворень будемо відносити перенесення, поворот і зміну масштабу уздовж координатних осей. Для проведення цих операцій досить помножити на відповідну матрицю кожну вершину об'єкта й отримати змінені координати цієї вершини:

$$(x', y', z', 1)^T = M(x, y, z, 1)^T,$$

де M – матриця модельно-видового перетворення. Перспективне перетворення й проектування виробляється аналогічно. Сама матриця може бути створена за допомогою таких команд:

```
void glTranslate[f d] (GLtype x, GLtype y, GLtype z)
void glRotate[f d] (GLtype angle, GLtype x, GLtype y,
                    GLtype z)
void glScale[f d] (GLtype x, GLtype y, GLtype z)
```

`glTranlsate*()` робить перенесення об'єкта, додаючи до координат його вершин значення своїх параметрів.

`glRotate*()` робить поворот об'єкта проти годинникової стрілки на кут *angle* (виміряється в градусах) навколо вектора (x,y,z).

`glScale*()` робить масштабування об'єкта (стиснення або розтягання) уздовж вектора (x,y,z), помноживши відповідні координати його вершин на значення своїх параметрів.

Всі ці перетворення змінюють поточну матрицю, а тому застосовуються до примітивів, які визначаються пізніше. У випадку, якщо треба, наприклад, повернути один об'єкт сцени, а інший залишити нерухомим, зручно спочатку зберегти поточну видову матрицю в стеці командою `glPushMatrix()`, потім викликати

типу зручно використовути спеціальні команди, які за значеннями своїх параметрів створюють потрібну матрицю й множать її на поточну.

У цілому, для відображення тривимірних об'єктів сцени у вікно додатка використовується послідовність, показана на рис.6.5.

Всі перетворення об'єктів і камери в OpenGL виводяться за допомогою множення векторів координат на матриці. Причому множення відбувається на *поточну матрицю* в момент визначення координати командою `glVertex*` і деякими іншими.

Модельно-видові перетворення

До модельно-видових перетворень будемо відносити перенесення, поворот і зміну масштабу уздовж координатних осей. Для проведення цих операцій досить помножити на відповідну матрицю кожну вершину об'єкта й отримати змінені координати цієї вершини:

$$(x', y', z', 1)^T = M(x, y, z, 1)^T,$$

де M – матриця модельно-видового перетворення. Перспективне перетворення й проектування виробляється аналогічно. Сама матриця може бути створена за допомогою таких команд:

```
void glTranslate[f d] (GLtype x, GLtype y, GLtype z)
void glRotate[f d] (GLtype angle, GLtype x, GLtype y,
                    GLtype z)
void glScale[f d] (GLtype x, GLtype y, GLtype z)
```

`glTranlsate*()` робить перенесення об'єкта, додаючи до координат його вершин значення своїх параметрів.

`glRotate*()` робить поворот об'єкта проти годинникової стрілки на кут *angle* (виміряється в градусах) навколо вектора (x,y,z).

`glScale*()` робить масштабування об'єкта (стиснення або розтягання) уздовж вектора (x,y,z), помноживши відповідні координати його вершин на значення своїх параметрів.

Всі ці перетворення змінюють поточну матрицю, а тому застосовуються до примітивів, які визначаються пізніше. У випадку, якщо треба, наприклад, повернути один об'єкт сцени, а інший залишити нерухомим, зручно спочатку зберегти поточну видову матрицю в стеці командою `glPushMatrix()`, потім викликати

glRotate*() з потрібними параметрами, описати примітиви, з яких складається цей об'єкт, а потім відновити поточну матрицю командою glPopMatrix(). Крім зміни положення самого об'єкта, часто буває необхідно змінити положення спостерігача. Це можна зробити за допомогою команди

```
void gluLookAt(GLdouble eyex, GLdouble eyey,
               GLdouble eyez, GLdouble centerx,
               GLdouble centery, GLdouble centerz,
               GLdouble upx, GLdouble upy,
               GLdouble upz),
```

де точка (*eyex,eyey,eyez*) визначає точку спостереження, (*centerx, centery, centerz*) задає центр сцени, що буде проектуватися в центр області виведення, а вектор (*upx,upy,upz*) задає позитивний напрямок осі, визначаючи поворот камери. Якщо, наприклад, камеру не треба повертати, то задається значення (0,1,0), а зі значенням (0,-1,0) сцена буде перевернена.

Власне кажучи, ця команда робить перенесення і поворот об'єктів сцени, але в такому виді задавати параметри буває зручніше. Слід зазначити, що викликати команду gluLookAt() має сенс *перед* визначенням перетворень об'єктів, коли модельно-видова матриця дорівнює одиничній.

Проекції

В OpenGL існують стандартні команди для задавання ортографічної (паралельної) і перспективної проекцій (рис. 6.6 і 6.7). Перший тип проекції може бути заданий командами

```
void glOrtho(GLdouble left, GLdouble right,
             GLdouble bottom, GLdouble top,
             GLdouble near, GLdouble far)

void gluOrtho2D (GLdouble left, GLdouble right,
                 GLdouble bottom, GLdouble top)
```

Перша команда створює матрицю проекції в усіченій області видимості (паралелепіпед видимості) у лівосторонній системі координат. Параметри команди задають точки (*left, bottom, znear*) і (*right, top, zfar*), які відповідають лівим нижнім і правому верхньому кутам

glRotate*() з потрібними параметрами, описати примітиви, з яких складається цей об'єкт, а потім відновити поточну матрицю командою glPopMatrix(). Крім зміни положення самого об'єкта, часто буває необхідно змінити положення спостерігача. Це можна зробити за допомогою команди

```
void gluLookAt(GLdouble eyex, GLdouble eyey,
               GLdouble eyez, GLdouble centerx,
               GLdouble centery, GLdouble centerz,
               GLdouble upx, GLdouble upy,
               GLdouble upz),
```

де точка (*eyex,eyey,eyez*) визначає точку спостереження, (*centerx, centery, centerz*) задає центр сцени, що буде проектуватися в центр області виведення, а вектор (*upx,upy,upz*) задає позитивний напрямок осі, визначаючи поворот камери. Якщо, наприклад, камеру не треба повертати, то задається значення (0,1,0), а зі значенням (0,-1,0) сцена буде перевернена.

Власне кажучи, ця команда робить перенесення і поворот об'єктів сцени, але в такому виді задавати параметри буває зручніше. Слід зазначити, що викликати команду gluLookAt() має сенс *перед* визначенням перетворень об'єктів, коли модельно-видова матриця дорівнює одиничній.

Проекції

В OpenGL існують стандартні команди для задавання ортографічної (паралельної) і перспективної проекцій (рис. 6.6 і 6.7). Перший тип проекції може бути заданий командами

```
void glOrtho(GLdouble left, GLdouble right,
             GLdouble bottom, GLdouble top,
             GLdouble near, GLdouble far)

void gluOrtho2D (GLdouble left, GLdouble right,
                 GLdouble bottom, GLdouble top)
```

Перша команда створює матрицю проекції в усіченій області видимості (паралелепіпед видимості) у лівосторонній системі координат. Параметри команди задають точки (*left, bottom, znear*) і (*right, top, zfar*), які відповідають лівим нижнім і правому верхньому кутам

вікна виведення. Параметри *near* і *far* задають відстань до ближньої й далекої площин відсікання по віддаленню від точки (0,0,0) і можуть бути негативними.

У другій команді, на відміну від точки, значення *near* і *far* встановлюються рівними –1 і 1 відповідно. Це зручно, якщо OpenGL використовується для малювання двовимірних об'єктів. У цьому випадку положення вершин можна задавати, використовуючи команди `glVertex2*()`.

Перспективна проекція визначається командою

```
void gluPerspective (GLdouble angley, GLdouble
                     aspect,GLdouble znear, GLdouble
                     zfar),
```

яка задає усічений конус видимості в лівосторонній системі координат.

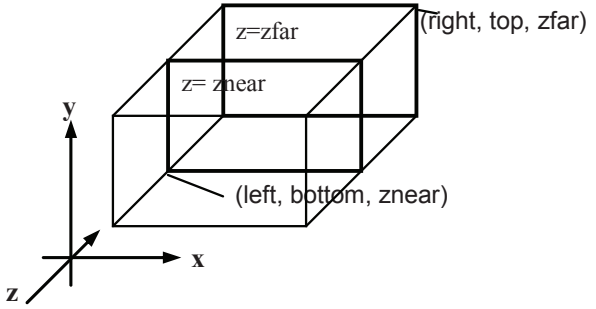
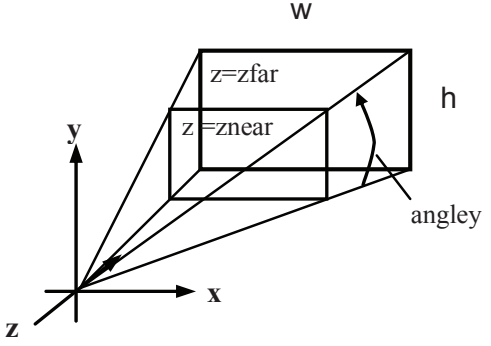


Рис.6.6. Ортографічна проекція



вікна виведення. Параметри *near* і *far* задають відстань до ближньої й далекої площин відсікання по віддаленню від точки (0,0,0) і можуть бути негативними.

У другій команді, на відміну від точки, значення *near* і *far* встановлюються рівними –1 і 1 відповідно. Це зручно, якщо OpenGL використовується для малювання двовимірних об'єктів. У цьому випадку положення вершин можна задавати, використовуючи команди `glVertex2*()`.

Перспективна проекція визначається командою

```
void gluPerspective (GLdouble angley, GLdouble
                     aspect,GLdouble znear, GLdouble
                     zfar),
```

яка задає усічений конус видимості в лівосторонній системі координат.

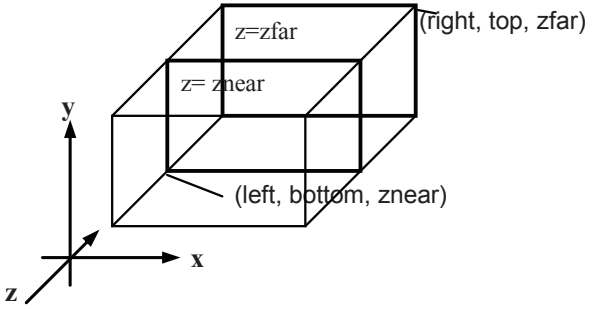


Рис.6.6. Ортографічна проекція

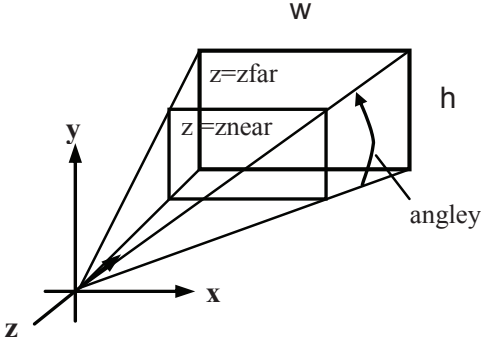


Рис.6.7. Перспективна проекція

Параметр *angley* визначає кут видимості в градусах по осі і повинен перебувати в діапазоні від 0 до 180. Кут видимості уздовж осі x задається параметром *aspect*, що звичайно задається як відношення сторін області виведення (як правило, розмірів вікна) Параметри *zfar* і *znear* задають відстань від спостерігача до площин відсікання по глибині й повинні бути позитивними. Чим більше відношення *zfar*/*znear*, тим гірше в буфері глибини будуть розрізнятися розташовані поруч поверхні, тому що за замовчуванням у нього буде записуватися ‘стисла’ глибина в діапазоні від 0 до 1.

Перш ніж задавати матриці проекцій, не забудьте включити режим роботи з потрібною матрицею командою `glMatrixMode GL_PROJECTION)` і скинути поточну, викликавши команду `glLoadIdentity()`.

Наприклад:

```
/* ортографічна проекція */
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(0, w, 0, h, -1.0, 1.0);
```

Область виведення

Після застосування матриці проекцій на вхід наступного перетворення подаються так звані усічені (clipped) координати. Потім визначаються нормалізовані координати вершин за формулою

$$(x_n, y_n, z_n)^T = (x_c/w_c , y_c/w_c, z_c/w_c)^T.$$

Область виведення являє собою прямокутник у віконній системі координат, розміри якого задаються командою:

```
void glViewPort (GLint x, GLint y, GLint width,
                 GLint height).
```

Значення всіх параметрів задаються в пікселах і визначають ширину й висоту області виведення з координатами лівого нижнього кута (*x*,*y*) у віконній системі координат. Розміри віконної системи координат визначаються поточними розмірами вікна додатка, точки (0,0) перебувають у лівому нижньому куті вікна. Використовуючи

Рис.6.7. Перспективна проекція

Параметр *angley* визначає кут видимості в градусах по осі і повинен перебувати в діапазоні від 0 до 180. Кут видимості уздовж осі x задається параметром *aspect*, що звичайно задається як відношення сторін області виведення (як правило, розмірів вікна) Параметри *zfar* і *znear* задають відстань від спостерігача до площин відсікання по глибині й повинні бути позитивними. Чим більше відношення *zfar*/*znear*, тим гірше в буфері глибини будуть розрізнятися розташовані поруч поверхні, тому що за замовчуванням у нього буде записуватися ‘стисла’ глибина в діапазоні від 0 до 1.

Перш ніж задавати матриці проекцій, не забудьте включити режим роботи з потрібною матрицею командою `glMatrixMode GL_PROJECTION)` і скинути поточну, викликавши команду `glLoadIdentity()`.

Наприклад:

```
/* ортографічна проекція */
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(0, w, 0, h, -1.0, 1.0);
```

Область виведення

Після застосування матриці проекцій на вхід наступного перетворення подаються так звані усічені (clipped) координати. Потім визначаються нормалізовані координати вершин за формулою

$$(x_n, y_n, z_n)^T = (x_c/w_c , y_c/w_c, z_c/w_c)^T.$$

Область виведення являє собою прямокутник у віконній системі координат, розміри якого задаються командою:

```
void glViewPort (GLint x, GLint y, GLint width,
                 GLint height).
```

Значення всіх параметрів задаються в пікселах і визначають ширину й висоту області виведення з координатами лівого нижнього кута (*x*,*y*) у віконній системі координат. Розміри віконної системи координат визначаються поточними розмірами вікна додатка, точки (0,0) перебувають у лівому нижньому куті вікна. Використовуючи

параметри команди `glViewPort()`, OpenGL обчислює віконні координати центра області виведення (o_x,o_y) за формулами:

$$o_x=x+width/2,\ o_y=y+height/2.$$

Нехай $p_x=width$, $p_y=height$, тоді можна знайти віконні координати кожної вершини:

$$(x_w, y_w, z_w)^T = ((p_x/2)\, x_n + o_x ,\ (p_y/2)\, y_n + o_y ,\ [(f-n)/2]\, z_n + (n+f)/2)^T.$$

При цьому цілі позитивні величини n і f задають мінімальну й максимальну глибину точки у вікні й за замовчуванням рівні 0 і 1 відповідно. Глибина кожної точки записується в спеціальний буфер глибини (z-буфер), що використовується для видалення невидимих ліній і поверхонь. Установити значення n і f можна викликом функції `void` **`glDepthRange`** `(GLclampd n, GLclampd f)`.

6.3. Моделювання освітлення та текстури

Матеріали й освітлення

Для створення реалістичних зображень необхідно визначити як властивості самого об'єкта, так і властивості середовища, в якому він перебуває. Перша група властивостей містить у собі параметри матеріалу, з якого зроблений об'єкт, способи нанесення текстури на його поверхню, ступінь прозорості об'єкта. До другої групи можна віднести кількість і властивості джерел світла, рівень прозорості середовища, а також модель освітлення. Всі ці властивості можна задавати, викликаючи відповідні команди OpenGL.

Модель освітлення

В OpenGL використовується модель освітлення, відповідно до якої кольори точки визначаються декількома факторами: властивостями матеріалу й текстури, величиною нормалі в цій точці, а також положенням джерела світла й спостерігача. Для коректного розрахунку освітленості в точці треба використати одиничні нормалі, однак команди типу `glScale*()`, можуть змінювати довжину нормалей.

Для завдання глобальних параметрів висвітлення використовуються команди

параметри команди `glViewPort()`, OpenGL обчислює віконні координати центра області виведення (o_x,o_y) за формулами:

$$o_x=x+width/2,\ o_y=y+height/2.$$

Нехай $p_x=width$, $p_y=height$, тоді можна знайти віконні координати кожної вершини:

$$(x_w, y_w, z_w)^T = ((p_x/2)\, x_n + o_x ,\ (p_y/2)\, y_n + o_y ,\ [(f-n)/2]\, z_n + (n+f)/2)^T.$$

При цьому цілі позитивні величини n і f задають мінімальну й максимальну глибину точки у вікні й за замовчуванням рівні 0 і 1 відповідно. Глибина кожної точки записується в спеціальний буфер глибини (z-буфер), що використовується для видалення невидимих ліній і поверхонь. Установити значення n і f можна викликом функції `void` **`glDepthRange`** `(GLclampd n, GLclampd f)`.

6.3. Моделювання освітлення та текстури

Матеріали й освітлення

Для створення реалістичних зображень необхідно визначити як властивості самого об'єкта, так і властивості середовища, в якому він перебуває. Перша група властивостей містить у собі параметри матеріалу, з якого зроблений об'єкт, способи нанесення текстури на його поверхню, ступінь прозорості об'єкта. До другої групи можна віднести кількість і властивості джерел світла, рівень прозорості середовища, а також модель освітлення. Всі ці властивості можна задавати, викликаючи відповідні команди OpenGL.

Модель освітлення

В OpenGL використовується модель освітлення, відповідно до якої кольори точки визначаються декількома факторами: властивостями матеріалу й текстури, величиною нормалі в цій точці, а також положенням джерела світла й спостерігача. Для коректного розрахунку освітленості в точці треба використати одиничні нормалі, однак команди типу `glScale*()`, можуть змінювати довжину нормалей.

Для завдання глобальних параметрів висвітлення використовуються команди

```
void glLightModel[i f] (GLenum pname, GLenum param)
void glLightModel[i f]v (GLenum pname,
  const GLtype *params)
```

Аргумент *pname* визначає, який параметр моделі освітлення буде настроюватися й може приймати такі значення:

GL_LIGHT_MODEL_LOCAL_VIEWER – параметр *param* повинен бути булевим і задавати положення спостерігача. Якщо він дорівнює **GL_FALSE**, то напрямок огляду вважається паралельним осі –z, поза залежністю від положення у видових координатах. Якщо ж він дорівнює **GL_TRUE**, то спостерігач перебуває на початку видової системи координат. Це може поліпшити якість освітлення, але ускладнює його розрахунок. Значення за замовчуванням: **GL_FALSE**.

GL_LIGHT_MODEL_TWO_SIDE – параметр *param* повинен бути булевим і керувати режимом розрахунку освітленості як для лицьових, так і для зворотних граней. Якщо він дорівнює **GL_FALSE**, то освітленість розраховується тільки для лицьових граней. Якщо ж він дорівнює **GL_TRUE**, розрахунок проводиться й для зворотних граней. Значення за замовчуванням: **GL_FALSE**.

GL_LIGHT_MODEL_AMBIENT – параметр *params* повинен містити чотири цілих або дійсних чисел, які визначають кольори фонового освітлення навіть у випадку відсутності певних джерел світла. Значення за замовчуванням: (0.2, 0.2, 0.2,1.0).

Специфікація матеріалів

Для задавання параметрів поточного матеріалу використовуються команди

```
void glMaterial[i f] (GLenum face, GLenum pname,
                      GLtype param)
void glMaterial[i f]v (GLenum face, GLenum pname,
                       GLtype *params)
```



```
void glLightModel[i f] (GLenum pname, GLenum param)
void glLightModel[i f]v (GLenum pname,
  const GLtype *params)
```

Аргумент *pname* визначає, який параметр моделі освітлення буде настроюватися й може приймати такі значення:

GL_LIGHT_MODEL_LOCAL_VIEWER – параметр *param* повинен бути булевим і задавати положення спостерігача. Якщо він дорівнює **GL_FALSE**, то напрямок огляду вважається паралельним осі –z, поза залежністю від положення у видових координатах. Якщо ж він дорівнює **GL_TRUE**, то спостерігач перебуває на початку видової системи координат. Це може поліпшити якість освітлення, але ускладнює його розрахунок. Значення за замовчуванням: **GL_FALSE**.

GL_LIGHT_MODEL_TWO_SIDE – параметр *param* повинен бути булевим і керувати режимом розрахунку освітленості як для лицьових, так і для зворотних граней. Якщо він дорівнює **GL_FALSE**, то освітленість розраховується тільки для лицьових граней. Якщо ж він дорівнює **GL_TRUE**, розрахунок проводиться й для зворотних граней. Значення за замовчуванням: **GL_FALSE**.

GL_LIGHT_MODEL_AMBIENT – параметр *params* повинен містити чотири цілих або дійсних чисел, які визначають кольори фонового освітлення навіть у випадку відсутності певних джерел світла. Значення за замовчуванням: (0.2, 0.2, 0.2,1.0).

Специфікація матеріалів

Для задавання параметрів поточного матеріалу використовуються команди

```
void glMaterial[i f] (GLenum face, GLenum pname,
                      GLtype param)
void glMaterial[i f]v (GLenum face, GLenum pname,
                       GLtype *params)
```

З їхньою допомогою можна визначити розсіяний, дифузійний і дзеркальний кольори матеріалу, а також ступінь дзеркального відбиття й інтенсивність випромінювання світла, якщо об'єкт повинен світитися. Який саме параметр буде визначатися значенням *param*, залежить від значення *pname*:

GL_AMBIENT – параметр *params* повинен містити чотири цілих або дійсних значення кольорів RGBA, які визначають розсіяні кольори матеріалу (кольори матеріалу в тіні). Значення за замовчуванням: (0.2, 0.2, 0.2, 1.0).

GL_DIFFUSE – параметр *params* повинен містити чотири цілих або дійсних значення кольорів RGBA, які визначають дифузійний колір матеріалу. Значення за замовчуванням: (0.8, 0.8, 0.8, 1.0).

GL_SPECULAR – параметр *params* повинен містити чотири цілих або дійсних кольорів RGBA, які визначають дзеркальний колір матеріалу. Значення за замовчуванням: (0.0, 0.0, 0.0, 1.0).

GL_SHININESS – параметр *params* повинен містити одне ціле або дійсне значення в діапазоні від 0 до 128, що визначає ступінь дзеркального відбиття матеріалу. Значення за замовчуванням: 0.

GL_EMISSION – параметр *params* повинен містити чотири цілих або дійсних значень кольорів RGBA, які визначають інтенсивність випромінюваного світла матеріалу. Значення за замовчуванням: (0.0, 0.0, 0.0, 1.0).

GL_AMBIENT_AND_DIFFUSE – еквівалентно двом викликам команди `glMaterial*()` зі значенням pname **GL_AMBIENT** і **GL_DIFFUSE** і однаковими значеннями *params*.

Із цього випливає, що виклик команди `glMaterial[i f]()` можливий тільки для установки ступеня дзеркального відбиття матеріалу (shininess). Команда `glMaterial[i f]v()` використовується для задавання інших параметрів.

З їхньою допомогою можна визначити розсіяний, дифузійний і дзеркальний кольори матеріалу, а також ступінь дзеркального відбиття й інтенсивність випромінювання світла, якщо об'єкт повинен світитися. Який саме параметр буде визначатися значенням *param*, залежить від значення *pname*:

GL_AMBIENT – параметр *params* повинен містити чотири цілих або дійсних значення кольорів RGBA, які визначають розсіяні кольори матеріалу (кольори матеріалу в тіні). Значення за замовчуванням: (0.2, 0.2, 0.2, 1.0).

GL_DIFFUSE – параметр *params* повинен містити чотири цілих або дійсних значення кольорів RGBA, які визначають дифузійний колір матеріалу. Значення за замовчуванням: (0.8, 0.8, 0.8, 1.0).

GL_SPECULAR – параметр *params* повинен містити чотири цілих або дійсних кольорів RGBA, які визначають дзеркальний колір матеріалу. Значення за замовчуванням: (0.0, 0.0, 0.0, 1.0).

GL_SHININESS – параметр *params* повинен містити одне ціле або дійсне значення в діапазоні від 0 до 128, що визначає ступінь дзеркального відбиття матеріалу. Значення за замовчуванням: 0.

GL_EMISSION – параметр *params* повинен містити чотири цілих або дійсних значень кольорів RGBA, які визначають інтенсивність випромінюваного світла матеріалу. Значення за замовчуванням: (0.0, 0.0, 0.0, 1.0).

GL_AMBIENT_AND_DIFFUSE – еквівалентно двом викликам команди `glMaterial*()` зі значенням pname **GL_AMBIENT** і **GL_DIFFUSE** і однаковими значеннями *params*.

Із цього випливає, що виклик команди `glMaterial[i f]()` можливий тільки для установки ступеня дзеркального відбиття матеріалу (shininess). Команда `glMaterial[i f]v()` використовується для задавання інших параметрів.

Параметр *face* визначає тип граней, для яких задається цей матеріал, і може приймати значення **GL_FRONT** або **GL_BACK**.

Якщо в сцені матеріали об'єктів розрізняються лише одним параметром, рекомендується спочатку встановити потрібний режим, викликавши `glEnable()` з параметром **GL_COLOR_MATE-RIAL**, а потім використати команду

```
void glColorMaterial(GLenum face, GLenum pname),
```

де параметр *face* має аналогічний сенс, а параметр *pname* може приймати всі перераховані значення. Після цього значення обраного за допомогою *pname* властивості матеріалу для конкретного об'єкта (або вершини) установлюються викликом команди `glColor*()`, що дозволяє уникнути викликів більш ресурсномісткої команди `glMaterial*()` і підвищує ефективність програми.

Приклад визначення властивостей матеріалу:

```
float mat_dif[]={0.8,0.8,0.8};
float mat_amb[] = {0.2, 0.2, 0.2};
float mat_spec[] = {0.6, 0.6, 0.6};
float shininess = 0.7 * 128;
…
glMaterialfv (GL_FRONT_AND_BACK,GL_AMBIENT, mat_amb);
glMaterialfv (GL_FRONT_AND_BACK,GL_DIFFUSE, mat_dif);
glMaterialfv (GL_FRONT_AND_BACK,GL_SPECULAR, mat_spec);
glMaterialf  (GL_FRONT,GL_SHININESS, shininess);
```

Опис джерел світла

Визначення властивостей матеріалу об'єкта має сенс, тільки якщо в сцені є джерела світла. Інакше всі об'єкти будуть чорними (або будуть мати колір, рівний розсіяному кольору матеріалу). Додати в сцену джерело світла можна за допомогою команд

```
void glLight[i f] (GLenum light, GLenum pname,
                   GLfloat param)
void glLight[i f] (GLenum light, GLenum pname,
                   GLfloat *params)
```

Параметр *light* однозначно визначає джерело світла. Він вибирається з набору спеціальних символічних імен виду **GL_LIGHTi** і

Параметр *face* визначає тип граней, для яких задається цей матеріал, і може приймати значення **GL_FRONT** або **GL_BACK**.

Якщо в сцені матеріали об'єктів розрізняються лише одним параметром, рекомендується спочатку встановити потрібний режим, викликавши `glEnable()` з параметром **GL_COLOR_MATE-RIAL**, а потім використати команду

```
void glColorMaterial(GLenum face, GLenum pname),
```

де параметр *face* має аналогічний сенс, а параметр *pname* може приймати всі перераховані значення. Після цього значення обраного за допомогою *pname* властивості матеріалу для конкретного об'єкта (або вершини) установлюються викликом команди `glColor*()`, що дозволяє уникнути викликів більш ресурсномісткої команди `glMaterial*()` і підвищує ефективність програми.

Приклад визначення властивостей матеріалу:

```
float mat_dif[]={0.8,0.8,0.8};
float mat_amb[] = {0.2, 0.2, 0.2};
float mat_spec[] = {0.6, 0.6, 0.6};
float shininess = 0.7 * 128;
…
glMaterialfv (GL_FRONT_AND_BACK,GL_AMBIENT, mat_amb);
glMaterialfv (GL_FRONT_AND_BACK,GL_DIFFUSE, mat_dif);
glMaterialfv (GL_FRONT_AND_BACK,GL_SPECULAR, mat_spec);
glMaterialf  (GL_FRONT,GL_SHININESS, shininess);
```

Опис джерел світла

Визначення властивостей матеріалу об'єкта має сенс, тільки якщо в сцені є джерела світла. Інакше всі об'єкти будуть чорними (або будуть мати колір, рівний розсіяному кольору матеріалу). Додати в сцену джерело світла можна за допомогою команд

```
void glLight[i f] (GLenum light, GLenum pname,
                   GLfloat param)
void glLight[i f] (GLenum light, GLenum pname,
                   GLfloat *params)
```

Параметр *light* однозначно визначає джерело світла. Він вибирається з набору спеціальних символічних імен виду **GL_LIGHTi** і

повинний лежати в діапазоні від 0 до константи **GL_MAX_LIGHT**, що звичайно не перевершує восьми.

Параметри *pname* і *params* мають сенс, аналогічний команді `glMaterial*()`. Розглянемо значення параметра *pname*:

GL_SPOT_EXPONENT – параметр *param* повинен містити ціле або дійсне число від 0 до 128, що задає розподіл інтенсивності світла. Цей параметр описує рівень сфокусованого джерела світла. Значення за замовчуванням: 0 (розсіяне світло).

GL_SPOT_CUTOFF – параметр *param* повинен містити ціле або дійсне число між 0 і 90 або рівне 180, що визначає максимальний кут розкиду світла. Значення цього параметра є половина кута у вершині конусоподібного світлового потоку, створюваного джерелом. Значення за замовчуванням: 180 (розсіяне світло).

GL_AMBIENT – параметр *params* повинен містити чотири цілих або дійсних значення кольорів RGBA, які визначають кольори фонового освітлення. Значення за замовчуванням: (0.0, 0.0, 0.0, 1.0).

GL_DIFFUSE – параметр *params* повинен містити чотири цілих або дійсних значення кольорів RGBA, які визначають кольори дифузійного освітлення. Значення за замовчуванням: (1.0, 1.0, 1.0, 1.0) для **GL_LIGHT0** і (0.0, 0.0, 0.0, 1.0) для інших.

GL_SPECULAR – параметр *params* повинен містити чотири цілих або речовинних значення квітів RGBA, які визначають кольори дзеркального відбиття. Значення за замовчуванням: (1.0, 1.0, 1.0, 1.0) для **GL_LIGHT0** і (0.0, 0.0, 0.0, 1.0) для інших.

GL_POSITION – параметр *params* повинен містити чотири цілих або дійсних числа, які визначають положення джерела світла. Якщо значення компонента w дорівнює 0.0, то джерело вважається нескінченно віддаленим і при розрахунку освітленості враховуються тільки напрямок на точку (x,y,z), у протилежному

повинний лежати в діапазоні від 0 до константи **GL_MAX_LIGHT**, що звичайно не перевершує восьми.

Параметри *pname* і *params* мають сенс, аналогічний команді `glMaterial*()`. Розглянемо значення параметра *pname*:

GL_SPOT_EXPONENT – параметр *param* повинен містити ціле або дійсне число від 0 до 128, що задає розподіл інтенсивності світла. Цей параметр описує рівень сфокусованого джерела світла. Значення за замовчуванням: 0 (розсіяне світло).

GL_SPOT_CUTOFF – параметр *param* повинен містити ціле або дійсне число між 0 і 90 або рівне 180, що визначає максимальний кут розкиду світла. Значення цього параметра є половина кута у вершині конусоподібного світлового потоку, створюваного джерелом. Значення за замовчуванням: 180 (розсіяне світло).

GL_AMBIENT – параметр *params* повинен містити чотири цілих або дійсних значення кольорів RGBA, які визначають кольори фонового освітлення. Значення за замовчуванням: (0.0, 0.0, 0.0, 1.0).

GL_DIFFUSE – параметр *params* повинен містити чотири цілих або дійсних значення кольорів RGBA, які визначають кольори дифузійного освітлення. Значення за замовчуванням: (1.0, 1.0, 1.0, 1.0) для **GL_LIGHT0** і (0.0, 0.0, 0.0, 1.0) для інших.

GL_SPECULAR – параметр *params* повинен містити чотири цілих або речовинних значення квітів RGBA, які визначають кольори дзеркального відбиття. Значення за замовчуванням: (1.0, 1.0, 1.0, 1.0) для **GL_LIGHT0** і (0.0, 0.0, 0.0, 1.0) для інших.

GL_POSITION – параметр *params* повинен містити чотири цілих або дійсних числа, які визначають положення джерела світла. Якщо значення компонента w дорівнює 0.0, то джерело вважається нескінченно віддаленим і при розрахунку освітленості враховуються тільки напрямок на точку (x,y,z), у протилежному

випадку вважається, що джерело розташоване в точці (x,y,z,w). У першому випадку ослаблення світла при віддаленні від джерела не відбувається, тобто джерело вважається нескінченно віддаленим. Значення за замовчуванням: (0.0, 0.0, 1.0, 0.0).

GL_SPOT_DIRECTION – параметр *params* повинен містити чотири цілих або дійсних числа, які визначають напрямок світла. Значення за замовчуванням: (0.0, 0.0, -1.0, 1.0). Ця характеристика джерела має сенс, якщо значення **GL_SPOT_CUTOFF** відмінне від 180 (яке, до речі, задано за замовчуванням).

GL_CONSTANT_ATTENUATION,

GL_LINEAR_ATTENUATION,

GL_QUADRATIC_ATTENUATION – параметр *params* задає значення одного із трьох коефіцієнтів, що визначають ослаблення інтенсивності світла при віддаленні від джерела. Допускаються тільки позитивні значення. Якщо джерело не є спрямованим (див. **GL_POSITION**), то ослаблення зворотно пропорційне сумі:

$\mathbf{att}_{constant}+att_{linear}*d+ att_{quadratic} *d^2$,

де d – відстань між джерелом світла й освітлюваною ним вершиною, $\mathbf{att}_{constant}$, att_{linear} і $att_{quadratic}$ дорівнюють параметрам, заданим за допомогою констант **GL_CONSTANT_ATTENUATION**, **GL_LINEAR_ATTENUATION** і **GL_QUADRATIC_ATTENUATION** відповідно. За замовчуванням ці параметри задаються трійкою (1, 0, 0), і фактично ослаблення не відбувається.

При зміні положення джерела світла варто враховувати наступний факт: в OpenGL джерела світла є об'єктами, багато в чому такими ж, як багатокутники й точки. На них поширюється основне правило обробки координат в OpenGL - параметри, що описують положення в просторі, перетворяться поточною модельно-видовою

випадку вважається, що джерело розташоване в точці (x,y,z,w). У першому випадку ослаблення світла при віддаленні від джерела не відбувається, тобто джерело вважається нескінченно віддаленим. Значення за замовчуванням: (0.0, 0.0, 1.0, 0.0).

GL_SPOT_DIRECTION – параметр *params* повинен містити чотири цілих або дійсних числа, які визначають напрямок світла. Значення за замовчуванням: (0.0, 0.0, -1.0, 1.0). Ця характеристика джерела має сенс, якщо значення **GL_SPOT_CUTOFF** відмінне від 180 (яке, до речі, задано за замовчуванням).

GL_CONSTANT_ATTENUATION,

GL_LINEAR_ATTENUATION,

GL_QUADRATIC_ATTENUATION – параметр *params* задає значення одного із трьох коефіцієнтів, що визначають ослаблення інтенсивності світла при віддаленні від джерела. Допускаються тільки позитивні значення. Якщо джерело не є спрямованим (див. **GL_POSITION**), то ослаблення зворотно пропорційне сумі:

$\mathbf{att}_{constant}+att_{linear}*d+ att_{quadratic} *d^2$,

де d – відстань між джерелом світла й освітлюваною ним вершиною, $\mathbf{att}_{constant}$, att_{linear} і $att_{quadratic}$ дорівнюють параметрам, заданим за допомогою констант **GL_CONSTANT_ATTENUATION**, **GL_LINEAR_ATTENUATION** і **GL_QUADRATIC_ATTENUATION** відповідно. За замовчуванням ці параметри задаються трійкою (1, 0, 0), і фактично ослаблення не відбувається.

При зміні положення джерела світла варто враховувати наступний факт: в OpenGL джерела світла є об'єктами, багато в чому такими ж, як багатокутники й точки. На них поширюється основне правило обробки координат в OpenGL - параметри, що описують положення в просторі, перетворяться поточною модельно-видовою

матрицею в момент формування об'єкта, тобто в момент виклику відповідних команд OpenGL. Таким чином, формуючи джерело світла одночасно з об'єктом сцени або камерою, його можна прив'язати до цього об'єкта. Або, навпаки, сформувати стаціонарне джерело світла, що буде залишатися на місці, поки інші об'єкти переміщаються.

Загальне правило таке:

Якщо положення джерела світла задається командою `glLight*()` перед визначенням положення віртуальної камери (наприклад, командою `glLookAt()`), то буде вважатися, що координати (0,0,0) джерела перебувають у точці спостереження й, отже, положення джерела світла визначається щодо положення спостерігача. Якщо положення встановлюється між визначенням положення камери й перетвореннями модельно-видової матриці об'єкта, то воно фіксується, тобто в цьому випадку положення джерела світла задається у світових координатах.

Для використання висвітлення спочатку треба встановити відповідний режим викликом команди `glEnable(GL_LIGHTNING)`, а потім включити потрібне джерело відповідною командою, наприклад `glEnable (GL_LIGHTi)`.

Ще раз зверніть увагу на те, що при вимкненому освітленні колір вершини дорівнює поточному кольору, який задається командами `glColor*()`. При включеному освітленні колір вершини обчислюється виходячи з інформації про матеріал, нормалі й джерела світла.

Створення ефекту туману

На завершення розглянемо одну цікаву й часто використовувану можливість OpenGL - створення ефекту туману. Легке затуманення сцени створює реалістичний ефект, та може й сховати деякі артефакти, які з'являються, коли в сцені присутні віддалені об'єкти.

Туман в OpenGL реалізується шляхом зміни кольорів об'єктів у сцені залежно від їхньої глибини, тобто відстані до точки спостереження. Зміна кольорів відбувається або для вершин примітивів, або для кожного піксела на етапі растеризації залежно від реалізації OpenGL. Цим процесом можна частково управляти.

Для включення ефекту затуманення необхідно викликати команду **`glEnable(GL_FOG)`**.

матрицею в момент формування об'єкта, тобто в момент виклику відповідних команд OpenGL. Таким чином, формуючи джерело світла одночасно з об'єктом сцени або камерою, його можна прив'язати до цього об'єкта. Або, навпаки, сформувати стаціонарне джерело світла, що буде залишатися на місці, поки інші об'єкти переміщаються.

Загальне правило таке:

Якщо положення джерела світла задається командою `glLight*()` перед визначенням положення віртуальної камери (наприклад, командою `glLookAt()`), то буде вважатися, що координати (0,0,0) джерела перебувають у точці спостереження й, отже, положення джерела світла визначається щодо положення спостерігача. Якщо положення встановлюється між визначенням положення камери й перетвореннями модельно-видової матриці об'єкта, то воно фіксується, тобто в цьому випадку положення джерела світла задається у світових координатах.

Для використання висвітлення спочатку треба встановити відповідний режим викликом команди `glEnable(GL_LIGHTNING)`, а потім включити потрібне джерело відповідною командою, наприклад `glEnable (GL_LIGHTi)`.

Ще раз зверніть увагу на те, що при вимкненому освітленні колір вершини дорівнює поточному кольору, який задається командами `glColor*()`. При включеному освітленні колір вершини обчислюється виходячи з інформації про матеріал, нормалі й джерела світла.

Створення ефекту туману

На завершення розглянемо одну цікаву й часто використовувану можливість OpenGL - створення ефекту туману. Легке затуманення сцени створює реалістичний ефект, та може й сховати деякі артефакти, які з'являються, коли в сцені присутні віддалені об'єкти.

Туман в OpenGL реалізується шляхом зміни кольорів об'єктів у сцені залежно від їхньої глибини, тобто відстані до точки спостереження. Зміна кольорів відбувається або для вершин примітивів, або для кожного піксела на етапі растеризації залежно від реалізації OpenGL. Цим процесом можна частково управляти.

Для включення ефекту затуманення необхідно викликати команду **`glEnable(GL_FOG)`**.

Метод обчислення інтенсивності туману у вершині можна визначити за допомогою команд

```
void glFog[if] (enum pname, T param) ;
void glFog[if]v (enum pname, T params) ;
```

Аргумент *pname* може приймати такі значення:

GL_FOG_MODE аргумент *param* визначає формулу, по якій буде обчислюватися інтенсивність туману в точці.

У цьому випадку *param* може приймати значення:

GL_EXP інтенсивність обчислюється за формулою f=exp(-d*z)

GL_EXP2 інтенсивність обчислюється за формулою f=exp(-(d*z)$^{2)}$

GL_LINEAR обчислюється за формулою f=e-z/e-s,

де z - відстань від вершини, у якій обчислюється інтенсивність туману, до точки спостереження.

Коефіцієнти d,e,s задаються за допомогою таких значень аргументу *pname:*

GL_FOG_DENSITY визначає коефіцієнт d

GL_FOG_START визначає коефіцієнт s

GL_FOG_END визначає коефіцієнт e

Кольори туману задаються за допомогою аргументу *pname*, що дорівнює **GL_FOG_COLOR,** у цьому випадку *params* – покажчик на масив з 4-х компонент кольорів.

Приведемо приклад використання цього ефекту:

```
GLfloat FogColor[4]={0.5,0.5,0.5,1};
glEnable(GL_FOG);
glFogi(GL_FOG_MODE,GL_LINEAR);
glFogf(GL_FOG_START,20.0);
glFogf(GL_FOG_END,100.0);
glFogfv(GL_FOG_COLOR,FogColor);
```

Накладання тектстури

Під *текстурою* будемо розуміти деяке зображення, яке треба певним чином нанести на об'єкт, наприклад, для додання ілюзії рельєфності поверхні.

Метод обчислення інтенсивності туману у вершині можна визначити за допомогою команд

```
void glFog[if] (enum pname, T param) ;
void glFog[if]v (enum pname, T params) ;
```

Аргумент *pname* може приймати такі значення:

GL_FOG_MODE аргумент *param* визначає формулу, по якій буде обчислюватися інтенсивність туману в точці.

У цьому випадку *param* може приймати значення:

GL_EXP інтенсивність обчислюється за формулою f=exp(-d*z)

GL_EXP2 інтенсивність обчислюється за формулою f=exp(-(d*z)$^{2)}$

GL_LINEAR обчислюється за формулою f=e-z/e-s,

де z - відстань від вершини, у якій обчислюється інтенсивність туману, до точки спостереження.

Коефіцієнти d,e,s задаються за допомогою таких значень аргументу *pname:*

GL_FOG_DENSITY визначає коефіцієнт d

GL_FOG_START визначає коефіцієнт s

GL_FOG_END визначає коефіцієнт e

Кольори туману задаються за допомогою аргументу *pname*, що дорівнює **GL_FOG_COLOR,** у цьому випадку *params* – покажчик на масив з 4-х компонент кольорів.

Приведемо приклад використання цього ефекту:

```
GLfloat FogColor[4]={0.5,0.5,0.5,1};
glEnable(GL_FOG);
glFogi(GL_FOG_MODE,GL_LINEAR);
glFogf(GL_FOG_START,20.0);
glFogf(GL_FOG_END,100.0);
glFogfv(GL_FOG_COLOR,FogColor);
```

Накладання тектстури

Під *текстурою* будемо розуміти деяке зображення, яке треба певним чином нанести на об'єкт, наприклад, для додання ілюзії рельєфності поверхні.

Накладення текстури на поверхню об'єктів сцени підвищує її реалістичність, однак при цьому треба враховувати, що цей процес вимагає обчислювальних витрат, особливо якщо OpenGL не підтримується апаратно.

Для роботи з текстурою варто виконати таку послідовність дій:

1) вибрати зображення й перетворити його в інший формат;
2) передати зображення в OpenGL;
3) визначити, як текстура буде наноситися на об'єкт і як вона буде з ним взаємодіяти;
4) зв'язати текстуру з об'єктом.

Підготовка текстури

Для використання текстури необхідно спочатку завантажити в пам'ять потрібне зображення й передати його OpenGL.

Зчитування графічних даних з файла та їх перетворення можна проводити вручну. Можна також скористатися функцією, що входить до складу бібліотеки GLAUX (для її використання треба додатково підключити бібліотеку glaux.lib), що сама проводить необхідні операції. Це функція

```
AUX_RGBImageRec* auxDIBImageLoad (const char *file),
```

де *file* – назва файла з розширенням *.bmp або *.dib. Функція повертає покажчик на область пам'яті, де зберігаються перетворені дані.

У ході створення образу текстури в пам'яті варто враховувати такі вимоги:

По-перше, розміри текстури, як по горизонталі, так і по вертикалі, повинні являти собою ступені двійки. Ця вимога накладається для компактного розміщення текстури в текстурній пам'яті й сприяє її ефективному використанню. Працювати тільки з такими текстурами звичайно незручно, тому після завантаження їх треба перетворити. Зміну розмірів текстури можна провести за допомогою команди

```
void gluScaleImage(GLenum format, GLint widthin,
                   GL heightin, GLenum typein,
                   const void *datain,
                   GLint widthout,
                   GLint heightout, GLenum typeout,
                   void *dataout)
```


Накладення текстури на поверхню об'єктів сцени підвищує її реалістичність, однак при цьому треба враховувати, що цей процес вимагає обчислювальних витрат, особливо якщо OpenGL не підтримується апаратно.

Для роботи з текстурою варто виконати таку послідовність дій:

1) вибрати зображення й перетворити його в інший формат;
2) передати зображення в OpenGL;
3) визначити, як текстура буде наноситися на об'єкт і як вона буде з ним взаємодіяти;
4) зв'язати текстуру з об'єктом.

Підготовка текстури

Для використання текстури необхідно спочатку завантажити в пам'ять потрібне зображення й передати його OpenGL.

Зчитування графічних даних з файла та їх перетворення можна проводити вручну. Можна також скористатися функцією, що входить до складу бібліотеки GLAUX (для її використання треба додатково підключити бібліотеку glaux.lib), що сама проводить необхідні операції. Це функція

```
AUX_RGBImageRec* auxDIBImageLoad (const char *file),
```

де *file* – назва файла з розширенням *.bmp або *.dib. Функція повертає покажчик на область пам'яті, де зберігаються перетворені дані.

У ході створення образу текстури в пам'яті варто враховувати такі вимоги:

По-перше, розміри текстури, як по горизонталі, так і по вертикалі, повинні являти собою ступені двійки. Ця вимога накладається для компактного розміщення текстури в текстурній пам'яті й сприяє її ефективному використанню. Працювати тільки з такими текстурами звичайно незручно, тому після завантаження їх треба перетворити. Зміну розмірів текстури можна провести за допомогою команди

```
void gluScaleImage(GLenum format, GLint widthin,
                   GL heightin, GLenum typein,
                   const void *datain,
                   GLint widthout,
                   GLint heightout, GLenum typeout,
                   void *dataout)
```

Як значення параметра *format* звичайно використовується значення **GL_RGB** або **GL_RGBA**, що визначає формат зберігання інформації. Параметри *widthin*, *heightin*, *widhtout*, *heightout* визначають розміри вхідного й вихідного зображень, а за допомогою *typein* і *typeout* задається тип елементів масивів, розташованих за адресами *datain* і *dataout*. Як і звичайно, це може бути тип **GL_UNSIGNED_BYTE, GL_SHORT, GL_INT** і так далі. Результат своєї роботи функція заносить в область пам'яті, на яку вказує параметр *dataout*.

По-друге, треба передбачити випадок, коли об'єкт після растеризації виявляється за розмірами значно меншим нанесеної на нього текстури. Чим менше об'єкт, тим меншою повинна бути нанесена на нього текстура й тому вводиться поняття *рівнів деталізації текстури* (mipmap). Кожний рівень деталізації задає деяке зображення, що є, як правило, зменшеною у два рази копією оригіналу. Такий підхід дозволяє поліпшити якість нанесення текстури на об'єкт. Наприклад, для зображення розміром 2^mx2^n можна побудувати max(m,n)+1 зменшених зображень, що відповідають різним рівням деталізації.

Ці два етапи створення образу текстури у внутрішній пам'яті OpenGL можна провести за допомогою команди

```
void gluBuild2DMipmaps(GLenum target, GLint
components,
                       GLint width, GLint height,
                       GLenum format, GLenum type,
                       const void *data),
```

де параметр *target* повинен дорівнювати **GL_TEXTURE_2D.** Параметр *components* визначає кількість колірних компонентів текстури й може приймати такі основні значення:

GL_LUMINANCE	один компонент – яскравість. (текстура буде монохромною)
GL_RGB	червоний, синій, зелений
GL_RGBA	всі компоненти.

Як значення параметра *format* звичайно використовується значення **GL_RGB** або **GL_RGBA**, що визначає формат зберігання інформації. Параметри *widthin*, *heightin*, *widhtout*, *heightout* визначають розміри вхідного й вихідного зображень, а за допомогою *typein* і *typeout* задається тип елементів масивів, розташованих за адресами *datain* і *dataout*. Як і звичайно, це може бути тип **GL_UNSIGNED_BYTE, GL_SHORT, GL_INT** і так далі. Результат своєї роботи функція заносить в область пам'яті, на яку вказує параметр *dataout*.

По-друге, треба передбачити випадок, коли об'єкт після растеризації виявляється за розмірами значно меншим нанесеної на нього текстури. Чим менше об'єкт, тим меншою повинна бути нанесена на нього текстура й тому вводиться поняття *рівнів деталізації текстури* (mipmap). Кожний рівень деталізації задає деяке зображення, що є, як правило, зменшеною у два рази копією оригіналу. Такий підхід дозволяє поліпшити якість нанесення текстури на об'єкт. Наприклад, для зображення розміром 2^mx2^n можна побудувати max(m,n)+1 зменшених зображень, що відповідають різним рівням деталізації.

