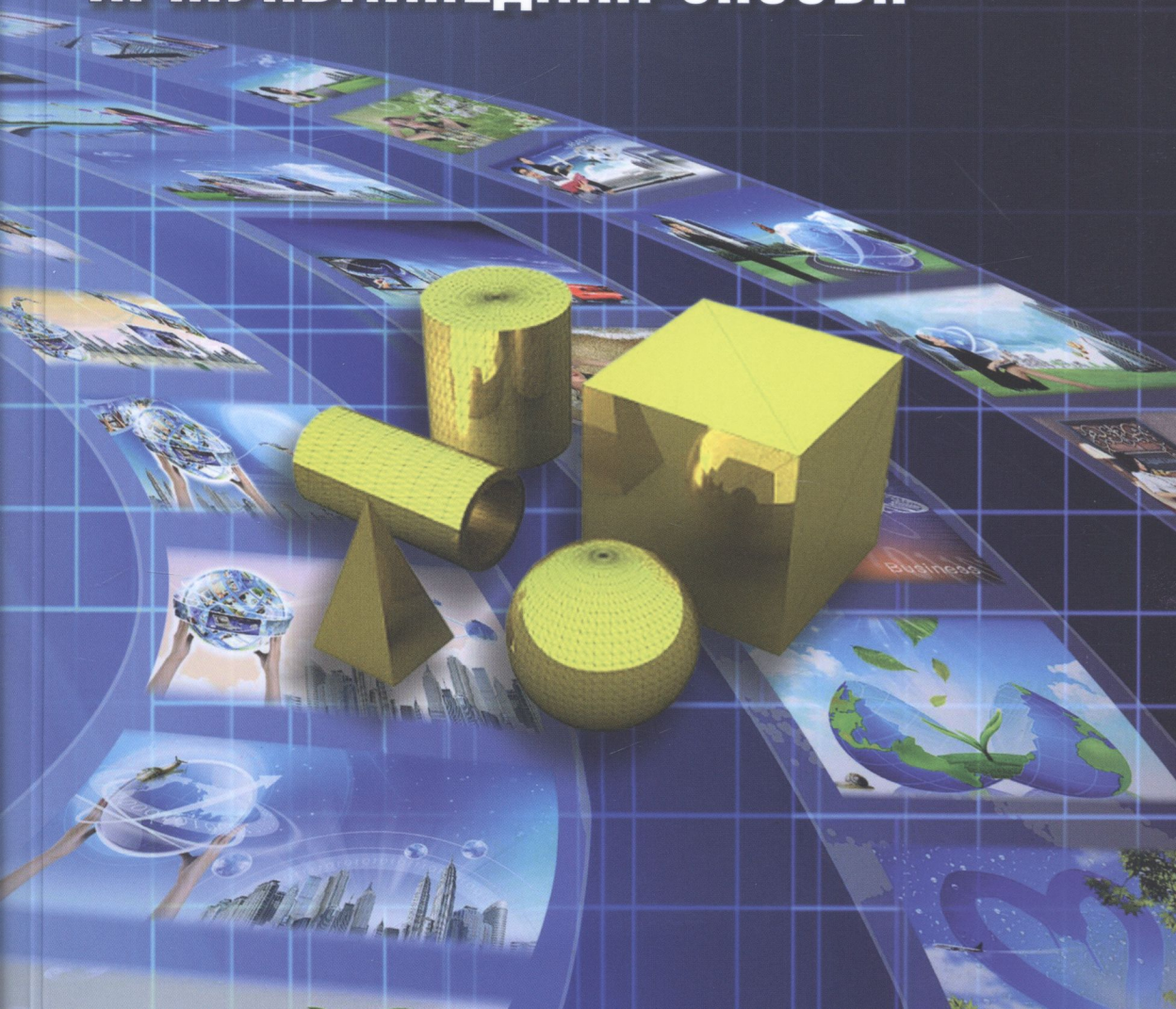
Л. М. Журавчак, О. М. Левченко

Видавництво Львівської політехніки

ПРОГРАМУВАННЯ КОМП'ЮТЕРНОЇ ГРАФІКИ ТА МУЛЬТИМЕДІЙНІ ЗАСОБИ



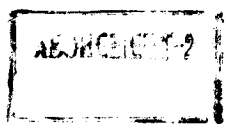
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Л. М. Журавчак, О. М. Левченко

ПРОГРАМУВАННЯ КОМП'ЮТЕРНОЇ ГРАФІКИ ТА МУЛЬТИМЕДІЙНІ ЗАСОБИ

Навчальний посібник

Рекомендувала Науково-методична рада
Національного університету "Львівська політехніка"



Львів
Видавництво Львівської політехніки
2019

УДК 365.881
Ж 85

Рецензенти:

Федасюк Д. В., доктор технічних наук, професор, проректор з науково-педагогічної роботи Національного університету "Львівська політехніка";

Воробель Р. А., доктор технічних наук, професор, завідувач відділу інтелектуальних технологій і систем діагностики Фізико-механічного інституту ім. Г. В. Карпенка НАН України;

Венгерський П. С., доктор фізико-математичних наук, професор кафедри інформаційних систем Львівського національного університету імені Івана Франка

Рекомендувала Науково-методична рада
Національного університету "Львівська політехніка"
як навчальний посібник для студентів спеціальності
"Інженерія програмного забезпечення"
(протокол № 35 від 03.05.2018 р.)

Журавчак Л. М.

Ж 85 Програмування комп'ютерної графіки та мультимедійні засоби : навч. посібник / Л. М. Журавчак, О. М. Левченко. – Львів : Видавництво Львівської політехніки, 2019. – 276 с.
ISBN 978-966-941-276-8

Посібник містить теоретичний та практичний матеріал із дисциплін "Засоби програмування комп'ютерної графіки" та "Програмування мультимедійних систем". У його першій частині викладено базові принципи розроблення і програмування систем комп'ютерної графіки як однієї зі складових мультимедіа, друга присвячена власне мультимедіа – моделям організації, структурі та засадам функціонування, взаємодії з комп'ютерними мережами. Практична складова посібника подає основи програмування на базі графічної бібліотеки OpenGL, опис нових технологій програмування мультимедіа (веб-анімація HTML 5, засоби WPF) та методів створення мультимедійної продукції за допомогою користувацьких програм (Adobe Animate).

Для студентів спеціальності "Інженерія програмного забезпечення", а також для всіх, хто бажає самостійно навчитися працювати із програмованими мультимедійними системами. Може бути корисним для фахівців-початківців, що працюють у галузі мультимедіа, аспірантів і науковців, а також розробників програмного забезпечення.

УДК 365.881

ISBN 978-966-941-276-8

ЗМІСТ

ВСТУП

За останні роки комп'ютерні інформаційні технології до невпізнанності змінили навколишній світ. Комп'ютери, швидко пройшовши еволюційний шлях від громіздких металевих шаф, що займали цілі кімнати, до невеликих, а то й мініатюрних красивих і зручних пристроїв, сьогодні надають користувачам не бачених раніше можливостей: від здійснення арифметичних операцій до розрахунків кліматичних змін на планеті, від пояснення значення слова – до повноцінного навчання найскладніших дисциплін, від простих екранних ігор – до участі у навдивовижу реалістичних подіях віртуального світу. І все більша роль у цьому належить мультимедійним засобам.

У посібнику викладено основи знань, необхідних для розуміння суті такого багатогранного поняття, як мультимедіа та створення найпростіших мультимедійних продуктів. Видання складається з двох частин, у першій з яких розглянуто теоретичні засади і практичні засоби програмування комп'ютерної графіки як однієї з основних складових програмованих мультимедіа, а у другій власне розкрито зміст, структуру і принципи функціонування мультимедіа та особливості їхнього сприйняття людиною.

Комп'ютерну графіку сьогодні повсюдно застосовують у найрізноманітніших галузях людської діяльності, і питання ефективності, реалістичності і швидкості генерації зображень продовжують залишатись актуальними. У посібнику розглянуто процедури формування і візуалізації об'єктів дво- і тривимірної комп'ютерної графіки, зокрема описано математичне подання криволінійних поверхонь, тривимірний конвеєр спостереження, моделі освітлення, методи візуалізації поверхонь і накладання текстур. Оскільки всі зазначені теоретичні методи вже ефективно реалізовано у програмних продуктах, практична складова цієї частини посібника містить базові знання щодо використання відкритої графічної бібліотеки OpenGL – потужного інструмента для створення високоякісних програмно генерованих графічних об'єктів та інтерактивних застосувань.

У другій частині посібника розглянуто вже самі мультимедіа: базові моделі їхньої організації, структуру та основи функціонування головних складових, принципи взаємодії з комп'ютерними мережами та використання мультимедійних засобів з Інтернету. Практичний аспект цієї частини посібника пов'язаний не лише з новими технологіями програмування мультимедіа (веб-анімація HTML 5, застосунки WPF у вікнах нестандартної форми), але й зі створенням мультимедійних продуктів за допомогою поширених програмних комплексів (наприклад, Adobe Animate).

Частина І

Засоби програмування комп'ютерної графіки

Розділ 1

ОСНОВНІ ПОНЯТТЯ КОМП'ЮТЕРНОЇ ГРАФІКИ

Комп'ютерна графіка (КГ) – це галузь інформатики, до сфери інтересів якої входять усі аспекти формування зображень за допомогою комп'ютерів. У цій галузі знань вивчають і розробляють засоби та методи створення і перетворення графічних зображень об'єктів.

Основні задачі КГ:

- введення до комп'ютера інформації, що має графічну форму або визначає її;
- оброблення, оптимізація характеристик, зберігання на носіях, захист, передавання засобами локальних та глобальних мереж цієї інформації;
- виведення інформації в графічній формі з комп'ютера.

Графічна форма подання інформації:

- ескізи, креслення;
- візуальне подання каркасних, поверхневих та твердотільних тривимірних моделей різноманітних об'єктів (природних та штучних, статичних і динамічних, живих та неживих);
- схеми, діаграми, графіки, рисунки;
- фотографії, відео, анімація, голограми, мультимедійна інформація тощо.

Відмінності графічних прикладних програм від звичайних:

1. Двовимірна і тривимірна геометрія графічних об'єктів визначає абсолютно інші типи даних, для яких дуже швидко розвиваються власні математичні методи й алгоритми. Особливо це стосується формування зображень тривимірних об'єктів.
2. Інтерактивні графічні програми відрізняються від програм, які використовують у пакетному режимі обробки, тим, що ними керують події. Це означає, що ці програми очікують яких-небудь дій користувача, потім певним чином реагують на них і знову переходять у режим очікування.
3. Використання апаратно-незалежного графічного пакета високого рівня суттєво спрощує інтерактивне рисування і підготовку простих графіків. Але у складніших застосуваннях проявляються особливості графічного програмування.

1.1. Архітектура графічних систем

За час існування графічних систем (ГС) запропоновано багато варіантів структурної організації засобів реалізації функцій графічного *інтерфейсу прикладного програмування* (ІПП, *application programming interface* – API).

1.1.1. Розвиток структури графічних систем

У перших графічних системах використовували обчислювальні машини загального призначення зі стандартною архітектурою, яку запропонував ще фон Нейман. У таких ЕОМ був один процесор, який у кожен момент часу виконував одну інструкцію. Ранні ГС мали таку структуру: **головний комп'ютер, цифро-аналоговий перетворювач, електронно-променева трубка (ЕПТ)**. Як пристрій відображення використовували дисплей, який формував на екрані відрізки прямих між двома заданими точками чи дуги за трьома точками. Головний (і єдиний у такій системі) комп'ютер виконував прикладну програму й обчислював координати характеристичних точок примітивів у системі координат екрана. Цю інформацію передавали на дисплей із доволі високою швидкістю, щоб уникнути миготіння відображення на екрані. В ті не такі вже й далекі часи швидкість обчислень була настільки низькою, що виведення на екран навіть декількох сотень відрізків потребувало використання практично всіх обчислювальних ресурсів машини.

Перші спроби створення спеціалізованих ГС мали на меті звільнити головний комп'ютер від рутинних операцій регенерації зображення на екрані дисплея. До складу дисплея ввели **дисплейний процесор**, який узяв на себе завдання регенерації. Він мав ту саму архітектуру, що й процесор загального призначення, але додатково до його системи команд було введено *інструкції формування графічних примітивів*. Перевага такої структурної організації всієї системи полягала в тому, що головному комп'ютеру потрібно було тільки сформувати опис зображення і передати його у вигляді **дисплейного файла** до дисплейного процесора. Той зберігав ці дані у власній пам'яті і зчитував їх із частотою регенерації, звільняючи таким чином головний комп'ютер від рутинної роботи. Цей варіант архітектури в загальних рисах відтворено у сучасних ГС.

Отже, сучасна система КГ є передусім обчислювальною системою і містить усі **компоненти** обчислювальної системи загального користування:

- процесор;
- пам'ять;
- буфер кадру;
- пристрої виведення;
- пристрої введення.

Ця модель має доволі загальний характер та відображає структуру і графічної робочої станції, і персонального комп'ютера, і графічного терміналу великої обчислювальної системи, яка працює в режимі розподілу машинного часу, й

інтелектуальної системи формування зображень. Хоч усі компоненти наявні і в стандартному комп'ютері (крім, можливо, буфера кадру), саме спеціалізація кожної компоненти відповідно до вимог задач КГ і робить систему графічною.

1.1.2. Буфер кадру

Сьогодні практично всі ГС використовують **растровий принцип створення зображень**. У растрових дисплеях дисплейні примітиви (відрізки, літери, зафарбовані ділянки, переважно багатокутники) зберігаються в пам'яті для регенерації у вигляді сукупності точок, що їх утворюють; ці точки називають *пікселями* (*pixels*) – від словосполучення *picture element*. Зображення формується на растрі, що є сукупністю горизонтальних рядків, кожен з яких складається з окремих пікселів. Отже, **растр** – це матриця з пікселів, що покриває всю площу екрана. Усе зображення послідовно відображається 30 разів за секунду по окремих рядках растру в напрямку згори донизу.

Буфер кадру – це окрема область пам'яті, в якій зберігається масив кодів, що визначають засвічення пікселів на екрані. В системах особливо високої якості для буфера кадру використовують спеціальні типи мікросхем: *відеопам'ять з довільним доступом* (VRAM) або *динамічну пам'ять з довільним доступом* (DRAM), які дають змогу швидко вивести вміст буфера на екран.

Глибина буфера кадру характеризує кількість бітів інформації, які визначають засвічення кожного окремого пікселя, зокрема кількість кольорів, яку можна подати на екрані цієї системи. Наприклад, буфер глибиною 1 біт дає змогу виводити тільки двоградаційне (чорно-біле) зображення, а буфер глибиною 8 бітів – зображення, що складається з елементів $2^8 = 256$ кольорів. Сучасні *повноколірні* системи характеризуються глибиною буфера 24 біти (а інколи й більше). Вони здатні створювати по-справжньому фотореалістичні зображення. Інколи їх називають системами з *правильним передаванням кольору* або *RGB-системами*, оскільки в кодуванні засвічення кожного пікселя можна виділити окремі групи бітів, які характеризують інтенсивність засвічення по кожному з основних кольорів – червоному, зеленому і синьому. У простіших системах буфер кадру виділяють в основній пам'яті комп'ютера. Розмір буфера кадру визначає одну з головних характеристик ГС – *роздільну здатність*.

У простих системах зазвичай використовують єдиний *процесор*, на який покладено розв'язування і "звичайних" задач, і задач КГ. Основні графічні функції, в принципі, зводяться до перетворення опису графічного примітиву (відрізка прямої, кола чи багатокутника), сформованого прикладною програмою, на коди засвічення певних пікселів у буфері кадру. Процес перетворення опису графічного примітиву на коди засвічення пікселів отримав назву *растрового перетворення*, або *сканувального перетворення*. В сучасних високопродуктивних ГС для такого перетворення використовують спеціалізовані процесори, причому не один, а декілька, і кожен з них виконує свій набір графічних функцій.

1.1.3. Пристрої виведення зображень

В останні роки все частіше графічні монітори є *дисплеями з плоским екраном*, оскільки ці відеопристрої мають менший об'єм, вагу і споживання енергії. Їх можна поділити на дві категорії: випромінювальні (емітери) і невипромінювальні (неемітери). Випромінювальні дисплеї – це пристрої, які перетворюють електричну енергію на світло, у невипромінювальних використовують оптичні ефекти, за допомогою яких сонячне світло чи світло якогось іншого джерела перетворюється на графічне зображення. До першої категорії належать плазмові табло, тонкоплівкові електролюмінісцентні дисплеї, світловипромінювальні діоди, до другої – пристрої на рідких кристалах.

1.2. Стандарти ПЗ КГ

У результаті об'єднаних зусиль міжнародних і національних організацій зі складання стандартів у 1984 році створено **базову графічну систему** (**GKS** – *Graphical Kernel System*). Її прийняли як перший стандарт графічного ПЗ Міжнародна організація зі стандартизації (ISO – *International Standards Organization*) та різні національні організації зі стандартизації. Незважаючи на те, що GKS спочатку розробляли як пакет для двовимірної графіки, незабаром з'явилось тривимірне доповнення до GKS. Другим розробленим і прийнятим організаціями зі стандартизації стандартом ПЗ був **PHIGS** (*Programmer's Hierarchical Interactive Graphics Standard* – ієрархічна інтерактивна графічна система програміста) як продовження GKS. Стандарт PHIGS відрізнявся ширшим діапазоном можливостей ієрархічного моделювання об'єктів, задання кольорів і виконання різних дій над зображеннями. Пізніше з'явилось його продовження **PHIGS+**, у якому додали можливості тривимірного зафарбовування поверхонь.

У той час, коли розробляли пакети GKS і PHIGS, все популярнішими ставали графічні робочі станції виробництва компанії Silicon Graphics, Inc., SGI. Їх випускали разом із набором стандартних функцій за назвою GL (*Graphics Library* – графічна бібліотека), який дуже швидко став доволі популярним у комп'ютерній графіці. Отож пакет GL став де-факто графічним стандартом. Тобто на початку 1990-х років розробили пакет OpenGL як апаратно-незалежну версію пакета GL. Зараз цей пакет підтримує і оновлює організація *OpenGL Architecture Review Board*, яка є консорціумом представників багатьох графічних компаній і організацій. Бібліотеку OpenGL розроблено спеціально для ефективної обробки тривимірних даних, але вона може працювати і з описами двовимірних сцен як із частковим випадком тривимірного зображення, де всі значення координати z дорівнюють 0.

Графічні функції в будь-якому пакеті зазвичай задають як набір описів, незалежних від мови програмування. Потім задають **прив'язку до мови** для певної мови програмування високого рівня. Ця прив'язка задає синтаксис, який дає змогу користуватися різними графічними функціями цієї мови. Кожну прив'язку до мови задають так, щоб можна було максимально використати відповідні можливості

мови і керувати різними моментами, пов'язаними з синтаксисом, такими як типи даних, задання параметрів і обробка помилок. Специфікації для реалізації графічного пакета у конкретній мові встановлює Міжнародна організація зі стандартизації. Прив'язки пакета OpenGL до мов C та C++ однакові. У наступних розділах користуватимемося прив'язкою до мов C/C++ для пакета OpenGL як основою для вивчення основних графічних понять і для розроблення та застосування графічних програмних пакетів. Приклади програм мовою C++ ілюструватимуть застосування пакета OpenGL і загальні алгоритми реалізації графічних функцій.

Розроблено й інші бібліотеки для програмування в галузі КГ. Деякі з них пропонують загальні графічні стандартні функції, а деякі призначені для спеціальних застосувань чи окремих аспектів КГ, таких як анімація, віртуальна реальність чи графіка в мережі Internet.

Для простого й ефективного вирішення завдань, пов'язаних з ігровим і відеопрограмуванням під Microsoft Windows, використовують набір API-функцій *DirectX*. Пакет за назвою *Open Inventor* надає набір об'єктно-орієнтованих стандартних функцій для опису сцен, які треба зобразити за допомогою звертань до OpenGL (цей інструментарій написано мовою C++). *Мова моделювання віртуальної реальності* (VRML), яка починалась як підмножина пакета Open Inventor, дає змогу створювати тривимірні моделі віртуальних світів у мережі Internet. Зображення на Web-сторінках можна створювати й засобами графічних бібліотек, розроблених для мови Java. За допомогою *Java 2D* й *Java 3D* можна рисувати відповідно дво- та тривимірні Web-зображення. А за допомогою *Renderman Interface* виробництва компанії Pixar Corporation можна генерувати сцени, використовуючи різноманітні моделі освітлення. І, нарешті, графічні бібліотеки часто пропонують частину систем іншого типу, таких як програми *Mathematica, MatLab* і *Maple*.

1.3. Системи координат

Система координат (СК) – це сукупність правил, які ставлять у відповідність кожному об'єкту (точці) набір чисел або координат. Кількість координат для задання точки називають *розмірністю простору*. За деякими винятками, в програмних пакетах загального призначення необхідно геометричні описи об'єктів задавати у стандартній правосторонній декартовій СК. Якщо значення координат рисунка задано в якійсь іншій СК (полярній, циліндричній, сферичній), перед їхнім введенням у програмний пакет їх треба перетворити на декартові. У деяких пакетах, розроблених для спеціальних класів задач, допускають використання інших СК.

Залежно від структури подання зображень та від процесу обробки графічних даних використовують різні **типи координат**:

- **абсолютні** – задають позицію точки відносно початку певної системи координат;

- **відносні** – задають позицію точки відносно деякої іншої адресованої точки;
- **координати користувача** – задані користувачем, показують позицію точки в системі координат, обраній ним, і не залежать від конкретних пристроїв відображення;
- **світові** – незалежні від пристрою декартові координати, які використовують у прикладній програмі для задання вхідних та вихідних графічних даних;
- **фізичні** – задані в системі координат, залежній від пристрою зображення;
- **нормалізовані** – задані в проміжній, незалежній від пристрою, системі координат і нормовані відносно деякого діапазону (від 0 до 1); при цьому зображення розміщується в тій самій відносній позиції під час візуалізації на будь-якому пристрої.

Під час роботи з першими системами КГ потрібно було у прикладній програмі всю геометричну інформацію задавати в термінах СК екрана пристрою відображення. Одна з переваг сучасних графічних програм – можливість використання в них будь-яких одиниць вимірювання, які захоче вибрати користувач, керуючись тільки специфікою задачі. Концепція *графіки, інваріантної до пристроїв*, звільнила прикладних програмістів від необхідності врахування деталей конструкції пристроїв уведення-виведення графічної інформації. Щодо застосовуваної користувачем СК використовують термін **світова** або **прикладна СК**. Користувач може вибирати будь-який діапазон значень координат, причому єдиним, і дуже незначним, обмеженням є діапазон подання дійсних чисел у сучасних комп'ютерах з плаваючою крапкою.

Систему відліку на екрані конкретного пристрою раніше називали **координатами фізичного пристрою** або **координатами пристрою**. Для растрових пристроїв, якими є більшість сучасних дисплеїв, зараз використовують термін **координати растра** або **координати екрана**. Координати растра завжди подають цілими числами, оскільки пікселі займають цілком визначене місце у вузлах фіксованої сітки, тобто пікселі за своєю природою є дискретними об'єктами і, отже, адресуються цілими числами. Перетворення з СК прикладної програми на координати растра виконує ГС, і програміст про це не повинен турбуватись. У прикладній програмі треба задати тільки декілька параметрів, необхідних для виконання такого перетворення – образ світового простору, який відображається, і розмір поля екрана.

У проектувальній геометрії *однорідні координати* застосовували ще задовго до виникнення й поширення КГ. Вони дають змогу n-вимірний об'єкт подати в $(n+1)$-вимірному просторі додаванням ще однієї координати – скалярного множника.

У загальному випадку будь-яку точку $(X_1, X_2, ..., X_n)$ в n-вимірному просторі можна записати у вигляді $(hX_1, hX_2, ..., hX_n, h)$ в $(n+1)$-вимірному просторі, де h – будь-яке ненульове число. Ця група з $(n+1)$ чисел визначає **однорідні координати** вихідної точки з n-вимірного простору.

Приклад: точку (4, 6) у двовимірному просторі можна записати в однорідних координатах як (12, 18, 3) або як (400, 600, 100) в тривимірному.

Основні властивості однорідних координат пов'язані з **можливостями**:

- виразити за допомогою єдиної матриці всі перетворення – обертання, перенесення тощо – і навіть проекції (аксонометричні чи перспективні);
- фіксувати віддалені або розміщені в нескінченності точки;
- реалізувати будь-які поєднання перетворень за допомогою простого перемноження матриць.

Ці переваги однорідних координат і зумовили їхнє широке використання в КГ.

1.4. Прикладний інтерфейс OpenGL

Структура OpenGL аналогічна структурі більшості графічних ІПП, зокрема PHIGS, GKS. OpenGL підтримує роботу як з дво-, так і з тривимірними примітивами.

Основна бібліотека в пакеті OpenGL створена для специфікації графічних примітивів, атрибутів, геометричних перетворень, перетворень спостереження і багатьох інших операцій. Як було зазначено раніше, пакет OpenGL розробляли незалежним від апаратних засобів, тому багато операцій, зокрема, функції введення-виведення, не входять до основної бібліотеки. Вони, а також багато додаткових функцій, доступні у допоміжних бібліотеках, розроблених для програм OpenGL.

Перед іменами функцій **основної бібліотеки OpenGL** (її ще називають кореневою бібліотекою) ставиться префікс **gl**, кожне слово, що входить до імені функції, починається з великої літери, а самі функції зберігаються в бібліотеці, яку позначають **GL**. Наступні приклади ілюструють такий принцип іменування:

`glBegin, glClear, glCopyPixels, glPolygonMode.`

Деякі функції вимагають, щоб одному (або декільком) їхнім аргументам присвоювалось значення символьної константи, яка позначає, наприклад, ім'я параметра чи певний режим. Кожна така константа починається з великих літер **GL,** слова, що складають її ім'я, теж пишуть великими літерами, а як розділювач між словами використовують знак підкреслення "_". Далі наведено приклади з декількох сотень символьних констант, які можуть бути використані функціями пакета OpenGL:

`GL_RGB, GL_POLYGON, GL_AMBIENT_AND_DIFFUSE.`

Функції пакета OpenGL приймають особливі типи даних. Наприклад, параметр функції може набувати значень, заданих як 32-розрядні цілі числа, але розмір специфікації цілого числа на різних комп'ютерах може відрізнятись. Для позначення особливого типу даних використовують спеціальні вбудовані назви типів даних

`GLbyte, GLshort, GLint, GLfloat, GLdouble, GLboolean.`

Кожна назва типу даних починається з великих літер **GL,** а інша частина назви – це стандартне позначення типу даних, написане малими.

Деяким аргументам функцій можна присвоювати значення за допомогою масиву, який містить набір значень. Ця опція специфікує список значень як вказівник на масив, а не як кожен елемент списку явно у вигляді аргументу параметра.

Крім основної, є й декілька **додаткових бібліотек.**

1. **Бібліотека графічних утиліт (GLU)** містить стандартні функції, які дають змогу налаштовувати матриці проекції та візуалізації, описувати складні об'єкти наближено прямими і багатокутниками, зображати квадратичні й бі-сплайни за допомогою лінійного наближення, зафарбовувати поверхні; до її складу входять функції формування складних об'єктів типу сферичних поверхонь. Кожна реалізація пакета OpenGL містить цю бібліотеку, а всі імена її функцій починаються з префікса **glu**. Функції цієї бібліотеки звертаються тільки до функцій GL.
2. Вікно зображення не можна створити безпосередньо за допомогою основних функцій OpenGL, оскільки ця бібліотека містить тільки незалежні від пристрою графічні функції, а операції керування вікнами залежать від конкретного типу комп'ютера. Тому розроблено **декілька бібліотек систем вікон,** які підтримують функції для різних комп'ютерів:

- GLX – розширення для системи X-Windows (стандартні функції починаються з префіксу **glx**);
- AGL – інтерфейс Apple GL (префікс **agl**);
- WGL – інтерфейс Windows-to-OpenGL (префікс **wgl**);
- PGL – інтерфейс Presentation Manager to OpenGL для OS/2 компанії IBM (префікс **pgl**).

3. Службовий інструментарій **OpenGL Utility Toolkit (GLUT)** містить функції для роботи з будь-якою системою вікон на екрані та методи, які дають змогу зафарбовувати криві та поверхні другого порядку, цій бібліотеці відповідає префікс **glut**. Вона забезпечує користувача основними можливостями, характерними для більшості сучасних багатовіконних систем.

1.4.1. Файли заголовків

У всіх графічних програмах мають бути файли заголовків для *системи вікон, основної бібліотеки* та *бібліотеки GLU.* Наприклад, для системи Microsoft Windows файл заголовка для доступу до стандартних функцій WGL – це windows.h, цей файл має бути розташований перед файлами заголовків для бібліотек OpenGL. Тому вихідний файл у цьому випадку буде починатися так:

```
#include <windows.h>
#include <GL/gl.h>
#include <GL/glu.h>
```

Однак якщо для виконання операцій керування вікном користуватися бібліотекою GLUT, то не треба включати файли **gl.h** і **glu.h**, оскільки в цій бібліотеці вже передбачено, що вони будуть враховані правильно. Тому можна замінити файли заголовків для основної бібліотеки та бібліотеки GLU на

```
#include <GL/glut.h>
```

Крім того, часто треба включати файли заголовків для коду C++:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
```

У новому стандарті ISO/ANSI для C++ їх названо **cstdio**, **cstdlib** і **cmath**.

1.4.2. Керування вікнами зображення за допомогою бібліотеки GLUT

Для початку розглянемо **мінімальний набір операцій**, необхідний для створення зображення.

1. Ініціалізація бібліотеки GLUT для створення сеансу зв'язку між підсистемою вікон і OpenGL:

```
glutInit(&argc, argv);
```

2. Створення вікна зображення з відповідною назвою в рядку заголовка:

```
glutCreateWindow("Заголовок вікна на екрані дисплея");
```

У момент опрацювання цієї команди вікно визначається, але не виводиться на екран, поки не будуть завершені всі операції налаштування GLUT.

Під час створення вікна буде встановлено його параметри, призначені графічною системою за замовчуванням: розмір, положення на екрані і режим RGB керування кольором. Для налаштування інших параметрів треба перед викликом **glutCreateWindow** звернутись до відповідних функцій їхнього встановлення.

У GLUT є дві функції для визначення вікна на екрані дисплея і вибору його розмірності та розташування:

```
glutInitWindowPosition(xTopLeft, yTopLeft);
glutInitWindowSize(dwWidth, dwHeight);
```

Перша з цих функцій за допомогою двох цілих чисел визначає розташування в екранних координатах лівого верхнього кута вікна зображення на екрані дисплея щодо лівого верхнього кута екрана. Якщо якась із координат від'ємна, точка вікна на екрані дисплея визначається системою керування вікнами. За допомогою *другої* функції задають ширину і висоту вікна на екрані дисплея в пікселях (додатне ціле число). Якщо ці функції задання розміру і розташування не використано, за замовчуванням розмір дорівнює 300 на 300, а розташування – (-1, -1), і вікно розміщує система керування вікнами.

За допомогою ще однієї функції вибирають різні параметри вікна на екрані дисплея:

```
glutInitDisplayMode(mode);
```

цю функцію використовують для вибору колірного режиму і різних комбінацій буферів; задані параметри об'єднують логічною операцією АБО. Режим за замовчуванням – проста буферизація (один буфер) і колірний режим RGB (чи RGBA):

```
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
```

Специфікація колірного режиму **GLUT_RGB** еквівалентна **GLUT_RGBA.**

Приклад встановлення вікна розміром 480 на 640 пікселів, верхній лівий кут якого розміщено на 50 пікселів праворуч від лівого краю і на 100 пікселів униз від верхнього краю екрана з використанням одного буфера регенерації та колірного режиму RGB:

```
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowPosition(50, 100);
glutInitWindowSize(480, 640);
```

3. Передання у вікно зображення створеного прикладною програмою рисунка за допомогою *функції відображення зі зворотним викликом*:

```
glutDisplayFunc(lineGraph);
```

4. Активізація всіх створених вікон зображень разом зі своїм графічним вмістом:

```
glutMainLoop();
```

Ця функція має стояти на останньому місці в програмі. Вона призначена для відображення початкових графічних елементів і вводить програму в нескінченний цикл, у якому система перевіряє дані, що надходять від клавіатури чи миші. Для неінтерактивних програм вона дає змогу користувачу розглядати створене зображення доти, поки він не натисне яку-небудь клавішу.

1.5. Категорії графічних функцій

До складу потужного пакета ІПП може входити кілька сотень функцій, а тому бажано відразу ж розділити їх на категорії. Розглянемо функції 6 категорій.

1. **Функції опису примітивів** визначають об'єкти нижнього рівня ієрархії – примітиви, які здатна відображати ГС. У більшості графічних ІПП є такі примітиви, як *точки, відрізки прямих ліній, багатокутники, пікселі, текст, різного роду криволінійні відрізки* і *ділянки криволінійних поверхонь.*
2. **Функції задання атрибутів** дають змогу прикладному програмісту виконувати широке коло операцій налаштування зображень: *вибору кольору,*

типу ліній, шрифту тексту для надписів на графіку, зразка (шаблону) заповнення окремих ділянок тощо.

Отже, якщо вибором *примітивів* визначають, що з'явиться на екрані, то за допомогою *атрибутів* – те, як виглядатимуть відображені об'єкти, тобто атрибути визначають спосіб виведення об'єктів на екран.

3. Одну з найцікавіших можливостей якісного графічного пакета надає її набір **функцій геометричних перетворень**, які дають змогу користувачу виконувати різні перетворення об'єктів – *поворот, плоско-паралельне перенесення, масштабування* тощо.
4. Потім треба задати параметри використовуваної моделі синтезованої камери, за допомогою якої буде створено зображення. Прикладний програміст повинен вибрати положення та орієнтацію камери в світовій системі координат (СК) і параметри об'єктива, зокрема фокусну відстань. Знаючи ці параметри системи, він зможе не тільки правильно будувати зображення, але й відтинати ті об'єкти, котрі опиняються поза полем зору. **Функції візуалізації** допомагають вибрати *точку спостереження на екрані*, *вигляд проекції*, яку треба використати в цьому випадку, і *місце на моніторі*, де буде розміщено зображення. Є також інші стандартні функції для керування областю зображення на екрані через задання її координат, розміру і структури. Для тривимірних сцен визначають видимі об'єкти і накладають відповідні умови освітлення, тобто функції візуалізації дають змогу задати різноманітні вигляди, хоч різні типи ІПП істотно відрізняються можливостями маніпулювання виглядами.
5. Для створення інтерактивних програм потрібно, щоб у складі ІПП були й **функції введення графічної інформації**. Вони відіграють роль проміжної ланки між пристроями введення, такими як клавіатура, миша, планшети різного роду, і прикладною програмою, перевіряють і обробляють потік даних від цих інтерактивних пристроїв.
6. У реальних застосуваннях програмісту доводиться думати і про керування середовищем функціонування програми, особливо у випадку, коли йдеться про багатопроцесорну систему, багатовіконне операційне середовище чи мережеве середовище з багатьма користувачами. Для впливу на процес виконання програми до складу ІПП зазвичай включають **спеціальні функції керування**. Вони дають змогу прикладній програмі взаємодіяти з операційною системою, ініціалізувати застосування та обробляти помилки інших графічних функцій.

1.6. Контрольні питання

1. Перерахуйте основні задачі комп'ютерної графіки.
2. Які відмінності між пасивними графічними системами та інтерактивною графікою?
3. Назвіть основні концепції, на яких ґрунтується графічна система?

4. Які відмінності графічних прикладних програм від звичайних?
5. З яких компонент складається графічна система?
6. Які переваги отримала графічна система після приєднання до неї дисплейного процесора?
7. У чому суть растрового принципу створення зображень?
8. Дайте визначення буфера кадру та його глибини.
9. Якою глибиною буфера кадру характеризуються сучасні RGB-системи?
10. Які є способи формування растра на екрані?
11. Охарактеризуйте стандарти програмного забезпечення КГ.
12. Які є типи координат?
13. Які переваги використання однорідних координат?
14. Який принцип іменування використовують для функцій основної бібліотеки OpenGL?
15. Охарактеризуйте додаткові бібліотеки OpenGL.
16. Яким є мінімальний набір операцій для створення зображення?
17. Опишіть процес створення вікна для виведення зображення.
18. Опишіть категорії графічних функцій.

1.7. Приклади тестових питань

1. Координати, які задані в проміжній, незалежній від пристрою, системі координат і нормовані щодо деякого діапазону – це координати:

   а) користувача;
   б) світові;
   в) нормалізовані.

2. Точка (4, 6, 7) у тривимірному просторі може бути записана в однорідних координатах у чотиривимірному як:

   а) (12, 18, 21, 3);
   б) (4, 6, 7, 4);
   в) (4, 6, 7, 1).

3. Координати, які задані в системі координат, залежній від пристрою зображення, – це координати:

   а) користувача;
   б) світові;
   в) фізичні.

4. Точка (12, 18, 24, 6) в однорідних координатах може бути записана у тривимірному просторі як:

   а) (12, 18, 24, 3);
   б) (2, 3, 4);
   в) (12, 18, 24).

5. Координати, які задають позицію точки щодо деякої іншої адресованої точки – це:

   а) абсолютні;
   б) відносні;
   в) нормалізовані.

6. Координати, які задають позицію точки щодо початку визначеної системи координат – це:

   а) абсолютні;
   б) відносні;
   в) нормалізовані.

7. Абсолютні координати в графічній системі – це такі координати, які:

   а) задають позицію точки щодо початку графічного пристрою;
   б) визначають позицію деякої довільної точки щодо початку системи координат;
   в) вводяться користувачем графічної системи.

8. Світові координати – це:

   а) координати, які використовують у прикладній програмі в процесі задавання графічних вхідних та вихідних даних;
   б) залежні від пристрою декартові координати;
   в) незалежні від пристрою декартові координати.

9. Першим стандартом графічного програмного забезпечення, який прийняла Міжнародна організація зі стандартизації, був:

   а) GKS;
   б) PHIGS;
   в) OpenGL.

10. Імена функцій основної бібліотеки OpenGL починаються з префікса:

    а) **gl**;
    б) **glu**;
    в) **glut**.

11. Імена функцій бібліотеки графічних утиліт OpenGL починаються з префікса:

    а) **gl**;
    б) **glu**;
    в) **glut**.

Розділ 2

ФУНКЦІЇ ОПИСУ ГРАФІЧНИХ ПРИМІТИВІВ OpenGL

Описувати можливості **інтерфейсу прикладного програмування (ІПП)** будемо через **функції його бібліотеки**.

2.1. Функції опису базових графічних примітивів OpenGL

За допомогою функцій примітивів, які є в основній бібліотеці, можна будувати точки, прямолінійні відрізки, зафарбовані опуклі багатокутники, а також піксельні чи растрові матричні структури. У бібліотеці GLUT можна знайти стандартні процедури для зображення рядків символів. Інші види примітивів, такі як кола, еліпси чи зафарбовані увігнуті багатокутні області, можна будувати за допомогою функцій основної бібліотеки або стандартними процедурами з бібліотек GLU і GLUT. Задавати значення координат можна цілими числами або дійсними з плаваючою крапкою, крім того, можна задавати положення за допомогою вказівника на масив координатних значень.

Базові примітиви OpenGL (точки, відрізки, ламані лінії, багатокутники) специфікують набором **точок** у просторі (вершин). Формат визначення об'єктів такий:

```
glBegin(тип);
   glVertex*(); // перша точка (вершина)
   ...;
   glVertex*(); // остання точка (вершина)
glEnd();
```

Значення "тип" визначає вигляд об'єкта і надає інформацію про те, як треба інтерпретувати подальший список вершин. Функція **glVertex** повинна міститись у програмі між функціями **glBegin** і **glEnd**. Аргумент функції **glBegin** визначає **тип графічного примітиву**, який треба зобразити, а функції **glEnd** не потрібні аргументи.

Символ * означає, що для цієї функції необхідні **індексні коди**:

- розмірність простору: **2**, **3** чи **4** (для однорідних координат);
- тип числових даних, які використовують у ролі значень координат: **i** (integer – цілі числа), **s** (short – короткі цілі числа), **f** (float – числа з плаваючою крапкою), **d** (double – числа подвійної точності);

- можливість подання координат у вигляді вектора (одновимірного масиву): третій індекс – **v** (vector).

2.1.1. Створення точок

Типом графічного примітиву для точок є **GL_POINTS**.

Приклад зображення на екрані трьох точок, які розташовані на однаковій відстані одна від однієї, коли координати задано *парами цілих чисел*:

```
glBegin(GL_POINTS);
   glVertex2i(50, 100);
   glVertex2i(75, 150);
   glVertex2i(100, 200);
glEnd();
```

2.1.2. Створення відрізків

В OpenGL є кілька способів **формування відрізків**:

- тип **GL_LINES** задає створення відрізків за двома вершинами, які є його початковою і кінцевою точками; помістивши між функціями **glBegin** і **glEnd** кілька пар точок, задають відразу кілька відрізків; у загальному випадку отримаємо набір *нез'єднаних* між собою відрізків, якщо значення координат не повторюються, причому перший відрізок з'єднує першу і другу точки, другий – третю і четверту точки, і так далі; якщо задати лише одну точку, то нічого не буде зображено, а якщо кількість точок у списку буде непарною, то останнє значення не опрацьовуватиметься;
- якщо необхідно сформувати *відкриту ламану лінію*, то використовують тип **GL_LINE_STRIP**; отримаємо послідовність *з'єднаних* між собою відрізків, яка починається у першій точці списку і закінчується в останній, причому перший відрізок з'єднує першу і другу точки, другий – другу і третю точки, і так далі;
- якщо необхідно сформувати *замкнуту ламану лінію, використовують тип* **GL_LINE_LOOP**; до описаної раніше послідовності відрізків додається ще один відрізок, який з'єднує останню точку послідовності з першою точкою списку.

Приклад зображення набору нез'єднаних відрізків, коли координати точок задано у вигляді *масивів двох цілих чисел*:

```
glBegin(GL_LINES);
   glVertex2iv(p1);
   glVertex2iv(p2);
   glVertex2iv(p3);
   glVertex2iv(p4);
glEnd();
```

2.1.3. Створення зафарбовуваних багатокутників

Примітив типу "замкнута лінія" не має внутрішньої області. Такою властивістю володіють примітиви іншого типу – **багатокутники**. Вони відіграють особливу роль у КГ, оскільки в растрових системах зображення таких примітивів формується дуже швидко, бо їхні межі описано лінійними рівняннями. До того ж криволінійні поверхні можна достатньо точно апроксимувати набором правильних багатокутників, а криву лінію – набором прямолінійних відрізків. А після накладання ефектів освітлення і затінення апроксимована криволінійна поверхня виглядає цілком реалістично. Об'єкти, описані за допомогою набору багатокутників, зазвичай називають *стандартними графічними об'єктами*. **Продуктивність ГС** прийнято зараз оцінювати кількістю багатокутників, що виводяться на екран упродовж 1 секунди.

Для коректного відображення внутрішньої області багатокутника він повинен мати 3 властивості – бути *простим* (його сторони не перетинаються), *опуклим* (всі його внутрішні кути менші або дорівнюють 180°, а відрізок, що з'єднує будь-які дві точки всередині багатокутника, теж повністю належить йому) *і плоским* (всі його вершини лежать в одній площині). У більшості графічних ІПП передбачено, що перевірка, чи багатокутник є простим, виконується у прикладній програмі. У деяких ІПП, зокрема і в OpenGL, гарантовано правильне виконання зафарбування тільки у випадку, якщо багатокутник є *опуклим*. Якщо використовувати для апроксимації криволінійних поверхонь тільки трикутники, то ГС матиме справу виняткова з плоскими багатокутниками, котрі завжди тонуються коректно. Крім того, зафарбування трикутних областей виконується апаратно чи програмно значно швидше, ніж багатокутників іншої форми.

Створити зафарбовані багатокутники можна, задавши одну з шести символьних констант у функції **glBegin** разом зі списком команд **glVertex**:

- **GL_POLYGON** – всі задані командою **glVertex** точки визначають багатокутник, його вершини задані *проти годинникової стрілки*; має бути як мінімум три вершини;
- **GL_TRIANGLES** – кожні три задані командою **glVertex** точки визначають трикутник: перші три точки задають вершини першого трикутника, наступні три – другого і т.д. (отримаємо набір *незєднаних* між собою трикутників, якщо жодні вершини не повторюватимуться; для кожного трикутника вершини задані проти годинникової стрілки); якщо список міститиме менше ніж три вершини, то нічого не буде зображено, якщо кількість заданих вершин не буде кратна трьом, то останні одна чи дві точки не буде враховано;
- **GL_TRIANGLE_STRIP** – поєднання (смуга) трикутників; у всіх послідовних трикутників є одна спільна сторона з попереднім, тому треба переконатися, що список вершин утворено так, щоб зображення було реальним; кожна вершина зі списку, яка йде після перших двох, задає ще один трикутник; отже, перші три вершини потрібно вказувати проти годин-

никової стрілки, якщо дивитися на передню (зовнішню) поверхню трикутника, вони задають перший трикутник; третя, друга і четверта вершини – другий трикутник і т.д.; вершини N-1, N-2 та N визначають N-2-й трикутник; ніякі вершини не можуть повторюватись; з N вершин, що не повторюються, отримують N–2 трикутники;

- **GL_TRIANGLE_FAN** – метод віяла; всі трикутники мають одну спільну вершину, тобто перша, друга і третя вершина задають перший трикутник, перша, третя і четверта вершини задають другий трикутник і т.д., вершини 1, N-1, N визначають N-й трикутник; з N вершин, що не повторюються, отримують також N–2 трикутники;
- **GL_QUADS** – кожні чотири задані командою **glVertex** точки визначають чотирикутник: перші чотири – перший, наступні чотири – другий і т.д.; отримаємо набір *не з'єднаних* між собою чотирикутників, якщо жодні вершини не повторюватимуться; для кожного чотирикутника вершини задано проти годинникової стрілки); якщо список міститиме менше чотирьох вершин, то нічого не буде зображено, якщо кількість заданих вершин не буде кратна чотирьом, то останні одна, дві чи три точки не буде враховано;
- **GL_QUAD_STRIP** – поєднання (смуга) чотирикутників: чотирикутник задається для кожної пари вершин, яка йде після перших двох у списку, тому вершини треба перераховувати так, щоб кожен чотирикутник мав правильний порядок (проти годинникової стрілки), тобто перша, друга, третя і четверта вершини визначають перший чотирикутник; третя, четверта, третя, шоста і п'ята вершини – другий чотирикутник і т.д., вершини N–3, N-2, N-1 і N визначають N/2–1-й чотирикутник; з N вершин отримують N/2–1 чотирикутники, за умови, що N більше або дорівнює 4; якщо N не кратне 4, то зайвих координат у списку враховано не буде.

Приклад зображення опуклого шестикутника, коли координати точок задано у вигляді *масивів двох цілих чисел*:

```
glBegin(GL_POLYGON);
   glVertex2iv(p1);
   glVertex2iv(p2);
   glVertex2iv(p3);
   glVertex2iv(p4);
   glVertex2iv(p5);
   glVertex2iv(p6);
glEnd();
```

Незважаючи на те, що коренева бібліотека OpenGL відображає тільки опуклі багатокутники, бібліотека GLU пропонує функції, що дають змогу працювати з увігнутими багатокутниками та іншими неопуклими об'єктами з лінійними межами. Є набір стандартних процедур бібліотеки GLU **для мозаїчного подання багатокутників**, які дають можливість перетворювати такі фігури на набір

трикутників, трикутні сітки, віяла трикутників і прямолінійні відрізки. Розклавши ці об'єкти на складові, їх опрацьовують за допомогою основних функцій OpenGL.

Оскільки графічні зображення дуже часто містять **зафарбовані прямокутники**, пакет OpenGL має спеціальну функцію, яка працює з вершинами в площині *xy*:

```
glRect*(x1, y1, x2, y2);
```

тут **(x1, y1)**, **(x2, y2)** – координати протилежних кутів прямокутника; індексні коди позначають тип даних (**i, s, f, d**) і те, чи подано координати як елементи масиву; сторони прямокутника паралельні до координатних осей площини *xy*.

У деяких версіях OpenGL наведена раніше процедура може виявитися ефективнішою, ніж побудова зафарбовуваного прямокутника за допомогою функції **glVertex**. Якщо прямокутник зображають функцією **glRect**, то його сторони рисують між вершинами в такій послідовності: **(x1, y1), (x2, y1), (x2, y2), (x1, y2)**, і знову до першої вершини, тобто список вершин формують за годинниковою стрілкою.

У багатьох двовимірних застосуваннях відмінність між передньою і задньою поверхнею прямокутника несуттєва. Якщо ж треба приписати цим поверхням різні властивості, то змінюють послідовність двох вершин, щоб вони розміщувались проти годинникової стрілки.

Приклади зображення квадрата, коли координати двох його протилежних вершин задано *в явному вигляді* цілими *числами та масивами двох цілих чисел*:

```
glRecti(200, 100, 50, 250);
```

або як масив даних

```
int ver1[] = {200, 100};
int ver2[] = {50, 250};
glRectiv(ver1, ver2);
```

2.1.4. Створення графічних об'єктів за допомогою масиву вершин

Якщо опис сцени складається з багатьох зафарбованих багатокутних поверхонь, можна раціонально створити зображення за допомогою **масиву вершин,** що дає змогу впорядкувати інформацію. Щоб його використати, необхідні такі кроки.

1. Викликати функцію **glEnableClientState(GL_VERTEX_ARRAY)**, щоб активізувати можливість створення масиву вершин в OpenGL.
2. Скористатись функцією **glVertexPointer** для задання розташування і формату даних координат вершин.
3. Зобразити сцену за допомогою стандартної процедури **glDrawElements**, яка може опрацьовувати декілька примітивів за допомогою невеликої кількості її викликів.

2.1.4.1. *Приклад* зображення одиничного куба

Спочатку визначимо тип даних координат точки у тривимірному просторі, а потім задамо координати кожної вершини як елементи масиву з одним індексом.

```
glEnableClientState(GL_VERTEX_ARRAY);
typedef GLint vertex3[3];
vertex3 pt[8] = {{0, 0, 0}, {0, 1, 0}, {1, 0, 0}, {1, 1, 0},
   {0, 0, 1}, {0, 1, 1}, {1, 0, 1}, {1, 1, 1}};
GLubyte vertIndex[] = {6, 2, 3, 7, 5, 1, 0, 4, 7, 3, 1, 5,
   4, 0, 2, 6, 2, 0, 1, 3, 7, 5, 4, 6};
glVertexPointer(3, GL_INT, 0, pt);
glDrawElements(GL_QUADS, 24, GL_UNSIGNED_BYTE, vertIndex);
```

На рис. 2.1 показано індекси масиву **pt**, які відповідають координатам вершин куба.

За допомогою першої команди **glEnableClientState(GL_VERTEX_ARRAY)** активізують можливість створення масиву вершин (у цьому випадку) з боку клієнта системи клієнт-сервер. Оскільки дані про рисунок зберігаються у клієнта (на комп'ютері, що запускає головну програму), то масив вершин теж має знаходитися там. Сервер (наприклад, робоча станція) віддає команди і виводить рисунок на екран. Зрозуміло, що той самий комп'ютер може бути одночасно і клієнтом, і сервером.

Відключити вказану команду в разі потреби можна командою

```
glDisableClientState(GL_VERTEX_ARRAY);
```

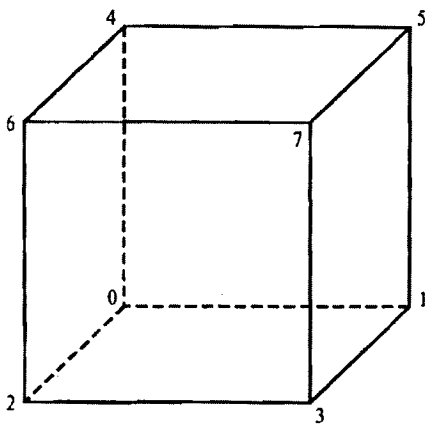


Рис. 2.1. Значення індексів масиву ***pt****,*
які відповідають координатам вершин одиничного куба

За допомогою функції **glVertexPointer(3, GL_INT, 0, pt)** задають розміщення і формат координат вершин об'єкта. Її параметри мають значення:

- кількість координат, які задають для кожної вершини;

- тип даних для координат вершин: окрім заданого **GL_INT**, може бути **GL_BYTE**, **GL_SHORT**, **GL_FLOAT**, **GL_DOUBLE**;
- зміщення (в байтах) між сусідніми вершинами, що дає змогу використати в одному масиві різні типи інформації, наприклад, координати і колір; якщо задано 0, то інформація тільки про координати;
- масив вершин, у якому містяться значення координат.

Усі індекси вершин куба записують у масив **vertIndex**; вони є індексами масиву **pt**. Цей список індексів є останнім параметром функції **glDrawElements**, а потім він же буде використаний примітивом **GL_QUADS** як перший параметр для зображення послідовності чотирикутних поверхонь куба.

Параметри команди **glDrawElements(GL_QUADS, 24, GL_UNSIGNED_BYTE, vertIndex)** мають значення:

- символьна константа, що задає тип примітивів для зображення їхнього набору: тут **GL_QUADS** визначає побудову чотирикутника за 4 вершинами;
- кількість елементів у масиві **vertIndex**; функція зображає грані куба за 4 вершинами, поки не опрацює всі 24 вершини; отож із 6 чотирикутників буде складено поверхню куба;
- тип значень індексів у **vertIndex**: **GL_UNSIGNED_BYTE** (якщо це невеликі цілі числа, під які можна виділити один байт), **GL_UNSIGNED_SHORT** та **GL_UNSIGNED_INT**;
- список індексів вершин куба, тобто номерів точок у масиві **pt** (від 0 до 7), які послідовно по 4 значення описують грані куба; у загальному випадку тут можна ще задавати коди кольорів та інші параметри об'єкта.

Щоб спростити опрацювання опису сцени, у масив вершин, крім значень координат, можна ввести деяку додаткову інформацію: задати коди кольору та інші параметри об'єкта. Доступ до них можна отримати за допомогою функції **glDrawElements**. Для підвищення ефективності можна також об'єднувати різні масиви, а способи реалізації таких масивів параметрів розглянемо в наступному розділі.

2.2. Функції опису додаткових примітивів OpenGL

До **додаткових** примітивів, засоби для створення яких є в графічних пакетах, належать зображення фігур, заданих за допомогою **прямокутного масиву кодів кольору**. Прямокутну сітку узору можна отримати методом оцифрування (сканування) фотографії чи іншого рисунка або створенням фігури за допомогою графічної програми. Після цього кожен код кольору в масиві присвоюють відповідним пікселям на екрані. Піксельний масив кодів кольору називають *піксельним зображенням*.

До параметрів піксельного масиву можна зарахувати посилання на матрицю кольорів, а також розташування і розмір області екрана, зайнятої кодами кольору.

Ще один тип піксельного масиву отримують, коли кожному елементу матриці присвоюють значення 1 або 0. Тоді масив є *бітовим відображенням*, яке інколи називають *маскою*, маючи на увазі, що пікселю присвоєно (чи об'єднано з ним) або не присвоєно певний колір.

До **додаткових** примітивів належать і **рядки символів**, які використовують для надписів на рисунках і графіках.

2.2.1. Функції піксельних масивів

У пакеті OpenGL є дві функції, які можна використати для опису фігур чи зразків, заданих прямокутним масивом, – **бітове** і **піксельне відображення**, а також стандартні процедури збереження, копіювання і різних операцій з масивами піксельних значень.

2.2.1.1. Функція бітового відображення

Масив бінарного зразка задають за допомогою функції

```
glBitmap(width, height, x0, y0, xOffset, yOffset, bitShape);
```

- параметри **width** і **height** – кількість стовпців і рядків у масиві **bitShape**;
- **x0**, **y0** – точка, яку розглядають як початок координат прямокутного масиву, яку задають щодо лівого нижнього кута масиву **bitShape**; значення можуть бути додатними і від'ємними;
- **xOffset**, **yOffset** – зміщення координат для оновлення *поточного растрового розміщення* в буфері кадру (це розміщення буфера кадру, з якого починається накладання даного зразка); зображення бітового масиву полягає в перенесенні його початку координат (**x0**, **y0**) у поточне растрове розміщення, а після цього значення, присвоєні зазначеним параметрам, використовують як зміщення координат;
- **bitShape** – масив бінарного зразка, значення кожного елемента якого **1** (відповідний піксель треба зобразити наперед заданим кольором або накласти на код кольору, записаного в буфері кадру) або **0** (значення пікселя не змінюється).

Значення координат `x0, y0, xOffset, yOffset`, а також поточне растрове розміщення зберігають як числа з плаваючою крапкою (хоч бітове відображення застосовують у пікселях з цілочисловими координатами), бо це дає змогу розміщувати бітове відображення через довільні інтервали, що особливо корисно у таких застосуваннях, як формування рядків символів за допомогою бітових зразків.

Значення одновимірного масиву **bitShape** задають за рядками, починаючи з нижнього краю прямокутної сітки зразка. Кожен рядок масиву записують не побітово, а за допомогою 8-бітових чисел без знаку (масив **bitShape** у прототипі функції **glBitmap** має тип **GLubyte**).

Щоб задати координати *поточного растрового розміщення*, застосовують процедуру

```
glRasterPos*();
```

параметри й індекси аналогічні тим, що використовують у функції **glVertex**; кольором бітового відображення є поточний колір у момент виклику цієї команди. Будь-які наступні зміни кольору не впливають на бітовий масив.

Режим збереження бітового зображення встановлюють процедурою

```
glPixelStorei(GL_UNPACK_ALIGMENT, val);
```

val може набувати значень **1, 2, 4, 8**; наприклад, **1** означає, що задані значення бінарного зразка треба розміщувати на межах окремих байтів.

Кожен рядок прямокутного бітового масиву записують числами, розмір яких кратний 8 бітам, де бінарні дані організовані як 8-бітові символи без знаку. Однак фігуру також можна описати за допомогою сітки будь-якого зручного розміру. На рис. 2.2 показано бітовий узор, заданий сіткою з 10 рядків і 9 стовпців, де бінарні дані задано 16 бітами в кожному рядку. Коли цей узор накладають на пікселі в буфері кадру, всі значення бітів за межами 9-го стовпця ігнорують.

```
00001000000   0x08 0x00
00011100000   0x1C 0x00
00111110000   0x3E 0x00
01111111000   0x7F 0x00
11111111100   0xFF 0x80
00011100000   0x1C 0x00
00011100000   0x1C 0x00
00011100000   0x1C 0x00
00011100000   0x1C 0x00
```

Рис. 2.2. Бітовий узор

Щоб накласти бітовий узор, зображений на рис. 2.2, на певну ділянку буфера кадру, скористаємось таким кодом:

```
GLubyte bitShape[20]=
  0x1c, 0x00, 0x1c, 0x00, 0x1c, 0x00, 0x1c, 0x00, 0x1c, 0x00,
  0xff, 0x80, 0x7f, 0x00, 0x3e, 0x00, 0x1c, 0x00, 0x08, 0x00;
glPixelStorei(GL_UNPACK_ALIGMENT, 1);
/* Встановлюємо режим збереження пікселів */
glRasterPos2i(30, 40);
glBitmap(9, 10, 0.0, 0.0, 20, 15, bitShape);
```

Значення масиву задають порядково, починаючи з нижнього краю прямокутної сітки узору. Потім встановлюють режим збереження бітового відображення за допомогою стандартної процедури **glPixelStore**. Значення 1 у цій функції

вказує на те, що задані значення треба розміщувати на межах окремих байтів. За допомогою функції **glRasterPos** встановлюємо поточне растрове розміщення з координатами (30, 40). Нарешті, функція **glBitmap** вказує, що бітовий узор задаємо масивом **bitShape**, і що цей масив складається з 9 стовпців і 10 рядків. Координати точки відліку цього узору – (0.0, 0.0), тобто вона розташована в лівому нижньому куті сітки. Задане у цьому прикладі зміщення координат на величину (20.0, 15.0) ніяк не використано.

2.2.1.2. Функція піксельного відображення

Піксельне відображення, на відміну від бітового, містить інформацію про колір кожного пікселя. Робота з піксельними масивами подібна до роботи з бітовими.

Зразок, визначений як масив значень кольору, **накладають на блок пікселів** у буфері кадру за допомогою функції

```
glDrawPixels(width, height, dataFormat, dataType, pixMap);
```

- параметри **width** і **height** – кількість стовпців і рядків масиву **pixMap**;
- **dataFormat** – константа, що показує, як задають і що означають елементи масиву; наприклад, **GL_BLUE** (всі пікселі будуть синього кольору), **GL_BGR** (задано порядок кольорів, відмінний від стандартного, – синій, зелений, червоний), **GL_DEPTH_COMPONENT** (значення масиву накладатимуться на буфер глибини), **GL_STENCIL_INDEX** (робота з буфером шаблонів);
- **dataType** (значення **GL_BYTE**, **GL_INT**, **GL_FLOAT**) визначає тип даних кодів кольору масиву; лівий нижній кут цього масиву відображається з поточного растрового розміщення, заданого за допомогою функції **glRasterPos**.

Приклад побудови піксельного відображення, заданого як масив кодів кольору в масиві RGB розміром 128 на 128:

```
glDrawPixels(128, 128, GL_RGB, GL_UNSIGNED_BYTE,
colorShape);
```

Оскільки OpenGL дає змогу використовувати декілька буферів, масив кодів кольору можна вставити в окремий буфер, зазначивши його у процедурі **glDrawPixels**. В одних буферах записують коди кольору, в інших, зокрема *буфері глибини* – відстань від точки спостереження до об'єкта (глибину), в *буфері шаблонів* зберігають зразки, утворені межами об'єктів сцени.

Пакет OpenGL має *чотири буфери кольору*, які можна використати для оновлення екрана. Два буфери кольору утворюють ліво-праву сцени під час створення стереоскопічних зображень. Для кожного стереоскопічного буфера є передньо-задня пара, яку використовують в анімації з подвійним буфером. Якщо не

підтримуються ні стереоскопічні ефекти, ні подвійна буферизація, то є лише один буфер регенерації, який спроектований як *передній лівий буфер кольору*. Крім того, підтримується низка допоміжних буферів кольору, які використовують, наприклад, для збереження рисунка, який пізніше буде копіюватися в буфер регенерації для створення зображення.

Один колірний чи допоміжний *буфер*, а також комбінацію буферів кольору для збереження піксельного відображення *вибирають* за допомогою команди:

```
glDrawBuffer(buffer);
```

buffer – символьна константа для опису одного чи декількох робочих буферів:

- **GL_FRONT_LEFT, GL_FRONT_RIGHT, GL_BACK_LEFT, GL_BACK_RIGHT, LEFT** – вибір одного буфера;
- **GL_FRONT, GL_BACK** – вибір обох передніх або обох задніх буферів; при цьому допускається, що стереоскопічна візуалізація активована;
- **GL_LEFT, GL_RIGHT** – вибір лівої або правої пари буферів;
- **GL_FRONT_AND_BACK** – вибір всіх доступних буферів;
- **GL_AUXk** – вибір допоміжного буфера, **k** – ціле число від 0 до 3, хоч в деяких версіях OpenGL може бути більше ніж чотири допоміжні буфери.

Крім записування масиву значень пікселів у буфер, можна зчитувати блок значень з буфера чи копіювати такий блок в іншу область буфера.

2.2.1.3. Растрові операції

Крім записування масиву значень пікселів у буфер, можна зчитувати блок значень з буфера чи копіювати такий блок в іншу область буфера, а також виконувати низку інших операцій з піксельним масивом. Поняття *растрова операція* означає будь-яку функцію, яку застосовують для обробки піксельного масиву. Растрову операцію, за якої масив значень пікселів переміщають з одного місця в інше, називають ще *перенесенням блоку* значень пікселів. Для дворівневої системи (чорно-білого зображення) таку операцію називають *перенесенням бітового рядка*, особливо якщо такі функції реалізовано на апаратному рівні. Для багаторівневої системи (кольорового зображення) перенесення блоків інколи називають *перенесенням піксельного рядка*.

Для **вибору прямокутного блоку пікселів** у заданому наборі буферів використовують функцію

```
glReadPixels(xmin, ymin, width, height, dataFormat,
dataType, array);
```

- (**xmin**, **ymin**) – координати точки екрана, де розташовано лівий нижній кут блоку;

- **array** – масив, куди буде поміщено результат операції; тип інформації залежить від вибраного за допомогою параметра **dataFormat** буфера;
- інші параметри такі, як і для процедури **glDrawPixels**.

Особливу комбінацію буферів кольору чи додатковий буфер вибирають для використання у процедурі **glReadPixels** за допомогою функції

```
glReadBuffer(buffer);
```

buffer – символьна константа для опису одного чи декількох буферів, не відрізняється від констант процедури **glDrawBuffer**, за винятком того, що не можна вибрати всі чотири буфери кольору.

За замовчуванням вибирають передню ліво-праву пару або лише передній правий буфер, залежно від стану стереоскопічної візуалізації.

Крім того, блок заданих пікселів можна **скопіювати** з одного місця в інше в межах набору буферів OpenGL за допомогою процедури:

```
glCopyPixels(xmin, ymin, width, height, pixelValues);
```

- **width** і **height** – додатні цілочислові значення, які позначають відповідно кількість стовпців і рядків, які треба скопіювати;
- параметру **pixelValues** присвоюють значення, що вказує на тип інформації, яку потрібно скопіювати: коди кольору, значення глибини чи шаблонів;
- інші параметри такі, як у процедурі **glReadPixels**.

Отже, блок значень пікселів копіюють з *буфера-джерела*, вибраного командою **glReadBuffer**, у *буфер призначення*, заданий командою **glDrawBuffer**; при цьому його нижній лівий кут переноситься в поточне растрове розміщення, обидві області повинні повністю лежати в межах координат екрана.

Для досягнення різних ефектів під час розміщення пікселів у буфер за допомогою команд **glDrawPixels** і **glCopyPixels** можна об'єднувати нові значення зі старими, зокрема, логічними операціями **AND** (*і*), **OR** (*або*) та **XOR** (*виключне або*). У пакеті OpenGL для вибору побітової логічної операції для **об'єднання** вхідних (нових) кодів кольору зі значеннями пікселів призначення (старими) використовують функції:

```
glEnable(GL_COLOR_LOGIC_OP);
glLogicOp(logicOp);
```

параметру **logicOp** можна присвоїти значення з набору символьних констант, зокрема **GL_AND, GL_OR** або **GL_XOR**.

Крім того, або вхідні значення бітів, або значення бітів призначення можна задавати в інвертованій формі (значення 0 і 1 поміняно місцями). Константу **GL_COPY_INVERTED** використовують у функції **glLogicOp** для того, щоб перетворити інвертовані вхідні значення бітів на правильні, а потім замінити

значення бітів призначення інвертованими значеннями вхідних бітів. За допомогою константи **GL_INVERT** можна також лише інвертувати значення бітів призначення без подальшої заміни їх вхідними значеннями. Різні операції інверсії також можна об'єднувати з логічними операціями **AND**, **OR** та **XOR**. Іншими можливостями можна вважати очищення всіх бітів призначення до значення 0 (**GL_CLEAR**) чи присвоєння всім бітам призначення значення 1 (**GL_SET**). За замовчуванням значення параметра **logicOp** є **GL_COPY**, за якого значення бітів призначення заміняють вхідними.

Існують також інші процедури для виконання різних дій з піксельними масивами, які опрацьовують за допомогою функцій **glDrawPixels**, **glReadPixels** і **glCopyPixels**. Наприклад, процедури **glPixelTransfer** і **glPixelMap** можна використовувати для зміщення чи вирівнювання кодів кольору, значень глибини чи значень шаблону. До операцій з пікселями ще повернемось у наступних розділах.

2.2.2. Функції зображення символів

Символи шрифтів можна сформувати з інших примітивів, тому в складі основної бібліотеки OpenGL немає спеціального примітиву для тексту. Явно будь-який символ можна задати бітовим масивом, як у прикладі, показаному на рис. 2.2, і записати набір растрових символів як список шрифту. Після цього рядок символів зображують – вибрану послідовність бітових масивів зі списку шрифту переносять на відповідні місця в буфері кадру.

Однак GLUT містить декілька визначених наборів символів та стандартні процедури для відображення **растрових** та **ескізних шрифтів**.

Символи растрового шрифту визначені на прямокутній області і є *блоками бітів*. Кожен блок задає певний символ як образ нулів та одиниць, які відповідають засвіченим і незасвіченим точкам растру. Під час відображення символ поміщають у буфер кадру за допомогою операції *побітового перенесення*, яка виконується дуже швидко. Збільшити розмір символів можна тільки дублюванням пікселів, що часто дає доволі "грубу" форму, геометричні перетворення виконувати немає сенсу.

Растровий символ візуалізують за допомогою функції

```
glutBitmapCharacter(font, character);
```

- **font** – значення символьної константи GLUT, яка вказує на певний набір накреслень:

  шрифт із постійною шириною символів (моноширинний) –

```
GLUT_BITMAP_8_BY_13,GLUT_BITMAP_9_BY_15;
```

  шрифт із пропорційними проміжками (пропорційний) –

```
GLUT_BITMAP_TIMES_ROMAN_10(12);
GLUT_BITMAP_HELVETICA_10(12, 18);
```

- **character** – код ASCII або окремий символ, який потрібно зобразити, наприклад, код 65 або 'A'.

Кожен символ, створений за допомогою зазначеної функції, зображується так, що початок координат (лівий нижній кут) бітового масиву розташований у поточному растровому розміщенні. Після того, як бітовий масив символу завантажується в буфер регенерації, до координати x поточного растрового розміщення додається зміщення, що дорівнює ширині символу. Символи зображають тим кольором, який був заданий до виконання цієї функції.

Приклад зображення текстового рядка, що містить 36 растрових символів:

```
glRasterPos2i(x, y);
for(k=0; k<36; k++)
   glutBitmapCharacter(GLUT_BITMAP_9_BY_15, text[k]);
```

символи зображаються тим кольором, який був заданий до виконання функції **glutBitmapCharacter**.

Ескізний (штриховий) шрифт сформовано за тим самим принципом, що й інші графічні примітиви. Накреслення символу визначають вершинами відповідних прямолінійних і криволінійних відрізків. Розмір і положення цих символів можна контролювати за допомогою операцій перетворень (розділ 4), які викликають перед виконанням процедури візуалізації символів; їх можна без будь-яких спотворень стискати чи розтягувати, оскільки вони складаються з геометричних ліній. Однак вони візуалізуються повільніше, ніж растрові шрифти.

Ескізний символ відображають за допомогою виклику функції

```
glutStrokeCharacter(font, character);
```

перший параметр може мати значення:

- моноширинний шрифт – **GLUT_STROKE_MONO_ROMAN**,
- пропорційний – **GLUT_STROKE_ROMAN**.

2.2.3. Опис криволінійних об'єктів

У більшості графічних систем використовують **два підходи** до опису криволінійних об'єктів:

- використовують раніше визначені примітиви для **апроксимації** кривих і поверхонь; наприклад, круг апроксимують правильним багатокутником, сферу – правильним багатогранником;
- створюють **математичний опис** криволінійних об'єктів (квадратичні поверхні, параметричні поліноміальні криві), а потім розробляють графічні функції реалізації цього опису; наприклад, сферу будують за двома точками: центром і довільною точкою на її поверхні, кубічну поліноміальну криву – за 4-ма точками.

2.3. Контрольні питання

1. Які примітиви належать до базових?
2. Як зображують точки в OpenGL?

3. Які є способи формування відрізків в OpenGL?
4. Які є способи створення зафарбованих багатокутників в OpenGL?
5. Якою функцією можна описати прямокутник?
6. За допомогою чого можна раціонально створити зображення у випадку опису багатьох багатокутних поверхонь?
7. Які примітиви належать до додаткових?
8. Якою функцією задають масив бінарного зразка?
9. Яку процедуру застосовують для задання поточного растрового розміщення?
10. Якою функцією задають режим збереження бітового зображення?
11. Яку функцію використовують для накладання масиву значень кольору на блок пікселів?
12. Які растрові операції ви знаєте?
13. Порівняйте переваги і недоліки растрових і ескізних шрифтів.
14. За допомогою якої функції отримують растровий символ?
15. За допомогою якої функції отримують ескізний символ?
16. Скільки буферів кольору має пакет OpenGL?
17. Які є підходи до опису криволінійних об'єктів?

2.4. Приклади тестових питань

1. Для формування замкнутої ламаної лінії в OpenGL використовують тип:

   а) **GL_LINE_STRIP**;
   б) **GL_LINE_LOOP**;
   в) **GL_LINES**.

2. Для формування відкритої ламаної лінії в OpenGL використовують тип:

   а) **GL_LINE_STRIP**;
   б) **GL_LINE_LOOP**;
   в) **GL_LINES**.

3. Для формування окремих відрізків у OpenGL використовують тип:

   а) **GL_LINE_STRIP**;
   б) **GL_LINE_LOOP**;
   в) **GL_LINES**.

4. Для формування окремих трикутників в OpenGL використовують символьну константу:

   а) **GL_TRIANGLES**;
   б) **GL_TRIANGLE_STRIP**;
   в) **GL_TRIANGLE_FAN**.

5. Для формування віяла трикутників в OpenGL використовують символьну константу:

   а) **GL_TRIANGLES**;
   б) **GL_TRIANGLE_STRIP**;
   в) **GL_TRIANGLE_FAN**.

6. Для формування смуги трикутників в OpenGL використовують символьну константу:

   а) **GL_TRIANGLES**;
   б) **GL_TRIANGLE_STRIP**;
   в) **GL_TRIANGLE_FAN**.

7. Для формування окремих чотирикутників в OpenGL використовують символьну константу:

   а) **GL_QUADS**;
   б) **GL_QUAD_STRIP**;
   в) **GL_QUAD_FAN**.

8. Для формування смуги чотирикутників в OpenGL використовують символьну константу:

   а) **GL_QUADS**;
   б) **GL_QUAD_STRIP**;
   в) **GL_QUAD_FAN**.

9. Швидкість візуалізації ескізних (штрихових) шрифтів порівняно з растровими шрифтами є:

   а) більшою;
   б) меншою;
   в) однаковою.

Розділ 3

ФУНКЦІЇ ЗАДАННЯ АТРИБУТІВ В OpenGL

Параметр, який впливає на спосіб зображення примітиву, називають **атрибутом**. Спочатку зупинимось на атрибутах, які призначені для керування основними властивостями зображення графічних примітивів і визначають їхні фундаментальні характеристики, – на **кольорі** та **розмірі**.

Графічну систему, у якій зберігається список поточних значень атрибутів та інших параметрів, називають *системою станів* або *апаратом станів*. Атрибути результуючих примітивів та деякі інші параметри, такі як поточне розміщення буфера, називають *змінними стану* чи *параметрами стану*. Після присвоєння значення одному чи декільком таким параметрам система входить у визначений ними стан, в якому залишається доти, поки значення параметрів стану не зміняться.

3.1. Налаштування кольору в OpenGL

Задаючи певний набір колірних значень примітивів, визначають *колірний стан* пакета OpenGL. Поточний колір буде застосовано до всіх описаних нижче примітивів аж до наступної зміни колірних параметрів.

У колірних растрових системах кількість можливих варіантів кольорів залежить від обсягу пам'яті, виділеного кожному пікселю в буфері кадру. Інформація про колір може бути записана в буфер кадру двома способами: безпосередньо або в окрему таблицю з кодів кольору, тоді координати пікселів використовують для запису значень індексів. Мінімальну кількість кольорів за першим способом можна одержати для 3 бітів пам'яті на один піксель.

Маючи 6 бітів на один піксель, можна встановити 4 різні рівні інтенсивності для кожної з трьох колірних компонент і забезпечити для кожного пікселя на екрані 64 колірні опції, що збільшить обсяг пам'яті, необхідний для буфера кадру. Для роздільної здатності 1024 на 1024 пікселів для буфера кадру у повноколірній (24 біти на піксель) RGB-системі необхідно 3 Мб пам'яті.

Коди кольору, записані в буфер кадру				Результуючий колір
код кольору	червоний	зелений	синій	
0	0	0	0	чорний
1	0	0	1	синій
2	0	1	0	зелений
3	0	1	1	голубий
4	1	0	0	червоний
5	1	0	1	пурпуровий
6	1	1	0	жовтий
7	1	1	1	білий

3.1.1. Колірні режими

Більшість налаштувань кольорів примітивів OpenGL виконують у *режимі RGB*, чотиривимірний опис кольору називають *RGBA-кольором*. Четвертий параметр – *коефіцієнт альфа* – можна використовувати для змішування кольорів під час накладання об'єктів один на один. Одна з важливих галузей його застосування – це моделювання ефектів прозорості: коли коефіцієнт дорівнює 0 – об'єкт повністю прозорий, дорівнює 1 – абсолютно непрозорий.

У режимі RGB чи RGBA поточні колірні компоненти задають за допомогою функції

```
glColor*(colorComponents);
```

індекси аналогічні введеним під час опису функції **glVertex()**, причому перший (розмірність) може мати значення лише 3 або 4.

Приклади:

- **glColor3f(1.0, 0.0, 0.0);** – колір об'єкта червоний,
- **glColor3f(0.0, 1.0, 1.0);** або **glColor3i(0, 255, 255);** – голубий для повноколірної системи, в якій виділено 24 біти на піксель (або 8 бітів, що відповідає 256 рівням, для кожної колірної компоненти);
- **glColor3fv(colorArray);** – колір визначають за допомогою масиву.

Кольори можна задавати і для окремих точок, використовуючи для цього пару **glBegin/glEnd**.

Насправді у позиціях буфера кадру зберігаються цілочислові значення, тому задані в будь-якому іншому форматі колірні значення перетворюються на цілочислові, які належать діапазону, визначеному кількістю бітів, доступних системі.

Опис кольору можна також задавати в *індексному колірному режимі*, в якому дають посилання на значення елементів колірної таблиці. У цьому режимі колір задають через значення індексу колірної таблиці:

```
glIndex*(colorIndex);
```

параметр **colorIndex** – ціле невід'ємне число, тип даних: **ub** (байтів без знака), **s**, **i**, **d**, **f**.

Кількість індексів у колірній таблиці завжди дорівнює числу 2 в степені, що дорівнює кількості бітів, виділених для зберігання одного кольору. Наприклад, індексну палітру з 256 8-бітових кольорів широко використовують для зображень в Інтернеті. У кореневій бібліотеці немає функцій для занесення значень до колірної пошукової таблиці, оскільки стандартні процедури для обробки таблиць є частиною системи вікон. Крім того, деякі системи вікон підтримують декілька таблиць кольорів, а інші мають лише одну з обмеженим вибором кольорів.

Однак є стандартна процедура бібліотеки GLUT, яка взаємодіє з системою вікон і заносить опис кольорів до таблиці в позицію із заданим індексом:

```
glutSetColor(colorIndex, red, green, blue);
```

Колірним параметрам **red**, **green**, **blue** присвоюють значення з плаваючою крапкою в діапазоні від 0.0 до 1.0, потім цей колір буде записано до таблиці у позицію, задану параметром **colorIndex**.

Стандартні процедури для обробки трьох інших колірних таблиць пропонують як доповнення до кореневої бібліотеки. Ці стандартні процедури є частиною *набору для створення зображення*. Декілька прикладів використання цих таблиць – встановлення ефекту фокусування камери, фільтрація визначених кольорів на зображенні, збільшення визначених інтенсивностей чи регулювання рівня яскравості, перетворення чорно-білих фотографій на кольорові й усунення контурних нерівностей. Крім того, ці таблиці можна використати для зміни колірних моделей, тобто можна змінити кольори RGB на інший опис за допомогою трьох інших "основних" кольорів (голубий, пурпуровий і жовтий).

Окрему колірну таблицю з набору для створення зображень активізують функцією **glEnable** з використанням однієї з трьох назв таблиць: **GL_COLOR_TABLE, GL_POST_CONVOLUTION_COLOR_TABLE** або **GL_POST_COLOR_MATRIX_COLOR_TABLE**. Потім можна скористатися стандартними процедурами з набору для створення зображень і вибрати певну таблицю кольорів, задати значення таблиці кольорів, скопіювати значення з таблиці чи вказати, яку компоненту кольору пікселя треба змінити і як саме це треба робити.

3.1.2. Змішування кольорів

Часто було б зручно поєднувати кольори накладених один на одного об'єктів або змішувати колір об'єкта з кольором фону. У більшості графічних пакетів пропонують різні способи створення ефектів змішування кольорів. Ці процедури називають *функціями змішування кольорів* або *функціями створення зображень*. У пакеті OpenGL кольори двох об'єктів можна змішати, спочатку завантаживши у буфер кадру колір одного об'єкта, а потім об'єднавши колір другого з кольором з буфера кадру. Поточний колір у буфері кадру називають *кольором приймача*, а колір другого об'єкта – *кольором джерела*. Змішування можна виконати лише в

режимі RGB чи RGBA. Щоб застосувати змішування кольорів, спочатку треба активувати цю можливість за допомогою функції

```
glEnable(GL_BLEND);
```

Щоб відключити стандартні процедури змішування кольорів, використовують функцію

```
glDisable(GL_BLEND);
```

Якщо можливість змішування кольорів не активовано, колір об'єкта просто замінить колір, записаний у буфері кадру в розміщенні цього об'єкта.

Кольори можна змішувати різними способами, залежно від ефекту, якого треба досягти, а різні колірні ефекти виникають після задання двох наборів коефіцієнтів змішування: для поточного об'єкта в буфері кадру (приймач) та нового (джерело). Новий змішаний колір, який потім завантажується в буфер кадру, знаходять так:

$$(S_r R_s + D_r R_d, S_g G_s + D_g G_d, S_b R_s + D_b B_d, S_a A_s + D_a A_d),$$

де колірні компоненти RGBA джерела та приймача – (R_s, G_s, B_s, A_s) та (R_d, G_d, B_d, A_d) відповідно, коефіцієнти змішування джерела та приймача – (S_r, S_g, S_b, S_a) та D_r, D_g, D_b, D_a) відповідно. Знайдені значення компонент комбінованого кольору мають потрапляти в діапазон від 0.0 до 1.0, тому будь-якій сумі, що перевищує 1.0, буде присвоєно значення 1.0, а будь-якій сумі, меншій за 1.0, – значення 0.0.

Значення коефіцієнтів змішування задають за допомогою функції

```
glBlendFunc(source_factor, destination_factor);
```

Кожному з параметрів **source_factor**, **destination_factor**, тобто коефіцієнтам джерела та приймача, присвоюють символьні константи OpenGL, які задають визначений набір із чотирьох коефіцієнтів змішування. Наприклад, константи **GL_ZERO, GL_ONE** дають коефіцієнти змішування (0.0, 0.0, 0.0, 0.0), (1.0, 1.0, 1.0, 1.0) відповідно. Можна присвоїти всім чотирьом коефіцієнтам змішування значення альфа або приймача, або джерела, що роблять за допомогою констант **GL_DST_ALPHA**, **GL_SRC_ALPHA**. До решти констант належать **GL_ONE_MINUS_DST_ALPHA**, **GL_ONE_MINUS_SRC_ALPHA**, **GL_DST_COLOR** і **GL_SRC_COLOR**. Ці коефіцієнти змішування часто використовують для моделювання прозорості, розглянемо їх детально у підрозділі 9.6.5. За замовчуванням параметрам **source_factor** та **destination_factor** присвоюють значення **GL_ONE** та **GL_ZERO** відповідно, отже, нові колірні значення замінюють поточні значення в буфері кадру.

До бібліотеки GLUT введено додаткові функції, зокрема процедуру встановлення змішаного кольору і процедуру задання рівняння змішування.

3.1.3. Колірні масиви

Значення кольору для опису сцени можна задавати разом з координатними значеннями в масиві вершин. Це можна зробити або в режимі RGB, або в індексному колірному режимі. Спочатку треба активувати можливість створення колірного масиву

```
glEnableClientState(GL_COLOR_ARRAY);
```

Потім для колірних режимів RGB чи RGBA задають положення і формат колірних компонент за допомогою функції

```
glColorPointer(nColorComponents, dataType, offset,
colorArray);
```

- параметру **nColorComponents** присвоюють значення 3 або 4 залежно від того, які компоненти заносять у масив **colorArray** – **RGB** чи **RGBA**;
- **dataType** вказує на тип значень цих компонент – **GL_INT** або **GL_FLOAT**;
- **offset** дорівнює 0 для окремого колірного масиву або кількості байтів між кожним набором колірних компонент у масиві, якщо в одному масиві **поєднано інформацію про колір та вершини**.

В індексному колірному режимі масив колірних значень задають функцією

```
glIndexPointer(dataType, offset, colorIndex);
```

- колірні індекси перераховують у масиві **colorIndex**,
- параметри **dataType** і **offset** такі самі, як у функції **glColorPointer**,
- параметр **nColorComponents** не потрібен, бо індекси колірної таблиці описують за допомогою одного значення.

Приклад використання колірних масивів для створення куба (колір вершин – синій на лицевій грані, червоний – на зворотній):

```
typedef GLint vertex3[3], color3[3];
vertex3 pt[8] = {0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1,
0, 1,
      1, 1, 0, 1, 1, 1, 1};
color3 hue[8] = {1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0,
1, 0,
      0, 0, 0, 1, 0, 0, 1};
glEnableClientState(GL_VERTEX_ARRAY);
glEnableClientState(GL_COLOR_ARRAY);
glVertexPointer(3, GL_INT, 0, pt);
glColorPointer(3, GL_INT, 0, hue);
```

Приклад об'єднаного масиву

```
static GLint hueAndPt[] = {1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
      0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0,
      1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1,
      0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1};
glVertexPointer(3, GL_INT, 6*sizeof(GLint), hueAndPt[3]);
glColorPointer(3, GL_INT, 6*sizeof(GLint), hueAndPt[0]);
```

Перші три елементи цього масиву задають колірне значення RGB, наступні три значення – набір координат вершини, і такий порядок зберігається до останнього опису кольору і вершини. Параметр **offset** дорівнює кількості байтів між сусідніми колірними значеннями або значеннями координат, яке в обох випадках дорівнює **6*sizeof(GLint)**. Колірні значення починаються з першого елемента об'єднаного масиву, який дорівнює **hueAndPt[0]**, а значення координат вершин – з четвертого елемента, що дорівнює **hueAndPt[3]**.

Зобразити куб з вершинами різних кольорів можна знову ж таки за допомогою стандартної процедури **glDrawElements**.

Якщо замінити колірні значення і координати вершин цілого типу на значення з плаваючою крапкою, то можна скористатись функцією, яка визначає зразу всі масиви вершин і кольорів, а також іншу інформацію:

```
glInterleavedArrays(GL_C3F_V3F, 0, hueAndPt);
```

перший параметр – константа, яка показує, що і колір (**C**), і координати вершин (**V**) описують за допомогою трикомпонентних значень з плаваючою крапкою. Крім того, ця функція автоматично активізує і колірні масиви, і масиви координат.

3.2. Інші функції задання атрибутів

У попередньому підрозділі було розглянуто атрибут, характерний для зображення усіх графічних примітивів, – колір. Нагадаємо, що колірні компоненти базових та додаткових графічних примітивів визначають за допомогою значень RGB або як елементи колірної таблиці функціями **glColor** або **glIndex**. Перейдемо до опису інших атрибутів.

3.2.1. Визначення атрибутів точок

Розмір точки задають командою

```
glPointSize(size);
```

точку зображають як квадратний блок пікселів; параметр **size** визначає кількість горизонтальних і вертикальних пікселів, які складають зображення точки; йому присвоюють додатне значення з плаваючою крапкою, яке буде заокруглене до цілого (якщо тільки не треба усувати нерівності країв точки); значення за

замовчуванням – 1.0. Отож за розміру точки 1.0 зображується один піксель, за розміру точки 2.0 – масив пікселів 2 на 2.

Функції задання атрибутів можна поміщати всередині пари **glBegin/glEnd** і поза нею.

Приклад зображення трьох точок різного кольору і розміру.

```
glColor3f (1.0, 0.0, 0.0);
glBegin(GL_POINTS);
   glVertex2i(50, 100);
   glPointSize(2.0);
   glColor3f (0.0, 1.0, 0.0);
   glVertex2i(75, 150);
   glPointSize(3.0);
   glColor3f (0.0, 0.0, 1.0);
   glVertex2i(100, 200);
glEnd();
```

Перша точка червоного кольору має стандартний розмір, друга – зелена подвійного розміру, третя – синя потрійного розміру.

3.2.2. Задання атрибутів відрізків

Зовнішній вигляд **прямолінійного відрізка**, окрім кольору, визначають ще два атрибути – **ширина** і **стиль**. **Ширину** задають функцією

```
glLineWidth(width);
```

параметру **width** присвоюють значення з плаваючою крапкою, яке заокруглюється до найближчого невід'ємного цілого числа, у разі заокруглення до 0 лінія матиме ширину 1,0 (це значення за замовчуванням).

За замовчуванням відрізок відображається як суцільна лінія, однак можна зображати ще штриховані й пунктирні лінії, а також лінії, складені з різних комбінацій точок і штрихів; можна змінювати довжину штрихів і відстань між точками і штрихами.

Стиль зображення ліній задають функцією

```
glLineStipple(repeatFactor, pattern);
```

- цілочисловий параметр **repeatFactor** повідомляє, скільки разів повинен повторюватись кожен розряд у зразку (шаблоні) перед застосуванням наступного розряду, за замовчуванням – 1;
- параметр **pattern** (16-ве ціле число) вказує, як повинна зображуватись лінія; в цьому шаблоні 1 означає стан пікселя "включено", 0 – "виключено", цей шаблон застосовують до пікселів, починаючи з молодших розрядів; за замовчуванням він має вигляд **0xFFFF** (в кожному розряді пікселя є значення 1), що відповідає суцільній лінії; інші можливі значення:

- **0x00FF** – штрихова лінія з 8 пікселів кожен штрих, між кожними двома штрихами є восьмипіксельна порожня відстань (якщо коефіцієнт повтору дорівнює 1), починається зі штриха;
- **0x1C47** – штрих-пунктирна лінія;
- **0x0101** – пунктирна лінія.

Для ламаної лінії заданий зразок стилю не відновлюється на початку кожного відрізка – він неперервно продовжується протягом усіх відрізків, починаючись в першій точці ламаної і закінчуючись в останній точці останнього відрізка цього списку.

Перед зображенням лінії за допомогою шаблону треба активізувати можливість побудови ліній різного стилю командою

```
glEnable(GL_LINE_STIPPLE);
```

якщо не включити цю функцію, то зображатимуться лише суцільні лінії; відключення – функцією

```
glDisable(GL_LINE_STIPPLE);
```

OpenGL дає змогу задавати лінії з градацією колірних відтінків, зокрема, суцільну лінію можна зображати за допомогою лінійної інтерполяції колірних кодів заданих кінців лінії.

Приклад зображення проінтерпольованої за кольором лінії:

```
glShadeModel(GL_SMOOTH);
glBegin(GL_LINES);
   glColor3f(0.0, 0.0, 1.0);
   glVertex2i(50, 50);
   glColor3f(1.0, 0.0, 0.0);
   glVertex2i(250, 250);
glEnd();
```

На одному кінці лінії задано синій колір, а на другому – червоний.

Аргументом функції **glShadeModel** може бути **GL_FLAT;** у цьому випадку відрізок зображується кольором другого кінця, за замовчуванням задано **GL_SMOOTH**.

3.2.3. Налаштування атрибутів багатокутників

Основний атрибут зафарбованих фігур, який пропонує універсальна графічна бібліотека, – це стиль зображення внутрішньої області. Її можна заповнити одним кольором, заданим зразком чи залишити чистою, показавши лише межі фігури (рис. 3.1).

В OpenGL наявні процедури зафарбування тільки *опуклих* багатокутників. За замовчуванням описані всередині пари **glBegin/glEnd** багатокутники будуть суцільно заповнені кольором, попередньо визначеним функцією **glColor**.

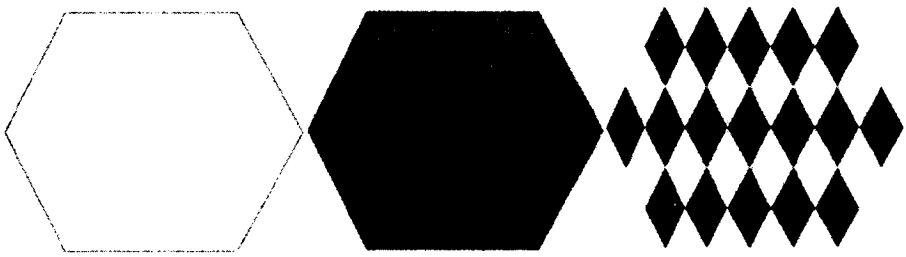


Рис. 3.1. Основні стилі зафарбування багатокутників: без зафарбовування (а), зафарбовування одним кольором (б), заповнення зразком (в)

Крім зафарбування багатокутників суцільним кольором, в OpenGL можна робити заповнення за допомогою шаблонів. У цьому випадку зображення отримують після виконання 4 дій.

1. Визначають зразок заповнення як поточний шаблон:

```
glPolygonStipple(fillPattern);
```

fillPattern – маска розміром 32 на 32 бітів, яку можна задати, наприклад, так:

```
GLubyte fillPattern[]=0xff, 0x00, 0xff, 0x00, ...;
```

біти треба задавати, починаючи з нижнього рядка шаблону в напрямку знизу догори до верхнього (32-го) рядка, як для функції **glBitmap** (підрозділ 2.2.1); цей шаблон багаторазово повторюється на всій площині вікна зображення, починаючи з нижнього лівого кута, а в тих місцях, де він накладається на задані багатокутники, їх заповнюють зазначеним шаблоном.

2. Задають режим зафарбування багатокутників за допомогою шаблона:

```
glEnable(GL_POLYGON_STIPPLE);
```

3. Активізують можливість OpenGL зафарбування багатокутників:

```
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
```

4. Описують багатокутники, які треба зафарбувати.

Зразок заповнення покриває багатокутник аж до його сторін, враховуючи їх. Отож, навколо зафарбованої області не буде жодних ліній контурів, якщо тільки не додати їх до зображення спеціально.

Багатокутники можна також зафарбувати, використовуючи **текстурні шаблони**, які розглянемо в розділі 10. Вони дають зразки, що нагадують поверхні дерева, цегли, нержавійної сталі чи якогось іншого матеріалу. Крім того, можна отримати **інтерполяційне розфарбування** внутрішньої області так само, як це робиться для відрізків. Щоб здійснити це, різним вершинам багатокутника надають різні кольори. Інтерполяційне зафарбовування внутрішніх областей багатокутників

використовують для створення реалістичних зображень затінених поверхонь за різних умов освітлення.

Інколи треба показати тільки сторони або вершини багатокутника, тобто отримати його **контурне зображення**. Для реалізації цих опцій використовують функцію

```
glPolygonMode(face, displayMode);
```

- **face** показує, до якої поверхні багатокутника треба застосувати режим, визначений другим параметром: **GL_FRONT** (до лицевої), **GL_BACK** (задньої), **GL_FRONT_AND_BACK** (обидвох);
- **displayMode** може мати одне з трьох значень:
    - **GL_LINE** (зображати тільки сторони),
    - **GL_POINT** (тільки вершини),
    - **GL_FILL** (за замовчуванням, відображати і сторони, і вершини, і заповнення).

Щоб зобразити *багатокутник зі сторонами іншого кольору*, потрібно створити його двічі: спочатку з параметром **GL_FILL** у функції **glPolygonMode**, а потім – з **GL_LINE**.

Для тривимірного багатокутника (у якого не всі вершини лежать у площині *xy*) за такого способу зображення сторін зафарбованої фігури можуть з'явитися *проміжки між сторонами і внутрішньою областю*. Цей ефект, який інколи називають **зшиванням**, зумовлений відмінностями обчислювальних операцій алгоритмів порядкового заповнення і побудови прямих сторін багатокутника, які дають різні значення глибини для точок межі. Тому під час перевірок на видимість для зображення деяких точок на межі багатокутника треба використовувати колір заповнення, а не колір сторін.

Одним зі способів усунення проміжків у зображенні сторін тривимірного багатокутника – *змістити значення глибини*, розраховані за допомогою процедури зафарбовування, так, щоб вони не накладались на значення глибини сторін цього багатокутника. Це можна зробити за допомогою таких двох функцій:

```
glEnable(GL_POLYGON_OFFSET_FILL);
glPolygonOffset(factor1, factor2);
```

Перша функція активізує процедуру зміщення для порядкового заповнення, а другу використовують для задання пари значень з плаваючою крапкою **factor1, factor2**, які застосовують для обчислення *величини зсуву глибини*:

$$\mathbf{depthOffset = factor1 \cdot maxSlope + factor2 \cdot const,}$$

де

- **maxSlope** – тангенс максимального кута нахилу багатокутника,
- **const** – константа реалізації.

Для багатокутника, який лежить у площині *xy*, кут нахилу дорівнює 0, інакше максимальний тангенс кута знаходять як зміну глибини багатокутника, поділену або на зміну величини *x*, або на зміну величини *y*. Зазвичай значення цих двох коефіцієнтів дорівнюють 0.75 або 1.0, хоча й часто, щоб отримати добрий результат, з ними треба поекспериментувати.

Для ілюстрації присвоєння значень коефіцієнтам зсуву змінимо попередній фрагмент програми так:

```
glColor3f(0.0, 1.0, 0.0);
glEnable(GL_POLYGON_OFFSET_FILL);
glPolygonOffset(1.0, 1.0);
/* Викликаємо процедуру створення багатокутника */
...
glColor3f(1.0, 0.0, 0.0);
glPolygonMode(GL_FRONT, GL_LINE);
/* Знову викликаємо процедуру створення багатокутника */
```

Внутрішня зафарбована частина багатокутника відсунеться трохи вглиб і не накладатиметься на глибини сторін багатокутника. Цей підхід можна також реалізувати, використавши зсув в алгоритмі прямих ліній – замінивши аргумент функції **glEnable** на **GL_POLYGON_OFFSET_LINE**. У цьому випадку застосовують від'ємні коефіцієнти, внаслідок чого глибина сторін багатокутника стане трохи меншою. А якщо треба зобразити лише різнокольорові точки у вершинах багатокутника, замість того, щоб виділяти сторони, то аргументом зазначеної функції має бути константа **GL_POLYGON_OFFSET_POINT**.

Ще один спосіб усунення ефекту зшивання на межах багатокутника – *використати буфер шаблонів* для обмеження області зафарбовування внутрішньої частини багатокутника, щоб вона не накладалась на сторони. Однак цей метод є складнішим і, в загальному випадку, повільнішим, тому методу зсуву глибини багатокутника варто надати перевагу.

Щоб за допомогою стандартних процедур OpenGL зобразити **увігнутий багатокутник**, його спочатку треба поділити на набір опуклих багатокутників, зазвичай трикутників, а потім зобразити як зафарбовану фігуру або відобразити лише вершини трикутників (якщо треба показати лише вершини багатокутника). Треба зазначити, що для зображення увігнутого багатокутника у вигляді контуру не можна просто задати режим зображення як **GL_LINE**, бо тоді буде зображено всі сторони трикутників, з яких складається увігнутий багатокутник. Тому треба усунути всі внутрішні сторони з контурного зображення. Для цього кожну вершину багатокутника записують з однобітовим **прапорцем**, який вказує, чи з'єднано цю вершину лінією з наступною вершиною. Треба перевести прапорець у стан "вимкнено", і тоді сторону, що йде за цією вершиною, зображено не буде. Прапорець для сторони встановлюють функцією

```
glEdgeFlag(flag);
```

flag має значення **GL_FALSE** (за вершиною немає сторони), **GL_TRUE** (є – за замовчуванням).

Ця функція може міститися всередині пари **glBegin/glEnd**.

Прапорці сторін багатокутників також задають як **масиви**, які можна об'єднати чи зв'язати з масивами вершин та кольорів (підрозділи 2.1.4 та 3.1.3). Використовують такі оператори створення масиву прапорців:

```
glEnableClientState(GL_EDGE_FLAG_ARRAY);
glEdgeFlagPointer(offset, edgeFlagArray);
```

параметр **offset** показує кількість байтів між значеннями прапорців у масиві **edgeFlagArray**, за замовчуванням дорівнює 0.

Хоча за замовчуванням передня і задня поверхні багатокутника ідентифікуються заданням його вершин, є можливість незалежного маркування вибраних поверхонь сцени (як передніх чи задніх):

```
glFrontFace(vertexOrder);
```

Якщо параметру присвоїти значення константи **GL_CW**, то багатокутник, вершини якого розміщено за годинниковою стрілкою, вважатимуть повернутим лицевою стороною; цією можливістю можна скористатися для переставлення місцями поверхонь багатокутників, вершини яких задано за годинниковою стрілкою; константа **GL_CCW** означає, що лицевій стороні багатокутника відповідає задання вершин у напрямі проти годинникової стрілки, що і є порядком задання вершин за замовчуванням.

3.2.4. Функції атрибутів символів

Зовнішній вигляд символів можна контролювати такими атрибутами: накреслення, розмір, колір і орієнтація, а для рядків символів – ще й вирівнювання.

OpenGL пропонує два способи зображення символів. Можна розробити шрифт, використовуючи функції бітового відображення з кореневої бібліотеки, або викликати стандартні процедури GLUT для створення окремих символів. Бібліотека GLUT містить функції для зображення визначених наборів бітових і векторних символів. Тому атрибути, які можна застосовувати до цих символів, є такими самими, які використовують до бітових масивів або відрізків.

Як для растрового, так і для ескізного (штрихового) шрифтів колір зображення визначається поточним станом кольору. В загальному випадку відстань між символами і розмір самих символів залежать від заданих параметрів шрифту. Крім того, можна задавати ширину (за допомогою функції **glLineWidth**) і тип ліній (функцією **glLineStipple**) ескізних символів.

3.2.5. Функції OpenGL для усунення контурних нерівностей

Стандартні процедури усунення контурних нерівностей активізують за допомогою команди

```
glEnable(primitiveType);
```

тут параметру присвоюють значення однієї з символьних констант **GL_POINT_SMOOTH, GL_LINE_SMOOTH, GL_POLYGON_SMOOTH**. Допустивши, що всі колірні значення задано в режимі RGBA, треба ще активізувати режими змішування кольорів:

```
glEnable(GL_BLEND);
```

Далі застосовують режим змішування кольорів, описаний у підрозділі 3.1.2, і викликають функцію

```
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SCR_ALPHA);
```

Операції згладжування будуть ефективнішими, якщо для колірних специфікацій об'єктів використовувати великі значення коефіцієнта альфа.

Усунення контурних нерівностей можна застосовувати й у випадку колірних таблиць. Однак у такому колірному режимі спочатку треба створити *лінійно змінний колірний шаблон* – таблицю відтінків кольорів від кольору фону до кольору об'єкта.

3.2.6. Функції запиту

Поточні значення будь-якого параметра стану, зокрема й налаштування атрибутів, можна з'ясувати за допомогою *функцій запиту*. Їх застосовують для копіювання заданих значень стану у масив, який можна зберігати для повторного використання чи перевірки поточного стану системи, якщо виникає якась помилка.

Для поточних значень атрибутів використовують функції

```
glGetBooleanv();
glGetFloatv();
glGetIntegerv();
glGetDoublev();
```

У кожній із наведених функцій задають два аргументи: символьну константу, яка визначає атрибут чи інший параметр стану, та посилання на масив того типу даних, який зазначений в імені функції. Наприклад, про поточні колірні налаштування в режимі RGBA з плаваючою крапкою можна довідатися за допомогою функції

```
glGetFloatv(GL_CURRENT_COLOR, colorValues);
```

яка поточні колірні компоненти передає у масив **colorValues**.

Щоб знайти цілочислові значення колірних компонент, викликають функцію **glGetIntegerv**.

У цих функціях для повернення поточних значень параметрів стану можна використовувати й інші константи: **GL_POINT_SIZE, GL_LINE_WIDTH** або **GL_CURRENT_RASTER_POSITION**. Діапазон значень розмірів точки чи ширини лінії, який підтримує система, можна перевірити за допомогою констант **GL_POINT_SIZE, GL_LINE_WIDTH_RANGE**.

3.2.7. Групи атрибутів

Атрибути й інші параметри стану утворюють **групи атрибутів**. Кожна група містить набір взаємно пов'язаних параметрів стану: в *групі атрибутів точок* є розмір і рівень згладжування; в *групі атрибутів прямих ліній* – ширина, тип пунктиру, зразок пунктирного рисунка, лічильник повтору цього рисунка і стан згладжування; в *групі атрибутів багатокутників* – 11 параметрів, зокрема зразок заповнення, прапорець позначення лицьового чи зворотного боку і стан згладжування. Оскільки колір є атрибутом всіх примітивів, він належить до своєї групи атрибутів. Деякі параметри можуть входити до кількох груп.

У пакеті OpenGL є понад *20 різних груп атрибутів*, але всі параметри однієї чи декількох груп можна зберігати або змінювати за допомогою однієї функції. Щоб **зберегти** всі параметри, які належать до заданої групи, використовують команду

```
glPushAttrib(attrGroup);
```

параметр **attrGroup** може мати такі значення – символьні константи:

- **GL_POINT_BIT**, **GL_LINE_BIT**, **GL_POLIGON_BIT** – параметри точок, ліній і багатокутників відповідно;
- **GL_CURRENT_BIT** – параметри кольору;
- **GL_ALL_ATTRIB_BITS** – всі параметри стану в усіх групах атрибутів.

Дві або декілька груп параметрів можна об'єднувати за допомогою логічної операції "або" (|).

Приклад занесення в стек атрибутів усіх параметрів точок, відрізків і багатокутників:

```
glPushAttrib(GL_POINT_BIT | GL_LINE_BIT | GL_POLIGON_BIT);
```

Функція **glPushAttrib()** призначена для занесення всіх параметрів, що входять до групи, в **стек атрибутів**. Записавши групу параметрів стану, всі значення зі стеку атрибутів можна відновити функцією

```
glPopAttrib();
```

вона не має аргументів, оскільки слугує **для зміни** поточного стану, а для цього використовують усі значення зі стеку.

Зазначені команди для збереження і перепризначення параметрів стану використовують *стек атрибутів сервера*. Пакет OpenGL має ще *стек атрибутів клієнта* для збереження і перевизначення параметрів стану клієнта; функціями доступу до нього є **glPushClientAttrib** і **glPopClientAttrib**. Є лише дві групи атрибутів клієнта: одна для режимів збереження пікселів, друга – для масивів вершин. До параметрів збереження пікселів належить інформація щодо розміщення байтів і тип масиву, які використовують для запису фрагментів зображення. Параметри масиву вершин дають інформацію про поточний стан масиву вершин, наприклад, про активацію різних масивів.

3.3. Загальна структура прикладної програми в OpenGL

Тут уже було розглянуто мінімальний набір операцій, необхідний для створення зображення (підрозділ 1.5). Залишилося ще декілька завдань, які необхідно виконати для того, щоб написати повну програму. Насамперед треба вибрати колір фону для вікна зображення, а також створити процедуру, яка містить усі функції, що підходять до того рисунка, який потрібно зобразити.

За допомогою колірних значень RGB встановлюють білий колір фону вікна зображення функцією

```
glClearColor(1.0, 1.0, 1.0, 0.0);
```

Якщо кожній складовій присвоїти якісь однакові проміжні значення між 0.0 та 1.0, то отримаємо якийсь відтінок сірого кольору.

Зазначена команда присвоює колір тла вікну зображення, а поміщає його на екран команда

```
glClear(GL_COLOR_BUFFER_BIT);,
```

яку викликають у потрібному місці програми; її аргумент повідомляє, що в буфері кольору (буфері регенерації) потрібно встановити колір фону, визначений у функції **glClearColor**.

Крім визначення кольору фону зображення, можна вибирати різні колірні схеми для об'єктів, які потрібно зобразити на екрані. У першій програмі зобразимо двовимірний відрізок червоним кольором:

```
glColor3f(1.0, 0.0, 1.0);
```

Потім треба повідомити програмі, як спроектувати рисунок на вікно зображення, оскільки створення двовимірного зображення пакет OpenGL розглядає як частковий випадок тривимірного. Тому OpenGL опрацьовує рисунок за допомогою операцій тривимірної візуалізації. Можна задати вигляд проекції (режим) та інші потрібні параметри візуалізації за допомогою двох функцій:

```
glMatrixMode(GL_PROJECTION);
gluOrtho2D(0.0, 200.0, 0.0, 150.0);
```

Тут виклик **gluOrtho2D(0.0, 200.0, 0.0, 150.0)** означає, що для відображення двовимірної прямокутної області на екрані треба використати ортогональну проекцію і що значення координати x цього прямокутника мають лежати в межах від 0.0 до 200.0, а значення координати y – від 0.0 до 150.0.

Нарешті, треба викликати відповідні стандартні функції OpenGL, щоб створити відрізок:

```
glBegin(GL_LINES);
   glVertex2i(180, 15);
   glVertex2i(10, 145);
glEnd();
```

Тепер можна зібрати всі частини разом. Наступна програма OpenGL складається з трьох процедур. Помістимо всі функції ініціалізації та присвоювання відповідних одноразових параметрів у процедуру **init**. Геометричний опис рисунка, який хочемо зобразити, буде в процедурі **lineSegment**, яку викликає функція бібліотеки GLUT **glutDisplayFunc**. А в процедурі **main** запишемо функції бібліотеки GLUT, що відображають відрізок на екран.

```
#include <GL/glut.h>
void init(void) {
   glClearColor(1.0, 1.0, 1.0, 0.0); // білий колір фону
вікна
   glMatrixMode(GL_PROJECTION); // задає параметри
проектування
   gluOrtho2D(0.0, 600.0, 0.0, 500.0);
}
void lineSegment (void){
   glClear(GL_COLOR_BUFFER_BIT); // очищує вікно зображення
   glColor3f(1.0, 0.0, 0.0); // задання червоного кольору
відрізка
   glBegin(GL_LINES);
      glVertex2i(180, 15); // описує геометрію відрізка
      glVertex2i(10, 145);
   glEnd();
   glFlush(); // опрацьовує всі функції якомога швидше
}
void main(int argc, char** argv) {
   glutInit (&argc, argv); // ініціалізація GLUT
   /* установка режиму дисплея */
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   /* задає верхній лівий кут вікна зображення */
   glutInitWindowPosition(50, 100);
   /* задає ширину і висоту вікна зображення */
   glutInitWindowSize(480, 640);
   /* створює вікно зображення */
   glutCreateWindow("Приклад програми");
   init(); // процедура ініціалізації
   glutDisplayFunc(lineSegment) // посилає графічне
зображення
                                // у вікно
   glutMainLoop(); // зображує все і очікує
}
```

Отже, функція **main** містить виклики функцій з бібліотеки GLUT, які встановлюють параметри вікна (чи вікон) конкретної програми і забезпечують підтримку процесу відображення з боку операційної системи. До неї можна було б включити і звертання до функцій з бібліотек GL і GLU, але текст програми буде зрозумілішим, якщо їх винести в окремий блок – функцію **init**, в якій налаштовують значення атрибутів та опцій прикладної програми.

У кінці процедури **lineSegment** є функція **glFlush**, яку тут ще не обговорювали. Це стандартна функція для прискорення виконання функцій OpenGL, записаних у буферах, які розташовані в різних місцях обчислювальної системи. В завантаженій мережі можуть виникати затримки під час обробки даних у деяких буферах. Але виклик цієї функції призводить до звільнення буферів і обробки функцій OpenGL.

Процедуру **lineSegment**, у якій описано рисунок, називають *функцією зворотного виклику зображення.* Її "реєструє" функція **glutDisplayFunc** як стандартну функцію, яку викликають щоразу, коли може знадобитись знову вивести на екран вікно відображення. Це може відбутися, наприклад, якщо вікно зображення пересунуто. У загальному випадку програми OpenGL будують у вигляді наборів функцій зворотного виклику, які викликають під час виконання певних дій.

3.4. Інші функції кольору

У програмі підрозділу 3.3 використано функцію, яка слугує для вибору колірних компонент RGB для вікна зображення:

```
glClearColor(red, green, blue, alpha);
```

Кожній колірній компоненті у цьому позначенні, а також параметру альфа присвоюють значення з плаваючою крапкою в діапазоні від 0.0 до 1.0. За замовчуванням значення всіх чотирьох параметрів дорівнюють 0.0, що дає чорний колір. Відтінки сірого кольору можна отримати за однакових значень усіх колірних компонент у зазначеному діапазоні. Змішування відбувається лише в тому випадку, коли активізована така можливість; воно є неможливим, якщо значення задано колірною таблицею.

Як зазначено у підрозділі 2.2.1, у пакеті OpenGL є декілька *буферів кольору*, які можна використовувати як поточний буфер регенерації, зображаючи сцену, а функція **glClearColor** слугує для специфікації кольору у всіх колірних буферах. Після цього до колірних буферів застосовують операцію зафарбовування кольором чистого вікна зображення, що виконують за допомогою команди

```
glClear(GL_COLOR_BUFFER_BIT);
```

За допомогою функції **glClear** можна також задати вихідні значення для буферів, які існують в OpenGL. Це *буфер нагромадження*, в якому зберігають інформацію про змішані кольори, *буфер глибини*, який містить значення глибини

(відстань до точки спостереження) об'єктів сцени, і *буфер шаблонів*, у якому зберігають дані, що визначають межі рисунка.

В індексному колірному режимі для задання кольору вікна зображення використовують замість **glClearColor** функцію

```
glClearIndex(index);
```

Після виконання цієї команди кольору фону присвоюють колір, записаний в позиції **index** у колірній таблиці. Виклик функції **glClear** зображає вікно саме в цьому кольорі.

У прикладі програми в підрозділі 1.5 колірний режим зображення RGB задано командою