Ці два етапи створення образу текстури у внутрішній пам'яті OpenGL можна провести за допомогою команди

```
void gluBuild2DMipmaps(GLenum target, GLint
components,
                       GLint width, GLint height,
                       GLenum format, GLenum type,
                       const void *data),
```

де параметр *target* повинен дорівнювати **GL_TEXTURE_2D.** Параметр *components* визначає кількість колірних компонентів текстури й може приймати такі основні значення:

GL_LUMINANCE	один компонент – яскравість. (текстура буде монохромною)
GL_RGB	червоний, синій, зелений
GL_RGBA	всі компоненти.

Параметри *width*, *height*, *data* визначають розміри й розташування текстури відповідно, а *format* і *type* мають аналогічний сенс, що й у команді `gluScaleImage()`.

Після виконання цієї команди текстура копіюється у внутрішню пам'ять OpenGL, і тому пам'ять, займану вихідним зображенням, можна звільнити.

В OpenGL допускається використання одномірних текстур, однак це завжди треба вказувати, задаючи як значення *target* константу **GL_TEXTURE_1D**. Користь одномірних текстур сумнівна, тому не будемо зупинятися на цьому докладно.

У разі використання в сцені декількох текстур, в OpenGL застосовується підхід, що нагадує створення списків зображень (так звані *текстурні об'єкти*). Спочатку за допомогою команди

```
void glGenTextures(GLsizei n, GLuint* textures)
```

треба створити *n* ідентифікаторів текстур, які будуть записані в масив *textures*. Перед початком визначення властивостей чергової текстури варто зробити її поточною («прив'язати» текстуру), викликавши команду

```
void glBindTexture(GLenum target, GLuint texture),
```

де *target* може приймати значення **GL_TEXTURE_1D** або **GL_TEXTURE_2D**, а параметр *texture* повинен дорівнювати ідентифікатору тієї текстури, до якої будуть ставитися наступні команди. Для того щоб у процесі малювання зробити поточну текстуру з деяким ідентифікатором, досить знову викликати команду `glBindTexture()` з відповідним значенням *target* і *texture*. Таким чином, команда `glBindTexture()` включає режим створення текстури з ідентифікатором *texture*, якщо така текстура ще не створена, або режим її використання робить цю текстуру поточною.

Оскільки не всі апаратури можуть оперувати текстурами великого розміру, доцільно обмежити розміри текстури до 256x256 або 512x512 пікселів. Відзначимо, що використання невеликих текстур підвищує ефективність програми.

Параметри *width*, *height*, *data* визначають розміри й розташування текстури відповідно, а *format* і *type* мають аналогічний сенс, що й у команді `gluScaleImage()`.

Після виконання цієї команди текстура копіюється у внутрішню пам'ять OpenGL, і тому пам'ять, займану вихідним зображенням, можна звільнити.

В OpenGL допускається використання одномірних текстур, однак це завжди треба вказувати, задаючи як значення *target* константу **GL_TEXTURE_1D**. Користь одномірних текстур сумнівна, тому не будемо зупинятися на цьому докладно.

У разі використання в сцені декількох текстур, в OpenGL застосовується підхід, що нагадує створення списків зображень (так звані *текстурні об'єкти*). Спочатку за допомогою команди

```
void glGenTextures(GLsizei n, GLuint* textures)
```

треба створити *n* ідентифікаторів текстур, які будуть записані в масив *textures*. Перед початком визначення властивостей чергової текстури варто зробити її поточною («прив'язати» текстуру), викликавши команду

```
void glBindTexture(GLenum target, GLuint texture),
```

де *target* може приймати значення **GL_TEXTURE_1D** або **GL_TEXTURE_2D**, а параметр *texture* повинен дорівнювати ідентифікатору тієї текстури, до якої будуть ставитися наступні команди. Для того щоб у процесі малювання зробити поточну текстуру з деяким ідентифікатором, досить знову викликати команду `glBindTexture()` з відповідним значенням *target* і *texture*. Таким чином, команда `glBindTexture()` включає режим створення текстури з ідентифікатором *texture*, якщо така текстура ще не створена, або режим її використання робить цю текстуру поточною.

Оскільки не всі апаратури можуть оперувати текстурами великого розміру, доцільно обмежити розміри текстури до 256x256 або 512x512 пікселів. Відзначимо, що використання невеликих текстур підвищує ефективність програми.

Накладення текстури на об'єкти

При накладенні текстури, як уже згадувалося, треба враховувати випадок, коли розміри текстури відрізняються від віконних розмірів об'єкта, на який вона накладається. При цьому можливо як розтягання, так і стиснення зображення, і те, як будуть проводитися ці перетворення, може серйозно вплинути на якість побудованого зображення. Для визначення положення точки на текстурі використовується параметрична система координат (s,t), причому значення s і t перебувають у відрізку [0,1] (рис. 6.8).

Для зміни різних параметрів текстури застосовуються команди:

```
void glTexParameter[i f] (GLenum target, GLenum
                          pname,GLenum param)

void glTexParameter[i f]v (GLenum target, GLenum
                          pname,Glenum* params)
```

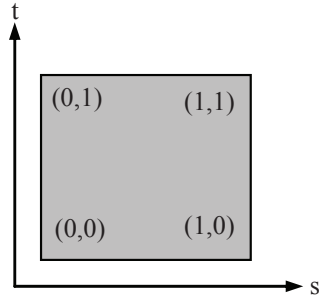


Рис.6.8. Текстурні координати

При цьому *target* може приймати значення **GL_TEXTURE_1D** або **GL_TEXTURE_2D**, *pname* визначає, яку властивість будемо міняти, а за допомогою *param* або *params* установлюється нове значення. Можливі значення *pname*:

GL_TEXTURE_MIN_FILTER параметр *param* визначає функцію, що буде використовуватися для стиснення текстури. При значенні **GL_NEAREST** буде використовуватися один (найближчий), а при значенні **GL_LINEAR** чотири найближчих елементи текстури. Значення за замовчуванням: **GL_LINEAR**.

GL_TEXTURE_MAG_FILTER параметр *param* визначає функцію, що буде використовуватися для збільшення (розтягання) текстури. При значенні **GL_NEAREST** буде використовуватися один (найближчий),

Накладення текстури на об'єкти

При накладенні текстури, як уже згадувалося, треба враховувати випадок, коли розміри текстури відрізняються від віконних розмірів об'єкта, на який вона накладається. При цьому можливо як розтягання, так і стиснення зображення, і те, як будуть проводитися ці перетворення, може серйозно вплинути на якість побудованого зображення. Для визначення положення точки на текстурі використовується параметрична система координат (s,t), причому значення s і t перебувають у відрізку [0,1] (рис. 6.8).

Для зміни різних параметрів текстури застосовуються команди:

```
void glTexParameter[i f] (GLenum target, GLenum
                          pname,GLenum param)

void glTexParameter[i f]v (GLenum target, GLenum
                          pname,Glenum* params)
```

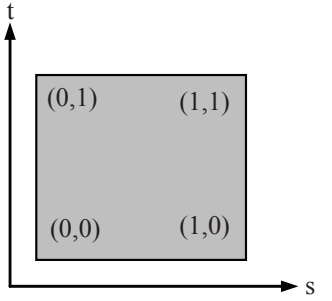


Рис.6.8. Текстурні координати

При цьому *target* може приймати значення **GL_TEXTURE_1D** або **GL_TEXTURE_2D**, *pname* визначає, яку властивість будемо міняти, а за допомогою *param* або *params* установлюється нове значення. Можливі значення *pname*:

GL_TEXTURE_MIN_FILTER параметр *param* визначає функцію, що буде використовуватися для стиснення текстури. При значенні **GL_NEAREST** буде використовуватися один (найближчий), а при значенні **GL_LINEAR** чотири найближчих елементи текстури. Значення за замовчуванням: **GL_LINEAR**.

GL_TEXTURE_MAG_FILTER параметр *param* визначає функцію, що буде використовуватися для збільшення (розтягання) текстури. При значенні **GL_NEAREST** буде використовуватися один (найближчий),

а при значенні **GL_LINEAR** чотири найближчих елементи текстури. Значення за замовчуванням:**GL_LINEAR**.

GL_TEXTURE_WRAP_S параметр *param* встановлює значення координати s, якщо воно не входить у відрізок [0,1]. При значенні **GL_ REPEAT** ціла частина s відкидається, і в результаті зображення розмножується по поверхні. При значенні **GL_CLAMP** використовуються крайові значення: 0 або 1, що зручно використовувати, якщо на об'єкт накладається один образ. Значення за замовчуванням: **GL_REPEAT**.

GL_TEXTURE_WRAP_T аналогічно попередньому значенню, тільки для координати t.

Використання режиму **GL_NEAREST** підвищує швидкість накладення текстури, однак при цьому знижується якість, тому що на відміну від **GL_LINEAR** інтерполяція не виробляється.

Для того щоб визначити, як текстура буде взаємодіяти з матеріалом, з якого зроблений об'єкт, використовуються команди

```
void glTexEnv[i f]  (GLenum target, GLenum pname,
                     GLtype param)
void glTexEnv[i f]v (GLenum target, GLenum pname,
                     GLtype *params)
```

Параметр *target* повинен дорівнювати **GL_TEXTURE_ENV**, а як *pname* розглянемо тільки одне значення **GL_TEXTURE_ENV_MODE**, що найбільш часто застосовується.

Найбільш часто використовувані значення параметра *param*:

GL_MODULATE кінцевий колір добувається як добуток кольорів точки на поверхні й кольору відповідної їй точки на текстурі.

GL_REPLACE як кінцевий колір використовується колір точки на текстурі.

Наступна програма демонструє загальний підхід до створення текстур:

а при значенні **GL_LINEAR** чотири найближчих елементи текстури. Значення за замовчуванням:**GL_LINEAR**.

GL_TEXTURE_WRAP_S параметр *param* встановлює значення координати s, якщо воно не входить у відрізок [0,1]. При значенні **GL_ REPEAT** ціла частина s відкидається, і в результаті зображення розмножується по поверхні. При значенні **GL_CLAMP** використовуються крайові значення: 0 або 1, що зручно використовувати, якщо на об'єкт накладається один образ. Значення за замовчуванням: **GL_REPEAT**.

GL_TEXTURE_WRAP_T аналогічно попередньому значенню, тільки для координати t.

Використання режиму **GL_NEAREST** підвищує швидкість накладення текстури, однак при цьому знижується якість, тому що на відміну від **GL_LINEAR** інтерполяція не виробляється.

Для того щоб визначити, як текстура буде взаємодіяти з матеріалом, з якого зроблений об'єкт, використовуються команди

```
void glTexEnv[i f]  (GLenum target, GLenum pname,
                     GLtype param)
void glTexEnv[i f]v (GLenum target, GLenum pname,
                     GLtype *params)
```

Параметр *target* повинен дорівнювати **GL_TEXTURE_ENV**, а як *pname* розглянемо тільки одне значення **GL_TEXTURE_ENV_MODE**, що найбільш часто застосовується.

Найбільш часто використовувані значення параметра *param*:

GL_MODULATE кінцевий колір добувається як добуток кольорів точки на поверхні й кольору відповідної їй точки на текстурі.

GL_REPLACE як кінцевий колір використовується колір точки на текстурі.

Наступна програма демонструє загальний підхід до створення текстур:

```
/* потрібне нам кількість текстур */
#define NUM_TEXTURES 10
/* ідентифікатори текстур */
int TextureIDs[NUM_TEXTURES];
/* образ текстури */
AUX_RGBImageRec *pImage;

…
/* 1) отримуємо ідентифікатори текстур */
glGenTextures(NUM_TEXTURES,TextureIDs);

/* 2) вибираємо текстуру для модифікації параметрів */
glBindTexture(TextureIDs[i]); /* 0<=i<NUM_TEXTURES*/
/* 3) завантажуємо текстуру. Розміри текстури - ступінь
2 */
pImage=dibImageLoad("texture.bmp");
if (Texture!=NULL)
{
  /* 4) передаємо текстуру OpenGL і задаємо параметри*/
  /* вирівнювання по байті */
  glPixelStorei(GL_UNPACK_ALIGNMENT,1);
  gluBuildMipmaps(GL_TEXTURE_2D,GL_RGB, pImage->size,
    pImage->size, GL_RGB, GL_UNSIGNED_BYTE,
    pImage->data);
  glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,
    (float)GL_LINEAR);
  glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
    (float)GL_LINEAR);
  glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
    (float)GL_REPEAT);
  glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
    (float)GL_REPEAT);
  glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
    (float)GL_REPLACE);
  /* 5) видаляємо вихідне зображення.*/
  free(Texture);
}else Error();
```

Текстурні координати

Перед нанесенням текстури на об'єкт необхідно встановити відповідність між точками на поверхні об'єкта й на самій текстурі. Задавати цю відповідність можна двома методами: окремо для кожної


```
/* потрібне нам кількість текстур */
#define NUM_TEXTURES 10
/* ідентифікатори текстур */
int TextureIDs[NUM_TEXTURES];
/* образ текстури */
AUX_RGBImageRec *pImage;

…
/* 1) отримуємо ідентифікатори текстур */
glGenTextures(NUM_TEXTURES,TextureIDs);

/* 2) вибираємо текстуру для модифікації параметрів */
glBindTexture(TextureIDs[i]); /* 0<=i<NUM_TEXTURES*/
/* 3) завантажуємо текстуру. Розміри текстури - ступінь
2 */
pImage=dibImageLoad("texture.bmp");
if (Texture!=NULL)
{
  /* 4) передаємо текстуру OpenGL і задаємо параметри*/
  /* вирівнювання по байті */
  glPixelStorei(GL_UNPACK_ALIGNMENT,1);
  gluBuildMipmaps(GL_TEXTURE_2D,GL_RGB, pImage->size,
    pImage->size, GL_RGB, GL_UNSIGNED_BYTE,
    pImage->data);
  glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,
    (float)GL_LINEAR);
  glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
    (float)GL_LINEAR);
  glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
    (float)GL_REPEAT);
  glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
    (float)GL_REPEAT);
  glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
    (float)GL_REPLACE);
  /* 5) видаляємо вихідне зображення.*/
  free(Texture);
}else Error();
```

Текстурні координати

Перед нанесенням текстури на об'єкт необхідно встановити відповідність між точками на поверхні об'єкта й на самій текстурі. Задавати цю відповідність можна двома методами: окремо для кожної

вершини або відразу для всіх вершин, задавши параметри спеціальної функції відображення.

Перший метод реалізується за допомогою команд

```
void glTexCoord[1 2 3 4][s i f d]  (type coord)
void glTexCoord[1 2 3 4][s i f d]v (type *coord)
```

Найчастіше використовуються команда **glTexCoord2*(type s, type t)**, що задає поточні координати текстури. Поняття поточних координат текстури аналогічно поняттям поточних кольорів і поточної нормалі, і є атрибутом вершини. Однак навіть для куба знаходження відповідних координат текстури є досить трудомістким заняттям, тому в бібліотеці GLU крім команд, що проводять побудову таких примітивів, як сфера, циліндр і диск, передбачене також накладення на них текстур. Для цього досить викликати команду

```
void gluQuadricTexture (GLUquadricObj* quadObject,
          GLboolean textureCoords)
```

з параметром *textureCoords* рівним **GL_TRUE**, і тоді поточна текстура буде автоматично накладатися на примітив.

Другий метод реалізується за допомогою команд

```
void glTexGen[i f d]  (GLenum coord, GLenum pname,
                       GLtype param)
void glTexGen[i f d]v (GLenum coord, GLenum pname,
                       const GLtype *params)
```

Параметр *coord* визначає, для якої координати задається формула, і може приймати значення **GL_S,GL_T**; *pname* може дорівнювати одному з таких значень:

GL_TEXTURE_GEN_MODE визначає функцію для накладення текстури.

У цьому випадку аргумент *param* приймає значення:

GL_OBJECT_LINEAR значення відповідної текстурної координати визначається відстанню до площини, що задається за допомогою значення pname **GL_OBJECT_PLANE** (див. нижче). Формула ви-

вершини або відразу для всіх вершин, задавши параметри спеціальної функції відображення.

Перший метод реалізується за допомогою команд

```
void glTexCoord[1 2 3 4][s i f d]  (type coord)
void glTexCoord[1 2 3 4][s i f d]v (type *coord)
```

Найчастіше використовуються команда **glTexCoord2*(type s, type t)**, що задає поточні координати текстури. Поняття поточних координат текстури аналогічно поняттям поточних кольорів і поточної нормалі, і є атрибутом вершини. Однак навіть для куба знаходження відповідних координат текстури є досить трудомістким заняттям, тому в бібліотеці GLU крім команд, що проводять побудову таких примітивів, як сфера, циліндр і диск, передбачене також накладення на них текстур. Для цього досить викликати команду

```
void gluQuadricTexture (GLUquadricObj* quadObject,
          GLboolean textureCoords)
```

з параметром *textureCoords* рівним **GL_TRUE**, і тоді поточна текстура буде автоматично накладатися на примітив.

Другий метод реалізується за допомогою команд

```
void glTexGen[i f d]  (GLenum coord, GLenum pname,
                       GLtype param)
void glTexGen[i f d]v (GLenum coord, GLenum pname,
                       const GLtype *params)
```

Параметр *coord* визначає, для якої координати задається формула, і може приймати значення **GL_S,GL_T**; *pname* може дорівнювати одному з таких значень:

GL_TEXTURE_GEN_MODE визначає функцію для накладення текстури.

У цьому випадку аргумент *param* приймає значення:

GL_OBJECT_LINEAR значення відповідної текстурної координати визначається відстанню до площини, що задається за допомогою значення pname **GL_OBJECT_PLANE** (див. нижче). Формула ви-

глядає так: g=x*xp+y*yp+z*zp+w*wp, де g - відповідна текстурна координата (s або p), x, y, z, w - координати відповідної точки. xp, yp, zp, wp - коефіцієнти рівняння площини. У формулі використовуються координати об'єкта.

GL_EYE_LINEAR аналогічно попередньому значенню, тільки у формулі використовуються видові координати. Тобто координати текстури об'єкта в цьому випадку залежать від положення цього об'єкта.

GL_SPHERE_MAP дозволяє моделювати відбиття від поверхні об'єкта. Текстура наче "обертається" навколо об'єкта. Для даного методу використовуються видові координати й необхідне задавання нормалей.

GL_OBJECT_PLANE дозволяє задати площину, відстань до якої буде використовуватися при генерації координат, якщо встановлено режим **GL_OBJECT_LINEAR**. У цьому випадку параметр *params* є покажчиком на масив із чотирьох коефіцієнтів рівняння площини.

GL_EYE_PLANE аналогічно попередньому значенню. Дозволяє задати площину для режиму **GL_EYE_LINEAR**

Для встановлення автоматичного режиму задавання текстурних координат необхідно викликати команду `glEnable` з параметром **GL_TEXTURE_GEN_S** або **GL_TEXTURE_GEN_P**.

Приклад 6.2. Розглянемо, як можна задати дзеркальну текстуру. При такому накладенні текстури зображення буде наче відбиватися від поверхні об'єкта, викликаючи цікавий оптичний ефект. Для цього спочатку треба створити два цілочисленних масиви коефіцієнтів `s_coeffs` і `t_coeffs` відповідно зі значеннями (1,0,0,1) і (0,1,0,1), а потім викликати команди:

```
glEnable (GL_TEXTURE_GEN_S);
glTexGeni (GL_S, GL_TEXTURE_GEN_MODE, GL_EYE_LINEAR);
glTexGendv (GL_S, GL_EYE_PLANE, s_coeffs);
```

і такої ж команди для координати *t* з відповідними змінами.

Операції з пікселами

Після проведення всіх операцій по перетворенню координат вершин, обчислення кольорів і тому подібне OpenGL переходить до

глядає так: g=x*xp+y*yp+z*zp+w*wp, де g - відповідна текстурна координата (s або p), x, y, z, w - координати відповідної точки. xp, yp, zp, wp - коефіцієнти рівняння площини. У формулі використовуються координати об'єкта.

GL_EYE_LINEAR аналогічно попередньому значенню, тільки у формулі використовуються видові координати. Тобто координати текстури об'єкта в цьому випадку залежать від положення цього об'єкта.

GL_SPHERE_MAP дозволяє моделювати відбиття від поверхні об'єкта. Текстура наче "обертається" навколо об'єкта. Для даного методу використовуються видові координати й необхідне задавання нормалей.

GL_OBJECT_PLANE дозволяє задати площину, відстань до якої буде використовуватися при генерації координат, якщо встановлено режим **GL_OBJECT_LINEAR**. У цьому випадку параметр *params* є покажчиком на масив із чотирьох коефіцієнтів рівняння площини.

GL_EYE_PLANE аналогічно попередньому значенню. Дозволяє задати площину для режиму **GL_EYE_LINEAR**

Для встановлення автоматичного режиму задавання текстурних координат необхідно викликати команду `glEnable` з параметром **GL_TEXTURE_GEN_S** або **GL_TEXTURE_GEN_P**.

Приклад 6.2. Розглянемо, як можна задати дзеркальну текстуру. При такому накладенні текстури зображення буде наче відбиватися від поверхні об'єкта, викликаючи цікавий оптичний ефект. Для цього спочатку треба створити два цілочисленних масиви коефіцієнтів `s_coeffs` і `t_coeffs` відповідно зі значеннями (1,0,0,1) і (0,1,0,1), а потім викликати команди:

```
glEnable (GL_TEXTURE_GEN_S);
glTexGeni (GL_S, GL_TEXTURE_GEN_MODE, GL_EYE_LINEAR);
glTexGendv (GL_S, GL_EYE_PLANE, s_coeffs);
```

і такої ж команди для координати *t* з відповідними змінами.

Операції з пікселами

Після проведення всіх операцій по перетворенню координат вершин, обчислення кольорів і тому подібне OpenGL переходить до

етапу *растеризації*, на якому відбувається растеризація всіх примітивів, накладення текстури, накладення ефекту туману. Для кожного примітива результатом цього процесу є займана ним у буфері кадру область, кожному пікселу цієї області приписується колір і значення глибини.

OpenGL використовує цю інформацію, щоб записати оновлені дані в буфер кадру. Для цього OpenGL має не тільки окремий конвеєр обробки пікселів, але й кілька додаткових буферів різного призначення. Це дозволяє програмістові гнучко контролювати процес візуалізації на найнижчому рівні.

Графічна бібліотека OpenGL підтримує роботу з такими буферами:

– кілька буферів кольорів;
– буфер глибини;
– буфер-накопичувач (акумулятор);
– буфер маски.

Група буферів кольору включає буфер кадру, але таких буферів може бути кілька. При використанні подвійної буферизації говорять про робочий (front) і фоновий (back) буфери. Як правило, у фоновому буфері програма створює зображення, що потім разом копіюється в робочий буфер. На екрані може з'явитися інформація тільки з буферів кольору.

Буфер глибини використовується для видалення невидимих поверхонь, і пряма робота з ним потрібна вкрай рідко.

Буфер-накопичувач можна застосовувати для різних операцій. Більш докладно робота з ним описана в [2, 15].

Буфер маски використовується для формування піксельних масок (трафаретів), призначених для вирізання із загального масиву тих пікселів, які варто вивести на екран.

Змішування зображень. Прозорість

Різноманітні прозорі об'єкти - скло, прозорий посуд і таке інше часто зустрічаються в реальності, тому важливо вміти створювати такі об'єкти в інтерактивній графіці. OpenGL надає програмістові механізм роботи з напівпрозорими об'єктами.

етапу *растеризації*, на якому відбувається растеризація всіх примітивів, накладення текстури, накладення ефекту туману. Для кожного примітива результатом цього процесу є займана ним у буфері кадру область, кожному пікселу цієї області приписується колір і значення глибини.

OpenGL використовує цю інформацію, щоб записати оновлені дані в буфер кадру. Для цього OpenGL має не тільки окремий конвеєр обробки пікселів, але й кілька додаткових буферів різного призначення. Це дозволяє програмістові гнучко контролювати процес візуалізації на найнижчому рівні.

Графічна бібліотека OpenGL підтримує роботу з такими буферами:

– кілька буферів кольорів;
– буфер глибини;
– буфер-накопичувач (акумулятор);
– буфер маски.

Група буферів кольору включає буфер кадру, але таких буферів може бути кілька. При використанні подвійної буферизації говорять про робочий (front) і фоновий (back) буфери. Як правило, у фоновому буфері програма створює зображення, що потім разом копіюється в робочий буфер. На екрані може з'явитися інформація тільки з буферів кольору.

Буфер глибини використовується для видалення невидимих поверхонь, і пряма робота з ним потрібна вкрай рідко.

Буфер-накопичувач можна застосовувати для різних операцій. Більш докладно робота з ним описана в [2, 15].

Буфер маски використовується для формування піксельних масок (трафаретів), призначених для вирізання із загального масиву тих пікселів, які варто вивести на екран.

Змішування зображень. Прозорість

Різноманітні прозорі об'єкти - скло, прозорий посуд і таке інше часто зустрічаються в реальності, тому важливо вміти створювати такі об'єкти в інтерактивній графіці. OpenGL надає програмістові механізм роботи з напівпрозорими об'єктами.

Прозорість реалізується за допомогою спеціального режиму змішування кольорів (blending). Алгоритм змішування комбінує кольори так званих вхідних пікселів (тобто «кандидатів» на переміщення в буфер кадру) із кольорів відповідних пікселів, що вже зберігаються в буфері. Для змішування використовується четвертий компонент кольорів - альфа-компонента, тому цей режим називають ще альфа-змішуванням. Програма може управляти інтенсивністю компонента-альфа-компонента точно так, як і інтенсивністю основних кольорів, тобто задавати значення інтенсивності для кожного піксела або кожної вершини примітива.Режим включається за допомогою команди `glEnable(GL_BLEND)`.

Визначити параметри змішування можна за допомогою команди:

```
void glBlendFunc(enum src,enum dst)
```

Параметр *src* визначає, як отримати коефіцієнт k_1 вихідних кольорів піксела, а *dst* задає спосіб отримання коефіцієнта k_2 для кольорів у буфері кадру. Для отримання результуючих кольорів використовується наступна формула: $res=c_{src}*k_1+c_{dst}*k_2$, де c_{src} – кольори вихідного піксела, c_{dst} – кольори піксела в буфері кадру (res, k_1, k_1, c_{src}, c_{dst} – чотирикомпонентні RGBA-вектори).

Приведемо найбільш часто використовувані значення аргументів `src` і `dst`.

GL_SRC_ALPHA	$k=(A_s,A_s,A_s,A_s)$
GL_SRC_ONE_MINUS_ALPHA	$k=(1,1,1,1)-(A_s,A_s,A_s,A_s)$
GL_DST_COLOR	$k=(R_d,G_d,B_d)$
GL_ONE_MINUS_DST_COLOR	$k=(1,1,1,1)-(R_d,G_d,B_d,A_d)$
GL_DST_ALPHA	$k=(A_d,A_d,A_d,A_d)$
GL_DST_ONE_MINUS_ALPHA	$k=(1,1,1,1)-(A_d,A_d,A_d,A_d)$
GL_SRC_COLOR	$k=(R_s,G_s,B_s)$
GL_ONE_MINUS_SRC_COLOR	$k=(1,1,1,1)-(R_s,G_s,B_s,A_s)$

Приклад 6.3. Припустимо, ми хочемо реалізувати виведення прозорих об'єктів. Коефіцієнт прозорості задається компонентом-альфа-компонентом

Прозорість реалізується за допомогою спеціального режиму змішування кольорів (blending). Алгоритм змішування комбінує кольори так званих вхідних пікселів (тобто «кандидатів» на переміщення в буфер кадру) із кольорів відповідних пікселів, що вже зберігаються в буфері. Для змішування використовується четвертий компонент кольорів - альфа-компонента, тому цей режим називають ще альфа-змішуванням. Програма може управляти інтенсивністю компонента-альфа-компонента точно так, як і інтенсивністю основних кольорів, тобто задавати значення інтенсивності для кожного піксела або кожної вершини примітива.Режим включається за допомогою команди `glEnable(GL_BLEND)`.

Визначити параметри змішування можна за допомогою команди:

```
void glBlendFunc(enum src,enum dst)
```

Параметр *src* визначає, як отримати коефіцієнт k_1 вихідних кольорів піксела, а *dst* задає спосіб отримання коефіцієнта k_2 для кольорів у буфері кадру. Для отримання результуючих кольорів використовується наступна формула: $res=c_{src}*k_1+c_{dst}*k_2$, де c_{src} – кольори вихідного піксела, c_{dst} – кольори піксела в буфері кадру (res, k_1, k_1, c_{src}, c_{dst} – чотирикомпонентні RGBA-вектори).

Приведемо найбільш часто використовувані значення аргументів `src` і `dst`.

GL_SRC_ALPHA	$k=(A_s,A_s,A_s,A_s)$
GL_SRC_ONE_MINUS_ALPHA	$k=(1,1,1,1)-(A_s,A_s,A_s,A_s)$
GL_DST_COLOR	$k=(R_d,G_d,B_d)$
GL_ONE_MINUS_DST_COLOR	$k=(1,1,1,1)-(R_d,G_d,B_d,A_d)$
GL_DST_ALPHA	$k=(A_d,A_d,A_d,A_d)$
GL_DST_ONE_MINUS_ALPHA	$k=(1,1,1,1)-(A_d,A_d,A_d,A_d)$
GL_SRC_COLOR	$k=(R_s,G_s,B_s)$
GL_ONE_MINUS_SRC_COLOR	$k=(1,1,1,1)-(R_s,G_s,B_s,A_s)$

Приклад 6.3. Припустимо, ми хочемо реалізувати виведення прозорих об'єктів. Коефіцієнт прозорості задається компонентом-альфа-компонентом

кольорів. Нехай 1 - непрозорий об'єкт; 0 - абсолютно прозорий, тобто невидимий. Для реалізації служить такий код:

```
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA,GL_SRC_ONE_MINUS_ALPHA);
```

Наприклад, напівпрозорий трикутник можна задати в такий спосіб:

```
glColor3f(1.0, 0.0, 0.0, 0.5);
glBegin(GL_TRIANGLES);
  glVertex3f(0.0, 0.0, 0.0);
  glVertex3f(1.0, 0.0, 0.0);
  glVertex3f(1.0, 1.0, 0.0);
glEnd();
```

Якщо в сцені є кілька прозорих об'єктів, які можуть перекривати один одного, коректне виведення можна гарантувати тільки у випадку виконання таких умов:

– всі прозорі об'єкти виводяться після непрозорих;

– при виведенні об'єкти із прозорістю повинні бути впорядковані по зменшенню глибини, тобто виводитися, починаючи з найбільш віддалених від спостерігача.

В OpenGL команди обробляються в порядку їхнього надходження, тому для реалізації перерахованих вимог досить розставити у відповідному порядку виклики команд glVertex*(), але й це в загальному випадку нетривіально.

Буфер-накопичувач

Буфер-накопичувач (*accumulation buffer*) – це один з додаткових буферів OpenGL. У ньому можна зберігати зображення, застосовуючи при цьому попіксельно спеціальні операції. Буфер-накопичувач широко використовується для створення різних спецефектів.Зображення береться з буфера, обраного для читання командою

```
void glReadBuffer(enum buf)
```

Аргумент *buf* визначає буфер для читання. Значення *buf*, рівні **GL_BACK**, **GL_FRONT**, визначають відповідні буфери кольорів для читання. **GL_BACK** задає як джерело пікселів позаекранний буфер; **GL_FRONT** – поточний вміст вікна виведення. Команда має

кольорів. Нехай 1 - непрозорий об'єкт; 0 - абсолютно прозорий, тобто невидимий. Для реалізації служить такий код:

```
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA,GL_SRC_ONE_MINUS_ALPHA);
```

Наприклад, напівпрозорий трикутник можна задати в такий спосіб:

```
glColor3f(1.0, 0.0, 0.0, 0.5);
glBegin(GL_TRIANGLES);
  glVertex3f(0.0, 0.0, 0.0);
  glVertex3f(1.0, 0.0, 0.0);
  glVertex3f(1.0, 1.0, 0.0);
glEnd();
```

Якщо в сцені є кілька прозорих об'єктів, які можуть перекривати один одного, коректне виведення можна гарантувати тільки у випадку виконання таких умов:

– всі прозорі об'єкти виводяться після непрозорих;

– при виведенні об'єкти із прозорістю повинні бути впорядковані по зменшенню глибини, тобто виводитися, починаючи з найбільш віддалених від спостерігача.

В OpenGL команди обробляються в порядку їхнього надходження, тому для реалізації перерахованих вимог досить розставити у відповідному порядку виклики команд glVertex*(), але й це в загальному випадку нетривіально.

Буфер-накопичувач

Буфер-накопичувач (*accumulation buffer*) – це один з додаткових буферів OpenGL. У ньому можна зберігати зображення, застосовуючи при цьому попіксельно спеціальні операції. Буфер-накопичувач широко використовується для створення різних спецефектів.Зображення береться з буфера, обраного для читання командою

```
void glReadBuffer(enum buf)
```

Аргумент *buf* визначає буфер для читання. Значення *buf*, рівні **GL_BACK**, **GL_FRONT**, визначають відповідні буфери кольорів для читання. **GL_BACK** задає як джерело пікселів позаекранний буфер; **GL_FRONT** – поточний вміст вікна виведення. Команда має

значення, якщо використовується дублююча буферизація. У протилежному випадку використовується тільки один буфер, що відповідає вікну виведення (всласне кажучи, OpenGL має набір додаткових буферів, використовуваних, зокрема, для роботи зі стереозображеннями, але тут ми їх розглядати не будемо).

Буфер-накопичувач є додатковим буфером кольорів. Він не використовується безпосередньо для виведення образів, але вони додаються в нього після виведення в один з буферів кольорів. Застосовуючи різні операції, описані нижче, можна потроху «накопичувати» зображення в буфері.

Потім отримане зображення переноситься з нагромаджувача-буфера-накопичувача в один з буферів кольорів, обраний на запис командою

```
void glDrawBuffer(enum buf)
```

Значення *buf* аналогічно значенню відповідного аргументу в команді **glReadBuffer**.

Всі операції з нагромаджувачем-буфером-накопичувачем контролюються командою

```
void glAccum(enum op,GLfloat value)
```

Аргумент *op* задає операцію над пікселами й може приймати такі значення:

GL_LOAD Піксел береться з буфера, обраного на читання, його значення множиться на *value* і заноситься в нагромаджувач-буфер-накопичувач.

GL_ACCUM Аналогічно попередньому, але отримане після множення значення складається із уже наявним у буфері.

GL_MULT Ця операція множить значення кожного піксела в буфері нагромадження на *value* .

GL_ADD Аналогічно попередньому, тільки замість множення використовується додавання.

значення, якщо використовується дублююча буферизація. У протилежному випадку використовується тільки один буфер, що відповідає вікну виведення (всласне кажучи, OpenGL має набір додаткових буферів, використовуваних, зокрема, для роботи зі стереозображеннями, але тут ми їх розглядати не будемо).

Буфер-накопичувач є додатковим буфером кольорів. Він не використовується безпосередньо для виведення образів, але вони додаються в нього після виведення в один з буферів кольорів. Застосовуючи різні операції, описані нижче, можна потроху «накопичувати» зображення в буфері.

Потім отримане зображення переноситься з нагромаджувача-буфера-накопичувача в один з буферів кольорів, обраний на запис командою

```
void glDrawBuffer(enum buf)
```

Значення *buf* аналогічно значенню відповідного аргументу в команді **glReadBuffer**.

Всі операції з нагромаджувачем-буфером-накопичувачем контролюються командою

```
void glAccum(enum op,GLfloat value)
```

Аргумент *op* задає операцію над пікселами й може приймати такі значення:

GL_LOAD Піксел береться з буфера, обраного на читання, його значення множиться на *value* і заноситься в нагромаджувач-буфер-накопичувач.

GL_ACCUM Аналогічно попередньому, але отримане після множення значення складається із уже наявним у буфері.

GL_MULT Ця операція множить значення кожного піксела в буфері нагромадження на *value* .

GL_ADD Аналогічно попередньому, тільки замість множення використовується додавання.

GL_RETURN Зображення переноситься з буфера-накопичувача в буфер, обраний для запису. Перед цим значення кожного піксела множиться на *value*.

Слід зазначити, що для використання нагромаджувача-буфера-накопичувача немає необхідності викликати які-небудь команди glEnable. Досить ініціалізувавати тільки сам буфер.

Буфер маски

Під час виведення пікселів у буфер кадру іноді виникає необхідність виводити не всі пікселі, а тільки деякі підмножини, тобто накласти трафарет (маску) на зображення. Для цього OpenGL надає так званий буфер маски (stencil buffer). Крім накладення маски, цей буфер надає ще кілька цікавих можливостей.

Перш ніж помістити піксел у буфер кадру, механізм візуалізації OpenGL дозволяє виконати порівняння (тест) між заданим значенням і значенням у буфері маски. Якщо тест проходить, піксел рисується в буфері кадру.

Механізм порівняння досить гнучкий і контролюється такими командами:

```
void glStencilFunc (enum func, int ref, uint mask)
void glStencilOp(enum sfail, enum dpfail, enum dppass)
```

Аргумент *ref* команди `glStencilFunc` задає значення для порівняння. Він повинен приймати значення від 0 до 2^s-1, s - число біт на точку в буфері маски.

За допомогою аргументу *func* задається функція порівняння. Він може приймати такі значення:

GL_NEVER тест ніколи не проходить, тобто завжди повертає false

GL_ALWAYS тест проходить завжди.

GL_LESS, GL_LEQUAL, GL_EQUAL,

GL_GEQUAL, GL_GREATE, GL_NOTEQUAL тест проходить у випадку, якщо *ref* відповідно менше значення в буфері маски, менше або дорівнює, дорівнює, більше, більше або дорівнює, або не дорівнює.

GL_RETURN Зображення переноситься з буфера-накопичувача в буфер, обраний для запису. Перед цим значення кожного піксела множиться на *value*.

Слід зазначити, що для використання нагромаджувача-буфера-накопичувача немає необхідності викликати які-небудь команди glEnable. Досить ініціалізувавати тільки сам буфер.

Буфер маски

Під час виведення пікселів у буфер кадру іноді виникає необхідність виводити не всі пікселі, а тільки деякі підмножини, тобто накласти трафарет (маску) на зображення. Для цього OpenGL надає так званий буфер маски (stencil buffer). Крім накладення маски, цей буфер надає ще кілька цікавих можливостей.

Перш ніж помістити піксел у буфер кадру, механізм візуалізації OpenGL дозволяє виконати порівняння (тест) між заданим значенням і значенням у буфері маски. Якщо тест проходить, піксел рисується в буфері кадру.

Механізм порівняння досить гнучкий і контролюється такими командами:

```
void glStencilFunc (enum func, int ref, uint mask)
void glStencilOp(enum sfail, enum dpfail, enum dppass)
```

Аргумент *ref* команди `glStencilFunc` задає значення для порівняння. Він повинен приймати значення від 0 до 2^s-1, s - число біт на точку в буфері маски.

За допомогою аргументу *func* задається функція порівняння. Він може приймати такі значення:

GL_NEVER тест ніколи не проходить, тобто завжди повертає false

GL_ALWAYS тест проходить завжди.

GL_LESS, GL_LEQUAL, GL_EQUAL,

GL_GEQUAL, GL_GREATE, GL_NOTEQUAL тест проходить у випадку, якщо *ref* відповідно менше значення в буфері маски, менше або дорівнює, дорівнює, більше, більше або дорівнює, або не дорівнює.

Аргумент *mask* задає маску для значень. Тобто у підсумку для цього тесту отримаємо наступну формулу: ((*ref* AND *mask*) *op* (svalue AND *mask*))

Команда `glStencilOp` призначена для визначення дій над пікселем буфера маски у випадку позитивного або негативного результату тесту.

Аргумент *sfail* задає дію у випадку негативного результату тесту і може приймати такі значення:

GL_KEEP, GL_ZERO, GL_REPLACE,
GL_INCR, GL_DECR, GL_INVERT,

відповідно зберігає значення в буфері маски, зводить до нуля, замінює на задане значення (*ref*), збільшує, зменшує або побітово інвертує.

Аргументи *dpfail* визначають дії у випадку негативного результату тесту на глибину в z-буфері, а *dppass* задає дія у випадку позитивного результату цього тесту. Аргументи приймають ті ж значення, що й аргумент *sfail*. За замовчуванням всі три параметри встановлені на **GL_KEEP**.

Для включення маскування необхідно виконати команду `glEnable(GL_STENCIL_TEST);`

Буфер маски використовується при створенні таких спецефектів, як падаючі тіні, відбивання, плавні переходи з однієї картинки в іншу та ін.

6.4. Приклади роботи з OpenGL

У цьому розділі ми розглянемо, як за допомогою OpenGL створювати деякі цікаві візуальні ефекти, безпосередня підтримка яких відсутня у стандарті бібліотеки.

Усунення ступінчастості

Почнемо із усунення ступінчастості (*antialiasing*). Ефект ступінчастості (*aliasing*) виникає в результаті похибок растеризації примітивів у буфері кадру через кінцеву (і, як правило, невелику) можливість буфера. Є кілька підходів до розв'язання даної проблеми.

Аргумент *mask* задає маску для значень. Тобто у підсумку для цього тесту отримаємо наступну формулу: ((*ref* AND *mask*) *op* (svalue AND *mask*))

Команда `glStencilOp` призначена для визначення дій над пікселем буфера маски у випадку позитивного або негативного результату тесту.

Аргумент *sfail* задає дію у випадку негативного результату тесту і може приймати такі значення:

GL_KEEP, GL_ZERO, GL_REPLACE,
GL_INCR, GL_DECR, GL_INVERT,

відповідно зберігає значення в буфері маски, зводить до нуля, замінює на задане значення (*ref*), збільшує, зменшує або побітово інвертує.

Аргументи *dpfail* визначають дії у випадку негативного результату тесту на глибину в z-буфері, а *dppass* задає дія у випадку позитивного результату цього тесту. Аргументи приймають ті ж значення, що й аргумент *sfail*. За замовчуванням всі три параметри встановлені на **GL_KEEP**.

Для включення маскування необхідно виконати команду `glEnable(GL_STENCIL_TEST);`

Буфер маски використовується при створенні таких спецефектів, як падаючі тіні, відбивання, плавні переходи з однієї картинки в іншу та ін.

6.4. Приклади роботи з OpenGL

У цьому розділі ми розглянемо, як за допомогою OpenGL створювати деякі цікаві візуальні ефекти, безпосередня підтримка яких відсутня у стандарті бібліотеки.

Усунення ступінчастості

Почнемо із усунення ступінчастості (*antialiasing*). Ефект ступінчастості (*aliasing*) виникає в результаті похибок растеризації примітивів у буфері кадру через кінцеву (і, як правило, невелику) можливість буфера. Є кілька підходів до розв'язання даної проблеми.

Наприклад, можна застосовувати фільтрацію отриманого зображення. Також цей ефект можна усувати на етапі растеризації, згладжуючи образ кожного примітива. Тут ми розглянемо прийом, що дозволяє усувати подібні артефакти для всієї сцени цілком.

Для кожного кадру необхідно намалювати сцену кілька разів, на кожному проході небагато зміщаючи камеру щодо початкового положення. Положення камер, наприклад, можуть утворювати окружність. Якщо зрушення камери відносно мале, то похибки дискретизації виявляться по-різному, і, усереднюючи отримані зображення, ми отримаємо згладжене зображення.

Найпростіше зрушувати положення спостерігача, але перед цим потрібно обчислити розмір зрушення так, щоб наведене до координат екрана значення не перевищувало, скажемо, половини розміру піксела.

Всі отримані зображення зберігаємо в буфері-накопичувачі з коефіцієнтом 1/n, де n - число проходів для кожного кадру. Чим більше таких проходів - тим нижче продуктивність, але кращий результат.

```
  for(i=0;i<samples_count;++i)
  /* звичайно samples_count лежить у межах від 5 до 10
*/
  {
    ShiftCamera(i); /* зрушуємо камеру */
    RenderScene();
    if (i==0)
     /* на першій ітерації завантажуємо зображення */
     glAccum(GL_LOAD,1/(float)samples_count);
    else
     /* додаємо до уже існуючому */
     glAccum(GL_ACCUM,1/(float)samples_count);
  }
  /* Пишемо те, що вийшло, назад у вихідний буфер */
  glAccum(GL_RETURN,1.0);
```

Слід зазначити, що усунення ступінчастості відразу для всієї сцени, як правило, пов'язане із серйозним падінням продуктивності візуалізації, тому що вся сцена рисується кілька разів. Сучасні прискорювачі звичайно апаратно реалізують інші методи, засновані на так званому ресамплінгу зображень.

Наприклад, можна застосовувати фільтрацію отриманого зображення. Також цей ефект можна усувати на етапі растеризації, згладжуючи образ кожного примітива. Тут ми розглянемо прийом, що дозволяє усувати подібні артефакти для всієї сцени цілком.

Для кожного кадру необхідно намалювати сцену кілька разів, на кожному проході небагато зміщаючи камеру щодо початкового положення. Положення камер, наприклад, можуть утворювати окружність. Якщо зрушення камери відносно мале, то похибки дискретизації виявляться по-різному, і, усереднюючи отримані зображення, ми отримаємо згладжене зображення.

Найпростіше зрушувати положення спостерігача, але перед цим потрібно обчислити розмір зрушення так, щоб наведене до координат екрана значення не перевищувало, скажемо, половини розміру піксела.

Всі отримані зображення зберігаємо в буфері-накопичувачі з коефіцієнтом 1/n, де n - число проходів для кожного кадру. Чим більше таких проходів - тим нижче продуктивність, але кращий результат.

```
  for(i=0;i<samples_count;++i)
  /* звичайно samples_count лежить у межах від 5 до 10
*/
  {
    ShiftCamera(i); /* зрушуємо камеру */
    RenderScene();
    if (i==0)
     /* на першій ітерації завантажуємо зображення */
     glAccum(GL_LOAD,1/(float)samples_count);
    else
     /* додаємо до уже існуючому */
     glAccum(GL_ACCUM,1/(float)samples_count);
  }
  /* Пишемо те, що вийшло, назад у вихідний буфер */
  glAccum(GL_RETURN,1.0);
```

Слід зазначити, що усунення ступінчастості відразу для всієї сцени, як правило, пов'язане із серйозним падінням продуктивності візуалізації, тому що вся сцена рисується кілька разів. Сучасні прискорювачі звичайно апаратно реалізують інші методи, засновані на так званому ресамплінгу зображень.

Побудова тіней

В OpenGL немає вбудованої підтримки побудови тіней на рівні базових команд. У значній мірі це пояснюється тим, що існує безліч алгоритмів їхньої побудови, які можуть бути реалізовані через функції OpenGL. Присутність тіней сильно впливає на реалістичність тривимірного зображення, тому розглянемо один з підходів до їх побудови.

Більшість алгоритмів, призначених для побудови тіней, використовують модифіковані принципи перспективної проекції. Тут розглядається один з найпростіших методів. З його допомогою можна отримувати тіні, що відкидаються тривимірним об'єктом на площину.

Загальний підхід такий: для всіх точок об'єкта перебуває їхня проекція паралельно вектору, що з'єднує дану точку й точку, у якій перебуває джерело світла, на якусь задану площину. Тим самим отримуємо новий об'єкт, що цілком лежить у заданій площині. Цей об'єкт і є тінню вихідного.

Розглянемо математичні основи даного методу.

Нехай:

P – точка у тривимірному просторі, що відкидає тінь.

L – положення джерела світла, що висвітлює дану точку.

S=a(**L**-**P**)-**P** - точка, в яку відкидає тінь точка P, де a – параметр.

Припустимо, що тінь падає на площину *z=0*. У цьому випадку $a=z_p/(z_l-z_p)$. Отже,

$x_s = (x_p z_l - z_l z_p) / (z_l - z_p)$,

$y_s = (y_p z_l - y_l z_p) / (z_l - z_p)$,

$z_s = 0$

Введемо однорідні координати:

$x_s=x_s'/w_s'$

$y_s=y_s'/w_s'$

$z_s=0$

$w_s'=z_l-z_p$

Звідси координати S можуть бути отримані з використанням множення матриць у такий спосіб:

Побудова тіней

В OpenGL немає вбудованої підтримки побудови тіней на рівні базових команд. У значній мірі це пояснюється тим, що існує безліч алгоритмів їхньої побудови, які можуть бути реалізовані через функції OpenGL. Присутність тіней сильно впливає на реалістичність тривимірного зображення, тому розглянемо один з підходів до їх побудови.

Більшість алгоритмів, призначених для побудови тіней, використовують модифіковані принципи перспективної проекції. Тут розглядається один з найпростіших методів. З його допомогою можна отримувати тіні, що відкидаються тривимірним об'єктом на площину.

Загальний підхід такий: для всіх точок об'єкта перебуває їхня проекція паралельно вектору, що з'єднує дану точку й точку, у якій перебуває джерело світла, на якусь задану площину. Тим самим отримуємо новий об'єкт, що цілком лежить у заданій площині. Цей об'єкт і є тінню вихідного.

Розглянемо математичні основи даного методу.

Нехай:

P – точка у тривимірному просторі, що відкидає тінь.

L – положення джерела світла, що висвітлює дану точку.

S=a(**L**-**P**)-**P** - точка, в яку відкидає тінь точка P, де a – параметр.