```
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
```

Перший параметр зі списку аргументів повідомляє, що для буфера кадру використовують один буфер, а другий параметр встановлює режим RGB (або RGBA) – колірний режим за замовчуванням, для вибору цього режиму використовують символьну константу **GLUT_RGB** або **GLUT_RGBA**. Якщо треба описати колір за допомогою елементів таблиці кольорів, використовують константу **GLUT_INDEX**.

У бібліотеці OpenGL є багато інших функцій для задання кольору, які підходять для виконання різних завдань: зміни колірних моделей, налаштування ефектів освітлення сцени, опису ефектів камери і зафарбування поверхонь об'єктів. Інші колірні функції розглянемо під час вивчення кожного окремого процесу в системі КГ. У цьому розділі розглянуто лише ті функції, які пов'язані зі специфікацією кольору графічних примітивів.

3.5. Контрольні питання

1. Які є колірні режими?
2. Де застосовують коефіцієнт альфа?
3. За допомогою яких функцій вибирають поточні колірні компоненти?
4. Яка функція заносить опис кольорів до колірної таблиці в позицію із заданим індексом?
5. Які функції використовують для змішування кольорів?
6. Для чого використовують колірні масиви та об'єднані масиви?
7. Які атрибути задають, описуючи точки, та які функції OpenGL використовують при цьому?
8. Які атрибути задають, описуючи відрізки, та які функції OpenGL використовують при цьому?
9. Опишіть алгоритм зафарбовування опуклих багатокутників.
10. Як можна отримати інтерполяційне розфарбування внутрішньої області?
11. Яку функцію використовують для контурного зображення багатокутника?

12. Що таке ефект зшивання і які є способи його усунення під час зображення сторін тривимірного багатокутника?
13. Як за допомогою стандартних процедур OpenGL можна зобразити увігнутий багатокутник?
14. Для чого потрібна функція **glFrontFace**?
15. Які є функції атрибутів растрових та ескізних символів?
16. Як усувають контурні нерівності?
17. Як можна довідатися про поточні значення будь-якого параметра стану, зокрема й налаштування атрибутів?
18. Для чого використовують групи атрибутів?
19. Назвіть функції доступу до стеків атрибутів сервера і клієнта.
20. Для чого призначені функції **main**, **init** та **glFlush**?
21. Що таке функція зворотного виклику зображень?
22. Які функції слугують для задання кольору вікна зображення в різних режимах?

3.6. Приклади тестових питань

1. Функція **glLineStipple(repeatFactor, pattern)** задає:

   а) ширину лінії;
   б) колір лінії;
   в) стиль зображення лінії.

2. Функція **glIndexPointer(dataType, offset, colorIndex)** задає масив значень у:

   а) колірному режимі RGB;
   б) колірному режимі RGBA;
   в) індексному колірному режимі.

3. Функція **glColorPointer(3, dataType, offset, colorArray)** задає масив значень у:

   а) колірному режимі RGB;
   б) колірному режимі RGBA;
   в) індексному колірному режимі.

4. Функція **glColorPointer(4, dataType, offset, colorArray)** задає масив значень у:

   а) колірному режимі RGB;
   б) колірному режимі RGBA;
   в) індексному колірному режимі.

5. Функція **glColor3f(0.0, 0.0, 1.0)** описує:

   а) червоний колір;

б) зелений колір;
в) синій колір.

6. Функція **glColor3f(1.0, 0.0, 1.0)** описує:

а) червоний колір;
б) зелений колір;
в) синій колір.

7. Функцію **glFlush** використовують для:

а) включення викликів функцій з бібліотек GL і GLU;
б) включення викликів функцій з бібліотеки GLUT;
в) прискорення виконання функцій OpenGL.

8. Функцію **main** використовують для:

а) включення викликів функцій з бібліотек GL і GLU;
б) включення викликів функцій з бібліотеки GLUT;
в) прискорення виконання функцій OpenGL.

9. Функцію **init** використовують для:

а) включення викликів функцій з бібліотек GL і GLU;
б) включення викликів функцій з бібліотеки GLUT;
в) прискорення виконання функцій OpenGL.

Розділ 4

ГЕОМЕТРИЧНІ ПЕРЕТВОРЕННЯ ЗОБРАЖЕНЬ У КОМП'ЮТЕРНІЙ ГРАФІЦІ

Як відомо, сцену можна описати через такі графічні примітиви, як відрізки і зафарбовані об'єкти, використовуючи параметри цих примітивів. Розглянемо перетворення, за дії яких об'єкти змінюють своє положення чи розміри. Операції, які застосовують до геометричного опису об'єкта для зміни його положення, орієнтації чи розміру, називають **геометричними перетвореннями**.

4.1. Афінні перетворення

Більшість перетворень, які використовують у КГ, можна звести до скінченної множини афінних перетворень. **Афінним перетворенням** (АП) називають перетворення координат вигляду

$$x' = a_{11}x + a_{12}y + a_{13}z + b_1,$$
$$y' = a_{21}x + a_{22}y + a_{23}z + b_2,$$
$$z' = a_{31}x + a_{32}y + a_{33}z + b_3.$$

Кожна з перетворених координат x', y', z' є лінійною функцією вихідних координат x, y, z, а параметри $a_{ij}(i, j = 1,...,3)$ і b_i – це константи, які визначають тип перетворення. АП будь-якої розмірності мають таку загальну властивість: паралельні лінії перетворюються на паралельні лінії і скінченні точки – на скінченні точки.

Плоскопаралельне перенесення (зсув), обертання, масштабування, відбиття і скіс є афінними перетвореннями. Будь-яке АП завжди можна виразити як певну комбінацію перелічених п'яти перетворень. Ще один приклад АП – перетворення координатних зображень сцени з однієї системи координат на іншу, оскільки такий перехід можна описати як комбінацію зсуву та обертання. Внісши невеликі зміни, так само можна виразити і проективні перетворення – паралельне і перспективне. Будь-яке АП можна зобразити матрицею 4×4 вигляду

$$A = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} & \alpha_{14} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} & \alpha_{24} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & \alpha_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Матричне подання операцій геометричних перетворень є ефективним, бо воно дає змогу скоротити обчислення, подіявши складною матрицею на опис об'єкта, й одержати його перетворене положення. В геометричних перетвореннях однорідному коефіцієнту присвоєно значення 1. В усіх матрицях перетворень однорідних координат четвертий рядок не залежить від характеру перетворення, а слугує для того, щоб після перемноження зберігалось значення 1 у четвертій компоненті точки.

4.2. Основні геометричні перетворення зображень

До основних геометричних перетворень графічних об'єктів належать перенесення, обертання і масштабування.

4.2.1. Плоскопаралельне перенесення (зсув)

Плоскопаралельне перенесення (зсув) – це операція, яка зміщує точки на фіксовану відстань уздовж заданого напряму. Зсув задають тільки вектором зміщення. Якщо точка **p** зсувається в **p'** на відстань **d**, то таке перетворення можна записати у вигляді

$$\mathbf{p'=p+d}.$$

Метод подання перетворення зсуву засобом *сумування* матриць-стовпців погано поєднується з іншими афінними перетвореннями. Є й інший метод подання зсуву – за допомогою *перемноження матриць*

$$\mathbf{p'=T\cdot p},$$

де $$\mathbf{T}(\alpha_x,\alpha_y,\alpha_z)=\begin{bmatrix}1&0&0&\alpha_x\\0&1&0&\alpha_y\\0&0&1&\alpha_z\\0&0&0&1\end{bmatrix},\ \mathbf{d}=\begin{bmatrix}\alpha_x\\\alpha_y\\\alpha_z\\0\end{bmatrix},\ \mathbf{p}=\begin{bmatrix}x\\y\\z\\1\end{bmatrix},\ \mathbf{p'}=\begin{bmatrix}x'\\y'\\z'\\1\end{bmatrix}.$$

Матрицю **T** називають *матрицею зсуву*, її записують у вигляді $\mathbf{T}(\alpha_x,\alpha_y,\alpha_z)$, щоб підкреслити три незалежні параметри перетворення. Отже, зсув має 3 степені свободи, оскільки можна довільно задати 3 компоненти вектора переміщення. Тут видно перевагу застосування однорідних координат, оскільки саме вони дали змогу замінити сумування тривимірних матриць-стовпців перемноженням чотиривимірних матриць.

Матрицю оберненого перетворення можна сформувати або за допомогою алгоритму обернення матриць, або взявши до уваги, що обернене перетворення – це зсув на відстань **–d**. У будь-якому разі отримаємо

$$\mathbf{T}^{-1}(\alpha_x,\alpha_y,\alpha_z)=\mathbf{T}(-\alpha_x,-\alpha_y,-\alpha_z)==\begin{bmatrix}1&0&0&-\alpha_x\\0&1&0&-\alpha_y\\0&0&1&-\alpha_z\\0&0&0&1\end{bmatrix}.$$

Якщо вихідний багатокутник треба видалити, його можна перед зсувом зафарбувати кольором фону. У деяких графічних пакетах використовують й інші методи видалення. Крім того, якщо вихідне розміщення багатокутника треба зберегти, перенесені точки можна записати в іншому масиві.

Для зсуву інших об'єктів використовують схожі методи. Щоб змінити розміщення кола чи еліпса, переносять координати центра, а потім фігуру перерисовують у новому місці. Для сплайнової кривої зсувають точки, які визначають її траєкторію, а потім відновлюють точки, що лежать між перенесеними.

4.2.2. Обертання

Обертання зображення спочатку розглянемо *навколо початку координат* (рис. 4.1, *а*).

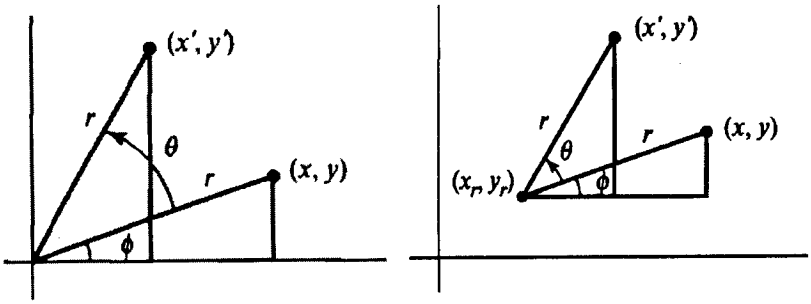


Рис. 4.1. Обертання точки відносно початку координат (а) та точки з координатами (x_r, y_r) (б)

У випадку повороту *навколо осі z* на кут θ

$$\mathbf{p}' = \mathbf{R}_z(\theta)\mathbf{p}\,, \tag{4.1}$$

де матриця обертання має вигляд:

$$\mathbf{R}_z = \mathbf{R}_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Рівняння перетворень для поворотів навколо двох інших координатних осей можна отримати за допомогою циклічної перестановки параметрів x, y і z у рівняннях (4.1): $x - y - z - x$. Так можна сформувати *матриці обертання навколо осей x і y*:

$$\mathbf{R}_x = \mathbf{R}_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{R}_y = \mathbf{R}_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Знак елементів матриць, що містять *sin*, відповідає прийнятому визначенню додатного напрямку повороту в правосторонній системі координат. Зазначимо, що елементи, які містять *cos*, завжди розташовані на головній діагоналі матриці, а пара елементів, що містять *sin*, – по різні боки від неї.

Оскільки після обертання на кут θ перетворені точки завжди можна повернути у вихідне положення, виконавши обертання навколо тієї самої осі на кут $-\theta$, маємо

$$\mathbf{R}^{-1}(\theta) = \mathbf{R}(-\theta) = \mathbf{R}^{T}(\theta),$$

оскільки із зміною знаку кута змінюється лише значення функції синус.

Обернену матрицю можна також отримати, помінявши рядки і стовпці вихідної, тому матрицю, обернену до будь-якої матриці повороту, можна знайти, обчисливши транспоновану.

Матриці, для яких операція транспонування дає такий самий результат, що й обернення, називають *ортогональними*.

Зсув і обертання належать до групи *ізометричних перетворень*. Комбінація цих перетворень не може змінити форми об'єкта, а змінює тільки його положення в просторі – позицію та орієнтацію.

4.2.3. Масштабування

Масштабування – це *анізометричне* афінне перетворення, яке збільшує або зменшує розміри об'єкта і має два варіанти: *пропорційне* (рівномірне у всіх напрямах або *перетворення подібності*) і *розтяг* тільки в одному напрямку.

Проста операція тривимірного масштабування полягає у множенні координат точок об'єкта на масштабні множники s_x, s_y, s_z, у результаті чого отримують перетворені координати:

$$x' = xs_x,\ y' = ys_y,\ z' = zs_z.$$

Матриця перетворення масштабування, що має фіксовану точку в початку координат, має вигляд

$$\mathbf{S}(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Набір параметрів цього перетворення містить фіксовану точку, напрямок, вздовж якого змінюється масштаб, і масштабний множник s:

- при $s > 1$ об'єкт розтягується в заданому напрямку;
- при $0 < s < 1$ – стискається;
- при $s = -1$ відбувається *відбиття* об'єкта щодо заданої фіксованої осі або площини відбиття у зазначеному напрямку (відбиття – це ізометричне перетворення);

- пропорційне (рівномірне) масштабування ($s_x = s_y = s_z$) зберігає вихідну форму об'єкта.

Для обернення матриці масштабування необхідно використати обернені значення масштабних коефіцієнтів за осями:

$$\mathbf{S}^{-1}(s_x, s_y, s_z) = \mathbf{S}(\frac{1}{s_x}, \frac{1}{s_y}, \frac{1}{s_z}).$$

4.2.4. Складні перетворення

Продемонструємо, як сформувати матрицю **складного перетворення**, перемножуючи матриці окремих базових (канонічних) перетворень. Отже, складне перетворення є суперпозицією базових.

Обертання навколо довільної фіксованої точки $\mathbf{p}_r$ *у разі збігання осі повороту з віссю z* (рис. 4.1, *б*) виконують як послідовність трьох перетворень:

- перенесення нерухомої точки у початок координат;
- обертання навколо осі z ;
- перенесення в початкове положення.

Як результат суперпозиції вказаних трьох перетворень одержимо матрицю

$$\mathbf{M}_z(\mathbf{p}_r, \theta) = \mathbf{T}(\mathbf{p}_r)\mathbf{R}_z(\theta)\mathbf{T}(-\mathbf{p}_r),$$

а після перемноження –

$$\mathbf{M}_z(\mathbf{p}_r, \theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & x_r - x_r\cos\theta + y_r\sin\theta \\ \sin\theta & \cos\theta & 0 & y_r - x_r\sin\theta - y_r\cos\theta \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Для обертання кривої спочатку обертають її визначальні точки, а потім за ними будують нову криву. Наприклад, щоб обернути еліпс навколо його центру, можна просто обертати велику і малу осі.

Масштабування відносно довільної нерухомої точки $\mathbf{p}_o\ (x_o, y_o, z_o)$ виконують як послідовність трьох перетворень:

- перенесення нерухомої точки у початок координат;
- масштабування відносно початку координат;
- перенесення в початкове положення.

Координати нерухомої точки вибирають так, щоб вона належала об'єкту, часто нею є центр мас. Тоді розміри об'єкта змінюються завдяки зміні відстані між його точками і нерухомою. Формули, що описують це масштабування, мають вигляд:

$$x' = xs_x + x_0(1 - s_x),\ y' = ys_y + y_0(1 - s_y),\ z' = zs_z + xz_0(1 - s_z).$$

Координати нерухомої точки включають в рівняння масштабування аналогічно до того, як у рівняння обертання включають координати центра обертання.

Результат масштабування, враховуючи згортку зазначених перетворень, а також некомутативність і асоціативність добутку матриць, матиме вигляд

$$\mathbf{T}(\mathbf{p}_0)\mathbf{S}(s_x,s_y,s_z)\mathbf{T}(-\mathbf{p}_0)=\begin{bmatrix} s_x & 0 & 0 & (1-s_x)x_0 \\ 0 & s_y & 0 & (1-s_x)y_0 \\ 0 & 0 & s_z & (1-s_x)z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Під час опрацювання об'єктів рівняння масштабування застосовують до параметрів, що визначають ці об'єкти. Щоб змінити розмір круга, можна масштабувати його радіус і розрахувати нові координати точок кола. Щоб змінити розмір еліпса, на головні осі діють параметрами масштабування, а потім будують новий еліпс (нерухомою точкою є центр еліпса).

Програмні процедури для побудови матриць тривимірного масштабування можна задати або з використанням послідовності зсув-масштабування-зсув, або безпосередньо включивши у процедуру координати нерухомої точки.

Обертання навколо осі, яка не паралельна до жодної з координатних осей, складається з п'яти кроків:

- перенесення осі обертання, щоб вона пройшла через початок координат;
- повороту об'єкта, щоб вісь обертання збіглася з однією з координатних осей;
- обертання навколо цієї осі;
- оберненого повороту для повернення осі обертання до вихідної орієнтації;
- оберненого перенесення осі в початкове просторове положення.

Вісь обертання можна визначити двома точками або однією точкою і напрямними кутами (чи напрямними косинусами), які вісь обертання утворює є двома координатними осями. Припустимо, що вісь обертання u визначено двома точками $\mathrm{P}_1(x_1,y_1,z_1)$ і $\mathrm{P}_2(x_2,y_2,z_2)$ і що напрям повороту – проти годинникової стрілки, якщо дивитися вздовж осі від P_1 до P_2, тобто її компонентами є $(x_2-x_1,y_2-y_1,z_2-z_1)$. Напрямні косинуси її одиничного вектора записують так:

$$u_x=\frac{x_2-x_1}{|u|},u_y=\frac{y_2-y_1}{|u|},\ u_z=\frac{z_2-z_1}{|u|},\ |u|=\sqrt{u_x^2+u_y^2+u_z^2}\ .$$

Матрицю перетворення, яка описує поворот на кут θ навколо довільної осі, можна виразити через сукупність семи окремих перетворень:

$$\mathbf{M}(\mathbf{p}_0,u_x,u_y,u_z,\theta)=\mathbf{T}(\mathbf{p}_0)\mathbf{R}(u_x,u_y,u_z,\theta)\mathbf{T}(-\mathbf{p}_0),$$

де

$$\mathbf{R}(u_x,u_y,u_z,\theta)=\mathbf{R}_x(-u_y,-u_z)\mathbf{R}_y(-u_x)\mathbf{R}_z(\theta)\mathbf{R}_y(u_x)\mathbf{R}_x(u_y,u_z),$$

$$\mathbf{R}_x(u_y,u_z)=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & u_z/d & -u_y/d & 0\\ 0 & u_y/d & u_z/d & 0\\ 0 & 0 & 0 & 1\end{bmatrix},\ \mathbf{R}_y(u_x)=\begin{bmatrix}d & 0 & -u_x & 0\\ 0 & 1 & 0 & 0\\ u_x & 0 & d & 0\\ 0 & 0 & 0 & 1\end{bmatrix}.$$

Тут вісь обертання суміщено з віссю z; $d=\sqrt{\alpha_y^2+\alpha_z^2}$.

Матрицю можна переписати так:

$$\mathbf{R}(u_x,u_y,u_z,\theta)=\begin{bmatrix}u_x^2a+\cos\theta & u_xu_ya-u_z\sin\theta & u_xu_za+u_y\sin\theta & 0\\ u_xu_ya+u_z\cos\theta & u_y^2a+\cos\theta & u_yu_za-u_x\sin\theta & 0\\ u_xu_za-u_y\cos\theta & u_zu_ya+u_x\sin\theta & u_z^2a+\cos\theta & 0\\ 0 & 0 & 0 & 1\end{bmatrix},$$

де $a=1-\cos\theta$.

Отже, для перетворення обертання треба специфікувати три параметри: фіксовану точку, кут повороту і пряму чи вектор, навколо якого виконується поворот (вісь обертання). Якщо фіксовану точку задано, то в нашому розпорядженні є 3 степені свободи: два кути, необхідні для задання орієнтації осі обертання, й один кут, що задає сам поворот.

4.2.5. Скіс

Хоча будь-яке афінне перетворення можна звести до послідовності обертань, перенесень і змін масштабу, є одне перетворення, яке настільки важливе в КГ, що розглядатимемо його також як базовий тип – це **перетворення скосу**.

Перетворення, яке так спотворює форму об'єкта, що нова форма виглядає ніби він складений з внутрішніх шарів, які ковзають один по одному, називають **скосом** (*shear*).

Розглянемо куб із центром у початку координат і ребрами, паралельними до осей координат. Якщо змістити його верхню грань праворуч, а нижню ліворуч, об'єкт буде *скошеним у напрямку осі x*, при цьому інші координати всіх точок залишаються незмінними. *Матриці x-скосу відносно лінії* $y=y_0$, *y-скосу відносно лінії* $x=x_0$, *z-скосу відносно опорної точки* z_0 мають вигляд

$$\mathbf{H}_x(h_x)=\begin{bmatrix}1 & h_x & 0 & -h_xy_o\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix},\ \mathbf{H}_y(h_y)=\begin{bmatrix}1 & 0 & 0 & 0\\ h_y & 1 & 0 & -h_yx_o\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix},$$

$$\mathbf{H}_z(h_{zx},h_{zx})=\begin{bmatrix}1 & 0 & h_{zx} & -h_{zx}z_o\\ 0 & 1 & h_{zy} & -h_{zy}z_o\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix},$$

де h_x,h_y,h_{zx},h_{zy} – параметри скосу, які можуть набувати будь-яких дійсних значень.

Матрицю *оберненого перетворення скосу* нескладно отримати, якщо врахувати, що треба виконати скіс у зворотному напрямку:

$$\mathbf{H}_x^{-1}(h_x) = \mathbf{H}_x^T(-h_x).$$

4.3. Геометричні перетворення в OpenGL

Коренева бібліотека OpenGL має окремі функції для кожного з базових геометричних перетворень, причому всі перетворення задають у трьох вимірах. Кожна процедура перетворення задає матрицю 4×4, яка діятиме на координати об'єктів.

4.3.1. Базові геометричні перетворення

Для **перенесення** OpenGL будує матрицю перетворення 4×4 за допомогою процедури

```
glTranslate*(tx, ty, tz);
```

параметри **tx**, **ty**, **tz** є компонентами вектора перенесення і можуть бути будь-якими дійсними числами; функція може мати коди **f** або **d**; у двовимірній графіці покладають **tz** = 0.0.

Матрицю **обертання** 4×4 генерують командою

```
glRotate*(theta, vx, vy, vz);
```

код – **f** або **d**; вектор **v** = (**vx**, **vy**, **vz**) визначає орієнтацію осі обертання, що проходить через початок координат; якщо не заданий як одиничний, то буде автоматично нормований перед обчисленням матриці, компоненти – будь-які значення з плаваючою крапкою; параметр **theta** – значення кута обертання в градусах, яке процедура переводить у радіани для тригонометричних розрахунків.

Матрицю **масштабування** 4×4 відносно початку координат задають процедурою

```
glScale*(sx, sy, sz);
```

код – **f** або **d**; параметри масштабування – будь-які дійсні числа. Функція підходить також і для генерації відбиття.

4.3.2. Операції з матрицями

Для встановлення *режиму проектування*, який визначає матрицю, яку буде використано в перетворенні проектування, щоб визначити, як сцена проектуватиметься на екран, застосовують функцію **glMatrixMode**:

```
glMatrixMode(GL_PROJECTION);
```

За допомогою тієї самої процедури **glMatrixMode** формують матрицю геометричних перетворень (ГМ); у цьому випадку її називають *матрицею вигляду (modelview matrix)* і задають командою

```
glMatrixMode(GL_MODELVIEW);
```

вона визначає матрицю 4×4 як *поточну матрицю* (ПМ), аргумент **GL_MODELVIEW** є значенням за замовчуванням.

Процедури геометричних перетворень, розглянуті в попередньому підрозділі, використовують для модифікації матриці вигляду, яку потім застосовують для перетворення точок сцени. За допомогою функції **glMatrixMode** можна задати ще два режими – *текстурний* і *колірний*. Матрицю текстури використовують для відображення текстурних узорів на поверхні, а матрицю кольорів – для переходу від однієї моделі кольору до іншої. Перетворення спостереження, проектування, накладання та зміни текстури й кольору будуть обговорені у наступному розділі 5 та розділі 10.

Після того, як задано режим, виклик процедури перетворення генерує матрицю, яку буде помножено на поточну матрицю вибраного режиму. Крім того, можна присвоювати значення елементам ПМ. У бібліотеці OpenGL є дві необхідні для цього функції:

```
glLoadIdentity();
```

тут як ПМ буде задано одиничну матрицю, та

```
glLoadMatrix*(elements16);
```

де елементам ПМ присвоюють інші значення: код – **f** або **d**; параметр **elements16** вказує на 16-елементний масив з одним індексом, який містить значення з плаваючою крапкою, задані **по стовпцях**.

Приклад ініціалізації матриці вигляду:

```
glMatrixMode(GL_MODELVIEW);
GLfloat elems[16];
GLint k;
for (k=0; k<nVerts; k++)
   elems[k]=float(k);
glLoadMatrixf(elems);
```

У результаті отримаємо матрицю:

$$\mathbf{M} = \begin{bmatrix} 0.0 & 4.0 & 8.0 & 12.0 \\ 1.0 & 5.0 & 9.0 & 13.0 \\ 2.0 & 6.0 & 10.0 & 14.0 \\ 3.0 & 7.0 & 11.0 & 15.0 \end{bmatrix}.$$

Крім того, можна виконати згортку заданої матриці з поточною

```
glMultMatrix*(otherElements16);
```

код – **f** або **d**; параметр передає 16-елементний масив з одним індексом, значення якого теж задано по стовпцях.

На ПМ *справа множиться* матриця, задана в процедурі **glMultMatrix**, і цей добуток стає поточною матрицею. Отже, якщо допустити, що поточною матрицею є матриця вигляду, яку позначимо через **M**, поновлену матрицю вигляду обчислимо так: **M=MM'**, де **M'** є матрицею, елементи якої задано параметром **otherElements16**.

Функцію **glMultMatrix** можна також використовувати, щоб задати *послідовність перетворень* для заданих матриць окремих перетворень.

При цьому слід пам'ятати, що перше застосоване перетворення в цій послідовності має бути останнім, заданим у коді (аналогія зі стеком). Отже, послідовність перетворень застосовується в порядку, **протилежному** тому, як її задано.

Важливо також пам'ятати, що в OpenGL матриці записують по стовпцях, звертання до елемента матриці m_{jk} є звертанням до елемента в стовпці j і рядку k **на відміну від стандартної** математичної домовленості, коли спочатку вказують рядок. Щоб уникнути помилок, пов'язаних з цим, матриці в OpenGL можна завжди задавати як 16-елементні масиви з одним індексом і пам'ятати, що елементи задано по стовпцях.

4.3.3. Стеки матриць

Для кожного з чотирьох режимів (матриці вигляду, проекції, текстури і кольору), вибраних за допомогою функції **glMatrixMode**, OpenGL підтримує стек матриць. На початку кожен стек містить лише одиничну матрицю. У будь-який час під час опрацювання сцени верхню матрицю кожного стеку називають *поточною матрицею* цього режиму. Після того, як буде задано перетворення спостереження і геометричне перетворення, зверху стеку матриць вигляду розміщується складна матриця 4 на 4, яка об'єднує перетворення спостереження і різні геометричні перетворення, які треба застосувати до сцени. У деяких випадках треба буде створити декілька послідовностей проекцій і перетворень, потім для кожної з них записати складну матрицю. Тому OpenGL підтримує стек матриць вигляду глибиною не меншою за 32. Кількість позицій, доступних в стеку матриць вигляду в конкретній реалізації OpenGL, можна визначити за допомогою запиту:

```
glGetIntegerv(GL_MAX_MODELVIEW_STACK_DEPTH, stackSize);
```

У результаті в масив **stackSize** повертається шукане цілочислове значення. Інші три режими мають мінімальну глибину стеку 2, а максимальна глибина конкретної реалізації визначається однією з констант:

GL_MAX_PROJECTION_STACK_DEPTH, GL_MAX_TEXTURE_STACK_DEPTH або **GL_MAX_COLOR_STACK_DEPTH.**

Крім того, можна визначити, скільки матриць у цей момент знаходиться в стеку:

```
glGetIntegerv(GL_MODELVIEW_STACK_DEPTH, numMats);
```

На початку стек матриць вигляду містить лише одиничну матрицю, тому якщо здійснити зазначений запит до початку роботи зі стеком, отримаємо значення 1. Подібні символьні константи існують і для визначення кількості матриць, які є в поточний момент у трьох інших стеках.

OpenGL має дві функції роботи з матрицями стеку. Використання цих функцій опрацювання стеків є ефективнішим, ніж робота з кожною матрицею стеку окремо, особливо, якщо стекові функції реалізовані апаратно. Наприклад, апаратна реалізація може копіювати декілька елементів матриці одночасно. Крім того, в стеку можна зберігати одиничну матрицю, щоб ініціалізувати поточну матрицю можна було швидше, ніж за допомогою повторювальних викликів функції **glLoadIdentity**.

Отже, використовуючи наведену нижче функцію, можна копіювати поточну матрицю у верхню позицію активного стеку і зберегти її копію у другій позиції стеку:

```
glPushMatrix();
```

У результаті отримаємо однакові матриці у двох верхніх позиціях стеку.

Іншою стековою функцією є

```
glPopMatrix();
```

Ця команда усуває матрицю у верхній позиції стеку, і поточною матрицею стає друга його матриця. Щоб виштовхнути вершину стеку, в ньому має бути не менше ніж дві матриці, інакше отримаємо помилку.

4.4. Контрольні питання

1. Яке перетворення називають ізометричним?
2. Якою матрицею і процедурою OpenGL описують зсув?
3. Якою матрицею і процедурою OpenGL описують обертання?
4. Якою матрицею і процедурою OpenGL описують масштабування?
5. Скільки ступенів свободи мають зсув, обертання?
6. Наведіть послідовність перетворень під час масштабування фігури відносно довільної нерухомої точки.
7. Наведіть послідовність перетворень під час обертання фігури навколо фіксованої точки у випадку збігання осі повороту з віссю z.
8. Наведіть послідовність перетворень під час обертання навколо осі, яка не паралельна жодній з координатних осей.

9. Яке перетворення називають скосом?
10. Дайте означення афінного перетворення.
11. Наведіть функції базових геометричних перетворень.
12. Для чого використовують процедуру **glMatrixMode**?
13. За допомогою яких функцій OpenGL задають поточну матрицю?
14. Для чого використовують функцію **glMultMatrix**?
15. Як записують матриці в OpenGL?
16. Поясніть, як можна зімітувати дзеркальне відображення фігури за допомогою певного виду обертання.
17. Для чого і як використовують стеки матриць?

4.5. Приклади тестових питань

1. Операція, яка зміщує точки на фіксовану відстань уздовж заданого напряму, – це:

   а) обертання;
   б) плоскопаралельне перенесення (зсув);
   в) масштабування.

2. Ізометричне перетворення:

   а) змінює форму об'єкта та його положення в просторі;
   б) не змінює ні форми об'єкта, ні його положення в просторі;
   в) не змінює форми об'єкта, а змінює тільки його положення в просторі.

3. Перетворення масштабування, за якого об'єкт стискається в заданому напрямі, має такий масштабний множник β:

   а) $0 \le \beta < 1$;
   б) $\beta > 1$;
   в) $\beta < 0$.

4. В OpenGL звертання до елемента матриці m_{jk} є:

   а) звертанням до елемента в рядку j і стовпці k;
   б) звертанням до елемента в стовпці j і рядку k;
   в) стандартним, як в лінійній алгебрі.

5. Перетворення зсуву має:

   а) 1 ступінь свободи;
   б) 2 ступені свободи;
   в) 3 ступені свободи.

6. Обертання навколо осі, яка не паралельна до жодної із координатних осей, з початком у довільній фіксованій точці має:

   а) 3 ступені свободи;
   б) 4 ступені свободи;
   в) 6 ступенів свободи.

7. Ізометричні перетворення – це:

   а) обертання, зсув, масштабування;
   б) обертання, зсув;
   в) масштабування, зсув;
   г) обертання, масштабування.

8. За допомогою процедури **glTranslate*(tx, ty, tz)** генерують матрицю:

   а) перенесення;
   б) обертання;
   в) масштабування.

9. За допомогою процедури **glRotate*(theta, vx, vy, vz)** генерують матрицю:

   а) перенесення;
   б) обертання;
   в) масштабування.

10. За допомогою процедури **glScale*(sx, sy, sz)** генерують матрицю

    а) перенесення;
    б) обертання;
    в) масштабування.

Розділ 5

ТРИВИМІРНЕ СПОСТЕРЕЖЕННЯ

Під час моделювання тривимірної сцени кожен об'єкт визначений набором поверхонь, які формують замкнутий контур навколо його внутрішньої частини. Крім того, в деяких застосуваннях потрібно задавати інформацію про внутрішню структуру об'єкта. Крім процедур, які генерують проекції деталей поверхні об'єкта, в графічних пакетах інколи є процедури виведення на екран внутрішніх компонент чи поперечних перерізів просторового об'єкта. Функції спостереження опрацьовують опис об'єкта за допомогою набору процедур, які проектують об'єкти на задану поверхню дисплея. Окрім того, треба визначати видимі частини сцени та враховувати ефекти освітлення і характеристики поверхонь, щоб зображення було реалістичним.

5.1. Тривимірний конвеєр спостереження

Процедури генерації комп'ютерних проекцій тривимірної сцени чимось подібні до дій фотографа. Насамперед треба вибрати *точку спостереження* (відповідає місцю, де розміщено камеру) відповідно до того, яка проекція сцени (фронтальна, задня, верхня чи нижня) потрібна. Місце спостереження може бути і всередині групи об'єктів чи навіть всередині одного об'єкта. Потім треба визначитися з орієнтацією камери: як вона має бути спрямована з точки фотографування і як повернути її навколо лінії спостереження, щоб задати верхнє розміщення зображення? Нарешті, коли натискаємо на кнопку, сцена обрізається до розміру вибраного відтинального вікна, яке відповідає апертурі чи типу лінз камери, і світло від видимих поверхонь проектується на плівку.

Однак слід пам'ятати, що аналогія з камерою справедлива лише частково, бо під час генерування проекцій сцени засобами програми КГ передбачає більшу гнучкість і вибір з набагато більшої кількості альтернатив, ніж у разі застосування реальної камери. Можна вибрати паралельну чи перспективну проекції, вибірково видаляти частини сцени вздовж лінії спостереження, площину проекції можна відсунути від розташування "камери", можна навіть отримати зображення об'єктів, які розташовані за "камерою".

Розглянемо загальні етапи обробки графічних даних під час створення і перетворення тривимірної сцени на координати пристрою.

У процесі створення й зображення сцени використовують декілька різних декартових координатних систем.

1. Форми окремих об'єктів задають в окремій СК для кожного об'єкта – такі СК називають *координатами моделювання, локальними* або *головними*. Задавши ці форми, можна скласти сцену, розмістивши об'єкти у відповідних місцях в СК сцени, яку називають *зовнішньою СК*. Цей етап потребує перетворення окремих СК моделювання на координати із заданим положенням і орієнтацією щодо зовнішньої СК. Зазначимо, що для опису не дуже складних сцен частину об'єктів можна вставляти безпосередньо до загальної структури сцени у зовнішніх координатах, пропускаючи етапи задання координат моделювання та їхнього перетворення на зовнішні.

2. Щоб створити зображення, загальний опис у зовнішніх координатах обробляють різними програмами в одній чи декількох СК пристроїв виведення – цей процес називають **конвеєром спостереження**. Спочатку зовнішні координати перетворюють на **координати спостереження**, які відповідають бажаному зображенню сцени. В основу цієї СК покладено положення та орієнтацію гіпотетичної камери.

3. Координати об'єкта перетворюють на двовимірну проекцію сцени, яка відповідає тому, що буде видно на пристрої виведення. Щоб перетворити опис сцени з координат спостереження на *координати проекції*, виконують *операції проектування*.

Рис. 5.1. Конвеєр загального тривимірного спостереження

4. Потім цю сцену записують у *нормованих координатах*, де значення кожної координати потрапляє в діапазон від –1 до 1 або від 0 до 1 залежно від системи. Ці координати ще називають *нормованими координатами пристрою,* оскільки такий опис робить графічний пакет незалежним від діапазону координат будь-якого спеціального пристрою виведення. Ще потрібно визначити видимі поверхні й обрізати частини рисунка, які виходять за межі поля огляду.

5. Кінцевим етапом є перетворення координат поля огляду на *координати пристрою* у вибраному вікні на екрані дисплея – розгортку рисунка записують у буфер регенерації растрової системи, щоб перетворити на зображення. СК пристрою зображення називають *координатами пристрою* або *екранними координатами* (у випадку монітора). Часто нормовані та екранні координати описують у лівосторонній СК, тоді збільшення додатної відстані від площини зображення інтерпретують як віддалення від точки спостереження.

На рис. 5.1 подано загальні етапи обробки під час створення і перетворення тривимірної сцени на координати пристрою.

5.2. Еталонна система спостережень

Під час формування тривимірної системи спостережень за початок координат вибирають точку із зовнішніми координатами $\mathbf{P}_0(x_0, y_0, z_0)$, яку називають *точкою спостереження*. Потім задають **вектор верху V (вектор вертикалі вигляду)**, який визначає додатний напрямок осі y_{view}. Із двох координатних осей, що залишились, вибирають вектор, який визначає вісь z_{view}, і вздовж цієї осі йде напрямок спостереження, тому *площину спостереження* (*площину проекції*) вибирають перпендикулярною до цієї осі (рис. 5.2). Отже, додатний напрямок осі z_{view} можна визначити як **вектор нормалі N** до площини спостереження, яка завжди буде паралельною до площини x_{view}, y_{view}. Часто напрямок вектора **N** вибирають у напрямку від опорної точки $\mathbf{P}_{ref}$ до початку координат $\mathbf{P}_0$, тоді точку $\mathbf{P}_{ref}$ називають *точкою погляду*.

Як вектор верху можна вибрати будь-який напрямок за умови, що він не паралельний **N**. Зручним є вибір напрямку паралельно до осі y_w зовнішньої СК, тобто можна покласти **V** = (0, 1, 0). Залишилось визначити тільки напрямок x_{view}.

Третій вектор **U**, обчислений як векторний добуток двох попередніх, є перпендикулярним до них і визначає додатний напрямок осі x_{view}. Напрямок **U** вибирають так, щоб у підсумку отримати правосторонню систему спостережень. Так одержують набір одиничних осьових векторів правосторонньої системи спостережень:

$$\mathbf{n} = \frac{\mathbf{N}}{|\mathbf{N}|} = (n_x, n_y, n_z),\ \mathbf{u} = \frac{\mathbf{V} \times \mathbf{n}}{|\mathbf{V}|} = (u_x, u_y, u_z),\ \mathbf{v} = \mathbf{n} \times \mathbf{u} = (v_x, v_y, v_z). \qquad (5.1)$$

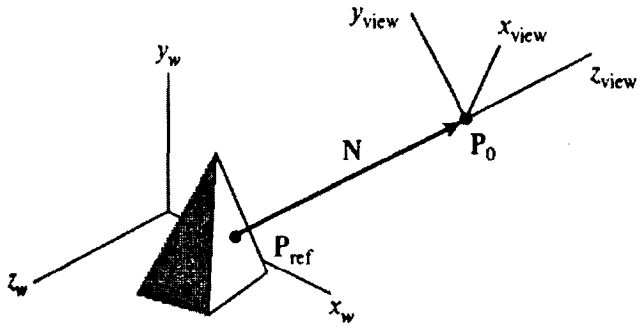


Рис. 5.2. Правостороння система спостереження

СК, сформовану цими одиничними векторами, часто називають *еталонною системою спостереження uvn.*

5.3. Перетворення із зовнішніх координат на координати спостереження

У тривимірному конвеєрі спостережень першою дією після побудови сцени є перенесення описів об'єктів у систему спостережень. Цей перехід виконують так.

1. Транслюють початок системи спостереження (точку з зовнішніми координатами $P_o(x_o, y_o, z_o)$) у початок зовнішньої СК за допомогою матриці

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & -x_o \\ 0 & 1 & 0 & -y_o \\ 0 & 0 & 1 & -z_o \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

2. За допомогою поворотів суміщають осі x_{view}, y_{view} і z_{view} з осями x_w, y_w і z_w зовнішньої СК відповідно за допомогою одиничних векторів **u, v** і **n** та матриці перетворення

$$\mathbf{R} = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

де елементи матриці – це компоненти осьових векторів **u, v** і **n.**

Матрицю перетворення координат одержують множенням наведених вище матриць трансляції і повороту:

$$\mathbf{M}_{W,V} = \mathbf{RT} = \begin{bmatrix} u_x & u_y & u_z & -x_o u_x - y_o u_y - z_o u_z \\ v_x & v_y & v_z & -x_o v_x - y_o v_y - z_o v_z \\ n_x & n_y & n_z & -x_o n_x - y_o n_y - z_o n_z \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

де параметри трансляції – це від'ємні проекції вектора $\mathbf{P_0}$, який йде з початку зовнішньої СК у початок координат системи спостереження, на всі координатні осі системи спостереження.

5.4. Перетворення проектування

На наступному етапі функціонування тривимірного конвеєра спостереження описи об'єктів проектують на площину спостереження.

У *паралельній проекції* точки переводять на площину спостереження вздовж паралельних ліній. Вона зберігає відносні пропорції об'єктів. Є два загальні методи одержання паралельної проекції об'єкта: можна проектувати вздовж ліній, перпендикулярних до площини спостереження, або під косим кутом до площини спостереження.

У випадку *перспективної проекції* положення об'єктів перетворюють на координати проекції вздовж ліній, які сходяться до точки за площиною спостереження. Такі проекції не зберігають відносних пропорцій об'єктів, однак є реалістичнішими, оскільки віддалені об'єкти після проектування виглядають меншими.

5.4.1. Ортогональні проекції

Перетворення описів об'єктів на площину спостереження вздовж ліній, паралельних до вектора нормалі **N** до площини спостереження, називають *ортогональною проекцією* (або *ортографічною*). Фронтальну, бокову і задню ортогональні проекції об'єкта називають *вертикальними*, верхню ортогональну проекцію – *горизонтальною*. Є й інші ортогональні проекції, на яких відображається *декілька граней* об'єкта. Їх називають *аксонометричними ортогональними проекціями*. Найпоширенішою з них є *ізометрична*; тоді площина проекції перетинає всі головні координатні осі на однаковій відстані від початку координат. В ізометричній проекції зберігаються відносні пропорції, для загальної аксонометричної проекції масштабні коефіцієнти можуть бути різними в трьох головних напрямках.

Коли напрямок проектування паралельний до осі z_{wiev}, рівняння перетворення ортогональної проекції прості:

$$x_p = x, y_p = y,\ z_p = 0.$$

Графічні пакети переважно дають змогу використовувати відтинальні прямокутні вікна стандартної орієнтації. У тривимірному спостереженні відтинальне вікно розміщується на площині спостереження, а його сторони паралельні до осей x_{view} і y_{view}. Сторони відтинального вікна задають межі зміни x та y і формують верхню, нижню та дві бокові сторони відтинальної області, яку називають *об'ємом спостереження ортогональної проекції*. Розмір ортогонального об'єму спостереження по осі z_{view} можна обмежити, вибравши положення однієї або двох

додаткових граничних площин, паралельних до площини спостереження. Їх називають *ближньою* і *дальньою* або *передньою* і *задньою площинами відтинання.*

Задавши межі об'єму спостереження, описи всередині цього прямокутного паралелепіпеда подають у координатах проекції, які можна відобразити в *нормований об'єм спостереження* (*канонічну зону видимості*) і завершити етапи обробки, пов'язані з проектуванням (рис. 5.3). Оскільки екранні координати часто задають у лівосторонній СК, нормовані координати задають так само.

Припустимо, що об'єм спостереження ортогональної проекції треба відобразити в симетричний нормований куб, розташований у лівосторонній СК. Крім того, координати *z* ближньої і дальньої площин записують як z_{near} і z_{far} відповідно. Щоб точка $(xw_{\min}, yw_{\min}, z_{near})$ відобразилась у точку з нормованими координатами (–1, –1, –1), а точка $(xw_{\max}, yw_{\max}, z_{far})$ – у точку (1, 1, 1), треба розв'язати дві задачі. Спочатку за допомогою перетворення зсуву змістимо центр заданого паралелепіпеда в центр куба – початок координат:

$$\mathbf{T}(-(xw_{\max} + xw_{\min})/2, -(yw_{\max} + yw_{\min})/2, -(z_{far} + z_{near})/2).$$

Потім змінимо масштабні коефіцієнти так, щоб кожне ребро паралелепіпеда мало довжину 2:

$$\mathbf{S}(2/(xw_{\max} - xw_{\min}), 2/(yw_{\max} - yw_{\min}), 2/(z_{far} - z_{near})).$$

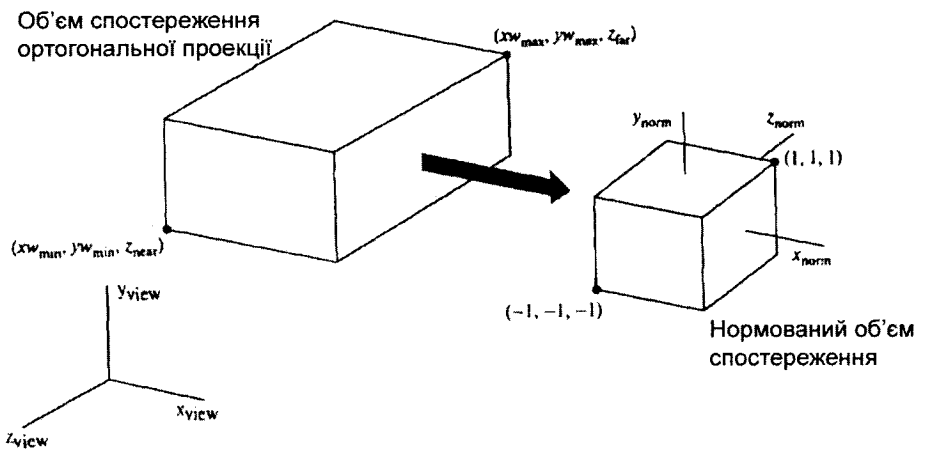


Рис. 5.3. Перетворення нормування з об'єму спостереження ортогональної проекції на симетричний нормований куб, визначений у лівосторонній системі координат

Матриця *нормування ортогонального об'єму* утворюється як суперпозиція цих матриць і може бути записана так:

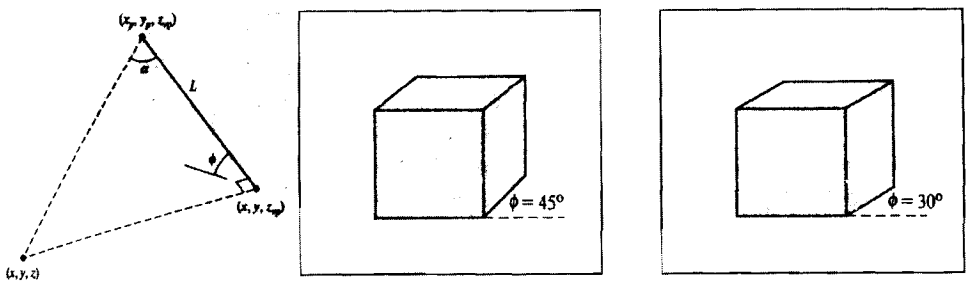
$$\mathbf{M}_{ort,norm} = \mathbf{ST} = \begin{bmatrix} \frac{2}{xw_{\max} - xw_{\min}} & 0 & 0 & -\frac{xw_{\max} + xw_{\min}}{xw_{\max} - xw_{\min}} \\ 0 & \frac{2}{yw_{\max} - yw_{\min}} & 0 & -\frac{yw_{\max} + yw_{\min}}{yw_{\max} - yw_{\min}} \\ 0 & 0 & \frac{2}{z_{far} - z_{near}} & -\frac{z_{near} + z_{far}}{z_{far} - z_{near}} \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

На цю матрицю справа множать матрицю $\mathbf{M}_{W,V}$, унаслідок чого знаходять повне перетворення зі зовнішніх координат на нормовані координати ортогональної проекції.

На цьому етапі конвеєра спостережень усі апаратно-незалежні перетворення координат завершено, і їх можна згорнути в одну складну матрицю. Тому процедури відтинання ефективніше виконати перед перетворенням нормування. Після відтинання можна викликати процедури перевірки видимості, візуалізації поверхонь і перетворення поля огляду, за допомогою яких генерують остаточне зображення сцени на екрані.

5.4.2. Косокутні паралельні проекції

Якщо лінії проектування не перпендикулярні до площини спостереження, таке відображення називають *косокутною паралельною проекцією*. Вона характеризується кутами θ і ϕ (рис. 5.4, *а*).



а *б* *в*

Рис. 5.4. Косокутні паралельні проекції: звичайна (а), кабінетні (б, в) для різних значень кута ϕ

З аналізу вигляду згори випливає, що x_p можна знайти за співвідношенням $tg\theta = \frac{z}{x - x_p}$, звідки матимемо $x_p = x - zctg\theta$. Аналогічно, аналізуючи вигляд збоку, одержимо $y_p = y - zctg\phi$. Отримані два рівняння подають перетворення скосу по

осі z, фактично вплив косокутної паралельної проекції полягає у зсуві площин сталої z і проектуванні їх на площину спостереження. Точки з координатами (x, y) на кожній площині сталої z зсуваються на величину, пропорційну до відстані до цієї площини від площини спостереження, тому кути, відстані і паралельні лінії на площині проектуються точно.

При $\theta = 90^\circ$ ($ctg\theta = 0$) маємо ортогональну проекцію.

Переважно кут ϕ вибирають у 30° чи 45°, при цьому отримують комбіноване відображення передньої, бокової і верхньої (чи передньої, бокової і нижньої) сторін об'єкта. Двома найпоширенішими значеннями θ є ті, для яких $tg\theta = 1$ і $tg\theta = 2$. У першому випадку $\theta = 45^\circ$, і отримані проекції називають *косоаксонометричними*; всі лініій, перпендикулярні до площини проекції, проектуються без змін довжини. Коли кут проекції θ вибрано так, що $tg\theta = 2$, її називають *кабінетною*. За такого кута (63.4°) лінії, перпендикулярні до площини проекції, проектуються у відрізки вдвічі меншої довжини. Кабінетні проекції видаються реалістичнішими, ніж косоаксонометричні, саме через це скорочення довжин перпендикулярів (рис. 5.4, *б, в*).

Враховуючи, що $z_p = 1$, запишемо матрицю перетворення однорідних координат:

$$\mathbf{M}_{oblique} = \begin{bmatrix} 1 & 0 & -ctg\theta & 0 \\ 0 & 1 & -ctg\phi & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \mathbf{M}_{ort}\mathbf{H}(\theta,\phi) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -ctg\theta & 0 \\ 0 & 1 & -ctg\phi & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

де $\mathbf{H}(\theta,\phi)$ є матрицею перетворення скосу. Отож, косокутне проектування можна подати як початковий скіс об'єктів, а потім виконання звичайного ортогонального проектування.

Щоб утворена після скосу зона видимості стала канонічною, треба скористатись перетвореннями масштабування і зсуву:

$$\mathbf{M}_{oblique,norm} = \mathbf{M}_{ort,norm}\mathbf{M}_{oblique} = \mathbf{STM}_{ort}\mathbf{H}(\theta,\phi).$$

На цю матрицю справа множать матрицю $\mathbf{M}_{W,V}$, внаслідок чого знаходять повне перетворення зі зовнішніх координат на нормовані координати косокутної паралельної проекції. Далі до нормованого об'єму спостереження можна застосувати процедури відтинання, визначити видимі об'єкти, виконати процедури візуалізації поверхонь і перетворення поля огляду.

5.4.3. Перспективні проекції

Принцип, покладений в основу реалізації перспективної проекції, такий самий, як і для паралельної: формують перетворення, яке призводить до такого деформування об'єктів, що після їхнього проектування стандартним способом

можна створити потрібне зображення. Спочатку треба вирішити, яку форму у випадку перспективної проекції має мати канонічна зона видимості, потім сформувати перетворення перспективної нормалізації, яке перетворить перспективну проекцію на ортогональну. Останній етап – виведення матриці перспективної проекції, яку й використовують в OpenGL.

Матриця *простого перспективного перетворення*, коли рівнянням площини спостереження є $z=-1$, а центр проекції поміщений у початок координат, має вигляд:

$$\mathbf{M}=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & -1 & 0\end{bmatrix}.$$

Для формування зображення нам треба знати параметри зони видимості. Допустимо, що вибраний кут зору 90°, тобто бокові грані зони видимості нахилені відносно площини спостереження на 45°. Зона видимості має вигляд піраміди, нескінченної в одному напрямку, грані якої – площини $x=\pm z$, $y=\pm z$. Цю зону можна обмежити, задавши передню і задню відтинальні площини $z=z_{near}$, $z=z_{far}$, де обидва значення від'ємні і $z_{far}<z_{near}$.

Розглянемо матрицю $\mathbf{N}=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & \alpha & \beta\\ 0 & 0 & -1 & 0\end{bmatrix}$, яка аналогічна $\mathbf{M}$, але є не виродженою, тобто $\alpha\neq 0, \beta\neq 0$. Якщо застосувати перетворення $\mathbf{N}$ до точки в однорідних координатах $\mathbf{p}=[x\ y\ z\ 1]^T$, то одержимо нову точку $\mathbf{q}=[x'\ y'\ z'\ h']^T$, де $x'=x, y'=y, z'=\alpha z+\beta, h'=-z$. Після ділення на h' одержимо координати точки в тривимірному просторі:

$$x''=-x/z, y''=-y/z, z''=-(\alpha+\beta/z).$$

Якщо застосувати до $\mathbf{N}$ перетворення ортогональної проекції вздовж осі z, одержимо матрицю:

$$\mathbf{M}_{ort}\mathbf{N}=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & -1 & 0\end{bmatrix},$$

яка і є матрицею простого перспективного перетворення на площину z=0. Проекція довільної точки **p** має вигляд:

$$\mathbf{p}' = \mathbf{M}_{орт}\mathbf{N}\mathbf{p} = \mathbf{M}_{орт}\mathbf{N}\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 0 \\ -z \end{bmatrix}.$$

Після виконання перспективного ділення одержимо шукані значення x_p, y_p: $x_p = -x/z, y_p = -y/z$. Отже, тут показано, що із застосуванням до довільної точки перетворення $\mathbf{N}$ її ортогональна проекція має такий самий вигляд, як і перспективна проекція вихідної точки. Тут є аналогія із застосуванням перетворення скосу на першому етапі зведення косокутної паралельної проекції до ортогональної.

Матриця $\mathbf{N}$ є не виродженою і перетворює вихідну зону видимості на нову. Параметри α і β можна вибрати так, щоб нова зона мала вигляд канонічної. Обмежувальні площини $x = \pm z$ (за допомогою $x'' = -x/z$) перетворюються на $x'' = \mp 1$, а площини $y = \pm z$ – на $y'' = \mp 1$. Передня і задня відтинальні площини зони видимості $z = z_{near}$, $z = z_{far}$ перетворюються на площини $z'' = -(\alpha + \beta / z_{near})$, $z'' = -(\alpha + \beta / z_{far})$.

Якщо вибрати значення цих параметрів відповідно до співвідношень:

$$\alpha = \frac{z_{near} + z_{far}}{z_{near} - z_{far}}, \beta = -\frac{2z_{far}z_{near}}{z_{near} - z_{far}},$$

то відображенням площин $z = z_{near}$, $z = z_{far}$ (обидва значення від'ємні і $z_{far} < z_{near}$) будуть площини $z'' = -1$, $z'' = 1$, тобто одержимо канонічну зону видимості.

Отже, $\mathbf{N}$ трансформує симетричну усічену піраміду видимості (рис. 5.5) у правильний паралелепіпед (рис. 5.6), а ортогональне перетворення трансформує його в канонічний куб. У підсумку ортогональна проекція здеформованого об'єкта матиме такий самий вигляд, як і перспективна проекція вихідного. Матрицю $\mathbf{N}$ називають *матрицею перспективної нормалізації.*

Якщо зона видимості не має форми симетричної (правильної) усіченої піраміди (рис. 5.7), то вона трансформується в симетричну з кутом нахилу бокових граней 45°. Кут скосу має бути таким, щоб точка $((x_{\min} + x_{\max})/2, (y_{\min} + y_{\max})/2, z_{near})$ зайняла положення $(0, 0, z_{near})$. Отже, матрицю скосу визначають так:

$$\mathbf{H}(\theta,\phi) = \mathbf{H}\left(arcctg\left(\frac{x_{\min} + x_{\max}}{2z_{near}} \right), arcctg\left(\frac{y_{\min} + y_{\max}}{2z_{near}} \right) \right).$$

Одержана в підсумку усічена піраміда видимості має такі площини граней:

$$x = \pm\frac{x_{\max} - x_{\min}}{2z_{near}},\ y = \pm\frac{y_{\max} - y_{\min}}{2z_{near}},\ z = z_{far},\ z = z_{near}.$$

Наступний етап – масштабування бокових граней піраміди так, щоб виконувались співвідношення $x = \mp z$, $y = \mp z$. При цьому положення передньої і задньої площин не змінюється. Таке перетворення виконується матрицею масштабування:

$$\mathrm{S}(-2z_{near}/(x_{\max} - x_{\min}), -2z_{near}/(y_{\max} - y_{\min}), 1).$$

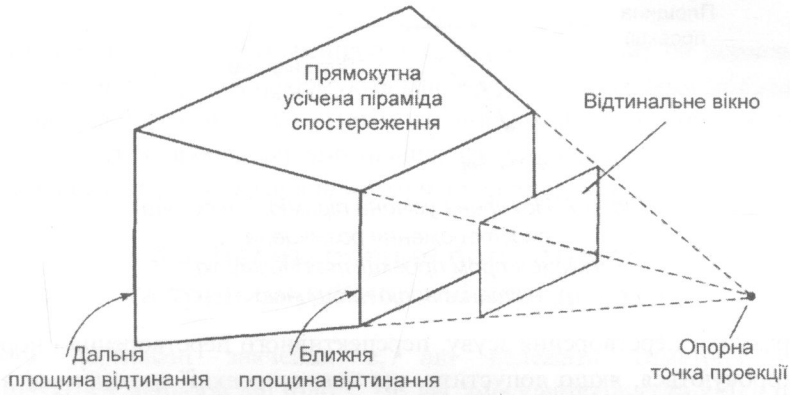


Рис. 5.5. Пірамідальний об'єм спостереження перспективної проекції, коли площину спостереження розміщено перед ближньою відтинальною площиною

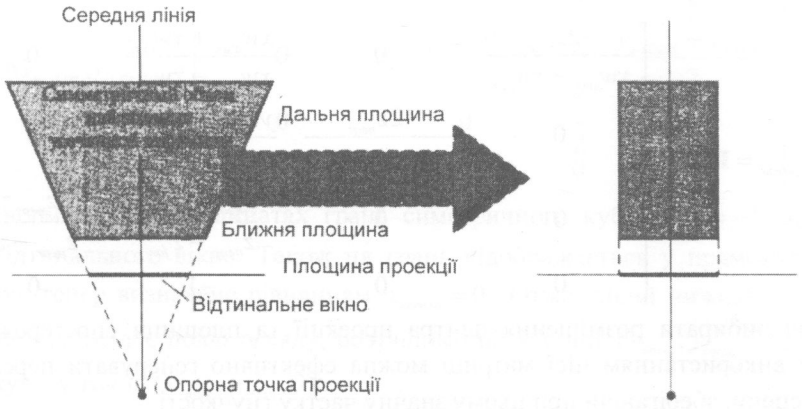


Рис. 5.6. Відображення об'єму спостереження симетричної усіченої піраміди перспективної проекції в ортогональний паралелепіпед

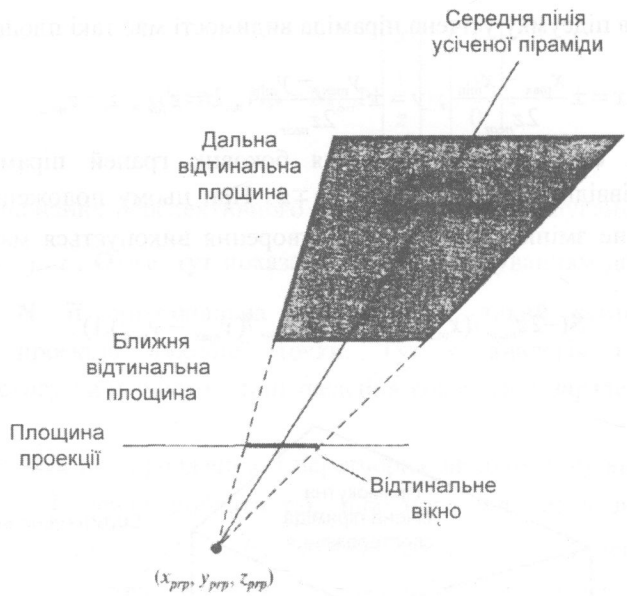


Рис. 5.7. Нахилена усічена піраміда, площина спостереження розміщена між центром проекції та ближньою відтинальною площиною

Розрахунки перетворення зсуву, перспективного перетворення і нормалізації істотно скорочуються, якщо допустити, що центр проекції знаходиться в початку системи координат, і площину спостереження розташовано на ближній площині відтинання.

Враховуючи нормалізацію проекції, одержимо вигляд *матриці косокутного перспективного перетворення*:

$$\mathbf{M}_{pers,norm} = \mathbf{PSH} = \begin{bmatrix} \frac{-2z_{near}}{xw_{\max} - xw_{\min}} & 0 & \frac{xw_{\max} + xw_{\min}}{xw_{\max} - xw_{\min}} & 0 \\ 0 & \frac{-2z_{near}}{yw_{\max} - yw_{\min}} & \frac{yw_{\max} + yw_{\min}}{yw_{\max} - yw_{\min}} & 0 \\ 0 & 0 & \frac{z_{near} + z_{far}}{z_{near} - z_{far}} & -\frac{2z_{near}z_{far}}{z_{near} - z_{far}} \\ 0 & 0 & -1 & 0 \end{bmatrix}.$$

Хоч вибирати розміщення центра проекції та площини спостереження не можна, з використанням цієї матриці можна ефективно генерувати перспективні проекції сцени, зберігаючи при цьому значну частку гнучкості.

Якщо об'єм спостереження перспективної проекції задано у вигляді симетричної усіченої піраміди ($xw_{\max} = -xw_{\min}$, $yw_{\max} = -yw_{\min}$), елементи нормованого

перспективного перетворення можна виразити через кут огляду θ між *верхньою і нижньою* площинами відтинання та розміри відтинального вікна:

$$\mathbf{M}_{pers,norm}^{sym} = \begin{bmatrix} \frac{ctg(\theta/2)}{a_s} & 0 & 0 & 0 \\ 0 & ctg(\theta/2) & 0 & 0 \\ 0 & 0 & \frac{z_{near}+z_{far}}{z_{near}-z_{far}} & -\frac{2z_{near}z_{far}}{z_{near}-z_{far}} \\ 0 & 0 & -1 & 0 \end{bmatrix},$$

де a_s – значення характеристичного відношення (ширина/висота) відтинального вікна.

Матрицю повного перетворення з зовнішніх координат на нормовані координати перспективної проекції формують множенням цієї матриці зліва на добуток перетворення спостереження **RT**. Потім до нормованого об'єму спостереження можна застосовувати процедури відтинання; нарешті, залишаться такі задачі, як визначення видимості, візуалізація поверхонь і перетворення поля огляду.

5.5. Перетворення поля огляду та тривимірні екранні координати

Часто нормовані значення z, що належать симетричному кубу, переформовують у діапазон від 0 до 1. Це дає змогу співвіднести z=0 з площиною екрана й обробку значень глибин виконати в одиничному інтервалі. Тоді матрицю перетворення можна записати так:

$$\mathbf{M}_{norm,3Dscreen} = \begin{bmatrix} \frac{x_{v\max}-x_{v\min}}{2} & 0 & 0 & \frac{x_{v\max}+x_{v\min}}{2} \\ 0 & \frac{y_{v\max}-y_{v\min}}{2} & 0 & \frac{y_{v\max}+y_{v\min}}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

У нормованих координатах грань симетричного куба $z_{norm} = -1$ відповідає області відтинального вікна. Також ця грань відображається у прямокутне вікно огляду, яке тепер визначене рівнянням $z_{screen} = 0$. Отже, лівий нижній кут області екрана, що відповідає полю огляду, розташований у точці $(x_{v\min}, y_{v\min}, 0)$, а правий верхній кут – у точці $(x_{v\max}, y_{v\max}, 0)$.

Кожна точка (x,y) в полі огляду відповідає позиції в буфері регенерації, який містить інформацію про колір цієї точки на екрані. Крім того, в іншій області буфера, яку називають *буфером глибини*, міститься інформація про глибини всіх

точок екрана. В наступних розділах розглянемо алгоритми визначення точок видимих поверхонь та їхніх кольорів.

Прямокутне поле огляду розміщується на екрані так: лівий нижній кут – у точці, заданій відносно лівого нижнього кута вікна на екрані. Крім того, щоб зберегти пропорції об'єкта, характеристичне відношення області поля огляду задають таким, як і для відтинального вікна.

5.6. Функції тривимірного спостереження OpenGL

Бібліотека GLU містить одну функцію для визначення параметрів тривимірного спостереження і ще одну – для задання симетричного перетворення перспективної проекції. Інші функції, наприклад, для ортогональної проекції, косокутної перспективної проекції і перетворення поля огляду, містяться в основній бібліотеці. Крім того, є функції GLUT для визначення вікон на екрані дисплея і роботи з ними (підрозділ 1.4.2).

5.6.1. Визначення параметрів спостереження

Для задання параметрів спостереження в OpenGL формують матрицю, яка згортається з поточною матрицею вигляду. Отже, ця матриця об'єднується з усіма заданими раніше геометричними перетвореннями. За дії цієї складної матриці описи об'єктів перетворюються з зовнішніх координат на координати спостереження. Режим роботи з цією матрицею задають командою

```
glMatrixMode(GL_MODELVIEW);
```

Параметри спостереження визначають за допомогою функції з бібліотеки GLU

```
gluLookAt(x0, y0, z0, xref, yref, zref, Vx, Vy, Vz);
```

яка викликає процедури зсуву і повороту з основної бібліотеки OpenGL.

Значення всіх параметрів цієї функції повинні бути задані як числа подвійної точності з плаваючою крапкою:

- $\mathbf{P}_0$ = (**x0**, **y0**, **z0**) – початок координат системи спостереження;
- $\mathbf{P}_{ref}$ = (**xref**, **yref**, **zref**) – опорна точка, її ще називають точкою погляду (look at point);
- **V** = (**Vx**, **Vy**, **Vz**) – координати вектора верху.

Додатний напрямок осі z_{view} системи спостереження йде вздовж напрямку $\mathbf{N} = \mathbf{P}_0 - \mathbf{P}_{ref}$, а одиничні осьові вектори системи спостереження обчислюють за рівняннями (5.1). За замовчуванням встановлено такі значення: $\mathbf{P}_0$ = (0, 0, 0), $\mathbf{P}_{ref}$ = (0, 0, –1), **V** = (0, 1, 0); за цих значень система спостереження збігається зі зовнішньою системою координат, а напрямок спостереження – з від'ємним напрямком її осі *z*.

Параметри спостереження, задані цією функцією, використовують для формування матриці перетворення спостереження $\mathbf{M}_{W,V}$.

5.6.2. Задання ортогональної та косокутної паралельних проекцій

Проекційні матриці записують у режимі проектування. Тому, щоб задати *матрицю проекції*, треба активувати цей режим командою

```
glMatrixMode(GL_PROJECTION);
```

Потім під час обробки будь-якої команди геометричного перетворення відповідна матриця буде згортатися з поточною проекційною матрицею.

Параметри *ортогональної проекції* визначають за допомогою функції

```
glOrtho(xmin, xmax, ymin, ymax, dnear, dfar);
```

Значення всіх аргументів цієї функції повинні бути задані як числа подвійної точності з плаваючою крапкою. Цю функцію використовують для вибору координат відтинального вікна (перші чотири параметри) і відстаней від початку системи спостережень до ближньої і дальньої площин відтинання (останні два). OpenGL **не дає змоги** вибирати положення площини спостереження, нею завжди є ближня площина відтинання. Параметрами **dnear** і **dfar** задають відстані вздовж від'ємного напрямку осі z_{view} від початку координат системи спостереження (наприклад, якщо **dfar** = 55.0, то ближня відтинальна площина проходить через точку $z_{far} = -55.0$). Можна присвоювати додатні, від'ємні (вони відповідають положенням "за" початком координат, проти додатного напрямку осі z_{view}) та нульові значення, головна умова – **dnear** < **dfar**.

Об'єктом спостереження є прямокутний паралелепіпед. Точки, що йому належать, за допомогою матриці $\mathbf{M}_{ort,norm}$, де $z_{near} = -$**dnear**, $z_{far} = -$**dfar**, переводять у точки всередині симетричного нормованого куба, визначеного в *лівосторонній* системі координат.

За замовчуванням значення параметрів цієї функції дорівнюють –1, +1 і дають об'єм симетричного нормованого куба, визначеного в *правосторонній* системі координат. Використання значень за замовчуванням рівносильне виклику команди:

```
glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
```

У двовимірних задачах для налаштування відтинального вікна використовують функцію

```
glOrtho2D(xmin, xmax, ymin, ymax);
```

яка фактично еквівалентна функції **glOrtho** з параметрами **dnear** = –1.0, **dfar** = 1.0.

Функції OpenGL для генерації косокутної проекції немає. Щоб отримати таку проекцію, можна задати власну проекційну матрицю, подібну до $\mathbf{M}_{oblique,norm}$. Потім треба зробити її поточною проекційною матрицею OpenGL, використовуючи функції роботи з матрицями для базових геометричних перетворень, розглянуті в підрозділі 4.2.2. Інший спосіб генерації косокутної проекції – так повернути сцену, щоб ортогональна проекція в напрямі z_{view} давала шукане зображення.

5.6.3. Одержання симетричної та довільної перспективних проекцій

Є **дві функції** для отримання перспективної проекції сцени. В обох центр проекції збігається з початком системи спостереження, а ближня площина відтинання – з площиною спостереження.

Одна з них генерує об'єм спостереження у вигляді **симетричної усіченої піраміди** у напрямі спостереження (від'ємний напрямок осі z_{view}):

```
gluPerspective(theta, aspect, dnear, dfar);
```

Значення всіх параметрів цієї функції повинні бути задані як числа подвійної точності з плаваючою крапкою.

Перші два визначають розмір і положення відтинального вікна на ближній площині, причому

- **theta** – кут огляду між *верхньою і нижньою* площинами відтинання (будь-яке значення від 0° до 180°),
- **aspect** – значення характеристичного відношення (ширина/висота) відтинального вікна.

Параметри **dnear** і **dfar** – відстані від початку координат до ближньої і дальньої площин відтинання, **обов'язково** додатні значення, бо жодна площина не може бути розміщена "за" точкою спостереження (це дає змогу уникнути виведення на екран перевернутої перспективної проекції об'єкта). Положення площин обчислюють так: $z_{near} = -\mathbf{dnear}$, $z_{far} = -\mathbf{dfar}$.

Для перетворення опису сцени в нормовані однорідні координати проекції використовують матрицю $\mathbf{M}^{sym}_{pers,norm}$.

Другу функцію можна використати для одержання **симетричної перспективної проекції** або **косокутної перспективної проекції**:

```
glFrustum(xmin, xmax, ymin, ymax, dnear, dfar);
```

вона має ті самі параметри, що й функція ортогональної паралельної проекції, але тепер відстані до ближньої і дальньої площин відтинання повинні бути **додатними** (їх задають уздовж від'ємного напрямку осі z_{viev}). Розташування площин обчислюємо так: $z_{near} = -\mathbf{dnear}$, $z_{far} = -\mathbf{dfar}$.

Вибравши координати відтинального вікна так:

xmin = –**xmax**, **ymin** = –**ymax**, –

одержимо *симетричну усічену піраміду*, середньою лінією якої є від'ємна вісь z_{viev}.

Якщо команди проектування не задано явно, OpenGL формує ортогональну проекцію сцени за замовчуванням.

5.6.4. Визначення поля огляду

Після завершення процедур відтинання в нормованих координатах вміст нормованого відтинального вікна та інформацію про глибину переводять у тривимірні екранні координати.

Прямокутне поле огляду визначають функцією