Припустимо, що тінь падає на площину *z=0*. У цьому випадку $a=z_p/(z_l-z_p)$. Отже,

$x_s = (x_p z_l - z_l z_p) / (z_l - z_p)$,

$y_s = (y_p z_l - y_l z_p) / (z_l - z_p)$,

$z_s = 0$

Введемо однорідні координати:

$x_s=x_s'/w_s'$

$y_s=y_s'/w_s'$

$z_s=0$

$w_s'=z_l-z_p$

Звідси координати S можуть бути отримані з використанням множення матриць у такий спосіб:

$$[x_s' \quad y_s' \quad 0 \quad w_s'] = [x_s \quad y_s \quad z_s \quad 1] \begin{bmatrix} z_l & 0 & 0 & 0 \\ 0 & z_l & 0 & 0 \\ -x_l & -y_l & 0 & -1 \\ 0 & 0 & 0 & z_l \end{bmatrix}$$

Для того щоб алгоритм міг розраховувати тінь, що падає на довільну площину, розглянемо довільну точку на лінії між **S** і **P**, подану в однорідних координатах:

a+b**,** де a і b – скалярні параметри.

Наступна матриця задає площину через координати її нормалі:

$$G = \begin{bmatrix} x_n \\ y_n \\ z_n \\ d \end{bmatrix}$$

Точка, в якій промінь, проведений від джерела світла через дану точку **P,** перетинає площина G, визначається параметрами a і b, що задовольняють рівняння

(a+b)**G** = 0.

Звідси отримуємо: a(**PG)** + b(**LG)** = 0. Цьому рівнянню задовольняють

a = (**LG),** b = -(**PG)**

Отже, координати шуканої точки **S** = (**LG)P-(PG)L.** Користуючись асоціативністю матричного добутку, одержимо

S = **P**[(**LG**)**I** - **GL**], де **I –** одинична матриця.

Матриця (**LG**)**I** - **GL** використовується для одержання тіней на довільній площині.

Розглянемо деякі аспекти практичної реалізації даного методу за допомогою OpenGL.

Припустимо, що матриця floorShadow була раніше отримана нами з формули (**LG**)**I** - **GL**. Наступний код з її допомогою будує тіні для заданої площини:

```
  /* Робимо тіні напівпрозорими з використанням
змішання кольорів(blending) */
  glEnable(GL_BLEND);
```


$$[x_s' \quad y_s' \quad 0 \quad w_s'] = [x_s \quad y_s \quad z_s \quad 1] \begin{bmatrix} z_l & 0 & 0 & 0 \\ 0 & z_l & 0 & 0 \\ -x_l & -y_l & 0 & -1 \\ 0 & 0 & 0 & z_l \end{bmatrix}$$

Для того щоб алгоритм міг розраховувати тінь, що падає на довільну площину, розглянемо довільну точку на лінії між **S** і **P**, подану в однорідних координатах:

a+b**,** де a і b – скалярні параметри.

Наступна матриця задає площину через координати її нормалі:

$$G = \begin{bmatrix} x_n \\ y_n \\ z_n \\ d \end{bmatrix}$$

Точка, в якій промінь, проведений від джерела світла через дану точку **P,** перетинає площина G, визначається параметрами a і b, що задовольняють рівняння

(a+b)**G** = 0.

Звідси отримуємо: a(**PG)** + b(**LG)** = 0. Цьому рівнянню задовольняють

a = (**LG),** b = -(**PG)**

Отже, координати шуканої точки **S** = (**LG)P-(PG)L.** Користуючись асоціативністю матричного добутку, одержимо

S = **P**[(**LG**)**I** - **GL**], де **I –** одинична матриця.

Матриця (**LG**)**I** - **GL** використовується для одержання тіней на довільній площині.

Розглянемо деякі аспекти практичної реалізації даного методу за допомогою OpenGL.

Припустимо, що матриця floorShadow була раніше отримана нами з формули (**LG**)**I** - **GL**. Наступний код з її допомогою будує тіні для заданої площини:

```
  /* Робимо тіні напівпрозорими з використанням
змішання кольорів(blending) */
  glEnable(GL_BLEND);
```

```
  glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA);
  glDisable(GL_LIGHTING);
  glColor4f(0.0, 0.0, 0.0, 0.5);
  glPushMatrix();
     /* Проектуємо тінь */
     glMultMatrixf((GLfloat *) floorShadow);
     /* Візуалізуємо сцену в проекції */
     RenderGeometry();
  glPopMatrix();
  glEnable(GL_LIGHTING);
  glDisable(GL_BLEND);
  /* Візуалізуємо сцену у звичайному режимі */
  RenderGeometry();
```

Матриця floorShadow може бути отримана за допомогою такої функції:

```
  /* параметри:   plane – коефіцієнти рівняння площини
        lightpos - координати джерела світла
     повертає: matrix - результуюча матриця
  */
  void shadowmatrix(GLfloat matrix[4][4], GLfloat
plane[4],
                    GLfloat lightpos[4])
  {
    GLfloat dot;

    dot = plane[0] * lightpos[0] +
          plane[1] * lightpos[1] +
          plane[2] * lightpos[2] +
          plane[3] * lightpos[3];

    matrix[0][0] = dot - lightpos[0] * plane[0];
    matrix[1][0] = 0.f - lightpos[0] * plane[1];
    matrix[2][0] = 0.f - lightpos[0] * plane[2];
    matrix[3][0] = 0.f - lightpos[0] * plane[3];

    matrix[0][1] = 0.f - lightpos[1] * plane[0];
    matrix[1][1] = dot - lightpos[1] * plane[1];
    matrix[2][1] = 0.f - lightpos[1] * plane[2];
    matrix[3][1] = 0.f - lightpos[1] * plane[3];

    matrix[0][2] = 0.f - lightpos[2] * plane[0];
```



```
  glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA);
  glDisable(GL_LIGHTING);
  glColor4f(0.0, 0.0, 0.0, 0.5);
  glPushMatrix();
     /* Проектуємо тінь */
     glMultMatrixf((GLfloat *) floorShadow);
     /* Візуалізуємо сцену в проекції */
     RenderGeometry();
  glPopMatrix();
  glEnable(GL_LIGHTING);
  glDisable(GL_BLEND);
  /* Візуалізуємо сцену у звичайному режимі */
  RenderGeometry();
```

Матриця floorShadow може бути отримана за допомогою такої функції:

```
  /* параметри:   plane – коефіцієнти рівняння площини
        lightpos - координати джерела світла
     повертає: matrix - результуюча матриця
  */
  void shadowmatrix(GLfloat matrix[4][4], GLfloat
plane[4],
                    GLfloat lightpos[4])
  {
    GLfloat dot;

    dot = plane[0] * lightpos[0] +
          plane[1] * lightpos[1] +
          plane[2] * lightpos[2] +
          plane[3] * lightpos[3];

    matrix[0][0] = dot - lightpos[0] * plane[0];
    matrix[1][0] = 0.f - lightpos[0] * plane[1];
    matrix[2][0] = 0.f - lightpos[0] * plane[2];
    matrix[3][0] = 0.f - lightpos[0] * plane[3];

    matrix[0][1] = 0.f - lightpos[1] * plane[0];
    matrix[1][1] = dot - lightpos[1] * plane[1];
    matrix[2][1] = 0.f - lightpos[1] * plane[2];
    matrix[3][1] = 0.f - lightpos[1] * plane[3];

    matrix[0][2] = 0.f - lightpos[2] * plane[0];
```

```
    matrix[1][2] = 0.f - lightpos[2] * plane[1];
    matrix[2][2] = dot - lightpos[2] * plane[2];
    matrix[3][2] = 0.f - lightpos[2] * plane[3];

    matrix[0][3] = 0.f - lightpos[3] * plane[0];
    matrix[1][3] = 0.f - lightpos[3] * plane[1];
    matrix[2][3] = 0.f - lightpos[3] * plane[2];
    matrix[3][3] = dot - lightpos[3] * plane[3];
}
```

Тіні, побудовані таким чином, мають ряд недоліків:

– Описаний алгоритм припускає, що площини нескінченні, і не відрізає тіні по границі. Наприклад, якщо деякий об'єкт відкидає тінь на стіл, вона не буде відтинатися по границі, і, тим більше, "загортатися" на бічну поверхню стола.

– У деяких місцях тіні може спостерігатися ефект "подвійного змішування" (reblending), тобто темні плями в тих ділянках, де спроектовані трикутники перекривають один одного.

– Зі збільшенням числа поверхонь складність алгоритму різко збільшується, тому що для кожної поверхні потрібно заново будувати всю сцену, навіть якщо проблема відсікання тіней по границі буде вирішена.

– Тіні звичайно мають розмиті границі, а в наведеному алгоритмі вони завжди мають різкі краї. Частково уникнути цього дозволяє розрахунок тіней з декількох джерел світла, розташованих поруч, і наступне змішування результатів.

Є рішення першої й другої проблеми. Для цього використовується буфер маски. Отже, завдавання – відітнути виведення геометрії (тіні, у цьому випадку) по границі деякої довільної області й уникнути "подвійного змішування". Загальний алгоритм рішення з використанням буфера маски такий:

1. Очищаємо буфер маски значенням 0.

2. Відображаємо задану область відсікання, установлюючи значення в буфері маски в 1.

3. Малюємо тіні в тих областях, де в буфері маски встановлені значення 1. Якщо тест проходить, установлюємо в цій області значення 2. Тепер розглянемо ці етапи більш докладно.

1.


```
    matrix[1][2] = 0.f - lightpos[2] * plane[1];
    matrix[2][2] = dot - lightpos[2] * plane[2];
    matrix[3][2] = 0.f - lightpos[2] * plane[3];

    matrix[0][3] = 0.f - lightpos[3] * plane[0];
    matrix[1][3] = 0.f - lightpos[3] * plane[1];
    matrix[2][3] = 0.f - lightpos[3] * plane[2];
    matrix[3][3] = dot - lightpos[3] * plane[3];
}
```

Тіні, побудовані таким чином, мають ряд недоліків:

– Описаний алгоритм припускає, що площини нескінченні, і не відрізає тіні по границі. Наприклад, якщо деякий об'єкт відкидає тінь на стіл, вона не буде відтинатися по границі, і, тим більше, "загортатися" на бічну поверхню стола.

– У деяких місцях тіні може спостерігатися ефект "подвійного змішування" (reblending), тобто темні плями в тих ділянках, де спроектовані трикутники перекривають один одного.

– Зі збільшенням числа поверхонь складність алгоритму різко збільшується, тому що для кожної поверхні потрібно заново будувати всю сцену, навіть якщо проблема відсікання тіней по границі буде вирішена.

– Тіні звичайно мають розмиті границі, а в наведеному алгоритмі вони завжди мають різкі краї. Частково уникнути цього дозволяє розрахунок тіней з декількох джерел світла, розташованих поруч, і наступне змішування результатів.

Є рішення першої й другої проблеми. Для цього використовується буфер маски. Отже, завдавання – відітнути виведення геометрії (тіні, у цьому випадку) по границі деякої довільної області й уникнути "подвійного змішування". Загальний алгоритм рішення з використанням буфера маски такий:

1. Очищаємо буфер маски значенням 0.

2. Відображаємо задану область відсікання, установлюючи значення в буфері маски в 1.

3. Малюємо тіні в тих областях, де в буфері маски встановлені значення 1. Якщо тест проходить, установлюємо в цій області значення 2. Тепер розглянемо ці етапи більш докладно.

1.

```
  /* очищаємо буфер маски*/
  glClearStencil(0x0);
  /* включаємо тест */
  glEnable(GL_STENCIL_TEST);
  2.
  /* умова завжди виконана й значення в буфері буде
рівним 1*/
  glStencilFunc (GL_ALWAYS, 0x1, 0xffffffff);
  /* у кожному разі заміняємо значення в буфері маски*/
  glStencilOp (GL_REPLACE, GL_REPLACE, GL_REPLACE);
  /* виводимо геометрію, по якій потім буде відсічена
тінь*/
  RenderPlane();
  3.
  /* умова виконана й тест дає істину тільки якщо
значення в буфері маски дорівнює 1 */
  glStencilFunc (GL_EQUAL, 0x1, 0xffffffff);
  /* значення в буфері дорівнює 2,якщо тінь уже
виведена */
  glStencilOp (GL_KEEP, GL_KEEP, GL_INCR);
  /* виводимо тіні */
  RenderShadow();
```

Отже, навіть при застосуванні маскування залишаються деякі проблеми, пов'язані з роботою z-буфера. Зокрема, деякі ділянки тіней можуть стати невидимими. Для розв’язання цієї проблеми можна трохи підняти тіні над площиною за допомогою модифікації рівняння, що описує площину. Опис інших методів виходить за рамки даного посібника.

Дзеркальні відбиття

Розглянемо алгоритм побудови відбиття від плоских об'єктів. Такі відбиття надають великої реалістичності побудованому зображенню і їх відносно легко реалізувати.

Алгоритм використовує інтуїтивне подавання повної сцени із дзеркалом як складеної із двох: «дійсної» та «віртуальної» - яка находиться за дзеркалом. Отже, процес малювання відбиття складається із двох частин: візуалізації звичайної та побудови й візуалізації віртуальної сцени. Для кожного об'єкта «дійсної» сцени будується його відбитий двійник, що спостерігач і побачить у дзеркалі.


```
  /* очищаємо буфер маски*/
  glClearStencil(0x0);
  /* включаємо тест */
  glEnable(GL_STENCIL_TEST);
  2.
  /* умова завжди виконана й значення в буфері буде
рівним 1*/
  glStencilFunc (GL_ALWAYS, 0x1, 0xffffffff);
  /* у кожному разі заміняємо значення в буфері маски*/
  glStencilOp (GL_REPLACE, GL_REPLACE, GL_REPLACE);
  /* виводимо геометрію, по якій потім буде відсічена
тінь*/
  RenderPlane();
  3.
  /* умова виконана й тест дає істину тільки якщо
значення в буфері маски дорівнює 1 */
  glStencilFunc (GL_EQUAL, 0x1, 0xffffffff);
  /* значення в буфері дорівнює 2,якщо тінь уже
виведена */
  glStencilOp (GL_KEEP, GL_KEEP, GL_INCR);
  /* виводимо тіні */
  RenderShadow();
```

Отже, навіть при застосуванні маскування залишаються деякі проблеми, пов'язані з роботою z-буфера. Зокрема, деякі ділянки тіней можуть стати невидимими. Для розв’язання цієї проблеми можна трохи підняти тіні над площиною за допомогою модифікації рівняння, що описує площину. Опис інших методів виходить за рамки даного посібника.

Дзеркальні відбиття

Розглянемо алгоритм побудови відбиття від плоских об'єктів. Такі відбиття надають великої реалістичності побудованому зображенню і їх відносно легко реалізувати.

Алгоритм використовує інтуїтивне подавання повної сцени із дзеркалом як складеної із двох: «дійсної» та «віртуальної» - яка находиться за дзеркалом. Отже, процес малювання відбиття складається із двох частин: візуалізації звичайної та побудови й візуалізації віртуальної сцени. Для кожного об'єкта «дійсної» сцени будується його відбитий двійник, що спостерігач і побачить у дзеркалі.

Для ілюстрації розглянемо кімнату із дзеркалом на стіні. Кімната й об'єкти, що перебувають у ній, виглядають у дзеркалі так, ніби дзеркало було вікном, а за ним була б ще одна така ж кімната з тими ж об'єктами, але симетрично відбитими щодо площини, проведеної через поверхню дзеркала.

Спрощений варіант алгоритму створення плоского відбиття складається з таких кроків:

1. Малюємо сцену як звичайно, але без об'єктів-дзеркал.
2. Використовуючи буфер маски, обмежуємо подальше виведення проекцією дзеркала на екран.
3. Робимо візуалізацію сцени, відбиту щодо площини дзеркала.

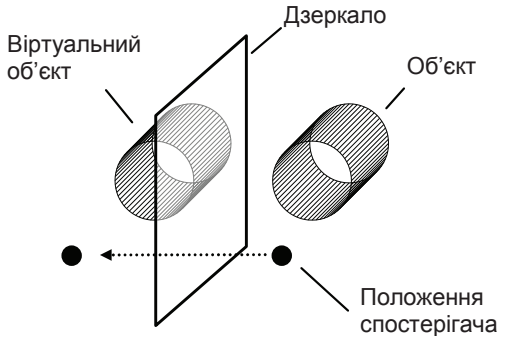


Рис.6.9. Дзеркальне відбиття

При цьому буфер маски дозволить обмежити виведення формою проекції об'єкта-дзеркала. Ця послідовність дій дозволить отримати переконливий ефект відбиття.

Розглянемо етапи більш докладно:

Спочатку необхідно намалювати сцену як звичайно. Не будемо зупинятися на цьому етапі докладно. Помітимо тільки, що, очищаючи буфер OpenGL безпосередньо перед малюванням, потрібно не забути очистити буфер маски:

```
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT|
      GL_STENCIL_BUFFER_BIT);
```


Для ілюстрації розглянемо кімнату із дзеркалом на стіні. Кімната й об'єкти, що перебувають у ній, виглядають у дзеркалі так, ніби дзеркало було вікном, а за ним була б ще одна така ж кімната з тими ж об'єктами, але симетрично відбитими щодо площини, проведеної через поверхню дзеркала.

Спрощений варіант алгоритму створення плоского відбиття складається з таких кроків:

1. Малюємо сцену як звичайно, але без об'єктів-дзеркал.
2. Використовуючи буфер маски, обмежуємо подальше виведення проекцією дзеркала на екран.
3. Робимо візуалізацію сцени, відбиту щодо площини дзеркала.

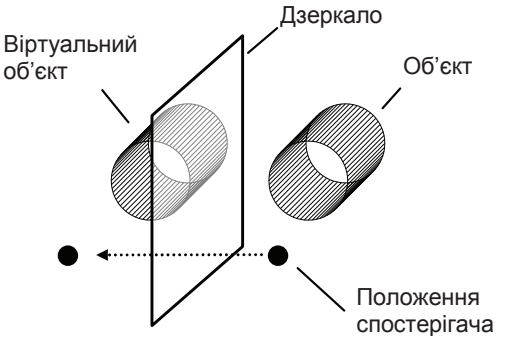


Рис.6.9. Дзеркальне відбиття

При цьому буфер маски дозволить обмежити виведення формою проекції об'єкта-дзеркала. Ця послідовність дій дозволить отримати переконливий ефект відбиття.

Розглянемо етапи більш докладно:

Спочатку необхідно намалювати сцену як звичайно. Не будемо зупинятися на цьому етапі докладно. Помітимо тільки, що, очищаючи буфер OpenGL безпосередньо перед малюванням, потрібно не забути очистити буфер маски:

```
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT|
      GL_STENCIL_BUFFER_BIT);
```

Під час візуалізації сцени краще не малювати об'єкти, які потім стануть дзеркальними.На другому етапі необхідно обмежити подальше виведення проекцією дзеркального об'єкта на екран. Для цього набудовуємо буфер маски й малюємо дзеркало

```
  glEnable(GL_STENCIL_TEST);
  /* умова завжди виконана й значення в буфері буде
дорівнює 1*/
  glStencilFunc(GL_ALWAYS, 1, 0);
  glStencilOp(GL_KEEP, GL_KEEP, GL_REPLACE);
  RenderMirrorObject();
```

У результаті ми отримали:

– у буфері кадру - коректно намальована сцена, за винятком області дзеркала;

– в області дзеркала (там, де ми хочемо бачити відбиття) значення буфера маски дорівнює 1.

На третьому етапі потрібно намалювати сцену, відбиту щодо площини дзеркального об'єкта.Спочатку набудовуємо матрицю відбиття. Матриця відбиття повинна дзеркально відбивати всю геометрію щодо площини, у якій лежить об'єкт-дзеркало. Її можна отримати, наприклад, за допомогою такої функції :

```
  void
  reflectionmatrix(GLfloat reflection_matrix[4][4],
                GLfloat plane_point[3],
                Glfloat plane_normal[3])
  {
    GLfloat* p;
    GLfloat* v;
  float pv;
  GLfloat* p = (Glfloat*)plane_point;
  Glfloat* v = (Glfloat*)plane_normal;
    float pv = p[0]*v[0]+p[1]*v[1]+p[2]*v[2];
    reflection_matrix[0][0] = 1 - 2 * v[0] * v[0];
    reflection_matrix[1][0] = - 2 * v[0] * v[1];
    reflection_matrix[2][0] = - 2 * v[0] * v[2];
    reflection_matrix[3][0] = 2 * pv * v[0];
    reflection_matrix[0][1] = - 2 * v[0] * v[1];
    reflection_matrix[1][1] = 1- 2 * v[1] * v[1];
```


Під час візуалізації сцени краще не малювати об'єкти, які потім стануть дзеркальними.На другому етапі необхідно обмежити подальше виведення проекцією дзеркального об'єкта на екран. Для цього набудовуємо буфер маски й малюємо дзеркало

```
  glEnable(GL_STENCIL_TEST);
  /* умова завжди виконана й значення в буфері буде
дорівнює 1*/
  glStencilFunc(GL_ALWAYS, 1, 0);
  glStencilOp(GL_KEEP, GL_KEEP, GL_REPLACE);
  RenderMirrorObject();
```

У результаті ми отримали:

– у буфері кадру - коректно намальована сцена, за винятком області дзеркала;

– в області дзеркала (там, де ми хочемо бачити відбиття) значення буфера маски дорівнює 1.

На третьому етапі потрібно намалювати сцену, відбиту щодо площини дзеркального об'єкта.Спочатку набудовуємо матрицю відбиття. Матриця відбиття повинна дзеркально відбивати всю геометрію щодо площини, у якій лежить об'єкт-дзеркало. Її можна отримати, наприклад, за допомогою такої функції :

```
  void
  reflectionmatrix(GLfloat reflection_matrix[4][4],
                GLfloat plane_point[3],
                Glfloat plane_normal[3])
  {
    GLfloat* p;
    GLfloat* v;
  float pv;
  GLfloat* p = (Glfloat*)plane_point;
  Glfloat* v = (Glfloat*)plane_normal;
    float pv = p[0]*v[0]+p[1]*v[1]+p[2]*v[2];
    reflection_matrix[0][0] = 1 - 2 * v[0] * v[0];
    reflection_matrix[1][0] = - 2 * v[0] * v[1];
    reflection_matrix[2][0] = - 2 * v[0] * v[2];
    reflection_matrix[3][0] = 2 * pv * v[0];
    reflection_matrix[0][1] = - 2 * v[0] * v[1];
    reflection_matrix[1][1] = 1- 2 * v[1] * v[1];
```

```
  reflection_matrix[2][1] = - 2 * v[1] * v[2];
  reflection_matrix[3][1] = 2 * pv * v[1];
  reflection_matrix[0][2] = - 2 * v[0] * v[2];
  reflection_matrix[1][2] = - 2 * v[1] * v[2];
  reflection_matrix[2][2] = 1 - 2 * v[2] * v[2];
  reflection_matrix[3][2] = 2 * pv * v[2];
  reflection_matrix[0][3] = 0;
  reflection_matrix[1][3] = 0;
  reflection_matrix[2][3] = 0;
  reflection_matrix[3][3] = 1;
}
```

і малюємо сцену ще раз (без дзеркальних об'єктів)

```
glPushMatrix();
glMultMatrixf((float *)reflection_matrix);
RenderScene();
GlPopMatrix();
```

Нарешті, відключаємо маскування

```
glDisable(GL_STENCIL_TEST);
```

Після цього можна ще раз вивести дзеркальний об'єкт, наприклад, з альфа-змішуванням – для створення ефекту затуманення дзеркала й т.д. Існує кілька модифікацій цього алгоритму, що відрізняються послідовністю дій, обмеженнями на геометрію й т.д.

Контрольні питання

1. В чому полягає необхідність створення стандартної графічної бібліотеки?
2. Що в OpenGL є атомарним графічним примітивом?
3. Перерахуйте способи зміни положення спостерігача в OpenGL.
4. Дайте характеристику структури команд в OpenGL.
5. Поясніть різницю між локальними та безмежно віддаленими джерелами освітлення.
6. Для чого використовуються рівні деталізації текстури (mip-mapping)?
7. Дайте характеристику структури GLUT-додатка.


```
  reflection_matrix[2][1] = - 2 * v[1] * v[2];
  reflection_matrix[3][1] = 2 * pv * v[1];
  reflection_matrix[0][2] = - 2 * v[0] * v[2];
  reflection_matrix[1][2] = - 2 * v[1] * v[2];
  reflection_matrix[2][2] = 1 - 2 * v[2] * v[2];
  reflection_matrix[3][2] = 2 * pv * v[2];
  reflection_matrix[0][3] = 0;
  reflection_matrix[1][3] = 0;
  reflection_matrix[2][3] = 0;
  reflection_matrix[3][3] = 1;
}
```

і малюємо сцену ще раз (без дзеркальних об'єктів)

```
glPushMatrix();
glMultMatrixf((float *)reflection_matrix);
RenderScene();
GlPopMatrix();
```

Нарешті, відключаємо маскування

```
glDisable(GL_STENCIL_TEST);
```

Після цього можна ще раз вивести дзеркальний об'єкт, наприклад, з альфа-змішуванням – для створення ефекту затуманення дзеркала й т.д. Існує кілька модифікацій цього алгоритму, що відрізняються послідовністю дій, обмеженнями на геометрію й т.д.

Контрольні питання

1. В чому полягає необхідність створення стандартної графічної бібліотеки?
2. Що в OpenGL є атомарним графічним примітивом?
3. Перерахуйте способи зміни положення спостерігача в OpenGL.
4. Дайте характеристику структури команд в OpenGL.
5. Поясніть різницю між локальними та безмежно віддаленими джерелами освітлення.
6. Для чого використовуються рівні деталізації текстури (mip-mapping)?
7. Дайте характеристику структури GLUT-додатка.

Глава 7
Характеристика основних можливостей пакета растрової графіки

7.1. Основні можливості та інструменти

Серед усіх програм для обробки растрової графіки особливе місце займає пакет *Photoshop* компанії *Adobe* [8, 10, 25]. По суті справи, сьогодні він є стандартом у комп'ютерній графіці, і всі інші програми порівнюють саме з ним (рис. 7.1).

Головні елементи управління програми Adobe Photoshop зосереджені в рядку меню, на панелі інструментів і панелі властивостей. Особливу групу становлять діалогові вікна – *інструментальні палітри.* Далі ми розглянемо функції перерахованих засобів.

Первинне отримання оригіналу відбувається через меню Файл командою Открыть або командою Импорт. Імпортом називають отримання зображення від зовнішнього джерела – сканера, цифрової фотокамери або з документа Adobe PDF з вбудованою графікою. Зв'язок графічного редактора із зовнішніми пристроями забезпечується через програмний інтерфейс TWAIN, що встановлює стандарт на параметри обміну даними з джерелами зображень.

Перш ніж почати операції з оригіналом зображення, слід з'ясувати його параметри. Для цього командою Изображение ►Размер изображения відкривають діалогове вікно Размер изображения. У групах Количество пикселей і *Размер* документа наведені ширина і висота оригіналу в пікселах і сантиметрах відповідно, а також розділення. Від встановлених значень залежать розмір і якість зображення.

Панель інструментів (рис. 7.2) є одним з основних засобів для роботи із зображеннями. Більшість інструментів, зображених на панелі,

Глава 7
Характеристика основних можливостей пакета растрової графіки

7.1. Основні можливості та інструменти

Серед усіх програм для обробки растрової графіки особливе місце займає пакет *Photoshop* компанії *Adobe* [8, 10, 25]. По суті справи, сьогодні він є стандартом у комп'ютерній графіці, і всі інші програми порівнюють саме з ним (рис. 7.1).

Головні елементи управління програми Adobe Photoshop зосереджені в рядку меню, на панелі інструментів і панелі властивостей. Особливу групу становлять діалогові вікна – *інструментальні палітри.* Далі ми розглянемо функції перерахованих засобів.

Первинне отримання оригіналу відбувається через меню Файл командою Открыть або командою Импорт. Імпортом називають отримання зображення від зовнішнього джерела – сканера, цифрової фотокамери або з документа Adobe PDF з вбудованою графікою. Зв'язок графічного редактора із зовнішніми пристроями забезпечується через програмний інтерфейс TWAIN, що встановлює стандарт на параметри обміну даними з джерелами зображень.

Перш ніж почати операції з оригіналом зображення, слід з'ясувати його параметри. Для цього командою Изображение ►Размер изображения відкривають діалогове вікно Размер изображения. У групах Количество пикселей і *Размер* документа наведені ширина і висота оригіналу в пікселах і сантиметрах відповідно, а також розділення. Від встановлених значень залежать розмір і якість зображення.

Панель інструментів (рис. 7.2) є одним з основних засобів для роботи із зображеннями. Більшість інструментів, зображених на панелі,

має альтернативні варіанти. Їх значки помічені маленьким трикутником (кнопка розгортання). При натисненні правою кнопкою миші на такому значку відкривається лінійка значків з варіантами інструменту.

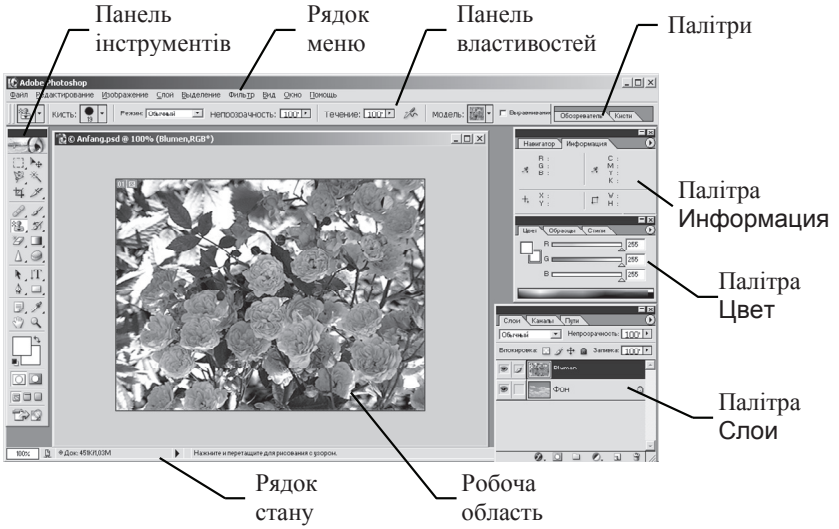


Рис. 7.1. Робоче вікно графічного редактора Adobe Photoshop

Панель інструментів розділена на області, в яких згруповані засоби для редагування певних властивостей зображення й елементи управління деякими параметрами програми. Всього таких областей вісім:

- інструменти для роботи з об’єктами;
- інструменти для малювання і ретуші;
- інструменти для створення нових об’єктів;
- допоміжні інструменти;
- засоби вибору кольорів;
- засоби управління поданням маски;
- засоби управління інтерфейсом програми;
- кнопка переходу до програми Adobe ImageReady.

має альтернативні варіанти. Їх значки помічені маленьким трикутником (кнопка розгортання). При натисненні правою кнопкою миші на такому значку відкривається лінійка значків з варіантами інструменту.

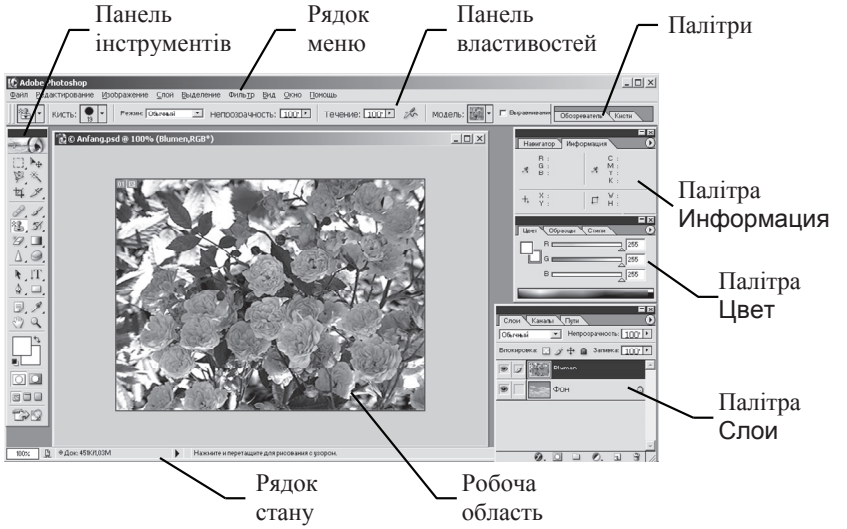


Рис. 7.1. Робоче вікно графічного редактора Adobe Photoshop

Панель інструментів розділена на області, в яких згруповані засоби для редагування певних властивостей зображення й елементи управління деякими параметрами програми. Всього таких областей вісім:

- інструменти для роботи з об’єктами;
- інструменти для малювання і ретуші;
- інструменти для створення нових об’єктів;
- допоміжні інструменти;
- засоби вибору кольорів;
- засоби управління поданням маски;
- засоби управління інтерфейсом програми;
- кнопка переходу до програми Adobe ImageReady.

Інструменти для роботи з об'єктами. Для роботи з об'єктами призначена група значків, яка об'єднує інструменти груп Область, Перемещение, Лассо, Волшебная палочка, Кадрирование і Срез. Інструментами Область і Лассо виділяють ділянку зображення, обмежену геометричною фігурою.

Рис. 7.2. Панель інструментів програми Adobe Photoshop

Інструмент Волшебная палочка здійснює вибірку області за принципом колірного (або ахроматичного) збігу в рамках меж обхвату, встановлених користувачем. Ці інструменти застосовуються для виконання операцій обведення контурів об'єктів зображення.

Інструменти для роботи з об'єктами. Для роботи з об'єктами призначена група значків, яка об'єднує інструменти груп Область, Перемещение, Лассо, Волшебная палочка, Кадрирование і Срез. Інструментами Область і Лассо виділяють ділянку зображення, обмежену геометричною фігурою.

Рис. 7.2. Панель інструментів програми Adobe Photoshop

Інструмент Волшебная палочка здійснює вибірку області за принципом колірного (або ахроматичного) збігу в рамках меж обхвату, встановлених користувачем. Ці інструменти застосовуються для виконання операцій обведення контурів об'єктів зображення.

Інструментом Перемещение пересувають виділені області і копіюють їх. Інструмент Срез дозволяє видалити області (поля) малюнка, а інструмент Кадрирование використовується, коли великий малюнок готується для відображення в Інтернеті. За допомогою цього інструменту його можна автоматично розділити на фрагменти прийнятного розміру, кожний з яких завантажуватиметься через мережу окремо.

Інструменти для малювання і ретуші. Наступна група інструментів призначена для малювання і ретуші. Сюди відноситься Лечащая кисть і Вставка, Кисть і Карандаш, Клонированный штамп і Штамп, Историческая кисть і Узорная кисть, Ластик, Заливка і Градиент. Для ретуші також використовують інструменти маніпуляції з різкістю Размытие, Резкость, Палец або яскравістю Осветление, Затемнение, Губка.

Інструмент Клонированный штамп дозволяє виконувати копіювання вибраних ділянок зображення за кожним клацанням миші. Штамп як наповнювач набивання використовує узори з палітри Pattern. Лечащая кисть схожа за способом дії на Клонированный штамп, проте наближає яскравість і колір копійованої області до параметрів пікселів цільової області. Інструмент Вставка дозволяє вибрати довільну область-джерело перетягуванням миші, а потім перемістити її в потрібне місце. При цьому кольори і яскравість області-джерела приводяться у відповідність до параметрів пікселів цільової області. Историческая кисть приводить параметри пікселів в області дії до початкового стану, який вони мали до редагування.

За допомогою інструменту Заливка виконують заповнення виділених ділянок зображення одним кольором. Плавний перехід між кольорами забезпечує інструмент Градиент. Інструменти з альтернативним вибором Размытие/Резкость дозволяють змінювати ці параметри на окремих ділянках зображення, а інструменти Осветление/Затемнение/Губка служать для місцевої корекції яскравості і колірної насиченості.

Інструменти для створення нових об'єктів. Третя група інструментів призначена для створення нових об'єктів і редагування існуючих. Інструмент Перо дозволяє малювати плавні криволінійні контури. Інструментом Текст виконують написи. Інструменти для

Інструментом Перемещение пересувають виділені області і копіюють їх. Інструмент Срез дозволяє видалити області (поля) малюнка, а інструмент Кадрирование використовується, коли великий малюнок готується для відображення в Інтернеті. За допомогою цього інструменту його можна автоматично розділити на фрагменти прийнятного розміру, кожний з яких завантажуватиметься через мережу окремо.

Інструменти для малювання і ретуші. Наступна група інструментів призначена для малювання і ретуші. Сюди відноситься Лечащая кисть і Вставка, Кисть і Карандаш, Клонированный штамп і Штамп, Историческая кисть і Узорная кисть, Ластик, Заливка і Градиент. Для ретуші також використовують інструменти маніпуляції з різкістю Размытие, Резкость, Палец або яскравістю Осветление, Затемнение, Губка.

Інструмент Клонированный штамп дозволяє виконувати копіювання вибраних ділянок зображення за кожним клацанням миші. Штамп як наповнювач набивання використовує узори з палітри Pattern. Лечащая кисть схожа за способом дії на Клонированный штамп, проте наближає яскравість і колір копійованої області до параметрів пікселів цільової області. Інструмент Вставка дозволяє вибрати довільну область-джерело перетягуванням миші, а потім перемістити її в потрібне місце. При цьому кольори і яскравість області-джерела приводяться у відповідність до параметрів пікселів цільової області. Историческая кисть приводить параметри пікселів в області дії до початкового стану, який вони мали до редагування.

За допомогою інструменту Заливка виконують заповнення виділених ділянок зображення одним кольором. Плавний перехід між кольорами забезпечує інструмент Градиент. Інструменти з альтернативним вибором Размытие/Резкость дозволяють змінювати ці параметри на окремих ділянках зображення, а інструменти Осветление/Затемнение/Губка служать для місцевої корекції яскравості і колірної насиченості.

Інструменти для створення нових об'єктів. Третя група інструментів призначена для створення нових об'єктів і редагування існуючих. Інструмент Перо дозволяє малювати плавні криволінійні контури. Інструментом Текст виконують написи. Інструменти для

створення фігур і ліній служать для малювання векторних об'єктів. Для вибірки цілих об'єктів або вузлів контуру служать інструменти Выделение пути і Прямой выбор відповідно.

Допоміжні інструменти. Група допоміжних інструментів об'єднує підсобні засоби. Інструментом Рука переміщають видиму область по зображенню, а інструмент Лупа призначений для збільшення/зменшення зображення у видимій області. Інструмент Пипетка служить для точного визначення кольору в будь-якій точці зображення й ухвалення його як зразка. У цьому ж наборі розміщений інструмент Измерение**.** Інструмент Заметки дозволяє прив'язати до малюнка текстові зауваження. Звукові коментарі створюють за допомогою альтернативного інструменту Звуковые заметки.

У нижній частині інструментальної панелі розташовані засоби для роботи з кольором, масками, формою відображення елементів управління програми. Засіб управління кольором показує основні кольори фону і переднього плану. У лівому нижньому куті розташований значок, клацання на якому встановлює кольори, прийняті за замовчуванням для фону і переднього плану. Елемент управління Маска дозволяє працювати в режимах Стандартный або Быстрая маска. Засіб управління режимом відображення дозволяє перемикатися між режимом стандартного вікна**,** повноекранним режимом з панеллю меню та повноекранним режимом, у якому панель меню згортається і розміщується у вигляді кнопки у верхній частині панелі інструментів. Останнім елементом управління на панелі інструментів є кнопка Переход к программе ImageReady.

Найважливішим засобом управління параметрами інструментів є інтерактивна панель властивостей. Під час вибору будь-якого інструменту автоматично з'являється відповідна до його властивостей палітра з необхідними елементами управління. Всього в програмі Adobe Photoshop– більше десяти інструментальних палітр.

Управління відображенням палітр здійснюється з меню Окно за допомогою прапорців меню, які відповідають існуючим палітрам. Палітри, які не вживаються, можна також видалити з екрана клацанням на кнопці закривання. Клацанням на кнопці згортання палітру скорочують до розміру рядка з ярликами вкладок. Праворуч від рядка заголовка вікно палітри має кнопку, клацання на якій відкриває доступ до

створення фігур і ліній служать для малювання векторних об'єктів. Для вибірки цілих об'єктів або вузлів контуру служать інструменти Выделение пути і Прямой выбор відповідно.

Допоміжні інструменти. Група допоміжних інструментів об'єднує підсобні засоби. Інструментом Рука переміщають видиму область по зображенню, а інструмент Лупа призначений для збільшення/зменшення зображення у видимій області. Інструмент Пипетка служить для точного визначення кольору в будь-якій точці зображення й ухвалення його як зразка. У цьому ж наборі розміщений інструмент Измерение**.** Інструмент Заметки дозволяє прив'язати до малюнка текстові зауваження. Звукові коментарі створюють за допомогою альтернативного інструменту Звуковые заметки.

У нижній частині інструментальної панелі розташовані засоби для роботи з кольором, масками, формою відображення елементів управління програми. Засіб управління кольором показує основні кольори фону і переднього плану. У лівому нижньому куті розташований значок, клацання на якому встановлює кольори, прийняті за замовчуванням для фону і переднього плану. Елемент управління Маска дозволяє працювати в режимах Стандартный або Быстрая маска. Засіб управління режимом відображення дозволяє перемикатися між режимом стандартного вікна**,** повноекранним режимом з панеллю меню та повноекранним режимом, у якому панель меню згортається і розміщується у вигляді кнопки у верхній частині панелі інструментів. Останнім елементом управління на панелі інструментів є кнопка Переход к программе ImageReady.

Найважливішим засобом управління параметрами інструментів є інтерактивна панель властивостей. Під час вибору будь-якого інструменту автоматично з'являється відповідна до його властивостей палітра з необхідними елементами управління. Всього в програмі Adobe Photoshop– більше десяти інструментальних палітр.

Управління відображенням палітр здійснюється з меню Окно за допомогою прапорців меню, які відповідають існуючим палітрам. Палітри, які не вживаються, можна також видалити з екрана клацанням на кнопці закривання. Клацанням на кнопці згортання палітру скорочують до розміру рядка з ярликами вкладок. Праворуч від рядка заголовка вікно палітри має кнопку, клацання на якій відкриває доступ до

меню, що містить команди роботи з об'єктами палітри і настройки параметрів палітри. Деякі палітри мають командні кнопки, списки, що розкриваються, поля введення та інші елементи управління. Призначення конкретного елемента управління пояснює спливаючий підказ, що з'являється у разі затримки покажчика миші на елементі, який вас цікавить.

Палітра Кисти управляє настроюванням параметрів інструментів редагування.

Палітра Обозреватель надає засоби навігації по файловій системі і перегляд зображень різного формату.

Палітра Информация забезпечує інформаційну підтримку засобів відображення. На ній наведені: поточні координати покажчика миші, розмір поточної виділеної області, колірні параметри елемента зображення та інші дані.

Палітра Навигатор дозволяє проглянути різні фрагменти зображення і змінити масштаб перегляду. У вікні палітри розміщена мініатюра зображення з виділеною областю перегляду.

Палітра Цвет відображає колірні значення поточних кольорів переднього плану і фону. Повзунки на колірній лінійці відповідної колірної системи дозволяють редагувати ці параметри.

Палітра Образцы містить набір доступних кольорів. Такий набір можна завантажити і відредагувати, додаючи і видаляючи кольори. Колірний тон переднього плану і фону вибирають із запропонованого набору. У стандартному комплекті поставки програми передбачено декілька колірних наборів.

Палітра Слои призначена для управління відображенням усіх шарів зображення, починаючи з самого верхнього. Можливі визначення параметрів шарів, зміна їх порядку, операції з шарами із застосуванням різних методів.

Палітру Каналы використовують для виділення, створення, дублювання і видалення колірних каналів, визначення їх параметрів, зміни порядку, перетворення каналів у самостійні об'єкти і формування суміщених зображень з декількох каналів.

Палітра Пути містить список усіх створених контурів. Контури можна використовувати для обведення – програма Adobe Photoshop дозволяє перетворити контур у межу виділеної області.

меню, що містить команди роботи з об'єктами палітри і настройки параметрів палітри. Деякі палітри мають командні кнопки, списки, що розкриваються, поля введення та інші елементи управління. Призначення конкретного елемента управління пояснює спливаючий підказ, що з'являється у разі затримки покажчика миші на елементі, який вас цікавить.

Палітра Кисти управляє настроюванням параметрів інструментів редагування.

Палітра Обозреватель надає засоби навігації по файловій системі і перегляд зображень різного формату.

Палітра Информация забезпечує інформаційну підтримку засобів відображення. На ній наведені: поточні координати покажчика миші, розмір поточної виділеної області, колірні параметри елемента зображення та інші дані.

Палітра Навигатор дозволяє проглянути різні фрагменти зображення і змінити масштаб перегляду. У вікні палітри розміщена мініатюра зображення з виділеною областю перегляду.

Палітра Цвет відображає колірні значення поточних кольорів переднього плану і фону. Повзунки на колірній лінійці відповідної колірної системи дозволяють редагувати ці параметри.

Палітра Образцы містить набір доступних кольорів. Такий набір можна завантажити і відредагувати, додаючи і видаляючи кольори. Колірний тон переднього плану і фону вибирають із запропонованого набору. У стандартному комплекті поставки програми передбачено декілька колірних наборів.

Палітра Слои призначена для управління відображенням усіх шарів зображення, починаючи з самого верхнього. Можливі визначення параметрів шарів, зміна їх порядку, операції з шарами із застосуванням різних методів.

Палітру Каналы використовують для виділення, створення, дублювання і видалення колірних каналів, визначення їх параметрів, зміни порядку, перетворення каналів у самостійні об'єкти і формування суміщених зображень з декількох каналів.

Палітра Пути містить список усіх створених контурів. Контури можна використовувати для обведення – програма Adobe Photoshop дозволяє перетворити контур у межу виділеної області.

Палітра Действия дозволяє створювати макрокоманди – задану послідовність операцій із зображенням. Макрокоманди можна записувати, виконувати, редагувати, видаляти, зберігати у вигляді файлів.

Особливу групу програмних засобів обробки зображень становлять фільтри. Це модулі, що під'єднуються до програми, вони дозволяють обробляти зображення за заданим алгоритмом. Іноді такі алгоритми бувають дуже складними, а вікно фільтра може мати безліч параметрів, що настроюються.

Програма Photoshop містить 99 вбудованих фільтрів, розділених на 19 груп. Вони знаходяться в меню Фильтр.

Всі ці групи можна об'єднати у 4 класи:

- фільтри, які поліпшують якість зображення;
- фільтри, що додають художніх ефектів;
- фільтри, що додають спеціальних ефектів;
- фільтри, що здійснюють технічну корекцію.

7.2. Малювання та ретушування зображень

Розглянемо інструменти Photoshop для роботи з кольором. Оскільки колір оточує нас з усіх боків, дуже важливо розуміти, як використовувати інструменти Photoshop для редагування та застосування кольорів. Важливо знати, як Photoshop «бачить» колір, як використовувати інструменти даної програми для отримання необхідних результатів.

Вибір кольору

На панелі інструментів передбачено чотири елементи для управління кольором (рис.7.3) ForegroundColor(Основний колір). Вказує колір, який буде використовуватися інструментами PaintBucket, Line, Brush, а також інструментами Type після натиснення кнопки <Alt>. З цього кольору починаються всі кольорові градієнтні переходи. Усі фігури, що створюються за допомогою інструменту Shapes, зафарбовуються основним кольором. Для того щоб зафарбувати цим кольором обрану ділянку, достатньо натиснути одночасно комбінацію клавішей <Alt+Backspace> або обрати команду Edit – Fill (Редагування – виконати заливку). Для того щоб змінити

Палітра Действия дозволяє створювати макрокоманди – задану послідовність операцій із зображенням. Макрокоманди можна записувати, виконувати, редагувати, видаляти, зберігати у вигляді файлів.

Особливу групу програмних засобів обробки зображень становлять фільтри. Це модулі, що під'єднуються до програми, вони дозволяють обробляти зображення за заданим алгоритмом. Іноді такі алгоритми бувають дуже складними, а вікно фільтра може мати безліч параметрів, що настроюються.

Програма Photoshop містить 99 вбудованих фільтрів, розділених на 19 груп. Вони знаходяться в меню Фильтр.

Всі ці групи можна об'єднати у 4 класи:

- фільтри, які поліпшують якість зображення;
- фільтри, що додають художніх ефектів;
- фільтри, що додають спеціальних ефектів;
- фільтри, що здійснюють технічну корекцію.

7.2. Малювання та ретушування зображень

Розглянемо інструменти Photoshop для роботи з кольором. Оскільки колір оточує нас з усіх боків, дуже важливо розуміти, як використовувати інструменти Photoshop для редагування та застосування кольорів. Важливо знати, як Photoshop «бачить» колір, як використовувати інструменти даної програми для отримання необхідних результатів.

Вибір кольору

На панелі інструментів передбачено чотири елементи для управління кольором (рис.7.3) ForegroundColor(Основний колір). Вказує колір, який буде використовуватися інструментами PaintBucket, Line, Brush, а також інструментами Type після натиснення кнопки <Alt>. З цього кольору починаються всі кольорові градієнтні переходи. Усі фігури, що створюються за допомогою інструменту Shapes, зафарбовуються основним кольором. Для того щоб зафарбувати цим кольором обрану ділянку, достатньо натиснути одночасно комбінацію клавішей <Alt+Backspace> або обрати команду Edit – Fill (Редагування – виконати заливку). Для того щоб змінити

основний колір, необхідно клацнути маніпулятором (мишею) на відповідній кнопці (при цьому на екрані з'явиться вікно діалогу вибору кольору) або скористатися інструментом Eyedropper. Цей інструмент може «брати» колір не тільки у вікні зображення, але у будь-якій частині екрана.

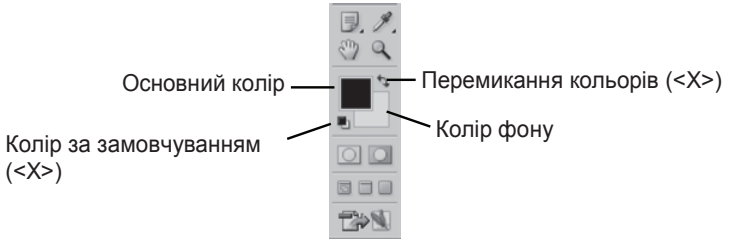


Рис. 7.3. Інструменти для управління кольором

BackgroundColor (Фоновий колір). Вказує колір, який використовується інструментом Eraser. Цим кольором завершуються всі градієнтні кольорові переходи. Зміна кольору фону відбувається аналогічно зміні основного кольору – за допомогою вікна синтезу кольору. Зафарбувати виділену ділянку кольором фону можна, натиснувши кнопку <Backspace> або <Delete>. Але якщо виділена область «плаває» або знаходиться в іншому шарі, за виключенням фонового, натиснення цих кнопок призведе не до заповнення цієї області кольором, а до її видалення.