```
glViewport(xvmin, yvmin, vpWidth, vpHeight);
```

Перші два параметри задають точку екрана з цілочисловими координатами, яка є лівим нижнім кутом поля огляду відносно лівого нижнього кута вікна екрана. Останні два параметри задають цілочислову ширину і висоту поля огляду. Щоб зберегти пропорції об'єктів на сцені, потрібно, щоби характеристичне відношення поля огляду дорівнювало характеристичному відношенню відтинального вікна. За замовчуванням поле огляду має такі самі розміри і розташування, що й поточне вікно на екрані.

5.7. Контрольні питання

1. Опишіть загальні етапи реалізації тривимірного конвеєра спостереження.
2. Як формують еталонну систему спостереження?
3. За допомогою яких кроків здійснюють перетворення із зовнішніх координат у координати спостереження? Запишіть матрицю цього перетворення.
4. Дайте означення паралельної та перспективної проекцій. Опишіть відмінності між ними.
5. Як отримують матрицю нормування ортогонального об'єму?
6. За допомогою послідовності яких геометричних перетворень здійснюють косокутне паралельне проектування?
7. Запишіть матрицю перспективного перетворення для випадку, коли зона видимості не має форми симетричної (правильної) усіченої піраміди.
8. Запишіть матрицю перспективного перетворення для випадку, коли об'єм спостереження перспективної проекції заданий у вигляді симетричної усіченої піраміди.
9. Запишіть матрицю перетворення нормованих координат у прямокутне вікно огляду, визначене рівнянням $z_{screen} = 0$.
10. За допомогою якої функції визначають параметри спостереження? У якій бібліотеці вона розташована?

11. Які функції визначають параметри ортогональної проекції у дво- та тривимірному випадках?
12. Як в OpenGL можна згенерувати косокутну паралельну проекцію?
13. Назвіть функції, що генерують об'єм спостереження у вигляді симетричної усіченої піраміди.
14. Якою командою можна одержати косокутну перспективну проекцію?
15. Яку проекцію сцени OpenGL виконує за замовчуванням?
16. Якою функцією визначають прямокутне поле огляду?

5.8. Приклади тестових питань

1. Проекція, за якої положення об'єктів перетворюється на координати проекції вздовж ліній, які сходяться до точки за площиною спостереження, – це:

   а) ортогональна;
   б) перспективна;
   в) косокутна паралельна.

2. Перспективна проекція порівняно з паралельною є:

   а) менш реалістичною і не зберігає відносних пропорцій об'єктів;
   б) більш реалістичною і не зберігає відносних пропорцій об'єктів;
   в) є більш реалістичною і зберігає відносні пропорції об'єктів.

3. Для загальної аксонометричної проекції масштабні коефіцієнти є:

   а) однаковими в трьох головних напрямках;
   б) різними в трьох головних напрямках;
   в) різними в двох головних напрямках.

4. Ортогональна проекція OpenGL:

   а) не дозволяє вибирати положення площини спостереження, нею завжди є ближня площина відтинання;
   б) не дозволяє вибирати положення площини спостереження, нею завжди є дальня площина відтинання;
   в) дозволяє вибирати положення площини спостереження.

5. Матриця $\mathbf{M}_{W,V} = \mathbf{R}(\mathbf{u},\mathbf{v},\mathbf{n})\mathbf{T}(-\alpha_x,-\alpha_y,-\alpha_z)$ описує перетворення:

   а) проектування;
   б) нормування і відтинання;
   в) із зовнішніх координат у координати спостереження.

6. За допомогою функції **gluLookAt(x0, y0, z0, xref, yref, zref, Vx, Vy, Vz)** визначають параметри:

а) паралельної проекції;
б) перспективної проекції;
в) спостереження.

7. За допомогою функції **glOrtho(xmin, xmax, ymin, ymax, dnear, dfar)** визначають параметри:

а) ортогональної проекції;
б) перспективної проекції;
в) спостереження.

8. За допомогою функції **gluPerspective(theta, aspect, dnear, dfar)** визначають параметри перспективної проекції:

а) несиметричної;
б) симетричної;
в) косокутної.

9. За допомогою функції **glFrustum(xmin, xmax, ymin, ymax, dnear, dfar)** визначають параметри:

а) косокутної паралельної проекції;
б) паралельної проекції;
в) косокутної перспективної проекції.

10. За допомогою функції **glViewport(xvmin, yvmin, vpWidth, vpHeight)** визначають:

а) прямокутне поле огляду;
б) перспективну проекцію;
в) параметри спостереження.

11. За замовчуванням OpenGL формує проекцію сцени:

а) ортогональну;
б) перспективну;
в) косокутну.

Розділ 6

ІНТЕРАКТИВНЕ ВВЕДЕННЯ ДАНИХ

У графічних програмах використовують декілька типів вхідних даних, наприклад, координати, значення атрибутів, геометричних перетворень, умови спостереження і параметри освітлення. Багато графічних пакетів пропонують розширений набір функцій уведення, призначених для обробки таких даних. При цьому процедури введення потребують взаємодії з програмами керування вікнами і специфічними апаратними пристроями. Тому деякі графічні системи, що пропонують апаратно-незалежні функції, часто передбачають відносно мало інтерактивних процедур обробки вхідних даних.

Стандартний відхід до процедур уведення в графічних пакетах – це класифікація функцій відповідно до типу даних, які опрацьовуватиме кожна функція. Така схема дає змогу з будь-якого фізичного пристрою, такого як клавіатура чи миша, вводити дані будь-якого допустимого класу, хоча майже всі пристрої введення працюють з деякими типами даних краще, ніж іншими.

6.1. Логічна класифікація пристроїв уведення

У класифікації *функцій уведення* відповідно до типу даних будь-який пристрій, який використовують для введення даних, називають *логічним пристроєм уведення* для даних цього типу. Стандартна класифікація логічних пристроїв за вхідними даними така:

- **LOCATOR** – пристрій уведення однієї точки;
- **STROKE** – пристрій уведення набору точок;
- **STRING** – пристрій уведення рядків;
- **VALUATOR** – пристрій уведення скалярного значення;
- **CHOICE** – пристрій вибору опції з меню;
- **PICK** – пристрій вибору компоненти зображення.

Інтерактивний *вибір координати* переважно супроводжується розміщенням курсору в деякій точці сцени, при цьому можна використовувати мишу, джойстик, трекбол, спейсбол (просторовий маніпулятор), набірний диск чи цифрове перо. Крім того, задати умови обробки для вибраного положення можна за допомогою

різних клавіш, кнопок чи перемикачів. Клавіатуру використовують для переміщення курсору в різних напрямках та для набору значень чи інших кодів, що визначають координати.

Пристрої введення набору координат використовують для введення послідовності координат, зокрема, неперервний рух миші, джойстика, трекболу чи курсору перетворюється на набір вхідних координат. Одним із найпоширеніших є графічний планшет, який застосовують у технічних схемах для відстеження й оцифрування даних, а також у системах малювання для отримання мазків пензлем.

Основним фізичним *пристроєм*, який використовують *для введення рядків*, є клавіатура. У застосуваннях КГ рядки символів переважно використовують для позначок на рисунках чи графіках.

Фізичні пристрої, які використовують *для введення скалярних значень* (чисел), такі: панель набірних дисків, повзункові потенціометри, клавіатура. Незважаючи на те, що перші два зазначені пристрої є ефективнішими для швидкого введення чисел порівняно з клавіатурою, їх використовують рідше. Іншим методом уведення чисел є відображення на моніторі графічних зображень повзунків, кнопок, обертальних масштабних лінійок і меню.

У графічних програмах *для вибору* умов обробки, значень параметрів і форм об'єктів, які буде застосовано для побудови зображення, переважно використовують меню. До розповсюджених пристроїв вибору належать такі пристрої позиціонування курсору, як миша, трекбол, клавіатура, сенсорна панель чи поля клавіш. Альтернативні методи вибору входу передбачають уведення з клавіатури (при цьому корисним є деякий скорочений формат) і голосове введення (за невеликої кількості альтернатив, до 20).

Вибрати компоненти відображуваної сцени можна за допомогою різних методів, і будь-які пристрої, що дають змогу це зробити, належать до *указок*. Доволі часто для вибору орієнтації указки використовують курсор. За допомогою миші, джойстика чи клавіатури можна навести указку на потрібну точку і натиснути клавішу чи кнопку для запису координат пікселя. До інших методів вибору належать схеми підсвічування, вибір об'єктів за назвою чи комбінація інших підходів.

6.1.1. Режими введення графічних даних

Графічні пакети, які використовують логічну класифікацію пристроїв уведення, пропонують декілька функцій вибору пристроїв і класів даних. Ці функції дають користувачу можливість вибору з таких альтернатив.

- Режим вхідної взаємодії для графічних програм і пристроїв уведення. Елемент даних може ініціювати програма, пристрій або ж вони можуть працювати одночасно.
- Вибір фізичного пристрою, який забезпечуватиме введення згідно з визначеною логічною класифікацією.
- Вибір часу введення і пристрою для визначеного набору елементів даних.

Для визначення взаємодії програми і пристроїв уведення в інтерактивній графічній системі використовують різні режими. Програма чекає на введення даних у визначений час для обробки (режим запиту), пристрій уведення незалежно надає оновлені вхідні дані (дискретний режим) або пристрій незалежно записує всі зібрані дані (режим подій).

У **режимі запиту** введення даних ініціює програма. Обробка призупиняється, поки всі необхідні дані не буде отримано. Цей режим відповідає типовій операції введення у звичайній мові програмування. Програма і пристрої введення працюють позмінно. Пристрої переходять у режим очікування, поки не буде зроблено запит на введення; потім програма чекає, поки не буде доставлено даних.

У **дискретному режимі** програма і пристрої введення працюють незалежно. Пристрої введення можуть працювати в той самий час, коли програма опрацьовує інші дані. Нові значення, отримані з пристроїв уведення, заміняють попередні вхідні значення. Коли програмі потрібні нові дані, вона вибирає поточні значення, отримані за допомогою пристроїв і записані в пам'ять.

У **режимі подій** введення даних у програму ініціюють пристрої введення. Програма і пристрої знову працюють одночасно, але тепер пристрої ставлять вхідні дані в чергу, яку також називають *чергою подій*. Усі вхідні дані записують. Коли програмі потрібні нові дані, вона звертається до черги даних.

Переважно в останніх двох режимах може одночасно працювати будь-яка кількість пристроїв, причому деякі можуть працювати в дискретному, а інші – в режимі подій. Однак у режимі запиту в будь-який момент часу вхідні дані може надавати тільки один пристрій.

6.1.2. Зворотний зв'язок та зворотний виклик

Щоб задати фізичний пристрій для логічних класів даних, використовують функції з бібліотеки введення.

У програмі інтерактивного введення переважно можна виконувати запити й отримувати вхідні дані, які дублюються на екрані разом із параметрами, пов'язаними з цими запитами чи даними. Якщо це передбачено, необхідна інформація відображається в заданій області на екрані. Такий зворотний зв'язок з дублюванням може надавати, наприклад, інформацію про розмір вікна вибору і мінімальну відстань вибору, тип і розмір курсору, підсвічування, яке використовують в операціях вибору, діапазон (мінімальне і максимальне значення) і роздільну здатність (масштаб) пристрою введення.

Для апаратно-незалежних графічних пристроїв у допоміжній бібліотеці будь-якого графічного пакета можна знайти обмежений набір функцій уведення. У таких випадках процедури введення організовують як функції зворотного виклику, які взаємодіють з програмним забезпеченням системи. Ці функції задають, які дії повинна зробити програма після настання події введення. Типовими подіями введення є рух миші, натиснення кнопки на ній чи клавіші на клавіатурі.

6.2. Інтерактивні функції пристроїв уведення бібліотеки GLUT

Оскільки інтерактивне введення за допомогою пристроїв у програмі з використанням OpenGL потребує узгодження із системою вікон, таке введення обробляють процедури з бібліотеки GLUT. Там є функції для одержання вхідних даних від стандартних пристроїв, таких як миша чи клавіатура, а також від графічних планшетів, спейболів, полів клавіш і набірних дисків.

Для кожного пристрою задають процедуру (функцію зворотного виклику), яка викликатиметься щоразу після настання події введення від цього пристрою. Відповідні команди розміщають у *головній* процедурі разом з іншими операторами GLUT. Крім того, об'єднуючи функції з основної бібліотеки і бібліотеки GLUT, можна реалізувати обробку за допомогою миші даних, уведених з використанням указки.

6.2.1. Миша

Щоб зареєструвати процедуру, яку буде викликано, коли вказівник миші розташований у вікні на екрані дисплея і натиснуто чи відпущено клавішу миші, використовують команду

```
glutMouseFunc(mouseFcn);
```

Це функція зворотного виклику процедури **mouseFcn** із 4-ма аргументами:

```
void mouseFcn(GLint button, GLint action, GLint xMouse,
     GLint yMouse);
```

параметр **button** повертає значення символьної константи GLUT, яка позначає одну з трьох кнопок миші, що була натиснута (допустимими значеннями **button** є **GLUT_LEFT_BUTTON**, **GLUT_MIDDLE_BUTTON** і **GLUT_RIGHT_BUTTON**), а параметру **action** система присвоює значення символьної константи, що визначає, яка подія від миші активізує виклик функції **mouseFcn**. Для двокнопкової миші застосовують лише позначення для лівої і правої кнопок, для однокнопкової – тільки лівої.

Подію, пов'язану з параметром **action**, визначають його значення, наприклад, **GLUT_DOWN** чи **GLUT_UP**, тобто натиснуто чи відпущено відповідну клавішу (подій від миші є багато, подвійний клік, наведення вказівника тощо). Під час виклику цієї процедури положення курсору миші у вікні на екрані дисплея повертається як координата (**xMouse**, **yMouse**), де **xMouse**, **yMouse** – це відстані в пікселях відповідно по горизонталі та вертикалі від верхнього лівого кута вікна.

Натискаючи кнопку миші, коли курсор розташований у межах вікна на екрані дисплея, можна вибрати точку для відображення примітиву, такого як окрема точка, відрізок чи зафарбована область. Крім того, мишу можна використовувати як

вказівний пристрій, порівнюючи повернену координату з межами відображених об'єктів на сцені (OpenGL пропонує процедури використання миші як вказівного пристрою).

Інша команда в GLUT, яку можна використати для роботи з мишею, –

```
glutMotionFunc(fcnDoSomething);
```

вона викликає функцію **fcnDoSomething** у випадку руху курсору миші у вікні з активацією однієї чи декількох клавіш миші; функція має 2 аргументи:

```
void fcnDoSomething(GLint xMouse, GLint yMouse);
```

тут (**xMouse**, **yMouse**) – положення курсору миші відносно верхнього лівого кута вікна.

Подібно можна виконати деякі дії під час руху мишею в межах вікна без клацань:

```
glutPassiveMotionFunc(fcnDoSomething);
```

6.2.2. Клавіатура

Для задання процедури, яку буде викликано після натиснення клавіші, використовують оператор

```
glutKeyboardFunc(keyFcn);
```

задана в ній функція має 3 аргументи:

```
void keyFcn(GLubyte key, GLint xMouse, GLint yMouse);
```

параметр **key** повертає символьне значення чи відповідний код ASCII натиснутої клавіші; положення курсору миші у вікні на екрані дисплея повертається як точка (**xMouse**, **yMouse**) відносно верхнього лівого кута вікна.

Натисненням означеної клавіші на основі положення курсору миші можна ініціювати деяку дію, незалежно від того, чи натиснуто ще якусь кнопку миші.

Для функціональних клавіш, клавіш керування курсором та інших спеціальних клавіш можна використовувати команду

```
glutSpecialFunc(specialKeyFcn);
```

задана в ній процедура має 3 аргументи:

```
void specialKeyFcn(GLint specialKey, GLint xMouse, GLint
yMouse);
```

однак тепер параметр **specialKey** повертає цілочислове значення символьної константи GLUT, що відповідає натиснутій клавіші.

Щоб визначити натиснуту функціональну клавішу, використовують одну з констант від **GLUT_KEY_F1** до **GLUT_KEY_F12**. Для клавіш керування курсором система повертає такі константи, як **GLUT_KEY_UP**,

GLUT_KEY_RIGHT, **GLUT_KEY_PAGE_DOWN**, **GLUT_KEY_HOME** тощо. Однак якщо натиснуто одну з клавіш **Backspace**, **Delete** і **Escape**, потрібна процедура **glutKeyboardFunc()** і коди ASCII цих клавіш – 8, 127 і 27 відповідно.

6.2.3. Графічний планшет

Зазвичай активізацію планшета виконують, коли курсор миші розташований у вікні дисплея. Потім, використовуючи наступну команду, записують подію для введення з графічного планшета

```
glutTabletButtonFunc(tabletFcn);
```

аргументи функції, яку буде викликано, подібні до тих, які використовують для роботи з мишею:

```
void tabletFcn(GLint tabletButton, GLint action, GLint
xTablet,
   GLint yTablet);
```

Натиснуту клавішу планшета **tabletButton** буде повернуто цілим ідентифікатором 1, 2, 3 і т.д., подію від клавіші **action** – константами **GLUT_UP** чи **GLUT_DOWN**. Повернені значення **xTablet** і **yTablet** є планшетними координатами. Кількість доступних кнопок планшета можна визначити командою

```
glutDeviceGet(GLUT_NUM_TABLET_BUTTONS);
```

Для обробки руху пера чи курсору по планшету використовують команду

```
glutTabletMotionFunc(tabletMotionFcn);
```

тут викликана функція має вигляд

```
void tabletMotionFcn(GLint xTablet, GLint yTablet);
```

повернені значення **xTablet** і **yTablet** дають координати пера чи курсору на поверхні планшета.

6.2.4. Спейсбол

Щоб задати операцію у вибраному вікні під час активізації клавіші спейсболу (просторового маніпулятора), використовують процедуру

```
glutSpaceballButtonFunc(spaceballFcn);
```

функція зворотного виклику має 2 параметри:

```
void spaceballFcn(GLint spaceballButton, GLint action);
```

натиснуті кнопки спейсболу визначають за допомогою таких самих цілих значень, як і для планшета, а параметр **action** повертає значення **GLUT_UP** чи **GLUT_DOWN**. Кількість доступних клавіш спейсболу можна визначити, викликавши процедуру

```
glutDeviceGet(GLUT_NUM_SPACEBALL_BUTTONS);
```

Паралельне перенесення спейсболу визначають за допомогою команди

```
glutSpaceballMotionFunc(spaceballTranslFcn);
```

тривимірні відстані буде передано викликаній функції

```
void spaceballTranslFcn(GLint tx, GLint ty, GLint tz);
```

причому ці відстані нормуються в діапазоні від –1000 до 1000.

Аналогічно *обертання* спейсболу визначають за допомогою процедури

```
glutSpaceballRotateFunc(spaceballRotFcn);
```

використовуючи функцію зворотного виклику, можна отримати кути тривимірного обертання:

```
void spaceballRotFcn(GLint thetaX, GLint thetaY, GLint
thetaZ);
```

6.2.5. Поле клавіш

Дані, введені з поля клавіш, отримують за допомогою оператора

```
glutButtonBoxFunc(buttonBoxFcn);
```

та функції зворотного виклику

```
void buttonBoxFcn(GLint button, GLint action);
```

для ідентифікації кнопок використовують цілі значення **button**, а дії, що відбуваються з кнопкою, визначають за допомогою констант **GLUT_UP** чи **GLUT_DOWN**.

6.3. Функції меню OpenGL

На доповнення до процедур роботи з пристроями введення GLUT містить функції для додавання до програм простих спливаючих меню. За допомогою цих функцій можна налаштувати різні меню та пов'язані з ними підменю і звертатись до них. Команди меню GLUT розміщають у процедурі **main** разом з іншими функціями GLUT.

6.3.1. Створення меню GLUT

Спливаюче меню створюють за допомогою оператора

```
glutCreateMenu(menuFcn);
```

тут параметр – це ім'я процедури, яку буде викликано в разі вибору певної опції меню:

```
void menuFcn(GLint menuItemNumber);
```

Аргумент цієї процедури – ціле значення, яке відповідає розташуванню вибраної опції; його використовує функція **menuFcn** для виконання деякої дії. Коли меню створюють, воно співвідноситься з поточним вікном на екрані дисплея.

Далі треба задати опції, які будуть перераховані в меню. Для цього використовують низку операторів, у яких зазначають ім'я і розташування кожної опції. Ці оператори мають вигляд

```
glutAddMenuEntry(charString, menuItemNumber);
```

параметр **charString** задає текст, який відображатиметься у меню, а параметр **menuItemNumber** дає розташування цієї позиції в меню.

Приклад створення меню з двома опціями:

```
glutCreateMenu(menuFcn);
glutAddMenuEntry("Перший пункт меню", 1);
glutAddMenuEntry("Другий пункт меню", 2);
```

Далі треба задати кнопку миші, яку буде використано для вибору опції меню, її задають оператором

```
glutAttachMenu(button);
```

тут параметру **button** присвоюють одну з 3 символьних констант GLUT, співвіднесених з лівою, середньою чи правою кнопками миші.

6.3.2. Створення декількох меню GLUT та керування ними

Як уже зазначено, створюване меню співвідноситься з поточним вікном на екрані дисплея. Для одного вікна можна створити декілька меню, а також передбачити різні меню для різних вікон. Кожному створюваному меню буде присвоєно цілочисловий ідентифікатор, починаючи зі значення 1, співвіднесеного з першим меню. Цілочисловий ідентифікатор меню повертає процедура **glutCreateMenu**, і це значення можна зберегти, використовуючи оператор

```
menuID = glutCreateMenu(menuFcn);
```

Щойно що створене меню стає *поточним меню* для поточного вікна на екрані дисплея. Щоб активувати інше меню поточного вікна, використовують оператор:

```
glutSetMenu(menuID);
```

після чого активоване меню стає поточним і спливатиме у вікні після натиснення кнопки миші, співвіднесеної з цим меню.

Для видалення меню використовують команду

```
glutDestroyMenu(menuID);
```

якщо видалене меню було поточним для вікна, цей статус не переходить на жодне з решти меню (тобто в цьому вікні не буде поточного меню).

Наведену далі функцію використовують для отримання ідентифікатора поточного меню в поточному вікні екрана:

```
currentMenuID=glutGetMenu();
```

значення 0 буде повернуто, якщо вікно не має меню або якщо попереднє поточне меню було видалено за допомогою функції **glutDestroyMenu**.

6.3.3. Створення підменю GLUT

Підменю можна співвіднести з меню, спочатку створивши підменю зі списком опцій за допомогою команди **glutCreateMenu**, а потім перерахувавши підменю як додаткові опції в основному меню. Щоб додати підменю до списку опцій основного меню (чи іншого підменю), використовують таку послідовність операторів:

```
submenuID = glutCreateMenu(submenuFcn);
glutAddMenuEntry("Перший пункт підменю", 1);
   ...
glutCreateMenu(menuFcn);
glutAddMenuEntry("Перший пункт меню", 1);
   ...
glutAddSubMenu("Опції підменю", submenuID);
```

Щоб додати підменю до поточного меню, можна також використати функцію **glutAddSubMenu**.

6.3.4. Модифікація меню GLUT

Якщо треба замінити кнопку миші, яку використовують для вибору опції меню, спочатку треба відмінити поточне прив'язання до неї:

```
glutDetachMenu(mouseButton);
```

параметру **mouseButton** присвоюють константу GLUT, яка визначає ліву, праву чи середню кнопку, раніше прив'язану до меню; потім з опцією пов'язують нову кнопку.

Уже створене меню можна модифікувати. Наприклад, для видалення опції з меню використовують команду

```
glutRemoveMenuItem(itemNumber);
```

тут параметру присвоюють ціле значення опції меню, яку треба видалити.

Інші процедури GLUT дають змогу модифікувати імена чи стани елементів наявного меню. Наприклад, їх можна використовувати, щоб змінити відображене ім'я опції меню, номер пункту чи опцію в підменю.

6.4. Контрольні питання

1. Назвіть логічні пристрої для введення однієї точки та набору точок. Які фізичні пристрої їм відповідають?
2. Назвіть логічні пристрої для введення рядків та скалярних значень і фізичні пристрої, що їм відповідають.
3. Назвіть логічні пристрої для вибору опції з меню та компоненти зображення. За допомогою яких фізичних пристроїв це здійснюють?
4. Опишіть режими введення графічних даних.
5. Як працюють програма і пристрій введення у режимі запиту?
6. Як працюють програма і пристрої введення у дискретному режимі?
7. Що називають чергою подій і в якому режимі введення даних її використовують?
8. Яку роль відіграють у програмах функції зворотного виклику?
9. Які три команди з GLUT можна використати для роботи з мишею? Опишіть функції зворотного виклику цих процедур.
10. Які команди використовують для роботи з клавіатурою, зокрема для функціональних клавіш, клавіш керування курсором та інших спеціальних клавіш? Опишіть функції зворотного виклику цих процедур.
11. Які команди та функції зворотного виклику використовують для обробки даних, отриманих з планшета, руху пера чи курсору по ньому?
12. Як в OpenGL описують використання спейсболу, зокрема його паралельне перенесення та обертання?
13. За допомогою якого оператора та якої функції зворотного виклику отримують дані, введені з поля клавіш?
14. Яка бібліотека містить функції для додавання до програм простих спливаючих меню?
15. Як створюють спливаюче меню?
16. Якими операторами задають команди, перелічені у меню, та кнопку миші, яку буде використано для вибору команди меню?
17. Якими командами створюють декілька меню?
18. За допомогою яких процедур створюють підменю?
19. Як модифікують і видаляють меню?

6.5. Приклади тестових питань

1. Логічний пристрій уведення **STRING** – це пристрій уведення:

   а) однієї точки;
   б) набору точок;
   в) рядків.

2. Логічний пристрій уведення **VALUATOR** – це пристрій:

   а) уведення скалярного значення;
   б) вибору опції з меню;
   в) вибору компоненти зображення.

3. Логічний пристрій уведення **CHOICE** – це пристрій:

   а) уведення скалярного значення;
   б) вибору опції з меню;
   в) вибору компоненти зображення.

4. Логічний пристрій уведення **PICK** – це пристрій:

   а) уведення скалярного значення;
   б) вибору опції з меню;
   в) вибору компоненти зображення.

5. Логічний пристрій уведення **STROKE** – це пристрій уведення:

   а) однієї точки;
   б) набору точок;
   в) рядків.

6. Логічний пристрій уведення **LOCATOR** – це пристрій уведення:

   а) однієї точки;
   б) набору точок;
   в) рядків.

7. Найефективнішим фізичним пристроєм, який використовують для уведення скалярних значень, є:

   а) клавіатура;
   б) панель набірних дисків;
   в) графічний планшет.

8. Одним з найпоширеніших пристроїв уведення набору координат є:

   а) клавіатура;
   б) спейсбол;
   в) графічний планшет.

9. У дискретному режимі програма і пристрої введення працюють:

   а) позмінно;
   б) незалежно;
   в) одночасно, причому пристрої введення ставлять дані у чергу.

10. У режимі подій програма і пристрої введення працюють:

    а) позмінно;
    б) незалежно;
    в) одночасно, причому пристрої введення ставлять дані в чергу.

11. У режимі запиту в будь-який момент часу ввід може надавати:

    а) декілька пристроїв;
    б) тільки один пристрій;
    в) два пристрої.

Розділ 7

ПОДАННЯ ТРИВИМІРНИХ ОБ'ЄКТІВ

Схеми подання суцільних об'єктів часто поділяють на дві категорії, хоча не всі подання можна беззаперечно зарахувати до котроїсь із них. *Контурні подання* (*boundary representations, B-rep*) описують тривимірний об'єкт як набір поверхонь, що відділяють внутрішню частину об'єкта від середовища. Типові приклади контурних подань – опис через грані багатокутника сплайнових ділянок. *Подання з розбиттям простору* (*space-partitioning representations*) дають змогу описувати властивості через розбиття внутрішньої області об'єкта на набір невеликих неперервних об'ємних тіл (переважно кубів), що не перетинаються.

7.1. Багатогранники

Найпоширенішим контурним поданням тривимірного графічного об'єкта є визначення набору поверхонь-багатокутників, що відмежовують його внутрішню частину. Багато графічних систем зберігають всі описи об'єктів саме у формі наборів багатокутних поверхонь. Це спрощує і прискорює візуалізацію поверхні і відображення об'єктів, бо всі поверхні описано лінійними рівняннями. Саме тому такі описи часто називають *стандартними графічними об'єктами*. Однак багато пакетів дають змогу описувати поверхні об'єктів за допомогою інших схем, наприклад, з використанням сплайнових поверхонь, які зазвичай перетворюються на багатокутні подання для опрацювання в конвеєрі спостереження.

Багатокутні поверхні в програмі OpenGL можна задати двома способами. За допомогою багатокутних примітивів, розглянутих у підрозділі 2.1.3, можна згенерувати різні форми багатокутників і розбиттів поверхонь. Крім того, можна використовувати функції GLUT виведення на екран п'яти правильних багатогранників.

7.1.1. Функції заповнення області багатокутника

Щоб ввести набір багатокутних ділянок для фрагмента поверхні об'єкта чи повний опис багатогранника, можна використати константи (примітиви) **GL_POLYGON, GL_TRIANGLES, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN, GL_QUADS, GL_QUAD_STRIP**. Так можна, наприклад, подати бокову

поверхню циліндра у вигляді мозаїчної структури за допомогою чотирикутної смуги. Аналогічно всі сторони паралелограма можна описати набором багатокутників, а всі грані трикутної піраміди задати набором зв'язаних трикутних поверхонь.

7.1.2. Функції GLUT правильних багатогранників

У процедурах бібліотеки GLUT визначено деякі стандартні форми (п'ять правильних багатогранників – *тіл Платона*). Їх особливістю є те, що всі їхні грані є однаковими правильними багатокутниками. Отже, всі сторони правильного багатогранника рівні, всі плоскі кути при вершинах рівні, всі кути між гранями також рівні. Багатогранники називають за кількістю граней: правильний тетраедр (трикутна піраміда, 4 грані), правильний гексаедр (куб, 6 граней), правильний октаедр (8 граней), правильний додекаедр (12 граней) і правильний ікосаедр (20 граней).

Бібліотека GLUT пропонує 10 функцій генерації зазначених тіл: 5 функцій дають каркасні об'єкти, 5 відображають грані багатогранників як затінені зафарбовані області. Характеристики зафарбовуваної області визначаються властивостями матеріалу та умовами освітлення, заданими для сцени. Усі правильні багатогранники описують у модельних координатах, так що всі вони центровані в початку зовнішньої системи координат.

Щоб отримати чотиригранну правильну трикутну піраміду, використовують одну з двох функцій:

```
glutWireTetrahedron();
```

або

```
glutSolidTetrahedron();
```

центр цього багатогранника збігається з початком зовнішньої системи координат, а радіус (відстань від центра тетраедра до будь-якої вершини) становить $3^{1/2}$.

Шестисторонній правильний гексаедр (куб) відображають за допомогою функцій

```
glutWireCube(edgeLength);
```

або

```
glutSolidCube(edgeLength);
```

параметру **edgeLength** можна присвоювати будь-яке додатне значення подвійної точності з плаваючою крапкою, а центр куба розташований у початку координат.

Щоб відобразити восьмигранний правильний октаедр, викликають одну з двох команд:

```
glutWireOctahedron();
```

або

```
glutSolidOctahedron();
```

цей багатогранник має рівносторонні трикутні грані, а його радіус (відстань від центра октаедра до будь-якої вершини) дорівнює 1.

Дванадцятигранний додекаедр, центрований у початку глобальної системи координат, генерують командою

```
glutWireDodecahedron();
```

або

```
glutSolidDodecahedron();
```

кожна грань цього багатогранника – п'ятикутник.

Наступні дві функції генерують двадцятигранний правильний ікосаедр:

```
glutWireIcosahedron();
```

або

```
glutSolidIcosahedron();
```

за замовчуванням радіус (відстань від центра багатогранника в початку координат до будь-якої його вершини) дорівнює 1, а всі його грані – рівносторонні трикутники.

7.2. Криволінійні поверхні

Рівняння об'єктів з криволінійними межами можна подати в параметричній або непараметричній формі. До об'єктів, які використовують у графічних пакетах, належать поверхні другого роду (квадрики), суперквадрики, поліноміальні й показникові функції, сплайнові криві та поверхні.

Квадрики описують рівняннями другого порядку (квадратними) – це сфери, еліпсоїди, тори, параболоїди й гіперболоїди. Об'єкти перших двох типів часто присутні у графічних сценах, тому графічні пакети містять процедури для їхньої генерації. Крім того, поверхні другого порядку можна отримати за допомогою раціональних сплайнів.

Суперквадрики є узагальненням поверхонь другого порядку. Для їхнього отримання до рівняння поверхні другого порядку вводять додаткові параметри, що дає більшу гнучкість у налаштуванні форм об'єктів. У рівняння кривих вводять один додатковий параметр, а в рівняння поверхонь – два.

Сферу і багато інших тривимірних об'єктів з поверхнями другого порядку можна вивести на екран, використовуючи функції, включені в набори GLUT і GLU. Крім того, GLUT містить одну функцію для відображення форми чайника, визначеного за допомогою бікубічних ділянок поверхні. Усі функції GLUT, які легко під'єднати до прикладної програми, мають дві версії. Один варіант кожної функції відображає каркасну поверхню, інший подає поверхню як візуалізований набір багатокутних зафарбованих ділянок. За допомогою функцій GLUT можна відобразити сферу, конус, тор або чайник. Функції GLU поверхонь другого порядку

трохи складніше налаштувати, але вони дають більше альтернатив – сферу, циліндр, конічний циліндр, конус, плоске кругове кільце (або шайбу) і ділянку кругового кільця (або диска).

7.2.1. Непараметричні і параметричні подання

Якщо характеристики об'єкта записують безпосередньо через координати використовуваної системи, подання називають *непараметричним*. Наприклад, *поверхню* можна описати однією з функцій у декартовій системі координат:

$$f(x,y,z)=0 \text{ або } z=f(x,y).$$

Першу форму називають *неявним* виразом для поверхні, другу – *явним поданням* (тоді x і y називають незалежними, а z – залежною змінною).

Подібно *тривимірну криву* можна подати в непараметричній формі як перетин двох поверхонь або системою з двох рівнянь:

$$f(x,y,z)=0,\ g(x,y,z)=0 \text{ або } y=f(x),\ z=g(x),$$

де координата x є незалежною змінною.

Непараметричні подання корисні для опису об'єктів у визначеній системі відліку, але вони мають деякі недоліки в разі використання у графічних алгоритмах. Якщо потрібен гладкий графік, незалежну змінну треба змінювати в той момент, коли перша похідна $f(x)$ чи $g(x)$ стає більшою за 1. Це потребує постійної перевірки значень похідних, щоб визначити, коли треба поміняти місцями незалежну і залежну змінні. Крім того, явна форма тривимірних кривих дає громіздкий (незручний) формат подання багатозначних функцій.

І в дво-, і в тривимірному просторі *не всі криві та поверхні* можна описати рівняннями в явному вигляді, зокрема, так не можна описати прямі, що лежать у вертикальній площині $x=const$, або коло чи сферу, оскільки заданим значенням x або x, y відповідають дві точки на колі чи сфері. Це є серйозним **недоліком** такої форми зображення.

Приклади неявного опису прямої, кола, площини і сфери:

$$ax+by+c=0;\ x^2+y^2-r^2=0;\ ax+by+cz+d=0;\ x^2+y^2+z^2-r^2=0;$$

де a, b, c, d, r – константи.

Неявна форма зображення є менш залежною від системи координат, оскільки дає змогу подати прямі і кола в усіх варіантах. У загальному випадку, однак, неявна функція **не дає можливості** визначити значення координати y точки на кривій для заданого значення x чи навпаки.

Поверхню, для якої функція $f(x,y,z)$ є сумою поліномів трьох змінних, вважають *алгебраїчною*. Частковий випадок алгебраїчної поверхні – *квадратичні поверхні (квадрики)*, в яких у функції f відсутні степені змінних вище 2; вони перетинаються прямою не більше ніж у двох точках.

У графічних алгоритмах описи об'єктів зручніше давати у формі параметричних рівнянь.

Об'єкти можна класифікувати за кількістю параметрів, необхідних для опису їхніх координат. Криву, наприклад, у декартовій системі відліку класифікують як одновимірний, а поверхню – як двовимірний евклідовий об'єкт. Опис об'єкта через його розмірність називають *параметричним поданням.*

У *параметричній формі* значення кожної координати точки, що належить кривій, зображають функцією незалежної змінної u, яку називають *параметром* цієї кривої. У тривимірному просторі *криву* описують системою з трьох параметричних рівнянь: $x = x(u)$, $y = y(u)$, $z = z(u)$. Для опису *поверхні* треба використати два параметри, система рівнянь поверхні має вигляд $x = x(u,v)$, $y = y(u,v)$, $z = z(u,v)$.

Одна з головних **переваг** цієї форми – її однаковий запис у дво- і тривимірному просторах; у першому випадку треба просто відкинути треті рівняння. Параметрична форма опису кривих і поверхонь є:

- найгнучкішою;
- стійкою до будь-яких варіацій форми та орієнтації об'єктів, що робить її особливо зручною для використання в математичному забезпеченні систем КГ.

Параметрична форма зображення визначеної кривої чи поверхні не є унікальною. Об'єкт можна зобразити багатьма способами, але тут зосередимо увагу на *поліноміальній формі*, тобто всі функції параметра u в описі кривих чи параметрів u і v в описі поверхонь є поліномами.

Розглянемо рівняння кривої та поверхні у вигляді

$$\mathbf{p}(u) = \begin{bmatrix} x(u) \\ y(u) \\ z(u) \end{bmatrix}, \quad \mathbf{p}(u,v) = \begin{bmatrix} x(u,v) \\ y(u,v) \\ z(u,v) \end{bmatrix}.$$

Поліноміальна параметрична крива степеня n і поверхня степенів n та m мають вигляд

$$\mathbf{p}(u) = \sum_{k=0}^{n} \mathbf{c}_k u^k, \quad \mathbf{p}(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} \mathbf{c}_{ij} u^i v^j, \tag{7.1}$$

де $\mathbf{c}_k$, $\mathbf{c}_{ij}$ мають незалежні компоненти x, y, z. Зазначимо, що в OpenGL часто використовують термін *порядок полінома*, який має значення, на 1 більше, ніж степінь полінома.

Матриця $\{\mathbf{c}_k\} = [\mathbf{c}_{xk}, \mathbf{c}_{yk}, \mathbf{c}_{zk}]^T$ складається з $n+1$ стовпців, тобто є $3\cdot(n+1)$ степенів свободи у виборі коефіцієнтів для конкретної кривої. Криву можна визначити на будь-якому інтервалі зміни параметра u: $u_{\min} \le u \le u_{\max}$, але, не втрачаючи загальності міркувань, можна покласти, що $0 \le u \le 1$. Тоді, коли u буде пробігати значення в цьому інтервалі, відображена точка переміщатиметься по *сегменту кривої.*

Для визначення конкретної поверхні необхідно задати $3\cdot(n+1)\cdot(m+1)$ коефіцієнтів. Параметри u і v теж можна змінювати на інтервалі $0 \le u, v \le 1$ і так визначити *ділянку поверхні*.

Основні **переваги** застосування поліноміальної параметричної форми зображення:

- можливість локального контролю форми об'єкта;
- гладкість і неперервність у математичному сенсі;
- можливість аналітичного обчислення похідних;
- стійкість до малих збурень;
- можливість використати відносно прості, а отже, високошвидкісні методи тонування.

7.2.2. Параметрична і геометрична неперервність

У комп'ютерній графіці *сплайновою кривою* називають будь-яку складену криву, сформовану поліноміальними ділянками, які задовольняють умови неперервності на їхніх межах. *Сплайнову поверхню* описують двома наборами ортогональних сплайнових кривих.

Сплайнову криву задають набором координат точок, названих *опорними*, які вказують загальну форму кривої. Набір опорних точок формує межу області простору, названу *опуклою оболонкою*. За заданими точками одним з двох способів підбирається кусково-неперервна параметрична поліноміальна функція. Якщо поліноміальні ділянки з'єднують усі опорні точки, як показано на рис. 7.1, *а*, отриману криву називають *інтерполяцією* набору опорних точок. Якщо генеровану поліноміальну криву будують так, що деякі або всі опорні точки не належать їй, криву називають *апроксимацією* набору опорних точок (рис. 7.1, *б*).

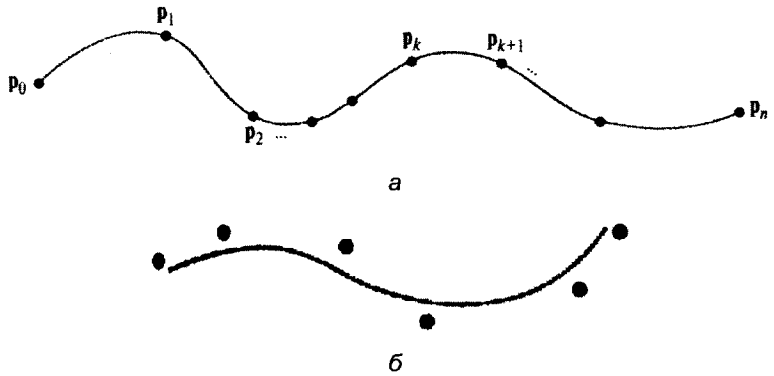


Рис. 7.1. Набір з шести контрольних точок, інтерпольованих кусково-гладкою неперервною поліноміальною кривою (а) і апроксимованих кусково-неперервними поліноміальними ділянками (б)

Методи інтерполяції широко використовують для оцифрування рисунків чи задання шляхів анімації. Методи апроксимації застосовують переважно в засобах проектування для створення форм об'єктів.

Щоб гарантувати гладкі переходи від однієї ділянки кусково-гладкого параметричного сплайну до іншої, можна накласти різні умови неперервності в точках з'єднання (спряження). **Параметрична неперервність нульового порядку C^0** означає, що в точці спряження сегментів кривої значення всіх трьох параметричних компонент поліномів є однакові, тобто $\mathbf{p}^{(1)}(1)=\mathbf{p}^{(2)}(0)$ (криві просто зустрічаються). **Параметрична неперервність першого порядку C^1** означає, що перші похідні за параметром (дотичні) координатних функцій двох послідовних ділянок кривої однакові в точці їхнього з'єднання ($\mathbf{p}^{(1)'}(1)=\mathbf{p}^{(2)'}(0)$, тоді умови неперервності для значень компонент виконуються автоматично). **Параметрична неперервність другого порядку C^2** означає, що перша і друга похідні за параметром двох послідовних ділянок кривої однакові в точці спряження ($\mathbf{p}^{(1)''}(1)=\mathbf{p}^{(2)''}(0)$, тоді однакові й швидкості зміни дотичних векторів). Неперервності першого порядку достатньо для оцифрування рисунків і в деяких конструкторських застосуваннях, а неперервність другого порядку корисна для опису руху камери в анімації та в багатьох точних застосуваннях автоматизованого проектування (рис. 7.2).

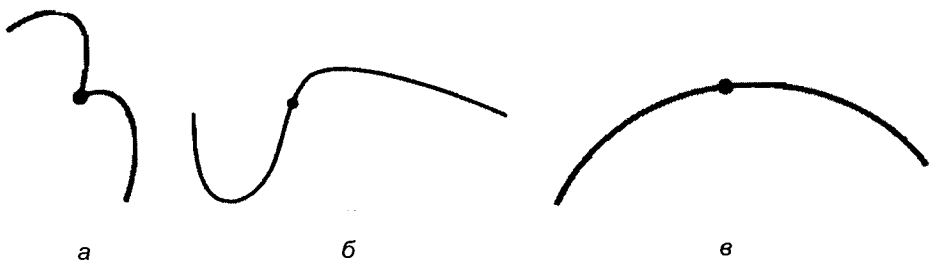


Рис. 7.2. Кусково-гладка побудова кривої з використанням параметричної неперервності нульового (а), першого (б) та другого (в) порядків

Інший метод з'єднання двох послідовних ділянок кривої полягає в накладанні умов *геометричної неперервності;* тоді вимагають, щоб похідні за параметром були лише пропорційні на спільній межі. **Геометрична неперервність нульового порядку G^0** – це те саме, що й C^0. **Геометрична неперервність першого порядку G^1** означає, що перші похідні за параметром пропорційні в точці перетину двох послідовних ділянок ($\mathbf{p}^{(1)'}(1)=k\mathbf{p}^{(2)'}(0)$, ділянки мають однаковий напрямок дотичного вектора, але не обов'язково однаковий модуль). **Геометрична неперервність другого порядку G^2** означає, що перша і друга похідні за параметром двох послідовних ділянок пропорційні на їхній межі ($\mathbf{p}^{(1)''}(1)=k\mathbf{p}^{(2)''}(0)$, кривизни обох ділянок однакові).

Крива, побудована з накладеними умовами геометричної неперервності, подібна до кривої, одержаної за умови параметричної неперервності (рис. 7.3); відмінність у тому, що геометрично неперервні криві притягуються до ділянки з більшою довжиною дотичного вектора – дотичний вектор кривої C_3 в точці P_1 має більшу довжину, ніж дотичний вектор кривої C_1 у точці P_1.

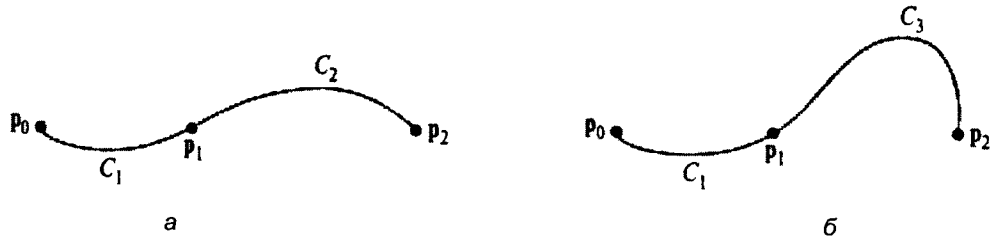


Рис. 7.3. Порівняння параметричної та геометричної неперервностей

7.2.3. Інтерполяція кубічними сплайнами

Вибравши клас поліноміальних кривих у параметричній формі, потрібно вибрати степінь полінома для опису кривої. Під час роботи з поліномами високих степенів зростає небезпека отримати криву хвилястої форми, а сам процес розрахунку координат точок на кривій потребує великої кількості обчислень. З іншого боку, вибір полінома занадто низького степеня може призвести до того, що в нашому розпорядженні виявиться мало регульованих параметрів – коефіцієнтів полінома – і не вдасться відтворити з потрібною точністю форму кривої. Вихід можна знайти в тому, щоб не описувати криву єдиним поліномом високого степеня, а розбити її на сегменти невеликої довжини, які можна описати поліномами низького степеня. Тому, формуючи криві, починають із кубічного полінома. Описати кубічну поліноміальну криву, виходячи із (7.1), можна так:

$$\mathbf{p}(u) = \mathbf{c}_0 + \mathbf{c}_1 u + \mathbf{c}_2 u^2 + \mathbf{c}_3 u^3 = \sum_{k=0}^{3} \mathbf{c}_k u^k = \mathbf{u}^T \mathbf{c}, \qquad (7.2)$$

де

$$\mathbf{c} = \begin{bmatrix} \mathbf{c}_0 \\ \mathbf{c}_1 \\ \mathbf{c}_2 \\ \mathbf{c}_3 \end{bmatrix}, \ \mathbf{u} = \begin{bmatrix} 1 \\ u \\ u^2 \\ u^3 \end{bmatrix}, \ \mathbf{c}_k = \begin{bmatrix} \mathbf{c}_{kx} \\ \mathbf{c}_{ky} \\ \mathbf{c}_{kz} \end{bmatrix}.$$

Тут потрібно буде обчислити матрицю коефіцієнтів полінома **c** за заданою множиною опорних точок. Розглянемо різні класи кубічних кривих, які відрізняються характером зіставлення з опорними точками. Оскільки параметричні функції для компонент *x, y* і *z* незалежні, систему з 12 рівнянь з 12-ма невідомими буде розділена на три групи по 4 рівняння з 4-ма невідомими.

Першим розглянемо клас *інтерполяційних поліномів* та проаналізуємо їх гладкість, розглянувши 4 поліноміальні стикувальні функції:

$$b_{I0}(u) = -\frac{9}{2}(u-\frac{1}{3})(u-\frac{2}{3})(u-1),\ b_{I1}(u) = \frac{27}{2}u(u-\frac{2}{3})(u-1),$$

$$b_{I2}(u) = -\frac{27}{2}u(u-\frac{1}{3})(u-1),\ b_{I3}(u) = \frac{9}{2}u(u-\frac{1}{3})(u-\frac{2}{3}).$$

Стикувальні функції характеризують внесок кожної опорної точки. Всі їхні нулі знаходяться в інтервалі [0, 1], тому їхні значення можуть суттєво змінюватись на цьому інтервалі, а самі функції **не є монотонними**, бо інтерполяційна крива має точно проходити через опорні точки, а не в їх найближчому околі. Вони мають **погану гладкість**, у точках спряження сегментів відсутня неперервність похідних. Тому застосуємо розглянуті методи інтерполяції та апроксимації для аналізу кривих і поверхонь інших типів, що більше підходитимуть для задач КГ.

Бікубічне рівняння *ділянки поверхні*, за (7.1), можна записати так:

$$\mathbf{p}(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} \mathbf{c}_{ij} u^i v^j,$$

де $\mathbf{c}_{ij}$ – трикомпонентна матриця-стовпець, елементами якої є коефіцієнти за однакових степенів незалежної змінної в рівняннях для *x, y* і *z* компонент.

Розширивши методику використання стикувальних функцій на поверхні, одержимо таке *рівняння для ділянки поверхні*:

$$\mathbf{p}(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} b_{Ii}(u) b_{Ij}(v) \mathbf{p}_{ij}.$$

Отже, тут розглянуто три еквівалентні методи визначення конкретного сплайна за заданого степеня полінома і розміщення опорних точок:

- задання умов спряження, які накладають на сплайн;
- задання матриці, що характеризує сплайн;
- задання набору стикувальних (базисних) функцій, які визначають, як задані умови використовують для розрахунку точок уздовж кривої.

7.2.4. Ермітова форма зображення кривих і поверхонь

У цьому випадку точка спряження є спільною для двох сусідніх сегментів, і похідні до кривої в ній для обох сегментів однакові.

Записавши поліном через базисні функції

$$\mathbf{p}(u) = \mathbf{b}_H^T(u)\mathbf{q} = \sum_{i=0}^{3} b_{Hi}^T(u)\mathbf{q}_i,$$

де $\mathbf{b}_{Hi}(u) = \mathbf{M}_H^T\mathbf{u}$, $b_{H0}(u) = 2u^3 - 3u^2 + 1$, $b_{H1}(u) = -2u^3 + 3u^2$,

$b_{H2}(u) = u^3 - 2u^2 + u$, $b_{H3}(u) = u^3 - u^2$,

одержимо, що нулі цих 4 поліномів розміщені за межами інтервалу [0,1], а тому базисні функції є більш гладкими, ніж для інтерполяційних поліномів. На відміну від інтерполяційних поліномів, ермітові сплайни можна налаштовувати локально, бо кожен сегмент кривої залежить лише від умов у кінцевих точках.

Поліноми Ерміта можуть бути корисними під час оцифрування, коли задати чи апроксимувати нахил кривої не дуже складно. Однак у більшості задач КГ корисніше генерувати сплайнові криві, не вимагаючи введення значень нахилів кривих чи іншої геометричної інформації, відмінної від координат опорних точок.

Скориставшись поданими раніше базисними функціями, можна визначити ділянку поверхні у формі Ерміта:

$$\mathbf{p}(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} b_{Hi}(u)b_{Hj}(v)\mathbf{q}_{ij},$$

де $\mathbf{Q} = [\mathbf{q}_{ij}]$ – набір даних, що зображає ділянку поверхні аналогічно до того, як $\mathbf{q}$ зображає сегмент. Оскільки в інтерактивних програмах користувачеві бажано специфікувати координати точок, варто сформулювати аналітичні вирази для похідних.

7.3. Криві та поверхні Без'є

Цей метод апроксимації сплайнами розробив французький інженер П'єр Без'є (*Pierre Bezier*). Сплайни Без'є мають низку властивостей, що роблять їх дуже корисними і зручними для побудови кривих і поверхонь. Окрім того, їх легко реалізувати. Ці причини зумовили їхнє широке використання в різних системах автоматизованого проектування, в універсальних графічних пакетах і в змішаних пакетах рисування і живопису. Ділянку кривої Без'є можна підібрати за довільною кількістю точок, хоч у деяких графічних пакетах їх кількість обмежують чотирма. Як і для інтерполяційних сплайнів, шлях кривої Без'є можна задати трьома перерахованими вище способами, однак найзручнішим поданням є специфікація стикувальних функцій.

7.3.1. Криві Без'є

Обидві розглянуті форми (Ерміта й інтерполяційний поліном) є кубічними поліноміальними кривими, але для їхнього формування використовують різні за характером набори даних. Отже, не можна порівняти криві в цих формах, одержані за тим самим набором даних. Спробуємо використати той самий набір опорних точок і для визначення інтерполяційного полінома, і для задання похідних у формі Ерміта. У підсумку одержимо *криву у формі Без'є*, яка є дуже хорошим наближенням кривої у формі Ерміта і яку можна порівнювати з інтерполяційним поліномом, сформованим за тим самим набором опорних точок. Крім того, оскільки визначення кривої у формі Без'є **не потребує** задання похідних, така процедура ідеально підходить для інтерактивної побудови криволінійних об'єктів у системах КГ й автоматизації проектування.

Складена крива, побудована за методом Без'є на довільному ансамблі опорних точок, належить до класу $\mathbf{C}^0$, бо дотичні праворуч і ліворуч від точки спряження апроксимуються різними формулами.

Проаналізуємо **важливі властивості кривих Без'є**. Вони випливають із властивостей стикувальних функцій для цього класу кривих (рис. 7.4).

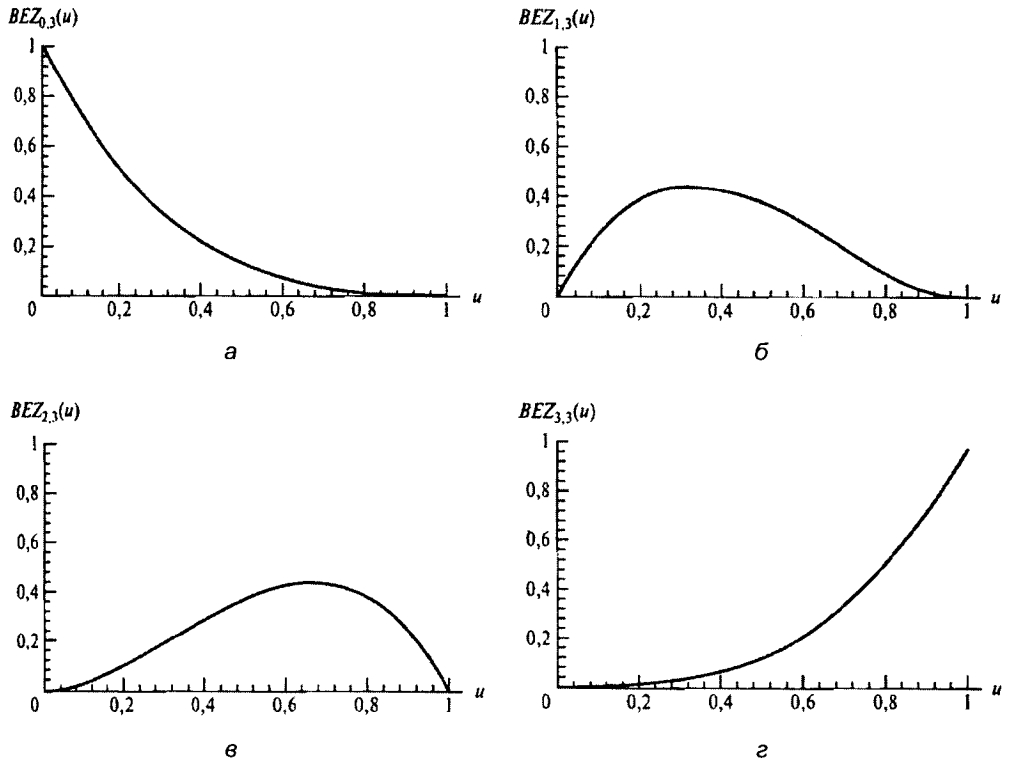


Рис. 7.4. Стикувальні функції Без'є для кубічних кривих

Криву можна подати у вигляді

$$\mathbf{p}(u) = \mathbf{b}_B^T(u)\mathbf{p},$$

де

$$\mathbf{b}_B(u) = \mathbf{M}_B^T\mathbf{u} = \begin{bmatrix} (1-u)^3 \\ 3u(1-u)^2 \\ 3u^2(1-u) \\ u^3 \end{bmatrix}.$$

Ці 4 поліноми є частковими випадками поліномів Бернштейна

$$J_{n,k}(u) = \frac{n!}{k!(n-k)!}u^k(1-u)^{n-k}.$$

Поліноми Бернштейна мають низку цікавих властивостей.

1. Усі нулі полінома розміщені або в точках $u = 0$, або в точках $u = 1$. Отже, кожна вагова функція задовольняє умову $J_{n,k}(u) > 0$ за $0 < u < 1$. Оскільки поліноміальні функції не мають нулів на інтервалі подання параметра, всі вони достатньо гладкі, але не допускають *локального контролю* за формою кривої, тобто якщо змістити будь-яку опорну точку, то зміниться вся крива. Враховуючи, що

$$J_{n,k}(u) \le 1, \sum_{k=0}^{n} J_{n,k}(u) = 1,$$

одержимо, що крива повинна лежати всередині випуклої багатокутної оболонки, утвореної 4-ма (якщо $n = 3$) заданими опорними точками. Отже, виявляється, що хоч поліноміальна крива Без'є і не проходить через всі задані точки, вона ніколи не виходить за межі області, обмеженої цими точками.

2. Під час конструювання таких кривих можна працювати тільки з координатами опорних точок.
3. Крива з'єднує першу і останню опорні точки.

Щоб згенерувати замкнуту криву Без'є, останню опорну точку розміщують так само, як першу. Крім того, введення декількох опорних точок з однаковими координатами рівнозначне присвоєнню більшого вагового коефіцієнта цій точці (тоді отримана крива притягується до цієї точки). Однак для реалізації таких можливостей переважно використовують гнучкіші бі-сплайни.

Щоб побудувати складні сплайнові криві, можна застосувати низку кубічних сегментів Без'є. Використовуючи вирази для перших похідних за параметром, можна прирівняти дотичні до кривих і отримати неперервність $\mathbf{C}^1$ між ділянками кривої (перші дві точки буде жорстко зафіксовано, рис. 7.5).

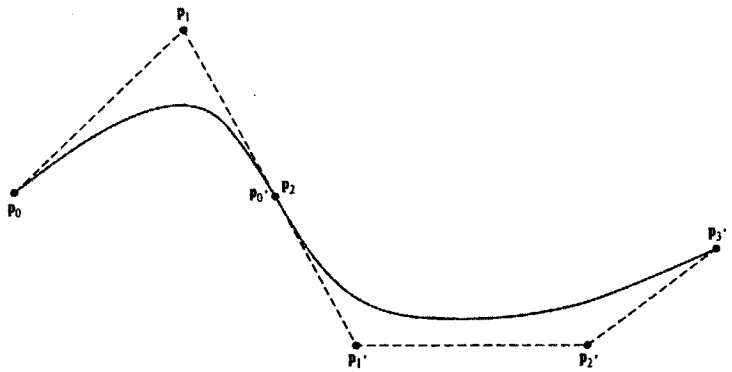


Рис. 7.5. Кусково-гладка апроксимація, сформована двома сегментами Без'є

Крім того, можна використати вирази для других похідних і отримати неперервність $\mathbf{C}^2$, хоч це жорстко зафіксує перші три опорні точки.

7.3.2. Ділянки поверхні у формі Без'є

Ділянки поверхні можна також записати у формі Без'є за допомогою вагових функцій. Якщо $\mathbf{P} = [\mathbf{p}_{ij}]$ – масив опорних точок розміром 4×4, то відповідну ділянку поверхні у формі Без'є описують співвідношенням

$$\mathbf{p}(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} b_{Bi}(u)b_{Bj}(v)\mathbf{p}_{ij} = \mathbf{u}^T\mathbf{M}_B\mathbf{P}\mathbf{M}_B^T\mathbf{v}\,.$$

Ділянка поверхні проходить через кутові точки $\mathbf{p}_{00}$, $\mathbf{p}_{03}$, $\mathbf{p}_{30}$ і $\mathbf{p}_{33}$ і не виходить за межі випуклого багатогранника, вершинами якого є опорні точки. 12 опорних точок із 16 можна інтерпретувати як дані, що визначають напрямок похідних за різними параметрами в кутових точках (рис. 7.6).

Кубічні криві та поверхні Без'є доволі поширені у задачах КГ й автоматизації проектування. Але об'єкти цього класу мають одне принципове **обмеження**: у точках спряження складених кривих чи поверхонь забезпечено виконання умов параметричної неперервності тільки класу $\mathbf{C}^0$. Залишається ще один шлях удосконалення методики побудови кубічних кривих: треба відмовитись від вимоги, щоб крива проходила через опорні точки і погодитись, щоб вона проходила близько до них. За цих умов доволі просто забезпечити неперервність ще й першої та другої похідних у точках спряження сегментів.

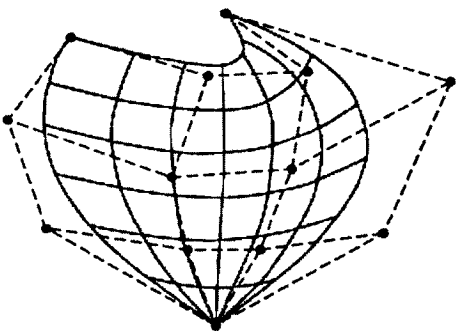


Рис. 7.6. Каркасна поверхня Без'є, побудована за 16-ма опорними точками (сітка 4 на 4), з'єднаними пунктирними лініями

7.4. Кубічні бі-сплайни

Бі-сплайни мають **дві переваги** порівняно зі сплайнами Без'є:

- степінь полінома бі-сплайна можна задати незалежно від кількості контрольних точок;
- вони допускають локальний контроль над формою кривої.

Платою за це є більша їхня складність порівняно зі сплайнами Без'є.

Розглянемо 4 послідовні опорні точки $\{\mathbf{p}_{i-2}, \mathbf{p}_{i-1}, \mathbf{p}_i, \mathbf{p}_{i+1}\}$. Раніше на основі 4-х точок було сформовано таку кубічну криву, яка під час зміни значення параметра u від 0 до 1 пробігала відстань від $\mathbf{p}_{i-2}$ до $\mathbf{p}_{i+1}$, причому проходила через точки $\mathbf{p}_{i-2}$ і $\mathbf{p}_{i+1}$. Тепер розглянемо умову: в разі зміни параметра u від 0 до 1 потрібно сформувати ділянку кривої між проміжними точками, що відповідають двом середнім із заданого набору.

7.4.1. Бі-сплайнові криві

Властивості полінома отримаємо, звернувшись до стикувальних функцій

$$\mathbf{b}_S(u) = \mathbf{M}_S^T\mathbf{u} = \frac{1}{6}\begin{bmatrix} (1-u)^3 \\ 4-6u^2+3u^3 \\ 1+3u+3u^2-3u^3 \\ u^3 \end{bmatrix}.$$

1. Як і у випадку з формою Без'є, поліноміальні стикувальні функції *бі-сплайна* володіють властивостями

$$\sum_{i=0}^{3} b_{Si}(u) = 1 \text{ і } 0 \le b_{Si}(u) \le 1$$

на інтервалі $0 \le u \le 1$. Отже, сформований сегмент кривої повинен лежати в межах, заданих випуклою оболонкою з опорних точок, причому він перекриває тільки частину діапазону зміни параметра в межах оболонки.

2. Якщо за одним і тим самим ансамблем опорних точок формувати бі-сплайнову криву, криву Без'є і складену інтерполяційну криву, то в першому випадку треба виконати майже втричі більше обчислень, ніж у двох інших. Це можна пояснити тим, що за один цикл обчислень у випадку апроксимації кривої бі-сплайном формується сегмент між сусідніми опорними точками (тому потрібно три цикли обчислень для формування кривої від першої до четвертої точки), а в двох інших випадках відразу будують увесь сегмент – від першої до четвертої точки.
3. Під час формування 4-х послідовних сегментів складеної бі-сплайнової кривої беруть до уваги кожну опорну точку. Це гарантує *локальний характер* кривих цього класу, тобто зміна положення окремої точки впливає на поведінку складеної кривої в достатньо малому околі цієї точки. Загальний внесок окремої точки можна подати у вигляді $B_i(u)\mathbf{p}_i$, де

$$B_i(u) = \begin{cases} 0, & u < i-2, \\ b_{S0}(u+2) & i-2 \le u < i-1, \\ b_{S1}(u+1) & i-1 \le u < i, \\ b_{S2}(u) & i \le u < i+1, \\ b_{S3}(u-1) & i+1 \le u < i+1, \\ 0 & u \ge i+2. \end{cases}$$

Задавши ансамбль опорних точок, можна сформувати кусково-поліноміальну криву на всьому інтервалі зміни значень параметра у вигляді *лінійної комбінації базисних функцій*. Сама назва *бі-сплайн* походить від *basis spline (базисний сплайн)*, тобто відображає той факт, що множина функцій утворює *базис* для цієї послідовності вузлів і заданого степеня полінома.

У теорії сплайнів загального вигляду ця ідея поширена на поліноми вищого степеня, і прийнято варіант апроксимації, коли різні сегменти складеної кривої зображено поліномами різних степенів.

7.4.2. Бі-сплайнові поверхні

Бі-сплайнова поверхня є розширенням бі-сплайнових кривих. Використовуючи базисні функції, одержимо опис ділянки поверхні

$$\mathbf{p}(u,v)=\sum_{i=0}^{3}\sum_{j=0}^{3} b_{Si}(u)b_{Sj}(v)\mathbf{p}_{ij}\,.$$

Його відмінність від аналогічного співвідношення для поверхні Без'є полягає в тому, що воно виконується лише для ділянки, розміщеної біля центрального чотирикутника масиву опорних точок. На розрахунок поверхні, що відповідає всьому масиву опорних точок, піде в 9 разів більше часу, ніж у випадку Без'є-поверхні. Але зате на цій самій множині вдасться побудувати плавнішу поверхню.

7.4.3. Узагальнені бі-сплайни

Підхід, який вибрано для формування бі-сплайнів, полягає в тому, щоб визначати сплайн у термінах множини базисних функцій, кожна з яких відмінна від нуля тільки на інтервалі в декілька вузлів. Отже, можна записати функцію $\mathbf{p}(u)$ у вигляді

$$\mathbf{p}(u)=\sum_{i=0}^{n} B_{id}(u)\mathbf{p}_i\,,$$

де кожна функція $B_{id}(u)$ є поліномом степеня d на інтервалі в декілька вузлів і дорівнює 0 за межами інтервалу $(u_{i\min}, u_{i\max})$, $u_{\min}=u_0\le u_1\le\ldots\le u_n=u_{\max}$. Є багато способів визначення базисних функцій, але особливе місце належить *методу рекурсивних функцій Кокса – де Бура*

$$B_{i0}(u)=\begin{cases}1, & u_i\le u\le u_{i+1},\\ 0, & \text{в протилежному випадку;}\end{cases}$$

$$B_{id}(u)=\frac{u-u_i}{u_{i+d}-u_i}B_{i(d-1)}(u)+\frac{u_{i+d}-u}{u_{i+d+1}-u_{i+1}}B_{(i+1)(d-1)}(u)\,.$$

Кожна функція з першої множини $B_{i0}(u)$ постійна на одному інтервалі і дорівнює нулю за його межами; кожна функція з другої множини $B_{i1}(u)$ лінійна на двох інтервалах і дорівнює 0 за їхніми межами; кожна функція з третьої мно-

жини $B_{i2}(u)$ має вигляд квадратичної кривої на трьох інтервалах і дорівнює 0 за їхніми межами і т.д.

У загальному випадку функція з множини $B_{id}(u)$ має відмінне від 0 значення на d+1 інтервалах між u_i і u_{i+d+1}, є поліномом степеня d на кожному з цих інтервалів, причому у вузлах забезпечена параметрична неперервність класу $\mathbf{C}^{d-1}$. Водночас кожна опорна точка впливає тільки на ту ділянку сумарної кривої, яка лежить всередині оболонки, утвореної d+1 опорною точкою. Кінцеву точку кожного підінтервалу називають *вузлом*, а весь набір кінцевих точок вибраних підінтервалів – *вектором вузлів*.

7.4.4. Відкриті рівномірні бі-сплайни

У випадку постійного кроку між вузлами бі-сплайн називають *рівномірним*, але можна забезпечити більшу гнучкість у формуванні кривих, якщо дозволити не тільки нерівномірне розміщення вузлів, але й наявність кратних ($u_i = u_{i+1}$) вузлів. Якщо кратність вузла дорівнює d+1, то бі-сплайн степеня d проходитиме через відповідну опорну точку. Якщо зробити кратними початкову і кінцеву точки, а інші вузли залишити рівновіддаленими, то такий бі-сплайн називають *відкритим* (він пройде через початкову і кінцеву точки).

Відкриті рівномірні бі-сплайни мають характеристики, схожі на характеристики сплайнів Без'є. Фактично для $d = n+1$ (степінь полінома дорівнює n) відкриті бі-сплайни зводяться до сплайнів Без'є, а всі значення вузлів дорівнюють 0 або 1. Наприклад, для формування кубічного відкритого бі-сплайну ($d = 4$) і чотирьох контрольних точок вектор вузлів дорівнює {0, 0, 0, 0, 1, 1, 1, 1} Він перетворює бі-сплайн на криву Без'є.

У загальному випадку можна робити кратними і внутрішні вузли, а також розміщувати їх нерівномірно.

7.4.5. NURBS – нерівномірний раціональний бі-сплайн

Аналізуючи бі-сплайни, припускали, що $\mathbf{p}(u)$ є тривимірним масивом $[x(u)\ y(u)\ z(u)]^T$. Співвідношення не зміняться, якщо перейти до 4-вимірних бі-сплайнів. Тривимірні опорні точки $\mathbf{p}_i = [x_i\ y_i\ z_i]^T$ можна зобразити в просторі однорідних координат як $\mathbf{q}_i = w_i[x_i\ y_i\ z_i\ 1]^T$. Ідея полягає в тому, щоб використати коефіцієнти w_i для збільшення чи зменшення "ваги" конкретної опорної точки. Ці зважені опорні точки можна використовувати для формування 4-вимірного бі-сплайна. Перші 3 компоненти будуть бі-сплайновим зображенням зважених опорних точок:

$$\mathbf{q}(u) = \begin{bmatrix} x(u) \\ y(u) \\ z(u) \end{bmatrix} = \sum_{i=0}^{m} B_{id}(u) w_i \mathbf{p}_i .$$

Компонента $w(u)$ є скалярним поліноміальним бі-сплайном, сформованим за множиною значень вагових коефіцієнтів

$$w(u) = \sum_{i=0}^{m} B_{id}(u) w_i .$$

У разі використання 4 параметричних сплайнів як однорідних координат значення $w(u)$ може не дорівнювати 1, отже, для переходу до тривимірного зображення точок треба буде використати перспективне ділення:

$$\mathbf{p}(u) = \frac{1}{w(u)} \mathbf{q}(u) = \frac{\sum_{i=0}^{m} B_{id}(u) w_i \mathbf{p}_i}{\sum_{i=0}^{m} B_{id}(u) w_i} .$$

Кожна компонента функції $\mathbf{p}(u)$ є раціональною функцією параметра u, а оскільки не було накладено жодних обмежень на розміщення вузлів, то вона належить до класу *нерівномірних раціональних бі-сплайнів* (*NURBS*). Ці криві зберігають усі властивості, притаманні тривимірним *бі-сплайнам,* зокрема обмеженість випуклою оболонкою і неперервність, але додатково до них мають ще дві унікальні властивості.

1. Після перспективного перетворення (яке не належить до класу афінних) NURBS-криві відображатимуться коректно, а не спотворено, як бі-сплайнові криві.
2. Оскільки аналітичні квадратичні криві є частковим випадком квадратичних NURBS-кривих, то за їхньою допомогою можна моделювати практично всі розповсюджені типи криволінійних геометричних об'єктів.

Збільшуючи кратність вузлів, можна налаштувати траєкторію кривої і вводити розриви. Зазначимо, що кратні значення вузлів зменшують неперервність на 1 після кожного повторення значення.

7.5. Криві та поверхні в OpenGL

У складі OpenGL є засіб підтримки роботи з кривими і поверхнями Без'є – *Без'є-обчислювач* (евалюатор), який дає змогу обчислювати значення поліномів Без'є будь-якого порядку. Він не передбачає, що опорні точки мають бути розміщені з постійним кроком, а тому дає змогу перетворювати на форму Без'є криві, визначені будь-яким з розглянутих раніше способів. Без'є-обчислювач можна використовувати для роботи з поліномами від однієї, двох, трьох чи чотирьох змінних. У бібліотеці GLU його використовують у 4-вимірному варіанті для підтримки методів обробки NURBS-кривих. Варіант із поліномами 2 змінних використовують переважно для формування поверхонь. Крім обробки кривих і поверхонь, Без'є-обчислювач забезпечує розрахунок кольорів, нормалей і координат текстур.

7.5.1. Створення кривих Без'є

Задання параметрів і активізація процедур відображення *кривої Без'є* (полінома однієї змінної) виконують за допомогою функцій