SwitchColor (Перемикання кольорів). Клацання на цій кнопці (або натиснення клавіші <X>) змінює місцями колір фону та основний колір.

DefaultColors (Кольори за замовчуванням). Клацання на цій кнопці (або натиснення клавіші <D>) встановлює чорний основний колір та білий фон.

Робота з вікном діалогу ColorPicker

Після того як ви клацнули по кнопці основного кольору або кольору фону, відкривається вікно діалогу ColorPicker (Палітра кольорів). На рис. 7.4 показані основні елементи вікна вибору кольору.

основний колір, необхідно клацнути маніпулятором (мишею) на відповідній кнопці (при цьому на екрані з'явиться вікно діалогу вибору кольору) або скористатися інструментом Eyedropper. Цей інструмент може «брати» колір не тільки у вікні зображення, але у будь-якій частині екрана.

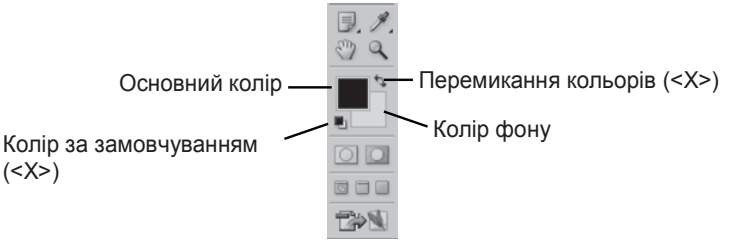


Рис. 7.3. Інструменти для управління кольором

BackgroundColor (Фоновий колір). Вказує колір, який використовується інструментом Eraser. Цим кольором завершуються всі градієнтні кольорові переходи. Зміна кольору фону відбувається аналогічно зміні основного кольору – за допомогою вікна синтезу кольору. Зафарбувати виділену ділянку кольором фону можна, натиснувши кнопку <Backspace> або <Delete>. Але якщо виділена область «плаває» або знаходиться в іншому шарі, за виключенням фонового, натиснення цих кнопок призведе не до заповнення цієї області кольором, а до її видалення.

SwitchColor (Перемикання кольорів). Клацання на цій кнопці (або натиснення клавіші <X>) змінює місцями колір фону та основний колір.

DefaultColors (Кольори за замовчуванням). Клацання на цій кнопці (або натиснення клавіші <D>) встановлює чорний основний колір та білий фон.

Робота з вікном діалогу ColorPicker

Після того як ви клацнули по кнопці основного кольору або кольору фону, відкривається вікно діалогу ColorPicker (Палітра кольорів). На рис. 7.4 показані основні елементи вікна вибору кольору.

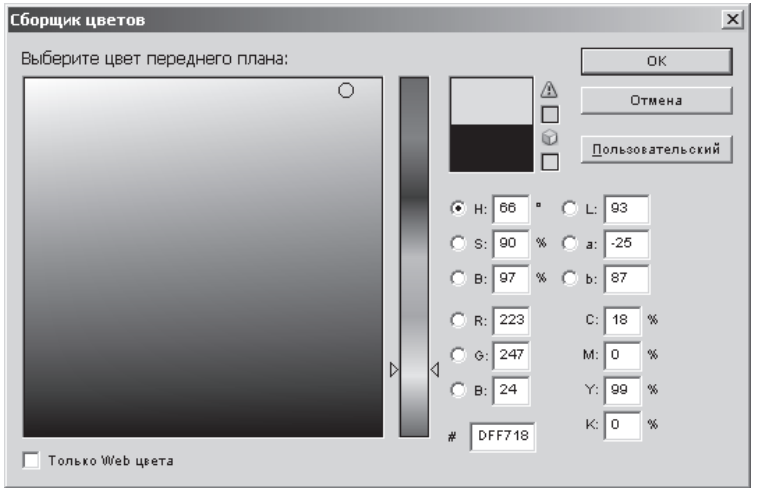


Рис. 7.4. Використання елементів вікна діалогу для вибору одного з 16 мільйонів кольорів

Кольорова лінійка.Вказує колір, який ви бажаєте обрати. Переміщення трикутників уверх або вниз дозволяє обрати колір у 8-розрядному діапазоні. Кольори лінійки відповідають обраному параметру. Наприклад, якщо обрати значення H (Hue – відтінок), то на кольоровій лінійці буде зображений повний діапазон відтінків певного кольору. Якщо обрати значення S (Saturation – насиченість), то у верхній частині лінійки буде знаходитися найбільш насичений, а в нижній – найменш насичений відтінок кольору. Значення B (Brightness) – яскравість, параметри R (Red), G (Green), B (Blue) дозволяють керувати окремими складовими – червоною, зеленою та синьою.

Кольорове поле. Це 16-розрядний діапазон варіантів поточного кольору, що вказаний на кольоровій лінійці. Для вибору кольору достатньо клацнути мишею в цьому полі.

Поточний колір. Колір, який обрали з кольорового поля, з'являється у верхньому прямокутнику праворуч від кольорової лінійки. Якщо натиснути кнопку OK або натиснути клавішу <Enter>, цей колір буде обраний як основний або фоновий.

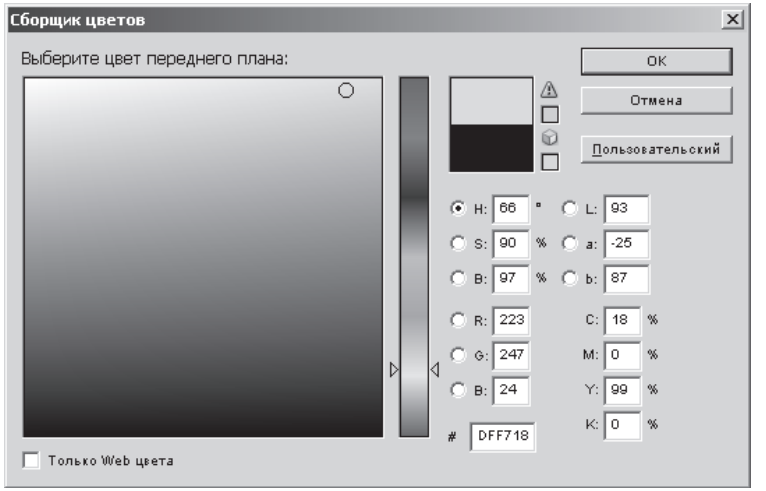


Рис. 7.4. Використання елементів вікна діалогу для вибору одного з 16 мільйонів кольорів

Кольорова лінійка.Вказує колір, який ви бажаєте обрати. Переміщення трикутників уверх або вниз дозволяє обрати колір у 8-розрядному діапазоні. Кольори лінійки відповідають обраному параметру. Наприклад, якщо обрати значення H (Hue – відтінок), то на кольоровій лінійці буде зображений повний діапазон відтінків певного кольору. Якщо обрати значення S (Saturation – насиченість), то у верхній частині лінійки буде знаходитися найбільш насичений, а в нижній – найменш насичений відтінок кольору. Значення B (Brightness) – яскравість, параметри R (Red), G (Green), B (Blue) дозволяють керувати окремими складовими – червоною, зеленою та синьою.

Кольорове поле. Це 16-розрядний діапазон варіантів поточного кольору, що вказаний на кольоровій лінійці. Для вибору кольору достатньо клацнути мишею в цьому полі.

Поточний колір. Колір, який обрали з кольорового поля, з'являється у верхньому прямокутнику праворуч від кольорової лінійки. Якщо натиснути кнопку OK або натиснути клавішу <Enter>, цей колір буде обраний як основний або фоновий.

Попередній колір. Нижній прямокутник, який розміщений праворуч від кольорової лінійки, демонструє колір, встановлений до відкриття вікна вибору кольору.

OnlyWebColors (Тільки Web-кольори).Цей параметр змінює зображення даних у полі Color таким чином, щоб відображалися лише кольори кольорової палітри Web.

Робота в різних кольорових режимах

Чотири набори числових полів у вікні синтезу кольору відповідають чотирьом різним кольоровим режимам. *Кольорова модель* – це спосіб указання кольору під час виведення його на екран або до друку.

За межами вікна діалогу вибору кольору перемикання між кольоровими режимами виконується за допомогою команд меню Image – Mode (Зображення – Режим). Перемикаючи режим за допомогою цього меню, користувач змінює кількість кольорів у зображенні на декілька сотень, а іноді і тисяч – в основному у бік зменшення. Єдиним виключенням є режим Lab, який відкриває доступ до всіх кольорів режимів CMYK та RGB.

Модель RGB іноді називають адитивною первинною моделлю, оскільки при збільшенні яскравості окремих кольорів результуючий колір теж стає більш яскравим. Ця модель використовується в усіх моніторах, проекторах та інших пристроях, які випромінюють або фільтрують світло, включаючи телевізори, кінопроектори, кольорові прожектори тощо. З точки зору редагування зображень на екрані кольорова модель RGB найбільш зручна, оскільки забезпечує доступ до всіх кольорів, які можуть вивести на екран 24-розрядні кольорові відеоплати. Більше того, зображення, яке створено у кольоровому режимі RGB, можна записати на диск у будь-якому з графічних форматів, що підтримуються програмою Photoshop, окрім формату GIF.

Недолік режиму RGB полягає в тому, що не всі кольори, що створюються в ньому, можна вивести для друку. Останнє призводить до того, що цей режим використовується для зображень, які не виводяться на друк. Уникнути втрати кольорів можна, використовуючи режим CMYK, але при цьому кількість кольорів буде строго обмежена.

Попередній колір. Нижній прямокутник, який розміщений праворуч від кольорової лінійки, демонструє колір, встановлений до відкриття вікна вибору кольору.

OnlyWebColors (Тільки Web-кольори).Цей параметр змінює зображення даних у полі Color таким чином, щоб відображалися лише кольори кольорової палітри Web.

Робота в різних кольорових режимах

Чотири набори числових полів у вікні синтезу кольору відповідають чотирьом різним кольоровим режимам. *Кольорова модель* – це спосіб указання кольору під час виведення його на екран або до друку.

За межами вікна діалогу вибору кольору перемикання між кольоровими режимами виконується за допомогою команд меню Image – Mode (Зображення – Режим). Перемикаючи режим за допомогою цього меню, користувач змінює кількість кольорів у зображенні на декілька сотень, а іноді і тисяч – в основному у бік зменшення. Єдиним виключенням є режим Lab, який відкриває доступ до всіх кольорів режимів CMYK та RGB.

Модель RGB іноді називають адитивною первинною моделлю, оскільки при збільшенні яскравості окремих кольорів результуючий колір теж стає більш яскравим. Ця модель використовується в усіх моніторах, проекторах та інших пристроях, які випромінюють або фільтрують світло, включаючи телевізори, кінопроектори, кольорові прожектори тощо. З точки зору редагування зображень на екрані кольорова модель RGB найбільш зручна, оскільки забезпечує доступ до всіх кольорів, які можуть вивести на екран 24-розрядні кольорові відеоплати. Більше того, зображення, яке створено у кольоровому режимі RGB, можна записати на диск у будь-якому з графічних форматів, що підтримуються програмою Photoshop, окрім формату GIF.

Недолік режиму RGB полягає в тому, що не всі кольори, що створюються в ньому, можна вивести для друку. Останнє призводить до того, що цей режим використовується для зображень, які не виводяться на друк. Уникнути втрати кольорів можна, використовуючи режим CMYK, але при цьому кількість кольорів буде строго обмежена.

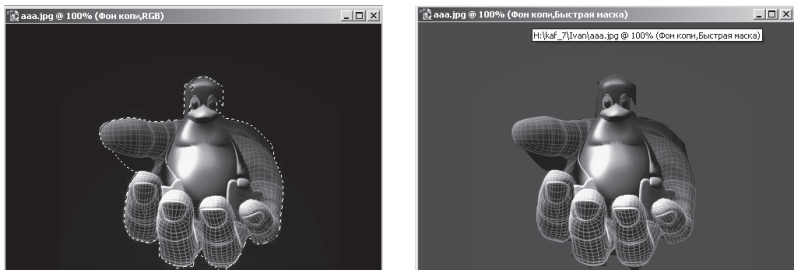
7.3. Виділені області, маски та фільтри

Маски та виділення фрагментів

Більшість користувачів Photoshop не застосовують у своїй роботі такий важливий інструмент, як маски. Зазвичай вони обмежуються такими інструментами, як Lasso, MagicWand та Pen, які ускладнюють процедуру отримання якісного зображення.

На відміну від останніх, маскам притаманні всі переваги інших інструментів. За допомогою масок можна створювати контури виділення довільної форми, виділяти області кольору, з високою точністю виділяти потрібні фрагменти зображення. Більше того, для масок характерна відсутність недоліків інших інструментів виділення. Вони дозволяють ураховувати різницю в рівнях фокусування та здійснюють повний контроль над тим, як будуть виглядати контури зображення, а виділені з їх допомогою фрагменти зображення є настільки ж природними, як і вихідне зображення.

Маска – це контур виділення,поданий у вигляді напівтонового зображення. Виділені області зображуються білим кольором, невиділені – чорним кольором, а частково виділені області зображення – напівтонами.



а б

Рис. 7.5. Застосування маски: а – контур розтушованої області; б – еквівалентна йому маска

На рис. 7.5 наведена виділена область та відповідна їй маска. З лівого боку показана овальна виділена область, по відношенню до якої була виконана операція інвертування (команда Image – Adjust– Invert (Зображення – Корекція – Інверсія)).

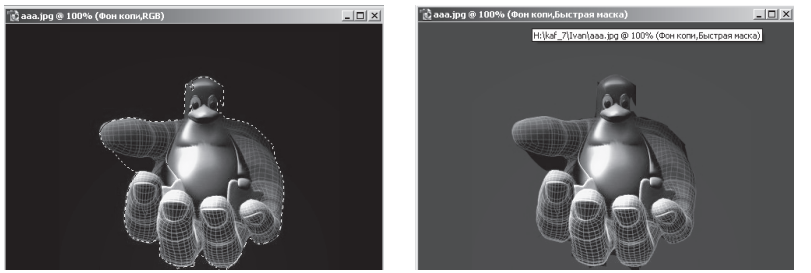
7.3. Виділені області, маски та фільтри

Маски та виділення фрагментів

Більшість користувачів Photoshop не застосовують у своїй роботі такий важливий інструмент, як маски. Зазвичай вони обмежуються такими інструментами, як Lasso, MagicWand та Pen, які ускладнюють процедуру отримання якісного зображення.

На відміну від останніх, маскам притаманні всі переваги інших інструментів. За допомогою масок можна створювати контури виділення довільної форми, виділяти області кольору, з високою точністю виділяти потрібні фрагменти зображення. Більше того, для масок характерна відсутність недоліків інших інструментів виділення. Вони дозволяють ураховувати різницю в рівнях фокусування та здійснюють повний контроль над тим, як будуть виглядати контури зображення, а виділені з їх допомогою фрагменти зображення є настільки ж природними, як і вихідне зображення.

Маска – це контур виділення,поданий у вигляді напівтонового зображення. Виділені області зображуються білим кольором, невиділені – чорним кольором, а частково виділені області зображення – напівтонами.



а б

Рис. 7.5. Застосування маски: а – контур розтушованої області; б – еквівалентна йому маска

На рис. 7.5 наведена виділена область та відповідна їй маска. З лівого боку показана овальна виділена область, по відношенню до якої була виконана операція інвертування (команда Image – Adjust– Invert (Зображення – Корекція – Інверсія)).

Праворуч – та ж область, але зображена у вигляді маски. Оскільки ця виділена область має різко окреслені краї, її маска також має різко окреслені контури.

Оскільки маска є незалежним та півтоновим зображенням, її можна відредагувати за допомогою будь-якого інструменту малювання та редагування, будь-якого фільтра, будь-яких параметрів кольорової корекції та практично за допомогою будь-якої іншої функції.

Найбільш природним середовищем для побудови маски є *режим швидкого маскування*. В цьому режимі виділена область зображується у вигляді плівки світло-червоного кольору. Всі невиділені області на зображенні вкриті плівкою, а на виділених областях таке покриття відсутнє.

Корегуюча фільтрація

У програмі Photoshop фільтри призначені для створення різноманітних ефектів. Названі за аналогією з фотофільтрами, які застосовуються в основному для корегування освітлення та перспективи, фільтри у Photoshop дозволяють розв'язувати більш складні завдання. Вам надається можливість збільшити різкість зображення, накласти випадкові пікселі, додати ефект об'єму або перетворити зображення в мозаїку. Завдяки використанню фільтрів вам доступні сотні спеціальних ефектів.

Усі фільтри спеціальних ефектів Photoshop доступні за допомогою команд меню Filter. Команди поділяються на дві групи – корегуючі та спотворюючі.

Корегуючі фільтри. Корегуючі фільтри щоденно використовуються для редагування сканованих зображень та підготування зображень до друку або перегляду на екрані.

У багатьох випадках робота корегуючого фільтра є настільки витонченою, що її складно помітити. Як показано на рис.7.6, ці фільтри дозволяють змінити різкість зображення, обробити кольорові переходи та усереднити кольори сусідніх пікселів. Фільтри доступні у підменю Filter – Blur (Фільтр – *Розмиття*), Filter – Noise (Фільтр – *Шум*), Filter – Sharpen (Фільтр – *Різкість*) та Filter – Other (Фільтр – *Інші*).

Спотворюючі фільтри. Спотворюючі фільтри надають більшого ефекту, тому, якщо використовувати їх невірно, вони можуть повністю знищити зображення, тобто результат застосування фільтра буде більш помітний, ніж саме зображення.

Праворуч – та ж область, але зображена у вигляді маски. Оскільки ця виділена область має різко окреслені краї, її маска також має різко окреслені контури.

Оскільки маска є незалежним та півтоновим зображенням, її можна відредагувати за допомогою будь-якого інструменту малювання та редагування, будь-якого фільтра, будь-яких параметрів кольорової корекції та практично за допомогою будь-якої іншої функції.

Найбільш природним середовищем для побудови маски є *режим швидкого маскування*. В цьому режимі виділена область зображується у вигляді плівки світло-червоного кольору. Всі невиділені області на зображенні вкриті плівкою, а на виділених областях таке покриття відсутнє.

Корегуюча фільтрація

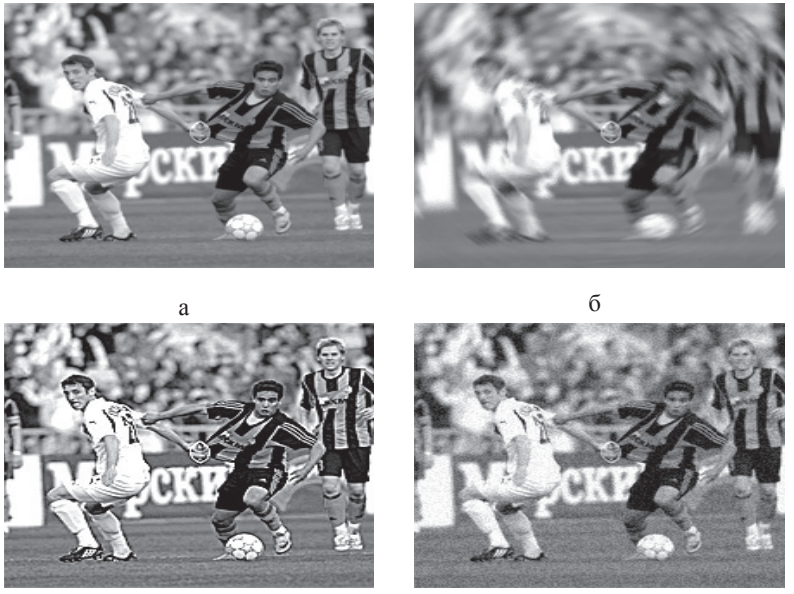
У програмі Photoshop фільтри призначені для створення різноманітних ефектів. Названі за аналогією з фотофільтрами, які застосовуються в основному для корегування освітлення та перспективи, фільтри у Photoshop дозволяють розв'язувати більш складні завдання. Вам надається можливість збільшити різкість зображення, накласти випадкові пікселі, додати ефект об'єму або перетворити зображення в мозаїку. Завдяки використанню фільтрів вам доступні сотні спеціальних ефектів.

Усі фільтри спеціальних ефектів Photoshop доступні за допомогою команд меню Filter. Команди поділяються на дві групи – корегуючі та спотворюючі.

Корегуючі фільтри. Корегуючі фільтри щоденно використовуються для редагування сканованих зображень та підготування зображень до друку або перегляду на екрані.

У багатьох випадках робота корегуючого фільтра є настільки витонченою, що її складно помітити. Як показано на рис.7.6, ці фільтри дозволяють змінити різкість зображення, обробити кольорові переходи та усереднити кольори сусідніх пікселів. Фільтри доступні у підменю Filter – Blur (Фільтр – *Розмиття*), Filter – Noise (Фільтр – *Шум*), Filter – Sharpen (Фільтр – *Різкість*) та Filter – Other (Фільтр – *Інші*).

Спотворюючі фільтри. Спотворюючі фільтри надають більшого ефекту, тому, якщо використовувати їх невірно, вони можуть повністю знищити зображення, тобто результат застосування фільтра буде більш помітний, ніж саме зображення.



а б в г

Рис. 7.6. Результати застосування корегуючих фільтрів:
а – вихідне зображення; б – фільтр *Радіальне розмиття*;
в – фільтр *Різкість*; г – фільтр *Шум*

Більшість спотворюючих фільтрів доступна з підменю Filter – Distort (Фільтр – *Спотворення*), Filter – Pixelate (Фільтр – *Оформлення*), Filter – Render (Фільтр – *Рендерінг*) та Filter – Stylize (Фільтр – *Стилізація*). Декілька прикладів спотворення зображення наведено на рис. 7.7.

Деструктивні фільтри дозволяють отримувати доволі цікаві ефекти, тому багато користувачів починають знайомство з Photoshop саме з них. Ці фільтри порушують вихідну композицію зображення, тому слід використовувати їх дуже обережно. Принципи побудови фільтрів засновані на використанні матричних операцій та більш докладно розглянуті в [10].

Фільтри ефектів. Фільтри ефектів відносяться до групи спотворюючих фільтрів, але розглядаються окремо. Приклад застосування цих фільтрів наведено на рис. 7.8.

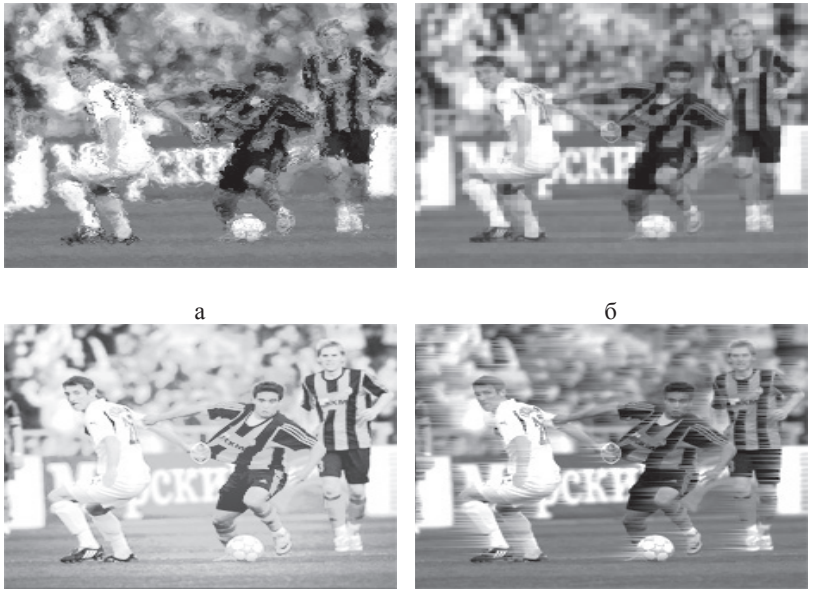
а б в г

Рис. 7.6. Результати застосування корегуючих фільтрів:
а – вихідне зображення; б – фільтр *Радіальне розмиття*;
в – фільтр *Різкість*; г – фільтр *Шум*

Більшість спотворюючих фільтрів доступна з підменю Filter – Distort (Фільтр – *Спотворення*), Filter – Pixelate (Фільтр – *Оформлення*), Filter – Render (Фільтр – *Рендерінг*) та Filter – Stylize (Фільтр – *Стилізація*). Декілька прикладів спотворення зображення наведено на рис. 7.7.

Деструктивні фільтри дозволяють отримувати доволі цікаві ефекти, тому багато користувачів починають знайомство з Photoshop саме з них. Ці фільтри порушують вихідну композицію зображення, тому слід використовувати їх дуже обережно. Принципи побудови фільтрів засновані на використанні матричних операцій та більш докладно розглянуті в [10].

Фільтри ефектів. Фільтри ефектів відносяться до групи спотворюючих фільтрів, але розглядаються окремо. Приклад застосування цих фільтрів наведено на рис. 7.8.



а б

в г

Рис. 7.7. Результати застосування спотворюючих фільтрів:
а – фільтр *Спотворення/Скло*; б – фільтр *Оформлення/Мозаїка*;
в – фільтр *Рендерінг/Освітлення*; г – фільтр *Стилізація/Вітер*

Рис. 7.8. Застосування фільтра *Перехресні штрихи*

а б

в г

Рис. 7.7. Результати застосування спотворюючих фільтрів:
а – фільтр *Спотворення/Скло*; б – фільтр *Оформлення/Мозаїка*;
в – фільтр *Рендерінг/Освітлення*; г – фільтр *Стилізація/Вітер*

Рис. 7.8. Застосування фільтра *Перехресні штрихи*

Кількість фільтрів та способів їх використання, що пропонує Photoshop, дуже велика, тому потребує окремого вивчення за допомогою додаткової літератури.

7.4. Шари, об'єкти, текст

Для користувачів – початківців знайомство з Photoshop необхідно розпочинати саме з цього питання. Що таке шари, для чого вони призначені та чому вони необхідні? Спробуємо з цими питаннями розібратися.

Основною властивістю шарів є їх непостійність. Кожен шар у композиції не залежить від решти, тому існує можливість у будь-який час робити зміни у зображенні. Зверніть увагу на рис. 7.9, а. Дане зображення складається лише з трикутника. Надалі були додані коло та квадрат з відповідним текстом та різною прозорістю (рис. 7.9,б та в). Все це досягається за рахунок використання шарів. Причому, якщо ви забажаєте змінити розмір, колір або форму будь-якої геометричної фігури, то все це легко зробити під час роботи з відповідними шарами. Якщо б не було шарів, то після побудови зображення було б практично неможливо змінювати положення та розміри окремих його елементів. Шари не дають вам робити помилки, вони полегшують процес внесення змін та розширюють діапазон можливостей. Якщо ви бажаєте додати новий фрагмент зображення в інше зображення, то простіше всього перетягнути цей фрагмент на його нове «місце проживання». Програма автоматично створить новий шар. Інші способи створення нового шару – це вибір команди Layer – New – Layer Via Cut (*Шар – Новий – Скопіювати на новий шар*) або натиснення комбінації клавіш <Ctrl+Shift+J> (рис.7.10). Щоб створити пустий шар (коли, наприклад, необхідно зробити декілька мазків пензликом, не порушуючи вихідного зображення), можна вибрати команду Layer – New – Layer (*Шар – Новий – Шар*).

Шари в роботі

Незалежно від того, як ви створили новий шар, редактор помістить на палітрі поруч з іменем невелике роз'яснення, що вказує на його зміст. Ім'я нового шару виділяється (це говорить про те, що даний шар активізований). Зображення ока (рис. 7.10) необхідне для того, щоб тимчасово приховати або відобразити шари. Той чи інший

Кількість фільтрів та способів їх використання, що пропонує Photoshop, дуже велика, тому потребує окремого вивчення за допомогою додаткової літератури.

7.4. Шари, об'єкти, текст

Для користувачів – початківців знайомство з Photoshop необхідно розпочинати саме з цього питання. Що таке шари, для чого вони призначені та чому вони необхідні? Спробуємо з цими питаннями розібратися.

Основною властивістю шарів є їх непостійність. Кожен шар у композиції не залежить від решти, тому існує можливість у будь-який час робити зміни у зображенні. Зверніть увагу на рис. 7.9, а. Дане зображення складається лише з трикутника. Надалі були додані коло та квадрат з відповідним текстом та різною прозорістю (рис. 7.9,б та в). Все це досягається за рахунок використання шарів. Причому, якщо ви забажаєте змінити розмір, колір або форму будь-якої геометричної фігури, то все це легко зробити під час роботи з відповідними шарами. Якщо б не було шарів, то після побудови зображення було б практично неможливо змінювати положення та розміри окремих його елементів. Шари не дають вам робити помилки, вони полегшують процес внесення змін та розширюють діапазон можливостей. Якщо ви бажаєте додати новий фрагмент зображення в інше зображення, то простіше всього перетягнути цей фрагмент на його нове «місце проживання». Програма автоматично створить новий шар. Інші способи створення нового шару – це вибір команди Layer – New – Layer Via Cut (*Шар – Новий – Скопіювати на новий шар*) або натиснення комбінації клавіш <Ctrl+Shift+J> (рис.7.10). Щоб створити пустий шар (коли, наприклад, необхідно зробити декілька мазків пензликом, не порушуючи вихідного зображення), можна вибрати команду Layer – New – Layer (*Шар – Новий – Шар*).

Шари в роботі

Незалежно від того, як ви створили новий шар, редактор помістить на палітрі поруч з іменем невелике роз'яснення, що вказує на його зміст. Ім'я нового шару виділяється (це говорить про те, що даний шар активізований). Зображення ока (рис. 7.10) необхідне для того, щоб тимчасово приховати або відобразити шари. Той чи інший

шар можна виділити, клацнувши на його імені на палітрі Layers. Після цього шар активізується, що дає можливість його редагування.

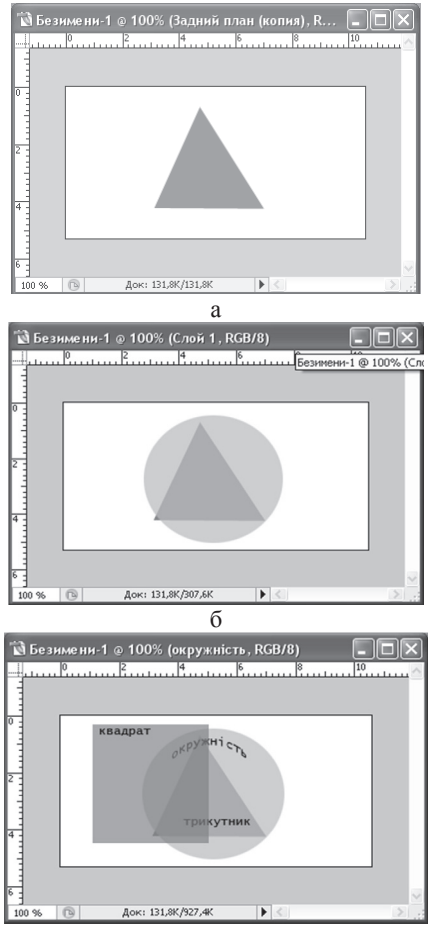


Рис. 7.9. Ілюстрація роботи з шарами:
а – шар з трикутником; б – шар з трикутником та колом;
в – комбіноване зображення

шар можна виділити, клацнувши на його імені на палітрі Layers. Після цього шар активізується, що дає можливість його редагування.

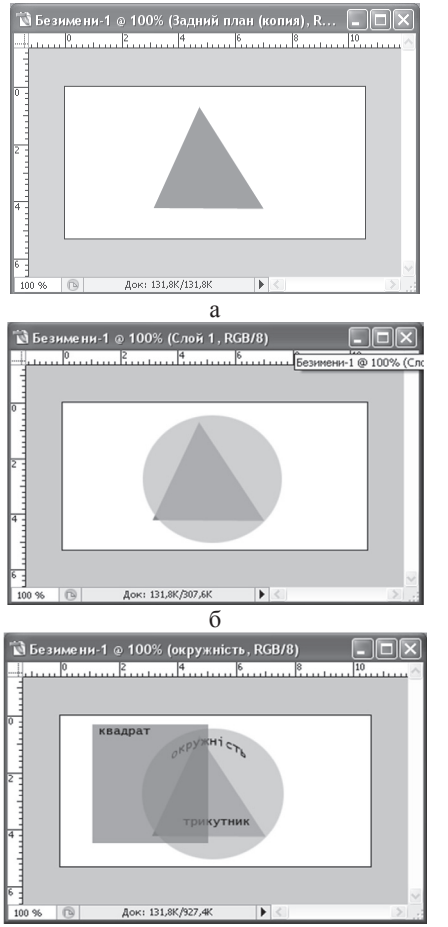


Рис. 7.9. Ілюстрація роботи з шарами:
а – шар з трикутником; б – шар з трикутником та колом;
в – комбіноване зображення

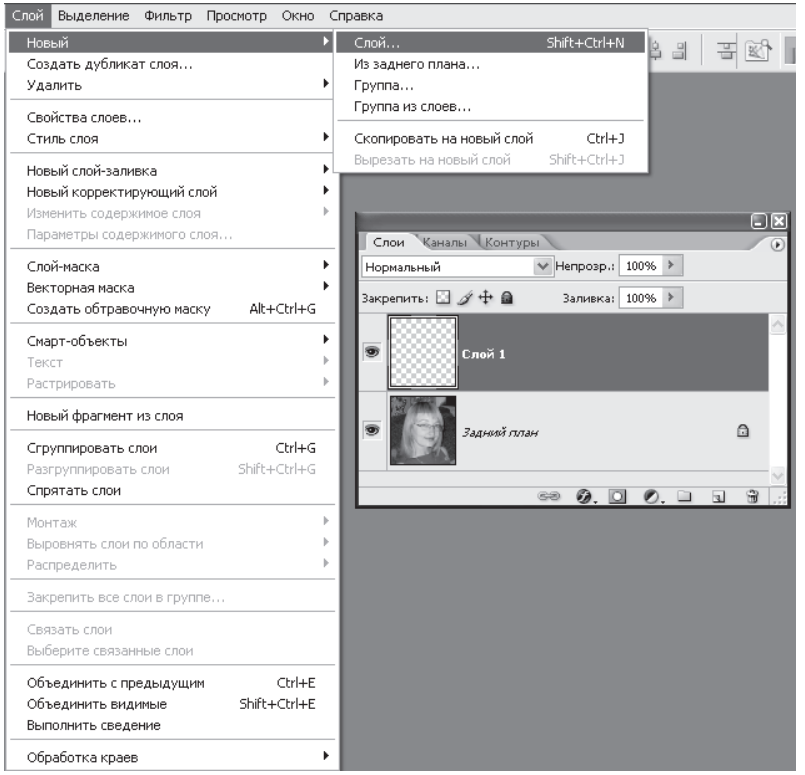


Рис. 7.10. Використання шарів

Переміщення шарів, відносно один одного дозволяє по-різному накладати об’єкти. Все це виконується за допомогою миші.

Хоч шари дуже корисні, їм притаманні певні недоліки. Шари призводять до збільшення обсягів пам’яті оперативно-запам’ятовуючого пристрою (ОЗП), що використовується для зберігання зображення, знижують загальну продуктивність системи.

7.5. Анімація та обробка відеофайлів

У графічному редакторі Photoshop є можливість створення анімаційних зображень. Хоч цей графічний редактор і не призначений безпосередньо для вирішення цих задач та дещо програє у порівнянні зі

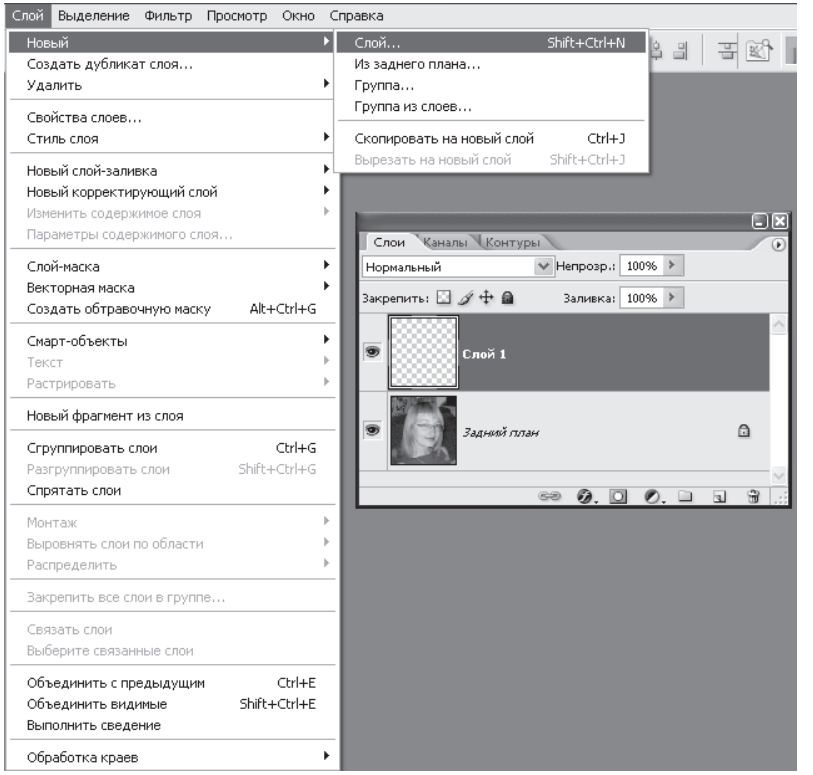


Рис. 7.10. Використання шарів

Переміщення шарів, відносно один одного дозволяє по-різному накладати об’єкти. Все це виконується за допомогою миші.

Хоч шари дуже корисні, їм притаманні певні недоліки. Шари призводять до збільшення обсягів пам’яті оперативно-запам’ятовуючого пристрою (ОЗП), що використовується для зберігання зображення, знижують загальну продуктивність системи.

7.5. Анімація та обробка відеофайлів

У графічному редакторі Photoshop є можливість створення анімаційних зображень. Хоч цей графічний редактор і не призначений безпосередньо для вирішення цих задач та дещо програє у порівнянні зі

спеціалізованими програмами зі створення анімації. Порядок створення анімаційних зображень у Photoshop розглянемо, скориставшись прикладом.

Почнемо з того, що виберемо фотографію або картинку, яка підходить для нанесення на неї анімації. Наприклад, оберемо новорічну картинку для створення анімації «падаючий сніг» (рис. 7.11).

Рис. 7.11. Вихідне зображення

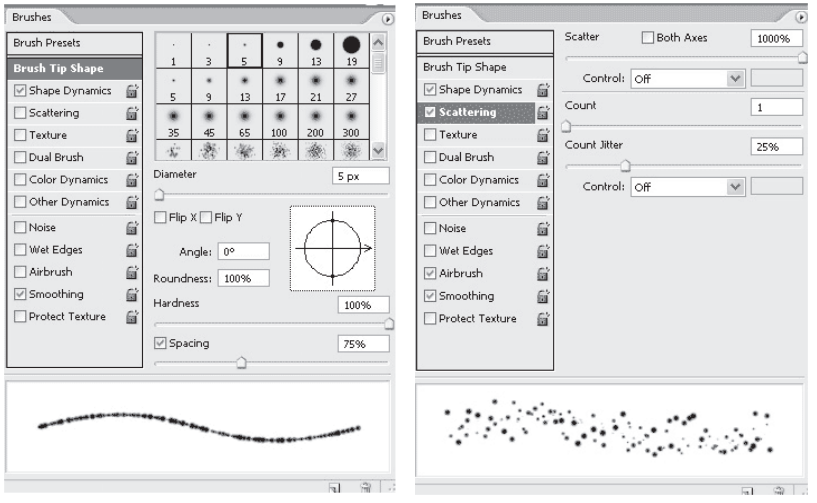


Рис. 7.12. Установки палітри Кисті

спеціалізованими програмами зі створення анімації. Порядок створення анімаційних зображень у Photoshop розглянемо, скориставшись прикладом.

Почнемо з того, що виберемо фотографію або картинку, яка підходить для нанесення на неї анімації. Наприклад, оберемо новорічну картинку для створення анімації «падаючий сніг» (рис. 7.11).

Рис. 7.11. Вихідне зображення

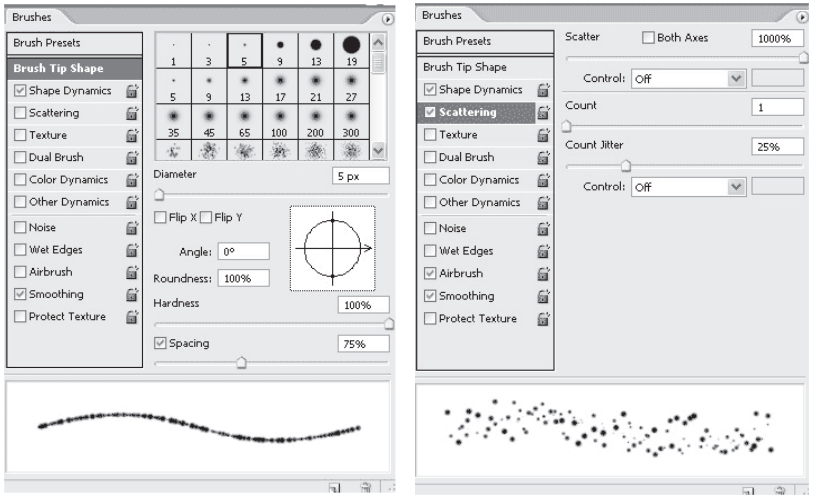


Рис. 7.12. Установки палітри Кисті

Далі вибираємо інструмент Пензель і відкриваємо палітру Кисті (F5). Вибираємо Brush Tip Shape (Форма відбитки кисті) і виставляємо такі установки (рис. 7.12).

Використовуємо білий колір для снігу, який створюємо на новому шарі (див. рис. 7.13).

Рис. 7.13. Утворення снігу на новому шарі

Створюємо копію снігового шару і піднімаємо її настільки вгору, щоб нижня частина нового снігового шару торкалась верхньої частини первісного снігового шару. Злийте обидва снігових шари - виділіть їх і натисніть Ctrl + E. Відкрийте вікно анімації. Для першого слайда залиште малюнок із сніговим шаром, як на рис. 7.14.

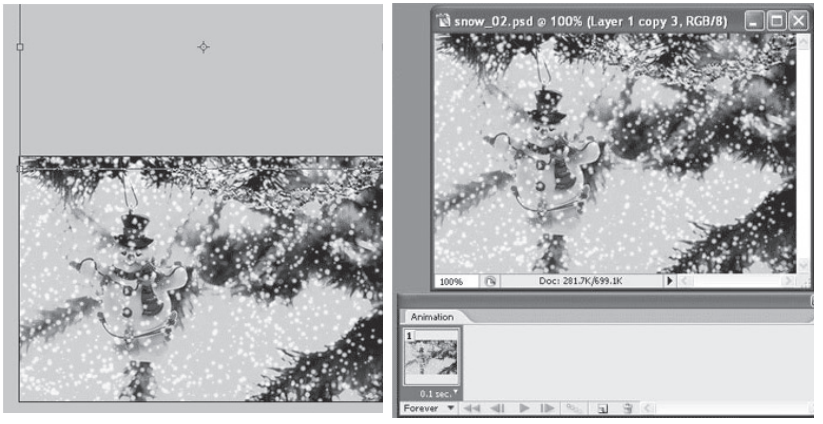


Рис. 7.14. Утворення снігу на новому шарі

Далі вибираємо інструмент Пензель і відкриваємо палітру Кисті (F5). Вибираємо Brush Tip Shape (Форма відбитки кисті) і виставляємо такі установки (рис. 7.12).

Використовуємо білий колір для снігу, який створюємо на новому шарі (див. рис. 7.13).

Рис. 7.13. Утворення снігу на новому шарі

Створюємо копію снігового шару і піднімаємо її настільки вгору, щоб нижня частина нового снігового шару торкалась верхньої частини первісного снігового шару. Злийте обидва снігових шари - виділіть їх і натисніть Ctrl + E. Відкрийте вікно анімації. Для першого слайда залиште малюнок із сніговим шаром, як на рис. 7.14.

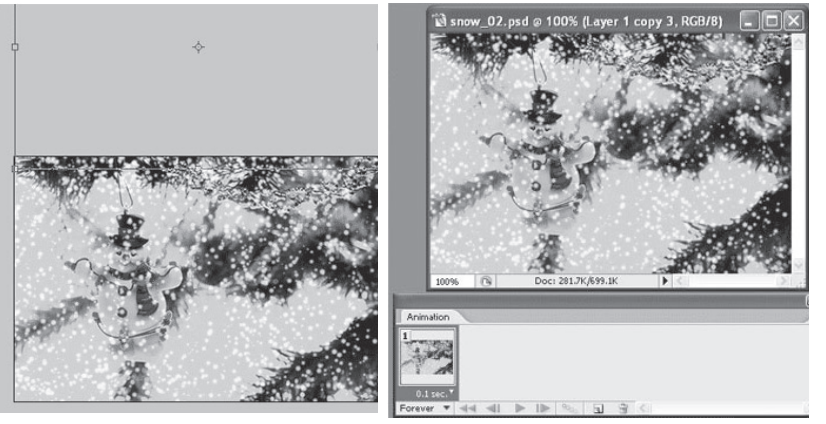


Рис. 7.14. Утворення снігу на новому шарі

Скопіюйте наявний слайд (натисніть на у вікні анімації) і плавно спустіть сніговий шар так, щоб на зображенні з'явилася верхня частина снігового шару (яка була за межами зображення). Виділіть обидва слайди та натисніть Tweens animation frames (Створити проміжні кадри) у вікні анімації (рис. 7.15).

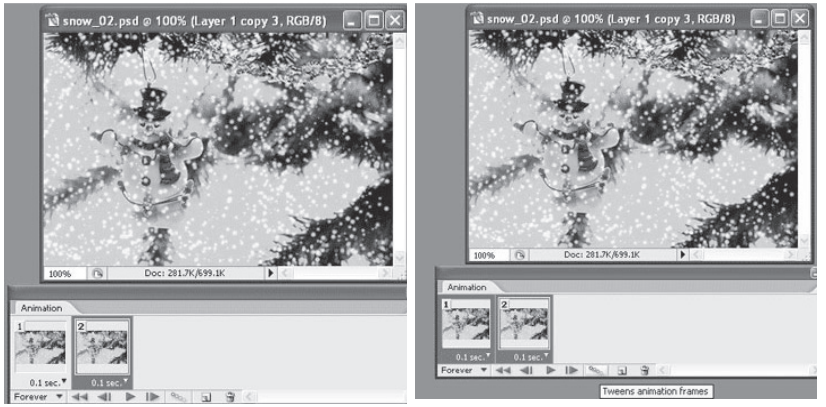
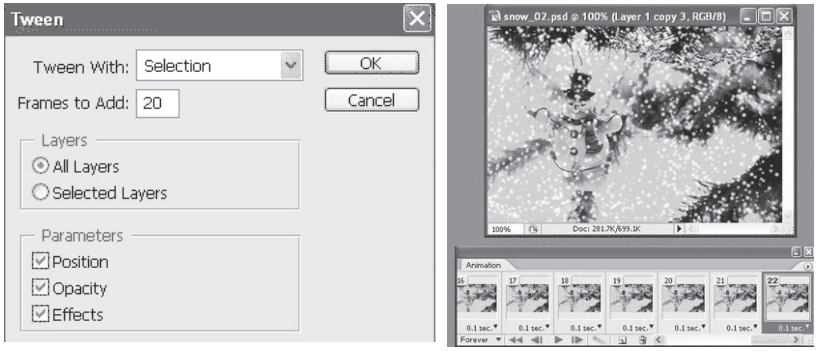


Рис. 7.15. Створення проміжних кадрів

Виберіть у вікні Frames дані параметри (рис. 7.16).



а б

Рис. 7.16. Створення анімації: а – настроювання параметрів; б – кінцевий результат

Скопіюйте наявний слайд (натисніть на у вікні анімації) і плавно спустіть сніговий шар так, щоб на зображенні з'явилася верхня частина снігового шару (яка була за межами зображення). Виділіть обидва слайди та натисніть Tweens animation frames (Створити проміжні кадри) у вікні анімації (рис. 7.15).

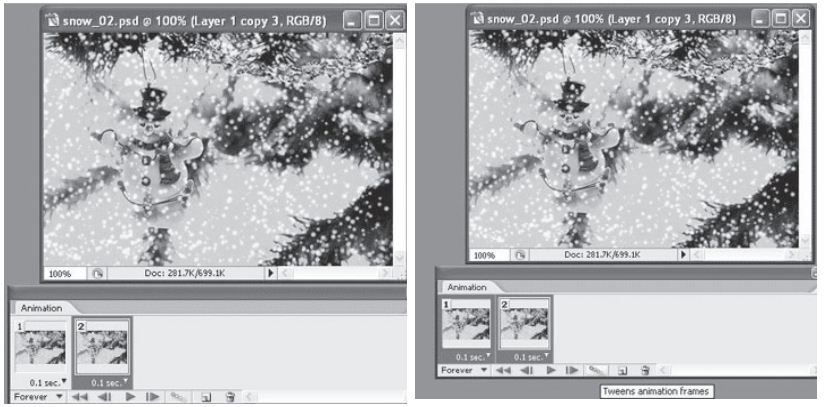
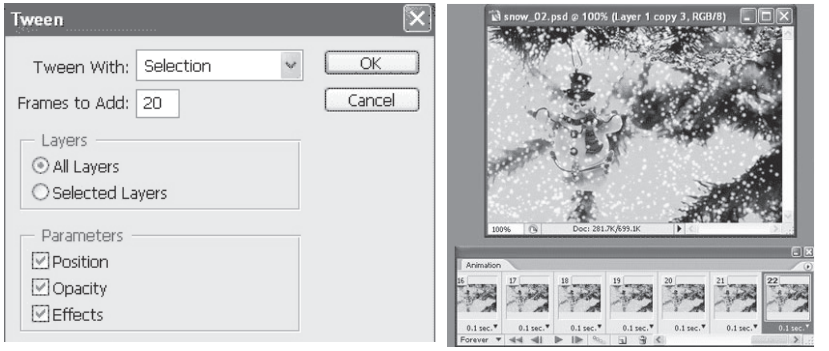


Рис. 7.15. Створення проміжних кадрів

Виберіть у вікні Frames дані параметри (рис. 7.16).



а б

Рис. 7.16. Створення анімації: а – настроювання параметрів; б – кінцевий результат

У вас автоматично вийдуть проміжні слайди з рухом снігу. Видалення останнього слайда закріпить кругообіг анімації падаючого снігу, зробить його безперервним. Відрегулюйте затримку тривалості кожного слайда (під слайдом встановіть час його тривалості), як показано на рис. 7.16, б.

Щоб зберегти анімацію, переходимо до меню Файл–Зберегти для web. Вибираємо формат GIF і кількість кольорів 256.

Обробка відеофайлів

Починаючи з версії PhotoshopCS4, у редакторі з'явилась можливість обробляти відеофайли. Це відкриває багаті можливості в кольоровій корекції плюс використання різноманітних професійних плагінів під фотошоп для відео тощо. Дуже часто в цьому сенсі виникає питання, навіщо це потрібно, якщо є багато спеціалізованих прикладних пакетів для роботи з відео. Відповідь на це запитання одна. Графічний редактор Photoshop і до теперішнього часу залишається найпотужнішим професійним інструментом для роботи з графічними зображеннями, і застосовування всіх його переваг для обробки відео-файлів надає практично необмежні можливості для творчої роботи. Хоч потрібно і відмітити наявність істотної складності та потреби в практичних навичках у використанні інструментів при розв'язанні подібних задач.

Контрольні питання

1. Які існують формати растрової графіки? Дайте стислу характеристику.
2. Проведіть порівняльний аналіз видів комп'ютерної графіки.
3. Дайте характеристику кольорових моделей растрової графіки.
4. Яке призначення фільтрів у пакеті Photoshop?
5. У чому полягає необхідність використання шарів у пакеті Photoshop?
6. Дайте стислу характеристику основних інструментів, що застосовуються у графічному редакторі Photoshop.
7. Починаючи з якої версії в редакторі Photoshop підтримується можливість обробки відеозображень? Чим обумовлено використання цього редактора для вирішення подібних задач?

У вас автоматично вийдуть проміжні слайди з рухом снігу. Видалення останнього слайда закріпить кругообіг анімації падаючого снігу, зробить його безперервним. Відрегулюйте затримку тривалості кожного слайда (під слайдом встановіть час його тривалості), як показано на рис. 7.16, б.

Щоб зберегти анімацію, переходимо до меню Файл–Зберегти для web. Вибираємо формат GIF і кількість кольорів 256.

Обробка відеофайлів

Починаючи з версії PhotoshopCS4, у редакторі з'явилась можливість обробляти відеофайли. Це відкриває багаті можливості в кольоровій корекції плюс використання різноманітних професійних плагінів під фотошоп для відео тощо. Дуже часто в цьому сенсі виникає питання, навіщо це потрібно, якщо є багато спеціалізованих прикладних пакетів для роботи з відео. Відповідь на це запитання одна. Графічний редактор Photoshop і до теперішнього часу залишається найпотужнішим професійним інструментом для роботи з графічними зображеннями, і застосовування всіх його переваг для обробки відео-файлів надає практично необмежні можливості для творчої роботи. Хоч потрібно і відмітити наявність істотної складності та потреби в практичних навичках у використанні інструментів при розв'язанні подібних задач.

Контрольні питання

1. Які існують формати растрової графіки? Дайте стислу характеристику.
2. Проведіть порівняльний аналіз видів комп'ютерної графіки.
3. Дайте характеристику кольорових моделей растрової графіки.
4. Яке призначення фільтрів у пакеті Photoshop?
5. У чому полягає необхідність використання шарів у пакеті Photoshop?
6. Дайте стислу характеристику основних інструментів, що застосовуються у графічному редакторі Photoshop.
7. Починаючи з якої версії в редакторі Photoshop підтримується можливість обробки відеозображень? Чим обумовлено використання цього редактора для вирішення подібних задач?

Глава 8
Характеристика основних можливостей пакета векторної графіки

8.1. Основні можливості та інструменти

Adobe Illustrator – редактор векторної графіки, який призначений для створення зображень, що використовуються у поліграфії в електронних презентаціях та Web-дизайні [1]. До зображень, з якими працює програма, можна віднести всілякі знаки, логотипи, технічні ілюстрації, схеми, плани тощо. Зображення, що створюються в Adobe Illustrator, легко інтегруються в мультимедійні програми.

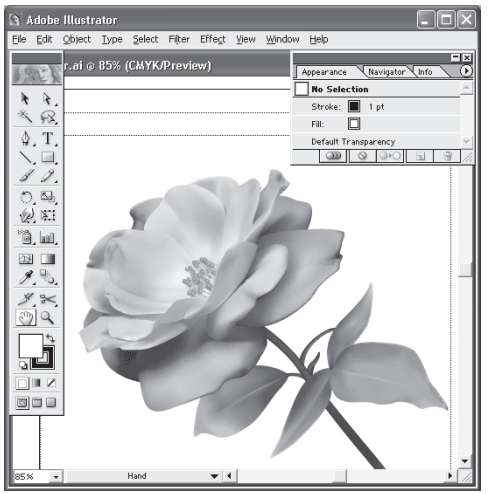


Рис. 8.1. Загальний вигляд інтерфейсу користувача програми Adobe Illustrator

Глава 8
Характеристика основних можливостей пакета векторної графіки

8.1. Основні можливості та інструменти

Adobe Illustrator – редактор векторної графіки, який призначений для створення зображень, що використовуються у поліграфії в електронних презентаціях та Web-дизайні [1]. До зображень, з якими працює програма, можна віднести всілякі знаки, логотипи, технічні ілюстрації, схеми, плани тощо. Зображення, що створюються в Adobe Illustrator, легко інтегруються в мультимедійні програми.

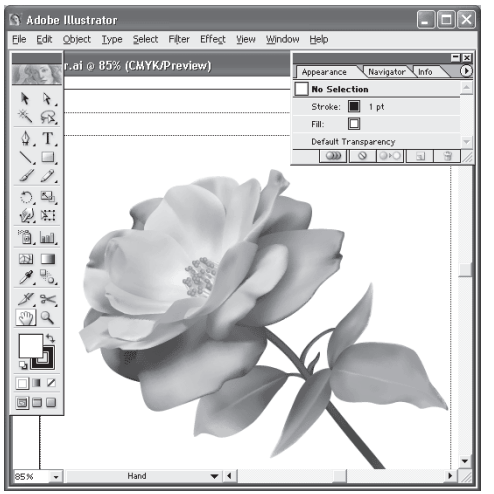


Рис. 8.1. Загальний вигляд інтерфейсу користувача програми Adobe Illustrator

Після завантаження програми та зображення заставки фірми на екрані з'являється вікно програми, яке називається інтерфейсом користувача (див. рис. 8.1). Інтерфейс є посередником між людиною та комп'ютером, він надає все необхідне для роботи: інструменти, палітри, діалогові вікна.

Інтерфейс користувача включає заголовок, головне командне меню, робочі вікна для відображення, а також сукупність різноманітних палітр, у тому числі палітри інструментів, за допомогою яких здійснюється створення та редагування зображень.

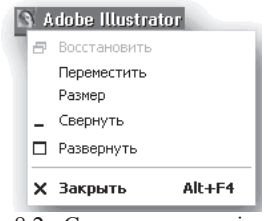


Рис. 8.2. Системне меню інтерфейсу користувача

Сама верхня смуга синього кольору – рядок заголовка – відображає назву та значок програми, а також містить три кнопки, які дозволяють управляти розмірами та розташуванням програмного вікна.

Клацання на значку програми в лівій частині смуги виводить на екран віконне меню (рис. 8.2).

Під смугою заголовка розміщена смуга головного командного меню (MenuBar), яка пропонує такі групи команд: File (Файл), Edit (Правка), Object (Об'єкт), Type (Текст), Select (Виділення), Filter (Фільтр), Effect (Ефект), View (Перегляд), Window (Вікно), Help (Допомога). Кожна група — це сукупність команд, що виконують функціонально близькі дії.

Наприклад, меню Filter (Фільтр) включає значне число вбудованих і додаткових команд, що виконують роль фільтрів для зображень, а меню Object (Об'єкт) пропонує команди для роботи з виділеними об'єктами.

Якщо в рядку назви команди зображена трикутна стрілка, це означає, що в даній команді є вкладене меню (підменю) — список команд, кожна з яких є самостійною командою.

Значення поточного масштабу зображення на екрані знаходиться в лівому нижньому кутку робочого вікна кожного документа. Діапазон збільшення або зменшення зображення величезний: від 3,13 до 6400%.

Після завантаження програми та зображення заставки фірми на екрані з'являється вікно програми, яке називається інтерфейсом користувача (див. рис. 8.1). Інтерфейс є посередником між людиною та комп'ютером, він надає все необхідне для роботи: інструменти, палітри, діалогові вікна.

Інтерфейс користувача включає заголовок, головне командне меню, робочі вікна для відображення, а також сукупність різноманітних палітр, у тому числі палітри інструментів, за допомогою яких здійснюється створення та редагування зображень.

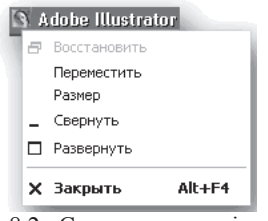


Рис. 8.2. Системне меню інтерфейсу користувача

Сама верхня смуга синього кольору – рядок заголовка – відображає назву та значок програми, а також містить три кнопки, які дозволяють управляти розмірами та розташуванням програмного вікна.

Клацання на значку програми в лівій частині смуги виводить на екран віконне меню (рис. 8.2).

Під смугою заголовка розміщена смуга головного командного меню (MenuBar), яка пропонує такі групи команд: File (Файл), Edit (Правка), Object (Об'єкт), Type (Текст), Select (Виділення), Filter (Фільтр), Effect (Ефект), View (Перегляд), Window (Вікно), Help (Допомога). Кожна група — це сукупність команд, що виконують функціонально близькі дії.

Наприклад, меню Filter (Фільтр) включає значне число вбудованих і додаткових команд, що виконують роль фільтрів для зображень, а меню Object (Об'єкт) пропонує команди для роботи з виділеними об'єктами.

Якщо в рядку назви команди зображена трикутна стрілка, це означає, що в даній команді є вкладене меню (підменю) — список команд, кожна з яких є самостійною командою.

Значення поточного масштабу зображення на екрані знаходиться в лівому нижньому кутку робочого вікна кожного документа. Діапазон збільшення або зменшення зображення величезний: від 3,13 до 6400%.

У нижній частині робочого екрану розташовується смуга стану (StatusBar), яка призначена для відображення службової інформації, наприклад, назви активного інструменту. При натисненні стрілки в правій частині смуги стану на екран виводиться список режимів (рис. 8.4).

Режими смуги стану:

– CurrentTool – назва активного інструменту;
– DateandTime – поточна дата і час;
– FreeMemory – об'єм вільної оперативної і віртуальної (дискової) пам'яті;
– FreeMemory – об'єм вільної оперативної і віртуальної (дискової) пам'яті;
– NumberofUndos – число доступних відмін і повторень виконаних операцій;
– DocumentColorProfile – колірний профіль документа.

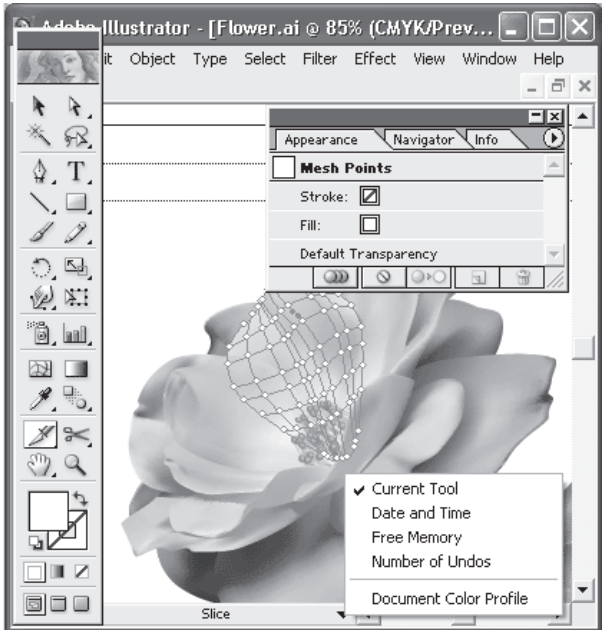


Рис. 8.3. Список режимів смуги стану

У нижній частині робочого екрану розташовується смуга стану (StatusBar), яка призначена для відображення службової інформації, наприклад, назви активного інструменту. При натисненні стрілки в правій частині смуги стану на екран виводиться список режимів (рис. 8.4).

Режими смуги стану:

– CurrentTool – назва активного інструменту;
– DateandTime – поточна дата і час;
– FreeMemory – об'єм вільної оперативної і віртуальної (дискової) пам'яті;
– FreeMemory – об'єм вільної оперативної і віртуальної (дискової) пам'яті;
– NumberofUndos – число доступних відмін і повторень виконаних операцій;
– DocumentColorProfile – колірний профіль документа.

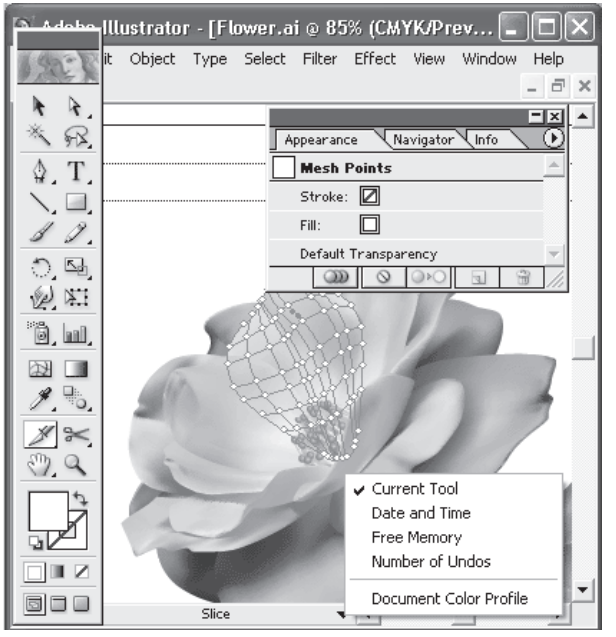


Рис. 8.3. Список режимів смуги стану

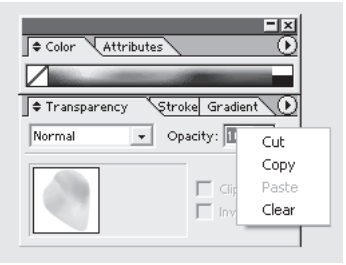


Рис. 8.4. Приклад контекстного меню палітри

На додаток до тих пунктів головного меню і пунктів меню, які пропонують всілякі палітри, в програмі Adobe Illustrator передбачені контекстні меню, які викликаються натисненням правої кнопки миші. Вміст цих меню знаходиться залежно від активного в даний момент інструменту, типу виділеного об'єкту або відкритої палітри (рис. 8.4).

Всі команди контекстного меню дублюють команди головного меню або меню палітр, але перевага його використання полягає в швидкості доступу. За замовчуванням у лівій частині робочого вікна розташована палітра інструментів (рис. 8.5), що містить всі інструменти, за допомогою яких можна створювати, виділяти, редагувати і переміщати графічні об'єкти.

Рис. 8.5. Вигляд палітри інструментів

У верхній частині палітри розташована темна смуга "заголовка". Під нею знаходиться декоративна емблема програми, клацання на емблемі виводить на екран діалогове вікно Adobe Online (Сервер Adobe).

У нижній частині палітри інструментів розташовано три кнопки (рис. 8.5), які визначають режими відображення робочого екрану програми. Ліва кнопка включена за замовчуванням і визначає стандартний режим (Standard Screen Mode) зі всіма меню, смугами прокрутки. Центральна кнопка включає повноекранне виведення (Full Screen Mode with Menu Rar) зображення із смугою меню, але без смуги заголовка і без смуг прокрутки. Права кнопка включає повноекранне виведення (Full Screen Mode) зображення без смуги заголовка, без смуги меню і без смуг прокрутки.

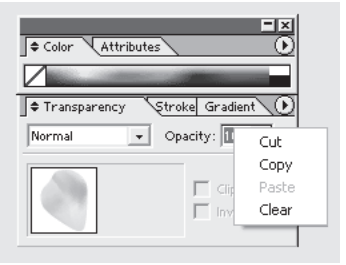


Рис. 8.4. Приклад контекстного меню палітри

На додаток до тих пунктів головного меню і пунктів меню, які пропонують всілякі палітри, в програмі Adobe Illustrator передбачені контекстні меню, які викликаються натисненням правої кнопки миші. Вміст цих меню знаходиться залежно від активного в даний момент інструменту, типу виділеного об'єкту або відкритої палітри (рис. 8.4).

Всі команди контекстного меню дублюють команди головного меню або меню палітр, але перевага його використання полягає в швидкості доступу. За замовчуванням у лівій частині робочого вікна розташована палітра інструментів (рис. 8.5), що містить всі інструменти, за допомогою яких можна створювати, виділяти, редагувати і переміщати графічні об'єкти.

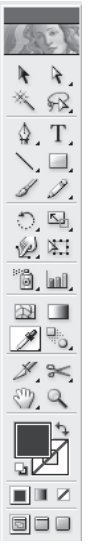


Рис. 8.5. Вигляд палітри інструментів

У верхній частині палітри розташована темна смуга "заголовка". Під нею знаходиться декоративна емблема програми, клацання на емблемі виводить на екран діалогове вікно Adobe Online (Сервер Adobe).

У нижній частині палітри інструментів розташовано три кнопки (рис. 8.5), які визначають режими відображення робочого екрану програми. Ліва кнопка включена за замовчуванням і визначає стандартний режим (Standard Screen Mode) зі всіма меню, смугами прокрутки. Центральна кнопка включає повноекранне виведення (Full Screen Mode with Menu Rar) зображення із смугою меню, але без смуги заголовка і без смуг прокрутки. Права кнопка включає повноекранне виведення (Full Screen Mode) зображення без смуги заголовка, без смуги меню і без смуг прокрутки.

Палітри

Програма Adobe Illustrator, окрім палітри інструментів, містить ще безліч палітр, що служать для зручності здійснення тих або інших функцій, наприклад вибору кольору та інших параметрів обведення і заливки, виконання різних трансформацій тощо.

Для відображення на екрані будь-якої палітри в меню Window (Вікно) представлені команди, що збігаються з назвою палітри. Якщо палітра знаходиться на екрані, її назва відмічена знаком "галочки".

Палітри займають значне місце на екрані. Для того щоб перетягнути їх з місця на місце, необхідно захопити заголовок палітри і перемістити її в потрібне місце.

За допомогою клавіші <Tab> можна тимчасово видалити всі відкриті палітри, включаючи і палітру інструментів. Повторне натиснення на ту ж клавішу повертає все на свої місця.

Комбінація клавіш <Shift>+<Tab> закриває всі відкриті палітри, окрім палітри інструментів. Повторне натиснення цих клавіш повертає на екран всі закриті палітри.

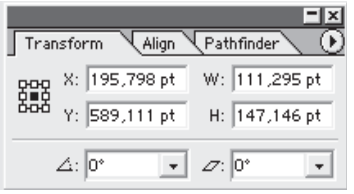


Рис. 8.6. Група палітр

Палітри за замовчуванням об'єднані в групи і відображаються в одному вікні (рис. 8.6). У складі такої групи перемикання палітр здійснюється клацанням на відповідній вкладці. Для того щоб відокремити яку-небудь палітру від групи, необхідно захопити вкладку цієї палітри і перетягнути умовний прямокутник, що відображає габарити палітри, за межі групи. Для того щоб включити палітру в групу, необхідно також захопити вкладку палітри і перетягнути її в межі палітри або групи палітр, до яких приєднана дана палітра.

Виправлення помилок

Не помиляється тільки той, хто нічого не робить. У реальній практиці навіть найуважнішого і найвдумливішого користувача підстерігають неминучі і прикрі помилки. Основним засобом боротьби з

Палітри

Програма Adobe Illustrator, окрім палітри інструментів, містить ще безліч палітр, що служать для зручності здійснення тих або інших функцій, наприклад вибору кольору та інших параметрів обведення і заливки, виконання різних трансформацій тощо.

Для відображення на екрані будь-якої палітри в меню Window (Вікно) представлені команди, що збігаються з назвою палітри. Якщо палітра знаходиться на екрані, її назва відмічена знаком "галочки".

Палітри займають значне місце на екрані. Для того щоб перетягнути їх з місця на місце, необхідно захопити заголовок палітри і перемістити її в потрібне місце.

За допомогою клавіші <Tab> можна тимчасово видалити всі відкриті палітри, включаючи і палітру інструментів. Повторне натиснення на ту ж клавішу повертає все на свої місця.

Комбінація клавіш <Shift>+<Tab> закриває всі відкриті палітри, окрім палітри інструментів. Повторне натиснення цих клавіш повертає на екран всі закриті палітри.

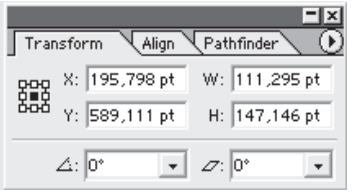


Рис. 8.6. Група палітр

Палітри за замовчуванням об'єднані в групи і відображаються в одному вікні (рис. 8.6). У складі такої групи перемикання палітр здійснюється клацанням на відповідній вкладці. Для того щоб відокремити яку-небудь палітру від групи, необхідно захопити вкладку цієї палітри і перетягнути умовний прямокутник, що відображає габарити палітри, за межі групи. Для того щоб включити палітру в групу, необхідно також захопити вкладку палітри і перетягнути її в межі палітри або групи палітр, до яких приєднана дана палітра.

Виправлення помилок

Не помиляється тільки той, хто нічого не робить. У реальній практиці навіть найуважнішого і найвдумливішого користувача підстерігають неминучі і прикрі помилки. Основним засобом боротьби з

цим є команди Undo (Відмінити) і Redo (Повторити) меню Edit (Правка), які дозволяють відмінити або повторити одну або декілька попередніх операцій.

Програма Adobe Illustrator допускає до 200 таких відмін (повторів), причому у програми є цікава властивість: відміни можна виконувати і після команди Save (Зберегти), якщо при цьому не закривати файл.

Такі приголомшливі можливості проте можуть бути обмежені обсягом доступної оперативної пам'яті. Якщо це трапляється, то на екран виводиться відповідне повідомлення.

За замовчуванням мінімальна кількість відмов приймається рівною 4. Це значення можна змінити в полі Undo (Відміна команд) (рис. 8.7) розділу Units&Undo (Одиниці виміру і відміна команд) діалогового вікна Preferences (Установки), яке викликається командою Preferences (Установки) меню Edit (Правка).

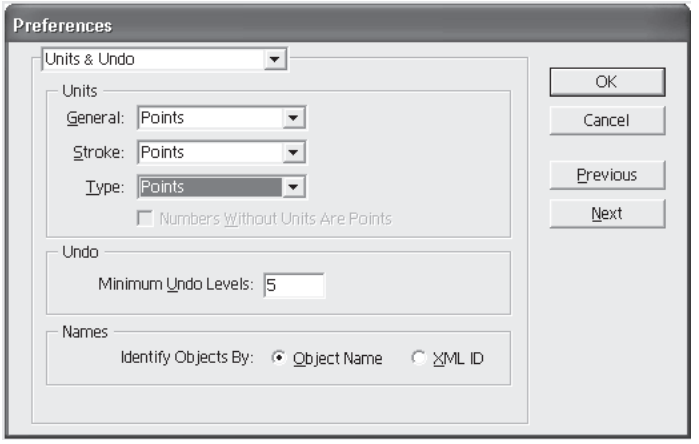


Рис. 8.7. Поле Undoрозділу Units&Undo

Створення, відкриття і закриття документів

Новий документ можна створити у будь-який момент, а також у будь-який момент відкрити який завгодно з існуючих документів.

Для створення нового документа необхідно виконати команду New (Новий) меню File (Файл).

цим є команди Undo (Відмінити) і Redo (Повторити) меню Edit (Правка), які дозволяють відмінити або повторити одну або декілька попередніх операцій.

Програма Adobe Illustrator допускає до 200 таких відмін (повторів), причому у програми є цікава властивість: відміни можна виконувати і після команди Save (Зберегти), якщо при цьому не закривати файл.

Такі приголомшливі можливості проте можуть бути обмежені обсягом доступної оперативної пам'яті. Якщо це трапляється, то на екран виводиться відповідне повідомлення.

За замовчуванням мінімальна кількість відмов приймається рівною 4. Це значення можна змінити в полі Undo (Відміна команд) (рис. 8.7) розділу Units&Undo (Одиниці виміру і відміна команд) діалогового вікна Preferences (Установки), яке викликається командою Preferences (Установки) меню Edit (Правка).

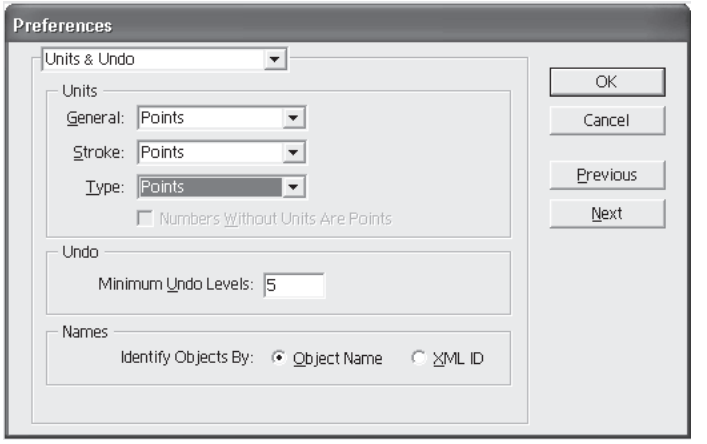


Рис. 8.7. Поле Undoрозділу Units&Undo

Створення, відкриття і закриття документів

Новий документ можна створити у будь-який момент, а також у будь-який момент відкрити який завгодно з існуючих документів.

Для створення нового документа необхідно виконати команду New (Новий) меню File (Файл).

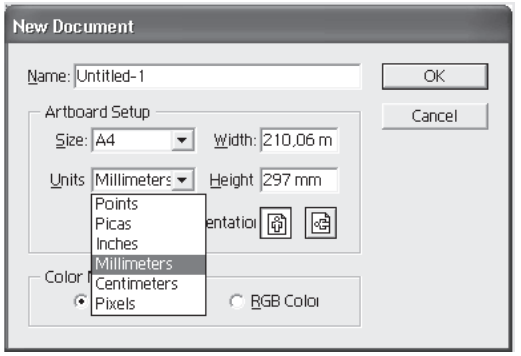


Рис. 8.8. Діалогове вікно NewDocument

На екран виводиться діалогове вікно New Document (Новий документ) (рис. 8.8), у якому можна визначити такі параметри:

– довільне ім'я документа в полі Name (Ім'я), за замовчуванням пропонується ім'я Untitled- з порядковим номером. Користувач має право прийняти дане ім'я або ввести будь-яке інше;

– колірну модель: у полі Color Mode (Колірна модель) пропонуються перемикачі CMYK Color (Кольори моделі CMYK) і RGB Color (Кольори моделі RGB);

– розмір сторінки документа: у полі Artboard Setup (Установка робочої сторінки) можна визначити її ширину (поле Width) і висоту (поле Height) або вибрати із списку Size (Розмір). У полі Orientation (Орієнтація) дві кнопки дозволяють наочно визначити розташування сторінки: книжне або альбомне.

У списку Units (Одиниці виміру) подані такі варіанти:

– Points (Пункти);
– Millimeters (Міліметри);
– Picas (Списи);
– Centimeters (Сантиметри);
– Inches (Дюйми);
– Pixels (Піксели).

Для відкриття існуючого документа необхідно виконати команду Open (Відкрити) меню File (Файл). На екран буде виведено однойменне діалогове вікно, яке нічим не відрізняється від звичайного вікна операційного середовища Windows за винятком вікна перегляду,

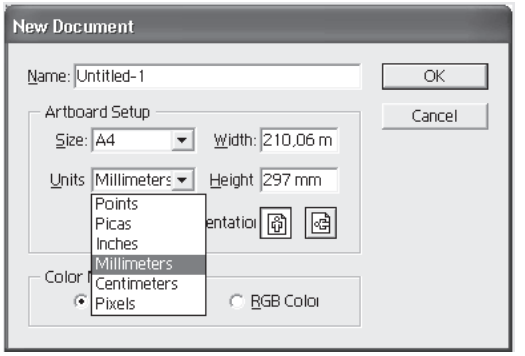


Рис. 8.8. Діалогове вікно NewDocument

На екран виводиться діалогове вікно New Document (Новий документ) (рис. 8.8), у якому можна визначити такі параметри:

– довільне ім'я документа в полі Name (Ім'я), за замовчуванням пропонується ім'я Untitled- з порядковим номером. Користувач має право прийняти дане ім'я або ввести будь-яке інше;

– колірну модель: у полі Color Mode (Колірна модель) пропонуються перемикачі CMYK Color (Кольори моделі CMYK) і RGB Color (Кольори моделі RGB);

– розмір сторінки документа: у полі Artboard Setup (Установка робочої сторінки) можна визначити її ширину (поле Width) і висоту (поле Height) або вибрати із списку Size (Розмір). У полі Orientation (Орієнтація) дві кнопки дозволяють наочно визначити розташування сторінки: книжне або альбомне.

У списку Units (Одиниці виміру) подані такі варіанти:

– Points (Пункти);
– Millimeters (Міліметри);
– Picas (Списи);
– Centimeters (Сантиметри);
– Inches (Дюйми);
– Pixels (Піксели).

Для відкриття існуючого документа необхідно виконати команду Open (Відкрити) меню File (Файл). На екран буде виведено однойменне діалогове вікно, яке нічим не відрізняється від звичайного вікна операційного середовища Windows за винятком вікна перегляду,

розташованого в нижній частині діалогового вікна. У вікні перегляду відображається сторінка документа, який зберігається разом з документом у вигляді невеликого піксельного зображення (рис. 8.9).

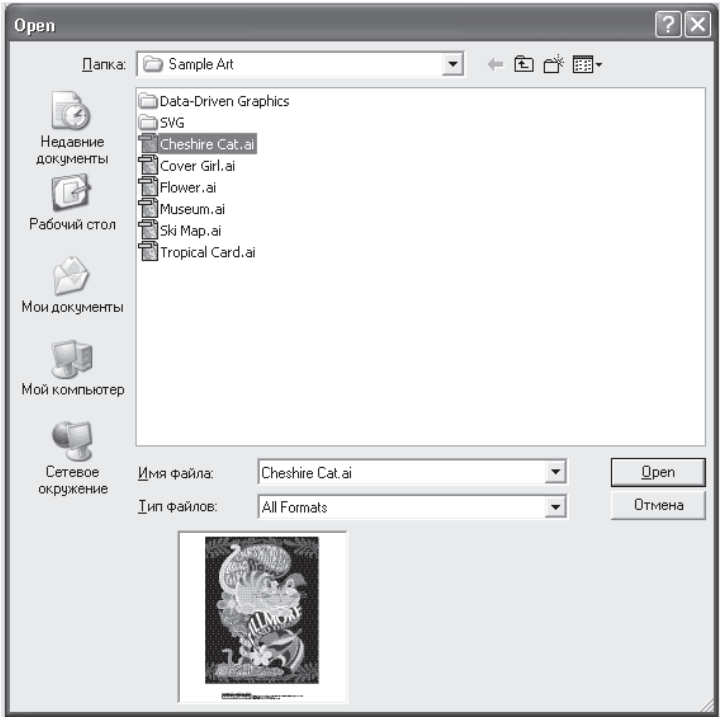


Рис. 8.9. Відображення документа в діалоговому вікні Open

Оскільки програма Adobe Illustrator дозволяє відкривати одночасно декілька документів, передбачена можливість закрити (видалити з оперативної пам'яті) непотрібні в даний момент документи. При виконанні команди Close (Закрити) меню File (Файл) закривається активний документ. Якщо при виконанні команди утримувати клавішу <Alt>, то будуть закриті всі документи. Якщо в яких-небудь документах були виконані зміни, надійде запит на їх збереження.

розташованого в нижній частині діалогового вікна. У вікні перегляду відображається сторінка документа, який зберігається разом з документом у вигляді невеликого піксельного зображення (рис. 8.9).

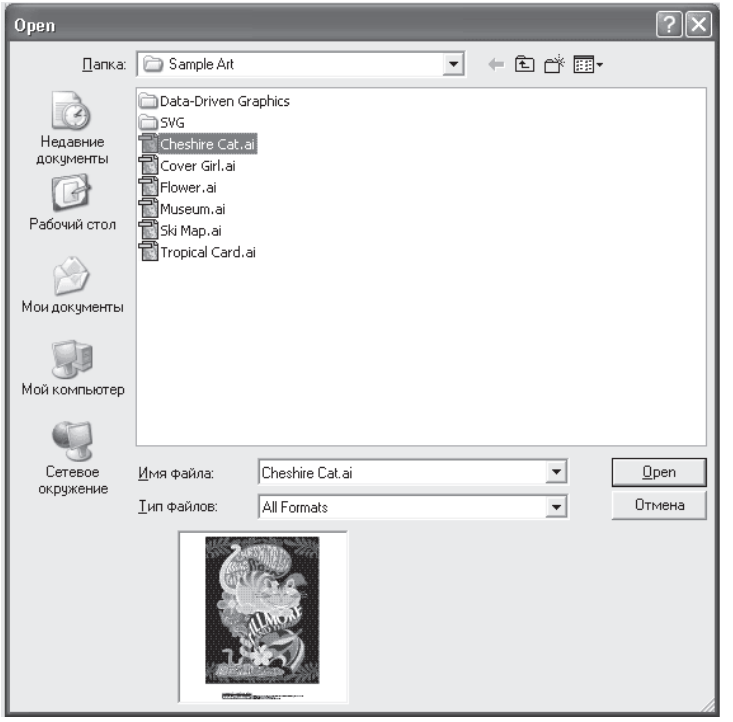


Рис. 8.9. Відображення документа в діалоговому вікні Open

Оскільки програма Adobe Illustrator дозволяє відкривати одночасно декілька документів, передбачена можливість закрити (видалити з оперативної пам'яті) непотрібні в даний момент документи. При виконанні команди Close (Закрити) меню File (Файл) закривається активний документ. Якщо при виконанні команди утримувати клавішу <Alt>, то будуть закриті всі документи. Якщо в яких-небудь документах були виконані зміни, надійде запит на їх збереження.

302

8.2. Розміщення об’єктів

Після створення об’єктів необхідно розташувати їх у належному порядку. Програма Adobe Illustrator має в своєму розпорядженні засоби, які надають користувачеві всі можливості щодо переміщення і вирівнювання об’єктів з максимальною точністю. Сюди відносяться засоби виміру, вирівнювання, групування, "фіксації" в певному положенні і тимчасового видалення об’єктів з екрана.

Палітра Transform (Трансформація) служить для інтерактивного переміщення і трансформації об’єктів.

Пересування і копіювання об’єктів

У програмі Adobe Illustrator об’єкти можна пересувати декількома способами:

1. Просто "перетягнути" за допомогою миші.
2. Пересунути за допомогою клавіш управління курсором.
3. Пересунути за допомогою палітри Transform (Трансформація).
4. З використанням діалогових вікон Move (Пересування) і Transform Each (Трансформувати кожен).

За допомогою "перетягання" можна також передавати зображення між відкритими документами програм Adobe Illustrator і Adobe Photoshop.

Спільні установки і способи пересування об’єктів

У розділі General (Основні) діалогового вікна Preferences (Установки), яке викликається командою Preferences (Установки) меню Edit (Правка), є параметри, що визначають можливості переміщення об’єктів (рис. 8.10).

У полі Constrain Angle (Кут повороту осей) вводиться значення кута в градусах, який визначає кут нахилу осей X і Y.

Установка прапорця Transform Pattern Tiles (Трансформувати образи) забезпечує трансформацію декоративних заливок відповідно до трансформації самого об’єкта, наприклад масштабування, обертання тощо.

У полі Keyboard Increment (Клавіатурний приріст) визначається відстань, на яку пересувається виділений об’єкт при однократному натисненні на будь-яку клавішу управління курсором. Допустимий

8.2. Розміщення об’єктів

Після створення об’єктів необхідно розташувати їх у належному порядку. Програма Adobe Illustrator має в своєму розпорядженні засоби, які надають користувачеві всі можливості щодо переміщення і вирівнювання об’єктів з максимальною точністю. Сюди відносяться засоби виміру, вирівнювання, групування, "фіксації" в певному положенні і тимчасового видалення об’єктів з екрана.

Палітра Transform (Трансформація) служить для інтерактивного переміщення і трансформації об’єктів.

Пересування і копіювання об’єктів

У програмі Adobe Illustrator об’єкти можна пересувати декількома способами:

1. Просто "перетягнути" за допомогою миші.
2. Пересунути за допомогою клавіш управління курсором.
3. Пересунути за допомогою палітри Transform (Трансформація).
4. З використанням діалогових вікон Move (Пересування) і Transform Each (Трансформувати кожен).

За допомогою "перетягання" можна також передавати зображення між відкритими документами програм Adobe Illustrator і Adobe Photoshop.

Спільні установки і способи пересування об’єктів

У розділі General (Основні) діалогового вікна Preferences (Установки), яке викликається командою Preferences (Установки) меню Edit (Правка), є параметри, що визначають можливості переміщення об’єктів (рис. 8.10).

У полі Constrain Angle (Кут повороту осей) вводиться значення кута в градусах, який визначає кут нахилу осей X і Y.

Установка прапорця Transform Pattern Tiles (Трансформувати образи) забезпечує трансформацію декоративних заливок відповідно до трансформації самого об’єкта, наприклад масштабування, обертання тощо.

У полі Keyboard Increment (Клавіатурний приріст) визначається відстань, на яку пересувається виділений об’єкт при однократному натисненні на будь-яку клавішу управління курсором. Допустимий

діапазон пересування– від 0 до 447,199 мм. За замовчуванням прийнята відстань в 1 пункт (0,343 мм).

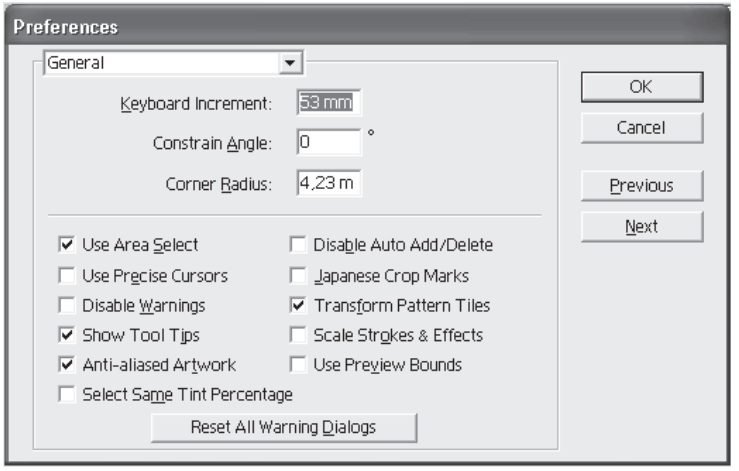


Рис. 8.10. Установки пересування в розділі General діалогового вікна Preferences

Для пересування об’єкта за допомогою клавіш управління курсором слід його виділити (це стосується і декількох об’єктів), а потім натискати клавішу відповідного напряму необхідне число разів.

Активізація команди Snapto Point (Вирівнювати по точках) меню View (Перегляд) забезпечує "прилипання" переміщуваного об’єкта до опорних точок, якщо даний об’єкт потрапляє в зону двох пікселів від цих точок. За замовчуванням ця команда активізована.

При "перетяганні" об’єкта за допомогою миші можна забезпечувати напрям переміщення строго по горизонталі або по вертикалі (а також напряму, кратного 45 градусам), якщо утримувати натиснутою клавішу <Shift>.

Якщо на новому місці необхідно отримати копію об’єкта, що пересувається, то слід утримувати натиснутою клавішу <Alt> (рис.8.11). Об’єкти можна пересувати за допомогою інструменту FreeTransform (Вільна трансформація) ( ). Для цього досить захопити будь-яку точку об’єкта, окрім маркерів "габаритного" прямокутника.

діапазон пересування– від 0 до 447,199 мм. За замовчуванням прийнята відстань в 1 пункт (0,343 мм).

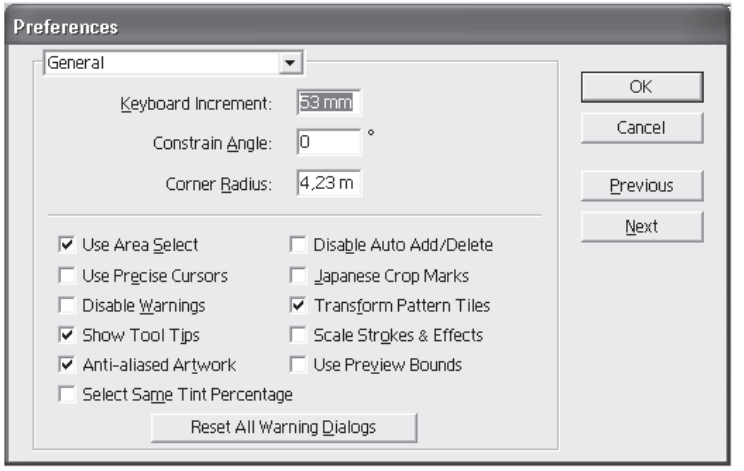


Рис. 8.10. Установки пересування в розділі General діалогового вікна Preferences

Для пересування об’єкта за допомогою клавіш управління курсором слід його виділити (це стосується і декількох об’єктів), а потім натискати клавішу відповідного напряму необхідне число разів.

Активізація команди Snapto Point (Вирівнювати по точках) меню View (Перегляд) забезпечує "прилипання" переміщуваного об’єкта до опорних точок, якщо даний об’єкт потрапляє в зону двох пікселів від цих точок. За замовчуванням ця команда активізована.

При "перетяганні" об’єкта за допомогою миші можна забезпечувати напрям переміщення строго по горизонталі або по вертикалі (а також напряму, кратного 45 градусам), якщо утримувати натиснутою клавішу <Shift>.

Якщо на новому місці необхідно отримати копію об’єкта, що пересувається, то слід утримувати натиснутою клавішу <Alt> (рис.8.11). Об’єкти можна пересувати за допомогою інструменту FreeTransform (Вільна трансформація) ( ). Для цього досить захопити будь-яку точку об’єкта, окрім маркерів "габаритного" прямокутника.

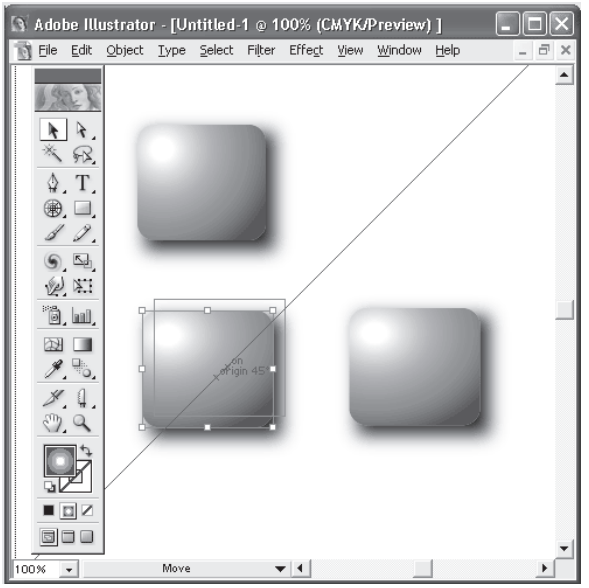


Рис. 8.11. Переміщення об’єкта з натиснутою клавішею <Alt>, що дозволяє отримати його копію

Для того щоб перемістити об’єкт на точно відому відстань, слід скористатися командою Move (Переміщення) меню Object | Transform (Об’єкт | Трансформація), яка відкриває діалогове вікно Move (Переміщення), яке зображено на рис.8.12.

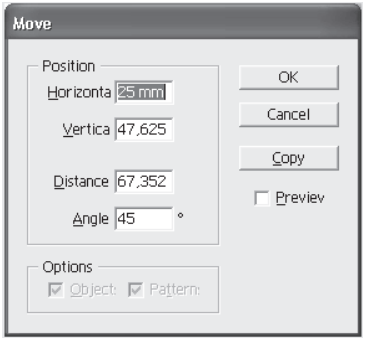


Рис. 8.12. Діалогове вікно Move

Якщо об’єкт, призначений для переміщення, вже виділений, то діалогове вікно Move (Переміщення) можна викликати подвійним клацанням на інструменті Selection (Виділення) ( ).

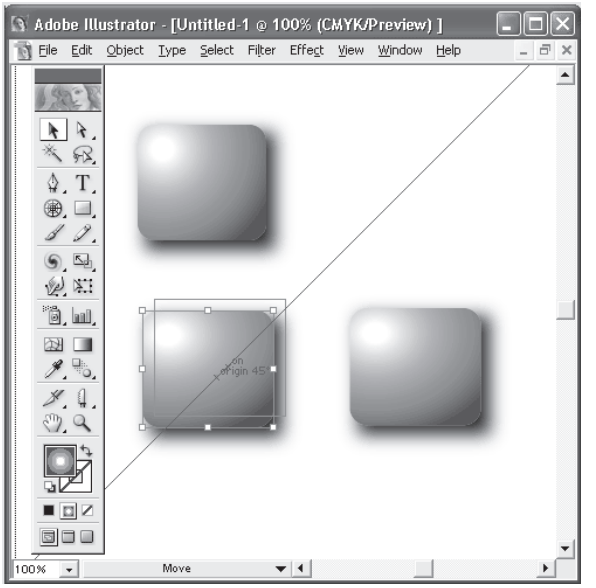


Рис. 8.11. Переміщення об’єкта з натиснутою клавішею <Alt>, що дозволяє отримати його копію

Для того щоб перемістити об’єкт на точно відому відстань, слід скористатися командою Move (Переміщення) меню Object | Transform (Об’єкт | Трансформація), яка відкриває діалогове вікно Move (Переміщення), яке зображено на рис.8.12.

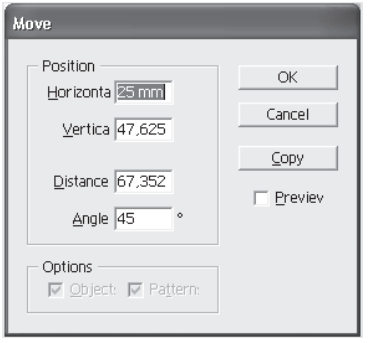


Рис. 8.12. Діалогове вікно Move

Якщо об’єкт, призначений для переміщення, вже виділений, то діалогове вікно Move (Переміщення) можна викликати подвійним клацанням на інструменті Selection (Виділення) ( ).

У діалоговому вікні відображаються результати попереднього переміщення або виміру.

Вирівнювання і розміщення об'єктів

У практиці графічного дизайну постійно потрібно вирівнювати об'єкти по одній лінії або розміщувати їх на рівній відстані один від одного.

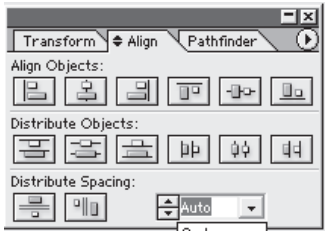


Рис. 8.13. Палітра Align

Полегшити цю достатньо рутинну і нудну операцію допомагає палітра Align (Вирівнювання) (рис. 8.13), яка відображається на екрані однойменною командою меню Window (Вікно).

У цій палітрі у ряді Align Objects (Вирівняти об'єкти) подані кнопки (зліва направо), які дозволяють вирівнювати виділені об'єкти.

У рядкуDistributeObjects (Розподілити об'єкти) подані кнопки, які дозволяють розміщувати виділені об'єкти:

– по рівних відстанях між верхніми краями;
– між горизонтальними центрами;
– між нижніми краями;
– між правими краями;
– між вертикальними центрами.

Обертання осей X і Y

За замовчуванням при відкритті нового документа осі X і Y розташовуються паралельно горизонтальній і вертикальній сторонах екрана (робочого листа). Осі можна за необхідності обернути на будь-який кут, якщо ввести відповідне значення в діалоговому вікні Preferences (Установки).

Обертання осей може бути потрібним, якщо всі елементи (у тому числі ще не створені) повинні розташовуватися під однаковим кутом. Після обертання осей на певний кут всі об'єкти, що поміщаються на сторінці, розташовуються під цим кутом (рис. 8.14).

У діалоговому вікні відображаються результати попереднього переміщення або виміру.

Вирівнювання і розміщення об'єктів

У практиці графічного дизайну постійно потрібно вирівнювати об'єкти по одній лінії або розміщувати їх на рівній відстані один від одного.

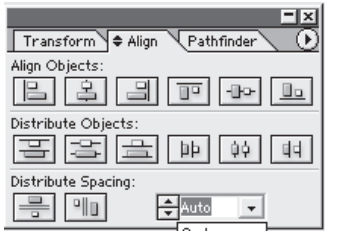


Рис. 8.13. Палітра Align

Полегшити цю достатньо рутинну і нудну операцію допомагає палітра Align (Вирівнювання) (рис. 8.13), яка відображається на екрані однойменною командою меню Window (Вікно).