```
glMap1*(type, u_min, u_max, stride, order, point_array);
glEnable(type);
```

Суфікс **f** або **d** свідчить про використання даних з плаваючою крапкою чи подвійної точності. Аргумент **type** задає тип об'єкта, який буде зображено поліномом Без'є; як значення цього аргументу можна використовувати константи, що визначають 3- чи 4-вимірні геометричні точки (**GL_MAP1_VERTEX_3** або **GL_MAP1_VERTEX_4**), колір у форматі RGBA (**GL_MAP1_COLOR_4**), 3-вимірні вектори нормалі (**GL_MAP1_NORMAL**), індексовані кольори (**GL_MAP1_INDEX**) та координати текстур (від одно- до 4-вимірних). Аргумент **point_array** – вказівник на масив опорних точок полінома; **u_min** та **u_max** визначають межі параметра, переважно 0 та 1; **stride** – кількість значень параметра між сегментами кривої, для кубічного полінома – 3. Значення аргументу **order** має бути на 1 більше від степеня полінома.

Процедури формування кривої Без'є відключають командою

```
glDisable(type);
```

Якщо функцію розрахунку активовано, то можна отримати за нею значення полінома за допомогою функції

```
glEvalCoord1*(u);
```

код-суфікс може мати значення **f** або **d**, крім того, можна використати суфікс **v**, який показує, що значення аргумента задано як масив.

Її виклик може замінити звертання до функцій **glVertex**, **glColor** і **glNormal**.

Якщо значення параметра **u** розподілено **рівномірно**, то для обчислення точок на кривій можна використати функції

```
glMapGrid1*(n, u1, u2);
glEvalMesh1(mode, n1, n2);
```

n – ціле число рівномірних поділів у діапазоні від **u1** до **u2**; **n1**, **n2** – цілочисловий діапазон, що відповідає проміжку від **u1** до **u2**; параметру **mode** присвоюють значення **GL_POINT** або **GL_LINE** залежно від того, як треба зобразити криву (з використанням дискретних точок чи прямих відрізків).

Після виклику **glMapGrid1** буде встановлено рівномірну сітку на **n** відрізків (кроків), а після виклику **glEvalMesh1** – сформовано криву.

Декілька функцій **glMap1** можна активізувати одночасно, тоді для кожного дозволеного типу даних виклики **glEvalCoord1** чи **glMapGrid1** і **glEvalMesh1**

дають точки даних. Це дає змогу генерувати комбінації координатних точок, кодів кольору, векторів нормалей до поверхні і даних, що стосуються текстури поверхні. Однак одночасно активізувати **GL_MAP1_VERTEX_3** і **GL_MAP1_VERTEX_4** не можна, і в будь-який момент може бути активним лише один генератор поверхневої текстури.

7.5.2. Формування поверхонь Без'є

Поверхні Без'є формують приблизно за тією самою методикою, що й криві, тільки роль функцій ініціалізації та активізації виконують

```
glMap2*(type, u_min, u_max, u_stride, u_order, v_min, v_max,
      v_stride, v_order, point_array);
glEnable(type);
```

а для одержання результатів треба звертатись до функцій

```
glEvalCoord2*(u, v);
glEvalCoord2*v(uvArray);
```

для **uvArray=(u, v)**.

Процедури формування поверхні Без'є відключають командою

```
glDisable(type);
```

Щоб відобразити на екрані поверхню Без'є, багатократно викликають процедуру **glEvalCoord2**, яка генерує низку функцій **glVertex3**.

Для роботи на **рівномірній сітці** можна використати функції

```
glMapGrid2*(nu, u1, u2, nv, v1, v2);
glEvalMesh2(mode, nu1, nu2, nv1, nv2);
```

- суфікс знову дорівнює **f** або **d**,
- параметру **mode** присвоюють значення **GL_POINT**, **GL_LINE** або **GL_FILL**.

Функцію **glMap2** можна використовувати для визначення інших типів даних, як це описано вище для функції **glMap1**. З цією метою використовують символьні константи **GL_MAP2_COLOR_4** і **GL_MAP2_NORMAL.** Крім того, можна активізувати декілька функцій **glMap2** для генерування різних комбінацій даних.

7.5.3. Функції GLU NURBS-кривих та бі-сплайнових кривих

Функції обчислення поліномів можна використати й на нерівномірній сітці задання параметрів для формування 4-вимірних кривих і поверхонь. Оскільки будь-який поліноміальний об'єкт можна звести до форми Без'є, дібравши множину опорних точок, то OpenGL надає повний арсенал засобів, необхідних для ви-

значення NURBS-кривих і поверхонь. До складу бібліотеки утиліт OpenGL GLU входить набір функцій для роботи з NURBS-об'єктами.

Для роботи з NURBS-кривими використовують п'ять функцій:

```
gluNewNurbsRenderer();
gluNurbsProperty(curveName, property, value);
gluBeginNurbsCurve(curveName);
gluNurbsCurve(curveName, nKnots, KnotVector, Stride,
      &ctrlPts[0][0], DegParam, GL_MAP1_VERTEX_4);
gluEndNurbsCurve(curveName);
```

Дві перші функції формують новий об'єкт і задають спосіб його відображення, наступні три використовують для формування кривої. Після виклику процедур візуалізації сплайна за допомогою команди **gluNewNurbsRenderer** можна задати кілька додаткових властивостей кривої – це роблять за допомогою виклику функції **gluNurbsCurve**. Якщо доступної пам'яті для зберігання сплайнового об'єкта бракує, система повертає у змінну **curveName** значення 0. Параметр **KnotVector** (число з плаваючою крапкою) визначає набір значень вузлів, кількість елементів у цьому векторі задають параметром **nKnots**, степінь полінома дорівнює **Stride** – 1, цілочисловий параметр **Stride** задає зсув між початками координат сусідніх точок у масиві **ctrlPts**, в якому перераховано значення з плаваючою крапкою тривимірних координат опорних точок. Якщо розміщення опорних точок записано неперервно (а не вставлено між даними інших типів), значення **Stride** має дорівнювати 3.

Створюючи раціональний сплайн, чотиривимірні однорідні координати використовують для задання опорних точок, і після однорідного ділення отримують шукану раціональну поліноміальну форму.

Крім того, можна використовувати функцію **gluNurbsCurve** і задати список кодів кольору, векторів нормалі чи властивостей текстурних поверхонь, як це було показано для функцій **glMap1** і **glMap2**. Як останній аргумент функції **gluNurbsCurve** можна використати будь-яку символьну константу, таку як **GL_MAP1_COLOR_4** і **GL_MAP1_NORMAL.** Кожну функцію викликають всередині пари **gluBeginNurbsCurve/gluEndNurbsCurve** із двома обмеженнями: не можна перераховувати більше ніж одну функцію для кожного типу даних і необхідно включити лише одну функцію генерації сплайна.

Під час використання процедур GLU сплайн автоматично ділиться на декілька ділянок і відображається як ламана лінія. Однак можна задати множину опцій візуалізації сплайна, багатократно викликаючи функцію

```
gluNurbsProperty(curveName, property, value);
```

параметру **curveName** присвоюють ім'я сплайна, параметру **property** – символьну константу, що визначає властивість візуалізації, яку треба задати, а

параметру **value** – або числове значення з плаваючою крапкою, або символьну константу, яка задає значення вибраної властивості.

Після оператора **gluNewNurbsRenderer** можна записати декілька функцій **gluNurbsProperty**.

Створену NURBS-криву видаляють командою

```
gluDeleteNurbsRenderer(curveName);
```

Для формування **бі-сплайнових кривих** за допомогою функції **gluNurbsCurve** використовують символьну константу **GL_MAP1_VERTEX_3** та оператори

```
gluBeginCurve(curveName);
gluEndCurve(curveName);
```

7.5.4. Функції GLU NURBS-поверхонь та бі-сплайнових поверхонь

У роботі з NURBS-поверхнями треба використовувати такі функції:

```
gluNewNurbsRenderer();
gluNurbsProperty(surfName, property, value);
gluBeginNurbsSurface(surfName);
gluNurbsSurface(surfName, nuKnots, uKnotVector, nvKnots,
   vKnotVector, uStride, vStride, &ctrlPts[0][0][0], uDegParam,
   vDegParam, GL_MAP2_VERTEX_4);
gluEndNurbsSurface(surfName);
```

Оператори і параметри, які використовують для визначення NURBS-поверхні, подібні до операторів і параметрів NURBS-кривої. Після виклику процедур візуалізації поверхні за допомогою команди **gluNewNurbsRenderer** можна задати декілька додаткових властивостей поверхні за допомогою функції **gluNurbsSurface**. Аналогічно можна визначити множину поверхонь з різними іменами-ідентифікаторами. Якщо доступної пам'яті бракує для зберігання об'єкта, система повертає до змінної **surfName** значення 0.

Параметри **uKnotVector** і **vKnotVector** (величини з плаваючою крапкою) визначають масиви значень вузлів у параметричних напрямах u і v. Кількість елементів у кожному векторі вузлів задають параметрами **nuKnots** і **nvKnots**, а степені поліномів за параметрами **u** і **v** дорівнюють **uDegParam** – 1, **vDegParam** – 1. Значення з плаваючою крапкою тривимірних координат контрольних точок перераховують у масиві **ctrlPts**, який містить (**nuKnots-uDegParam**)×(**nvKnots-vDegParam**) елементів. Цілочислові зсуви між початками записів сусідніх контрольних точок у параметричних напрямах u і v задають цілочисловими параметрами **uStride**, **vStride**.

Видаляють NURBS-поверхні з пам'яті, викликаючи ту саму функцію, яку використовують для кривої:

```
gluDeleteNurbsRenderer(surfName);
```

Для формування **бі-сплайнових поверхонь** за допомогою функції **gluNurbsSurface** використовують символьну константу **GL_MAP2_VERTEX_3** та оператори

```
gluBeginSurface(surfName);
gluEndSurface(surfName);
```

7.5.5. Вирізання на поверхні отворів криволінійної форми

У бібліотеці GLU є функції, що дають змогу виконувати операцію вирізання отворів криволінійної форми у сформованій поверхні. У прикладній програмі вирізальну NURBS-криву формують за допомогою функції **gluNurbsCurve**. Можна сформувати й кусково-лінійний контур отвору за допомогою функції

```
gluPwlCurve(surfName, nPts, *curvePts, stride,
GLU_MAP1_TRIM_2);
```

Тут параметр **surfName** – це ім'я поверхні, в якій буде вирізано отвір. Набір координат (значення з плаваючою крапкою) для вирізальної кривої задають у параметрі масиву **curvePts**, який містить **nPts** координатних позицій, цілочисловий зсув між послідовними координатними позиціями задають у параметрі **stride**. Якщо точки кривої задано в тривимірному параметричному просторі (u, v, h), останній аргумент у цій функції покладають рівним символьній константі **GLU_MAP1_TRIM_3**.

Можна також побудувати вирізальні криві, які є комбінаціями функцій **gluPwlCurve** і **gluNurbsCurve**. Будь-яка задана вирізальна крива GLU має бути замкнутою і несамоперетинальною.

Контури отворів буде сформовано після виклику **gluNurbsSurface**, причому відповідні вершини задають в операторних дужках – між викликами функцій **gluBeginTrim(surfName)** і **gluEndTrim(surfName)**.

7.6. Контрольні питання

1. Які ви знаєте функції GLUT правильних багатогранників?
2. Дайте означення квадрики та суперквадрики.
3. Які ви знаєте функції GLUT поверхонь другого порядку (сфери, конуса, тора, чайника)?
4. Які ви знаєте функції GLU поверхонь другого порядку (каркасної сфери, конуса, циліндра, конічного циліндра, плоского кругового кільця, суцільного диска, сегмента кругового кільця)?
5. Назвіть три головні форми математичного зображення кривих і поверхонь.
6. Порівняйте явну та неявну форми зображення кривих та поверхонь.
7. Які основні переваги параметричної і поліноміальної зокрема форми зображення кривої та поверхні?

8. Опишіть поліноміальні параметричні криві степеня n і поверхню степенів n та m.
9. Дайте означення параметричної неперервності нульового, першого та другого порядків та поясніть, у яких галузях КГ їх застосовують.
10. Дайте означення геометричної неперервності нульового, першого та другого порядків та порівняйте її з параметричною.
11. Порівняйте властивості вагових функцій інтерполяційних поліноміальних кривих та поліноміальних кривих у формі Ерміта.
12. Назвіть три еквівалентні методи знаходження сплайна для заданого ступеня полінома і розміщення контрольних точок.
13. Якого типу неперервність і якого порядку мають інтерполяційні поліноміальні криві?
14. Якого типу неперервність і якого порядку мають криві у формі Ерміта?
15. Чому криві та поверхні у формі Без'є набули поширення в КГ?
16. Якого типу неперервність і якого порядку мають криві Без'є?
17. Назвіть властивості вагових функцій кривих та поверхонь у формі Без'є.
18. Яке принципове обмеження виникає під час використання об'єктів цього класу?
19. Порівняйте властивості стикувальних функцій сплайна Без'є та бі-сплайна.
20. Порівняйте час розрахунку кривої Без'є та бі-сплайнової кривої на однаковому масиві опорних точок.
21. Яка з поверхонь (Без'є чи бі-сплайнова) потребує довшого часу розрахунку на однаковому масиві опорних точок? У скільки разів?
22. Охарактеризуйте узагальнені бі-сплайни.
23. Якого типу неперервність і якого порядку мають узагальнені бі-сплайни?
24. Дайте означення рівномірного та відкритого бі-сплайна.
25. Як можна звести бі-сплайн до сплайна Без'є?
26. Як конструюють нерівномірні раціональні бі-сплайни?
27. Опишіть властивості NURBS-кривих.
28. Для чого використовують евалюатор в OpenGL?
29. Які функції OpenGL використовують для обчислення точок на кривій Без'є у випадку нерівномірного розподілу значень параметра u?
30. Якими процедурами OpenGL описують побудову кривої Без'є у випадку рівномірного розподілу значень параметра u?
31. Які функції OpenGL використовують для обчислення точок на поверхні Без'є у випадку нерівномірного розподілу значень параметрів u та v?
32. Якими процедурами OpenGL описують побудову поверхні Без'є у випадку рівномірної за параметрами u та v сітки?
33. Які функції OpenGL визначають і формують NURBS-криву?
34. Як в OpenGL визначають і будують NURBS-поверхню?
35. Які функції OpenGL використовують для формування бі-сплайнових кривих і поверхонь?
36. Які функції OpenGL дають змогу вирізати отвори криволінійної форми у сформованій поверхні?

7.7. Приклади тестових питань

1. Якщо в точці спряження вимагати тільки збереження пропорційності компонент похідних з деяким коефіцієнтом k: $\mathbf{p}'(1) = k\ \mathbf{q}'(0)$, то складена крива володіє:

   а) параметричною неперервністю класу C^1;
   б) параметричною неперервністю класу C^2;
   в) геометричною неперервністю класу G^1.

2. Задано точки кривої Без'є $B_0(1,1)$, $B_1(2,3)$, $B_2(4,3)$, $B_3(3,1)$. Координати точок цієї кривої при u = 0 та u = 1 будуть такими:

   а) (1,1), (3,1);
   б) (1,1), $B_1(2,3)$;
   в) $B_2(4,3)$, $B_3(3,1)$.

3. Нулі поліномів Бернштейна розміщено:

   а) за межами інтервалу [0,1];
   б) у межах інтервалу (0,1);
   в) у точках $u = 0$ або $u = 1$.

4. Крива, яка є дуже хорошим наближенням кривої у формі Ерміта і яку можна порівнювати з інтерполяційним поліномом, сформованим на тому самому наборі опорних точок, – це:

   а) кубічний бі-сплайн;
   б) крива Без'є;
   в) NURBS-крива.

5. Під час конструювання кривих Без'є працюють:

   а) лише з координатами опорних точок;
   б) з координатами опорних точок та значеннями похідних у них;
   в) лише зі значеннями похідних в опорних точках.

6. Криві Без'є проходять:

   а) через деякі опорні точки;
   б) через всі опорні точки;
   в) не проходять через опорні точки.

7. Стикувальні функції кубічних бі-сплайнів:

   а) є гладкими і допускають локальний контроль за формою кривої;
   б) є гладкими і не допускають локального контролю за формою кривої;
   в) не є гладкими і допускають локальний контроль за формою кривої.

8. За допомогою виклику функції **glMap1*(type, u_min, u_max, stride, order, point_array)** налаштовують обробку:

   а) кривої Без'є у випадку рівномірного розподілу параметра;
   б) кривої Без'є у випадку нерівномірного розподілу параметра;
   в) поверхні Без'є.

9. За допомогою виклику функції **glMapGrid2*(nu, u1, u2, nv, v1, v2)** налаштовують обробку:

   а) поверхні Без'є у випадку рівномірного розподілу параметра;
   б) поверхні Без'є у випадку нерівномірного розподілу параметра;
   в) кривої Без'є.

10. NURBS-поверхні – це:

   а) двовимірні сплайни;
   б) тривимірні сплайни;
   в) чотиривимірні сплайни.

Розділ 8

АЛГОРИТМИ ФОРМУВАННЯ ЗОБРАЖЕННЯ

Під час створення реалістичних графічних зображень необхідно визначити, що видно на сцені з вибраної точки спостереження. Щоб відповісти на це запитання, для різних типів застосувань розроблено багато алгоритмів ефективного виявлення і відображення видимих об'єктів. На вибір методу впливають тип відображуваних об'єктів, їхня статичність чи динамічність. Усі такі алгоритми називають методами *дослідження видимих поверхонь* або інколи *усуненням невидимих поверхонь*, хоч відмінність між цими двома процесами може бути значною.

8.1. Алгоритми двовимірного відтинання

У загальному випадку будь-яку процедуру, яка усуває ті ділянки зображення, що розташовані всередині чи зовні заданої області простору, називають *алгоритмом відтинання* або просто *відтинанням*. Переважно відтинальна область – це прямокутник стандартної орієнтації, хоч в алгоритмі відтинання можна використовувати будь-яку форму.

Найчастіше відтинання застосовують у конвеєрі спостереження, де воно слугує для визначення ділянки сцени (дво- чи тривимірної) з метою відображення на пристрої виведення. Методи відтинання також використовують для захисту від накладання меж об'єктів, побудови об'єктів методами об'ємного моделювання, керування середовищем з декількома вікнами, а також переміщення, копіювання чи видалення ділянок зображення в різних програмах малювання.

Алгоритми відтинання застосовують до двовимірних процедур спостереження, щоб визначити ті частини зображення, які розташовані всередині відтинального вікна. Потім усе, що є поза відтинальним вікном, видаляють з опису сцени, який передають на пристрій виведення для відображення. Для ефективної реалізації відтинання у конвеєрі спостереження алгоритми застосовують до нормованих меж відтинального вікна. Це скорочує розрахунки, оскільки всі матриці геометричних перетворень і перетворень спостереження можна об'єднати й застосувати до опису сцени перед відтинанням. Обрізану сцену можна потім перевести в екранні координати для остаточного опрацювання.

Відтинання точки, лінії і багатокутника є стандартними компонентами графічних пакетів. Однак подібні методи можна застосувати й до інших об'єктів, зокрема конічних перерізів, еліпсів і сфер, а також до сплайнових кривих і

поверхонь. Однак переважно для скорочення обчислень об'єкти з криволінійними межами апроксимують прямими відрізками чи багатокутними поверхнями.

Якщо не зазначено інше, допускатимемо, що відтинальна область – це прямокутне вікно стандартної орієнтації, кути якого розташовані в точках з координатами xw_{min}, xw_{max}, yw_{min}, yw_{max}. Ці кути переважно відповідають нормованому квадрату, в якому значення x і y належать діапазону від 0 до 1 чи -1 до 1.

8.1.1. Двовимірне відтинання точки

Двовимірна точка $P(x, y)$ залишається для зображення, якщо виконуються нерівності: $xw_{min} \le x \le xw_{max}$, $yw_{min} \le y \le yw_{max}$. Якщо будь-яка з цих нерівностей не задовольняється, точка відтинається (не зберігається для відображення).

Хоч відтинання точки застосовують рідше, ніж відтинання лінії чи багатокутника, в деяких ситуаціях воно корисне, особливо коли зображення змодельовані системою багатьох частинок. Наприклад, відтинання точки можна застосувати до сцен, які містять хмари, морську піну, дим чи вибухи, які змодельовано "частинками" (координатами центрів маленьких кіл чи сфер).

8.1.2. Двовимірне відтинання ліній. Алгоритм Коена–Сазерленда

Суть задачі відтинання двовимірних відрізків пояснено на рис. 8.1. Припустимо, що вже виконано проектування і є двовимірний опис зображення в площині спостереження. На цій самій площині визначено і рамку відтинання, яка відповідає вікну на екрані дисплея. Як бачимо з рисунка, відрізок **AB** повністю потрапляє на екран, жодна з ділянок відрізків **EF**, **GH** не потрапляє. Відрізки **IJ** і **CD** необхідно вкоротити перед тим, як виводити на екран. Треба мінімізувати об'єм обчислень і обійтись без визначення точок перетину, яке обов'язково передбачає операцію ділення чисел з плаваючою крапкою. Історично першим, що відповідав цим вимогам, був *алгоритм Коена–Сазерленда (Cohen–Sutherland)*, в якому більшість операцій множення і ділення замінено операціями додавання та віднімання дійсних чисел і побітовими логічними операціями булевої алгебри.

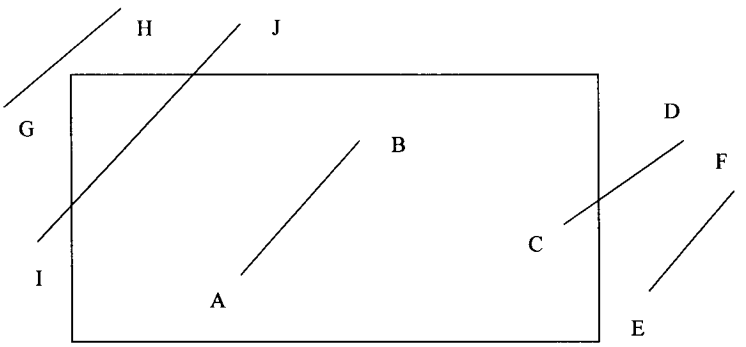


Рис. 8.1. Відтинання прямих відрізків з використанням стандартного прямокутного відтинального вікна

Сторони рамки відтинання можна пронумерувати в будь-якому порядку, наприклад, за допомогою двійкових розрядів їх нумерують так: 1 (0001) – ліва межа, 2 (0010) – права, 3 (0100) – нижня, 4 (1000) – верхня. Внаслідок такого впорядкування крайній справа розряд (біт 1) відповідає лівій межі відтинального вікна, а крайній справа розряд (біт 4) – верхній. Отож, наприклад, кінцевій точці відрізка, яка розташована нижче і зліва від відтинального вікна, присвоюють код області 0101. Значення 1 у будь-якому розряді вказує, що кінцева точка відрізка розташована поза відповідною межею вікна, а значення 0 – що кінцева точка знаходиться не зовні, а всередині відповідної межі або на ній. Кожна сторона відтинального вікна ділить площину на внутрішню частину і зовнішню. Чотири межі вікна формують 9 підобластей, на рис. 8.2 перелічено значення двійкових кодів у всіх цих підобластях:

1001	1000	1010
0001	0000 відтинальне вікно	0010
0101	0100	0110

Рис. 8.2. Двійкові коди, які ідентифікують положення кінцевої точки відрізка щодо меж відтинального вікна

Двійкові значення коду області визначають порівнянням координат (x, y) кінцевої точки з межами відтинального прямокутника. Біт 1 покладають рівним 1, якщо $x < x_{wmin}$, біт 2 – якщо $x > x_{wmax}$, біт 3 – якщо $y < y_{wmin}$, біт 4 – якщо $y > y_{wmax}$, де рівняння $x = x_{wmin}, x = x_{wmax}, y = y_{wmin}, y = y_{wmax}$ описують відповідно ліву, праву, нижню і верхню межі відтинального вікна.

Під час аналізу відрізка спочатку визначають, у яких областях розташовані його кінцеві точки, і їм присвоюють відповідні характеристичні коди. Ця процедура потребує виконання 8 операцій віднімання на кожен відрізок.

Розглянемо відрізок, кінцеві точки якого мають характеристичні коди $o_1 = outcode(x_1, y_1)$ і $o_2 = outcode(x_2, y_2)$. Можливі 4 варіанти поєднання характеристичних кодів двох кінцевих точок (рис. 8.1).

1. $o_1 = o_2 = 0$. Обидві кінцеві точки лежать усередині рамки відтинання – цей випадок ілюструє відрізок **AB**. Відрізок може бути переданий для виконання растрового перетворення і відображення на екрані.
2. $o_1 \neq 0, o_2 = 0$ або навпаки. Одна точка розташована всередині рамки відтинання, а інша поза нею (відрізок **CD**). Відмінний від 0 характеристичний код показує, в якій зовнішній області розташована одна з кінцевих точок і яку саме межу (чи межі) рамки перетинає відрізок. У цьому випадку треба обчислити одну або дві точки перетину відрізка з межами рамки.
3. $o_1 \wedge o_2 \neq 0$. За результатом побітової операції **AND** над характеристичними кодами крайніх точок можна з'ясувати, чи лежать вони з одного боку від

межі рамки, чи з різних. Якщо результат відмінний від 0, то ці точки лежать з одного боку від якоїсь межі, а отже, весь відрізок лежить поза рамкою відтинання, і його можна відкинути (**EF** на рис. 8.1).

4. $o_1 \wedge o_2 = 0$. Обидві кінцеві точки лежать поза рамкою відтинання, але з різних боків від обох її меж (відрізки **GH**, **IJ**), і не можна з певністю сказати, чи перетинає відрізок зону видимості. Потрібен детальніший аналіз: треба обчислити точку перетину з однією межею і проаналізувати характеристичні коди крайніх точок двох нових відрізків.

Розглянемо, як обчислити точки перетину відрізка з межами рамки. Вибір методу залежить від форми зображення відрізка в програмі, але в будь-якому випадку потрібна, як мінімум, одна операція ділення дійсних чисел. Щоб визначити точки перетину рамки відрізком, можна використати рівняння прямої з кутовим коефіцієнтом. Для прямої з координатами кінцевих точок (x_0, y_0), (x_1, y_1) координату y точки перетину з вертикальною відтинальною межею знаходять як $y = y_0 + m(x - x_0)$, де значення x покладають рівним $xw_{\min}$ або $xw_{\max}$, а тангенс кута нахилу прямої обчислюють за формулою $m = (y_1 - y_0)/(x_1 - x_0)$. Якщо ж шукають точку перетину з горизонтальною межею, координату x можна обчислити як $x = x_0 + (y - y_0)/m$, де y дорівнює $yw_{\min}$ або $yw_{\max}$.

Розширення цього алгоритму на тривимірну ситуацію очевидне, а детальніше тривимірне спостереження розглянемо в підрозділі 8.2.1.

8.1.3. Відтинання багатокутників

Графічні пакети зазвичай підтримують лише зафарбовані області, які є багатокутниками, причому часто – тільки опуклими. Можна розробити алгоритм відтинання багатокутників, узявши за основу алгоритм відтинання відрізків і застосувавши його послідовно до всіх ребер багатокутника, пам'ятаючи при цьому, що під час відтинання може бути сформовано декілька нових об'єктів-багатокутників.

Якщо застосувати до *увігнутого багатокутника* відтинання прямокутною рамкою, то одержимо кілька нових багатокутників (рис. 8.3, *а*, *б*). На жаль, реалізувати модуль відтинання, який був би здатний з одного об'єкта сформувати кілька, доволі складно. Тому частіше результат відтинання подають у програмі як єдиний багатокутник, для чого об'єднують фрагменти в один об'єкт (рис. 8.3, *в*). У такому багатокутнику деякі ребра накладаються одне на одне, що інколи стає джерелом проблем під час розв'язування інших задач.

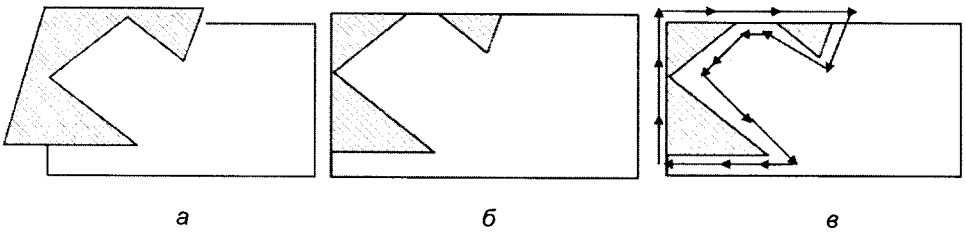


Рис. 8.3. Відтинання увігнутого багатокутника

У разі відтинання одного *опуклого багатокутника* іншим опуклим багатокутником не може утворитися більше ніж один опуклий багатокутник. Тому в більшості графічних систем або дозволено формувати лише опуклі багатокутники, або є засоби розтинання багатокутника загального вигляду на опуклі (в OpenGL така функція входить до складу бібліотеки GLU).

8.1.4. Метод Сазерленда–Ходгмана

Для відтинання багатокутників прямокутною рамкою можна використати алгоритм відтинання відрізків Коена–Сазерленда, застосовуючи його в циклі до ребер багатокутника. Однак у системах з конвеєрною архітектурою є ефективнішим підхід, який запропонували Сазерленд і Ходгман (*Hodgman*). Вхідними даними для модуля відтинання відрізка є дві пари координат кінцевих точок початкового відрізка, а вихідними – дві пари координат кінцевих точок ділянки цього відрізка, яка лежить усередині зони відтинання; якщо відрізок не перетинає зони відтинання, то нічого не буде сформовано (рис. 8.4, *а*).

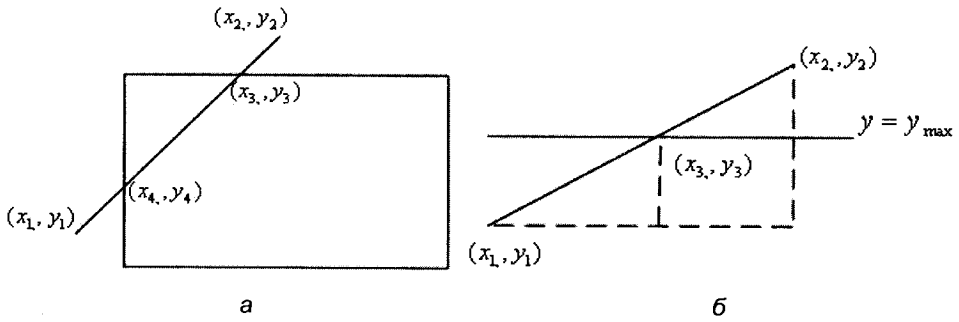


Рис. 8.4. Відтинання відрізка прямокутною рамкою (а) та відтинання верхньою межею рамки (б)

Розглядатимемо рамку відтинання як об'єкт, сформований перетином чотирьох прямих, що відповідають верхній, нижній, правій і лівій межам видимості. Тоді й модуль відтинання відрізка можна подати як послідовність чотирьох простіших модулів (конвеєр), кожен із яких працює лише з однією межею.

Розглянемо відтинання верхньою межею рамки. Модуль, що виконує цю операцію, отримує на вході координати крайніх точок відрізка і значення y_{max} як

параметр, що задає зону відтинання (рис. 8.4, *б*). Легко показати, використовуючи подібність трикутників, що в разі перетину відрізком межі рамки координати точки перетину визначають за формулами

$$x_3 = x_1 + (y_{\max} - y_1)\frac{x_2 - x_1}{y_2 - y_1},\ y_3 = y_{\max}.$$

У підсумку вихідними даними модуля відтинання буде одна з трьох пар координат $\{(x_1, y_1), (x_2, y_2)\}$, $\{(x_1, y_1), (x_3, y_{\max})\}$ або $\{(x_3, y_{\max}), (x_2, y_2),\}$.

За цією самою схемою організують і модулі відтинання нижньої, правої і лівої меж рамки, тільки у відповідних формулах використовують інші параметри рамки, а координати x і y можна міняти місцями.

Під час обробки багатокутників загального вигляду, які мають достатньо багато вершин, можна значно скоротити затрати, провівши попередній аналіз з використанням *прямокутної оболонки* складного об'єкта. Така оболонка – це прямокутник мінімального розміру зі сторонами, паралельними до меж рамки відтинання, в яку вписується об'єкт. Потім можна застосувати алгоритм відтинання до оболонки, що працює значно швидше, бо, по-перше, оболонка має тільки 4 сторони, а, по-друге, ці сторони паралельні до меж рамки.

8.2. Тривимірне відтинання

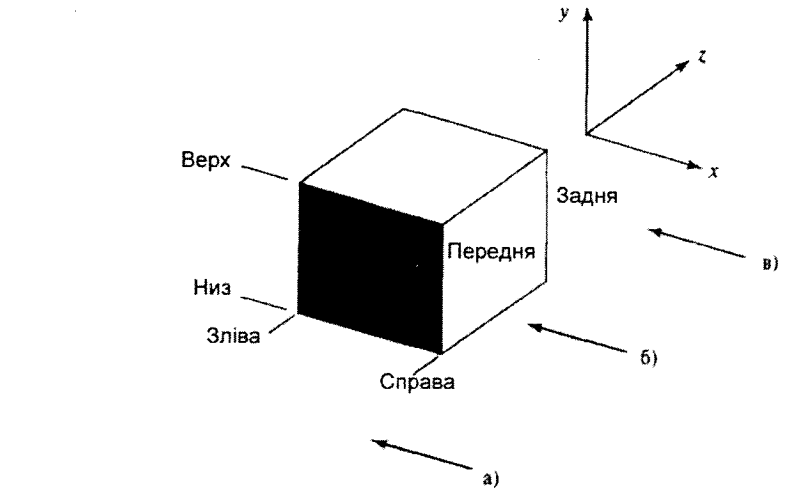
У тривимірному просторі під час відтинання аналізують положення об'єкта щодо просторової зони видимості.

8.2.1. Алгоритми тривимірного відтинання

Найпростіше розглянуті двовимірні алгоритми поширити на тривимірний простір у тому випадку, коли зона видимості має форму паралелепіпеда, заданого співвідношеннями $x_{\min} \le x \le x_{\max}$, $y_{\min} \le y \le y_{\max}$, $z_{\min} \le z \le z_{\max}$.

Алгоритми відтинання Коена–Сазерленда та Сазерленда–Ходгмана можна модифікувати стосовно тривимірного випадку. Зокрема, модифікація алгоритму Коена–Сазерленда полягає в тому, що замість 4-розрядного характеристичного коду використовують 6-розрядний. Одному з додаткових розрядів надають значення 1, якщо точку розташовано перед передньою відтинальною площиною, другому – якщо точка перебуває за задньою відтинальною площиною (рис. 8.5, *а*, *в*), якщо точка належить об'єму спостереження, додатковим розрядам присвоюють нульові значення. Методика аналізу та прийняття рішення нічим не відрізняється від двовимірного випадку.

Щоб спростити відтинання довільних багатогранників, багатокутні поверхні часто ділять на трикутні ділянки і описують трикутними фрагментами. У такому випадку для відтинання можна використати підхід Сазерленда–Ходгмана. Кожна смуга з трикутників опрацьовується послідовно шістьма площинами відтинання, і на виході отримують кінцевий список вершин смуги.



011001	011000	011010
010001	010000	010010
010101	010100	010110

а

001001	001000	001010
000001	000000	000010
000101	000100	000110

б

101001	101000	101010
100001	100000	100010
100101	100100	100110

в

Рис. 8.5. Тривимірна модифікація алгоритму Коена–Сазерленда

8.2.2. Додаткові площини відтинання в OpenGL

Крім шести площин відтинання, що утворюють об'єм спостереження, OpenGL дає змогу задавати на сцені додаткові площини відтинання. На відміну від площин відтинання об'єму спостереження, кожна з яких перпендикулярна до однієї з координатних осей, ці додаткові площини можуть мати будь-яку орієнтацію.

Щоб увести додаткову площину й активізувати відтинання по ній, використовують оператори:

```
glClipPlane(id, planeParameters);
glEnable(id);
```

параметр **id** – це ідентифікатор площини відтинання; йому присвоюють одне зі значень **GL_CLIP_PLANE0, GL_CLIP_PLANE1** і так далі до максимальної глибини. Саму площину визначають, використовуючи чотириелементний масив **planeParameters**, який складається з чисел подвійної точності з плаваючою крапкою – параметрів *A, B, C* і *D* рівняння площини: $Ax+By+Cz+D=0$; за замовчуванням параметрам відтинальної площини присвоєно нульові значення для всіх додаткових площин, крім того, спочатку всі додаткові площини відключено.

Щоб відключити активовану площину з ідентифікатором **id**, використовують вираз:

```
glDisable(id);
```

Параметри площини *A, B, C* і *D* буде перетворено на координати спостереження і використано для перевірки в координатах спостереження точок сцени. Наступні зміни параметрів спостереження чи геометричних перетворень не впливають на записані параметри площини. Крім того, оскільки процедури відтинання для цих площин застосовують у координатах спостереження, а не в нормованому координатному просторі, продуктивність програми може знизитись у разі активації додаткових площин відтинання.

Будь-яка точка, розташована за активованою площиною відтинання, буде видалена, тобто точка з координатами в системі (x, y, z) відтинається, якщо вона задовольняє умову $Ax+By+Cz+D<0$.

У будь-якій реалізації OpenGL доступні 6 додаткових площин відтинання, але їх може бути й більше. Щоб довідатися, скільки додаткових площин відтинання доступні у конкретній реалізації, використовують запит:

```
glGetIntegerv(GL_MAX_CLIP_PLANES, numPlanes);
```

параметр **numPlanes** – це ім'я цілочисельної змінної, в яку буде записано значення, що дорівнює кількості додаткових площин відтинання, які можна використовувати.

8.3. Методи дослідження видимих поверхонь

Алгоритми дослідження видимих поверхонь класифікують відповідно до того, що в них фігурує – визначення об'єктів чи їхні спроектовані образи. Ці підходи називають методами *простору об'єктів* і *простору зображень* відповідно. У першій групі методів об'єкти та їхні частини порівнюють один з одним у контексті запропонованого визначення сцени, і в підсумку деякі поверхні позначають як видимі. Це й алгоритми відображення ліній, зокрема ідентифікація видимих ліній каркасних зображень. У другій групі методів видимість визначають для кожного пікселя на площині проекції. У більшості алгоритмів виявлення видимих поверхонь використовують методи простору зображень.

Базові підходи, прийняті в різних алгоритмах виявлення видимих поверхонь, можуть значно відрізнятися, однак переважно для підвищення продуктивності використовують методи сортування і когерентності. Сортування впорядковує окремі поверхні на сцені за відстанню до площини спостереження. Методи когерентності побудовано за принципом регулярності структур на сцені. Окремий рядок розгортки дуже часто містить інтервали (серії) пікселів постійної інтенсивності, крім того, структура рядка розгортки часто зберігається після переходу до наступного рядка. Кадри з анімацією, наприклад, змінюються лише в околі рухомих об'єктів; між об'єктами на сцені можна встановити постійні зв'язки.

8.3.1. Локалізація невидимих поверхонь

Найпростішою перевіркою на видимість є *алгоритм дослідження задніх граней*, швидкий і ефективний для початкового відсіювання, який усуває з подальших перевірок на видимість значну кількість багатокутників. Для опуклого багатогранника виявлення задніх граней виключає всі невидимі поверхні, але в загальному випадку це не дає змоги повністю знайти всі невидимі поверхні.

Якщо сцена складається тільки з опуклих багатокутників, то їхні внутрішні грані спостерігач ніколи не побачить. Але й частина зовнішніх граней (так звані *нелицьові грані*) також не буде видимою. Врешті-решт нелицьові грані будуть перекриті іншими, і це зможе виявити універсальний алгоритм усунення невидимих граней, але можна спростити його роботу, зразу ж викресливши нелицьові грані.

Принцип *усунення нелицьових граней* полягає в такому. Зовнішній бік багатокутника спостерігач побачить тільки тоді, коли нормаль до площини багатокутника буде спрямована до спостерігача. Позначимо через θ кут між зовнішньою нормаллю і напрямком на спостерігача, тоді лицьовою грань об'єкта буде тільки у випадку, якщо $-90^\circ \le \theta \le 90^\circ$, тобто $\cos\theta \ge 0$. Другу умову проаналізувати легше, бо знак косинуса визначають за скалярним добутком, і умова виглядає так: $\mathbf{N} \cdot \mathbf{V} \ge 0$. Якщо ж врахувати, що цей аналіз виконують переважно після перетворення на нормалізовану систему координат пристрою відображення, в якій напрямок на спостерігача задано віссю z, а рівняння площини, де лежить багатокутник, має вигляд $Ax+By+Cz+D=0$, то, щоб визначити, чи є грань лицьовою, треба перевірити знак коефіцієнта C (він має бути додатним).

8.3.2. Алгоритм буфера глибини

Одним із найпоширеніших алгоритмів усунення видимих поверхонь у просторі зображень є алгоритм Z-буфера, або алгоритм буфера глибини, в якому для кожного пікселя на поверхні проекції порівнюють значення глибин поверхонь на сцені. Кожну поверхню сцени опрацьовують окремо, при цьому розглядають усі пікселі. Алгоритм застосовують переважно до сцен, які містять лише багатокутні поверхні, бо для них глибини можна обчислити дуже швидко, і метод легко реалізувати.

На рис. 8.6 показано три поверхні з різною відстанню по лінії ортографічної проекції від точки (x, y). Ці поверхні можна досліджувати в довільній послідовності. Під час опрацювання кожної поверхні її глибину від площини спостереження порівнюють із глибинами раніше опрацьованих поверхонь. Якщо поверхня ближча до точки спостереження, ніж усі раніше опрацьовані поверхні, її колір обчислюють і записують разом із глибиною. Після завершення обробки всіх поверхонь набір записаних кольорів поверхні визначає видимі поверхні на сцені. Зазначимо, що цей алгоритм переважно реалізують у нормованих координатах, тому значення глибини змінюються від 0 біля ближньої площини відтинання (площини спостереження) до 1 біля дальньої.

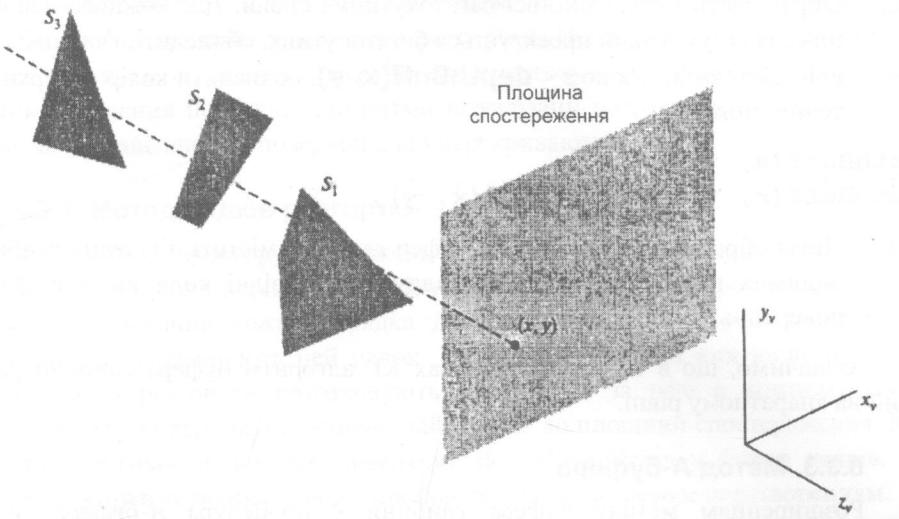


Рис. 8.6. В один піксель площини спостереження проектуються три точки накладених одна на одну поверхонь. Найменше значення глибини має видима поверхня S_1

Для роботи методу потрібно *два* буфери. Під час опрацювання поверхні глибини для кожної точки (x, y) записують у буфер глибини, а в буфері кадрів зберігають код кольору кожного пікселя. Спочатку всі позиції в буфері глибини мають значення 1 (максимальна глибина), а буфер кадрів (буфер регенерації) ініціалізований кольором фону. Потім опрацьовують усі поверхні, перераховані в таблицях багатокутників, по одному рядку розгортки за раз, і при цьому для кожного пікселя (x, y) розраховують глибину, яку потім порівнюють зі значенням, раніше записаним у буфері кадрів для цього пікселя. Якщо обчислена глибина менша від тієї, що зберігається в буфері глибини, то в нього записують нове значення; це означає, що поточний багатокутник закриває опрацьовані раніше, тому його код кольору повинен замінити в буфері кадрів код кольору, сформований раніше для цього пікселя. Якщо ж відстань до поточного багатокутника більша за значення, що зберігається в буфері глибини, отже, раніше був опрацьований багатокутник, який розміщений ближче до спостерігача і тому закриває поточний. Отож, ця точка поточного багатокутника не буде видима, й інформацію про колір не треба заносити в буфер кадрів.

У наведеному далі алгоритмі припущено, що значення глибини нормовані в проміжку 0–1, причому площині спостереження відповідає глибина 0.

Алгоритм буфера глибини

1. Ініціалізувати буфер глибини і буфер кадрів так, щоб для всіх позицій (x, y) буферів :

```
depthBuff(x, y) = 1.0;
frameBuff(x, y) = backgndColor;
```

2. Опрацювати послідовно всі багатокутники сцени. Для кожного положення пікселя (x, y), в який проектується багатокутник, обчислити глибину z (якщо вона невідома). Якщо $z <$ **depthBuff(x, y)**, обчислити колір поверхні в цій точці і покласти