У цій палітрі у ряді Align Objects (Вирівняти об'єкти) подані кнопки (зліва направо), які дозволяють вирівнювати виділені об'єкти.

У рядкуDistributeObjects (Розподілити об'єкти) подані кнопки, які дозволяють розміщувати виділені об'єкти:

– по рівних відстанях між верхніми краями;
– між горизонтальними центрами;
– між нижніми краями;
– між правими краями;
– між вертикальними центрами.

Обертання осей X і Y

За замовчуванням при відкритті нового документа осі X і Y розташовуються паралельно горизонтальній і вертикальній сторонах екрана (робочого листа). Осі можна за необхідності обернути на будь-який кут, якщо ввести відповідне значення в діалоговому вікні Preferences (Установки).

Обертання осей може бути потрібним, якщо всі елементи (у тому числі ще не створені) повинні розташовуватися під однаковим кутом. Після обертання осей на певний кут всі об'єкти, що поміщаються на сторінці, розташовуються під цим кутом (рис. 8.14).

Для того щоб установити кут повороту осей, необхідно командою Preferences (Установки) меню Edit (Правка) викликати діалогове вікно Preferences (Установки) і відкрити розділ General (Основні).

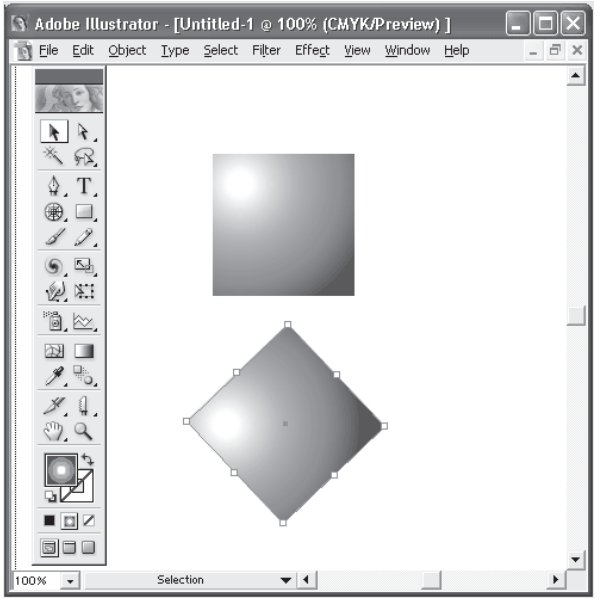


Рис. 8.14. Вплив установки значення в полі Constrain Angle: малювання еліпса в осях за замовчанням (1), малювання еліпса в осях, повернених на 30 градусів (2)

У полі Constrain Angle (Кут повороту осей) слід ввести значення кута обертання осей: позитивне значення обертає осі проти годинникової стрілки, а негативне – за годинниковою стрілкою.

Значення кута зберігається у файлі установок програми Adobe Illustrator, і все нові документи використовуватимуть цю установку до тих пір, поки не буде введено нове значення або не буде видалений файл установок (у останньому випадку програма повертає всі установки за замовчуванням).

Ця установка значень впливає на розташування текстових об’єктів, на кути градієнтних розтяжок, на об’єкти, які створюються

Для того щоб установити кут повороту осей, необхідно командою Preferences (Установки) меню Edit (Правка) викликати діалогове вікно Preferences (Установки) і відкрити розділ General (Основні).

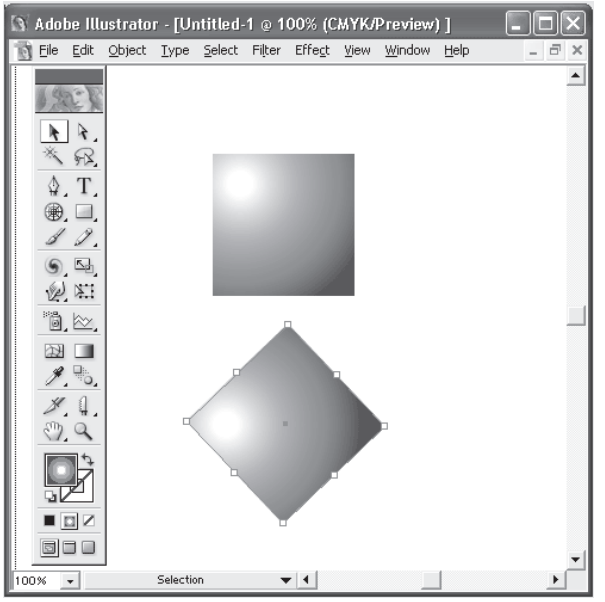


Рис. 8.14. Вплив установки значення в полі Constrain Angle: малювання еліпса в осях за замовчанням (1), малювання еліпса в осях, повернених на 30 градусів (2)

У полі Constrain Angle (Кут повороту осей) слід ввести значення кута обертання осей: позитивне значення обертає осі проти годинникової стрілки, а негативне – за годинниковою стрілкою.

Значення кута зберігається у файлі установок програми Adobe Illustrator, і все нові документи використовуватимуть цю установку до тих пір, поки не буде введено нове значення або не буде видалений файл установок (у останньому випадку програма повертає всі установки за замовчуванням).

Ця установка значень впливає на розташування текстових об’єктів, на кути градієнтних розтяжок, на об’єкти, які створюються

за допомогою інструментів груп Rectangle (Прямокутник), Ellipse (Еліпс) і Graph (Діаграма).

Кут обертання осей впливає і на операції масштабування, дзеркального віддзеркалення, переміщення об'єктів за допомогою клавіш управління курсором, а також на відображення кутів у палітрі Info (Інфо).

Розташування об'єктів у вертикальній "стопці"

У векторних програмах, у тому числі в програмі Adobe Illustrator, всі об'єкти незалежні один від одного: кожен об'єкт можна довільно переміщати і трансформувати без щонайменшого впливу на інші об'єкти. Проте якщо об'єкти перекриваються, то виявляється єдина залежність їх один від одного: об'єкт із заливкою, який розташовується на самому верхньому рівні, може перекривати будь-які об'єкти, що пролягають нижче.

Розташування векторних об'єктів на площині нагадує принцип аплікації: об'єкт, який "пришивається" або "приклеюється" раніше, може перекриватися об'єктом, який "пришивається" або "приклеюється" пізніше.

Це означає, що всі векторні об'єкти розташовуються у вигляді деякої нескінченної "стопки". Навіть найнепоказніший об'єкт має в своєму розпорядженні свій власний рівень у спільній "стопці". Природно, що об'єкти цієї "стопки" можуть вільно "пересуватися". Подальшим розвитком принципу аплікації є механізм шарів (layers), який ми розглянемо пізніше.

Вставка об'єкта в "стопку"

Команди Pastein Front (Вклеїти вперед) і Pastein Back (Вклеїти назад) меню Edit (Правка) дозволяють помістити об'єкт, який зберігається в буфері обміну Clipboard, під виділений об'єкт або над виділеним об'єктом.

Ці команди зручно використовувати замість команд Bring Forward (Зрушити вперед) і Send Backward (Зрушити назад), якщо в документі безліч об'єктів і потрібно помістити об'єкт щодо іншого об'єкта. Для цього використовується команда Cut (Вирізувати) меню Edit (Правка), яка тимчасово переносить виділений об'єкт у

за допомогою інструментів груп Rectangle (Прямокутник), Ellipse (Еліпс) і Graph (Діаграма).

Кут обертання осей впливає і на операції масштабування, дзеркального віддзеркалення, переміщення об'єктів за допомогою клавіш управління курсором, а також на відображення кутів у палітрі Info (Інфо).

Розташування об'єктів у вертикальній "стопці"

У векторних програмах, у тому числі в програмі Adobe Illustrator, всі об'єкти незалежні один від одного: кожен об'єкт можна довільно переміщати і трансформувати без щонайменшого впливу на інші об'єкти. Проте якщо об'єкти перекриваються, то виявляється єдина залежність їх один від одного: об'єкт із заливкою, який розташовується на самому верхньому рівні, може перекривати будь-які об'єкти, що пролягають нижче.

Розташування векторних об'єктів на площині нагадує принцип аплікації: об'єкт, який "пришивається" або "приклеюється" раніше, може перекриватися об'єктом, який "пришивається" або "приклеюється" пізніше.

Це означає, що всі векторні об'єкти розташовуються у вигляді деякої нескінченної "стопки". Навіть найнепоказніший об'єкт має в своєму розпорядженні свій власний рівень у спільній "стопці". Природно, що об'єкти цієї "стопки" можуть вільно "пересуватися". Подальшим розвитком принципу аплікації є механізм шарів (layers), який ми розглянемо пізніше.

Вставка об'єкта в "стопку"

Команди Pastein Front (Вклеїти вперед) і Pastein Back (Вклеїти назад) меню Edit (Правка) дозволяють помістити об'єкт, який зберігається в буфері обміну Clipboard, під виділений об'єкт або над виділеним об'єктом.

Ці команди зручно використовувати замість команд Bring Forward (Зрушити вперед) і Send Backward (Зрушити назад), якщо в документі безліч об'єктів і потрібно помістити об'єкт щодо іншого об'єкта. Для цього використовується команда Cut (Вирізувати) меню Edit (Правка), яка тимчасово переносить виділений об'єкт у

буфер обміну. Якщо при виконанні цих команд немає жодного виділеного об'єкта, то об'єкт з буфера поміщається на самий верхній або на самий нижній рівень відповідно.

При вставці з буфера декількох об'єктів всі вони зберігають власний порядок і розташовуються під або над виділеним об'єктом.

Для того щоб перемістити об'єкт на певний шар, необхідно виділити об'єкт, виділити шар у палітрі Layers (Шари), а потім виконати команду Moveto Current Layer (Переміщати на поточний шар) меню Object | Arrange (Об'єкт | Монтаж).

Палітра Transform

У палітрі Transform (Трансформація) (рис. 8.15) відображається деяка поточна інформація про виділений об'єкт, крім того, вона дозволяє трансформувати об'єкт (переміщати, масштабувати, обертати і змінювати розмір) шляхом введення нових значень у відповідні поля.

Для відображення палітри на екрані слід виконати команду Transform (Трансформація) меню Window (Вікно).

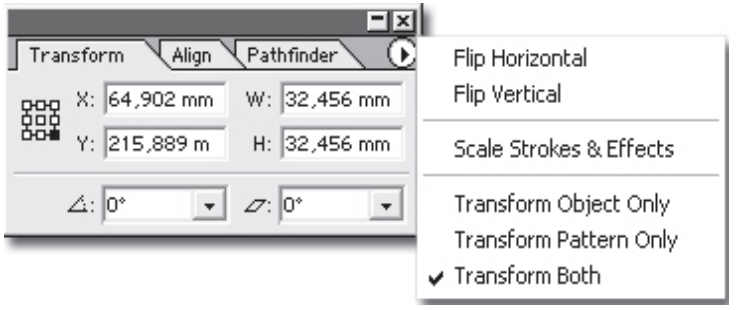


Рис. 8.15. Палітра Transform з розкритим меню команд

Як можна побачити з рисунка, у лівій частці палітри зображено умовне позначення основних (фіксованих) точок об'єкта (кутових точок, центрів сторін і центральної точки прямокутника, в який вписується об'єкт). Виділена точка служить точкою, щодо якої відбувається трансформація (наприклад, переміщення або обертання).

буфер обміну. Якщо при виконанні цих команд немає жодного виділеного об'єкта, то об'єкт з буфера поміщається на самий верхній або на самий нижній рівень відповідно.

При вставці з буфера декількох об'єктів всі вони зберігають власний порядок і розташовуються під або над виділеним об'єктом.

Для того щоб перемістити об'єкт на певний шар, необхідно виділити об'єкт, виділити шар у палітрі Layers (Шари), а потім виконати команду Moveto Current Layer (Переміщати на поточний шар) меню Object | Arrange (Об'єкт | Монтаж).

Палітра Transform

У палітрі Transform (Трансформація) (рис. 8.15) відображається деяка поточна інформація про виділений об'єкт, крім того, вона дозволяє трансформувати об'єкт (переміщати, масштабувати, обертати і змінювати розмір) шляхом введення нових значень у відповідні поля.

Для відображення палітри на екрані слід виконати команду Transform (Трансформація) меню Window (Вікно).

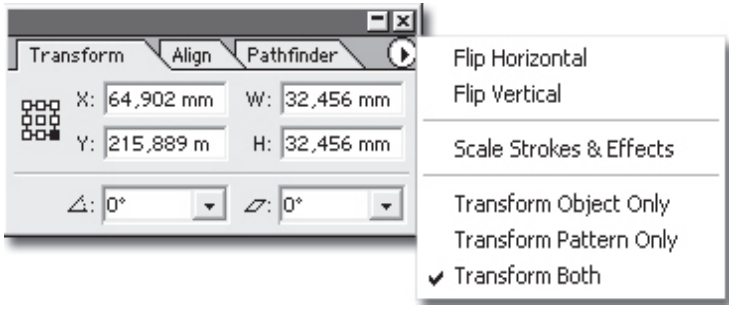


Рис. 8.15. Палітра Transform з розкритим меню команд

Як можна побачити з рисунка, у лівій частці палітри зображено умовне позначення основних (фіксованих) точок об'єкта (кутових точок, центрів сторін і центральної точки прямокутника, в який вписується об'єкт). Виділена точка служить точкою, щодо якої відбувається трансформація (наприклад, переміщення або обертання).

Для переміщення об’єкта по горизонталі і/або по вертикалі слід ввести відповідні значення в поля X і/або Y. Для визначення розміру об’єкта необхідно ввести відповідні значення в поля W (Ширина) і H (Висота).

Інструмент Measure

Робота оформлювача (дизайнера), на відміну від творчості вільного художника, вимагає часом великої точності в розташуванні елементів або витримки яких-небудь розмірів, порушення яких неприпустимо.

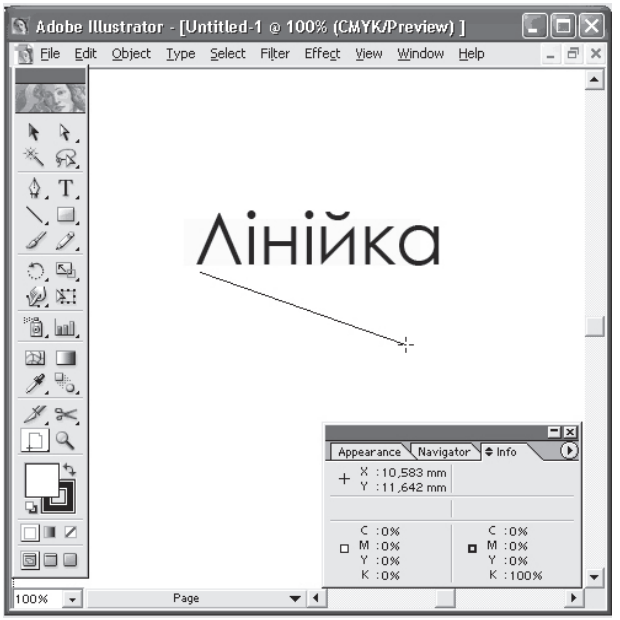


Рис. 8.16. Інструмент Measure і палітра Info

У програмі Adobe Illustrator передбачений спеціальний інструмент Measure (Лінійка) ( ), за допомогою якого легко зміряти відстань між будь-якими точками і визначити кут нахилу.

Свідчення інструменту відображаються в палітрі Info (Інфо) (рис.8.16):

- відстань по осях X і Y (поля X, Y, відповідно);

Для переміщення об’єкта по горизонталі і/або по вертикалі слід ввести відповідні значення в поля X і/або Y. Для визначення розміру об’єкта необхідно ввести відповідні значення в поля W (Ширина) і H (Висота).

Інструмент Measure

Робота оформлювача (дизайнера), на відміну від творчості вільного художника, вимагає часом великої точності в розташуванні елементів або витримки яких-небудь розмірів, порушення яких неприпустимо.

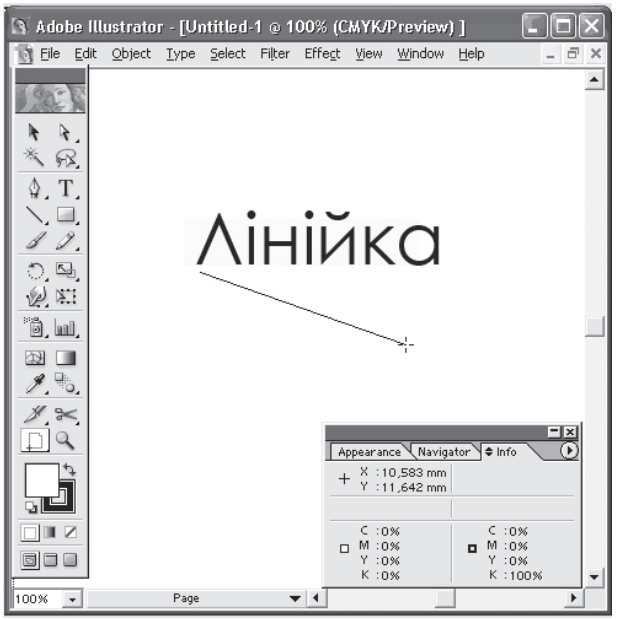


Рис. 8.16. Інструмент Measure і палітра Info

У програмі Adobe Illustrator передбачений спеціальний інструмент Measure (Лінійка) ( ), за допомогою якого легко зміряти відстань між будь-якими точками і визначити кут нахилу.

Свідчення інструменту відображаються в палітрі Info (Інфо) (рис.8.16):

- відстань по осях X і Y (поля X, Y, відповідно);

- абсолютна відстань по горизонталі і вертикалі (поля W, H, відповідно);
- спільна відстань (поле D);
- кут нахилу (поле ( )).

Всі значення, окрім кута, відображаються в одиниці виміру, визначеній у розділі Units&Undo (Одиниці виміру і відміна команд) діалогового вікна Preferences (Установки) або в діалоговому вікні Document Setup (Параметри документа).

Направляючі лінії і сітка

Найважливішими засобами, призначеними для підвищення точності роботи, є направляючі лінії (guides), аналог яких – тонкі олівцеві розмічальні лінії і сітка (grid), аналог якої – міліметрівка. Немає необхідності додавати, що комп'ютерні варіанти цих засобів мають значно більше корисних якостей, чим їх "бідні предки".

Розмірність сітки може довільно змінюватися в діапазоні від 0,01 до 342,77 мм.

Направляючі лінії і сітка можуть служити не лише в справі забезпечення точності і зручності, але і для створення модульних сіток, розмітки документа по естетичних критеріях пропорційності і міри.

Строго кажучи, термін "направляючі лінії" є узагальненням направляючих двох типів:

- лінії-направляючих, які перетинають по горизонталі і по вертикалі все робоче поле програми;
- об'єктів-направляючих, які конвертуються з будь-яких об'єктів (окрім шрифту).

За замовчуванням всі новостворювані направляючі лінії фіксуються (locked). Проте у будь-який момент фіксацію можна від-ключити і перемістити в нове положення, видалити, змінити параметри і так далі.

Для того щоб дістати можливість створювати направляючі лінії, необхідно вивести на екран вимірювальні лінійки, виконавши команду Show Rulers (Показати лінійки) меню View (Перегляд).

Вимірювальні лінійки необхідні, оскільки вони "приховують" нескінченне число допоміжних лінійок, які "витягуються" з них за допомогою кнопки миші (рис. 8.17):

- абсолютна відстань по горизонталі і вертикалі (поля W, H, відповідно);
- спільна відстань (поле D);
- кут нахилу (поле ( )).

Всі значення, окрім кута, відображаються в одиниці виміру, визначеній у розділі Units&Undo (Одиниці виміру і відміна команд) діалогового вікна Preferences (Установки) або в діалоговому вікні Document Setup (Параметри документа).

Направляючі лінії і сітка

Найважливішими засобами, призначеними для підвищення точності роботи, є направляючі лінії (guides), аналог яких – тонкі олівцеві розмічальні лінії і сітка (grid), аналог якої – міліметрівка. Немає необхідності додавати, що комп'ютерні варіанти цих засобів мають значно більше корисних якостей, чим їх "бідні предки".

Розмірність сітки може довільно змінюватися в діапазоні від 0,01 до 342,77 мм.

Направляючі лінії і сітка можуть служити не лише в справі забезпечення точності і зручності, але і для створення модульних сіток, розмітки документа по естетичних критеріях пропорційності і міри.

Строго кажучи, термін "направляючі лінії" є узагальненням направляючих двох типів:

- лінії-направляючих, які перетинають по горизонталі і по вертикалі все робоче поле програми;
- об'єктів-направляючих, які конвертуються з будь-яких об'єктів (окрім шрифту).

За замовчуванням всі новостворювані направляючі лінії фіксуються (locked). Проте у будь-який момент фіксацію можна від-ключити і перемістити в нове положення, видалити, змінити параметри і так далі.

Для того щоб дістати можливість створювати направляючі лінії, необхідно вивести на екран вимірювальні лінійки, виконавши команду Show Rulers (Показати лінійки) меню View (Перегляд).

Вимірювальні лінійки необхідні, оскільки вони "приховують" нескінченне число допоміжних лінійок, які "витягуються" з них за допомогою кнопки миші (рис. 8.17):

- з верхньої вимірювальної лінійки — горизонтальні лінійки;
- з лівої вимірювальної лінійки — вертикальні лінійки.

Втім, якщо утримувати клавішу <Alt>, то можна з верхньої лінійки отримати вертикальні направляючі, а з лівої – горизонтальні.

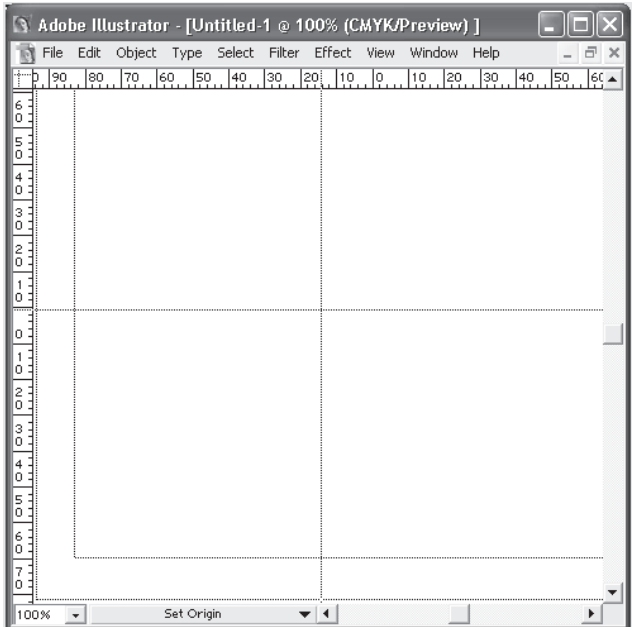


Рис. 8.17. Приклад горизонтальних і вертикальних направляючих ліній

Для того щоб будь-який об'єкт (або сукупність об'єктів) перетворити в напрямний (рис. 8.18), досить виділити його і виконати команду Make Guides (Утворити напрямні) меню View | Guides (Перегляд | Напрямні).

Якщо виникає необхідність виконати які-небудь дії (перемістити, видалити і т. д.) з направляючими лініями, спочатку їх потрібно звільнити від фіксації командою Lock Guides меню View | Guides (Перегляд | Напрямні).

- з верхньої вимірювальної лінійки — горизонтальні лінійки;
- з лівої вимірювальної лінійки — вертикальні лінійки.

Втім, якщо утримувати клавішу <Alt>, то можна з верхньої лінійки отримати вертикальні направляючі, а з лівої – горизонтальні.

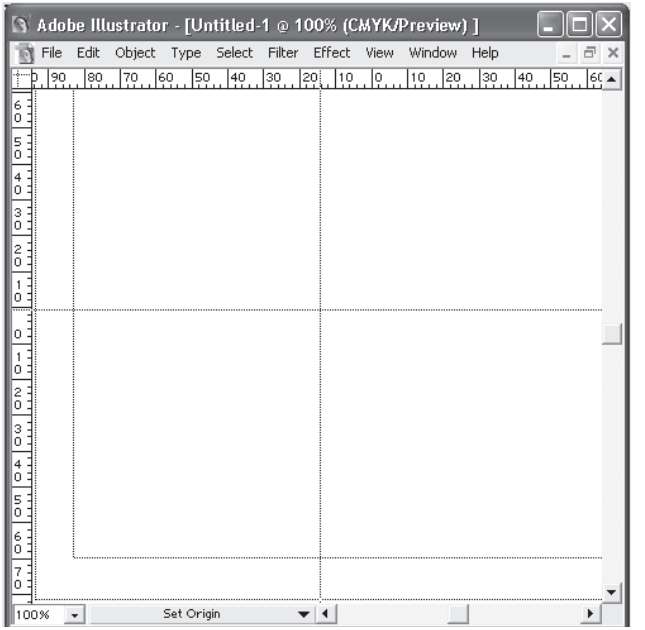


Рис. 8.17. Приклад горизонтальних і вертикальних направляючих ліній

Для того щоб будь-який об'єкт (або сукупність об'єктів) перетворити в напрямний (рис. 8.18), досить виділити його і виконати команду Make Guides (Утворити напрямні) меню View | Guides (Перегляд | Напрямні).

Якщо виникає необхідність виконати які-небудь дії (перемістити, видалити і т. д.) з направляючими лініями, спочатку їх потрібно звільнити від фіксації командою Lock Guides меню View | Guides (Перегляд | Напрямні).

Якщо команда відмічена "міткою", то всі допоміжні лінії фіксовані. Після виконання команди (зняття "мітки") всі допоміжні лінії стають доступними для роботи з ними. Направляючі лінії переміщаються, як і будь-які об'єкти, простим перетягненням за допомогою миші, а віддаляються за допомогою клавіші <Delete> або "поверненням" назад на вимірювальні лінійки.

Об'єкти-напрямні можна знову повернути в ранг "просто об'єктів". Для цього необхідно їх виділити і виконати команду Release Guides (Відмінити напрямні).

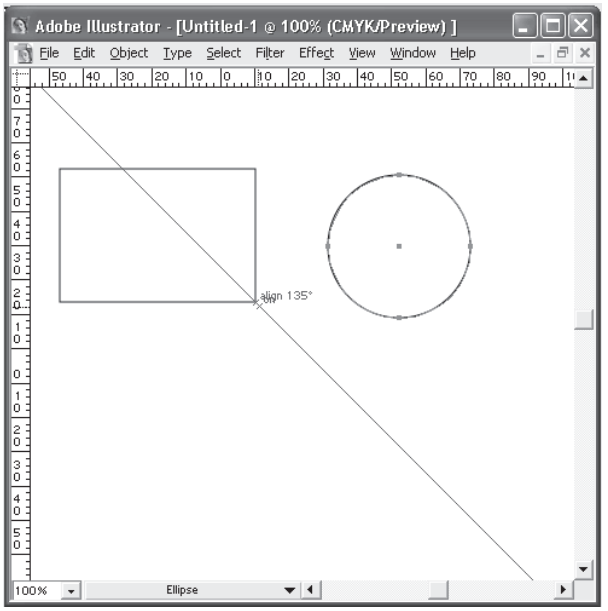


Рис. 8.18. Будь-який об'єкт, окрім шрифту, може бути напрямним

Проте наявність на робочому листі розмічальних ліній, незважаючи на їх необхідність і корисність, у певні моменти може заважати побачити свій твір у всій красі. Якщо видаляти допоміжні лінії ще рано, можна тимчасово їх заховати. У меню View | Guides подана команда Show Guides/Hide Guides (Показати / Приховати), яка видаляє лінії з поля зору або повертає їх назад.

Якщо команда відмічена "міткою", то всі допоміжні лінії фіксовані. Після виконання команди (зняття "мітки") всі допоміжні лінії стають доступними для роботи з ними. Направляючі лінії переміщаються, як і будь-які об'єкти, простим перетягненням за допомогою миші, а віддаляються за допомогою клавіші <Delete> або "поверненням" назад на вимірювальні лінійки.

Об'єкти-напрямні можна знову повернути в ранг "просто об'єктів". Для цього необхідно їх виділити і виконати команду Release Guides (Відмінити напрямні).

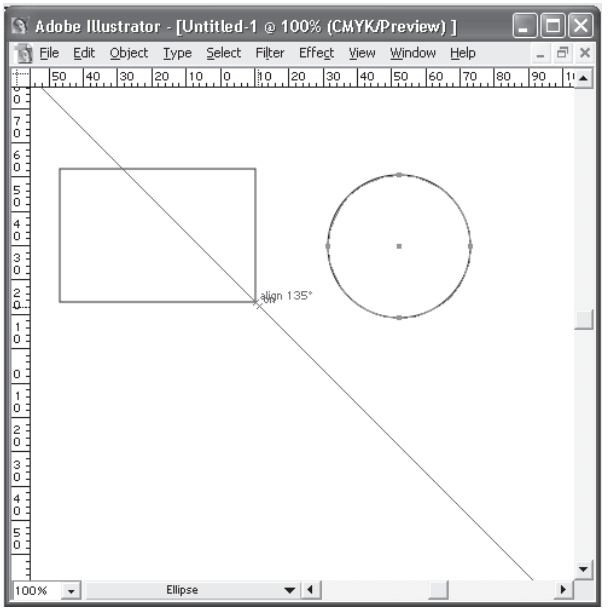


Рис. 8.18. Будь-який об'єкт, окрім шрифту, може бути напрямним

Проте наявність на робочому листі розмічальних ліній, незважаючи на їх необхідність і корисність, у певні моменти може заважати побачити свій твір у всій красі. Якщо видаляти допоміжні лінії ще рано, можна тимчасово їх заховати. У меню View | Guides подана команда Show Guides/Hide Guides (Показати / Приховати), яка видаляє лінії з поля зору або повертає їх назад.

Сітка

На відміну від направляючих ліній сітка задає регулярну систему ліній (рис. 8.19), форму і колір яких можна змінювати. Основним змінним параметром служить розмірність сітки, що задається двома значеннями: відстанню між основними лініями (наприклад, сантиметровими) і кількістю внутрішніх ліній (наприклад, 10).

Ви можете за необхідності відображення сітки на екрані включати або відключати. Для цього слід використовувати, відповідно, команду Show Grid (Показати сітку) або HideGrid (Заховати сітку) меню View (Перегляд).

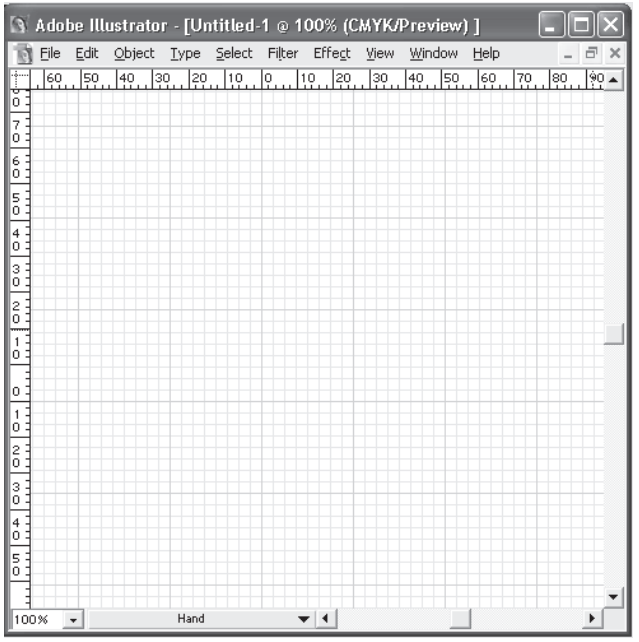


Рис. 8.19. Відображення сітки на екрані

Для установки режиму "прилипання" або його відключення необхідно виконати команду Snapto Grid (Вирівняти по сітці) меню View (Перегляд).

Сітка

На відміну від направляючих ліній сітка задає регулярну систему ліній (рис. 8.19), форму і колір яких можна змінювати. Основним змінним параметром служить розмірність сітки, що задається двома значеннями: відстанню між основними лініями (наприклад, сантиметровими) і кількістю внутрішніх ліній (наприклад, 10).

Ви можете за необхідності відображення сітки на екрані включати або відключати. Для цього слід використовувати, відповідно, команду Show Grid (Показати сітку) або HideGrid (Заховати сітку) меню View (Перегляд).

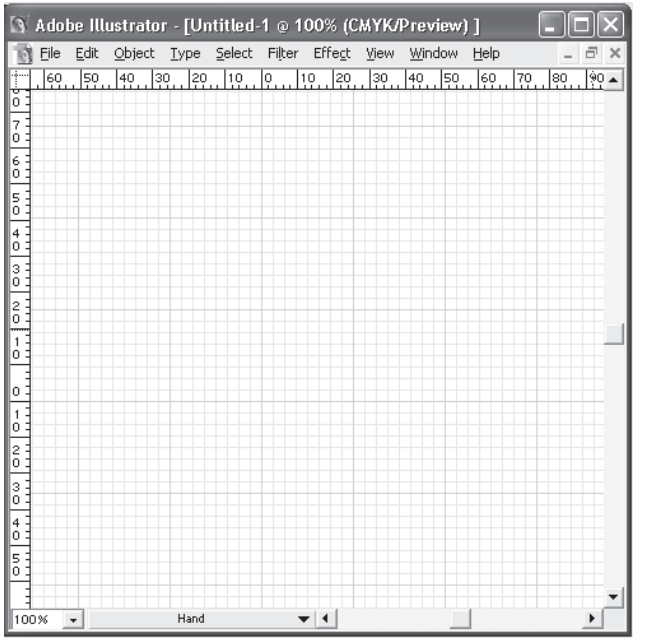


Рис. 8.19. Відображення сітки на екрані

Для установки режиму "прилипання" або його відключення необхідно виконати команду Snapto Grid (Вирівняти по сітці) меню View (Перегляд).

Об’єднання об’єктів у групу

У процесі роботи вдало знайдені поєднання об’єктів розумно об’єднувати в групу з тим, щоб захистити від можливих випадкових зрушень, а також для зручності подальших комбінаційних побудов. Адже, як відомо, в хорошій композиції все тримається на дуже точних розрахунках взаємного розташування об’єктів та їх масштабної взаємодії.

Об’єднання об’єктів у групу дозволяє поводитися з нею як з окремим об’єктом. Усі індивідуальні об’єкти, що входять до групи, підкоряються всім діям і операціям, вживаним до групи.

Наприклад, всі елементи фірмового знака, сполучені в групу, можна легко пересувати, масштабувати, обертати і виконувати з ними всі мислимі операції, не піклуючись про те, що який-небудь об’єкт випадково опиниться поза дією (ця ситуація можлива, коли об’єкти, що входять у фірмовий знак, вибираються послідовно або за допомогою штрихової рамки – який-небудь об’єкт може залишитися невиділеним і до нього не будуть застосовані операції).

Для того щоб помістити сукупність об’єктів у групу, їх необхідно виділити і виконати команду Group (Згрупувати) меню Object (Об’єкт). Для зворотної операції призначена команда Ungroup (Розгрупувати) того ж меню. Група може бути "вкладеною", тобто мати певну ієрархію (рис. 8.20).

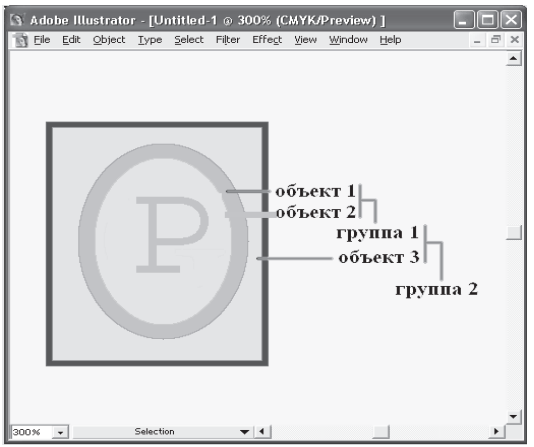


Рис. 8.20. Ієрархія об’єктів у двох групах

Об’єднання об’єктів у групу

У процесі роботи вдало знайдені поєднання об’єктів розумно об’єднувати в групу з тим, щоб захистити від можливих випадкових зрушень, а також для зручності подальших комбінаційних побудов. Адже, як відомо, в хорошій композиції все тримається на дуже точних розрахунках взаємного розташування об’єктів та їх масштабної взаємодії.

Об’єднання об’єктів у групу дозволяє поводитися з нею як з окремим об’єктом. Усі індивідуальні об’єкти, що входять до групи, підкоряються всім діям і операціям, вживаним до групи.

Наприклад, всі елементи фірмового знака, сполучені в групу, можна легко пересувати, масштабувати, обертати і виконувати з ними всі мислимі операції, не піклуючись про те, що який-небудь об’єкт випадково опиниться поза дією (ця ситуація можлива, коли об’єкти, що входять у фірмовий знак, вибираються послідовно або за допомогою штрихової рамки – який-небудь об’єкт може залишитися невиділеним і до нього не будуть застосовані операції).

Для того щоб помістити сукупність об’єктів у групу, їх необхідно виділити і виконати команду Group (Згрупувати) меню Object (Об’єкт). Для зворотної операції призначена команда Ungroup (Розгрупувати) того ж меню. Група може бути "вкладеною", тобто мати певну ієрархію (рис. 8.20).

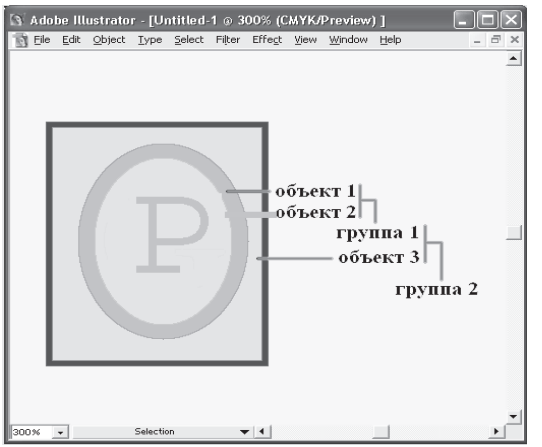


Рис. 8.20. Ієрархія об’єктів у двох групах

У цьому полягає прекрасна можливість будувати композицію з окремих закінчених сукупностей об'єктів, які поступово, за необхідністю, включаються в групу. Розгруповування вкладеної групи здійснюється в зворотному порядку і вимагає стільки ж етапів, скільки і її створення.

Фіксація і приховування об'єктів

Окрім фіксації положення об'єктів відносно один одного в групі існують і інші корисні функції щодо забезпечення недоторканості об'єктів – фіксація об'єктів відносно сторінки, а також тимчасове "видалення" їх з екрана.

Об'єкти, які фіксовані на сторінці або приховані, вже не можна виділити, а отже, не можна застосувати до них які-небудь дії. Ця можливість особливо може стати в нагоді при роботі з безліччю об'єктів, що перекриваються.

Фіксовані об'єкти зберігають цю свою властивість навіть після збереження файла і повторного відкриття, а приховані об'єкти при повторному відкритті втрачають його і стають видимими.

Для того щоб зафіксувати виділені об'єкти, необхідно виконати команду Lock (Закріпити) меню Object (Об'єкт). Можна, навпаки, зафіксувати невиділені об'єкти, але при виконанні команди Lock (Закріпити) слід утримувати клавішу <Alt>.

Для того щоб приховати виділені об'єкти, необхідно виконати команду Hide Selection (Заховати) меню Object (Об'єкт). Заховати можна і невиділені об'єкти, але при виконанні команди слід утримувати комбінацію клавіш <Shift>+<Alt>.

Для того щоб зняти фіксацію об'єктів, досить виконати команду Unlock All (Звільнити все) меню Object (Об'єкт). Всі фіксовані об'єкти будуть одночасно "звільнені". Так само можна повернути на екран всі приховані об'єкти, виконавши команду Show All (Показати все) меню Object (Об'єкт).

8.3. Векторні трансформації і фільтри

Векторна програма Adobe Illustrator дозволяє легко змінювати форму і розміри об'єктів, їх орієнтацію в просторі. Разом з тим, ок-

У цьому полягає прекрасна можливість будувати композицію з окремих закінчених сукупностей об'єктів, які поступово, за необхідністю, включаються в групу. Розгруповування вкладеної групи здійснюється в зворотному порядку і вимагає стільки ж етапів, скільки і її створення.

Фіксація і приховування об'єктів

Окрім фіксації положення об'єктів відносно один одного в групі існують і інші корисні функції щодо забезпечення недоторканості об'єктів – фіксація об'єктів відносно сторінки, а також тимчасове "видалення" їх з екрана.

Об'єкти, які фіксовані на сторінці або приховані, вже не можна виділити, а отже, не можна застосувати до них які-небудь дії. Ця можливість особливо може стати в нагоді при роботі з безліччю об'єктів, що перекриваються.

Фіксовані об'єкти зберігають цю свою властивість навіть після збереження файла і повторного відкриття, а приховані об'єкти при повторному відкритті втрачають його і стають видимими.

Для того щоб зафіксувати виділені об'єкти, необхідно виконати команду Lock (Закріпити) меню Object (Об'єкт). Можна, навпаки, зафіксувати невиділені об'єкти, але при виконанні команди Lock (Закріпити) слід утримувати клавішу <Alt>.

Для того щоб приховати виділені об'єкти, необхідно виконати команду Hide Selection (Заховати) меню Object (Об'єкт). Заховати можна і невиділені об'єкти, але при виконанні команди слід утримувати комбінацію клавіш <Shift>+<Alt>.

Для того щоб зняти фіксацію об'єктів, досить виконати команду Unlock All (Звільнити все) меню Object (Об'єкт). Всі фіксовані об'єкти будуть одночасно "звільнені". Так само можна повернути на екран всі приховані об'єкти, виконавши команду Show All (Показати все) меню Object (Об'єкт).

8.3. Векторні трансформації і фільтри

Векторна програма Adobe Illustrator дозволяє легко змінювати форму і розміри об'єктів, їх орієнтацію в просторі. Разом з тим, ок-

316

рім ручних операцій для векторних об'єктів передбачені багаточисельні засоби по їх зміні і деформації – векторні ефекти, частка з яких реалізується у вигляді інструментів, а частка – у вигляді фільтрів.

Векторні фільтри не слід плутати з фільтрами піксельної графіки, які також включені в програму. Опис піксельних фільтрів буде розглянуто далі.

Трансформуючі інструменти

Виділені об'єкти можна трансформувати — змінювати їх розмір, масштабувати, обертати і так далі. Для цього в програмі передбачений різноманітний інструментарій.

До інструментів, які змінюють форму об'єктів (трансформують їх), відносяться інструменти Rotate (Поворот) (), Reflect (Дзеркало) (), Scale (Розмір) (), Shear (Нахил) () і Blend (Перетворення) ().

З цими інструментами можна працювати в інтерактивному режимі або використовувати відповідне діалогове вікно, яке дозволяє вводити точні числові дані і змінювати різні параметри роботи інструменту. Ці діалогові вікна і команди можна викликати за допомогою контекстних меню, які відкриваються клацанням правої кнопки миші.

Інструмент Rotate

За допомогою інструменту Rotate (Поворот) можна обернути будь-яку сукупність об'єктів на довільний кут не лише навколо геометричного центру, але і навколо точки "додатка" (центру пере-творення), яка за замовчуванням хоча і розташовується в геометричному центрі об'єкта або групи об'єктів, але може бути переміщена.

Під час обертання об'єкти можна копіювати, що дозволяє достатньо легко створювати всілякі декоративні елементи (розетки, кругові орнаменти і т. п.).

Для обертання об'єкта в інтерактивному режимі його необхідно виділити і включити інструмент Rotate (Поворот). Далі, якщо об'єкт треба обертати навколо свого власного центру, досить натискувати кнопку миші і переміщати її в потрібному напрямі. Якщо ж

рім ручних операцій для векторних об'єктів передбачені багаточисельні засоби по їх зміні і деформації – векторні ефекти, частка з яких реалізується у вигляді інструментів, а частка – у вигляді фільтрів.

Векторні фільтри не слід плутати з фільтрами піксельної графіки, які також включені в програму. Опис піксельних фільтрів буде розглянуто далі.

Трансформуючі інструменти

Виділені об'єкти можна трансформувати — змінювати їх розмір, масштабувати, обертати і так далі. Для цього в програмі передбачений різноманітний інструментарій.

До інструментів, які змінюють форму об'єктів (трансформують їх), відносяться інструменти Rotate (Поворот) (), Reflect (Дзеркало) (), Scale (Розмір) (), Shear (Нахил) () і Blend (Перетворення) ().

З цими інструментами можна працювати в інтерактивному режимі або використовувати відповідне діалогове вікно, яке дозволяє вводити точні числові дані і змінювати різні параметри роботи інструменту. Ці діалогові вікна і команди можна викликати за допомогою контекстних меню, які відкриваються клацанням правої кнопки миші.

Інструмент Rotate

За допомогою інструменту Rotate (Поворот) можна обернути будь-яку сукупність об'єктів на довільний кут не лише навколо геометричного центру, але і навколо точки "додатка" (центру пере-творення), яка за замовчуванням хоча і розташовується в геометричному центрі об'єкта або групи об'єктів, але може бути переміщена.

Під час обертання об'єкти можна копіювати, що дозволяє достатньо легко створювати всілякі декоративні елементи (розетки, кругові орнаменти і т. п.).

Для обертання об'єкта в інтерактивному режимі його необхідно виділити і включити інструмент Rotate (Поворот). Далі, якщо об'єкт треба обертати навколо свого власного центру, досить натискувати кнопку миші і переміщати її в потрібному напрямі. Якщо ж

об’єкт (або об’єкти) необхідно обертати навколо іншої точки, то точку обертання слід перенести, клацнувши лівою кнопкою миші в передбачуваному центрі обертання (рис. 8.21).

Для того щоб початковий об’єкт залишився на місці, а в новому положенні опинилася копія (обертання з копіюванням), необхідно під час обертання утримувати натиснутою клавішу <Alt>.

При цьому слід звернути увагу на сам механізм обертання за допомогою миші: чим далі від об’єкта, що обертається, знаходиться "важіль обертання", тим точніше можна виконати обертання.

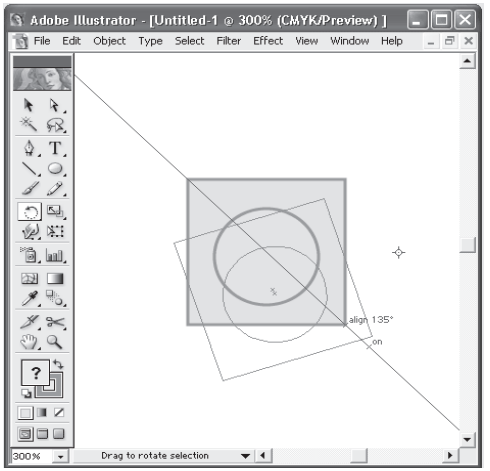


Рис. 8.21.Інструмент Rotate в палітрі інструментів і процес обертання при переміщеному центрі перетворення

Віддзеркалення

За допомогою інструменту Free Transform (Вільна трансформація) можна виконати віддзеркалення об’єкта. Для цього необхідно за допомогою інструменту Selection (Виділення) виділити об’єкт або об’єкти, призначені для віддзеркалення, включити інструмент Free Transform (Вільна трансформація) і, захопивши відповідний маркер "габаритного" прямокутника, протягнути його через об’єкт.

На жаль, цей спосіб віддзеркалення не дає можливості упевнено отримати точну копію об’єкта.

об’єкт (або об’єкти) необхідно обертати навколо іншої точки, то точку обертання слід перенести, клацнувши лівою кнопкою миші в передбачуваному центрі обертання (рис. 8.21).

Для того щоб початковий об’єкт залишився на місці, а в новому положенні опинилася копія (обертання з копіюванням), необхідно під час обертання утримувати натиснутою клавішу <Alt>.

При цьому слід звернути увагу на сам механізм обертання за допомогою миші: чим далі від об’єкта, що обертається, знаходиться "важіль обертання", тим точніше можна виконати обертання.

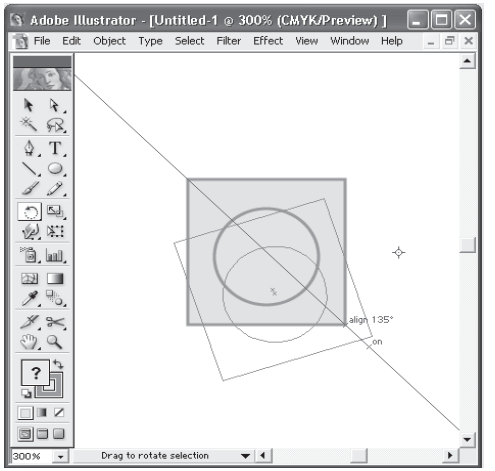


Рис. 8.21.Інструмент Rotate в палітрі інструментів і процес обертання при переміщеному центрі перетворення

Віддзеркалення

За допомогою інструменту Free Transform (Вільна трансформація) можна виконати віддзеркалення об’єкта. Для цього необхідно за допомогою інструменту Selection (Виділення) виділити об’єкт або об’єкти, призначені для віддзеркалення, включити інструмент Free Transform (Вільна трансформація) і, захопивши відповідний маркер "габаритного" прямокутника, протягнути його через об’єкт.

На жаль, цей спосіб віддзеркалення не дає можливості упевнено отримати точну копію об’єкта.

Масштабування

Якщо при роботі з піксельною графікою бажано якомога менше застосовувати зменшення і збільшення зображення, то у векторних програмах масштабування – найпоширеніша (після переміщення) операція.

Масштабування полягає в зміні розміру об'єкта. Розрізняють пропорційне і непропорційне масштабування. Існує декілька способів масштабування. Масштабування можна здійснити за допомогою маркерів "габаритного" прямокутника (bounding box). Для виконання масштабування необхідно виділити об'єкт або сукупність об'єктів.

Переміщення бічних маркерів забезпечує масштабування тільки по горизонталі або по вертикалі, а переміщення кутових маркерів – одночасно по горизонталі і по вертикалі.

Якщо необхідно зберегти пропорції, слід використовувати клавішу <Shift>. Клавіша <Alt> дозволить виконати масштабування з центру об'єкта.

За допомогою інструменту Free Transform (Вільна трансформація) теж можна виконати масштабування. Для цього необхідно за допомогою інструменту Selection (Виділення) виділити об'єкт або об'єкти, призначені для масштабування, включити інструмент Free Transform (Вільна трансформація) і, захопивши відповідний маркер "габаритного" прямокутника, протягнути його в потрібному напрямі.

Штрихування

Фільтр PenandInk (Штрихування) – оригінальний векторний фільтр, який імітує малюнок пером і тушшю і служить для створення різних штрихувань, наприклад "дерев'яних" узорів, а також штрихувань з випадковим малюнком.

Достатньо складний малюнок може включати величезну кількість об'єктів (рис. 8.22), що є причиною значного об'єму файла і збільшення часу обробки документа з такими малюнками. Тому використовувати фільтр PenandInk (Штрихування) слід на завершальній стадії підготовки документа.

Для роботи з штрихуванням призначені команди, які розташовуються в меню Filter | PenandInk(Фільтр | Штрихування).

Масштабування

Якщо при роботі з піксельною графікою бажано якомога менше застосовувати зменшення і збільшення зображення, то у векторних програмах масштабування – найпоширеніша (після переміщення) операція.

Масштабування полягає в зміні розміру об'єкта. Розрізняють пропорційне і непропорційне масштабування. Існує декілька способів масштабування. Масштабування можна здійснити за допомогою маркерів "габаритного" прямокутника (bounding box). Для виконання масштабування необхідно виділити об'єкт або сукупність об'єктів.

Переміщення бічних маркерів забезпечує масштабування тільки по горизонталі або по вертикалі, а переміщення кутових маркерів – одночасно по горизонталі і по вертикалі.

Якщо необхідно зберегти пропорції, слід використовувати клавішу <Shift>. Клавіша <Alt> дозволить виконати масштабування з центру об'єкта.

За допомогою інструменту Free Transform (Вільна трансформація) теж можна виконати масштабування. Для цього необхідно за допомогою інструменту Selection (Виділення) виділити об'єкт або об'єкти, призначені для масштабування, включити інструмент Free Transform (Вільна трансформація) і, захопивши відповідний маркер "габаритного" прямокутника, протягнути його в потрібному напрямі.

Штрихування

Фільтр PenandInk (Штрихування) – оригінальний векторний фільтр, який імітує малюнок пером і тушшю і служить для створення різних штрихувань, наприклад "дерев'яних" узорів, а також штрихувань з випадковим малюнком.

Достатньо складний малюнок може включати величезну кількість об'єктів (рис. 8.22), що є причиною значного об'єму файла і збільшення часу обробки документа з такими малюнками. Тому використовувати фільтр PenandInk (Штрихування) слід на завершальній стадії підготовки документа.

Для роботи з штрихуванням призначені команди, які розташовуються в меню Filter | PenandInk(Фільтр | Штрихування).

Спеціальні фільтри для об’єктів

Програма Adobe Illustrator має в своєму розпорядженні засоби, що іменуються по аналогії з програмами піксельної графіки фільтрами, які дозволяють змінювати форму об’єктів. Багато хто з цих фільтрів повторюється в меню Effect (Ефект). Як приклад розглянемо фільтр ScribbleandTweak.

Фільтр Scribbleand Tweak (Карлючки і помарки) меню Filter | Distort&Transform (Фільтр | Спотворення і трансформація) служить для довільної деформації об’єктів у заданих межах.

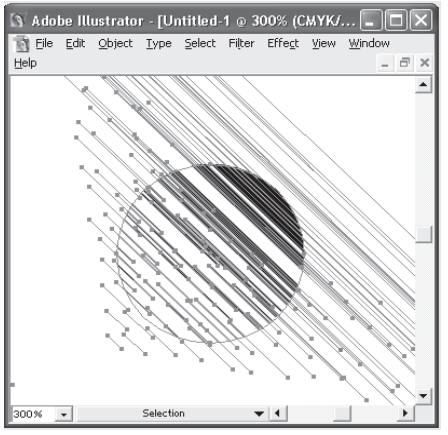


Рис. 8.22. Зовнішній вигляд штрихування

Під час роботи даного фільтра відбувається випадкове переміщення опорних (anchor points) точок та точок управління(control points).

У діалоговому вікні Scribble and Tweak (Карлючки і помарки) (рис.8.23) здійснюється управління деформацією виділеного об’єкта по горизонталі і по вертикалі в діапазоні від 0 до 100%.

Варіант Scribble (Карлючки) забезпечує переміщення опорних точок випадковим чином у сторони від початкового стану, а варіант Tweak (Помарки) – у межах виділеного об’єкта.

Фільтр Drop Shadow (Тінь) меню Filter | Stylize (Фільтр | Стилізація) дозволяє без проблем створити ефект падаючої тіні. Цей

Спеціальні фільтри для об’єктів

Програма Adobe Illustrator має в своєму розпорядженні засоби, що іменуються по аналогії з програмами піксельної графіки фільтрами, які дозволяють змінювати форму об’єктів. Багато хто з цих фільтрів повторюється в меню Effect (Ефект). Як приклад розглянемо фільтр ScribbleandTweak.

Фільтр Scribbleand Tweak (Карлючки і помарки) меню Filter | Distort&Transform (Фільтр | Спотворення і трансформація) служить для довільної деформації об’єктів у заданих межах.

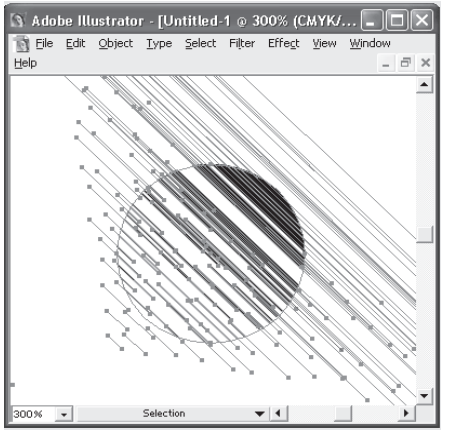


Рис. 8.22. Зовнішній вигляд штрихування

Під час роботи даного фільтра відбувається випадкове переміщення опорних (anchor points) точок та точок управління(control points).

У діалоговому вікні Scribble and Tweak (Карлючки і помарки) (рис.8.23) здійснюється управління деформацією виділеного об’єкта по горизонталі і по вертикалі в діапазоні від 0 до 100%.

Варіант Scribble (Карлючки) забезпечує переміщення опорних точок випадковим чином у сторони від початкового стану, а варіант Tweak (Помарки) – у межах виділеного об’єкта.

Фільтр Drop Shadow (Тінь) меню Filter | Stylize (Фільтр | Стилізація) дозволяє без проблем створити ефект падаючої тіні. Цей

фільтр автоматично виконує серію операцій по дублюванню об’єкта, переміщенню його по горизонталі і вертикалі, а також на один рівень нижче за об’єкт, по фарбуванню його в заданий відтінок і, нарешті, угруповання.

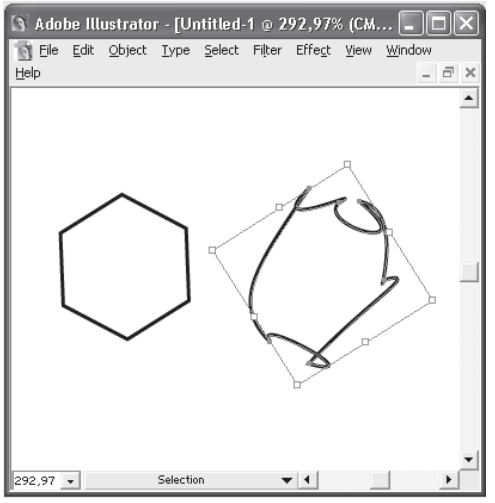


Рис. 8.23. Приклад застосування фільтра Scribbleand Tweak

У списку Mode (Режими накладення) можна вибрати відповідний режим накладення, найчастіше для тіні використовується режим Multiply (Множення).

Включивши перемикач Color (Колір), можна викликати діалогове вікно ColorPicker (Колірна палітра), в якому визначити довільний колір для тіні. Включивши перемикач Darkness (Інтенсивність затінювання), в полі справа можна встановити процентний вміст чорного кольору, що додається до кольору об’єкта в діапазоні від –0 до 100%. У полі Blur (Розмиття) визначається ступінь розмитості країв тіні в діапазоні від 0 до 50,79 мм.

Перетворення векторних об’єктів у растрове зображення

Команда Rasterize (Растеризувати) меню Object (Об’єкт) призначена для конвертації векторних об’єктів, що створюються в програмі Adobe Illustrator, в піксельне зображення, яке стає цілісним

фільтр автоматично виконує серію операцій по дублюванню об’єкта, переміщенню його по горизонталі і вертикалі, а також на один рівень нижче за об’єкт, по фарбуванню його в заданий відтінок і, нарешті, угруповання.

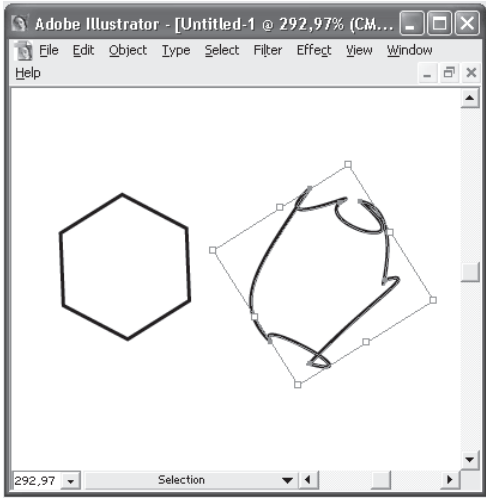


Рис. 8.23. Приклад застосування фільтра Scribbleand Tweak

У списку Mode (Режими накладення) можна вибрати відповідний режим накладення, найчастіше для тіні використовується режим Multiply (Множення).

Включивши перемикач Color (Колір), можна викликати діалогове вікно ColorPicker (Колірна палітра), в якому визначити довільний колір для тіні. Включивши перемикач Darkness (Інтенсивність затінювання), в полі справа можна встановити процентний вміст чорного кольору, що додається до кольору об’єкта в діапазоні від –0 до 100%. У полі Blur (Розмиття) визначається ступінь розмитості країв тіні в діапазоні від 0 до 50,79 мм.

Перетворення векторних об’єктів у растрове зображення

Команда Rasterize (Растеризувати) меню Object (Об’єкт) призначена для конвертації векторних об’єктів, що створюються в програмі Adobe Illustrator, в піксельне зображення, яке стає цілісним

об’єктом і може піддаватися обробці за допомогою фільтрів піксельної графіки, також включених у програму Adobe Illustrator.

Для конвертації слід виділити потрібні об’єкти і виконати команду Rasterize (Растеризувати), яка виводить на екран однойменне діалогове вікно (рис. 8.24).

У списку ColorModel (Колірний режим), що розкривається, можна вибрати необхідну колірну модель: RGB, CMYK, Grayscale (Градації сірого), Bitmap (Бітовий). Останній варіант означає чорно-біле штрихове зображення (глибина кольору дорівнює 1 біту).

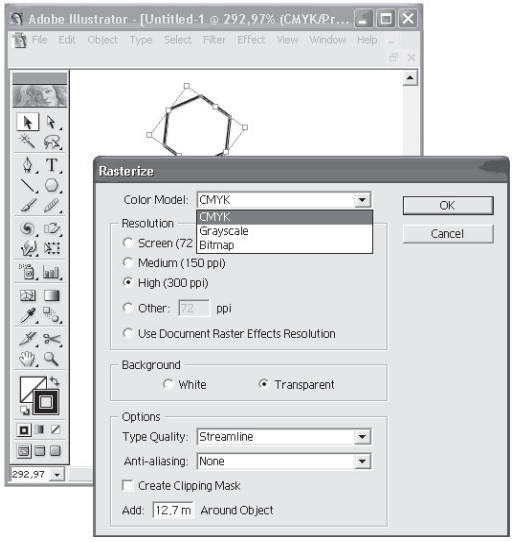


Рис. 8.24. Діалогове вікно Rasterize, початковий векторний об’єкт і отримане піксельне зображення

Включення якого-небудь перемикача з групи Resolution (Дозвіл) надає можливість вибору рівня дозволу для піксельного зображення:

- перемикач Screen (Екранне) вибирається, якщо піксельне зображення призначене для виводу на екран, наприклад, як елемент комп’ютерної презентації або Web-сторінка;

об’єктом і може піддаватися обробці за допомогою фільтрів піксельної графіки, також включених у програму Adobe Illustrator.

Для конвертації слід виділити потрібні об’єкти і виконати команду Rasterize (Растеризувати), яка виводить на екран однойменне діалогове вікно (рис. 8.24).

У списку ColorModel (Колірний режим), що розкривається, можна вибрати необхідну колірну модель: RGB, CMYK, Grayscale (Градації сірого), Bitmap (Бітовий). Останній варіант означає чорно-біле штрихове зображення (глибина кольору дорівнює 1 біту).

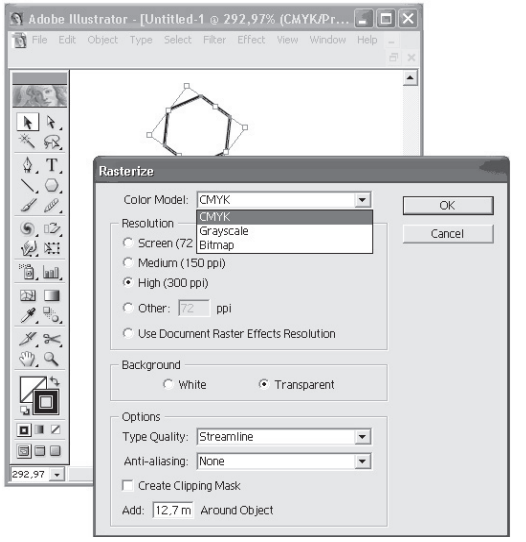


Рис. 8.24. Діалогове вікно Rasterize, початковий векторний об’єкт і отримане піксельне зображення

Включення якого-небудь перемикача з групи Resolution (Дозвіл) надає можливість вибору рівня дозволу для піксельного зображення:

- перемикач Screen (Екранне) вибирається, якщо піксельне зображення призначене для виводу на екран, наприклад, як елемент комп’ютерної презентації або Web-сторінка;

- перемикач Medium (Середнє) встановлюється, якщо піксельне зображення призначене для виводу на струменевий або лазерний принтер (офісний принтер);
- перемикач High (Високе) вибирається, якщо піксельне зображення призначене для виводу на високоякісний лазерний принтер або фотонабірний автомат;
- у полі праворуч від перемикача Other (Інше) можна ввести довільне значення дозволу в діапазоні від 1 до 2400 dpi (пікселів на дюйм);
- перемикач Use Document Raster Effects Resolution (Використовувати дозвіл документа) призначений для обліку глобальної установки дозволу документа. Такий дозвіл визначається командою Document Praster Effects Settings (Установки дозволу документа) меню Effect (Ефект).

До групи Background (Фон) входять два перемикачі:

- White (Білий);
- Transparent (Прозорий).

У групі Options (Додаткові параметри) визначаються параметри згладжування і обтравочного контура.

Прапорець Create Clipping Mask (Створити обтравочну маску) забезпечує обтравочний контур (clipping path), який дозволяє зберегти прозорі області зображення, що дуже важливо для складної верстки.

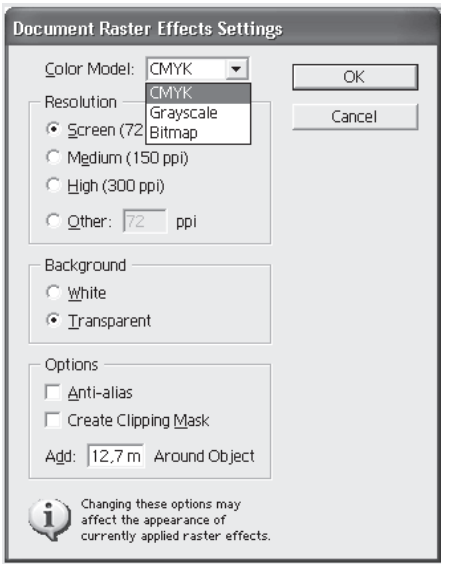


Рис. 8.25.Діалогове вікно Document Raster Effects Settings

У полі Add ... Around Object (Додати ... навколо об'єкта) можна задати поля навколо малюнка в діапазоні від 0 до 254 мм.

- перемикач Medium (Середнє) встановлюється, якщо піксельне зображення призначене для виводу на струменевий або лазерний принтер (офісний принтер);
- перемикач High (Високе) вибирається, якщо піксельне зображення призначене для виводу на високоякісний лазерний принтер або фотонабірний автомат;
- у полі праворуч від перемикача Other (Інше) можна ввести довільне значення дозволу в діапазоні від 1 до 2400 dpi (пікселів на дюйм);
- перемикач Use Document Raster Effects Resolution (Використовувати дозвіл документа) призначений для обліку глобальної установки дозволу документа. Такий дозвіл визначається командою Document Praster Effects Settings (Установки дозволу документа) меню Effect (Ефект).

До групи Background (Фон) входять два перемикачі:

- White (Білий);
- Transparent (Прозорий).

У групі Options (Додаткові параметри) визначаються параметри згладжування і обтравочного контура.

Прапорець Create Clipping Mask (Створити обтравочну маску) забезпечує обтравочний контур (clipping path), який дозволяє зберегти прозорі області зображення, що дуже важливо для складної верстки.

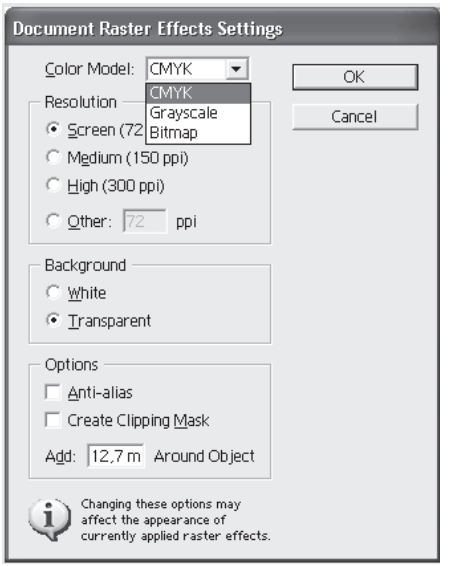


Рис. 8.25.Діалогове вікно Document Raster Effects Settings

У полі Add ... Around Object (Додати ... навколо об'єкта) можна задати поля навколо малюнка в діапазоні від 0 до 254 мм.

323

Палітра Pathfinder

Палітра Pathfinder (Обробка контурів), яка викликається на екран однойменною командою меню Window (Вікно), призначена для комбінування об'єктів різними способами, часто з утворенням складених контурів (compound path). Палітра Pathfinder (Обробка контурів) є дуже потужним засобом векторного формоутворення, що надзвичайно цінно для створення досить складних форм. У зв'язку з цим можна тимчасово не звертати уваги на параметри заливок і обведення.

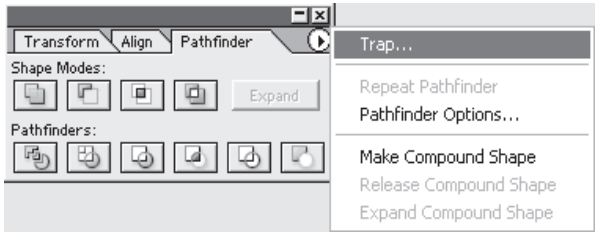


Рис. 8.26. Палітра Pathfinder з відкритим меню

При комбінуванні контурів з різними кольоровими заливками, як правило, результуючому об'єкта привласнюється заливка верхнього об'єкта. Для комбінування об'єктів слід розмістити і виділити необхідні об'єкти, а потім натискувати одну з кнопок палітри Pathfinder (Обробка контурів).

Залежно від виконуваної операції на екран може бути виведене діалогове вікно, в якому слід ввести необхідні додаткові дані.

Біля команд палітри Pathfinder (Обробка контурів) існує декілька спільних установок, які змінюються в діалоговому вікні PathfinderOptions (Обробка контурів: Параметри), яке викликається командою PathfinderOptions (Параметри обробки контурів) меню палітри.

Робота з масками

Механізм маскування – надзвичайно продуктивний спосіб створення складних графічних композицій. Ми вже розглядали роботу цього механізму у редакторі Photoshop, глава 7. Його ефективність

Палітра Pathfinder

Палітра Pathfinder (Обробка контурів), яка викликається на екран однойменною командою меню Window (Вікно), призначена для комбінування об'єктів різними способами, часто з утворенням складених контурів (compound path). Палітра Pathfinder (Обробка контурів) є дуже потужним засобом векторного формоутворення, що надзвичайно цінно для створення досить складних форм. У зв'язку з цим можна тимчасово не звертати уваги на параметри заливок і обведення.

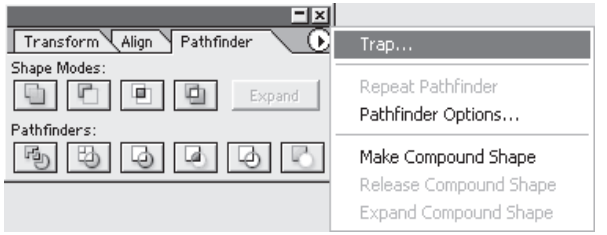


Рис. 8.26. Палітра Pathfinder з відкритим меню

При комбінуванні контурів з різними кольоровими заливками, як правило, результуючому об'єкта привласнюється заливка верхнього об'єкта. Для комбінування об'єктів слід розмістити і виділити необхідні об'єкти, а потім натискувати одну з кнопок палітри Pathfinder (Обробка контурів).

Залежно від виконуваної операції на екран може бути виведене діалогове вікно, в якому слід ввести необхідні додаткові дані.

Біля команд палітри Pathfinder (Обробка контурів) існує декілька спільних установок, які змінюються в діалоговому вікні PathfinderOptions (Обробка контурів: Параметри), яке викликається командою PathfinderOptions (Параметри обробки контурів) меню палітри.

Робота з масками

Механізм маскування – надзвичайно продуктивний спосіб створення складних графічних композицій. Ми вже розглядали роботу цього механізму у редакторі Photoshop, глава 7. Його ефективність

полягає в тому, що можна компонувати об’єкти без кадрування (обрізання) їх вручну. Досить помістити їх у маскуючий об’єкт необхідної форми – обтравочну маску (clipping mask), і компоновані об’єкти матимуть запланований вигляд (рис. 8.27). Маскованими об’єктами можуть бути як прості контури, так і складні (compound paths). При активній роботі з масками слід мати на увазі, що достатньо складні маски можуть викликати проблеми під час роздрукування документів. Просте вирішення проблеми, як і для складних (складених) контурів, – це спрощення.

Для того щоб з’ясувати кількість масок у документі, необхідно виконати команду Document Info (Інформація про документ) меню Window (Вікно), яка виводить на екран діалогове вікно.

У розділі Objects (Об’єкти) діалогового вікна приводиться інформація про наявні об’єкти, у тому числі про кількість масок.

Для того щоб дізнатися, чи є даний об’єкт маскою, необхідно його виділити і виконати команди Selection Only (Тільки виділені об’єкти) і Objects (Об’єкти) меню палітри.

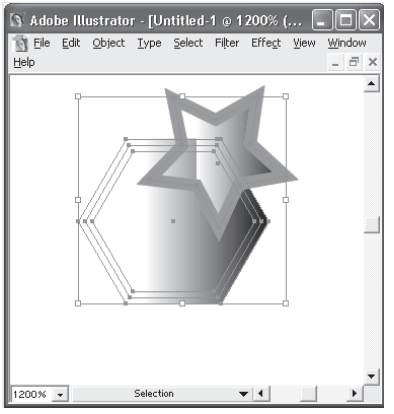


Рис. 8.27. Початкові об’єкти і отриманий маскований об’єкт

Крім того, для виділення всіх масок у документі існує спеціальна команда Clipping Masks (Обтравочні маски) меню Select | Object (Виділення | Об’єкт). Для створення маски необхідно виділити відповідні об’єкти, причому маскуючий об’єкт (форма якого маскує решту всіх об’єктів) повинен займати саме верхнє положення.В тому випадку, якщо масковані об’єкти розташовуються на різних шарах, цей факт слід мати на увазі при від-ключенні шарів або при переміщенні об’єктів на шарах.

полягає в тому, що можна компонувати об’єкти без кадрування (обрізання) їх вручну. Досить помістити їх у маскуючий об’єкт необхідної форми – обтравочну маску (clipping mask), і компоновані об’єкти матимуть запланований вигляд (рис. 8.27). Маскованими об’єктами можуть бути як прості контури, так і складні (compound paths). При активній роботі з масками слід мати на увазі, що достатньо складні маски можуть викликати проблеми під час роздрукування документів. Просте вирішення проблеми, як і для складних (складених) контурів, – це спрощення.

Для того щоб з’ясувати кількість масок у документі, необхідно виконати команду Document Info (Інформація про документ) меню Window (Вікно), яка виводить на екран діалогове вікно.

У розділі Objects (Об’єкти) діалогового вікна приводиться інформація про наявні об’єкти, у тому числі про кількість масок.

Для того щоб дізнатися, чи є даний об’єкт маскою, необхідно його виділити і виконати команди Selection Only (Тільки виділені об’єкти) і Objects (Об’єкти) меню палітри.

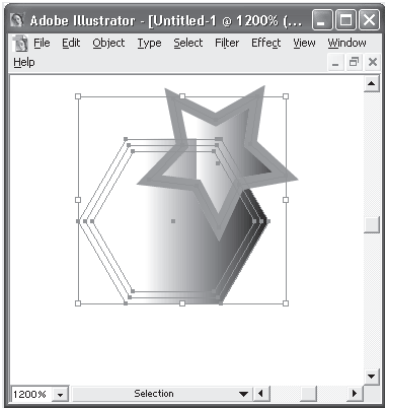


Рис. 8.27. Початкові об’єкти і отриманий маскований об’єкт

Крім того, для виділення всіх масок у документі існує спеціальна команда Clipping Masks (Обтравочні маски) меню Select | Object (Виділення | Об’єкт). Для створення маски необхідно виділити відповідні об’єкти, причому маскуючий об’єкт (форма якого маскує решту всіх об’єктів) повинен займати саме верхнє положення.В тому випадку, якщо масковані об’єкти розташовуються на різних шарах, цей факт слід мати на увазі при від-ключенні шарів або при переміщенні об’єктів на шарах.

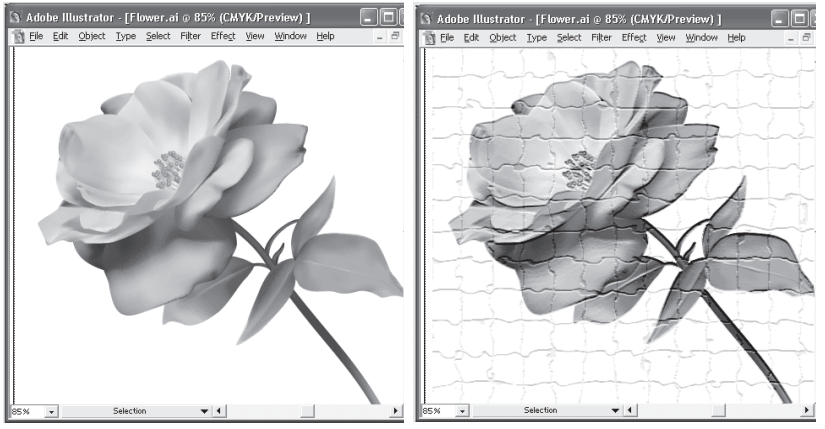
Потім виконується команда Make (Утворити) меню Object | Clipping Mask (Об'єкт | Обтравочна маска), яка видаляє параметри контура і заливки маскуючого об'єкта (він "ховається", виконавши свою місію).

Для "витягання" об'єктів "з-під маски" потрібно виділити маскуючий об'єкт і виконати команду Release (Відмінити) меню Object | Clipping Mask (Об'єкт | Обтравочна маска). При цьому слід не забувати, що біля маскуючого об'єкта зберігся контур, який за відсутності параметрів продовжує залишатися невидимим. Створену обтравочну маску можна у будь-який момент відредагувати.

Перетворення піксельних зображень у векторні

Фільтр Object Mosaic (Векторна мозаїка) меню Filter | Create (Фільтр | Створити) призначений для перетворення піксельного зображення на векторну мозаїку (рис. 8.28) – задане число забарвлених прямокутників, у своїй сукупності тих, що представляють початкове зображення.

Даний фільтр "працює" зі всіма типами піксельних зображень, у тому числі й із створеними за допомогою команди Rasterize (Растеризувати).



а б

Рис. 8.28. Початкове піксельне зображення (а) і результат роботи фільтра Object Mosaic (б)

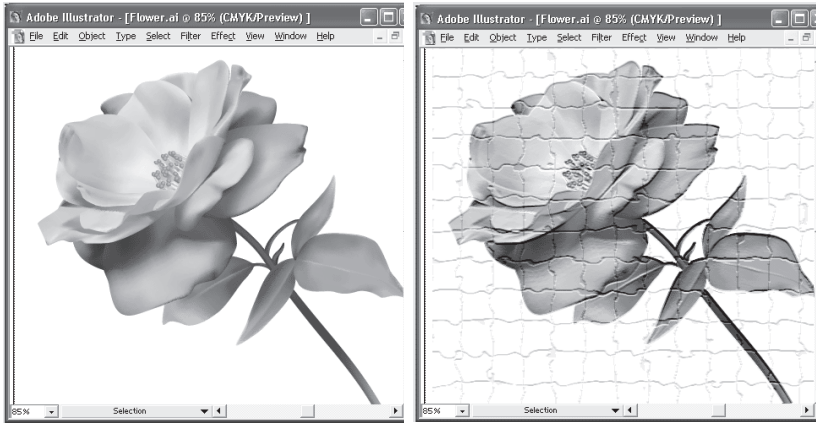
Потім виконується команда Make (Утворити) меню Object | Clipping Mask (Об'єкт | Обтравочна маска), яка видаляє параметри контура і заливки маскуючого об'єкта (він "ховається", виконавши свою місію).

Для "витягання" об'єктів "з-під маски" потрібно виділити маскуючий об'єкт і виконати команду Release (Відмінити) меню Object | Clipping Mask (Об'єкт | Обтравочна маска). При цьому слід не забувати, що біля маскуючого об'єкта зберігся контур, який за відсутності параметрів продовжує залишатися невидимим. Створену обтравочну маску можна у будь-який момент відредагувати.

Перетворення піксельних зображень у векторні

Фільтр Object Mosaic (Векторна мозаїка) меню Filter | Create (Фільтр | Створити) призначений для перетворення піксельного зображення на векторну мозаїку (рис. 8.28) – задане число забарвлених прямокутників, у своїй сукупності тих, що представляють початкове зображення.

Даний фільтр "працює" зі всіма типами піксельних зображень, у тому числі й із створеними за допомогою команди Rasterize (Растеризувати).



а б

Рис. 8.28. Початкове піксельне зображення (а) і результат роботи фільтра Object Mosaic (б)

Діалогове вікно Object Mosaic (Векторна мозаїка), яке викликається однойменною командою меню Filter | Create (Фільтр | Створити), дозволяє визначити такі параметри зображення:

– група NewSize (Новий розмір) призначена для зміни розмірності зображення, початкові значення якого відображаються в сусідній групі Current Size (Поточний розмір);

– група Tile Spacing (Інтервал між елементами) дозволяє встановити відстань між елементами мозаїки. Інтервал створюється за рахунок зменшення елементу мозаїки, тому діапазон його значень обмежений;

– група Number of Tiles (Кількість елементів) служить для установки кількості елементів по горизонталі і по вертикалі.

8.4. Робота з шарами

Шари є подальшим розвитком методу аплікації. Вони вже теж розглядались у графічному редакторі. У кожному документі, що створюється в програмі Adobe Illustrator, вже спочатку є один шар. Додавання нових шарів дозволяє легко управляти значною масою об'єктів (дублювати, перерозподіляти, зливати і виконувати інші дії), їх відображенням на екрані і виводом на зовнішні друкуючі пристрої. Крім того, в програмі існують додаткові властивості шарів: розміщення на шарах не лише об'єктів і груп об'єктів, але й інших шарів, що дозволяє створювати складні вкладені комплекси.

У програмі передбачений також спеціальний шаблонний шар (template) для розміщення будь-яких об'єктів, що мають допоміжне значення, зокрема це стосується піксельних зображень, призначених для трасування. З метою подальшої інтеграції продуктів фірми Adobe у програмі передбачено збереження шарів при експортуванні документа у формат програми Adobe Photoshop.

Основні особливості шарів:

- у межах одного шару об'єкти розташовуються звичайним способом — "стопкою": кожен об'єкт має свій рівень;
- згруповані об'єкти розташовуються на одному і тому ж шарі;
- при маскуванні об'єктів, що знаходяться на різних шарах, об'єкти проміжних шарів також стають часткою маски.

Діалогове вікно Object Mosaic (Векторна мозаїка), яке викликається однойменною командою меню Filter | Create (Фільтр | Створити), дозволяє визначити такі параметри зображення:

– група NewSize (Новий розмір) призначена для зміни розмірності зображення, початкові значення якого відображаються в сусідній групі Current Size (Поточний розмір);

– група Tile Spacing (Інтервал між елементами) дозволяє встановити відстань між елементами мозаїки. Інтервал створюється за рахунок зменшення елементу мозаїки, тому діапазон його значень обмежений;

– група Number of Tiles (Кількість елементів) служить для установки кількості елементів по горизонталі і по вертикалі.

8.4. Робота з шарами

Шари є подальшим розвитком методу аплікації. Вони вже теж розглядались у графічному редакторі. У кожному документі, що створюється в програмі Adobe Illustrator, вже спочатку є один шар. Додавання нових шарів дозволяє легко управляти значною масою об'єктів (дублювати, перерозподіляти, зливати і виконувати інші дії), їх відображенням на екрані і виводом на зовнішні друкуючі пристрої. Крім того, в програмі існують додаткові властивості шарів: розміщення на шарах не лише об'єктів і груп об'єктів, але й інших шарів, що дозволяє створювати складні вкладені комплекси.

У програмі передбачений також спеціальний шаблонний шар (template) для розміщення будь-яких об'єктів, що мають допоміжне значення, зокрема це стосується піксельних зображень, призначених для трасування. З метою подальшої інтеграції продуктів фірми Adobe у програмі передбачено збереження шарів при експортуванні документа у формат програми Adobe Photoshop.

Основні особливості шарів:

- у межах одного шару об'єкти розташовуються звичайним способом — "стопкою": кожен об'єкт має свій рівень;
- згруповані об'єкти розташовуються на одному і тому ж шарі;
- при маскуванні об'єктів, що знаходяться на різних шарах, об'єкти проміжних шарів також стають часткою маски.

Палітра Layers

Всі операції з шарами (створення, видалення, фіксація і т. п.) здійснюються за допомогою палітри Layers (Шари) (рис. 8.29), яка відкривається командою Layers (Шари) меню Window (Вікно).

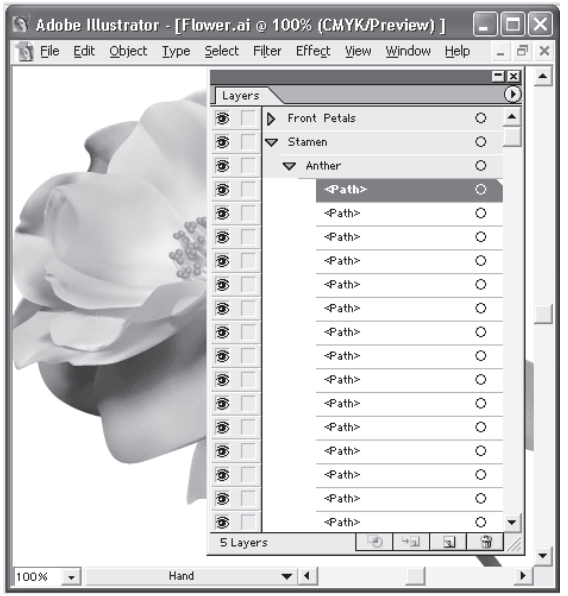


Рис. 8.29. Палітра Layers

Крім того, палітра Layers (Шари) використовується для виділення і видалення об’єктів і їх груп, фіксації і захована їх, а також привласнення параметрів.

Розробники програми передбачають такі умови роботи з шарами, групами і об’єктами:

- можна створювати довільне число шарів, груп і об’єктів;
- на шарах можуть розташовуватися інші шари (вкладені шари), групи і окремі об’єкти. Групи у свою чергу можуть включати інші групи і об’єкти, але не шари;
- якщо до шарів і груп на шарі застосовуються які-небудь команди, наприклад Lock (Закріпити) меню Object (Об’єкт), слід враховувати ієрархію шарів, груп і об’єктів;

Палітра Layers

Всі операції з шарами (створення, видалення, фіксація і т. п.) здійснюються за допомогою палітри Layers (Шари) (рис. 8.29), яка відкривається командою Layers (Шари) меню Window (Вікно).

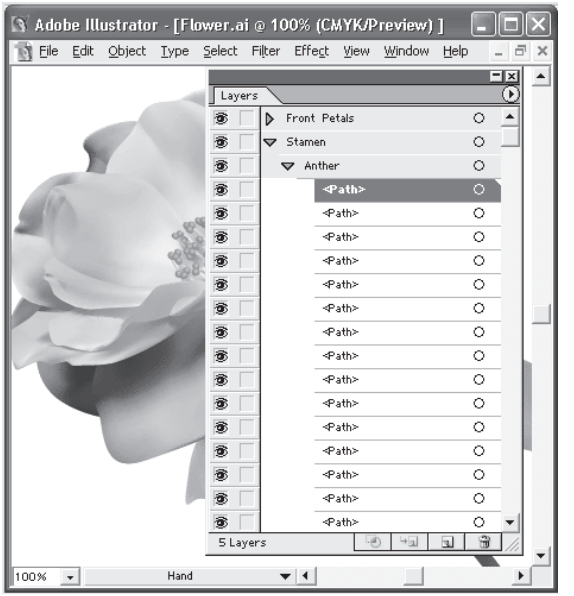


Рис. 8.29. Палітра Layers

Крім того, палітра Layers (Шари) використовується для виділення і видалення об’єктів і їх груп, фіксації і захована їх, а також привласнення параметрів.

Розробники програми передбачають такі умови роботи з шарами, групами і об’єктами:

- можна створювати довільне число шарів, груп і об’єктів;
- на шарах можуть розташовуватися інші шари (вкладені шари), групи і окремі об’єкти. Групи у свою чергу можуть включати інші групи і об’єкти, але не шари;
- якщо до шарів і груп на шарі застосовуються які-небудь команди, наприклад Lock (Закріпити) меню Object (Об’єкт), слід враховувати ієрархію шарів, груп і об’єктів;

- переміщення шару спричиняє за собою переміщення всіх інших шарів, груп і об'єктів, розташованих на ньому;
- об'єкти, що входять до групи, розташовуються на одному шарі. Якщо в групу об'єднуються об'єкти, розташовані на різних шарах, то вони "підтягуються" на той шар, який включає самий верхній об'єкт групи.

У палітрі відображаються імена шарів, починаючи з самого верхнього. Всі операції (наприклад, малювання, переміщення, вставка об'єктів з буфера обміну Clipboard) здійснюються тільки на активному шарі.

Фіксація шарів, груп і об'єктів

У програмі AdobeIllustrator можна фіксувати не лише окремі об'єкти (для чого, зокрема, призначена команда Lock (Закріпити) меню Object (Об'єкт)), але також цілі групи і шари. Фіксований шар є "непорушним цілим": об'єкти на такому шарі не виділяються, не змінюються, не переміщаються і не видаляються.

Для того щоб зафіксувати шар або групу, досить клацнути на кнопці, розташованій лівіше за ім'я шару або групи. Символом фіксованого шару є "замок" (рис. 8.30) (). Крім того, курсор малюючих інструментів приймає вид "перекресленого олівця" ().

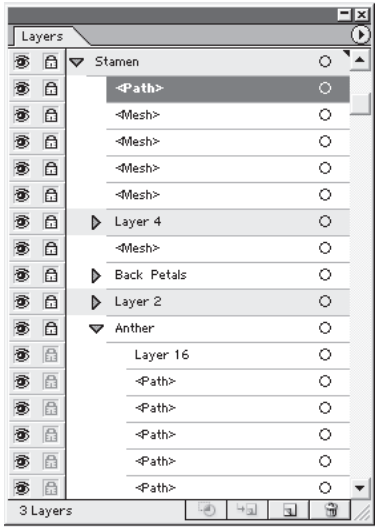


Рис. 8.30. Піктограма "замок" символізує фіксований шар

Повторне клацання на кнопці з "замком" знімає фіксацію. Виконати цю ж операцію можна і за допомогою діалогового вікна Layer Options (Параметри шару), встановивши або видаливши прапорець Lock (Закріпити). Якщо зафіксовано декілька шарів, то для їх одночасного "звільнення" необхідно виконати команду Unlock All (Звільнити всі). Якщо потрібно зафіксувати всі шари, окрім

- переміщення шару спричиняє за собою переміщення всіх інших шарів, груп і об'єктів, розташованих на ньому;
- об'єкти, що входять до групи, розташовуються на одному шарі. Якщо в групу об'єднуються об'єкти, розташовані на різних шарах, то вони "підтягуються" на той шар, який включає самий верхній об'єкт групи.

У палітрі відображаються імена шарів, починаючи з самого верхнього. Всі операції (наприклад, малювання, переміщення, вставка об'єктів з буфера обміну Clipboard) здійснюються тільки на активному шарі.

Фіксація шарів, груп і об'єктів

У програмі AdobeIllustrator можна фіксувати не лише окремі об'єкти (для чого, зокрема, призначена команда Lock (Закріпити) меню Object (Об'єкт)), але також цілі групи і шари. Фіксований шар є "непорушним цілим": об'єкти на такому шарі не виділяються, не змінюються, не переміщаються і не видаляються.

Для того щоб зафіксувати шар або групу, досить клацнути на кнопці, розташованій лівіше за ім'я шару або групи. Символом фіксованого шару є "замок" (рис. 8.30) (). Крім того, курсор малюючих інструментів приймає вид "перекресленого олівця" ().

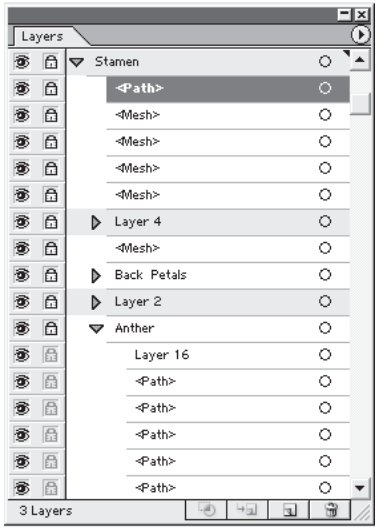


Рис. 8.30. Піктограма "замок" символізує фіксований шар

Повторне клацання на кнопці з "замком" знімає фіксацію. Виконати цю ж операцію можна і за допомогою діалогового вікна Layer Options (Параметри шару), встановивши або видаливши прапорець Lock (Закріпити). Якщо зафіксовано декілька шарів, то для їх одночасного "звільнення" необхідно виконати команду Unlock All (Звільнити всі). Якщо потрібно зафіксувати всі шари, окрім

одного, досить клацнути в рядку на кнопці фіксації даного шару при натиснутій клавіші <Alt>.

Імпортування і експортування шарів

Найближчим "соратником" векторної програми Adobe Illustrator є редактор піксельної графіки Adobe Photoshop, в якому також використовується механізм шарів. У зв'язку з цим необхідно зрозуміти їх взаємодію. При імпортуванні файла з програми Adobe Photoshop, в якому декілька шарів, відбувається зведення шарів, і піксельне зображення розташовується на одному шарі.

У свою чергу, при експортуванні документа з програми Adobe Illustrator в піксельне зображення всі шари можуть бути зведені в один (при цьому "заховані" і шаблонні шари не експортуються).

Якщо ж необхідно зберегти шари, то в діалоговому вікні Photoshop Options (Параметри документа Photoshop) встановлюються прапорці Write Layers (Зберегти шари), Write Nested Layers (Зберегти вкладені шари) і Hidden Layers (Приховані шари). Документ зберігається у форматі програми Adobe Photoshop (PSD): кожен шар векторного зображення растеризується в шар піксельного зображення, а також додається фоновий шар.

8.5. Робота з текстом і шрифтом

Однією з найсильніших сторін програми Adobe Illustrator є робота з текстом, форматування тексту і використання шрифту. І це недивно, оскільки саме фірма Adobe стоїть біля витоків розробки цифрових шрифтів. Створений нею формат Adobe Type 1 став стандартом (і символом) друку високої якості.

У програмі можна виконувати всі загальноприйняті операції з шрифтом (змінювати гарнітуру, зображення, кегль, кернінг і трекінг) і дещо понад це (направляти по довільному контуру, здійснювати набір по вертикалі). Необмежені можливості представляються по зовнішньому оформленню шрифту за допомогою декоративних контурів, декоративних заливок, розтяжок і так далі.

Введення тексту

Інструменти групи Typi (Текст) служать для набору горизонтального або вертикального тексту в будь-якому місці документа. Природно, можна використовувати текст, набраний в інших програмах, імпортуючи і конвертуючи його з різних форматів.

одного, досить клацнути в рядку на кнопці фіксації даного шару при натиснутій клавіші <Alt>.

Імпортування і експортування шарів

Найближчим "соратником" векторної програми Adobe Illustrator є редактор піксельної графіки Adobe Photoshop, в якому також використовується механізм шарів. У зв'язку з цим необхідно зрозуміти їх взаємодію. При імпортуванні файла з програми Adobe Photoshop, в якому декілька шарів, відбувається зведення шарів, і піксельне зображення розташовується на одному шарі.

У свою чергу, при експортуванні документа з програми Adobe Illustrator в піксельне зображення всі шари можуть бути зведені в один (при цьому "заховані" і шаблонні шари не експортуються).

Якщо ж необхідно зберегти шари, то в діалоговому вікні Photoshop Options (Параметри документа Photoshop) встановлюються прапорці Write Layers (Зберегти шари), Write Nested Layers (Зберегти вкладені шари) і Hidden Layers (Приховані шари). Документ зберігається у форматі програми Adobe Photoshop (PSD): кожен шар векторного зображення растеризується в шар піксельного зображення, а також додається фоновий шар.

8.5. Робота з текстом і шрифтом

Однією з найсильніших сторін програми Adobe Illustrator є робота з текстом, форматування тексту і використання шрифту. І це недивно, оскільки саме фірма Adobe стоїть біля витоків розробки цифрових шрифтів. Створений нею формат Adobe Type 1 став стандартом (і символом) друку високої якості.

У програмі можна виконувати всі загальноприйняті операції з шрифтом (змінювати гарнітуру, зображення, кегль, кернінг і трекінг) і дещо понад це (направляти по довільному контуру, здійснювати набір по вертикалі). Необмежені можливості представляються по зовнішньому оформленню шрифту за допомогою декоративних контурів, декоративних заливок, розтяжок і так далі.

Введення тексту

Інструменти групи Typi (Текст) служать для набору горизонтального або вертикального тексту в будь-якому місці документа. Природно, можна використовувати текст, набраний в інших програмах, імпортуючи і конвертуючи його з різних форматів.

За допомогою інструментів Typi (Текст) або Vertical Type (Вертикальний текст) можна в будь-якому місці вводити довільний текст, який не обмежується ні рамкою, ні колонкою.

Ці інструменти є ідеальним засобом для набору заголовків, крупних і коротких написів, тому такий спосіб введення можна назвати заголовочним, а текст, що вводиться, заголовочним текстом.

Параметри шрифту

Програма Adobe Illustrator пропонує управління всіма мислимими параметрами шрифту: вибір гарнітури і зображення, визначення кегля і інтерліньяжа, кернінгу і трекінгу, можливість зрушення по вертикалі і обертання. Причому параметри можна визначати як до набору тексту, так і змінювати параметри вже набраного форматованого раніше тексту.

Управління параметрами шрифту здійснюється за допомогою двох палітр — Character (Символ) і Paragraph (Абзац). Окрім цього, деякі параметри мають власні меню або палітри.

Можливість вирішити одне і те ж завдання різними способами може полегшити роботу, а може і заплутати. Зокрема, варто було б грунтовно затвердити список "гарячих" клавіш (вони насправді прискорюють роботу і дуже зручні), якби... вони не мінялися від версії до версії і не відрізнялися від інших програм.

8.6. Робота з піксельними зображеннями

Програма Adobe Illustrator — яскравий представник програм векторної графіки, вона втілює практично всі досягнення в цій області. Однією з переваг векторної графіки є можливість інтеграції у векторний документ піксельних зображень, проте, як правило, їх обробка раніше зводилася тільки до переміщення і масштабування.

Фірма Adobe, проводячи послідовну політику з'єднання всіх своїх продуктів у єдиний і взаємозв'язаний технологічний ланцюжок, істотно розширила можливості перетворення піксельних зображень і включила в програму Adobe Illustrator більше половини своїх фільтрів, використовуваних у програмі піксельної графіки Adobe Photoshop. Це дозволяє, не покидаючи векторної програми, отримувати всілякі візуальні ефекти.

За допомогою інструментів Typi (Текст) або Vertical Type (Вертикальний текст) можна в будь-якому місці вводити довільний текст, який не обмежується ні рамкою, ні колонкою.

Ці інструменти є ідеальним засобом для набору заголовків, крупних і коротких написів, тому такий спосіб введення можна назвати заголовочним, а текст, що вводиться, заголовочним текстом.

Параметри шрифту

Програма Adobe Illustrator пропонує управління всіма мислимими параметрами шрифту: вибір гарнітури і зображення, визначення кегля і інтерліньяжа, кернінгу і трекінгу, можливість зрушення по вертикалі і обертання. Причому параметри можна визначати як до набору тексту, так і змінювати параметри вже набраного форматованого раніше тексту.