```
depthBuff(x, y) = z;
frameBuff(x, y) = surfColor(x, y);
```

3. Після обробки всіх поверхонь буфер глибини містить значення глибин для видимих поверхонь, а буфер кадру – відповідні коди кольору для цих поверхонь.

Зазначимо, що в складних системах КГ алгоритм буфера глибини реалізований на апаратному рівні.

8.3.3. Метод А-буфера

Розширенням методу буфера глибини є процедура *А-буфера* (в назві використано протилежну до z букву алфавіту, а z є глибиною). Це метод із захистом від накладання, усередненням по області і виявленням видимих поверхонь. Область буфера для цієї процедури називають *буфером нагромадження*, бо в ній на доповнення до значень глибин зберігають різні дані про поверхню. Недоліком методу буфера глибини є те, що він визначає лише одну видиму поверхню в кожному пікселі. Інакше кажучи, він працює лише з непрозорими поверхнями і не може нагромаджувати коди кольорів для декількох поверхонь, а це необхідно, якщо треба відобразити прозору поверхню. Метод А-буфера розширює алгоритм буфера глибини, щоб кожній позиції в ньому міг відповідати цілий список поверхонь. Це дає змогу обчислити колір пікселя як комбінацію кольорів різних поверхонь, розраховуючи прозорість чи захист від накладань.

Кожна позиція в А-буфері має два поля:

- поле глибини, в якому зберігається дійсне значення (додатне, від'ємне чи нуль);
- поле даних, в якому містяться дані про поверхню чи вказівник.

Якщо поле глибини невід'ємне, то число, записане в цій позиції, – це глибина поверхні, яка проектується у відповідну область пікселя. Наступне за ним поле даних про поверхню містить таку інформацію: її колір для цієї точки і відсоток охоплення пікселя. Якщо поле глибини для позиції в А-буфері від'ємне, то колір пікселя визначається внесками декількох поверхонь. Наступне за ним поле містить вказівник на зв'язаний список даних про поверхню. Інформація про поверхню в А-буфері містить таке: значення інтенсивності RGB-компонент; параметр непрозорості (відсоток прозорості); глибину; відсоток охоплення площі; ідентифікатор поверхні; інші параметри, необхідні для візуалізації поверхні.

Схему дослідження видимих поверхонь в А-буфері можна реалізувати за допомогою методів, подібних до наведених для алгоритму буфера глибини. Щоб

визначити, яку частину площі пікселя вздовж окремих рядків розгортки займає поверхня, треба опрацювати всі рядки розгортки. Поверхні ділять на багатокутну сітку і обрізають по межах пікселів. Використовуючи параметри прозорості і відсоток охоплення поверхні, алгоритми візуалізації розраховують колір кожного пікселя як середнє внесків поверхонь, що перекриваються.

8.3.4. Метод рядків розгортки

Цей метод простору зображень призначений для виявлення видимих компонент поверхонь і порівняння глибин уздовж різних рядків розгортки на сцені. Під час опрацювання кожного рядка досліджують проекції усіх багатокутних поверхонь, що перетинають цей рядок, і визначають, яка з них видима. Вздовж кожного рядка розгортки розраховують глибину, а за нею в кожному пікселі визначають, яку поверхню розміщено найближче до площини спостереження. Коли для пікселя визначено видиму поверхню, її колір вводять у буфер кадрів. Цей алгоритм поєднує усунення невидимих поверхонь із растровим перетворенням.

Поверхні опрацьовують з використанням інформації, записаної в таблиці багатокутників. Координати кінцевих точок усіх відрізків на сцені, обернених нахилів кожного відрізка і вказівників на таблицю граней, які визначають поверхні, обмежені кожним відрізком, записано в таблиці сторін. Таблиця граней містить коефіцієнти площини, властивості матеріалів поверхні, інші дані про поверхню і, можливо, посилання на таблицю сторін. Щоб полегшити пошук поверхонь, які перетинає рядок розгортки, для кожного рядка розгортки під час опрацювання формують активний список сторін, у якому за зростанням x записано сторони, які перетинаються з ним. Крім того, для кожної поверхні визначають мітку (*flag*), яку встановлюють у розташування "включено" або "виключено" і яка вказує, де розміщено точку рядка розгортки – всередині чи зовні поверхні. Розташування пікселів уздовж кожного рядка розгортки опрацьовується зліва направо. В крайній зліва точці перетину з плоскою проекцією опуклого багатокутника мітка поверхні включається, а в крайній справа точці перетину вздовж рядка розгортки виключається. Для рядків вгнутого багатокутника перетини з рядками розгортки теж можна сортувати зліва направо, а мітку поверхні включати між парою перетинів.

Метод рядків розгортки проілюструємо на рис. 8.7. Розглянемо два багатокутники, які перетинаються в просторі. Переміщаючись вздовж рядка растру i, перетинаємо ребро a багатокутника **A**. Оскільки це перший багатокутник, який перетинає рядок, немає сенсу виконувати якісь обчислення, що аналізують його глибину. Починаючи з цього пікселя, наступним присвоюють код кольору, що відповідає зафарбуванню багатокутника **A**. Але якщо під час переміщення по рядку трапиться наступне ребро b цього багатокутника, то це означає, що всім наступним пікселям можна присвоїти код кольору фону. Так триватиме доти, поки не трапиться ребро c багатокутника **B**. Оскільки перед цим на рядку відображався фон, можна знову не брати до уваги інформації про глибину і вважати, що наступна ділянка рядка належить образу багатокутника **B**.

Складнішою є ситуація на рядку *j*. Спочатку знову знаходимо ребро *a* і, не аналізуючи глибину, можемо, починаючи з цього пікселя, присвоювати наступним код кольору, що відповідає зафарбуванню багатокутника **A**. Але далі трапляється ребро іншого багатокутника, і потрібно аналізувати глибину для всіх пікселів рядка *j*, поки не трапиться ребро *d*.

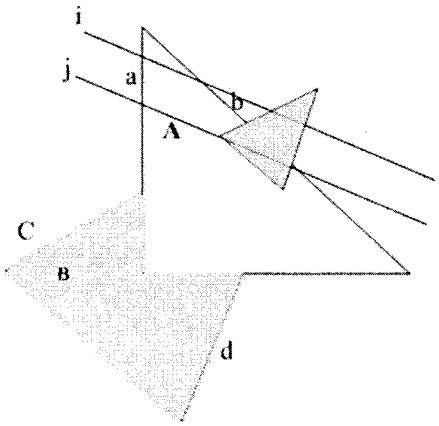


Рис. 8.7. Метод рядків розгортки

Хоча на перший погляд видається, що цей алгоритм нагадує алгоритм Z-буфера, між ними є одна принципова відмінність: алгоритм рядків розгортки під час обробки кожного пікселя повинен опрацьовувати всі багатокутники, що претендують на відображення в цій зоні екрана, а алгоритм Z-буфера в кожний момент працює лише з одним багатокутником. Реалізація цього алгоритму потребує уважно продуманої організації даних про багатокутники, яка дасть змогу швидко відбирати серед них ті, що претендують на відображення в поточному рядку. Найчастіше структура даних ґрунтується на масиві вказівників, по одному на рядок, кожен з них вказує на спеціальну структуру опису ребер багатокутників, які перетинає цей рядок.

За допомогою описаного методу можна опрацювати довільну кількість накладених багатокутних поверхонь. Для них встановлюють мітки, які вказують, де розміщено точку – всередині чи ззовні поверхні, а глибину розраховують лише на краях накладених поверхонь. Ця процедура коректно працює, коли немає циклічного накладання; якщо ж воно є, то щоб його усунути, поверхні можна поділити на частини.

8.3.5. Метод сортування за глибиною (алгоритм художника)

Припустимо, що є набір багатокутників, відсортованих за відстанню від них до спостерігача. Для коректного відображення такої сцени можна сформувати образи об'єктів від найбільш віддаленого до найменш віддаленого. У результаті образи об'єктів, розміщених ближче до спостерігача, *автоматично* перекриють

образи об'єктів, розміщених далі, і немає необхідності в аналізі перекривання. Для такого методу є спеціальний термін – *відображення багатокутників від дальнього до ближнього*. Але реалізація цього підходу потребує вирішення двох проблем: як посортувати багатокутники в просторі сцени і що робити у разі перетину багатокутників. Обидві ці проблеми вирішують методом сортування за глибиною.

Використовуючи процедури, визначені в просторі об'єктів і просторі зображень, за методом сортування за глибиною реалізують такі базові функції:

- поверхні сортують у порядку зменшення глибини (у двох просторах);
- їх впорядковують, починаючи з найглибшої (лише в просторі зображень).

Спочатку поверхні сортують за їхньою відстанню до площини спостереження, потім у буфер регенерації вносять коди кольорів найдальшої. Опрацьовуючи кожну наступну поверхню (за зменшенням глибини) до буфера кадрів заносять колір поверх кольорів раніше опрацьованих поверхонь.

Припустимо, що для кожного багатокутника обчислено параметри *прямокутної оболонки* (*обмежувального прямокутника*), в якій він міститься. Наступна операція – посортувати всі багатокутники за ступенем віддаленості від спостерігача, причому за параметром сортування беруть максимальне значення z-координати оболонки. Якщо мінімальна глибина $z_{\min}$ для цього багатокутника більша, ніж максимальна глибина наступного, то після формування їхніх образів на екрані у зворотній послідовності буде отримано цілком коректне зображення сцени. Першим формують образ багатокутника, розміщеного позаду всіх інших.

Впорядковане малювання багатокутних поверхонь у буфері кадрів виконують у декілька етапів. Припустивши, що сцену спостерігають у напрямку осі z, під час першого проходу алгоритму поверхні впорядковують за найменшим значенням z кожної. Потім поверхню S у кінці списку (з найбільшою глибиною) порівнюють з іншими поверхнями списку і визначають, чи перекривається вона з ними за глибиною. Якщо ні, то S є найвіддаленішою поверхнею, і її перетворюють на стандарт розгортки. Потім цей процес повторюють для наступної поверхні списку. Якщо перекриття не спостерігаємо, поверхні обробляємо у порядку, визначеному глибиною, поки вони всі не будуть перетворені на стандарт розгортки.

Якщо для якоїсь позиції списку виявляють перекриття за глибиною, необхідні додаткові порівняння, щоби з'ясувати, чи не треба перевпорядкувати якісь поверхні. Тоді лише значень z-координати оболонок недостатньо, порядок формування їхніх образів визначають індивідуальним порівнянням кожної пари таких об'єктів; при цьому необхідно використати багато складних перевірок.

Ще складнішими є такі ситуації. По-перше, три багатокутники можуть *перекривати* один одного *циклічно*. В цьому випадку немає такого порядку формування образів, який дав би змогу отримати коректне зображення. Можна хіба поділити хоча б один із них на дві частини і спробувати проаналізувати утворений набір із 4 багатокутників. По-друге, можлива ситуація, коли один багатокутник пронизує інший багатокутник. У такій ситуації доведеться детально проаналізувати

перетин, використовуючи узагальнені методи відтинання одного багатокутника загального вигляду іншим аналогічним багатокутником.

8.4. Методи визначення видимості для каркасних зображень

Часто тривимірну сцену треба зобразити у вигляді контурів, щоб швидко отримати деталі об'єкта. Найшвидший спосіб одержання каркасного вигляду сцени – це відобразити всі грані об'єктів. Однак у такому зображенні може бути важко ідентифікувати передні і задні елементи об'єкта.

Одним з вирішень цієї проблеми є застосування *загасання з глибиною* (*depth cueing*), щоб відображена інтенсивність відрізка була функцією його відстані від спостерігача. Цей метод переважно застосовують, використовуючи лінійну функцію

$$f_{depth}(d) = \frac{d_{\max} - d}{d_{\max} - d_{\min}}, \tag{8.1}$$

де d – відстань до точки від спостерігача; можна вважати, що значення максимальної та мінімальної глибин $d_{\max}, d_{\min}$ дорівнює величинам, зручним у конкретному застосуванні, або задати, допускаючи нормований діапазон частот: $d_{\max} = 1.0, d_{\min} = 0.0$. Під час опрацювання кожного пікселя його колір множиться на $f_{depth}(d)$, при цьому ближні точки відображаються з більшою інтенсивністю, а точки на максимальній глибині мають інтенсивність 0.

Функцію загасання з глибиною можна реалізувати з різними варіаціями. У деяких графічних бібліотеках доступна загальна атмосферна функція (підрозділ 9.6.4), яка дає змогу об'єднати загасання з глибиною і атмосферні ефекти, наприклад, для імітації диму чи туману, тому колір об'єкта можна модифікувати функцією загасання з глибиною, а потім об'єднати з кольором атмосфери.

Як альтернативу можна використовувати перевірки видимості й або усунути сховані відрізки, або зобразити їх інакше, ніж видимі межі. Процедури визначення видимості меж об'єктів називають *методами дослідження видимості для каркасних зображень, методами дослідження видимих ліній* або *методами дослідження невидимих ліній*.

Крім того, деякі методи дослідження видимих поверхонь, розглянуті в попередньому підрозділі, можна використовувати для перевірки видимості меж. Зокрема, використовуючи алгоритм дослідження задніх граней, можна визначити всі задні поверхні об'єкта і відобразити лише межі видимих поверхонь. Використовуючи сортування за глибиною, поверхні можна нарисувати в буфері регенерації так, щоб їхні внутрішні частини мали колір фону, а межі – колір переднього плану. Якщо опрацьовувати поверхні від заднього плану до переднього, невидимі лінії будуть закриті ближчими поверхнями. Для відображення точок перетину з рядками розгортки на межах видимих поверхонь можна використовувати методи рядків розгортки.

Прямий підхід до визначення видимих відрізків полягає в порівнянні розташувань країв з розташуваннями поверхонь на сцені, тут використовують ті самі методи, що і в алгоритмах відтинання ліній, але у разі перевірки на видимість додатково треба порівняти глибини меж і поверхні.

8.5. Функції дослідження видимих поверхонь в OpenGL

Методи видалення невидимих граней і методи перевірки видимості з використанням буфера глибини можна застосувати до сцен, скориставшись функціями з основної бібліотеки. Крім того, функції OpenGL застосовують для побудови каркасного зображення сцени з віддаленими невидимими лініями і відображення сцен із загасанням з глибиною.

8.5.1. Функції вибору багатокутників

Видаляють *задні грані* за допомогою функції

```
glEnable(GL_CULL_FACE);
glCullFace(mode);
```

де параметру **mode** присвоюють значення **GL_BACK** (значення за замовчуванням); тобто, якщо активізувати функцію **glEnable** без явного виклику функції **glCullFace**, зі сцени будуть видалені задні грані.

Фактично цю функцію можна використати і для видалення передніх граней, до того ж, можна забрати і передні, і задні грані. Якщо точка спостереження розташована, наприклад, всередині приміщення, то спостерігатимемо лише задні сторони об'єкта (внутрішні частини кімнати). У цьому випадку можна або присвоїти параметру **mode** значення **GL_FRONT**, або змінити визначення передніх багатокутників, використовуючи функцію **glFrontFace**, розглянуту в підрозділі 3.2.3. Потім, якщо точка спостереження переміщується з приміщення, можна забрати задні грані.

Крім того, в деяких застосуваннях може бути вимога відобразити на сцені лише певні примітиви, такі як набори точок і окремі відрізки. Щоб видати всі багатокутні поверхні зі сцени, параметру **mode** присвоюють значення **GL_FRONT_AND_BACK.**

Процедуру вибору відключають командою

```
glDisable(GL_CULL_FACE);
```

8.5.2. Функції буфера глибини

Щоб використати процедури дослідження видимих поверхонь за допомогою буфера глибини, спочатку необхідно модифікувати функцію ініціалізації GLUT для режиму відображення, щоби вона містила запит до буфера глибини і буфера регенерації. Це можна зробити, наприклад, за допомогою оператора

```
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
```

потім значення буфера глибини ініціалізують функцією

```
glClear(GL_DEPTH_BUFFER_BIT);
```

Зазвичай буфер глибини ініціалізують тим самим оператором, що й буфер регенерації з кольором фону, для того ж буфер глибини необхідно очищати щоразу, коли треба відобразити на екрані новий кадр. Зазначимо, що в OpenGL значення глибини нормують у діапазон 0–1, тому попередня ініціалізація за замовчуванням встановлює всі значення на максимум (1).

Процедури дослідження видимих поверхонь за допомогою буфера глибини активують і деактивують функціями

```
glEnable(GL_DEPTH_TEST);
glDisable(GL_DEPTH_TEST);
```

Перевіряти видимість за допомогою буфера глибини можна також з іншими вихідними значеннями максимальної глибини, які задають функцією

```
glClearDepth(maxDepth);
```

тут параметру **maxDepth** можна присвоїти будь-яке значення між 0 і 1. Щоб завантажити його в буфер глибини, треба викликати функцію

```
glClear(GL_DEPTH_BUFFER_BIT);
```

інакше буфер глибини ініціалізуватиметься зі значенням за замовчуванням (1.0).

Оскільки розрахунок кольору поверхні та інші опрацювання не виконуються для об'єктів, розміщених за заданою максимальною глибиною, цю функцію можна використовувати для прискорення процедур буфера глибини, коли сцена містить багато віддалених об'єктів, розташованих позаду об'єктів переднього плану.

В OpenGL координати проекції нормують у діапазон від –1.0 до 1.0, і значення глибини між ближньою (площина проекції) і дальньою площинами відтинання потім додатково нормують у діапазон від 0.0 до 1.0. Використовуючи функцію

```
glDepthRange(nearNormDepth, farNormDepth);
```

можна обмежити перевірки за допомогою буфера глибини будь-якою областю об'єму спостереження і навіть поміняти місцями ближню і дальню площини відтинання. За замовчуванням **nearNormDepth**=0.0, **farNormDepth**=1.0, але тут можна присвоїти цим параметрам будь-які значення з діапазону 0–1, включно з **nearNormDepth** > **farNormDepth**.

Є також умова перевірки, яку використовуватимуть процедури буфера глибини. Її задають функцією

```
glDepthFunc(testCondition);
```

параметру можна присвоїти будь-яке з 8 значень: **GL_LESS** (значення за замовчуванням – значення глибини опрацьовують, якщо воно менше за поточне значення в буфері глибини для цього пікселя), **GL_GREATER, GL_EQUAL, GL_NOTEQUAL, GL_LEQUAL, GL_GEQUAL, GL_NEVER** (точки не опрацьовують), **GL_ALWAYS** (опрацьовують усі точки). Ці перевірки можуть бути корисними в різних застосуваннях, де вони дають змогу скоротити обчислення в буфері глибини.

Крім того, можна задати статус буфера глибини – лише читання або читання-запис; для цього використовують функцію:

```
glDepthMask(writeStatus);
```

Якщо **writeStatus=GL_TRUE** (значення за замовчуванням), з буфера глибини можна зчитувати значення і записувати в буфер; якщо **write Status=GL_FALSE**, режим запису в буфер глибини неможливий, допускається тільки зчитування значень для порівняння у перевірках глибин. Ця можливість корисна, коли треба використати якийсь складний фон, відображаючи різні об'єкти переднього плану. Після запису фону в буфер глибини режим запису відключають і опрацьовують передній план. Це дає змогу згенерувати низку кадрів з різними об'єктами переднього плану чи одним об'єктом у різних позах для анімаційної послідовності, записуючи лише значення глибини фону.

Іншою сферою застосування цієї функції є відображення ефектів прозорості (підрозділ 9.6.5). Тоді треба записати лише глибини непрозорих об'єктів для перевірок видимості, але не глибини точок прозорої поверхні, тому під час опрацювання прозорої поверхні режим запису для буфера глибини відключають.

8.5.3. Функції OpenGL дослідження видимих каркасних поверхонь

Каркасне зображення стандартних графічних об'єктів отримують за допомогою генерування лише меж цих об'єктів. Для цього функцію режиму багатокутників (підрозділ 3.2.3) задають, наприклад, у вигляді:

```
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
```

Після виконання цієї команди будуть відображатися видимі і невидимі межі об'єктів.

Щоб забрати невидимі лінії з каркасного зображення, можна скористатися методом зсуву глибини, описаним у підрозділі 3.2.3. Отже, спочатку за допомогою кольору переднього плану задають каркасну версію об'єкта, потім зсувом по глибині задають схему заповнення внутрішньої частини і колір фону для внутрішнього заповнення. Зсув по глибині гарантує, що заповнення кольором фону не перешкоджатиме відображенню видимих меж. Наприклад, у наступному фрагменті коду згенеровано каркасне зображення з використанням білого кольору переднього плану і чорного кольору фону.

```
glEnable(GL_DEPTH_TEST);
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
glColor3f(1.0, 1.0, 1.0);
/* Викликаємо процедуру опису об'єкта */
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
glEnable(GL_POLYGON_OFFSET_FILL);
glPolygonOffset(1.0, 1.0);
glColor3f(0.0, 0.0, 0.0);
/* Знову викликаємо процедуру опису об'єкта */
glDisable(GL_POLYGON_OFFSET_FILL);
```

8.5.4. Функції OpenGL загасання з глибиною

Яскравість об'єкта можна змінювати як функцію його відстані від точки спостереження, використовуючи оператори

```
glEnable(GL_FOG);
glFogi(GL_FOG_MODE, GL_LINEAR);
```

Ці команди задають режим застосування лінійної функції глибини, записаної в рівнянні (8.1), до кольорів об'єкта, використовуючи $d_{\max} = 1.0, d_{\min} = 0.0$. Можна задати й інші значення для $d_{\max}, d_{\min}$, викликаючи функції:

```
glFogf(GL_FOG_START, minDepth);
glFogf(GL_FOG_END, maxDepth);
```

тут параметрам **minDepth** і **maxDepth** присвоюють значення з плаваючою крапкою, хоча, змінивши суфікс функції на **i**, можна буде використовувати цілі значення.

Крім того, за допомогою функції **glFog** можна задати колір атмосфери, який буде об'єднано з кольором об'єкта після застосування лінійної функції загасання з глибиною. Моделювання інших атмосферних ефектів буде обговорено в підрозділі 9.6.4.

8.6. Контрольні питання

1. Яку процедуру називають алгоритмом відтинання?
2. Як здійснюють двовимірне відтинання точки?
3. На яких арифметичних операціях ґрунтується алгоритм Коена–Сазерленда? Для чого він призначений?
4. Опишіть основні кроки алгоритму Коена–Сазерленда у випадку відтинання двовимірних відрізків.
5. Де відносно відтинального вікна розташована точка, якій присвоєно код 1101?
6. З якими багатокутниками (опуклими чи ввігнутими) працює більшість графічних систем? Чому?
7. Який алгоритм використовують для відтинання багатокутників у системах з конвеєрною архітектурою? Опишіть його суть.

8. Яким чином можна скоротити затрати під час обробки багатокутників загального вигляду, що мають достатньо багато вершин?
9. Опишіть модифікацію алгоритму Коена–Сазерленда для тривимірного випадку.
10. Як OpenGL дає змогу задавати на сцені додаткові площини відтинання?
11. Охарактеризуйте методи простору об'єктів і простору зображень у дослідженні видимих поверхонь.
12. Для чого призначений алгоритм дослідження задніх граней?
13. У чому полягає принцип усунення нелицьових граней?
14. Для чого призначений алгоритм Z-буфера? Скільки буферів потрібно для його роботи?
15. У якому випадку інформацію про колір треба заносити до буфера кадру?
16. Опишіть алгоритм буфера глибини.
17. Для чого призначений алгоритм A-буфера? Порівняйте його з алгоритмом Z-буфера.
18. Що поєднує в собі метод рядків розгортки?
19. Яка принципова відмінність між алгоритмами Z-буфера та рядків розгортки?
20. Для чого призначений метод сортування за глибиною?
21. Які методи визначення видимості для каркасних зображень ви знаєте?
22. За допомогою якої функції видаляють задні грані?
23. Які є функції дослідження видимих поверхонь за допомогою буфера глибини?
24. Опишіть функції OpenGL для дослідження видимих каркасних поверхонь.
25. Які функції OpenGL загасання з глибиною ви знаєте?

8.7. Приклади тестових питань

1. У разі реалізації алгоритму Коена–Сазерленда кінцевій точці відрізка, що розташована за дальньою площиною просторової зони видимості вище і зліва від відтинального вікна, буде присвоєно код:

   а) 100110;
   б) 011001;
   в) 101001.

2. Алгоритм Коена–Сазерленда призначений для:

   а) швидкого відтинання відрізків;
   б) побудови прямих ліній у растровій візуалізації;
   в) усунення прямих невидимих ліній у разі візуалізації.

3. Якщо застосувати операцію логічного перемноження до кодів кінців відрізка, то значення (0000) свідчить про те, що відрізок розташований

   а) поза областю відтинання;
   б) всередині області відтинання;
   в) на межі області відтинання.

4. Метод Сазерленда–Ходгмана призначений для:

   а) швидкого відтинання відрізків;
   б) відтинання багатокутників;
   в) усунення прямих невидимих ліній під час візуалізації.

5. Алгоритм Z-буфера призначений для:

   а) швидкого відтинання відрізків;
   б) побудови прямих ліній у разі растрової візуалізації;
   в) усунення невидимих поверхонь.

6. Якщо відстань до поточного багатокутника більша за значення, що зберігається в Z-буфері, то:

   а) інформацію про колір не треба заносити в буфер кадру;
   б) інформацію про колір треба заносити в буфер кадру;
   в) поточний багатокутник закриває опрацьовані раніше.

7. Якщо відстань до поточного багатокутника менша за значення, що зберігається в Z-буфері, то:

   а) інформацію про колір не треба заносити в буфер кадру;
   б) інформацію про колір треба заносити в буфер кадру;
   в) поточний багатокутник не закриває опрацьовані раніше.

8. Метод сортування за глибиною призначений для:

   а) швидкого відтинання відрізків;
   б) побудови прямих ліній у растровій візуалізації;
   в) усунення невидимих поверхонь.

9. Метод рядків розгортки призначений для:

   а) швидкого відтинання відрізків;
   б) побудови прямих ліній у растровій візуалізації;
   в) усунення невидимих поверхонь.

10. Алгоритм рядків розгортки під час обробки кожного пікселя повинен опрацьовувати такі багатокутники, що претендують на відображення в цій зоні екрана:

    а) всі;
    б) деякі;
    в) один.

Розділ 9

МОДЕЛІ ОСВІТЛЕННЯ І МЕТОДИ ВІЗУАЛІЗАЦІЇ ПОВЕРХОНЬ

Щоб отримати реалістичне зображення сцени, треба згенерувати перспективні проекції об'єктів і застосувати ефекти природного освітлення видимих поверхонь. Для обчислення кольору освітленої точки на поверхні об'єкта використовують *модель освітлення*. Розрахунок кольору за моделлю освітлення застосовують у *методі візуалізації поверхонь* для визначення кольору пікселів, у які переходять усі спроектовані точки сцени.

Щоб досягти в КГ фотореалістичності, потрібні дві речі: точне подання властивостей поверхні і добрий фізичний опис ефектів освітлення на сцені. До цих ефектів належать відбиття світла, прозорість, текстура поверхні та тіні.

Моделі освітлення в КГ часто є апроксимацією фізичних законів, що описують ефекти освітлення поверхні. Для скорочення обсягу обчислень у більшості пакетів використовують емпіричні базові моделі, які ґрунтуються на спрощених фотометричних розрахунках. У точніших моделях, таких як алгоритми дифузного відбиття, для обчислення інтенсивності світла розглядають поширення енергії випромінювання від джерела світла до різних поверхонь сцени.

9.1. Локальні моделі освітлення. Колір випромінювання

Спочатку зосередимо увагу на *локальних моделях* розподілу світла. Вони, на відміну від *глобальних моделей* освітлення, дають змогу обчислити відтінок окремої точки на деякій поверхні незалежно від інших поверхонь як цього, так і інших об'єктів сцени. У процесі обчислень враховують лише характеристики матеріалу, асоційованого з цією поверхнею, її локальну геометрію, розміщення і властивості джерел світла.

Світло може виходити від поверхні об'єкта у двох випадках: або об'єкт сам випромінює, або об'єкт відбиває світло, яке падає на нього зовні. У *спрощеній* моделі джерелом світла вважатимемо об'єкти, які *випромінюють*. Тоді кожна точка поверхні об'єкта (x, y, z) може випромінювати світловий промінь, який

характеризується напрямком емісії (θ, ϕ) і розподілом світлової енергії за довжинами хвиль λ. Отож елементарне джерело світла в загальному випадку характеризується *функцією випромінювання* $\mathbf{I}(x, y, z, \theta, \phi, \lambda)$. Напрямок випромінювання описують двома кутами, а фізичне джерело поліхромного випромінювання зображають як множину незалежних елементарних джерел монохроматичного світла.

Розглядаючи освітлену таким джерелом поверхню, можна отримати освітленість кожної точки, інтегруючи функцію випромінювання по поверхні джерела світла. Під час інтегрування треба враховувати, по-перше, тільки ті кути випромінювання, які забезпечують потрапляння променів на розглядану точку освітленої поверхні, а, по-друге, відстань між елементарним джерелом і цією точкою. Для спрощення джерело довільної форми часто моделюють багатокутниками, кожен із яких є *елементарним джерелом*, або апроксимують реальне джерело *множиною точкових*.

Прийнята колірна модель передбачає, що для отримання всієї гами кольорів достатньо враховувати три основні (первинні) складові: червоний, зелений і синій кольори. Тому надалі вважатиме, що будь-яке джерело складено з трьох незалежних джерел первинних кольорів, і відповідно описувати його трикомпонентною функцією випромінювання $\mathbf{I} = [I_r \; I_g \; I_b]^T$. Наприклад, для обчислення розподілу червоного світла в зображенні буде використано компоненту I_r функції випромінювання. Оскільки обчислення для всіх трьох кольорів виконують за єдиною схемою, то надалі розглядатимемо одне скалярне рівняння кожного типу і застосовуватимемо до будь-якого з первинних кольорів.

9.2. Джерела світла

Розглянемо чотири основні типи джерел світла: *фонове* освітлення, *точкові джерела, прожектори* і *віддалені* джерела світла. Цих чотирьох типів цілком достатньо для моделювання освітленості в більшості простих сцен.

9.2.1. Фонове освітлення

Джерело, яке забезпечує рівномірне освітлення по всьому простору сцени, називають *джерелом фонового світла.* Тому можна під час моделювання вважати, що кожну точку поверхні об'єктів цієї сцени освітлено однаково. Отже, функція освітленості кожної точки поверхні характеризується тільки заданою трикомпонентною функцією інтенсивності $\mathbf{I}_a = [I_{ar} \; I_{ag} \; I_{ab}]^T$. Надалі треба враховувати, що хоч кожна точка на поверхні будь-якого з об'єктів сцени отримує від джерела світло однакової інтенсивності, відбивають його вони по-різному.

9.2.2. Точкове джерело світла

Найпростішою моделлю об'єкта, який випромінює світлову енергію, є *точкове джерело*. Ідеальне *точкове джерело світла* випромінює світло однаково в усіх напрямках. Таке джерело, розміщене в точці $\mathbf{p}_0$, характеризується трикомпонентним вектором кольору

$$\mathbf{I}(\mathbf{p}_0) = [I_r(\mathbf{p}_0)\, I_g(\mathbf{p}_0)\, I_b(\mathbf{p}_0)]^T .$$

Освітленість деякої точки поверхні світлом від цього джерела обернено пропорційна квадрату відстані між цією точкою і джерелом:

$$\mathbf{I}_p(\mathbf{p}, \mathbf{p}_0) = \frac{1}{|\mathbf{p} - \mathbf{p}_0|^2} \mathbf{I}(\mathbf{p}_0) .$$

Точкові джерела доволі прості у використанні, однак неточно передають характеристики реальних фізичних освітлювальних приладів. Зображення сцени, сформовані з урахуванням лише точкових джерел, виходять дуже контрастними, всі об'єкти виявляються або дуже яскравими, або занадто темними. Реально кожен фізичний освітлювальний прилад має скінченні розміри, що призводить до плавнішого переходу від повністю світлих ділянок до повністю затінених. Ця зона переходу виявляється частково затіненою – перебуває в напівтіні. У графічній програмі можна імітувати зниження контрасту від точкового джерела, додавши джерело фонового світла – так зване джерело *світла заповнення*.

Врахування за наведеним законом відстані від конкретної точки до точкового джерела світла також впливає на контрастність. Хоча з фізичного погляду обернено пропорційна залежність між освітленістю і відстанню до джерела цілком коректна, в КГ нерідко використовують модифіковану залежність

$$f_{\text{рад.затух.}}(\mathbf{p}, \mathbf{p}_0) = \frac{1}{(a + bd + cd^2)}, \qquad (9.1)$$

де d – відстань між $\mathbf{p}$ і $\mathbf{p}_0$. Константи a, b і c вибирають за умови "пом'якшення" світлотіньового переходу. Якщо точкове джерело розташоване достатньо далеко від усіх об'єктів сцени, то можна знехтувати відмінностями у відстанях до різних точок сцени і вважати, що джерело освітлює їх цілком однаково.

9.2.3. Прожектори

Джерела світла типу *прожектор* відрізняються тим, що випускають світло лише в одному напрямку. Найпростіше змоделювати прожектор за допомогою точкового джерела світла, обмеживши для нього напрямок, у якому поширюються світлові промені. Для цього формують конус із вершиною в точці $\mathbf{p}_0$, вісь якого збігається з вектором напрямлення світла $\mathbf{L}$ (рис. 9.1), а кут нахилу твірної до основи дорівнює θ, за $\theta = 180^\circ$ отримуємо точкове джерело.

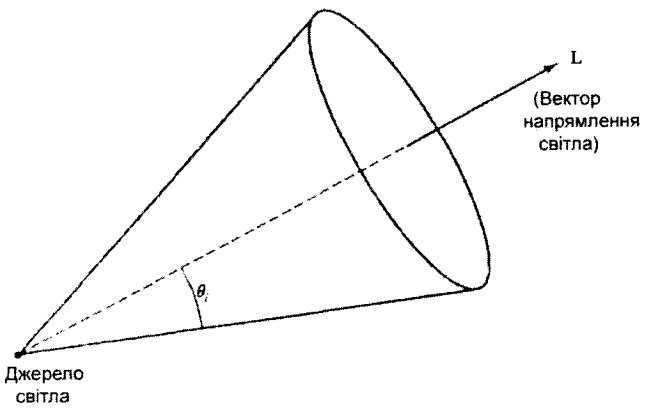


Рис. 9.1. Модель прожектора

Модель прожектора, ближча до реального фізичного приладу, характеризується функцією розподілу інтенсивності в конусі випромінювання. Природно, що найбільшу інтенсивність мають промені, спрямовані вздовж осі конуса. Інтенсивність випромінювання прожектора є функцією кута ϕ між віссю конуса і вектором **s**, спрямованим на певну точку освітленої поверхні, якщо цей кут менший за θ. Хоча функцію розподілу інтенсивності можна апроксимувати по-різному, найчастіше її задають у вигляді

$$f_{\text{angatten}}(\phi) = \cos^k \phi, \tag{9.2}$$

де показник k (параметр загасання) визначає, наскільки швидко спадає інтенсивність у разі збільшення кута ϕ. Якщо **s** і **L** є одиничними векторами (ортами), то значення косинуса можна обчислити за допомогою скалярного добутку цих векторів: $\cos\phi = \mathbf{s} \cdot \mathbf{L}$.

9.2.4. Віддалене джерело світла

Характерною особливістю віддаленого джерела є те, що всі промені, які воно випускає, можна вважати паралельними. Використання такого джерела у сцені суттєво підвищує швидкість формування зображення, бо усуває необхідність розраховувати напрямок променів. Прикладом такого джерела є Сонце.

На практиці замість положення джерела світла враховують напрямок його променів. Якщо використати математичний апарат однорідних координат, точкове джерело світла $\mathbf{p}_0$ можна зобразити 4-вимірним стовпцем $\mathbf{p}_0 = [x\ y\ z\ 1]^T$, а віддалене джерело світла – вектором напрямку у вигляді $\mathbf{p}_0 = [x\ y\ z\ 0]^T$.

9.3. Модель відбиття Фонга

Отож маємо доволі складне завдання: з одного боку, потрібна модель взаємодії світла і матеріалу, яка б відповідала реальним фізичним процесам, а з іншого боку, реалізація цієї моделі в графічній системі повинна бути підпо-

рядкована конвеєрній архітектурі і не надто перевантажувати комп'ютер. Розглянемо модель процесу відбиття світла об'єктами сцени, яку запропонував Фонг. Вона дає змогу формувати напівтонові зображення високої якості для різних умов освітлення і властивостей матеріалів.

9.3.1. Загальний опис моделі

Для обчислення кольору довільної точки **p** на поверхні об'єкта в моделі використовують 4 вектори. Вектор **N** є нормаллю до поверхні в точці **p**; вектор **V** спрямований від точки **p** до спостерігача чи центра проекції, а вектор **L** задає напрямок від точки **p** до довільної точки джерела розподіленого світла (у розглядуваному випадку – точкового джерела світла), вектор **R** – це напрямок ідеального відбиття променя, який падає на поверхню вздовж вектора **L**; він однозначно визначається векторами **N** і **L** (рис. 9.2).

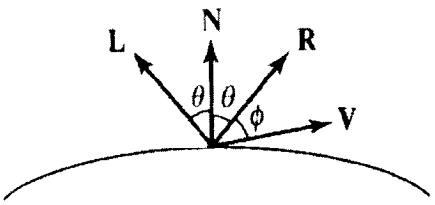


Рис. 9.2. Вектори у моделі Фонга

Модель Фонга (*Phong*) адекватно передає три типи взаємодії світла і матеріалу: *фонове освітлення* (розглянуте раніше з індексом *a*), *дзеркальне відбиття* (в математичних виразах ця компонента матиме індекс *s*) і *дифузне відбиття* (індекс *d*). Коротко опишемо останні два. У випадку **дзеркального відбиття** поверхні виглядають блискучими, бо більша частина світлової енергії відбивається або розсіюється у вузькому діапазоні кутів, близьких до кута відбиття. Дзеркало – це поверхня з ідеальним відбиттям. Хоча невелика частина енергії падаючого променя й поглинається, решта світла відбивається під одним кутом, причому цей кут дорівнює куту падіння променя. У **дифузному відбитті** падаюче світло розсіюється в різних напрямках. Такий тип взаємодії характерний для поверхні рівномірно пофарбованої стіни чи поверхні Землі, якою її бачить пілот літака чи космічного літального апарата.

Припустимо, що є множина *точкових джерел* світла. Модель джерел світла містить компоненти фонового, дифузного і дзеркального типів, для кожної точки **p** поверхні можна обчислити матриці освітленості розміром 3×3 для *i*-го джерела світла

$$\mathbf{L}_i = \begin{bmatrix} L_{ira} & L_{iga} & L_{iba} \\ L_{ird} & L_{igd} & L_{ibd} \\ L_{irs} & L_{igs} & L_{ibs} \end{bmatrix}.$$

Елементи першого рядка – це інтенсивності фонового світла відповідно для червоного, зеленого і синього кольорів. Елементи другого рядка – колірні компоненти інтенсивності для дифузного світла, а третього – для дзеркального. Ефектів ослаблення інтенсивності освітлення, пов'язаних з відстанню до джерела, не враховано.

Модель формують у припущенні, що є методи обчислення інтенсивності світла, відбитого будь-якою точкою поверхні, на основі значень елементів матриці $\mathbf{L}_i$ для цієї точки. При цьому треба знати ще й коефіцієнти відбиття, які залежать від властивостей матеріалу поверхні, її орієнтації, напрямку на джерело світла та відстані між ним і освітленою точкою. Для кожної точки можна обчислити свій набір коефіцієнтів і об'єднати їх у матрицю

$$\mathbf{R}_i = \begin{bmatrix} R_{ira} & R_{iga} & R_{iba} \\ R_{ird} & R_{igd} & R_{ibd} \\ R_{irs} & R_{igs} & R_{ibs} \end{bmatrix}.$$

Для кожної колірної складової світла, яка приходить від певного джерела, можна обчислити сумарну інтенсивність відбитого світла. Зокрема, для i-го джерела інтенсивність складової червоного світла, відбитої від поверхні в точці **p**:

$$I_{ir} = R_{ira}L_{ira} + R_{ird}L_{ird} + R_{irs}L_{irs} = I_{ira} + I_{ird} + I_{irs}.$$

Повну інтенсивність відбитого світла легко знайти, просумувавши складові від усіх джерел, які присутні у сцені, та *глобального* фонового освітлення:

$$I_r = \sum_i (I_{ira} + I_{ird} + I_{irs}) + I_{ar},$$

де I_{ar} – червона складова глобального фонового освітлення.

9.3.2. Відбиття фонового світла

Інтенсивність L_a фонового світла, яке падає на поверхню, однакова у всіх точках цієї поверхні (тут і надалі пропущено індекс кольору, оскільки міркування будуть справедливими для кожного з трьох основних кольорів). Частково енергія цього світла поглинається матеріалом поверхні, а частково відбивається з коефіцієнтом $0 \le k_a \le 1$, тобто в підсумку

$$I_a = k_a L_a, \tag{9.3}$$

де L_a може бути різним для кожного з джерел або єдиним для всіх джерел у сцені.

9.3.3. Дифузне відбиття

Поверхня з ідеальним дифузним відбиттям відбиває світло, що падає на неї, однаково в усіх напрямках, тому така поверхня виглядає однаковою для всіх спостерігачів. Однак кількість відбитого світла залежить від матеріалу поверхні, бо частина світлового потоку поглинається. Здатність поверхні до дифузного відбиття

характеризується її шорсткістю (негладкістю). Поверхні з ідеальним дифузним відбиттям інколи називають *ламбертовими*, а математично характер відбиття описує *закон Ламберта.*

Розглянемо плоску шорстку поверхню, освітлену сонячними променями. Найяскравішою вона буде опівдні, а ближче до вечора здаватиметься все темнішою, бо, відповідно до закону Ламберта, для спостерігача яскравість поверхні визначається тільки вертикальною складовою відбитого від неї світла (рис. 9.3). Що нижче до горизонту нахилятиметься джерело світла, та сама кількість світлової енергії розподілятиметься на все більшу площу, і поверхня сприйматиметься як темніша.

Закон Ламберта твердить, що R_d пропорційне до косинуса кута між нормаллю **N** до поверхні в точці спостереження і напрямком на джерело світла **L**. Якщо ці вектори є одиничними векторами, то $\cos\theta = \mathbf{N}\cdot\mathbf{L}$. Врахувавши коефіцієнт пропорційності k_d, який визначає частину відбитого світла, та ослаблення світлового потоку в разі віддалення джерела від точки спостереження, одержимо такий вираз для складової дифузного відбиття в загальному відбитому світловому потоці:

$$I_d = \frac{k_d}{a + bd + cd^2}(\mathbf{N}\cdot\mathbf{L})L_d\,. \tag{9.4}$$

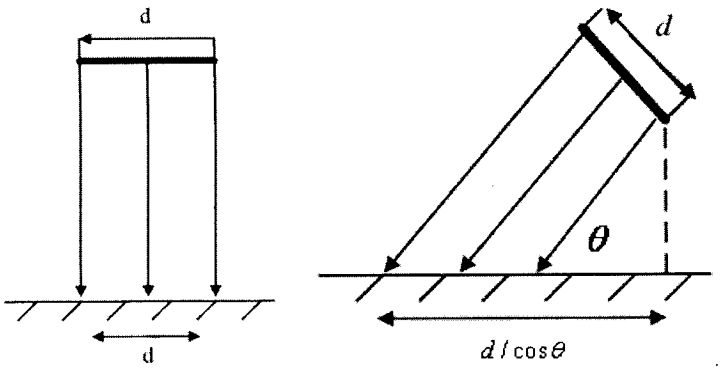


Рис. 9.3. Графічна інтерпретація закону Ламберта

9.3.4. Дзеркальне відбиття

Включивши в модель тільки фонове освітлення і дифузне відбиття, можна одержати зображення, на якому будуть тіні, тобто буде передано тривимірну форму об'єктів, але всі поверхні виглядатимуть матовими, без яскравих плям. Тоді як дифузне відбиття властиве шорстким поверхням, дзеркальне відбиття з'являється на гладких поверхнях. Точне моделювання явища дзеркального відбиття – завдання непросте, бо розсіювання світла є несиметричним і залежить від довжини хвилі падаючого світла, а також від зміни променя відбиття.

Фонг запропонував наближену модель, у якій врахування дзеркального відбиття дуже незначно збільшує обсяг обчислень порівняно з моделлю, яка враховує тільки дифузне відбиття. Поверхню зображають шорсткою для складових дифузного відбиття і гладкою для складових дзеркального відбиття. Інтенсивність світлового потоку, який досягає спостерігача, залежить від кута ϕ між вектором **R**, що характеризує напрямок ідеального дзеркального відбиття, і вектором **V**, спрямованим до спостерігача. У моделі Фонга використано рівняння

$$I_s = k_s L_s \cos^{\alpha} \phi , \tag{9.5}$$

де коефіцієнт k_s $(0 \le k_s \le 1)$ визначає ту частину світлового потоку, яка відбивається, показник степеня α – це *коефіцієнт різкості відблисків (дзеркальних "зайчиків")*. Зі збільшенням значення α відбите світло концентрується в зоні, близькій до кута ідеального дзеркального відбиття, в разі зростання α до нескінченності отримаємо ідеальне дзеркало. Значення в діапазоні від 100 до 500 відповідають характеру відбиття від більшості металевих поверхонь, від 1 до 100 – найпоширенішим матеріалам із загальними оптичними властивостями, менше ніж 1 – матовим поверхням.

Врахувавши коефіцієнт пропорційності k_s, який визначає частину відбитого світла, та ослаблення світлового потоку в разі віддалення джерела від точки спостереження, одержимо такий вираз для складової дзеркального відбиття в загальному відбитому світловому потоці:

$$I_s = \frac{k_s L_s (\mathbf{R} \cdot \mathbf{V})^{\alpha}}{a + bd + cd^2} . \tag{9.6}$$

Маючи дані про нормалізовані вектори **R** і **N**, отримаємо для моделі Фонга

$$I = \frac{1}{a + bd + cd^2} \left(k_d L_d \mathbf{N} \cdot \mathbf{L} + k_s L_s (\mathbf{R} \cdot \mathbf{V})^{\alpha} \right) + k_a L_a . \tag{9.7}$$

Обчислення за цією формулою виконують окремо для кожного джерела світла й окремо для всіх трьох кольорів.

9.4. Обчислення векторів

Описана модель відбиття світла об'єктами сцени ніяк не пов'язана з виглядом проекції (паралельної чи перспективної) і є доволі загальною в тому сенсі, що може бути застосована і до плоских, і до криволінійних поверхонь для довільних відстаней між поверхнею і спостерігачем. Розрахунок інтенсивності за моделлю освітленості можна різними способами застосувати до візуалізації поверхонь. У графічних пакетах поверхні переважно візуалізують за допомогою алгоритмів рядків розгортки – вони скорочують час обробки, оскільки використовують дискретизацію поверхні багатокутниками, обчислюють інтенсивність лише в їхніх вершинах, а потім інтерполюють її на інші точки внутрішніх областей багатокутника.

Більша частина обчислень у процесі тонування зображення просторової сцени припадає на визначення компонент векторів, які використовують у моделі Фонга, та їхніх скалярних добутків.

9.4.1. Визначення нормалі до поверхні

Для гладкої поверхні вектор нормалі є в кожній точці і визначає локальну орієнтацію ділянки поверхні в околі цієї точки. Метод обчислення компонент вектора нормалі залежить від прийнятого способу математичного опису поверхні.

Розглянемо площину, описану рівнянням $Ax + By + Cz + D = 0$. Рівняння площини можна записати й через нормаль $\mathbf{N}$ у точці $\mathbf{p}_0$, яка належить цій площині, як $\mathbf{N} \cdot (\mathbf{p} - \mathbf{p}_0) = 0$, де $\mathbf{p}$ (x, y, z) – будь-яка точка на цій самій площині. Порівнюючи ці два рівняння, одержимо вираз для вектора нормалі: $\mathbf{N} = [A\ B\ C]^T$ або в однорідних координатах $\mathbf{N} = [A\ B\ C0]^T$.

Площину можна також однозначно задати сукупністю трьох точок $\mathbf{p}_0$, $\mathbf{p}_1$, $\mathbf{p}_2$, які не лежать на одній прямій. Вектори $\mathbf{p}_2 - \mathbf{p}_0$ і $\mathbf{p}_1 - \mathbf{p}_0$ паралельні до площини, і для визначення нормалі можна використати їхній векторний добуток $\mathbf{N} = (\mathbf{p}_2 - \mathbf{p}_0) \times (\mathbf{p}_1 - \mathbf{p}_0)$.

9.4.2. Відшукання кута відбиття

Визначивши нормаль до поверхні в точці спостереження і знаючи положення джерела світла, можна обчислити й напрямок ідеально відбитого променя. В ідеальному дзеркалі виконується закон: кут падіння дорівнює куту відбиття. Кут падіння вимірюють між нормаллю і напрямком на джерело світла, а кут відбиття – між нормаллю і відбитим променем.

У двовимірному просторі напрямок відбитого променя задано цим законом однозначно, а в тривимірному просторі для обчислення напрямку відповідного вектора треба ввести додаткову умову: в точці $\mathbf{p}$ поверхні падаючий і відбитий промені, а також нормаль до поверхні повинні лежати в одній площині, тобто бути *компланарними* векторами. Разом ці дві умови дають змогу однозначно визначити напрямок відбитого променя $\mathbf{R}$ за заданим вектором нормалі та ортом падаючого променя $\mathbf{L}$.

Оскільки тут потрібен тільки напрямок відбитого променя $\mathbf{R}$, то надалі припускаємо, що всі розглянуті вектори є ортами, тобто $|\mathbf{L}| = |\mathbf{N}| = |\mathbf{R}| = 1$. Використовуючи скалярний добуток, одержимо співвідношення, яке пов'язує кути падіння і відбиття:

$$\cos\theta_i = \mathbf{L} \cdot \mathbf{N} = \cos\theta_R = \mathbf{N} \cdot \mathbf{R}, \text{ бо } \theta_i = \theta_R.$$

Умова компланарності трьох векторів з погляду математики означає, що $\mathbf{R}$ можна виразити лінійною комбінацією $\mathbf{L}$ і $\mathbf{N}$: $\mathbf{R} = \alpha\,\mathbf{L} + \beta\,\mathbf{N}$. Помноживши скалярно на $\mathbf{N}$ останню рівність, одержимо:

$$\mathbf{R} \cdot \mathbf{N} = \alpha\,\mathbf{L} \cdot \mathbf{N} + \beta = \mathbf{L} \cdot \mathbf{N}.$$

Друге співвідношення, яке пов'язує α і β, можна знайти з умови, що вектор **R** повинен бути ортом:

$$1 = \mathbf{R} \cdot \mathbf{R} = \alpha^2 + 2\alpha\beta \mathbf{L} \cdot \mathbf{N} + \beta^2 .$$

Розв'язавши систему рівнянь, одержимо, що $\mathbf{R} = (\mathbf{L} \cdot 2\mathbf{N})\,\mathbf{N} - \mathbf{L}$.

9.4.3. Обчислення вектора половинного напряму

Під час відтворення ефекту дзеркального відбиття на основі моделі Фонга скалярні добутки $\mathbf{R} \cdot \mathbf{V}$ треба знаходити для кожної точки поверхонь об'єктів сцени. Можна трохи спростити цей процес, розглянувши проміжний вектор одиничної довжини, спрямований між векторами **V** і **L**, $\mathbf{H} = \dfrac{\mathbf{L} + \mathbf{V}}{|\mathbf{L} + \mathbf{V}|}$ (рис. 9.4).

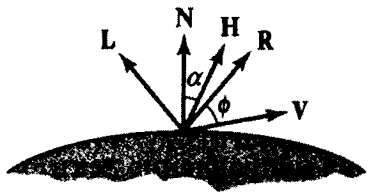


Рис. 9.4. Вектор половинного напрямку, який йде по бісектрисі кута між L і V

Кут ψ між векторами **N** і **H** називають *кутом половинного напряму*. Якщо вектор **V** лежить у тій самій площині, що й **L**, **N** і **R**, то маємо $2\psi = \phi$. Замінивши обчислення скалярного добутку $\mathbf{R} \cdot \mathbf{V}$ на $\mathbf{N} \cdot \mathbf{H}$, можна уникнути обчислення вектора **R**, при цьому замість $(\mathbf{R} \cdot \mathbf{V})^{\alpha}$ у формулі (9.7) треба використовувати доволі близьке значення $(\mathbf{N} \cdot \mathbf{H})^{\alpha}$.

Для неплоских поверхонь $\mathbf{N} \cdot \mathbf{H}$ потребує менше обчислень, ніж $\mathbf{R} \cdot \mathbf{V}$, бо в розрахунках **R** у кожній точці поверхні фігурує змінний вектор **N**. Крім того, якщо і спостерігач, і джерело світла розташовані достатньо далеко від поверхні, вектори **V** і **L** є константами, тому **H** теж константа для всіх точок поверхні. Якщо кут між **H** і **N** більший за 90°, $\mathbf{N} \cdot \mathbf{H}$ є від'ємним, і внесок дзеркального відбиття дорівнює 0.

Вектор **H** подає орієнтацію поверхні, яка дає максимальне дзеркальне відбиття в напрямку спостереження для певного розташування точкового джерела світла. З цієї причини його інколи називають *напрямком орієнтації поверхні з максимальною підсвіткою*. Крім того, якщо вектор **V** компланарний векторам **L** і **R** (а тому і **N**), кут дорівнює $\psi = \phi/2$. Якщо **V**, **L** і **N** некомпланарні, $\psi > \phi/2$ і залежить від просторового зв'язку цих трьох векторів.

9.5. Візуалізація (зафарбовування) багатокутників

У більшості графічних систем, зокрема й в OpenGL, використовують полігональну модель для зображення об'єктів сцени, яка передбачає апроксимацію криволінійних поверхонь багатьма маленькими плоскими багатокутниками.

9.5.1. Плоске зафарбування

Під час переміщення від однієї точки на поверхні до іншої в загальному випадку можуть змінюватись три вектори: **L**, **N** і **V**. Однак, якщо поверхня плоска, вектор **N** залишається постійним для всіх точок цієї поверхні. Якщо спостерігач є достатньо далеко від цієї поверхні, то зміною вектора **V** після переходу від однієї точки до іншої на поверхні плоского багатокутника невеликого розміру теж можна знехтувати і вважати його постійним. І, нарешті, якщо у сцені використано віддалене джерело світла, то вектор **L** також можна вважати постійним для всіх точок поверхні цього багатокутника. При цьому термін "віддалене" розглядають у відносному сенсі, порівнюючи розміри багатокутника з відстанню до спостерігача чи до джерела. Для реалізації алгоритму зафарбовування треба замість розташування джерела задавати напрямок до нього.

Якщо три зазначені вектори постійні для всіх точок багатокутника, то всі необхідні обчислення для його зафарбування можна виконати лише раз і застосувати результати до всіх точок багатокутника. Цей метод одержав назву *плоского* або *рівномірного зафарбування*. У OpenGL режим плоского зафарбування задають аргументом **GL_FLAT** у виклику функції **glShadeModel**:

```
glShadeModel(GL_FLAT);
```

У режимі плоского зафарбування OpenGL використовує вектор нормалі, асоційований з першою вершиною кожного чергового багатокутника, тоді на зображенні чітко видно різницю у відтінках кольору окремих багатокутників сітки, причому якщо джерело світла чи спостерігач розташовані достатньо близько до об'єктів сцени, то цю відмінність породжує не тільки різна орієнтація багатокутників (значення векторів нормалей), але й різні напрямки векторів **L** і **V**. В області світлотіньового переходу око сприймає *смуги Маха*, тобто контраст видається більш різким, ніж насправді.

9.5.2. Інтерполяційне зафарбування і зафарбування методом Гуро

OpenGL має засоби інтерполяції кольору, асоційованого з окремими вершинами. Якщо встановити режим згладжування для зафарбовування, передавши аргумент **GL_SMOOTH** функції **glShadeModel**, то OpenGL інтерполюватиме колір уздовж графічного примітива, наприклад, відрізка. Припустимо, що в програмі встановлено режими згладжування зафарбування та врахування освітлення і що з кожною вершиною асоційовано вектор нормалі відповідного багатокутника. Під час обчислення освітлення кожної точки визначають її колір, який залежить від

властивостей матеріалу і векторів **L** і **V**, обчислених раніше для цієї вершини. У разі використання віддалених джерел світла і відсутності дзеркальної складової алгоритм інтерполяційного зафарбування сформує однаковий колір для всієї внутрішньої області багатокутника.

Повертаючись до полігональної сітки, зазначимо, що ідея використання в обчисленнях нормалей, асоційованих з вершинами такої сітки, з математичного погляду абсолютно некоректна. Оскільки вершина є точкою перетину, як мінімум, двох по-різному орієнтованих багатокутників, то в ній відбувається розрив неперервності функції вектора нормалі.

Метод *зафарбування Гуро (Gouraud)* полягає в тому, що з вершинами пов'язують нормалі, отримані в результаті усереднення нормалей багатокутників, що перетинаються в цій вершині (рис. 9.5).

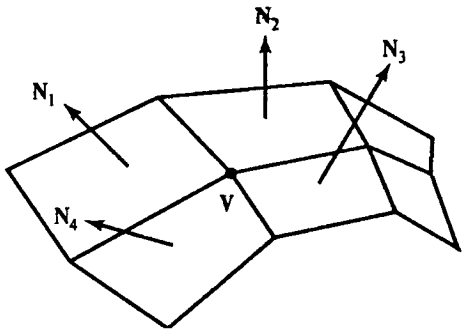


Рис. 9.5. Визначення нормалі у вершині полігональної сітки

Алгоритм методу зафарбування Гуро

1. Визначити середній одиничний вектор нормалі в кожній вершині багатокутника.
2. Застосувати модель освітленості в кожній вершині багатокутника, щоб отримати інтенсивність у цій точці.
3. Лінійно інтерполювати інтенсивності вершин на внутрішню область багатокутника (рис. 9.6, *а*). Інтенсивність у точках 4 і 5 є лінійною інтерполяцією інтенсивностей вершин 1 і 2 та 2 і 3 відповідно. Внутрішня точка **p** отримує значення інтенсивності, яке є лінійною інтерполяцією інтенсивностей у точках 4 і 5.
4. Описаний метод інтерполяції інтенсивностей усуває розриви, наявні у плоскій візуалізації, але має і свої недоліки. Відблиски на поверхні інколи відображаються з аномальними формами, а лінійна апроксимація інтенсивності може давати на поверхні яскраві або темні прожилки (смуги Маха). Ці ефекти можна зменшити, поділивши поверхню на більшу кількість багатокутних граней або використавши точніший розрахунок інтенсивності.

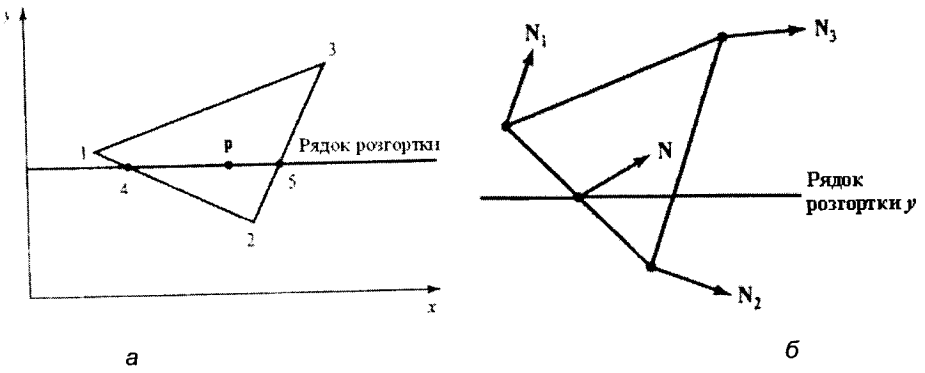


Рис. 9.6. Модель лінійної інтерполяції інтенсивності вершин на внутрішню область багатокутника (а) та нормалей поверхні вздовж сторони багатокутника (б)

Зазначимо, що в літературі часто не розрізняють метод Гуро та інтерполяційний метод зафарбовування.

9.5.3. Зафарбування методом Фонга

Навіть у разі використання методу Гуро часто не вдається уникнути появи на зображенні смуг Маха. Фонг запропонував інтерполювати не *колір точок* від вершини до вершини, а *напрямок нормалей* послідовних точок на ребрах кожного багатогранника.

Алгоритм методу Фонга

Метод Фонга опрацьовує кожну мозаїчну область мозаїчної криволінійної поверхні з використанням таких процедур.

1. Визначити середній одиничний вектор нормалі в кожній вершині багатокутника.
2. Використовуючи нормалі вершин, за допомогою лінійної інтерполяції знайти нормалі у всіх точках внутрішньої області багатокутника.
3. Застосувати модель освітленості до точок уздовж рядків розгортки і розрахувати інтенсивності пікселів з використанням інтерпольованих векторів нормалі.

Процедури інтерполяції для векторів нормалі в методі Фонга ідентичні процедурам пошуку значень інтенсивності в методі Гуро. Вектор нормалі **N** на рис. 9.6, *б* знайдемо за допомогою вертикальної інтерполяції векторів у вершинах 1 і 2:

$$\mathbf{N} = \frac{y - y_2}{y_1 - y_2}\mathbf{N}_1 + \frac{y_1 - y}{y_1 - y_2}\mathbf{N}_2.$$

Щоб отримати вектори нормалі в послідовних рядках розгортки і пікселі вздовж рядків розгортки, використовують ті самі методи послідовного розрахунку.

Різниця між двома підходами до візуалізації поверхні полягає в тому, що тепер, щоб знайти інтенсивності точок поверхні, модель освітленості треба застосовувати в кожному пікселі вздовж рядків розгортки.

Метод Фонга дає змогу одержати більш гладке тонування зображення, але при цьому об'єм обчислень різко зростає. Розроблено багато варіантів апаратної реалізації методу Гуро, які дають змогу отримувати зображення достатньої якості, практично не збільшуючи часу зафарбування, чого не скажеш про метод Фонга. Тому тепер метод Фонга використовують тільки в тих системах, де не потрібно формувати зображення в реальному масштабі часу.

9.6. Налаштування параметрів освітлення в OpenGL

В OpenGL реалізовано багато процедур встановлення точкових джерел світла, підбору коефіцієнтів відбиття поверхні і вибору значень інших параметрів у стандартній моделі освітлення. Крім того, можна моделювати прозорість, а також відображати об'єкти, використовуючи плоске зафарбовування або зафарбовування за методом Гуро. Однак в OpenGL немає процедур для зафарбовування поверхонь за методом Фонга, моделлю трасування променів чи методом дифузного відбиття.

Система OpenGL підтримує джерела світла чотирьох розглянутих раніше типів, причому в одній програмі можна використовувати до 8 джерел світла. Кожне джерело має свій набір параметрів, зокрема й програмний прапорець увімкнення-вимкнення. Параметри, що описують джерело світла, відповідають моделі Фонга.

9.6.1. Опис точкових джерел світла

Для задання властивостей точкових джерел світла використовують функцію

```
glLight*(lightName, lightProperty, propertyValue);
```

код суфікса – **i**, **f**, для векторних параметрів додають **v**; аргументу **lightName** присвоюють значення однієї з символьних констант **GL_LIGHT0**, ..., **GL_LIGHT7**, які відповідають можливим джерелам світла; параметр **propertyValue** є вказівником на масив значень властивостей зазначеного джерела.

Параметру **lightProperty** можна присвоювати значення однієї з 10 символьних констант, які визначають тип встановлюваних властивостей:

- **GL_POSITION** – для одночасного задання *розташування і типу* 4-елементним вектором **propertyValue**, де перші три елементи – це глобальні координати точки-джерела (числа з плаваючою крапкою), четвертий – 0.0 (*віддалене* джерело, світло напрямлене) або 1.0 (*локальне* джерело); значення за замовчуванням – (0.0, 0.0, 1.0, 0.0), що вказує на віддалене джерело, промені світла від якого поширюються вздовж від'ємного напрямку осі *z*;

три константи властивостей кольору RGBA:

- **GL_AMBIENT** – фонове освітлення;
- **GL_DIFFUSE** – дифузне освітлення;
- **GL_SPECULAR** – ефекти дзеркального відбиття;

щоб визначити кольори, задають 4-елементний масив **propertyValue** значень із плаваючою крапкою; компоненти кожного кольору подають у порядку RGBA, альфа-компоненту використовують лише в активних процедурах змішування кольорів; значення за замовчуванням: для віддаленого джерела – чорне фонове освітлення, біле дифузне і дзеркальне відбиття, для решти джерел – 3 чорні кольори;

три константи для радіального загасання інтенсивності:

- **GL_CONSTANT_ATTENUATION**,
- **GL_LINEAR_ATTENUATION**,
- **GL_QUADRATIC_ATTENUATION**,

які відповідають коефіцієнтам загасання a, b і c у формулі (9.1); можна використовувати додатні цілі числа або величини з плаваючою крапкою; значення за замовчуванням: $a = 1.0$, $b = 0.0$, $c = 0.0$, тобто радіального загасання немає;

три константи для задання напрямлених ефектів (прожекторів):

- **GL_SPOT_DIRECTION** – напрямок світла задають тривимірним вектором у глобальних координатах як цілі числа чи значення з плаваючою крапкою; за замовчуванням – (0.0, 0.0, –1.0), тобто світло поширюється паралельно до від'ємного напрямку осі z;
- **GL_SPOT_CUTOFF** – кут θ з (9.2) задають у градусах як ціле число чи величину з плаваючою крапкою, може бути будь-яким значенням від 0° до 90° або рівним 180° (за замовчуванням, тоді джерело випромінює в усіх напрямках);
- **GL_SPOT_EXPONENT** – параметр загасання k з (9.2) задають цілим числом або значенням із плаваючою крапкою з діапазону від 0 (за замовчуванням) до 128.

Отже, за замовчуванням використовують точкове джерело світла, яке випромінює у всіх напрямках без кутового загасання. Значення фонової, дифузної та дзеркальної складових кольору світла при куті ϕ обчислюють множенням компонентів інтенсивності на параметр кутового загасання k. Якщо $\phi > \theta$, то об'єкт розташований поза конусом і цим джерелом не освітлюється. Для променів світла, розміщених всередині конуса, також можна ввести радіальне загасання інтенсивності.

Режим освітлення сцени вмикають командою

```
glEnable(GL_LIGHTING);
```

а конкретне джерело світла після задання всіх його властивостей – командою

```
glEnable(lightName);
```

потім поверхні об'єктів візуалізують з використанням обчисленого освітлення, в якому враховують внески від усіх ввімкнених джерел світла.

Розміщення джерела світла входять до опису сцени і разом з розміщеннями об'єктів перетворюють на координати спостереження за допомогою матриць геометричного перетворення і перетворення точки спостереження. Отже, якщо треба зафіксувати джерело світла відносно об'єктів на сцені, його розміщення задають після зазначення в програмі цих двох перетворень. Якщо ж треба, щоб джерело світла рухалось із зміною точки спостереження, його розміщення задають перед специфікацією перетворення точки спостереження. Окрім того, якщо потрібно, щоб джерело світла рухалось навколо стаціонарної сцени, до його розміщення можна застосувати перетворення зсуву чи обертання.

Після задання значень коефіцієнтів до всіх трьох кольорів (фонове освітлення, дифузне і дзеркальне відбиття) застосовують функцію радіального загасання. Хоч радіальне загасання може давати реалістичніші зображення, відповідні розрахунки є більш об'ємними.

9.6.2. Параметри глобального освітлення

Окрім фонового кольору, породженого окремими джерелами світла, можна як глобальну величину задати незалежне значення фонового освітлення. Деякі параметри освітлення можна задати на глобальному рівні функцією

```
glLightModel*(paramName, paramValue);
```

код суфікса – **i** або **f**; для векторних параметрів додають **v**.

Параметру **paramName** присвоюють значення символьної константи OpenGL, яка ідентифікує встановлений глобальний режим (наприклад, **GL_LIGHT_MODEL_AMBIENT** – фонове освітлення), а параметру **paramValue** присвоюють відповідне до **paramName** одне значення або набір значень (за замовчуванням для фонового освітлення задано (0.2, 0.2, 0.2, 1.0) – слабко інтенсивний темно-сірий колір).

Використовуючи цю функцію, можна задати глобальний рівень фонового освітлення, зазначити, як треба обчислювати відблиски (дзеркальні "зайчики") або застосувати модель освітлення до невидимих граней багатокутної поверхні.

Розрахунок дзеркального відображення потребує значень декількох векторів, включаючи вектор **V** від точки поверхні до точки спостереження. Щоб прискорити такий розрахунок, у процедурах OpenGL використовують постійний напрямок цього вектора; за замовчуванням покладають, що він збігається з додатним напрямком осі *z*, тобто (0.0, 0.0, 1.0). Для використання реальної точки спостереження (початку системи координат) у дзеркальному відбитті й обчислення вектора **V** записують команду

```
glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE);
```

якщо другий параметр має значення **GL_FALSE** (0 чи 0.0), розрахунок вектора V не виконують.

Хоч розрахунок дзеркального відбиття потребує більше часу у разі використання реальної точки спостереження, у підсумку отримуємо реалістичніші зображення.

Якщо в розрахунках освітлення наявні поверхневі текстури, відблиски на поверхні можуть бути тьмяними, а текстурні зразки – спотвореними дзеркальними доданками. Тому текстурні зразки можуть бути застосовані лише до недзеркальних доданків, які роблять внесок у колір поверхні, тобто ці доданки включають ефекти фонового освітлення, випромінювання з поверхні і дифузного відбиття. Використовуючи цю можливість, процедури освітлення OpenGL генерують два кольори в кожному циклі розрахунку поверхні: внески дзеркального і недзеркального кольорів. Текстурні зразки накладають лише на недзеркальний колір, потім два кольори об'єднують. Щоб використати описану можливість введення двох кольорів, викликають команду

```
glLightModeli(GL_LIGHT_MODEL_COLOR_CONTROL,
    GL_SEPARATE_SPECULAR_COLOR);
```

Якщо текстурні зразки не використовують, розбивати світло на два доданки необов'язково, тоді розрахунок освітлення виконуватиметься ефективніше. За замовчуванням цю властивість задано зі значенням **GL_SINGLE_COLOR**, і колір дзеркального відбиття не відділяється від інших компонент кольору поверхні.

У деяких застосуваннях може виникнути потреба у відображенні задніх поверхонь об'єкта, наприклад, у розрізі твердого тіла, коли окрім передніх поверхонь треба показати деякі задні. За замовчуванням під час розрахунку освітлення накладені властивості матеріалу застосовують лише до передніх граней. Щоб застосувати розрахунок освітлення *до передніх і задніх граней*, враховуючи відповідні властивості матеріалів, використовують команду

```
glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);
```

Під час виконання цієї команди вектори нормалі до поверхні беруть оберненими, і розраховують освітленість з використанням властивостей матеріалів, приписаних заднім граням. Для відключення двостороннього розрахунку освітлення у функції **glLightModel** використовують значення за замовчуванням **GL_FALSE** (0 чи 0.0).

9.6.3. Специфікація матеріалів

Коефіцієнти відбиття та інші оптичні властивості поверхонь задають функцією

```
glMaterial*(surfFace, surfProperty, propertyValue);
```

код суфікса – **i** або **f**, для векторних параметрів додають **v**.

Параметру **surfFace** присвоюють значення однієї з символьних констант **GL_FRONT**, **GL_BACK**, **GL_FRONT_AND_BACK** залежно від граней, до яких потрібно застосувати параметри; аргумент **surfProperty** – символьна константа, яка визначає властивість поверхні, яку буде задано, наприклад, k_a (9.3), k_d (9.4) k_s (9.6) чи параметр дзеркального відбиття α (9.5); параметру **propertyValue** присвоюють відповідне значення. Усі властивості, окрім параметра дзеркального відбиття, задають як векторні значення. Щоб встановити всі властивості освітленості об'єкта, використовують послідовність функцій **glMaterial**, а вже потім задають команди для опису геометрії об'єкта.

Значення RGBA для кольору випромінювання з поверхні I_{surf} задають з використанням символьної константи властивості поверхні **GL_EMISSION** (за замовчуванням – чорний (0.0, 0.0, 0.0, 1.0); це випромінювання не освітлюватиме інші об'єкти на сцені, якщо не визначити поверхню як джерело світла). Використовуючи символьні константи **GL_AMBIENT** (фонове освітлення, значення за замовчуванням – (0.2, 0.2, 0.2, 1.0)), **GL_DIFFUSE** (дифузне освітлення, за замовчуванням – (0.8, 0.8, 0.8, 1.0)) і **GL_SPECULAR** (випромінювання, за замовчуванням – (1.0, 1.0, 1.0, 1.0)), встановлюють значення коефіцієнтів відбиття від поверхні. Щоб зображення виглядало реалістичним, коефіцієнтам фонового і дифузного освітлення треба присвоїти однакові векторні значення; це можна зробити, використовуючи символьну константу **GL_AMBIENT_AND_DIFFUSE**. Константа **GL_SHININESS** задає показник дзеркального відбиття (коефіцієнт різкості відблисків), значення з діапазону від 0 (за замовчуванням) до 128.

Компоненти коефіцієнтів відбиття також можна задати з використанням таблиці кольорів символьною константою **GL_COLOR_INDEXES**. Індекси таблиці кольорів задають триелементним масивом цілих чисел або значень з плаваючою крапкою, за замовчуванням – (0, 1, 1).

9.6.4. Атмосферні ефекти

Застосувавши для отримання кольорів поверхні модель освітлення OpenGL, можна задати колір атмосфери на сцені й об'єднати кольори поверхні з кольором атмосфери.

Ілюзію глибини простору в зображенні можна передати не тільки суто геометричним перетворенням (перспективним спотворенням розмірів), але й "квазіоптичними" засобами, зокрема, сформувавши напівпрозоре середовище між спостерігачем та об'єктами сцени, і так можна одержати ефект туману. Технічно це виконують під час формування растрового зображення об'єкта змішуванням кольору об'єкта з кольором, густина якого залежить від відстані між спостерігачем і об'єктом – *коефіцієнта туманності* або *функції атмосферного загасання* $f(z)$. Її використовують для імітації спостереження сцени через мутну чи задимлену атмосферу. OpenGL підтримує лінійну, експоненціальну і гауссову форми кривої оптичної прозорості середовища.

Різні атмосферні параметри задають, використовуючи таку послідовність функцій:

```
glEnable(GL_FOG);
glFog*(atmoParameter, paramValue);
```

Щоб зазначити тип даних, до функції **glFog** додають код суфікса **i** або **f**, для векторних даних додають **v**. Щоб задати колір атмосфери, параметру **atmoParameter** присвоюють значення **GL_FOG_COLOR**. За замовчуванням колір атмосфери чорний (0.0, 0.0, 0.0, 1.0). Щоб вибрати функцію атмосферного загасання, яку використовуватимуть для об'єднання кольорів об'єкта й атмосфери, застосовують символьну константу **GL_FOG_MODE**:

```
glFogi(GL_FOG_MODE, atmoAttenFunc);
```

Якщо параметру **atmoAttenFunc** присвоєно значення **GL_EXP** (воно є за замовчуванням), то функцію атмосферного загасання описують формулою $f(z) = \exp(-az)$, а за значення **GL_EXP2** – формулою $f(z) = \exp(-(az)^2)$, де a – значення густини атмосфери, яке для будь-якої експоненційної функції задають так:

```
glFog*(GL_FOG_DENSITY, atmoDensity);
```

Третя можливість у виборі атмосферного поглинання – використати лінійну функцію загасання з глибиною, тоді параметру **atmoAttenFunc** присвоюють значення **GL_LINEAR**.

Вибрану функцію атмосферного поглинання застосовують для обчислення кольору об'єкта, який визначається змішаним впливом кольорів поверхні та атмосфери. У процедурі створення атмосферних ефектів OpenGL для цього використовують рівняння:

$$I = f(z)I_{obj} + [1 - f(z)I_{atmo}].$$

9.6.5. Функції прозорості

Об'єкт називають *прозорим*, якщо видно предмети, що розташовані за ним. Якщо їх бачити не можна, об'єкт вважають *непрозорим*. Деякі матеріали є *півпрозорими*, коли світло, що пройшло крізь них, розсіюється в усіх напрямках; об'єкти, які спостерігають через такі матеріали, видаються розмитими (нечіткими), і часто їх не можна точно ідентифікувати. У загальному випадку прозора поверхня дає і відбите, і пропущене світло. Світло, яке пройшло крізь поверхню, є результатом випромінювання і відбиття від об'єктів і джерел, розташованих за прозорим об'єктом.

Деякі ефекти прозорості можна зімітувати, використовуючи процедури змішування кольорів, описані в підрозділі 3.1.2. Однак у загальному випадку прямо реалізувати прозорість у програмі OpenGL неможливо. Для простої сцени, яка містить декілька прозорих і непрозорих поверхонь, можна скомбінувати кольори

об'єктів, використовуючи *альфа-змішування* (alpha blending), задавши ступінь прозорості й опрацювавши поверхні за збільшенням глибини. Хоча в операціях змішування кольорів проігноровано ефекти заломлення, опрацювання прозорих поверхонь складних сцен з різними умовами освітлення чи анімацією може бути доволі тривалим. Крім того, OpenGL не пропонує прямих засобів імітації зовнішнього вигляду поверхні напівпрозорого об'єкта (такого, як зерниста поверхня пластмаси чи панель "морозного" скла), який дифузно розсіює світло, що проходить крізь напівпрозорий матеріал.

Тому для відображення напівпрозорих поверхонь чи ефектів освітлення, породжених заломленням, треба написати власну програму. Щоб зімітувати ефекти освітленості, характерні для напівпрозорих об'єктів, слід комбінувати текстуру поверхні і властивості матеріалу. Для врахування ефектів заломлення можна змістити пікселі для поверхонь за напівпрозорим об'єктом, використовуючи закон Снелла $\sin\theta_r = \frac{\eta_i}{\eta_r}\sin\theta_i$ та рівняння:

$$\mathbf{T} = (\frac{\eta_i}{\eta_r}\cos\theta_i - \cos\theta_r)\mathbf{N} - \frac{\eta_i}{\eta_r}\mathbf{L},$$

де

- **T, N, L** – одиничні вектори перенесення в напрямку заломлення, нормалі до поверхні та напрямку від точки поверхні до джерела світла відповідно;
- θ_i, θ_r – кути падіння і відбивання;
- η_i, η_r – показники заломлення середовищ, в яких розповсюджується падаюче і заломлене світло.

Цей механізм формування прозорих об'єктів дає змогу обійтись без трасування променів у явному вигляді.

Прикладна програма керує інтенсивністю альфа-каналу так само, як і інтенсивністю кожного з основних кольорів, тобто задає її значення для кожного пікселя. Але система OpenGL під час виконання програми трактує це значення інакше: для неї воно визначає коефіцієнт перерахунку значень основних кольорів перед їхнім записом у буфер кадру. У результаті в код засвічення пікселя роблять свій внесок декілька геометричних об'єктів, формуючи *змішане* або *комбіноване* зображення.

У системах комп'ютерної графіки використовують поняття *пікселя-джерела* і *пікселя-приймача*. Використання α-каналу є одним зі способів виконати накладання (змішування) кольорів на рівні окремих пікселів. Таке накладання моделює заміну двох напівпрозорих кольорових скелець одним, яке матиме колір і коефіцієнт поглинання, відмінні від відповідних характеристик кожного з вихідних компонентів.

Якщо зобразити піксель-джерело і піксель-приймач у вигляді масивів з чотирьох елементів $\mathbf{s}=[s_r\ s_g\ s_b\ s_a]$, $\mathbf{d}=[d_r\ d_g\ d_b\ d_a]$, то результатом змішування кольорів буде новий код засвічення

$$\mathbf{d}'=[b_r s_r + c_r d_r b_g s_g + c_g d_g b_b s_b + c_b d_b b_a s_a + c_a\ d_a].$$

Масиви констант **b** і **c** називають відповідно *коефіцієнтами змішування* джерела і приймача. Як і базові колірні складові RGB, значення параметра α не може перевищувати 1.0 і бути меншим за 0.0. Підбираючи значення компонента α і коефіцієнтів змішування, можна отримати багато цікавих ефектів під час побудови зображень.

Щоби зарахувати об'єкти на сцені до напівпрозорих, використовують параметр *альфа* в таких командах опису RGBA-кольору поверхні, як **glMaterial** і **glColor**. Для поверхні значення параметра альфа можна вважати рівним коефіцієнту прозорості k_t, який визначає кількість пропущеного фонового світла, за рівняння цього об'єкта

$$I = (1 - k_t)\, I_{\text{refl}} + k_t I_{\text{trans}}, \qquad (9.8)$$

де I_{refl}, I_{trans}, I – інтенсивності відбитого від поверхні, прониклого крізь прозору поверхню та сумарного світла поверхні відповідно; $(1\text{-}k_t)$ – коефіцієнт непрозорості, наприклад, якщо коефіцієнт прозорості має значення 0,3, то 30 % фонового світла об'єднується з 70 % відбитої освітленості поверхні.

Колір прозорої поверхні можна задати функцією

```
glColor4f(R, G, B, A);
```

У цьому випадку вважають, що значення параметра *альфа* A дорівнює k_t; нагадаємо, що повністю прозора поверхня має $A = 0.0$, а непрозора – $A = 1.0$.

Після присвоєння значень прозорості активізують функції змішування кольорів і опрацьовують поверхні, починаючи з найвіддаленіших до найближчих до точки спостереження об'єктів. Кожен колір поверхні буде об'єднано з усіма накладеними поверхнями, занесеними в буфер кадру, з використанням співвіднесених з ними альфа-факторів.

Режим змішування зображень в OpenGL вмикають за допомогою функції

```
glEnable(GL_BLEND);
```

Потім треба налаштувати значення коефіцієнтів змішування джерела і приймача:

```
glBlendFunc(source_factor, destination_factor);
```

Коефіцієнти змішування кольорів задають так, щоб усі компоненти кольору поточної поверхні (джерела) множились на $(1 - A) = (1 - k_t)$, а всі компоненти кольору відповідних позицій буфера кадру (приймач) множились на величину $A = k_t$, тобто

```
glBlendFunc(GL_ONE_MINUS_SRC_ALPHA, GL_SRC_ALPHA);
```

Аналогічне налаштування приймача виконують константи **GL_ONE_MINUS_DST_ALPHA** і **GL_DST_ALPHA**.

Потім два кольори змішують відповідно до рівняння (9.8), де альфа-фактор дорівнює k_t, а в буфері кадрів використовують кольори поверхні, розташованої за прозорим об'єктом, який опрацьовуємо. Наприклад, якщо $A = 0{,}3$, то для кожної точки поверхні новий колір у буфері кадрів є сумою 30 % поточного кольору буфера і 70 % відбитого кольору об'єкта. Якщо використовувати замість альфа-фактора показник непрозорості і вважати, що A дорівнює йому, то у функції **glBlendFunc** треба поміняти місцями значення двох аргументів.

У прикладній програмі спочатку задають необхідні налаштування, після чого можна використовувати формат подання кольору RGBA.

Часто альфа-канал застосовують для усунення похибок зображення, пов'язаних з дискретизацією растру. *Режим згладжування* для прямих і точок, наприклад, встановлюють такою послідовністю операторів:

```
glEnable(GL_POINT_SMOOTH);
glEnable(GL_LINE_SMOOTH);
glEnable(GL_SMOOTHING);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

Перевіряти видимість можна з використанням функцій буфера глибини, описаних у підрозділі 8.6.2. Під час опрацювання кожної видимої непрозорої поверхні записують кольори поверхні та її глибину. Однак під час обробки видимої прозорої поверхні треба записати лише її кольори, бо вона не затінює фонових поверхнень, тому в цьому випадку буферу глибини за допомогою функції **glDepthMask** присвоюють статус "лише читання".

Якщо об'єкти опрацьовують за зменшенням глибини, режим запису буфера глибини виимкають, а потім знову вмикають під час обробки кожної прозорої поверхні. Альтернативний спосіб – можна поділити об'єкти на два класи.

Приклад опрацювання непрозорих і прозорих поверхонь:

```
glEnable(GL_DEPTH_TEST);
/* Опрацьовуємо всі непрозорі поверхні */
glEnable(GL_BLEND);
glDepthMask(GL_FALSE);
glBlendFunc(GL_ONE_MINUS_SRC_ALPHA, GL_SRC_ALPHA);
/* Опрацьовуємо всі прозорі поверхні */
glDepthMask(GL_TRUE);
glDisable(GL_BLEND);
glutSwapBuffers;
```

Якщо прозорі об'єкти не опрацьовують у строгому порядку від фону до переднього плану, за цим підходом не завжди вдається точно їх розфарбувати. Однак для простих сцен він є простим і ефективним методом генерації приблизних зображень ефектів прозорості.

9.6.6. Функції візуалізації поверхні

Як уже було сказано, метод візуалізації вибирають за допомогою функції

```
glShadeModel(surfRenderingMethod);
```

параметру присвоюють значення **GL_FLAT** для плоского зафарбування, **GL_SMOOTH** – для методу Гуро (за замовчуванням).

Коли функцію **glShadeModel** застосовують до мозаїчної криволінійної поверхні, такої як сфера, апроксимована багатокутниками, процедури візуалізації OpenGL використовують для розрахунку кольору багатокутника вектори нормалі до поверхні у його вершинах. Декартові компоненти вектора нормалі до поверхні в OpenGL одержують командою:

```
glNormal3*(Nx, Ny, Nz);
```

код суфікса **b, s, i, f, d,** а також **v**. Значення типу **b, s, i** перетворюються на формат **f** з діапазону від –1.0 до 1.0. Ця функція задає компоненти вектора нормалі до поверхні як значення параметра стану, які застосовують до всіх наступних команд **glVertex**, за замовчуванням вектор нормалі йде в додатному напрямку осі *z*.

У випадку плоскої візуалізації поверхонь для кожного багатокутника потрібен лише один вектор нормалі до поверхні. Тому нормаль до всіх багатокутників можна, наприклад, одержати так:

```
glNormal3fv(normalVector);
glBegin(GL_TRIANGLES);
   glVertex3fv(vertex1);
   glVertex3fv(vertex2);
   glVertex3fv(vertex3);
glEnd();
```

Якщо до описаного тут трикутника потрібно застосувати процедуру візуалізації Гуро, то одержують вектор нормалі для кожної вершини.

Хоч вектори нормалі не обов'язково задавати як одиничні, але якщо їх все-таки пронормувати, то обсяг обчислень стане менше. Будь-яка неодинична нормаль до поверхні автоматично перетворюється на одиничну, якщо ввести до програми команду:

```
glEnable(GL_NORMALIZE);
```

Ця команда знову нормує вектори нормалі, якщо вони змінились після геометричних перетворень, таких як масштабування чи зсув.

Інша можливість – позначити список векторів нормалі, які будуть об'єднані чи співвіднесені з масивом вершин. Для створення масиву векторів нормалей використовують команди:

```
glEnableClientState(GL_NORMAL_ARRAY);
glNormalPointer(dataType, offset, normalArray);
```

параметру **dataType** присвоюють значення **GL_BYTE, GL_SHORT, GL_INT, GL_FLOAT** (за замовчуванням) або **GL_DOUBLE**. Кількість байтів між послідовними векторами нормалі в масиві **normalArray** задають параметром **offset** (за замовчуванням 0).

9.7. Глобальне тонування

Природно, що *локальна модель освітленості* має певні обмеження. Якщо в групі об'єктів один частково затінює інший, використання локальної моделі не дасть змоги сформувати тінь від одного об'єкта на інший, а також врахувати відблиски, спричинені дзеркальним відбиттям світла від одного об'єкта в напрямі іншого. Якщо потрібно створити зображення, в якому були б присутні всі описані оптичні ефекти, треба використати складніші, а тому й повільніші *моделі глобального розподілу світла*, з яких найпопулярнішими сьогодні є модель трасування променів та метод дифузного відбиття (аналізу випромінювання). Перша адекватно передає оптичні ефекти у сценах із дзеркальними поверхнями, наприклад, якщо значну частину об'єктів зроблено зі скла, полірованого металу та інших аналогічних матеріалів. Друга краще передає оптичні ефекти у сценах з матовим поверхнями, наприклад, в інтер'єрі приміщень.

9.8. Контрольні питання

1. Чим відрізняються локальні моделі розподілу світла від глобальних?
2. Від яких параметрів залежить функція випромінювання елементарного джерела світла?
3. Перерахуйте чотири основні типи джерел світла.
4. Чим характеризується джерело фонового світла?
5. Якими виходять зображення сцени, сформовані з урахуванням лише точкових джерел?
6. Як у графічній програмі можна імітувати зниження контрасту від точкового джерела?
7. Запишіть модифіковану залежність між освітленістю і відстанню до джерела, яку використовують у КГ для "пом'якшення" світлотіньового переходу.
8. Як можна змоделювати прожектор?
9. Як апроксимують функцію розподілу інтенсивності в конусі випромінювання?
10. Які переваги використання віддаленого джерела у сцені?
11. Які зображення дають змогу сформувати модель відбиття Фонга? Яким критеріям відповідає реалізація цієї моделі в графічній системі?
12. Які три типи взаємодії світла і матеріалу адекватно передає модель Фонга?
13. Запишіть матриці освітленості та коефіцієнтів відбиття матеріалу поверхні, які використовують у моделі Фонга.
14. Як обчислюють для i-го джерела інтенсивність складових червоного, зеленого і синього світла, відбитих від поверхні в деякій точці?

15. Як знаходять повну інтенсивність колірних складових відбитого світла?
16. Запишіть формулу для знаходження інтенсивності глобального фонового освітлення.
17. Сформулюйте закон Ламберта.
18. Запишіть вираз для складової дифузного відбиття в загальному відбитому світловому потоці для моделі Фонга з урахуванням відстані від точки спостереження до джерела світла.
19. Як виглядатимуть поверхні, коли до моделі Фонга введено лише фонове освітлення та дифузне відбиття?
20. Запишіть вираз для складової дзеркального відбиття в загальному відбитому світловому потоці для моделі Фонга з урахуванням відстані від точки спостереження до джерела світла.
21. Яким методом обчислюють компоненти вектора нормалі?
22. Як знайти напрям ідеально відбитого променя?
23. Як можна спростити відтворення ефекту дзеркального відбиття на основі моделі Фонга?
24. Для яких поверхонь використовують метод плоского зафарбування? Назвіть його переваги та недоліки.
25. За допомогою якої функції OpenGL реалізують режим плоского зафарбування?
26. У чому суть інтерполяційного зафарбування?
27. За допомогою якої функції OpenGL реалізують режим інтерполяційного зафарбування?
28. Опишіть алгоритм методу зафарбування Гуро. Назвіть переваги та недоліки цього методу.
29. Опишіть алгоритм методу Фонга для зафарбування багатокутників.
30. У яких графічних системах використовують метод Фонга?
31. Порівняйте методи зафарбування Гуро та Фонга.
32. Яку функцію OpenGL використовують для задання точкових джерел світла?
33. Які типи джерел світла підтримує OpenGL?
34. Скільки джерел світла можна використовувати в одній програмі?
35. Яка символьна константа задає положення і тип джерела світла?
36. Якими символьними константами описують фонове й дифузне освітлення та ефекти дзеркального відбиття?
37. Як в OpenGL враховують радіальне загасання інтенсивності?
38. За допомогою яких символьних констант описують прожектори?
39. Якою функцією задають параметри освітлення на глобальному рівні?
40. Яку команду використовують для обчислення вектора **V** (від точки поверхні до точки спостереження) у дзеркальному відбитті? У яких випадках це необхідно робити?
41. За допомогою якої функції задають метод зафарбування?
42. Які методи зафарбування реалізовані в OpenGL?
43. Якою функцією задають коефіцієнти відбиття та інші оптичні властивості поверхонь?

44. Як надати реалістичності зображенню? Яку символьну константу для цього треба використати?
45. Як в OpenGL задають атмосферні ефекти?
46. Як моделюють ефекти прозорості?
47. Якою командою задають декартові компоненти вектора нормалі до поверхні?
48. Коли використовують моделі глобального розподілу світла?
49. У яких сценах модель трасування променів адекватно передає оптичні ефекти?
50. Для яких сцен краще застосовувати модель аналізу випромінювання?
51. Якою послідовністю операторів встановлюють режим згладжування для прямих і точок?

9.9. Приклади тестових питань

1. Модель освітлення, яка дає змогу сформувати тінь від одного об'єкта на інший і врахувати відблиски, спричинені дзеркальним відбиттям світла від одного об'єкта в напрямку іншого, називають:

   а) локальною;
   б) глобальною;
   в) моделлю Фонга.

2. Складову дзеркального відбиття в загальному відбитому світловому потоці для моделі Фонга описують виразом:

   а) $\frac{k_d L_d \mathbf{N} \cdot \mathbf{L}}{a + bd + cd^2}$;

   б) $\frac{k_s L_s (\mathbf{R} \cdot \mathbf{V})^{\alpha}}{a + bd + cd^2}$;

   в) $\frac{k_a L_a}{a + bd + cd^2}$.

3. У разі фонового освітлення всі точки на поверхні будь-якого з об'єктів сцени отримують від джерела світла:

   а) однакової інтенсивності і відбивають його однаково;
   б) однакової інтенсивності і відбивають його по-різному;
   в) різної інтенсивності і відбивають його по-різному.

4. Всі промені, які випускає віддалене джерело світла;

   а) розходяться в усіх напрямах;
   б) є паралельними;
   в) формують конус з вершиною в точковому джерелі.

5. Поверхня з ідеальним дифузним відбиттям відбиває світло, що падає на неї:

   а) по-різному в усіх напрямках;

б) однаково в усіх напрямках;
в) в одному напрямку.

6. В області світлотіньового переходу:

а) контраст видається менш різким, ніж насправді;
б) контраст видається більш різким, ніж насправді;
в) око не помічає контрасту.

7. Метод Фонга порівняно з методом Гуро дає змогу одержати тонування зображення:

а) менш гладке, і при цьому обсяг обчислень не зростає;
б) більш гладке, але при цьому обсяг обчислень різко зростає;
в) більш гладке, і при цьому обсяг обчислень не зростає.

8. OpenGL має процедури для зафарбовування поверхонь методом:

а) Фонга;
б) Гуро;
в) дифузного відбиття.

9. В OpenGL немає процедур для:

а) плоского зафарбовування;
б) моделі трасування променів;
в) зафарбовування поверхонь за методом Гуро.

10. Модель трасування променів:

а) є локальною моделлю освітленості;
б) адекватно передає оптичні ефекти в сценах з дзеркальними поверхнями;
в) краще передає оптичні ефекти в сценах з матовими поверхнями.

11. Механізм формування прозорих об'єктів, який дає змогу обійтись без трасування променів у явному вигляді, отримав назву:

а) аналізу випромінювання;
б) альфа-змішування;
в) режиму згладжування.

12. Метод аналізу випромінювання:

а) є сенс застосовувати для відображення сцен, де більшість об'єктів мають дзеркальні властивості;
б) є локальною моделлю освітленості;
в) добре підходить для тонування сцен, де більшість об'єктів має поверхні з дифузійним розсіюванням.

13. Функцію **glLight*(lightName, lightProperty, propertyValue)** використовують для:

    а) задання точкових джерел світла;
    б) вибору методу зафарбування;
    в) вибору коефіцієнтів відбиття та інших оптичних властивостей поверхонь.

14. Функцію **glMaterial*(surfFace, surfProperty, propertyValue)** використовують для:

    а) задання точкових джерел світла;
    б) вибору методу зафарбування;
    в) вибору коефіцієнтів відбиття та інших оптичних властивостей поверхонь.

Розділ 10

ОПЕРАЦІЇ ЗІ ЗОБРАЖЕННЯМ НА РІВНІ РАСТРОВОГО ПОДАННЯ

У деяких алгоритмах, розглянутих раніше, вже використано два внутрішні буфери графічної системи – буфер кадру і буфер глибини (або Z-буфер). У всіх внутрішніх буферів є одна спільна риса – їхня дискретна структура: вони мають обмежену кількість комірок (просторову роздільну здатність) і обмежену розрядність (роздільну здатність за глибиною).

10.1. Буфери і накладання

Двовимірний буфер можна означити як блок пам'яті з просторовою розмірністю $n \times m$ комірок і розрядністю кожної комірки k бітів, де параметри n і m буфера глибини відповідають просторовій роздільній здатності екрана. Терміном *k-а бітова площина* назвемо сукупність k-х розрядів у всіх $n \times m$ комірках буфера, а терміном *піксель* – всі k розрядів однієї комірки, яка відповідає одному елементу зображення. Це означає, що піксель може мати тип **byte**, **integer** чи **float** залежно від того, чи використовують цей буфер і який формат вибрано для зберігання в ньому інформації.

Спробуємо внести нерегулярність у двовимірний образ моделі на стадії тонування і растрового перетворення замість того, щоб ускладнювати тривимірну модель. Побудований образ криволінійної поверхні чи плоского багатокутника можна розбити на дрібні фрагменти, кожен із яких має розмір, що не перевищує розміру пікселя екрана. Для формування остаточного зображення треба призначити колір чи тон кожному фрагменту.

Усе починається з вибору моделі зафарбовування. Далі розпочинає роботу *алгоритм накладання*. Його можна розглядати або як алгоритм модифікації параметрів моделі зафарбовування, який опирається на деякий двовимірний масив даних – *карту*, або як алгоритм модифікації параметрів *поверхні*, яку обробляє алгоритм зафарбовування, наприклад, властивостей поверхні або напряму нормалі. Отож, за цим підходом можна виділити:

- накладання проективної текстури;
- накладання мікрорельєфу,
- накладання параметрів середовища.

В алгоритмі *накладання проективної текстури* використано деякий шаблон (або *текстуру*) для формування кольору фрагмента. Текстура може мати регулярний характер і зберігатись як фіксований масив; інколи застосовують методи динамічного формування текстур або в ролі текстури використовують зовнішнє зображення. У будь-якому разі можна вважати, що накладання проективної текстури на поверхню є частиною процесу тонування цієї поверхні. Накладанням проективних текстур можна опрацювати деталі образу гладкої поверхні.

Другий метод (*накладання мікрорельєфу)* використовують, щоб зробити поверхню менш гладкою, наклавши на неї "бульбашки" – *карту мікрорельєфу*. Накладання карт відбиття або *карт середовища* дає змогу сформувати зображення, подібне до створеного під час трасування променів, хоч самої процедури трасування і не виконують. У цьому випадку зображення предметів оточення (середовища) накладають на поверхню і створюють ілюзію дзеркального відбиття.

Три названі групи методів мають багато спільного. Вони модифікують результат тонування, ніяк не заторкуючи геометричної моделі об'єкта. Всі методи реалізовують синхронно з тонуванням як частину кінцевого етапу цього процесу. Всі вони ґрунтуються на деяких зразках – картах, що зберігаються у вигляді одно-, дво- чи тривимірного дискретизованого зображення. Всі методи потребують прийняття спеціальних заходів для згладжування сходинок чи зубців на межах областей.

10.2. Накладання проективних текстур

Зразки текстур можуть мати різний вигляд – від смуг і кліток до складних зображень, які відтворюють зрізи натуральних матеріалів, таких як мармур, граніт, дерево. Розглянемо роботу лише з двовимірними текстурами, хоч методи, які тут буде використано, цілком підходять до одно-, три- і чотиривимірних текстур.

10.2.1. Двоетапний процес накладання текстур

Процес накладання проективних текстур виконують у декілька етапів. Перед початком процедури треба мати двовимірний зразок (шаблон) текстури – функцію інтенсивності $T(s,t)$. Незалежні змінні s і t називають *координатами текстури*, елементи масиву зразка текстури інколи за аналогією з пікселями називають *текселями*. *Карта накладання* асоціює з кожною точкою геометричного об'єкта інтенсивність T відповідної точки образу, причому точки поверхні, зі свого боку, відображаються на простір координат екрана.

Під час накладення текстури, по-перше, треба визначитися з методом зіставлення координат текстури з геометричними координатами. Двовимірний зразок зазвичай визначають на прямокутній області в просторі текстур. Відобразити цей прямокутник на область довільної форми в тривимірному просторі може бути доволі складно або й неможливо, не спотворивши форми зразка або пропорцій його компонент (співвідношень між відстанями). По-друге, сама організація процесу тонування (його виконують послідовним перебором пікселів) змушує цікавитись швидше зворотною функцією відображення, тобто відображенням координат екрана на координати текстури. По-третє, оскільки тут обчислюють коди засвічення

пікселів, кожен з яких визначає колір елементарної прямокутної області зображення, то більший інтерес становить функція відображення *області* (а не точки) в одному просторі на *область* (а не точку) в іншому. Тому виникає проблема згладжування меж між областями, без вирішення якої можуть з'явитися несподівані ефекти на зразок муару.

Розглянемо підхід до вирішення проблеми вибору функції відображення, який передбачає розподіл процесу на дві стадії. На першій текстуру відображають на проміжну поверхню стандартного вигляду – сферу, циліндр чи куб, на другій – стандартну поверхню відображають на поверхню тонованого об'єкта. Такий двоетапний процес можна застосовувати як до параметрично заданих поверхонь, так і до поверхонь, описаних функціями в геометричних координатах.

Припустимо, що координати текстури мають інтервал зміни (0, 1) і що в ролі проміжного об'єкта використано циліндр висотою h і радіусом r. Точки циліндричної поверхні задано параметрично: $x = r\cos(2\pi u)$, $y = r\sin(2\pi u)$, $z = v/h$, причому параметри u і v також змінюються в інтервалі (0, 1). Отже, можна використати функцію відображення у вигляді $s = u$, $t = v$. Зразок буде накладено на циліндр без спотворення форми, тобто без зміни співвідношення між окремими елементами. Якщо в ролі проміжного об'єкта використовують сферу радіуса r, то один із варіантів функції відображення має вигляд $x = r\cos(2\pi u)$, $y = r\sin(2\pi u)\cos(2\pi v)$, $z = r\sin(2\pi u)\sin(2\pi v)$.

Суть другого етапу – відображення проміжної поверхні з уже накладеною текстурою на поверхню тонованого об'єкта сцени. Є три можливі варіанти стратегії відображення. У першому значення інтенсивності зразка в точці проміжної поверхні передають точці об'єкта, в яку "впирається" нормаль до проміжної поверхні (варіант використання нормалі до проміжної поверхні). Другий реалізує приблизно таку саму стратегію, але в зворотному напрямку – від об'єкта до проміжної поверхні (використання нормалі до поверхні тонованого об'єкта). Точці об'єкта передають значення інтенсивності зразка текстури в тій точці проміжної поверхні, в яку "впирається" нормаль до поверхні об'єкта. Третій варіант реалізує "центральну" стратегію: якщо об'єкт має таку форму, що в ній можна визначити центральну точку, то на поверхню об'єкта передають значення інтенсивності текстури в точках проміжної поверхні, які лежать на перетині з променем, який виходить з центра (використання центра тонованого об'єкта).

10.2.2. Проективне накладання двовимірної текстури в OpenGL

Для формування текстурних узорів можна використати велику кількість параметрів і опцій. Перед тим, як перейти до розгляду процедур роботи з текстурою, вивчимо базові функції, необхідні для генерації дво- і тривимірних текстурних узорів.

Обов'язковою компонентою програми накладання двовимірної текстури є масив текселів. Припустимо, раніше програмою якимось чином сформовано або зчитано з файла зображення розміром 512×512 і збережено в масиві **my_texels**:

```
GLubyte my_texels[512][512];
```

Параметри двовимірного текстурного узору RGBA задають у двовимірному масиві кольорів за допомогою функцій:

```
glTexImage2D(GL_TEXTURE_2D, level, col_comp, width, height,
   border, dataFormat, dataType, my_texels);
glEnable(GL_TEXTURE_2D);
```

Перший аргумент обох функцій показує, що масив текстури визначено для двовимірного об'єкта (**GL_TEXTURE_1D**, **GL_TEXTURE_3D** – для одно- та тривимірного відповідно). Якщо невідомо, чи система підтримує текстурні узори із заданими параметрами, використовують константу **GL_PROXY_NEXTURE_2D** і задають значення першого аргумента функції **glTexImage2D**. Це дасть змогу змогу спочатку запитати систему про підтримку такої текстури, а потім визначити елементи масиву текстури.

За допомогою другого і шостого параметрів (**level** і **border**) можна виконувати тонке налаштування режиму роботи зі зразком. Значення 0 другого аргумента означає, що цей масив не є скороченою версією якогось більшого масиву текстури. Значення 0 шостого аргументу означає, що навколо текстури не треба створювати рамки, а значення 1 (єдина друга можливість) – що узор буде відображено з однопіксельною межею навколо нього (це корисно, якщо треба з'єднати цей узор із сусіднім текстурним узором).

Значення параметра **col_comp** визначає кількість (від 1 до 4) колірних компонент (RGBA); зазначимо, що 4-вимірні значення (символьна константа GL_RGBA) часто обробляються ефективніше, бо вони узгоджені з межами областей пам'яті процесора. Є багато інших специфікацій кольору, враховуючи задання кольору єдиним значенням інтенсивності чи яскравості.

Масив для зберігання зразка текстури **my_texels**, який описує його кольори і межі, повинен мати розмірність **width**×**height**, причому ширина і висота мають бути степенями 2 (якщо межу не використовують) або 2 плюс степінь 2 (якщо є межа). Якщо треба визначити текстурний узор, складений з 8 кольорів без використання меж, масив текстури має містити 4 × 8 × 8 = 256 елементів. Для двовимірного текстурного узору присвоїмо елементам масиву текстури коди кольорів у висхідному (від низу масиву до верху) порядку. Починаючи з лівого нижнього кута колірного узору задамо елементи першого рядка масиву RGBA-значень, які відповідають верху прямокутного простору текстур.

Параметри **dataFormat**, **dataType** подібні до аргументів функцій **glDrawPixels** та **glReadPixels** (підрозділи 2.2.1.2, 2.2.1.3). Параметру **dataFormat** присвоюють символьну константу, щоб визначити, як задають коди кольору в масиві текстури: наприклад, **GL_BRGA** показує, що компоненти кольору задають у порядку "синій, зелений, червоний, альфа". Щоб визначити тип даних **GL_RGBA** чи **GL_BRGA**, можна присвоїти параметру **dataType** значення **GL_BYTE**, інші значення, залежні від вибраного формату даних, – **GL_INT**, **GL_FLOAT**.

Приклад задання двовимірного текстурного узору:

```
glTexImage2D(GL_TEXTURE_2D, 0, 3, 512, 512, 0, GL_RGB,
   GL_UNSIGNED_BYTE, my_texels);
```

На об'єкт сцени можна накласти декілька копій текстури або будь-який неперервний піднабір кольорів текстури. Під час відображення групи текстурних елементів на одну чи декілька областей пікселів межі текстурних елементів переважно не вирівняно за межами пікселів. Область пікселя може поміщатися в межі одного текстурного елемента **RGB** (або **RGBA**) або ж вона може захоплювати декілька текстурних елементів.

Пікселям поверхні на сцені можна присвоювати кольори ближчих елементів текстури чи інтерпольований колір текстур. Режим *збільшення чи зменшення узору текстури*, щоб він заповнював задану область, налаштовують за допомогою відповідних аргументів у виклику функції **glTexParameteri**:

```
glTexParameteri(GL_TEXTURE_2D, GL_ TEXTURE_MAG_FILTER,
GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_ TEXTURE_MIN_FILTER,
GL_NEAREST);
```

Заданий тут останнім параметром режим вибору ближчого текселя може призвести до зниження якості зображення. Кращі результати дає *усереднення інтенсивності текселя*; цей режим встановлюють, замінивши константу **GL_NEAREST** на **GL_LINEAR**, яка дає змогу присвоїти пікселям поверхні інтерпольований колір.

Наступний етап – повідомити графічній системі, як текстуру потрібно відображати на геометричний об'єкт. Зразок текстури має дві координати *s* і *t*, які зазвичай змінюються в інтервалі (0.0, 1.0). У розглянутому прикладі пара значень (0.0, 0.0) відповідає елементу масиву **my_texels[0][0]**, а пара (1.0, 1.0) – елементу **my_texels[511][511]**.

Координати у двовимірному просторі текстур вибирають за допомогою команди

```
glTexCoord2*(sCoord, tCoord);
```

простір текстур нормований так, щоб узор було задано координатами з діапазону 0.0–1.0. Однак можна використовувати будь-які координати у просторі текстур і задавати їх у різних форматах: **b**, **s**, **i**, **f** чи **d**, а також у формі масиву з додаванням суфікса **v**.

10.3. Накладання мікрорельєфу

Технологія накладання рельєфу дещо спотворює параметри форми поверхні, зокрема компоненти нормалі до різних її ділянок, які використовують в алгоритмах тонування. Внаслідок цього з'являється нерегулярність в обчислених кольорах окремих пікселів зображення поверхні на екрані.

Оскільки нормаль до ділянки поверхні у певній точці характеризує форму поверхні в ній, то внесення невеликих спотворень до компонентів вектора нормалі призведе до появи локальних спотворень форми поверхні – виникнення ефекту мікрорельєфу. Врахування цих спотворень на стадії тонування збереже вплив на зображення поточної конфігурації джерел світла і решти факторів, які потрібно враховувати у тонуванні.

Внести спотворення в компоненти нормалі можна різними способами. Покажемо, як це робити, якщо поверхню задано параметрично. Нехай $\mathbf{p}(u,v)$ – точка на поверхні. Одиничний вектор нормалі до поверхні в цій точці є векторним добутком векторів дотичних, утворених як часткові похідні за основними параметрами:

$$\mathbf{N}=\frac{\mathbf{p}_u\times\mathbf{p}_v}{|\mathbf{p}_u\times\mathbf{p}_v|},\ \text{де}\ \mathbf{p}_u=\begin{bmatrix}\frac{\partial x}{\partial u}\\ \frac{\partial y}{\partial u}\\ \frac{\partial x}{\partial u}\end{bmatrix},\ \mathbf{p}_v=\begin{bmatrix}\frac{\partial x}{\partial v}\\ \frac{\partial y}{\partial v}\\ \frac{\partial x}{\partial v}\end{bmatrix}.$$

Припустимо, що в деякій точці виконано зміщення в напрямку нормалі малої ділянки поверхні, описане відомою *функцією мікрорельєфу* $d(u,v)$, причому значення функції доволі малі ($|d(u,v)<<1|$): $\mathbf{p}'=\mathbf{p}+d(u,v)\mathbf{N}$. Але тут потрібно не змінювати геометрії об'єкта, а тільки створити ілюзію такої зміни. Для цього можна змінювати не координати точки $\mathbf{p}$, а компоненти вектора $\mathbf{N}$, а потім змінений вектор передавати до алгоритму тонування. Нормаль у точці $\mathbf{p}'$ – це векторний добуток

$$\mathbf{N}'=\mathbf{p}'_u\times\mathbf{p}'_v. \tag{10.1}$$

Вектори $\mathbf{p}'_u,\mathbf{p}'_v$ можна обчислити так:

$$\mathbf{p}'_u=\mathbf{p}_u+\frac{\partial d(u,v)}{\partial u}\mathbf{N}+d(u,v)\mathbf{N}_u,$$

$$\mathbf{p}'_v=\mathbf{p}_v+\frac{\partial d(u,v)}{\partial v}\mathbf{N}+d(u,v)\mathbf{N}_v. \tag{10.2}$$

Якщо d мале, то останніми членами в правих частинах обох рівнянь (10.2) можна знехтувати і, враховуючи $\mathbf{N}\times\mathbf{N}=0$, після підстановки (10.2) в (10.1) одержимо вираз для обчислення компонент спотвореної нормалі:

$$\mathbf{N}'=\mathbf{N}+\frac{\partial d(u,v)}{\partial u}\mathbf{N}\times\mathbf{p}_v+\frac{\partial d(u,v)}{\partial v}\mathbf{N}\times\mathbf{p}_u.$$

Вектор різниці між вихідною і спотвореною нормалями лежить у площині, дотичній до поверхні в точці $\mathbf{p}$. Щоб виконати накладання карти мікрорельєфу, потрібні два масиви, які містять значення $\partial d/\partial u$ і $\partial d/\partial v$. Елементи цих масивів треба обчислити наперед, скориставшись методами, аналогічними до тих, які було застосовано для формування проективних структур. Спотворення компонент нормалей реалізують у процесі тонування.

10.4. Накладання зображення предметів оточення

Якщо до складу сцени входять предмети з гладкою поверхнею, то в реальній обстановці на цій поверхні завжди видно відбите зображення близько розміщених об'єктів сцени – відбиток середовища. Таке зображення у графічній системі можна сформувати, скориставшись методом трасування променів. Але його застосування

пов'язано з великим обсягом обчислень, тому відбите зображення доцільно сформувати іншим способом, схожим на накладання текстури. У цьому методі використано приблизно таку саму технологію, як і у методі накладання текстур, але замість штучно створених зразків текстур використовують *карти середовища* або *карти відбиття*.

В основу методу покладено доволі просту ідею. Як відомо, на поверхні дзеркала відбивається навколишнє середовище. Якщо не пробувати відтворити програмно процес відбиття, а просто взяти до уваги, що на поверхні треба сформувати схоже зображення, то це можна зробити за допомогою двоетапної процедури. На першому етапі потрібно одержати зображення сцени в тому вигляді, в якому воно проектується на проміжну уявну поверхню спостереження (не обов'язково площину). Центр проекції при цьому розміщений у центрі об'єкта з відбивальною поверхнею, а під час формування проекції сам об'єкт забирають зі сцени. Сформовані в підсумку проекції зображення надалі використовують як звичайні зразки текстури, а їхнє накладання виконують одним із раніше розглянутих способів. Але варто врахувати, що в процесі перенесення зображення з проміжної поверхні об'єкта необхідно змінити положення спостерігача і напрямок нормалі до поверхні.

Технологію накладання зображень навколишніх предметів підтримує набір функцій OpenGL. Після того, як сформовано потрібну текстуру, – чи скануванням, чи за допомогою проекції сцени – OpenGL автоматично формує координати текстури для сферичного накладання. Алгоритми, реалізовані як функції OpenGL, зіставляють дотичну до поверхні об'єкта з дотичною до поверхні сфери, на яку накладається текстура. Маючи координати вершин і компоненти векторів нормалей, можна знайти кут відбиття. Потім треба відшукати на сфері точку, дотична до якої має такий самий напрямок. У програмі цей алгоритм реалізує така послідовність операторів:

```
glTexGenfv(GL_S, GL_SPHERE_MAP, 0);
glTexGenfv(GL_T, GL_SPHERE_MAP, 0);
glEnable(GL_TEXTURE_GEN_S);
glEnable(GL_TEXTURE_GEN_T);
```

10.5. Контрольні питання

1. Дайте означення бітової площини.
2. Перелічіть алгоритми накладання. Чи модифікують вони геометричну модель об'єкта?
3. У чому суть алгоритму накладання проективної текстури?
4. Коли використовують метод накладання мікрорельєфу?
5. Який ефект створює накладання карт середовища?
6. Опишіть двоетапний процес відображення проективної текстури.
7. Якою функцією визначають масив для зберігання двовимірного зразка текстури?
8. За допомогою якої команди можна відобразити координати текстури на геометричний об'єкт?
9. Опишіть технологію накладання мікрорельєфу.

10. Яку ідею покладено в основу методу накладання параметрів середовища?
11. Який набір функцій OpenGL підтримує технологію накладання зображень навколишніх предметів?

10.6. Приклади тестових питань

1. Алгоритм накладання проективної текстури:
   а) використовує для формування кольору фрагмента деякий штучно створений шаблон, який може мати регулярний характер і зберігатись як фіксований масив;
   б) вносить деякі спотворення в параметри форми поверхні і нерегулярність в обчисленні кольору окремих пікселів зображення поверхні на екрані;
   в) використовує для формування кольору фрагмента карти середовища або карти відбиття.
2. Алгоритм накладення мікрорельєфу:
   а) використовує для формування кольору фрагмента деякий штучно створений шаблон, який може мати регулярний характер і зберігатись як фіксований масив;
   б) вносить деякі спотворення в параметри форми поверхні і нерегулярність в обчисленні кольору окремих пікселів зображення поверхні на екрані;
   в) використовує для формування кольору фрагмента карти середовища або карти відбиття.
3. Алгоритм накладання параметрів середовища:
   а) використовує для формування кольору фрагмента деякий штучно створений шаблон, який може мати регулярний характер і зберігатись як фіксований масив;
   б) вносить деякі спотворення в параметри форми поверхні і нерегулярність в обчисленні кольору окремих пікселів зображення поверхні на екрані;
   в) використовує для формування кольору фрагмента карти середовища або карти відбиття.
4. Алгоритми накладання:
   а) модифікують, і геометричну модель об'єкта теж;
   б) модифікують, а геометричну модель об'єкта ні;
   в) не модифікують, але змінюють геометричну модель об'єкта.
5. Алгоритми накладання:
   а) не потребують спеціальних засобів для згладжування сходинок чи зубців на межах областей;
   б) потребують спеціальних засобів для згладжування сходинок чи зубців на межах областей;
   в) потребують ускладення тривимірної моделі.

Частина II

Мультимедійні засоби

Розділ 1

ВСТУП ДО МУЛЬТИМЕДІА

Будь-яке явище реального світу можна описати за допомогою декількох засобів подання інформації, кожен із яких передає це явище своїми способами, впливаючи на різні органи чуттів. Одним із ключових моментів комп'ютерних технологій є те, що всі ці види інформації можуть бути цифровими й існувати у вигляді структурованого набору бітів. Ними можна керувати за допомогою комп'ютерних програм, записувати на пристрої для зберігання інформації та передавати через мережу. Те, що всі вони цифрові, означає, що різні види інформації можна поєднувати в одне ціле й одержувати те, що називають *мультимедіа* (від англ. *multi* – "багато"; *media* – "засіб інформації").

1.1. Поняття мультимедіа

Згідно з означенням Європейської комісії з проблем впровадження і використання нових технологій

> *мультимедіа* – це програмний продукт, що містить колекції зображень, текстів і даних, які супроводжуються звуком, відео, анімацією та іншими візуальними ефектами, а також оснащений інтерактивним інтерфейсом з елементами керування.

У вужчому значенні під цим терміном розуміють звукові записи, анімаційні та відеозображення, збережені у вигляді аудіо- та відеофайлів.

Власне об'єднання різних видів інформації не є чимось новим. Навпаки, різні засоби поєднуються щодня у звичному для нас вигляді: випуск телевізійних новин, наприклад, може містити звук і відео, нерухомі зображення, графічні ілюстрації і текст.

Чим же цифрові мультимедійні засоби відрізняються від попередніх форм об'єднання видів інформації? Комп'ютерні програми можуть сприймати біти, якими представлені текст, звук, малюнки тощо як звичайні дані, керувати порядком, у якому подаються і поєднуються різні компоненти, причому робити це відповідно до дій користувача. Інакше кажучи, цифрові мультимедійні засоби можуть бути *інтерактивними,* що неможливо у випадку звичайних видів

інформації і що виходить далеко за межі простого керування, здійснюваного за допомогою відеомагнітофона або DVD-плеєра.

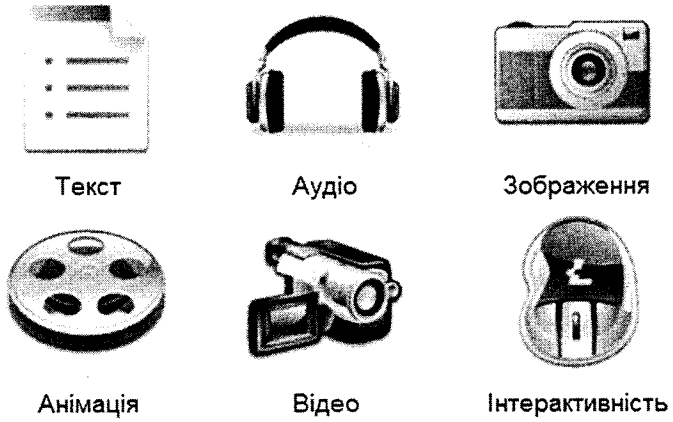


Рис. 1.1. Основні складові мультимедіа

1.1.1. Історія виникнення терміна "мультимедіа"

Порівняно з такими традиційними формами, як фільми й телепередачі, мультимедійні засоби усе ще перебувають на ранньому етапі свого розвитку. Опис компакт-диска опубліковано в 1985 році, а приводи компакт-дисків у настільних комп'ютерах з'явилися приблизно в 1989 році; World Wide Web стала загальнодоступною на початку 1992 року; до січня 1997 року, коли було прийнято специфікацію HTML 3.2, аудіо та відеофайли підтримувалися на Web-сторінках тільки з використанням своїх запатентованих засобів. У 1995 році оголошено про специфікацію DVD (Digital Video Disk), який став новою можливістю для інтерактивних мультимедійних засобів. До початку XXI століття користувачі Інтернету нарешті отримали високу пропускну здатність мереж, достатню не лише для завантаження мультимедійних продуктів, але й для одержання їх у потоковому форматі та в режимі реального часу.

Що ж до самого терміна "мультимедіа", то в 1965 році його вперше використано для опису **Exploding Plastic Inevitable** – шоу, що поєднувало в собі живу рок-музику, кіно, експериментальні світлові ефекти і нетрадиційне мистецтво. Це шоу організував Енді Воргол, засновник художньої школи поп-арту. До речі, ім'я Енді Воргола від народження – Андрій Варгола. Він народився у серпні 1928 року в місті Піттсбурзі (штат Пенсильванія, США) в родині лемків-українців, емігрантів першого покоління.

Упродовж сорока років термін набував різних значень. Наприкінці 1970-х років він позначав презентації, складені із зображень, одержуваних від декількох проекторів та синхронізованих зі звуковою доріжкою. У 1990-х термін "мультимедіа" набув сучасного значення, а в 1995 році Товариство німецької мови, щоб визнати значущість і повсюдність цього слова, нагородило його званням "Слово року".

1.1.2. Термінологія

Як можна назвати поєднання різних видів інформації із програмним керуванням? Якщо головна мета такого поєднання – одержання зображення і демонстрація мультимедійних елементів, то прийнято говорити про *мультимедійну продукцію.* Якщо ж робота з мультимедійним файлом більше пов'язана з обчисленням, то говорять про *мультимедійний застосунок.*

Людина *читає* текст, *бачить* малюнки, *дивиться* кіно й *чує* звуки, але що вона робить із мультимедійною продукцією? Напевно, все перераховане разом; крім того, ми взаємодіємо з нею. Важливим аспектом є те, що тут користувач є *активним* у прямому значенні цього слова, керуючи мультимедіа за допомогою миші, клавіатури чи інших пристроїв (у тих межах, які допускає виробник мультимедіа). Що ж до відображення мультимедійної продукції для перегляду та прослуховування її елементів спеціальними програмами-програвачами, то цей процес найчастіше називають *відтворенням.*

Отож, з погляду сприйняття

> *мультимедіа* називають будь-яку комбінацію двох чи більше видів інформації, поданих у цифровій формі, достатньо ефективно інтегрованих для того, щоб їх можна було відображати з використанням одного інтерфейсу або керувати за допомогою однієї комп'ютерної програми.

1.1.3. Доставляння

Щоб будь-яка мультимедійна продукція могла потрапити від її виробника до споживачів, необхідні засоби для її доставляння. Варто розрізняти *автономне (offline)* і *неавтономне (online)* доставляння.

За неавтономного доставляння для передавання інформації з одного комп'ютера на інший використовують мережу. Це може бути локальна мережа, що обслуговує одну організацію, але найчастіше це всесвітня глобальна мережа Інтернет.

Щоб мультимедіа доставлялися автономно, необхідно використати якийнебудь переносний пристрій для зберігання інформації. Популярні сьогодні пристрої флеш-пам'яті здатні зберігати величезний обсяг даних, мають значно вищу швидкість доступу до інформації, набагато краще переносять зовнішні впливи та мають менший фізичний розмір, аніж CD- та DVD-диски.

Хоча використання переносних пристроїв має свої очевидні переваги, неавтономне доставляння надає таких можливостей, яких немає в автономного. Зокрема, воно дає змогу доставляти мультимедійну інформацію майже в реальному часі, що, зі свого боку, призводить до виникнення нових сфер застосування, наприклад відеоконференцій та мультимедійного радіомовлення.

1.2. Організація мультимедіа

Сьогодні є три базові моделі, які використовують у мультимедіа для організації взаємного розташування елементів різних типів інформації: *контроль за сторінками (page-based)*, *контроль за часом (time-based)* і *контроль за сценою (scene-based).*

1.2.1. Базові моделі організації мультимедіа

У першому випадку текст, малюнки або відео впорядковано у двовимірній структурі, подібно до тексту й малюнків у книгах. Часові елементи, наприклад, відеокліпи й звук, розташовані на сторінці так, неначебто вони займають фіксовану площу; для запуску й зупинки відтворення можуть бути надані панелі керування. Окремі сторінки можна об'єднувати за допомогою посилань. Такі часові мультимедійні продукти зі зв'язками можуть утворювати *гіпермедіа*. Найвідоміший приклад гіпермедійної системи – World Wide Web.

У мультимедійних засобах другого типу основою організації є час. Елементи так розташовують у часі (часто за допомогою часової шкали), що вони утворюють послідовність, яка прокручується з певною швидкістю, створюючи враження неперервного руху в реальному часі. Або ж це можуть бути окремі сторінки, які відображаються одна за однією на зразок слайдів. Мультимедійна презентація (продукція, базована на часовій шкалі) часто реалізує паралелізм: кілька відеокліпів можуть відтворюватися одночасно або ж звукова доріжка може програватися одночасно з анімацією. Елементи можуть бути синхронізовані один з одним і відтворюватися узгоджено.

За третьою моделлю організації мультимедійних продуктів елементи розташовані у просторі тривимірної сцени, користувач може переміщатися з одного місця на інше й розглядати предмети, які йому трапляються. Як її приклад можна розглядати мультимедіа, основане на використанні мови моделювання віртуальної реальності VRML та її наступниці X3D, а також, певною мірою, – стандарт MPEG4. Сьогодні цю модель використовують переважно в комп'ютерних іграх, і хоча їх ще донедавна не вважали мультимедійною продукцією, вони відповідають усім критеріям належності до мультимедіа і є, мабуть, її найперспективнішою частиною.

На практиці зазвичай використовують не якусь одну модель, а їхнє поєднання.

Усі моделі організації мультимедіа об'єднано поняттями *нелінійності* та *інтерактивності*.

1.2.2. Нелінійність

У багатьох традиційних засобах передавання інформації існує очевидна послідовність від однозначно заданого початку до кінця. Звичайно, користувач може, якщо захоче, відійти від цієї лінійної послідовності, пропустивши кілька сторінок у книзі або якийсь фрагмент фільму, але задум автора твору полягає в тому, що воно повинно сприйматися лінійно.

На відміну від цього, суть багатьох часових засобів мультимедіа полягає саме в нелінійності. Це складне поняття, для якого дуже важко знайти якесь єдине тлумачення, і простіше відштовхуватися від конкретного контексту. У розгляданому випадку ключовим є термін *лінійний* – такий, у якому що-небудь розміщується у певній послідовності, в один ряд. Отож, у сфері мультимедіа

нелінійність – це здатність мультимедійного продукту відтворюватись не як однозначна, наперед визначена, послідовність елементів, а в довільній, задаваній користувачем послідовності.

Сама по собі нелінійність не нова. Наприклад, багато книг призначені для нелінійного прочитання: енциклопедії, словники або інша довідкова література. Зміст, предметний покажчик або перехресні посилання – це текстові апарати, які використовують для того, щоб допомогти читачеві знайти те, що йому потрібно. Нелінійністю апріорі володіють гіпертекстові та гіпермедійні документи, а також інтерактивні анімаційні ролики.

А ось постановка п'єси є демонстрацією того, що межа між лінійністю й нелінійністю не зовсім чітка. Незважаючи на те, що п'єсу грають як лінійну послідовність сцен, глядач може звертати увагу на різні ділянки сцени, що дає йому змогу зосереджуватися на різних аспектах дії. Це відрізняється від кінофільму, у якому камера змушує дивитися на кожну сцену певним чином.

1.2.3. Інтерактивність

Мультимедійні продукти часто доповнено інтерактивними особливостями, які дають змогу користувачеві керувати їхнім відтворенням.

Інтерактивність – це також доволі неоднозначне поняття. У кожній сфері застосування цей термін має своє тлумачення. Тому тут знову потрібне прив'язання до конкретної галузі, в цьому випадку – до сфери мультимедіа.

Інтерактивність – це здатність інформаційно-комунікаційної системи активно й різноманітно реагувати на дії користувача, коли цілей можна досягти за допомогою інформаційного обміну між елементами цієї системи. Конкретніше, *інтерактивність у сфері мультимедіа* – це здатність мультимедійних засобів взаємодіяти чи перебувати в режимі діалогу з користувачем для досягнення ним певної мети.

Інтерактивність часто описують як особливість, що відрізняє цифрові мультимедійні засоби від інших видів комбінованих засобів передавання інформації, наприклад, телебачення. Дуже часто кажуть, що вона передає важелі керування в руки користувачів. Однак ступінь пропонованого контролю строго обмежений параметрами, встановленими виробником мультимедіа. Можливі тільки ті варіанти вибору, які закладено в програму.

Оскільки інтерактивність – це новий внесок комп'ютерних засобів у мультимедіа, зараз маємо потужну тенденцію до того, щоб усі мультимедійні продукти були інтерактивними й мали якомога більшу кількість варіантів вибору. Однак не варто забувати, що зростання цих можливостей може виявитися недоречним або зайвим: це зовсім необов'язково буде те, чого хочуть всі й завжди. Іноді користувачі можуть бути цілком задоволені тим, що події розвиваються точно за тим сценарієм, що відповідає задуму автора.