Управління параметрами шрифту здійснюється за допомогою двох палітр — Character (Символ) і Paragraph (Абзац). Окрім цього, деякі параметри мають власні меню або палітри.

Можливість вирішити одне і те ж завдання різними способами може полегшити роботу, а може і заплутати. Зокрема, варто було б грунтовно затвердити список "гарячих" клавіш (вони насправді прискорюють роботу і дуже зручні), якби... вони не мінялися від версії до версії і не відрізнялися від інших програм.

8.6. Робота з піксельними зображеннями

Програма Adobe Illustrator — яскравий представник програм векторної графіки, вона втілює практично всі досягнення в цій області. Однією з переваг векторної графіки є можливість інтеграції у векторний документ піксельних зображень, проте, як правило, їх обробка раніше зводилася тільки до переміщення і масштабування.

Фірма Adobe, проводячи послідовну політику з'єднання всіх своїх продуктів у єдиний і взаємозв'язаний технологічний ланцюжок, істотно розширила можливості перетворення піксельних зображень і включила в програму Adobe Illustrator більше половини своїх фільтрів, використовуваних у програмі піксельної графіки Adobe Photoshop. Це дозволяє, не покидаючи векторної програми, отримувати всілякі візуальні ефекти.

Окрім своїх "рідних" фільтрів, включених у дистрибутив – програми, програма Adobe Illustrator може працювати і з фільтрами сторонніх розробників, так званими додатковими модулями (plug-ins). Одного дня інстальовані, вони викликаються з меню Filter (Фільтр) і працюють відповідно до завдання.

Для роботи з піксельними зображеннями надається все більше можливостей. Їх можна вільно імпортувати в документ, використовуючи команду Place (Помістити), управляти ними за допомогою палітри Links (Зв'язки), трасувати інструментом AutoTrace (Автотрасування), використовувати безліч піксельних фільтрів, перенесених з програми Adobe Photoshop, а крім того, піксельне зображення може бути конвертоване у векторну мозаїку за допомогою оригінального фільтра Object Mosaic (Векторна мозаїка) або у векторну гравюру – фільтр Photo Crosshatch (Векторна фотогравюра).

Спільні відомості про фільтри

Всі фільтри піксельної графіки подані в меню Filter (Фільтр) і розділені на 10 груп: Artistic (Імітація), Blur (Розмиття), BrushStrokes(Штрихи), Distort (Деформація), Pixelate (Оформлення), Sharpen (Різкість), Sketch (Ескіз), Stylize (Стилізація), Texture (Текстура) і Video (Відео).

Виділивши піксельне зображення, слід виконати відповідну команду (застосувати фільтр). Як правило, на екран виводиться відповідне діалогове вікно, в якому необхідно визначити параметри фільтра.

Якщо потрібно повторити дану команду ще раз, наприклад до іншого зображення, слід пам'ятати, що ім'я фільтра, що застосовувався останнім, відображається у верхніх рядках списку команд меню Filter (Фільтр). Вибір першої команди, що починається із слова Apply (Застосувати), приводить до виконання команди з попередніми установками, а виконання другої команди виведе на екран діалогове вікно відповідного фільтра.

Для попередньої оцінки дії фільтра на зображення слід скористатися наявністю вікна Preview (Перегляд) у переважної більшості фільтрів. При розміщенні курсору у вікні попереднього перегляду він набуває вигляду інструменту Hand (Рука). Натискуючи кнопку

Окрім своїх "рідних" фільтрів, включених у дистрибутив – програми, програма Adobe Illustrator може працювати і з фільтрами сторонніх розробників, так званими додатковими модулями (plug-ins). Одного дня інстальовані, вони викликаються з меню Filter (Фільтр) і працюють відповідно до завдання.

Для роботи з піксельними зображеннями надається все більше можливостей. Їх можна вільно імпортувати в документ, використовуючи команду Place (Помістити), управляти ними за допомогою палітри Links (Зв'язки), трасувати інструментом AutoTrace (Автотрасування), використовувати безліч піксельних фільтрів, перенесених з програми Adobe Photoshop, а крім того, піксельне зображення може бути конвертоване у векторну мозаїку за допомогою оригінального фільтра Object Mosaic (Векторна мозаїка) або у векторну гравюру – фільтр Photo Crosshatch (Векторна фотогравюра).

Спільні відомості про фільтри

Всі фільтри піксельної графіки подані в меню Filter (Фільтр) і розділені на 10 груп: Artistic (Імітація), Blur (Розмиття), BrushStrokes(Штрихи), Distort (Деформація), Pixelate (Оформлення), Sharpen (Різкість), Sketch (Ескіз), Stylize (Стилізація), Texture (Текстура) і Video (Відео).

Виділивши піксельне зображення, слід виконати відповідну команду (застосувати фільтр). Як правило, на екран виводиться відповідне діалогове вікно, в якому необхідно визначити параметри фільтра.

Якщо потрібно повторити дану команду ще раз, наприклад до іншого зображення, слід пам'ятати, що ім'я фільтра, що застосовувався останнім, відображається у верхніх рядках списку команд меню Filter (Фільтр). Вибір першої команди, що починається із слова Apply (Застосувати), приводить до виконання команди з попередніми установками, а виконання другої команди виведе на екран діалогове вікно відповідного фільтра.

Для попередньої оцінки дії фільтра на зображення слід скористатися наявністю вікна Preview (Перегляд) у переважної більшості фільтрів. При розміщенні курсору у вікні попереднього перегляду він набуває вигляду інструменту Hand (Рука). Натискуючи кнопку

миші, можна перемістити найбільш важливий фрагмент зображення, щоб саме на ньому побачити результат дії фільтра і підібрати необхідні параметри дії фільтра для адекватного втілення художнього задуму. У момент переміщення зображення набуває початкового вигляду, для забезпечення цього процесу дія фільтру тимчасово припиняється.

У полі попереднього перегляду передбачена можливість масштабування зображення. Для цієї мети під вікном розташовано дві кнопки:

- для збільшення фрагмента слід використовувати кнопку із знаком "плюс";
- для зменшення фрагмента — кнопку із знаком "мінус".

Між кнопками відображається величина зменшення або збільшення зображення у відсотках. Миготлива лінія підкреслення під значенням масштабу свідчить про те, що "перемальовування" вмісту поля перегляду ще не закінчене.

Після установки всіх необхідних параметрів слід натискувати кнопку ОК . Якщо виконання фільтра передбачає складну обробку і зажадає достатньо довгого часу, програма Adobe Illustrator виводить на екран вікно Progress (Процес виконання) з назвою виконуваного фільтра і "смугою прогресу", яка дозволить приблизно оцінити час, необхідний для роботи фільтра. У вікні також представлена кнопка Stop (Перервати), що забезпечує можливість перервати процес.

8.7. Інформаційна графіка (діаграми)

Програма Adobe Illustrator має в своєму розпорядженні спеціальний інструмент, який дозволить при розумному використанні створити лаконічні, елегантні і грамотні діаграми.

Типи діаграм

Вибір типу діаграми залежить у значній мірі від виду цифрової інформації і завдання, яке на нього покладається: порівняння окремих даних, динаміка одного або декількох процесів і так далі.

Програма Adobe Illustrator пропонує на вибір дев’ять типів діаграм.

- Тип Column (Вертикальні смуги) — вертикальна смугова діаграма, завдання якої – відображати для порівняння два або

миші, можна перемістити найбільш важливий фрагмент зображення, щоб саме на ньому побачити результат дії фільтра і підібрати необхідні параметри дії фільтра для адекватного втілення художнього задуму. У момент переміщення зображення набуває початкового вигляду, для забезпечення цього процесу дія фільтру тимчасово припиняється.

У полі попереднього перегляду передбачена можливість масштабування зображення. Для цієї мети під вікном розташовано дві кнопки:

- для збільшення фрагмента слід використовувати кнопку із знаком "плюс";
- для зменшення фрагмента — кнопку із знаком "мінус".

Між кнопками відображається величина зменшення або збільшення зображення у відсотках. Миготлива лінія підкреслення під значенням масштабу свідчить про те, що "перемальовування" вмісту поля перегляду ще не закінчене.

Після установки всіх необхідних параметрів слід натискувати кнопку ОК . Якщо виконання фільтра передбачає складну обробку і зажадає достатньо довгого часу, програма Adobe Illustrator виводить на екран вікно Progress (Процес виконання) з назвою виконуваного фільтра і "смугою прогресу", яка дозволить приблизно оцінити час, необхідний для роботи фільтра. У вікні також представлена кнопка Stop (Перервати), що забезпечує можливість перервати процес.

8.7. Інформаційна графіка (діаграми)

Програма Adobe Illustrator має в своєму розпорядженні спеціальний інструмент, який дозволить при розумному використанні створити лаконічні, елегантні і грамотні діаграми.

Типи діаграм

Вибір типу діаграми залежить у значній мірі від виду цифрової інформації і завдання, яке на нього покладається: порівняння окремих даних, динаміка одного або декількох процесів і так далі.

Програма Adobe Illustrator пропонує на вибір дев’ять типів діаграм.

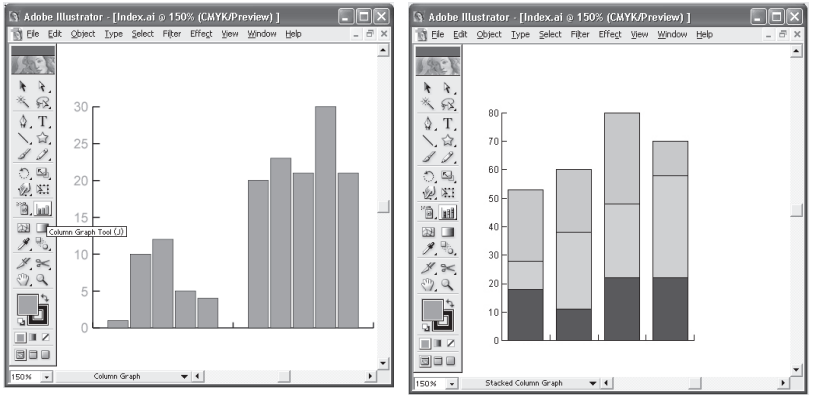
- Тип Column (Вертикальні смуги) — вертикальна смугова діаграма, завдання якої – відображати для порівняння два або

більші за значенням параметри за допомогою смуг, довжина яких пропорційна його значенню (рис. 8.31, а).

- Тип Stacked Column (Вертикальний стек) завдання – відображати для порівняння не лише два або більше значення за допомогою смуг, але і показати співвідношення цих смуг до цілого (рис. 8.31, б).
- Тип Bar (Горизонтальні смуги) – той же тип діаграми, що і Column (Вертикальні смуги), але розташовується горизонтально. Цей тип можна використовувати для порівняння декількох значень, пов'язаних з тимчасовими параметрами, наприклад, для порівняння швидкодії процесорів комп'ютерів.
- Тип Stacked Bar (Горизонтальний стек) – той же тип діаграми, що і Stacked Column (Вертикальний стек), але розташовується горизонтально. Цей тип можна використовувати для порівняння декількох значень, пов'язаних з тимчасовими параметрами, наприклад порівняння швидкодії комп'ютерів з урахуванням вкладу процесора, звернення до дисків і так далі.
- Тип Line (Лінійний графік) – лінійна діаграма, завдання якої – відображати динаміку зміни дискретних значень у певні інтервали часу (рис. 8.32, а). Цей тип діаграми часто використовується, наприклад, для зображення динаміки курсу валют.
- Тип Area (Адитивний графік) – площинна діаграма, заснована на лінійній, з тією різницею, що майдан під кожною лінією також має смислове значення.
- Тип Scatter (Точкова діаграма) – діаграма розсіяння, яка може застосовуватися для відображення даних, що відхиляються від якого-небудь значення (рис. 8.32, б).
- Тип Pie (Кругова діаграма) – кругова діаграма, вживана для відображення процентного вмісту складових частин по відношенню до цілого (рис. 8.33, а).
- Тип Radar (Радар) – "павутинова" діаграма, яка може застосовуватися для відображення порівняльних значень у часі або по категоріях (рис. 8.33, б). Такий тип діаграм зручно використовувати під час дослідження системи за великою кількістю різних параметрів.

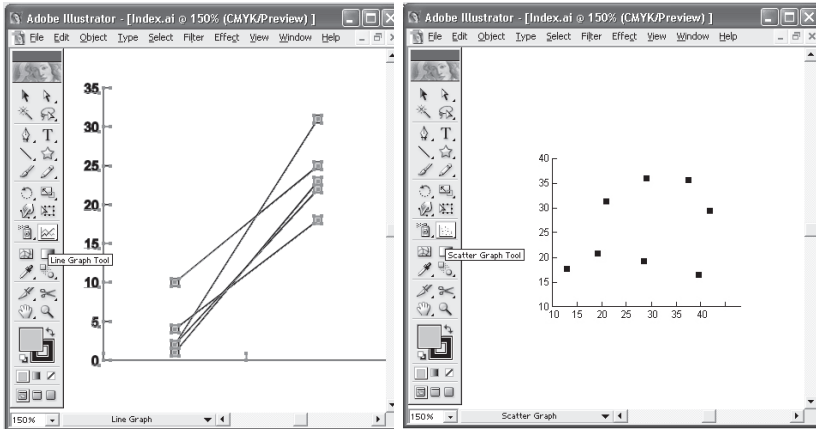
більші за значенням параметри за допомогою смуг, довжина яких пропорційна його значенню (рис. 8.31, а).

- Тип Stacked Column (Вертикальний стек) завдання – відображати для порівняння не лише два або більше значення за допомогою смуг, але і показати співвідношення цих смуг до цілого (рис. 8.31, б).
- Тип Bar (Горизонтальні смуги) – той же тип діаграми, що і Column (Вертикальні смуги), але розташовується горизонтально. Цей тип можна використовувати для порівняння декількох значень, пов'язаних з тимчасовими параметрами, наприклад, для порівняння швидкодії процесорів комп'ютерів.
- Тип Stacked Bar (Горизонтальний стек) – той же тип діаграми, що і Stacked Column (Вертикальний стек), але розташовується горизонтально. Цей тип можна використовувати для порівняння декількох значень, пов'язаних з тимчасовими параметрами, наприклад порівняння швидкодії комп'ютерів з урахуванням вкладу процесора, звернення до дисків і так далі.
- Тип Line (Лінійний графік) – лінійна діаграма, завдання якої – відображати динаміку зміни дискретних значень у певні інтервали часу (рис. 8.32, а). Цей тип діаграми часто використовується, наприклад, для зображення динаміки курсу валют.
- Тип Area (Адитивний графік) – площинна діаграма, заснована на лінійній, з тією різницею, що майдан під кожною лінією також має смислове значення.
- Тип Scatter (Точкова діаграма) – діаграма розсіяння, яка може застосовуватися для відображення даних, що відхиляються від якого-небудь значення (рис. 8.32, б).
- Тип Pie (Кругова діаграма) – кругова діаграма, вживана для відображення процентного вмісту складових частин по відношенню до цілого (рис. 8.33, а).
- Тип Radar (Радар) – "павутинова" діаграма, яка може застосовуватися для відображення порівняльних значень у часі або по категоріях (рис. 8.33, б). Такий тип діаграм зручно використовувати під час дослідження системи за великою кількістю різних параметрів.



а б

Рис. 8.31. Приклади діаграм Column (а) та Stacked Column (б)

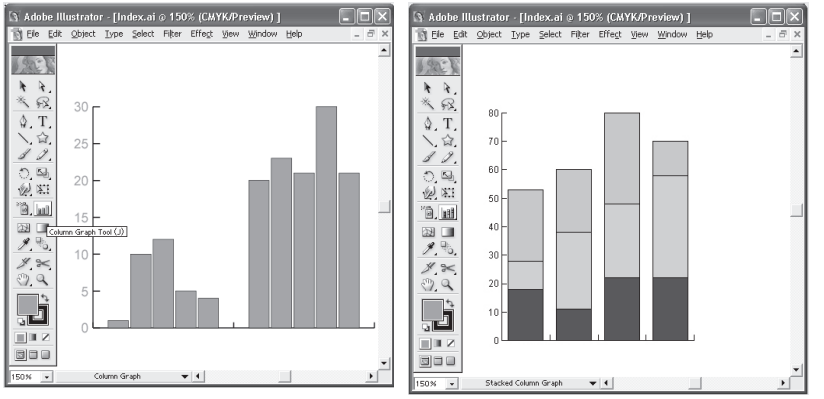


а б

Рис. 8.32. Приклади діаграм Line (а) та Scatter (б)

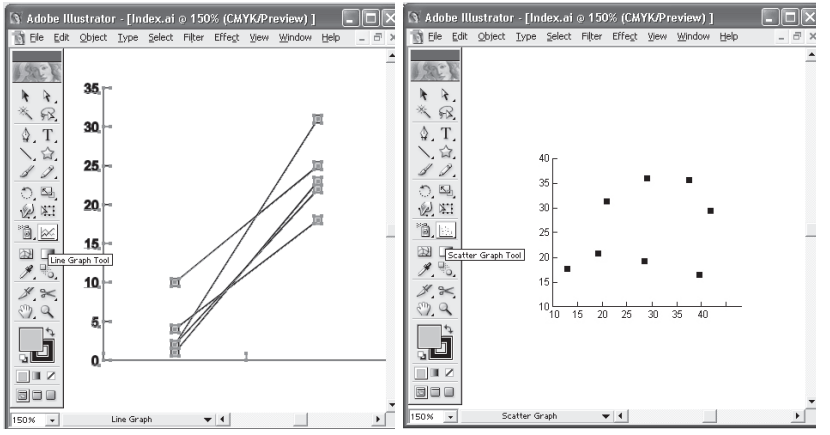
Робота з діаграмами

Під час роботи з діаграмами слід мати на увазі декілька спільних положень, єдиних для всіх типів діаграм. Після створення діаграми вона є сукупністю згрупованих об’єктів, які можна трактувати як звичайні векторні об’єкти, тобто змінювати всі доступні параметри,



а б

Рис. 8.31. Приклади діаграм Column (а) та Stacked Column (б)



а б

Рис. 8.32. Приклади діаграм Line (а) та Scatter (б)

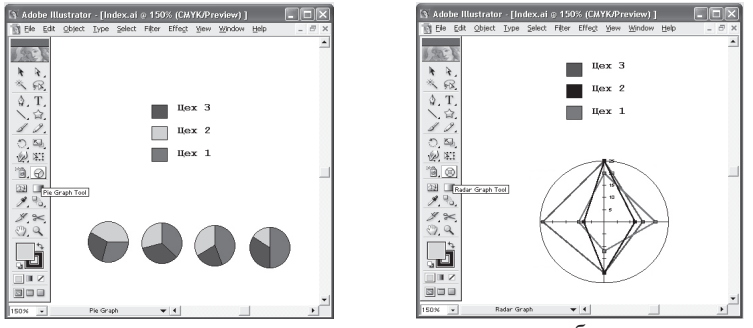
Робота з діаграмами

Під час роботи з діаграмами слід мати на увазі декілька спільних положень, єдиних для всіх типів діаграм. Після створення діаграми вона є сукупністю згрупованих об’єктів, які можна трактувати як звичайні векторні об’єкти, тобто змінювати всі доступні параметри,

а також переміщати і трансформувати їх всіма доступними засобами. У разі введення нових початкових цифрових даних діаграма автоматично перебудовується, хоча, на жаль, вона виключає всі зміни, пов'язані із зовнішнім оформленням (колір, розмір шрифту тощо), і повертає діаграму до вигляду за замовчуванням.

Для створення діаграм необхідно включити один з інструментів групи Graph (Діаграма) ( ). Подальші етапи можуть бути такими.

- Необхідно позначити за допомогою штрихового прямокутника майдан, який повинна займати діаграма.
- Відразу ж на екран виводиться діалогове вікно Graph Data (Дані діаграми), яке служить для введення цифрових даних діаграми. Необхідно заповнити таблицю і закрити її.
- Програма автоматично сформує діаграму, біля якої тип і всі параметри встановлюються за замовчуванням. Тип і вид діаграми можна надалі змінити (рис. 8.33).



а б

Рис. 8.33. Приклади діаграм Pie (а) та Radar (б)

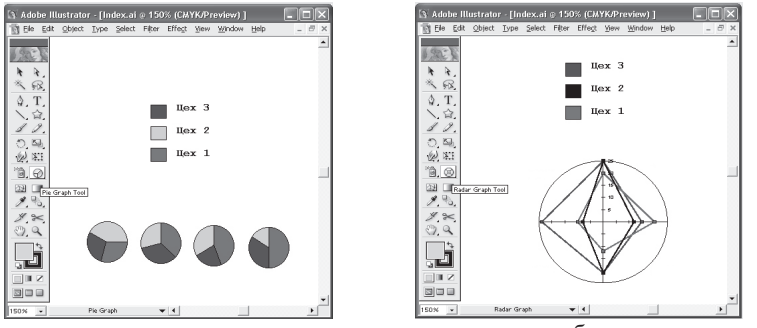
Таблиця даних, яка служить цифровою базою для формування діаграми, є типовим прикладом електронних таблиць. Діалогове вікно Graph Data (Дані діаграми) можна викликати і за допомогою команди Data (Дані) меню Object | Graphs (Об'єкт | Діаграма).

Дані, що використовуються як початкові для створення діаграми, можуть бути підготовлені заздалегідь або іншими виконавцями і збережені в текстовому файлі. Такі дані може надати і замовник, не-

а також переміщати і трансформувати їх всіма доступними засобами. У разі введення нових початкових цифрових даних діаграма автоматично перебудовується, хоча, на жаль, вона виключає всі зміни, пов'язані із зовнішнім оформленням (колір, розмір шрифту тощо), і повертає діаграму до вигляду за замовчуванням.

Для створення діаграм необхідно включити один з інструментів групи Graph (Діаграма) ( ). Подальші етапи можуть бути такими.

- Необхідно позначити за допомогою штрихового прямокутника майдан, який повинна займати діаграма.
- Відразу ж на екран виводиться діалогове вікно Graph Data (Дані діаграми), яке служить для введення цифрових даних діаграми. Необхідно заповнити таблицю і закрити її.
- Програма автоматично сформує діаграму, біля якої тип і всі параметри встановлюються за замовчуванням. Тип і вид діаграми можна надалі змінити (рис. 8.33).



а б

Рис. 8.33. Приклади діаграм Pie (а) та Radar (б)

Таблиця даних, яка служить цифровою базою для формування діаграми, є типовим прикладом електронних таблиць. Діалогове вікно Graph Data (Дані діаграми) можна викликати і за допомогою команди Data (Дані) меню Object | Graphs (Об'єкт | Діаграма).

Дані, що використовуються як початкові для створення діаграми, можуть бути підготовлені заздалегідь або іншими виконавцями і збережені в текстовому файлі. Такі дані може надати і замовник, не-

обхідно тільки при цьому зобов'язати його дотриматись деяких формальних умов. Зокрема, дані кожної комірки повинні бути розділені табулятором (клавіша<Tab>), а кожен рядок повинен закінчуватися переведенням каретки (клавіша<Enter>).

Для імпорту даних необхідно в діалоговому вікні Graph Data (Дані діаграми) клацнути по кнопці Import Data (Імпорт даних ) і в діалоговому вікні, що відкрилося, звернутися до потрібного файла у відповідній папці. Якщо дані подані в якій-небудь програмі електронних таблиць (spread sheet application), наприклад в Microsoft Excel, то вони можуть бути перенесені в діалогове вікноGraph Data (Дані діаграми) за допомогою буфера обміну Clipboard.

На жаль, обсяг навчального посібника не дозволяє більш детально розглянути можливості даного програмного продукту та принципи роботи з ним. Все це можна знайти в спеціальній літературі [1, 7, 12, 21].

Контрольні питання

1. Призначення редактора векторної графіки Adobe Illustrator.
2. Дати характеристику основних інструментів редактора векторної графіки.
3. Навести основні переваги та недоліки векторної графіки у порівнянні з растровою.
4. Який математичний апарат найчастіше за все застосовується у векторних редакторах під час створення графічних зображень?
5. Дати характеристику основних можливостей Adobe Illustrator під час роботи з ілюстративною (інформаційною) графікою.
6. Які можливості надає Adobe Illustrator під час роботи з піксельними (растровими) зображеннями?
7. Дати характеристику колірних моделей, що застосовуються у Adobe Illustrator.

обхідно тільки при цьому зобов'язати його дотриматись деяких формальних умов. Зокрема, дані кожної комірки повинні бути розділені табулятором (клавіша<Tab>), а кожен рядок повинен закінчуватися переведенням каретки (клавіша<Enter>).

Для імпорту даних необхідно в діалоговому вікні Graph Data (Дані діаграми) клацнути по кнопці Import Data (Імпорт даних ) і в діалоговому вікні, що відкрилося, звернутися до потрібного файла у відповідній папці. Якщо дані подані в якій-небудь програмі електронних таблиць (spread sheet application), наприклад в Microsoft Excel, то вони можуть бути перенесені в діалогове вікноGraph Data (Дані діаграми) за допомогою буфера обміну Clipboard.

На жаль, обсяг навчального посібника не дозволяє більш детально розглянути можливості даного програмного продукту та принципи роботи з ним. Все це можна знайти в спеціальній літературі [1, 7, 12, 21].

Контрольні питання

1. Призначення редактора векторної графіки Adobe Illustrator.
2. Дати характеристику основних інструментів редактора векторної графіки.
3. Навести основні переваги та недоліки векторної графіки у порівнянні з растровою.
4. Який математичний апарат найчастіше за все застосовується у векторних редакторах під час створення графічних зображень?
5. Дати характеристику основних можливостей Adobe Illustrator під час роботи з ілюстративною (інформаційною) графікою.
6. Які можливості надає Adobe Illustrator під час роботи з піксельними (растровими) зображеннями?
7. Дати характеристику колірних моделей, що застосовуються у Adobe Illustrator.

Список скорочень

АРМ – автоматизоване робоче місце
ГІС – геоінформаційні системи
ДКП – дискретне косинусно-синусне перетворення
ЕПТ – електронно-променева трубка
КГ – комп’ютерна графіка
КЗВІ – колективні засоби відображення інформації
ПЕОМ – персональна електронно-обчислювальна машина
РКД – рідкокристалічний дисплей
САПР – системи автоматизованого проектування
ЦАП – цифро-аналоговий перетворювач
ШСЗ – штучний супутник Землі

Список скорочень

АРМ – автоматизоване робоче місце
ГІС – геоінформаційні системи
ДКП – дискретне косинусно-синусне перетворення
ЕПТ – електронно-променева трубка
КГ – комп’ютерна графіка
КЗВІ – колективні засоби відображення інформації
ПЕОМ – персональна електронно-обчислювальна машина
РКД – рідкокристалічний дисплей
САПР – системи автоматизованого проектування
ЦАП – цифро-аналоговий перетворювач
ШСЗ – штучний супутник Землі

Глосарій

Animation – штучне уявлення руху в кіно, на телебаченні або в комп'ютерній графіці шляхом відображення послідовності малюнків або кадрів з частотою, при якій забезпечується цілісне зорове сприйняття образів.

API – інтерфейс для розробки прикладних програм.

BMP – Microsoft Windows BitMaP - растровий формат. Розроблений фірмою Microsoft для роботи з растровими зображеннями в середовищах Microsoft Windows і OS / 2 фірми IBM.

Bit – Біт – двійковий знак, «0» або «1», що використовується в обчислювальній техніці для внутрімашинного подання інформації. У комп'ютерній графіці служить як одиниця глибини кольору.

Bitmap – бітова карта – таблиця цифрових значень, що кодують колір кожного піксела зображення. Зазвичай служить позначенням чорно-білого штрихового зображення.

Bezier – крива Без'є – поліноміальна крива, що задається набором визначальних точок. Являє собою рівняння на один ступінь менше кількості врахованих точок. Криві Без'є записуються в пам'яті комп'ютера у вигляді математичних формул, тому малюнки, отримані за допомогою цих кривих, забезпечують можливість масштабування без втрати якості зображення.

CMYK – кольорова модель, колірний простір, оснований на чотирьох кольорах поліграфічного процессу: блакитному, пурпурному, жовтому і чорному. Одна з двох основних систем представлення кольору. Заснована на субтрактивній моделі змішання блакитного, пурпурного, жовтого і чорного кольорів (останній доданий для відтворення при друкуванні більш глибокого чорного кольору у зв'язку з недосконалістю барвників).

CDR – CorelDRaw – Векторний формат - робочий формат графічного пакета CorelDRAW фірми Corel.

Глосарій

Animation – штучне уявлення руху в кіно, на телебаченні або в комп'ютерній графіці шляхом відображення послідовності малюнків або кадрів з частотою, при якій забезпечується цілісне зорове сприйняття образів.

API – інтерфейс для розробки прикладних програм.

BMP – Microsoft Windows BitMaP - растровий формат. Розроблений фірмою Microsoft для роботи з растровими зображеннями в середовищах Microsoft Windows і OS / 2 фірми IBM.

Bit – Біт – двійковий знак, «0» або «1», що використовується в обчислювальній техніці для внутрімашинного подання інформації. У комп'ютерній графіці служить як одиниця глибини кольору.

Bitmap – бітова карта – таблиця цифрових значень, що кодують колір кожного піксела зображення. Зазвичай служить позначенням чорно-білого штрихового зображення.

Bezier – крива Без'є – поліноміальна крива, що задається набором визначальних точок. Являє собою рівняння на один ступінь менше кількості врахованих точок. Криві Без'є записуються в пам'яті комп'ютера у вигляді математичних формул, тому малюнки, отримані за допомогою цих кривих, забезпечують можливість масштабування без втрати якості зображення.

CMYK – кольорова модель, колірний простір, оснований на чотирьох кольорах поліграфічного процессу: блакитному, пурпурному, жовтому і чорному. Одна з двох основних систем представлення кольору. Заснована на субтрактивній моделі змішання блакитного, пурпурного, жовтого і чорного кольорів (останній доданий для відтворення при друкуванні більш глибокого чорного кольору у зв'язку з недосконалістю барвників).

CDR – CorelDRaw – Векторний формат - робочий формат графічного пакета CorelDRAW фірми Corel.

339

DirectX – набір API-функцій, що надають доступ до ресурсів персонального комп'ютера для створення високопродуктивних додатків, що працюють у реальному часі. Відкрита архітектура PC дозволяє використовувати графічні та звукові плати від різних виробників.

FPS (Frames Per Second) – основна одиниця виміру продуктивності 3D- прискорювачів. Показує кількість кадрів, які прискорювач здатний відмалювати за секунду. Чим вищий цей показник, тим потужніше відеокарта.

HSB – колірна модель. HSB – Кольоровий простір, оснований на трьох характеристиках кольору: колірному тоні (Hue), насиченості (Saturation), і яскравості (Brightness)

Interactive – інтерактивний. Додаток, результат роботи якого залежить від користувача і він може змінювати як результат, так і подання результату.

OpenGL – програмний інтерфейс 3D (3D API) для WindowsNT і Windows 95, історично створений для станцій IrisGL фірми Silicon Graphics. Зараз саме OpenGL часто застосовується замість інтерфейсів серії DirectX від Microsof.

Plug-in – додатковий модуль, програмне забезпечення (найчастіше фільтри), розроблене сторонніми компаніями для використання з програмою AdobePhotoshop і деякими іншими.

RGB – колірна модель RGB, колірний простір, оснований на трьох кольорах:червоному, зеленому та синьому. RGB – це колірний простір моніторів, сканерів, відеокамер. RGB називається адитивним колірним простором, оскільки компоненти додаються до чорного кольору.

Rendering – візуалізація, у тривимірній графіці – операції відтворення, вимагають обчислень для кожного піксела зображення. Процес створення растрового цифрового зображення об'єктів по комп'ютерній моделі.

Z-buffering – механізм видалення невидимих поверхонь, що використовує буфер пам'яті, в якому зберігається значення глибини зображення (тобто координата по осі z) для кожного піксела зображення. Z-буферизація вимагає виділення значних обсягів пам'яті.

Альфа-канал – спеціальний шар зображення (як шари Red, Green, і Blue), що визначає «прозорість» RGB-точки вихідного зображення.

Векторна графіка – метод графічного подання об'єкта у вигляді відрізків прямих (векторів). У поліграфії векторна графіка зазвичай використовується для підготовки макетів.

DirectX – набір API-функцій, що надають доступ до ресурсів персонального комп'ютера для створення високопродуктивних додатків, що працюють у реальному часі. Відкрита архітектура PC дозволяє використовувати графічні та звукові плати від різних виробників.

FPS (Frames Per Second) – основна одиниця виміру продуктивності 3D- прискорювачів. Показує кількість кадрів, які прискорювач здатний відмалювати за секунду. Чим вищий цей показник, тим потужніше відеокарта.

HSB – колірна модель. HSB – Кольоровий простір, оснований на трьох характеристиках кольору: колірному тоні (Hue), насиченості (Saturation), і яскравості (Brightness)

Interactive – інтерактивний. Додаток, результат роботи якого залежить від користувача і він може змінювати як результат, так і подання результату.

OpenGL – програмний інтерфейс 3D (3D API) для WindowsNT і Windows 95, історично створений для станцій IrisGL фірми Silicon Graphics. Зараз саме OpenGL часто застосовується замість інтерфейсів серії DirectX від Microsof.

Plug-in – додатковий модуль, програмне забезпечення (найчастіше фільтри), розроблене сторонніми компаніями для використання з програмою AdobePhotoshop і деякими іншими.

RGB – колірна модель RGB, колірний простір, оснований на трьох кольорах:червоному, зеленому та синьому. RGB – це колірний простір моніторів, сканерів, відеокамер. RGB називається адитивним колірним простором, оскільки компоненти додаються до чорного кольору.

Rendering – візуалізація, у тривимірній графіці – операції відтворення, вимагають обчислень для кожного піксела зображення. Процес створення растрового цифрового зображення об'єктів по комп'ютерній моделі.

Z-buffering – механізм видалення невидимих поверхонь, що використовує буфер пам'яті, в якому зберігається значення глибини зображення (тобто координата по осі z) для кожного піксела зображення. Z-буферизація вимагає виділення значних обсягів пам'яті.

Альфа-канал – спеціальний шар зображення (як шари Red, Green, і Blue), що визначає «прозорість» RGB-точки вихідного зображення.

Векторна графіка – метод графічного подання об'єкта у вигляді відрізків прямих (векторів). У поліграфії векторна графіка зазвичай використовується для підготовки макетів.

Графічний примітив – найпростіший геометричний об'єкт, який відображається на екрані дисплея або на робочому полі графопобудовника: точка, відрізок прямої, дуга кола або еліпса, прямокутник і т.п.

Ділова графіка – технологія створення зображень із супроводжуючим текстом для потреб комерції.

Комп'ютерна графіка – технологія створення та обробки графічних зображень засобами обчислювальної техніки. Комп'ютерна графіка вивчає методи отримання зображень на підставі невізуальних даних або даних, створених користувачем.

Мультимедіа – сукупність комп'ютерних технологій, одночасно використовують кілька інформаційних середовищ: графіку, текст, відео, фотографію, анімацію, звукові ефекти, високоякісний звуковий супровід. Технологію мультимедіа складають спеціальні апаратні і програмні засоби.

Обробка зображень – область комп'ютерної графіки, що досліджує завдання, в яких і вхідні і вихідні дані є зображеннями.

Растрова графіка – метод графічного подання об'єкта у вигляді безлічі точок.

Тривимірна графіка – технологія мультимедіа; комп’ютерна графіка, створювана за допомогою зображень, що мають довжину, ширину і глибину.

Фрактал – об'єкт, що має розгалужену структуру. Частини фрактала подібні всьому об'єкту. Фрактали використовуються в комп'ютерній графіці для створення ліній дерев, хмар та інших графічних об'єктів.

Графічний примітив – найпростіший геометричний об'єкт, який відображається на екрані дисплея або на робочому полі графопобудовника: точка, відрізок прямої, дуга кола або еліпса, прямокутник і т.п.

Ділова графіка – технологія створення зображень із супроводжуючим текстом для потреб комерції.

Комп'ютерна графіка – технологія створення та обробки графічних зображень засобами обчислювальної техніки. Комп'ютерна графіка вивчає методи отримання зображень на підставі невізуальних даних або даних, створених користувачем.

Мультимедіа – сукупність комп'ютерних технологій, одночасно використовують кілька інформаційних середовищ: графіку, текст, відео, фотографію, анімацію, звукові ефекти, високоякісний звуковий супровід. Технологію мультимедіа складають спеціальні апаратні і програмні засоби.

Обробка зображень – область комп'ютерної графіки, що досліджує завдання, в яких і вхідні і вихідні дані є зображеннями.

Растрова графіка – метод графічного подання об'єкта у вигляді безлічі точок.

Тривимірна графіка – технологія мультимедіа; комп’ютерна графіка, створювана за допомогою зображень, що мають довжину, ширину і глибину.

Фрактал – об'єкт, що має розгалужену структуру. Частини фрактала подібні всьому об'єкту. Фрактали використовуються в комп'ютерній графіці для створення ліній дерев, хмар та інших графічних об'єктів.

Список літератури

1. *Adobe Illustrator:* Учебник. – К.: Диасофт, 2002. – 368 с.
2. *Баяковский Ю.М., Игнатенко А.В., Фролов А.И.* Графическая библиотека OpenGL: методическое пособие. – М.: Издательство Триумф, 2002. – 117 с.
3. *Блінова Т.О., Порєв В.М.* Комп'ютерна графіка /за ред. В.М. Порєва – К.: Видавництво «Юніор», 2004. – 456 с.
4. *Борн Г.* Форматы данных. – СПб.: BHV, 1995. – 472 с.
5. *Воротніков В.В., Канкін І.О.* Компютерна графіка: навчальний посібник. – Житомир: ЖВІ НАУ, 2009. – 88 с.
6. *Веселовська Г.В., Ходаков В.Є., Веселовський В.М.* Комп'ютерна графіка: навч. посібник для студентів вищих навчальних закладів / за ред. В.Є. Ходакова. – Херсон: ОЛДІ-плюс, 2008. – 584 с.
7. *Горобець С.М.* Основи комп'ютерної графіки: навч. посібник /за ред. М.В. Левківського. – К.: Центр навчальної літератури, 2006. – 232с.
8. *Компьютерная графика:* учебник (+CD) /М.Н. Петров, В.П. Молочков – СПб.:Питер, 2002. – 736 с.
9. *Лигун А.О.* Комп'ютерна графіка (Обробка та стиск зображень): навч. посібник/А.О. Лигун, О.О. Шумейко. – Д.: Біла К.О., 2010. – 114с.
10. *Мак-Клелланд, Дик, Фуллер, Лори, Ульрих.* Photoshop CS2. Библия пользователя: пер. с англ. – М.: ООО «И.Д. Вильямс», 2008. – 944 с.
11. *Павлидис Т.* Алгоритмы машинной графики и обработки изображений/Т.Павлидис. – М: Радио и связь, 1986. – 400 с.
12. *Пореев В.Н.* Компьютерная графика. – СПб.: БХВ-Петербург, 2002. – 432 с.
13. *Роджерс Д., Адамс Дж.* Математические основы машинной графики. – М.: Машиностроение, 1980. – 240 с.
14. *Уэлстид С.* Фракталы и вейвлеты для сжатия изображений в действии: Учебное пособие – М.: Издательство Триумф, 2003. – 320 с.

Список літератури

1. *Adobe Illustrator:* Учебник. – К.: Диасофт, 2002. – 368 с.
2. *Баяковский Ю.М., Игнатенко А.В., Фролов А.И.* Графическая библиотека OpenGL: методическое пособие. – М.: Издательство Триумф, 2002. – 117 с.
3. *Блінова Т.О., Порєв В.М.* Комп'ютерна графіка /за ред. В.М. Порєва – К.: Видавництво «Юніор», 2004. – 456 с.
4. *Борн Г.* Форматы данных. – СПб.: BHV, 1995. – 472 с.
5. *Воротніков В.В., Канкін І.О.* Компютерна графіка: навчальний посібник. – Житомир: ЖВІ НАУ, 2009. – 88 с.
6. *Веселовська Г.В., Ходаков В.Є., Веселовський В.М.* Комп'ютерна графіка: навч. посібник для студентів вищих навчальних закладів / за ред. В.Є. Ходакова. – Херсон: ОЛДІ-плюс, 2008. – 584 с.
7. *Горобець С.М.* Основи комп'ютерної графіки: навч. посібник /за ред. М.В. Левківського. – К.: Центр навчальної літератури, 2006. – 232с.
8. *Компьютерная графика:* учебник (+CD) /М.Н. Петров, В.П. Молочков – СПб.:Питер, 2002. – 736 с.
9. *Лигун А.О.* Комп'ютерна графіка (Обробка та стиск зображень): навч. посібник/А.О. Лигун, О.О. Шумейко. – Д.: Біла К.О., 2010. – 114с.
10. *Мак-Клелланд, Дик, Фуллер, Лори, Ульрих.* Photoshop CS2. Библия пользователя: пер. с англ. – М.: ООО «И.Д. Вильямс», 2008. – 944 с.
11. *Павлидис Т.* Алгоритмы машинной графики и обработки изображений/Т.Павлидис. – М: Радио и связь, 1986. – 400 с.
12. *Пореев В.Н.* Компьютерная графика. – СПб.: БХВ-Петербург, 2002. – 432 с.
13. *Роджерс Д., Адамс Дж.* Математические основы машинной графики. – М.: Машиностроение, 1980. – 240 с.
14. *Уэлстид С.* Фракталы и вейвлеты для сжатия изображений в действии: Учебное пособие – М.: Издательство Триумф, 2003. – 320 с.

15. *Тихомиров Ю.В.* OpenGL. Программирование трехмерной графики. – 2-е изд. – СПб.: БХВ-Петербург, 2002. – 304 с.
16. *Фень Юань.* Программирование графики для Windows/Юань Фень. – СПб.: Изд. Питер, 2002. – 1070 с.
17. *Федер Е.* Фракталы. – М.: Мир, 1991. – 254 с.
18. *Эгрон Ж.* Синтез изображений. Базовые алгоритмы. – М.: Радио и связь, 1993. – 232 с.
19. http://dgec.mirea.org
20. http://intuit.ru
21. http://ru.wikipedia.org
22. http://compression.ru
23. http://www.compdoc.ru
24. http://megagraphix.org
25. http://wladm.narod.ru

15. *Тихомиров Ю.В.* OpenGL. Программирование трехмерной графики. – 2-е изд. – СПб.: БХВ-Петербург, 2002. – 304 с.
16. *Фень Юань.* Программирование графики для Windows/Юань Фень. – СПб.: Изд. Питер, 2002. – 1070 с.
17. *Федер Е.* Фракталы. – М.: Мир, 1991. – 254 с.
18. *Эгрон Ж.* Синтез изображений. Базовые алгоритмы. – М.: Радио и связь, 1993. – 232 с.
19. http://dgec.mirea.org
20. http://intuit.ru
21. http://ru.wikipedia.org
22. http://compression.ru
23. http://www.compdoc.ru
24. http://megagraphix.org
25. http://wladm.narod.ru

343

Предметний покажчик

А
Антіелайзинг, 17
Анімація комп'ютерна, 19
Аттрактор, 188
Алгоритм
Брезенхема, 119
Коена-Сазерленда, 121, 124

Б
Барнслі, 187
Без'є, 135
Бітмап, 160
Браузер, 152

В
Відображення, 82
Векторні перетворення, 314-320

Г
Геометрія
афінна, 109
евклідова, 109
нарисна, 109
Графіка
двовимірна, 77
ділова, 18
зображувальна, 9,10
ілюстративна, 19
комп'ютерна графіка, 9,10,11
растрова, 19-24
векторна, 24-29
фрактальна, 29-30
наукова, 18
рекламна, 19
тривимірна, 93-100
художня, 19

Д
Дигітайзер, 54-55
Драйвер, 53
Джойстик, 58

З
Зображення
точок, 75
ліній, 76
кривих, 133

К
Кодування фрактальне, 187
Колірні моделі
адитивна, 33-37
перцепційна, 42-43
субтрактивна, 37-41
Колірний тон, 43
Контент, 172
Крива Без'є, 136

М
Масштабування, 83
Матриця Без'є, 136
Метафайл, 158

Предметний покажчик

А
Антіелайзинг, 17
Анімація комп'ютерна, 19
Аттрактор, 188
Алгоритм
Брезенхема, 119
Коена-Сазерленда, 121, 124

Б
Барнслі, 187
Без'є, 135
Бітмап, 160
Браузер, 152

В
Відображення, 82
Векторні перетворення, 314-320

Г
Геометрія
афінна, 109
евклідова, 109
нарисна, 109
Графіка
двовимірна, 77
ділова, 18
зображувальна, 9,10
ілюстративна, 19
комп'ютерна графіка, 9,10,11
растрова, 19-24
векторна, 24-29
фрактальна, 29-30
наукова, 18
рекламна, 19
тривимірна, 93-100
художня, 19

Д
Дигітайзер, 54-55
Драйвер, 53
Джойстик, 58

З
Зображення
точок, 75
ліній, 76
кривих, 133

К
Кодування фрактальне, 187
Колірні моделі
адитивна, 33-37
перцепційна, 42-43
субтрактивна, 37-41
Колірний тон, 43
Контент, 172
Крива Без'є, 136

М
Масштабування, 83
Матриця Без'є, 136
Метафайл, 158

О
Однорідні координати, 85-89
OpenGL, 214-270

П
Плоттер, 18
Піксель, 20, 21, 34, 117
Плагін, 292
Проекція
аксонометрична, 110
діаметрична,110
ізометрична,110
косокутна,110
паралельна, 110
перспективна, 110

Р
Растр, 20-25
Рендерінг, 273
Роздільна здатність, 23

С
Сканер, 51-54
барабанний, 53
кольоровий, 52
лазерний, 55
листовий, 53
магнітний, 54
планшетний, 54
ручний, 52
чорно-білий, 51-52
Силізація, 276

Т
Тачпад, 59
Трекбол, 58
Трекпойнт, 59

Ф
Формати файлів
AVI, 168
BMP, 140
CAM, 140
CLP, 140
CPT, 140
CUR, 140
DCM, 140
DCX, 140
DjVu, 158
IMG, 140
IFF, 140
FIF, 140
GIF, 148,
JPEG, 140,182-187
KDC, 140
MPEG, 170-175
PCD, 140
PCX, 140
PDL, 140,145
PSD, 140,164
PIC, 140
PGM, 140
RAW, 140
RAS, 140
RBS, 140
SUN, 140
STING, 140
TIFF, 140,157
TGA, 140
VRML, 140,144,
WMF, 141,162.
Фрактал, 30, 187

О
Однорідні координати, 85-89
OpenGL, 214-270

П
Плоттер, 18
Піксель, 20, 21, 34, 117
Плагін, 292
Проекція
аксонометрична, 110
діаметрична,110
ізометрична,110
косокутна,110
паралельна, 110
перспективна, 110

Р
Растр, 20-25
Рендерінг, 273
Роздільна здатність, 23

С
Сканер, 51-54
барабанний, 53
кольоровий, 52
лазерний, 55
листовий, 53
магнітний, 54
планшетний, 54
ручний, 52
чорно-білий, 51-52
Силізація, 276

Т
Тачпад, 59
Трекбол, 58
Трекпойнт, 59

Ф
Формати файлів
AVI, 168
BMP, 140
CAM, 140
CLP, 140
CPT, 140
CUR, 140
DCM, 140
DCX, 140
DjVu, 158
IMG, 140
IFF, 140
FIF, 140
GIF, 148,
JPEG, 140,182-187
KDC, 140
MPEG, 170-175
PCD, 140
PCX, 140
PDL, 140,145
PSD, 140,164
PIC, 140
PGM, 140
RAW, 140
RAS, 140
RBS, 140
SUN, 140
STING, 140
TIFF, 140,157
TGA, 140
VRML, 140,144,
WMF, 141,162.
Фрактал, 30, 187

345

НАВЧАЛЬНЕ ВИДАННЯ

Михайло Федорович ПІЧУГІН
Іван Олегович КАНКІН
Володимир Володимирович ВОРОТНІКОВ

КОМП'ЮТЕРНА ГРАФІКА

НАВЧАЛЬНИЙ ПОСІБНИК

Редактор **Л.А. Климчук**
Технічний редактор **О.В. Белікова**

Підписано до друку 04.01.2013 р. Формат 60x84 1/16.
Друк лазерний. Папір офсетний. Гарнітура Times New Roman.
Ум. друк. арк. 19,46.Наклад 300 прим.

ТОВ «Видавництво «Центр учбової літератури»
вул. Електриків, 23 м. Київ 04176
тел./факс 044-425-01-34
тел.: 044-425-20-63; 425-04-47; 451-65-95
800-501-68-00 (безкоштовно в межах України)

Свідоцтво про внесення суб'єкта видавничої справи до державного реєстру видавців,
виготівників і розповсюджувачів видавничої продукції
ДК № 4162 від 21.09.2011 р.

НАВЧАЛЬНЕ ВИДАННЯ

Михайло Федорович ПІЧУГІН
Іван Олегович КАНКІН
Володимир Володимирович ВОРОТНІКОВ

КОМП'ЮТЕРНА ГРАФІКА

НАВЧАЛЬНИЙ ПОСІБНИК

Редактор **Л.А. Климчук**
Технічний редактор **О.В. Белікова**

Підписано до друку 04.01.2013 р. Формат 60x84 1/16.
Друк лазерний. Папір офсетний. Гарнітура Times New Roman.
Ум. друк. арк. 19,46.Наклад 300 прим.

ТОВ «Видавництво «Центр учбової літератури»
вул. Електриків, 23 м. Київ 04176
тел./факс 044-425-01-34
тел.: 044-425-20-63; 425-04-47; 451-65-95
800-501-68-00 (безкоштовно в межах України)

Свідоцтво про внесення суб'єкта видавничої справи до державного реєстру видавців,
виготівників і розповсюджувачів видавничої продукції
ДК № 4162 від 21.09.2011 р.