1.3. Оцифрування мультимедійних даних

Використання цифрових мультимедійних засобів ґрунтується на здатності комп'ютерів з високою швидкістю виконувати певні операції. Для цього дані, що містяться в різних видах інформації, потрібно подати в цифровому вигляді. Це означає, що все розмаїття вхідних сигналів від датчиків, яке містить малюнки, текст, рухомі зображення й звук, повинне бути зведене до комбінацій із двійкових цифр. Після виконання такого перетворення для зміни, об'єднання, збереження й відображення інформації всіх типів можна користуватися комп'ютерними програмами.

1.3.1. Цифрове подання даних

Комп'ютери створено на основі пристроїв, які можуть перебувати тільки в одному із двох станів. Можна сказати, що ці пристрої призначені для записування й виконання різних операцій над *бітами* – одиницями інформації, які можуть набувати лише одного з двох значень – або 0, або 1. Байт, що складається з восьми бітів, наприклад, 0, 1, 1, 0, 0, 0, 0 і 1, можна прочитати як двійкове число 01100001, що у десятковій системі числення дорівнює 97.

Однак можна вибрати інший спосіб інтерпретації комбінацій бітів, і це також буде одним з видів цифрового подання різних типів інформації. Комбінація 01100001 може позначати, наприклад, код якогось символа (неоднакового у різних системах кодування). Взаємозв'язок можна встановити й між числами і такими величинами, як, наприклад, яскравість зображення в заданій точці або амплітуда звукової хвилі.

Загальноприйняте подання даних у вигляді набору бітів означає, що за допомогою комп'ютерних програм можна оперувати даними, що належать до всіх видів інформації, як окремо, так і в різних комбінаціях. Сьогодні є чітко визначені способи подання тексту, малюнків, звуку, відео й анімації за допомогою бітів, і можна бути повністю впевненим у тому, що будь-який вид інформації, винайдений у майбутньому, також можна буде відобразити в цифровому вигляді.

1.3.2. Оцифрування мультимедійних даних

Іноді мультимедійні дані вже з'являються в цифровому вигляді, як це відбувається у разі створенні зображень за допомогою програм для малювання чи створення анімації, цифрових відеокамер або диктофонів, але часто доводиться перетворювати якийсь вид аналогового подання інформації на цифровий, наприклад, за допомогою сканування зображення.

У мультимедійних даних є значення, які змінюються кількома способами: або в часі, або в просторі, або й у часі та просторі одночасно. Відповідно до традиції вимірювані значення називають *сигналом,* не роблячи жодної різниці між сигналами, змінними в часі та змінними у просторі.

Коли є аналоговий сигнал, що має здатність змінюватися неперервно, то значення, які вимірюють, та інтервал, на якому роблять вимірювання, можуть

змінюватися на нескінченно малу величину. Якщо потрібно перетворити цей сигнал на цифровий, то в обох випадках (для величини й інтервалу) доводиться обмежитися набором дискретних значень.

Оцифрування – процес перетворення аналогового сигналу на цифровий, який містить набір дискретних значень. Оцифрування складається з двох етапів: *дискретизації,* коли вимірюють величину сигналу на певному наборі точок (вузлів), і *квантування,* коли ці виміряні величини обмежують певним набором значень (рівнів).

Дискретизація і квантування можна виконувати в довільному порядку; на рис. 1.2 показано аналоговий сигнал, який спочатку дискретизують, а потім квантують. Ці процеси зазвичай виконують спеціальні пристрої, які узагальнено називають *аналого-цифровими перетворювачами (АЦП)*. Інтервал між сусідніми точками, на яких беруть виміри, майже завжди фіксований. Кількість елементів вибірки (тобто взятих значень) на фіксованому відрізку часу або простору називають *частотою дискретизації.* Так само рівні, на які квантують сигнал (*рівні квантування*), здебільшого перебувають на однаковій відстані один від одного, хоча інколи на практиці трапляється й інше розташування.

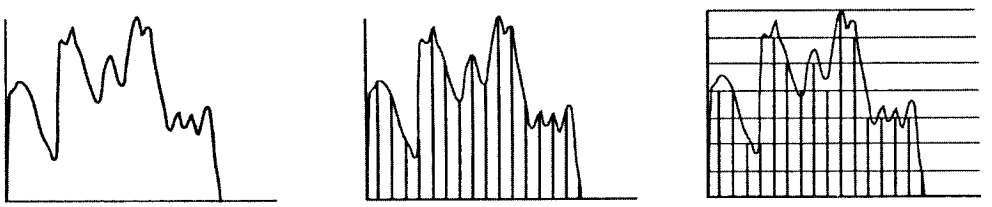


Рис. 1.2. Аналоговий сигнал, дискретизований і квантований

Одна з найбільших переваг цифрового подання порівняно з аналоговим полягає в тому, що коректними є тільки певні значення сигналу – ті, які потрапляють на рівні квантування. Якщо сигнал передають дротами чи бездротовим способом або записують на фізичний носій, на нього неминуче накладаються випадкові шуми, які призводять до зміни значення сигналу. Якщо сигнал аналоговий, то шуми можуть виявитися недетектованими: адже допускаються будь-які аналогові значення, тому сигнал, забруднений шумами, неможливо відрізнити від чистого сигналу. Але якщо сигнал цифровий, то будь-які незначні флуктуації, зумовлені *шумом, зазвичай переводять значення в розряд неприпустимих, які перебувають* між рівнями квантування. Після цього дуже просто відновити вихідний сигнал, *проквантувавши його ще раз.*

Для того, щоб відновити аналоговий сигнал за набором дискретних значень, усе, що потрібно зробити, – це вирішити, чим заповнити проміжки між цими точками. Можна точно описати процес відновлення сигналу за допомогою математичних виразів. На практиці ж використовують простіші методи, які набагато легше реалізувати.

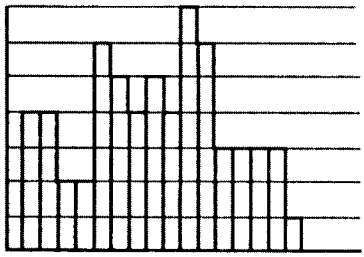


Рис. 1.3. Відновлення сигналу методом вибірки-збереження

Один з можливих способів – використання *методу вибірки-збереження* (або дискретизації із затримкою), який полягає в тому, що значення в точці вибірки залишається постійним на всьому інтервалі між цією точкою й наступною. Як видно з рис. 1.3, це дає сигнал з різкими переходами, що насправді є не дуже доброю апроксимацією оригіналу. Однак, коли сигнал подається на пристрій виведення, такий як екран електронно-променевої трубки або гучномовець, затримки й похибки, які властиві будь-яким фізичним приладам, призводять до того, що різкі переходи згладжуються, і в результаті виходить достатньо добра, з теоретичного погляду, апроксимація.

Проте якщо дискретні точки, на яких взято значення вхідного сигналу, перебувають занадто далеко одна від однієї, будь-яке відновлення ризикує бути неадекватним, оскільки аналоговий сигнал може містити деталі, пропущені в процесі дискретизації. Вплив такої *дискретизації з недостатньою частотою* на результат відновлення сигналу залежить від того, чим є цей сигнал (звуком, зображенням тощо), і від того, змінюється він у часі чи в просторі. Однак ефект в усіх випадках проявляється у вигляді перекручувань і артефактів (помилок), які завжди небажані.

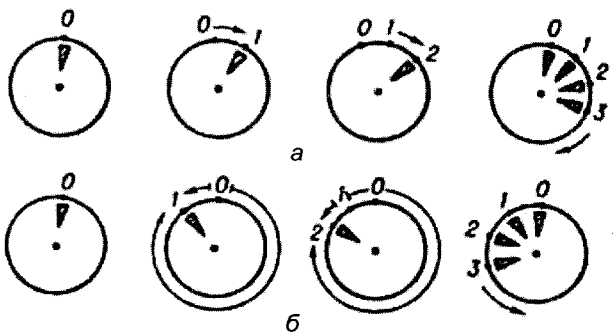


Рис. 1.4. Пояснення стробоскопічного ефекту

У загальному випадку, якщо виконувати дискретизацію сигналу з недостатньою частотою, у процесі відновлення одні частотні компоненти вихідного сигналу можуть перетворюватися на інші. Це явище відоме як *накладання спектрів* і в різних видах інформації проявляється по-різному. У випадку запису звуку воно

чується як спотворення; на зображеннях воно зазвичай проявляється у вигляді зубчастих країв або муарових картин; для рухомих зображень недостатня частота дискретизації за часом призводить до уривчастих рухів, а також до проявів стробоскопічного ефекту – явищ, аналогічних до обертання диска у зворотний до реального бік (рис. 1.4).

1.4. Основні види мультимедійних складових

Як видно з означень мультимедіа, його складові входять в остаточний продукт у взаємодії, як об'єднання чи комбінація. Наприклад, не може бути мультимедійної слайдової презентації з самих лише зображень, без тексту; важко уявити відеозапис, не супроводжуваний звуком чи субтитрами. Та все ж доцільно виокремити основні складові мультимедіа, з якими, власне, й ототожнюють це поняття у повсякденному житті: це *анімація*, *відео* та *аудіо*.

1.4.1. Відео та анімація

Усі наявні сьогодні методи демонстрації рухомих зображень, які записані на плівці або в цифровому вигляді, ґрунтуються *на інертності зорового сприйняття* – затримці реакції ока на зоровий подразник, через що виникають залишкові зображення. Завдяки цьому послідовність нерухомих картинок (*кадрів*), демонстрованих оку з достатньо високою швидкістю (вищою від тієї, котру називають *частотою злиття миготінь*), сприймається ним як неперервний зоровий образ. Якщо послідовні зображення відрізняються незначно, будь-яка плавна зміна сприйматиметься як рух елементів зображення.

Частота злиття миготінь має індивідуальне для кожної людини значення, максимальне становить близько 40 зображень за секунду. Якщо використано нижчу частоту, може бути відчутним ефект миготіння, який стане помітнішим у разі зменшення частоти, поки, нарешті, відчуття руху не буде повністю втрачене.

Є два способи генерації рухомих зображень для мультимедійної продукції. По-перше, за допомогою відеокамери можна записати послідовність кадрів реального руху в реальному світі. По-друге, можна створити всі кадри окремо – або за допомогою комп'ютера, або записуючи по одному нерухомі зображення. Для першого випадку використовують термін *відео,* а для другого – *анімація*.

Анімація (з лат. anima – душа, фр. animation – оживлення) – вид кіномистецтва, твори якого створюють методом знімання послідовних фаз руху мальованих або об'ємних об'єктів.

Інколи вживають також термін *мультиплікація* (з лат. multiplicatio – розмноження, збільшення, зростання).

Відео (від лат. video – дивлюся, бачу) – широкий спектр електронних технологій записування, обробки, передавання, зберігання і відтворення візуального й аудіовізуального змінюваного в часі матеріалу на моніторах.

У побутовому значенні відео означає відеоматеріал, телесигнал або кінофільм, записаний на фізичному носії (відеокасеті, відеодиску тощо).

Анімаційні та відеоматеріали можуть бути аналоговими або цифровими. Історично анімація та відео, записані в аналоговій формі на кіно-, а потім – на відеоплівці, передували сучасним цифровим засобам зберігання.

На відміну від анімації та кіно, телебачення вже на ранніх стадіях використовувало поєднання оптичних, механічних та електронних технологій для отримання, передавання та відображення рухомих зображень. Спочатку телесигнал можна було тільки передавати у режимі реального часу, але від початку 1960-х років для зберігання телевізійних передач почали використовувати магнітну стрічку. В наступному десятилітті аналогові відеозаписи на відеокасетах набули неабиякої популярності, і лише в середині 2000-х їх почали повсюдно замінювати цифрові формати. На зміну традиційним аналоговим технологіям в анімації, кіно і телебаченні прийшли цифрові.

Цифрова (або *комп'ютерна*) *анімація* – технології створення, збереження і демонстрації рухомих зображень за допомогою комп'ютерів.

Цифрове відео – сукупність технологій записування, обробки, передавання та зберігання відеозображень і звуку в цифровому вигляді.

Основна відмінність цифрового відео від аналогового полягає в тому, що відеосигнал і звук кодують і передають не в початковому вигляді, а після аналогово-цифрового перетворення на *бітові потоки* (часову послідовність бітів) відео- та аудіоданих. Бітовий потік можна захоплювати і зберігати на носії інформації у вигляді комп'ютерного файла. Переважно цифрове відео компресують (стискають) для зменшення обсягу даних, призначених для передавання та зберігання.

1.4.2. Цифрове аудіо

Звук, як і відео та анімація, є важливою складовою мультимедійної продукції, де його використовують у формі аудіозапису.

Аудіо (лат. audio "чую") – загальний термін, що стосується звукових технологій; найчастіше під ним розуміють звук, записаний на фізичному носії; рідше мають на увазі записування і відтворення звуку та відповідну апаратуру.

Аудіозапис (або *звукозапис*) – записана на носії (магнітній стрічці, грамофонній платівці, кіноплівці, цифрових носіях інформації тощо) звукова інформація.

Розрізняють аналогове та цифрове аудіо (звук). Термін *аналогове аудіо* загалом означає інформацію про звук, зафіксовану в аналоговому електричному сигналі, отриманому внаслідок записування звуку за допомогою мікрофона. Він може бути збережений на аналогових носіях: грамофонній платівці, магнітній стрічці у вигляді котушки (бобіни) чи магнітофонної касети тощо.

Під *цифровим аудіо* (або цифровим звуком) у загальному випадку розуміють інформацію про звук, зафіксовану в цифровому сигналі, який одержують після оцифрування аналогового сигналу; він може бути збережений на цифрових носіях (жорстких дисках комп'ютера, компакт- чи DVD-дисках, DAT-касетах тощо) у вигляді аудіофайлів.

1.5. Контрольні питання

1. Що таке мультимедіа?
2. Назвіть та охарактеризуйте базові моделі організації мультимедіа.
3. Перелічіть основні складові мультимедіа.
4. Які є способи доставляння мультимедіа і в чому їхні переваги?
5. Поясніть ключову різницю між аналоговим і цифровим сигналами.
6. Що таке оцифрування мультимедійних даних і з яких етапів воно складається?
7. Що називають відновленням сигналу?
8. Охарактеризуйте поняття нелінійності та інтерактивності у сфері мультимедіа.
9. Опишіть основну причину появи стробоскопічного ефекту і наведіть його приклади.
10. Назвіть основні види мультимедійних складових та їхні особливості.

1.6. Приклади тестових питань

1. Поняття відео:
   а) це набір технологій роботи тільки з візуальним матеріалом;
   б) це набір технологій роботи з візуальним та аудіовізуальним матеріалом;
   в) це набір візуальних матеріалів, записаних на цифровому носії.
2. Поняття аудіо:
   а) цей термін стосується технологій роботи тільки зі звуковим матеріалом;
   б) цей термін стосується технологій роботи зі звуковим та візуальним матеріалом;
   в) цей термін стосується звукових матеріалів, записаних на цифровому носії.
3. Поняття мультимедіа:
   а) це об'єднання різних видів інформації, збережене у цифровому або аналоговому форматі;
   б) це синонім до множинних засобів подання інформації;
   в) елементи мультимедіа мають бути подані лише у цифровому вигляді.
4. Поняття інтерактивності у сфері мультимедіа:
   а) це коли мультимедійна система може перебувати в режимі діалогу з користувачем;

б) це здатність апаратних засобів відтворення мультимедіа реагувати на дії користувача;
в) це коли відтворення мультимедіа відбувається не як наперед визначена послідовність елементів.

5. Поняття нелінійності у сфері мультимедіа:

а) це коли відтворення мультимедіа відбувається не як наперед визначена послідовність елементів;
б) це коли мультимедійна система може перебувати в режимі діалогу з користувачем;
в) нелінійність не властива звичайним видам інформації.

6. Моделі організації мультимедіа з контролем за:

а) нелінійністю;
б) інтерактивністю;
в) сайтами;
г) часом.

7. Основні складові мультимедіа:

а) символи тексту;
б) звукові коливання;
в) комп'ютерні зображення;
г) елементи керування.

8. Поняття дискретизації сигналу:

а) дискретизація – етап квантування;
б) квантування – етап дискретизації;
в) характеристикою дискретизації є частота;
г) характеристикою дискретизації є рівні.

9. Поняття квантування сигналу:

а) дискретизація – етап квантування;
б) квантування – етап дискретизації;
в) характеристикою квантування є частота;
г) характеристикою квантування є рівні.

10. Історія терміна "мультимедіа":

а) термін уперше використано для опису комп'ютерних презентацій;
б) термін виник у в 1965 році;
в) термін походить від використання багатьох проекторів;
г) термін став одним зі "слів століття".

Розділ 2

ЦИФРОВА ДВОВИМІРНА І ТРИВИМІРНА АНІМАЦІЯ

Анімацію можна загалом означити як технологію створення ілюзії рухомих зображень (руху та зміни форми об'єктів) за допомогою послідовності нерухомих картинок (кадрів), що змінюють один одного з певною частотою. У XX столітті анімацію широко використовували в індустрії розваг, рекламі, навчанні, мистецтві, пропаганді, в художніх і документальних фільмах, у відео. Сьогодні її широко застосовують в Інтернеті та мультимедійних презентаціях.

2.1. Принципи функціонування і види анімації

Днем народження анімації вважають 30 серпня 1877 року: саме тоді запатентовано винахід французького винахідника Еміля Рено, який розробив основи технології виготовлення анімаційних фільмів, що залишалися незмінними до впровадження комп'ютерних технологій: "покадрова зйомка" малюнків за фазами руху.

Щоб зрозуміти, що таке анімація, потрібно створити на папері послідовність малюнків, що відрізняються елементами, які повинні змінюватися або рухатися. Після створення малюнків їхню послідовність фотографують у певній послідовності по одному за раз. Під час відтворення анімаційного фільму така послідовність нерухомих зображень через інертність зорового сприйняття сприйматиметься наче зображення, що неперервно рухається. Якщо потрібно створити ілюзію швидкого руху або зміни, різниця між сусідніми зображеннями послідовності повинна бути значно більшою, ніж у випадку плавних змін.

Звичайний кінофільм демонструють зі швидкістю 24 кадри за секунду, ефективна частота зміни кадрів для традиційної анімації становить 12 кадрів за секунду. У цифровій анімації частоту задають відповідно до сфери використання готового продукту: для нескладної анімації можна взяти й 12 кадрів за секунду, для професійного мультфільму – стандартне значення 24, для рекламного ролика на телебаченні – 25, для інтерактивної гри – 30.

Анімацію можна створити багатьма різними способами. Деякі з них використовували на початкових етапах становлення мультиплікації, і тепер вони повністю втратили своє значення. Сьогодні виокремлюють такі загальні види анімації:

- *графічна* (або *мальована*) – класичний вид анімації, що ґрунтується на серії мальованих зображень, на кожному з яких анімований об'єкт показано в іншій фазі руху;
- *об'ємна* (*матеріальна*) анімація – її об'єкти є окремими елементами матеріального світу (ляльки, пластилінові форми, витинанки, сипкі матеріали, голки тощо);
- *комп'ютерна* анімація – вид анімації, в якому об'єкти створюють за допомогою комп'ютерних засобів; залежно від розмірності простору сцени розрізняють *двовимірну* і *тривимірну* комп'ютерну анімацію.

2.2. Від традиційної до комп'ютерної анімації

Якщо анімацію створюють винятково за малюнками на папері, кожен аспект зображення потрібно повторювати на всіх малюнках. Щоб скоротити величезний обсяг роботи, яку необхідно виконати в цьому процесі, у традиційній анімації розробили багато технологій. Основні традиційні форми анімації породили аналоги у цифрових мультимедіа.

2.2.1. Основні технології традиційної анімації

Серед технологій, яких у традиційній анімації було розроблено чимало, варто особливо відзначити ті, що були легко і природно перенесені у цифрове середовище.

2.2.1.1. Пошарова (келева) анімація

Однією з найбільш відомих і поширених технологій традиційної анімації був метод *пошарової (келевої) анімації*. Його суть полягає в тому, що елементи сцени, які можуть рухатися, зображують на прозорому матеріалі, який називають келем, і накладають на зображене окремо нерухоме тло. Для створення послідовності зображень для кожного кадру потрібно перемальовувати тільки рухомі елементи на келі, фіксовані ж частини сцени зображають тільки один раз. Келі можна накладати один на одного, змінювати їх на різних кадрах, щоб одержати більш складні й динамічні сцени.

2.2.1.2. Технологія ключових кадрів

Упродовж 30–40-х років XX століття провідні американські виробники мультфільмів, очолювані Уолтом Диснеєм, розробили принцип масового виготовлення анімаційних продуктів. Визначальним моментом цієї розробки був поділ праці. Ідея Диснея полягала в розділенні процесу створення послідовності зображень на підзадачі, принаймні частину з яких міг виконувати відносно некваліфікований персонал. Звісно, розроблення персонажів, придумування ідей і сюжетних ліній, перевірка і певна частина анімації завжди потребували досвідчених і талановитих художників, однак, коли справа доходила до виробництва келів фільму, дії талановитих аніматорів зводилися до створення *ключових кадрів*. У тра-

диційній анімації значення цього терміна полягає в тому, що ключові кадри зазвичай створюють провідні аніматори, які зображають на них пози і детальні характеристики персонажів у важливих точках анімації (рис. 2.1). Після цього помічники аніматора практично механічно можуть малювати проміжні кадри.

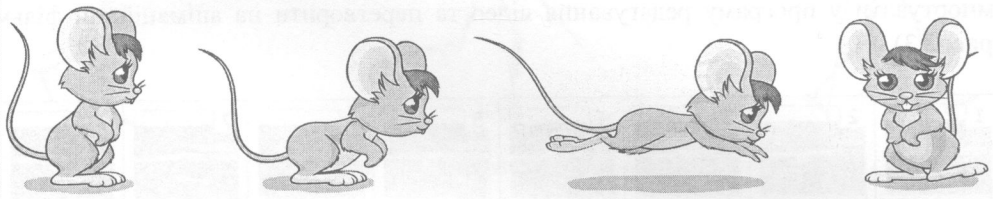


Рис. 2.1. Ключові кадри для мишки, що стрибає

2.2.1.3. Об'ємна анімація

Зовсім інші підходи застосовували у традиційній тривимірній анімації. У цьому понятті об'єднано кілька технологій з використанням мініатюрних тривимірних наборів, у яких об'єкти акуратно переміщували між кадрами. Об'єктами такої анімації можуть бути шарнірні тіла, кінцівки яких можна переміщувати, або нерухомі фігури, частини яких заміняють (або підставляють) між кадрами, щоб створити ефект рухів тіла, ходьби тощо (рис. 2.2).

Рис. 2.2. Кадри традиційної тривимірної (ляльковой) анімації

Можна використовувати фігури та інші об'єкти, виготовлені з податливих матеріалів (наприклад, пластиліну): у цьому випадку фігури можна вручну модифікувати між кадрами, одержуючи природний рух, а також зміни і перетворення, які неможливо створити іншими засобами. Часто трапляються також гібридні форми анімації, наприклад, комбінація келевої і тривимірної анімації.

2.2.2. Перехід до цифрової анімації через оцифрування кадрів

У традиційній анімації кожен її кадр – незалежно від того, зображено його на папері чи келі, побудовано за допомогою тривимірної установки чи створено з використанням іншої технології, – записували на кіно- або відеоплівку. Для найпростішого одержання цифрової анімації логічно підключити відеокамеру

безпосередньо до комп'ютера і зберігати кожен анімаційний кадр на диск у вигляді набору послідовно пронумерованих файлів нерухомих зображень.

Послідовності файлів зображень є дуже гнучким поданням анімації. Окремі файли можна відкривати у графічній програмі й редагувати; файли можна видаляти з послідовності, заміняти іншими або додавати нові. Набір зображень можна також імпортувати у програму редагування відео та перетворити на анімаційний фільм (рис. 2.3).

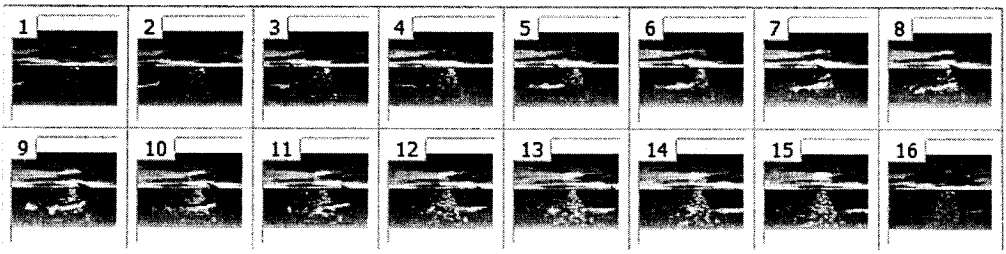


Рис. 2.3. Набір послідовно пронумерованих файлів зображень як кадри анімації

Для мальованої анімації можна обійтися й без традиційної зовнішньої форми та всього процесу оцифрування, використовуючи графічну програму для малювання і записуючи результати роботи послідовність файлів зображень або як готовий ролик.

2.2.3. Цифровий кель і спрайтова анімація

Традиційні келі в анімації можна асоціювати з шарами у графічних програмах, тому ця технологія також допускає природний перехід до цифрової форми. Використовуючи шари як цифровий еквівалент келів, можна заощадити час аніматора, але в такому випадку не залишається незмінним спосіб запису повної анімації: кожен кадр зберігають як файл зображення, а їхню послідовність пізніше перетворять на ролик, анімовану GIF-картинку або іншу форму анімації.

Очевидно, що така послідовність кадрів матиме суттєву надмірність збереженої інформації. Можливо, якщо підсумкова послідовність буде стиснута, надлишковість вдасться зменшити, але стискання послідовності повних кадрів навряд чи буде настільки ж успішним, як її збереження у формі, де вже враховано надмірність. Загалом це означає однократне записування всіх статичних шарів і всіх об'єктів на інших шарах разом з описом принципів перетворення рухомих елементів під час переходу між кадрами.

Таку форму анімації, що ґрунтується на описі рухомих об'єктів, називають *спрайтовою анімацією*, а самі об'єкти – *спрайтами*. Щоб одержати складніший рух, із кожним спрайтом можна співвіднести набір зображень – *граней* (*faces*). Послідовно змінюючи положення спрайта і циклічно переставляючи грані, можна змусити персонаж рухатись (рис. 2.4).

Рис. 2.4. Спрайтові грані циклу ходьби

Тут описано спрайтову анімацію як спосіб записування анімаційної послідовності, проте її також часто використовують по-іншому. Замість того, щоб записувати зміни властивостей спрайтів, ці значення можна генерувати динамічно за допомогою програми. Такі послідовності рухомих зображень, які можна описати алгоритмічно, будуть іще компактнішими. Більш того, обчислення властивостей спрайтів можна зробити залежним від таких зовнішніх подій, як рух мишею, натискання клавіш чи інші дії користувача. Інакше кажучи, рух і зовнішній вигляд анімованих об'єктів може контролювати користувач. Такий спосіб використання спрайтів широко застосовують у комп'ютерних іграх, але за його допомогою можна створювати й динамічну форму взаємодії в інших контекстах, наприклад, для тренажерів.

2.2.4. GIF-анімація як найпростіший вид рухомих зображень

Здатність GIF-файлів зберігати послідовність зображень застосовують для одержання простої і дешевої форми анімації для веб-сторінок. Більшість браузерів послідовно демонструють усі зображення, які містяться у GIF-файлі, вже у процесі його завантаження. Якщо таке відображення відбувається достатньо швидко, набір малюнків виглядатиме як анімація. Залежно від заданих параметрів відтворення анімації триватиме або задану кількість разів, або нескінченно; можна також задавати частоту зміни кадрів.

Основною перевагою анімованих GIF-зображень є те, що вони не залежать від наявності додаткових модулів (plug-in) або використання сценаріїв, так що їх можна переглядати практично у всіх браузерах. Проте GIF-анімація має багато недоліків. До файла неможливо додати звук, у палітрі є лише 256 кольорів, зображення стиснуті із втратами, що може позначитися на їхній якості, при цьому не давши істотного виграшу в розмірі файла. Тому цей вид анімації використовують лише для "оживляння" невеличких простих зображень.

Завдяки своїй простоті GIF-анімацію раніше широко застосовували в рекламних блоках на веб-сайтах. Потім її витіснили Flash-ролики зі значно потужнішими візуальними можливостями, зі звуком та елементами керування для інтерактивної взаємодії з користувачем. Цьому посприяло широке використання у

браузерах Flash-плеєрів, які фактично стали стандартними плагінами. Однак впровадження специфікації HTML 5, яка спрямована на скорочення використання основаних на плагінах технологій, різко зменшило присутність у Всесвітній павутині веб-анімації Flash. Це відродило цікавість розробників і власників сайтів до GIF-анімації, яку тепер знову можна зустріти в рекламі.

2.3. Типові технології створення цифрової анімації

Сьогодні для створення цифрової анімації використовують чимало різних технологій. Розглянемо головні з них.

2.3.1. Анімація ключових кадрів

У традиційній анімації, як було вже сказано, використання цієї технології полягає в тому, що ключові кадри створюють провідні аніматори, а їхні помічники практично механічно малюють проміжні. З погляду математики побудову проміжних кадрів можна описати терміном *інтерполяція* – це розрахунок значень функції, що проходить між відомими точками. Інтерполяція – одне з тих завдань, які доволі вдало можна вирішити за допомогою комп'ютерних програм (рис. 2.5).

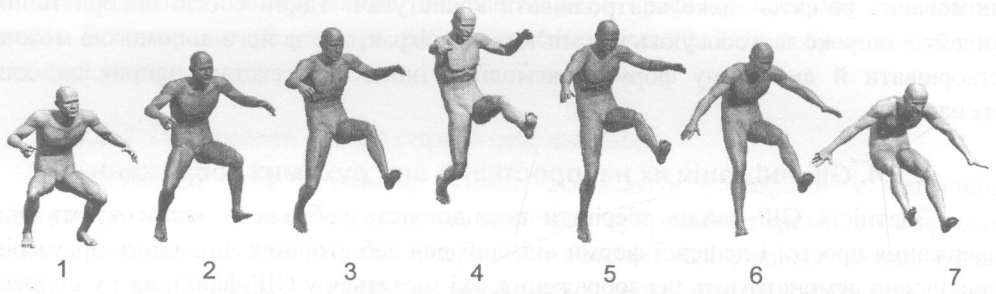


Рис. 2.5. Ключові (1, 4, 7) та проміжні (2, 3, 5, 6) кадри анімації стрибуна

Цифрове подання векторних зображень набагато простіше, ніж растрових, і це робить їх суттєво привабливішими з погляду числової інтерполяції. Перетворення, які застосовні до векторних форм (переміщення, поворот, масштабування тощо), є арифметичними операціями, які можна виконувати за допомогою інтерполяції. Отже, рух, що складається з комбінації цих операцій, можна згенерувати в процесі числової побудови проміжних кадрів на основі наявних ключових. Це означає, що мультиплікацію можна створювати цифровими засобами в таких програмах, як, наприклад, Flash, зі значним скороченням обсягу роботи порівняно із традиційними методами.

2.3.2. Процедурна анімація

Процедурна анімація (*procedural animation*) – вид комп'ютерної анімації, що автоматично генерує послідовності кадрів у режимі реального часу відповідно до встановлених правил, законів та обмежень (рис. 2.6). На відміну від інших видів

анімації, коли аніматор визначає кадри і параметри створюваного продукту, у процедурній анімації результат може бути певною мірою непередбачуваним, і під час кожного запуску генерувати різні результати.

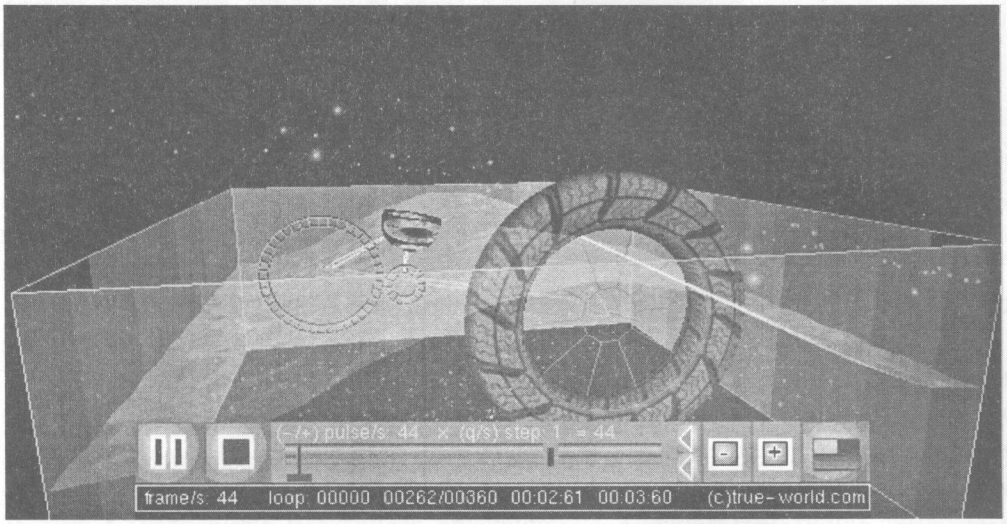


Рис. 2.6. Приклад процедурної анімації, створений у програмі "TRUE"

Процедурну анімацію використовують для створення і моделювання поведінки систем часток (дим, вогонь, вода), тканин і одягу, визначення динаміки твердих тіл і волосся, а також для анімації гуманоїдних і негуманоїдних персонажів. У комп'ютерних іграх процедурну анімацію часто застосовують для таких рухів, як повертання голови персонажа тощо.

2.3.3. Технологія записування руху

Суть цієї технології, яку використовують для створення високоякісної реалістичної тривимірної анімації, полягає в тому, що за допомогою спеціального устаткування записують реальні об'єкти, що рухаються, і переносять одержані дані на їхню імітацію в комп'ютері. Актори у спеціальних костюмах з датчиками виконують рухи, які записують камерами й аналізують спеціальним програмним забезпеченням. Підсумкові дані про переміщення суглобів і кінцівок акторів застосовують до тривимірних кістяків віртуальних персонажів, чим домагаються високого ступеня достовірності їхнього руху. Такий самий метод використовують для перенесення міміки живого актора на його тривимірний аналог у комп'ютері.

Відомий приклад такої техніки – *захоплення рухів* (*motion capture*, рис. 2.7). Цей метод застосовують у виробництві мультфільмів, для створення спецефектів у фільмах, широко використовують в ігровій індустрії.

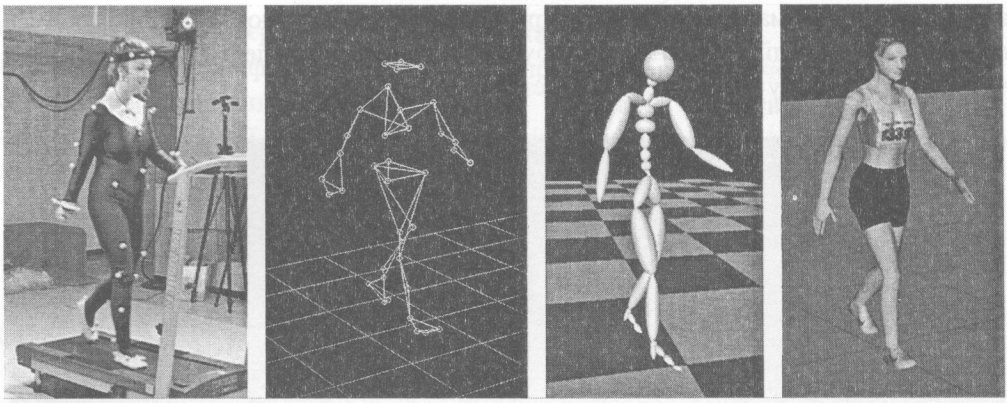


Рис. 2.7. Використання технології захоплення рухів у комп'ютерній анімації

2.4. Засоби створення двовимірної анімації

Виготовити найпростішу двовимірну анімацію не так уже й складно: рухомі GIF-зображення можна створити, наприклад, за допомогою графічного редактора Adobe Photoshop. Для створення складніших кліпів зі звуком та елементами керування, нескладних комп'ютерних ігор або навіть анімаційних фільмів можна скористатись технологією Flash. Для розроблення веб-анімації сьогодні використовують новий тег HTML 5 **<canvas>**, який дає змогу створювати на веб-сторінці двовимірні зображення, та анімують їх засобами JavaScript.

2.4.1. Анімація Flash

Ще донедавна найпопулярнішим форматом веб-анімації був *SWF* (*Shockwave Flash*), файли якого зазвичай генерують з використанням програми Adobe Flash. SWF – це векторний анімаційний формат, де графічні об'єкти можна компактно зобразити у векторній формі, а рух подати за допомогою числових операцій з векторними даними. Тому зазвичай анімація у форматі SWF має менший розмір, ніж відео або будь-який растровий формат. Звичайно, сама лише векторна анімація не дає всіх тих можливостей, які пропонує бітове зображення, однак Flash дає можливість імпортувати в анімацію й растрові об'єкти.

Із запровадженням специфікації HTML 5, спрямованої на поступову відмову від використання плагінів, зокрема плагіна Flash Player, який став уже майже стандартним для всіх браузерів, корпорація Adobe змушена була пристосовуватися до нових реалій. У її продукті для створення анімації Flash Professional з кожною новою версією з'являлося все більше засобів для взаємодії та сумісності з HTML 5, зокрема й можливість публікації анімаційних роликів як документів HTML 5 Canvas. А вже у лютому 2016 року програма Flash Professional змінила назву на Animate.

Однак принципи і ключові методи створення анімації в Adobe Animate залишилися тими самими: вони ґрунтуються на технології ключових кадрів, а в

їхню основу покладено векторний морфінг – плавне "перетікання" одного ключового кадра в інший, що дає змогу формувати доволі складні сцени, задаючи всього лише кілька ключових кадрів для кожного анімованого об'єкта. Отож, поки назва нової анімаційної програми не стала широко відомою, за традицією називатимемо цей спосіб створення анімації *технологією Flash*.

2.4.1.1. Часова шкала, сцена і символи Flash

Для організації анімації у Flash (рис. 2.8) використовують *часову шкалу* (*timeline*) – графічне зображення послідовності кадрів, подібне до часової шкали в застосунках для редагування відео. Анімацію можна формувати з окремих кадрів, послідовно розташовуючи ключові кадри на часовій шкалі, або використовувати інтерполяційні методи.

Сцена (*scene*) у Flash – це підвікно, у якому кадри формують, малюючи або імпортуючи об'єкти. Для організації елементів кадру використовують шари.

Графічні об'єкти, створені чи імпортовані у Flash, можна записувати в бібліотеку у спеціальній формі так званих *символів*, що дає змогу повторно їх використовувати. На сцені можна розміщувати багато *екземплярів символу*. У Flash є три основні види символів. *Графічні символи* (Graphic) – це просто векторні об'єкти з можливістю повторного використання. Спеціальним типом символів є *кнопкові символи* (Button), які використовують для інтерактивної взаємодії з роликами Flash. *Символи кліпу* (Movie clip) – це незалежні анімації з власними часовими шкалами, які відтворюються на тлі основного кліпу.

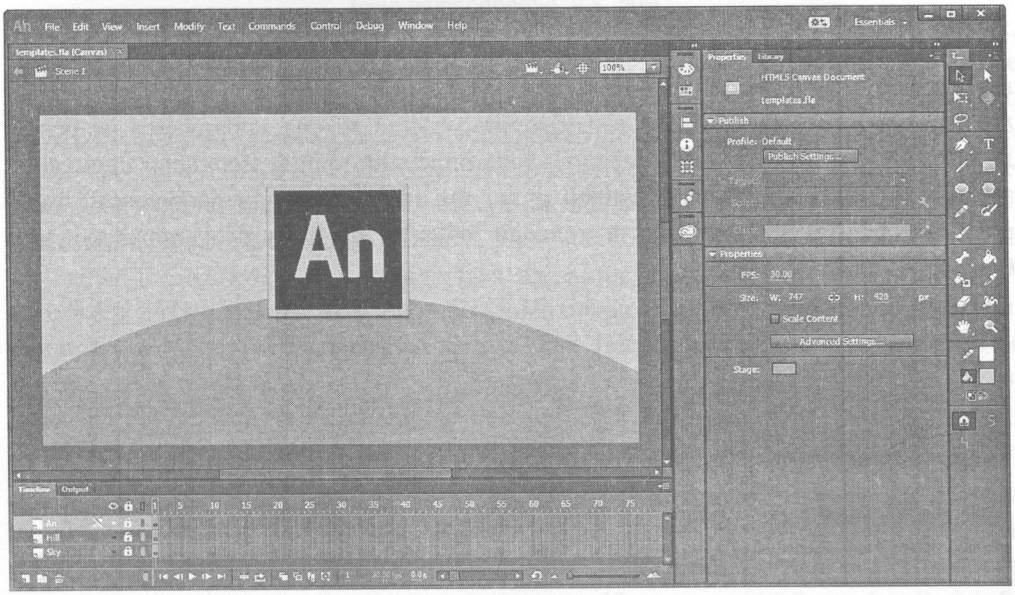


Рис. 2.8. Робоче вікно програми Adobe Animate CC

2.4.1.2. Побудова проміжних зображень

Анімацію можна створювати як послідовність сегментів, для яких між упорядкованими вручну ключовими кадрами програма автоматично будує проміжні зображення за допомогою інтерполяції. Цей процес у Flash одержав назву *tweening*, і термін виявився таким вдалим, що його тепер використовують і в інших програмних засобах.

Побудову проміжних зображень між ключовими кадрами, що містять той самий об'єкт, лише зі зміненими положенням у просторі сцени, розміром та деякими іншими параметрами, називають *інтерполяцією руху* (*motion tween*, рис. 2.9). Побудову проміжних зображень можна застосовувати до різних шарів, розміщаючи ключові кадри в різних місцях і при цьому застосовуючи незалежну анімацію до багатьох символів.

Рис. 2.9. Інтерполяція руху

Щоби ще більше спростити анімацію руху для її розробника, об'єкт можна прив'язати до лінії, намальованої на невидимому рівні. За допомогою такої *траєкторії руху* криволінійне переміщення об'єкта можна побудувати на основі лише двох ключових кадрів. Нарешті, хоча описаний процес побудови проміжних зображень називають інтерполяцією руху, так само можна інтерполювати зміну розміру, орієнтації, прозорості й кольорів об'єкта, причому все одночасно чи в будь-яких комбінаціях.

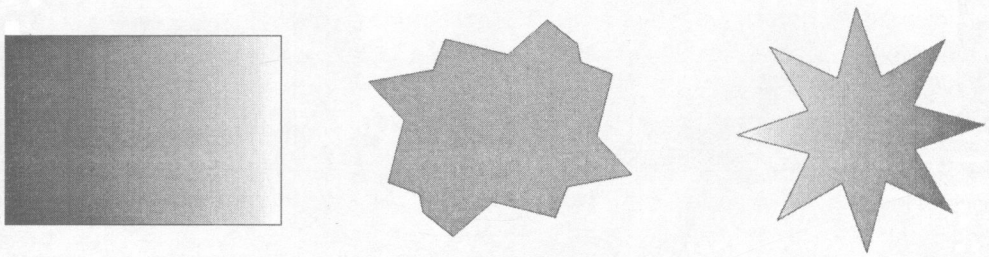


Рис. 2.10. Інтерполяція форми (shape tween)

Крім інтерполяції руху, Flash підтримує генерування проміжних зображень для трансформації (зміни форми) об'єкта – *інтерполяцію форми* (*shape tween*). Трансформація є різновидом інтерполяції, коли між ключовими кадрами перетворюються форми графічних об'єктів: наприклад, квадрат поступово переходить у зірку (рис. 2.10), багатокутник згладжується до овалу тощо.

2.4.1.3. Організація взаємодії з користувачем

Flash-анімація має потужні можливості для інтерактивної взаємодії з користувачем. Вона підтримує потужну мову підготовки сценаріїв Action Script, що дає змогу інтерактивно взаємодіяти з анімаційним продуктом і створювати веб-застосунки з основаними на мультимедіа користувацькими інтерфейсами, розробленими у Flash. За допомогою сценаріїв можна створювати анімацію, використовуючи для задання позицій об'єктів кліпу спеціальні алгоритми. Замість формування руху вручну можна просто зазначити, що об'єкти рухаються відповідно до законів фізики або деяких математичних алгоритмів, наприклад, тих, які визначають політ зграї птахів як групи.

На сцені можна розміщувати кнопки та інші інтерактивні елементи і визначати їхню реакцію. Це дає можливість створювати ділянки сцени, які "відгукуються" на клацання мишею, наведення вказівника та інші дії. У такий спосіб можна запрограмовувати відповідь будь-яких елементів кліпу на будь-які дії користувача.

2.4.2. Веб-анімація у HTML 5

До HTML 5 анімацію, призначену для розміщення на веб-сторінках, створювали переважно з використанням технології Flash і власне програми Adobe Flash, щоб потім відтворювати за допомогою плагіна Adobe Flash Player. З появою HTML 5 стало можливим створювати анімацію іншим способом. Адже HTML 5 – це не просто розширення і вдосконалення мови розмітки гіпертексту, а нова відкрита платформа, призначена для створення веб-застосунків, що використовують аудіо, відео, графіку, анімацію тощо.

Веб-анімація HTML 5 ґрунтується на застосуванні нового тегу **<canvas>** (полотно), за допомогою якого на веб-сторінках можна малювати векторні фігури, вводити текст, вставляти растрові зображення тощо – однак усе це працює лише через відповідні сценарії (скрипти) мовою JavaScript. Цими самими програмними засобами організовують і анімацію розміщених на сторінці об'єктів.

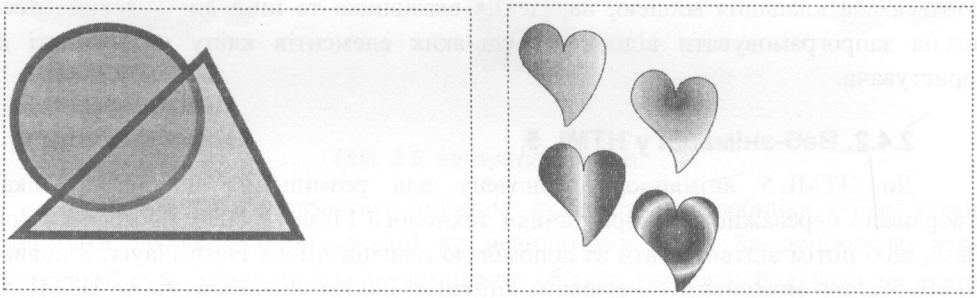
2.4.2.1. Базові можливості полотна

Найважливішим новим інструментом для веб-застосунків HTML 5 є *полотно* (canvas) – поверхня для малювання, на якій розробник може створювати власні векторні, текстові, растрові об'єкти. Полотно суттєво відрізняється від інших елементів HTML тим, що для роботи з ним потрібен програмний код JavaScript, бо іншого способу для креслення фігур або імпортування зображень немає.

Тег **<canvas>** надає розробнику робочий простір для малювання і має три атрибути – **id** (однозначне ім'я, необхідне для ідентифікації полотна кодом JavaScript), **width** (ширина) і **height** (висота полотна у пікселях). Щоб малювати на полотні, потрібно у відповідному скрипті отримати об'єкт полотна (яких може бути багато) за методом **document.getElementById**, а потім отримати двовимірний контекст малювання методом **getContext**.

Найпростішою фігурою, яку можна намалювати на полотні, є пряма. Для цього потрібно виконати три дії з контекстом: спершу треба задати початкову точку лінії за допомогою методу **moveTo()**, потім за допомогою методу **lineTo()** задати кінцеву точку лінії, і, нарешті, за методом **stroke()** власне намалювати лінію. Аналогічними способами можна малювати шляхи (ламані лінії) і замкнуті полігональні фігури, прямокутники, дуги і кола, криві лінії, зокрема криві Безьє. Перед викликом методу **stroke()** потрібно налаштувати властивості ліній, контурів і заповнення фігур (рис. 2.11).

Корисним для розміщення фігур у різних місцях полотна є механізм трансформацій – зміни положення, орієнтації та інших властивостей системи координат, близький до того, який використовують у графічній бібліотеці OpenGL.



а б

Рис. 2.11. Малювання на полотні: найпростіші фігури з прозорістю (а) та криволінійні фігури з градієнтним заповненням (б)

На полотні можна розміщувати зображення з файла за методом **drawImage()**, у параметрах якого слід задати об'єкт зображення і координати полотна, куди це зображення треба вставити. За допомогою необов'язкових параметрів метод **drawImage()** дає також змогу змінювати розміри розміщуваного на полотні зображення, а, крім того, вставляти у полотно лише частину зображення. Обертання, скошування та інші перетворення зображень можна реалізувати через трансформації.

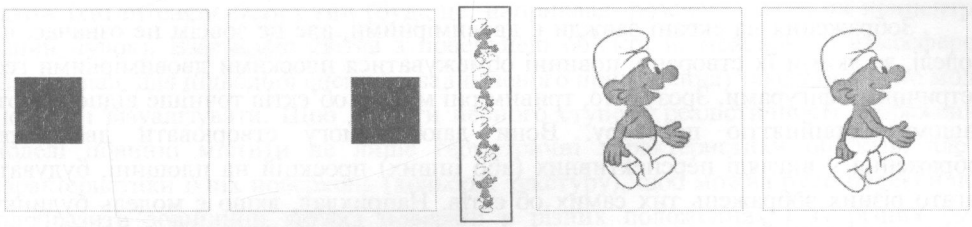
2.4.2.2. Анімація засобами HTML 5

Анімація на полотні HTML 5, як і будь-яка інша, складається з набору кадрів, які потрібно відображати з достатньою для створення ефекту руху швидкістю.

Щоб сформувати на полотні перший кадр, необхідно очистити його (наприклад, за допомогою методу **clearRect()**) і намалювати об'єкти анімації у початковому стані. Перед створенням чергового кадру полотно також здебільшого очищають, хоча й не завжди (наприклад, якщо необхідно поступово вимальовувати якусь лінію чи інші об'єкти).

Реалізувати анімацію на полотні HTML 5 доволі просто: для цього встановлюють таймер, який постійно викликає функцію оновлення кадру, зазвичай 30–40 разів за секунду, і за кожним викликом код повинен оновлювати вміст полотна. Якщо код написано правильно, постійно змінювані кадри зіллються у плавну анімацію. JavaScript надає дві функції для керування циклічним оновленням вмісту полотна – **setInterval()** і **setTimeout()**.

Найпростіша анімація руху на полотні HTML 5 – квадрат, що рухається горизонтально від лівого до правого краю полотна і потім знову починає рух спочатку (рис. 2.12, *а*).



а *б* *в*

Рис. 2.12. Анімація на полотні HTML 5: рухомий квадрат (а); зображення спрайта (б) і його анімація (в)

Для створення складніших анімацій на основі HTML 5 можна використовувати спрайти. Тут під спрайтом розумітимемо одне зображення, яке містить набір граней – усіх кадрів анімації об'єкта. За допомогою описаного раніше методу контексту **drawImage()** можна розміщувати на полотні спрайт-зображення і, циклічно закриваючи непотрібні частини, створювати ілюзію безперервної анімації. До прикладу, спрайт, що містить чотири кадри (рис. 2.12, *б*), за допомогою нескладної функції, що викликає метод **drawImage()**, можна "оживити" (рис. 2.12, *в*).

Найзначніший недолік такого "ручного" способу анімації на полотні HTML 5 полягає в необхідності виконувати всі обчислення самому розробнику. Наприклад, якщо потрібно, щоб зображення рухалося від одного боку полотна до іншого, розробнику треба розрахувати координати об'єктів кожного кадру, а потім намалювати їх у відповідній позиції. Якщо ж треба анімувати одночасно кілька об'єктів різними способами, обсяг і складність необхідних для цього обчислень можуть дуже швидко вийти з-під контролю.

Набагато легше працювати, використовуючи програми-редактори на кшталт Adobe Flash (Animate), що дають змогу створювати HTML5-анімацію за допомогою командно-візуальних засобів. Однак експорт готового проекту в HTML5-документ поки що породжує чимало проблем: замість одного файла – набір файлів різних типів (зображення, мультимедійні ресурси, сценарії JavaScript); неоптимальний чи неефективний код; проблеми з інтерактивними елементами; відсутність підтримки деяких ефектів, доступних у редакторі; труднощі в експорті кліпів, що складаються з кількох сцен, тощо.

Можливо, вже найближчим часом буде створено високорівневу систему анімації на полотні, орієнтовану безпосередньо на одержання ефективного продукту власне як документа HTML 5. Проте сьогодні найбільших результатів можна досягти за допомогою розвиненої уяви, не дуже важких, але часом доволі громіздких математичних розрахунків та написання вручну програмного коду JavaScript.

2.5. Тривимірна графіка та анімація

Зображення на екрані завжди є двовимірними, але це зовсім не означає, що моделі, за якими їх створено, повинні обмежуватися плоскими двовимірними геометричними фігурами. Зрозуміло, тривимірні моделі об'єктів точніше відповідають нашому сприйняттю простору. Вони дають змогу створювати двовимірні зображення у вигляді перспективних (або інших) проекцій на площині, будувати багато різних зображень тих самих об'єктів. Наприклад, якщо є модель будинку, можна одержати його вигляд спереду і ззаду, з обох боків, здалеку і з близької відстані, зверху й т.д., використовуючи при цьому одну-єдину модель (рис. 2.13).

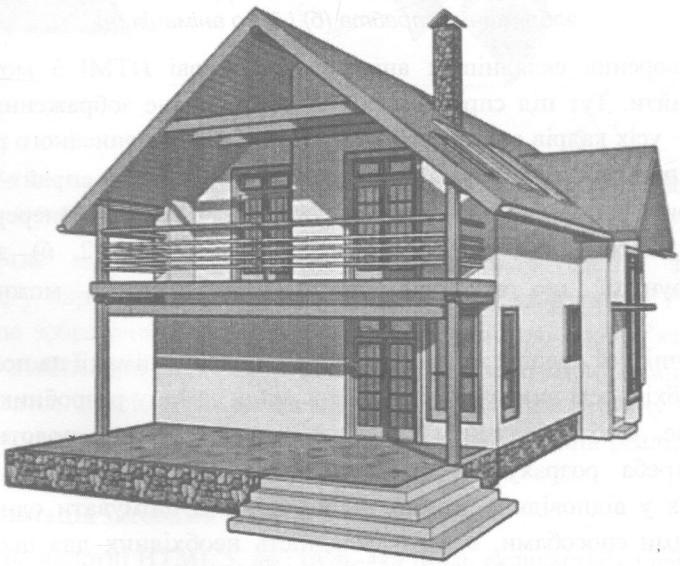


Рис. 2.13. Тривимірна модель будинку

2.5.1. Двовимірна і тривимірна графіка

Користуючись абстрактними математичними поняттями й узагальненнями, можна безпосередньо перейти від двовимірної координатної геометрії до тривимірної. Тривимірний вектор задає зсув у просторі так само, як двовимірний вектор на площині, і це дає змогу описувати просторові перетворення об'єктів.

Складнощі тривимірної графіки зумовлені не просто додаванням третього виміру, зайвими координатою чи двома поворотами. У випадку тривимірної графіки візуалізація об'єктів уже не буде порівняно простим процесом, пов'язаним із формуванням пікселів відповідно до математичного опису. У тривимірній графіці подано математичні моделі об'єктів у просторі, а на екрані потрібно відобразити плоский малюнок. Щоб одержати проекцію тривимірної моделі на плоскій поверхні, потрібно брати до уваги точку спостереження (або точку, в якій перебуває уявна камера), причому зі зміною відстані до камери масштаб малюнка повинен змінюватися так, як це відбувається насправді.

Крім того, потрібно враховувати освітлення – розташування джерел світла, а також їхні інтенсивність і тип (буде це, наприклад, розсіяне світло чи концентрований пучок). Взаємодію світла з поверхнею об'єкта й, можливо, з атмосферою (наприклад, для підводної сцени чи задимленого приміщення) також потрібно моделювати і візуалізувати. Щоб досягти певного ступеня реалістичності зображення, моделі повинні містити не лише геометричні характеристики об'єктів, але й характеристики їхніх поверхонь (кольори, текстуру), щоб можна було переконливо відобразити зовнішній вигляд поверхні в різних положеннях і за різних умов освітлення.

Ці додаткові труднощі означають, що теорія тривимірної графіки значно складніша, ніж у двовимірному випадку, а працювати з відповідним програмним забезпеченням набагато важче.

2.5.2. Тривимірна анімація

Якщо просту тривимірну анімацію (наприклад, обертальні логотипи чи глобуси) справді можна зробити дуже просто, високоякісна фотореалістична анімація, подібна до використовуваної в телерекламі, музичних відеокліпах і фільмах зі спецефектами, потребує величезних ресурсів – часу, потужності процесора і пам'яті, спеціалізованого програмного забезпечення, але насамперед – висококваліфікованого персоналу.

2.5.2.1. Комп'ютерна тривимірна анімація

Властивості тривимірних моделей визначені числовими величинами, і зміна цих величин призводить до зміни таких властивостей, як розташування об'єкта у просторі, його орієнтація, характеристики поверхні й навіть форма. Інтенсивність і напрямок джерел світла, а також положення й орієнтація камери теж задано числово. Тому, щоб анімувати тривимірну сцену, потрібно лише задати вихідну сцену, відобразити її як перший кадр анімації, далі, змінивши параметри, візуа-

лізувати наступний кадр і т.д. Оскільки змінюються числові значення, можлива інтерполяція ключових кадрів; для організації анімації зручно використовувати часову шкалу; крім того, для опису руху можна застосовувати тривимірні траєкторії (часто у вигляді тривимірних сплайнів Безьє). Можна також переміщувати камеру, змушуючи її "літати" над сценою або повертатися навколо певних об'єктів.

Для створення комп'ютерної тривимірної анімації розроблено вже чимало програмних засобів, кращими з яких є, зокрема, безкоштовний пакет Blender, програма розроблення 3D-графіки для кіно і телебачення Autodesk Maya та повнофункціональна професійна програмна система для створення і редагування тривимірної графіки й анімації 3D Studio MAX (або просто 3ds MAX) тієї самої компанії Autodesk.

2.5.2.2. Реалістична тривимірна анімація

Є кілька факторів, що роблять реалістичну тривимірну анімацію складнішою, ніж вона здається спочатку. Перший – більшість людей мають труднощі з наочним відображенням тривимірних сцен. Якщо ще додати час, одержимо чотири виміри, з якими потрібно працювати за допомогою двовимірного комп'ютерного екрана. Цю складність посилює інша проблема – велика обчислювальна потужність, необхідна для візуалізації реалістичної анімації.

На найвищому рівні тривимірної комп'ютерної анімації розробнику надають потужний інтерфейс, який дає повний контроль над рухом: іноді навіть використовують скафандри, обладнані датчиками руху, які застосовують для керування анімованими маріонетками. У підсумку можна отримати анімовані об'єкти і персонажі, вживлені у реальне середовище настільки, що їх буває навіть важко ідентифікувати (рис. 2.14).

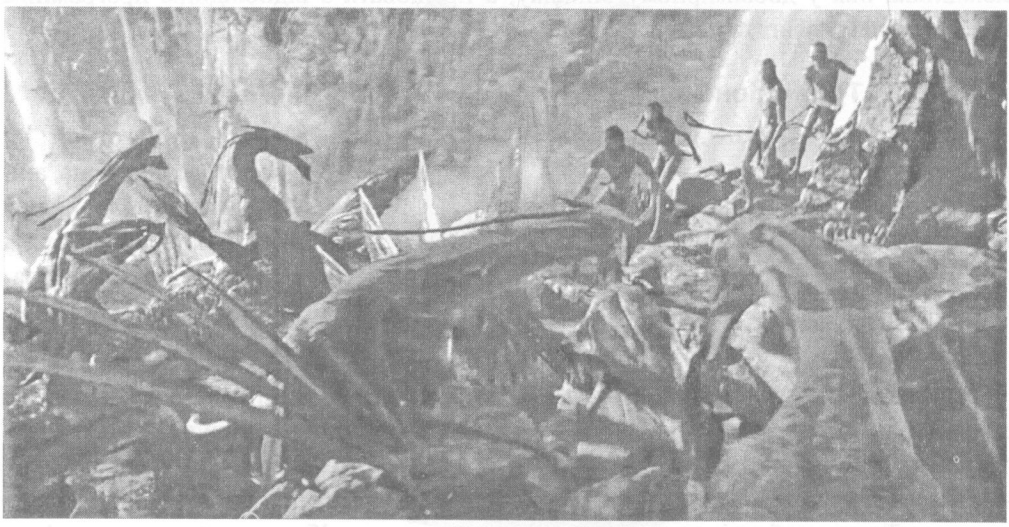


Рис. 2.14. Високоякісна реалістична анімація у фільмі "Аватар"

2.6. Віртуальна реальність

Зазвичай фразу "віртуальна реальність" використовують для опису максимально реалістичного ступеня сприйняття штучного світу. За допомогою встановлених на голові дисплеїв, чутливих до рухів голови, зображення проектують в очі користувача. Потім ці зображення модифікуються під час переміщення голови, так що користувачеві здається, начебто він перебуває усередині тривимірного світу й оглядається в ньому. "Інформаційні рукавички" відстежують рухи рук, даючи змогу дисплею відображати кисті користувача. Тактильні інтерфейси забезпечують дотиковий зворотний зв'язок, так що користувачі можуть торкати й відчувати об'єкти у віртуальному світі.

Висока вартість апаратури інтерфейсу, необхідної для створення глибокої віртуальної реальності, поки що обмежує широке застосування цієї технології переважно льотними і промисловими тренажерами та спеціальними іграми. Однак спроби створити порівняно недороге обладнання для глибокої віртуальної реальності тривають постійно.

Якщо відмовитися від надмірного і дорогого реалізму у сприйнятті віртуальної реальності, то дві її технології можна більш-менш вдало запровадити у Всесвітній павутині. Можна сподіватися, що з підвищенням потужності комп'ютерів ці технології можуть стати важливішими й більш відповідними до очікувань. Їх буде описано нижче, а одну з практичних реалізацій такого типу віртуальної реальності – віртуальні мандрівки – розглянуто у відповідному підрозділі розділу 6.

2.6.1. Мова VRML та формат X3D

Мову моделювання віртуальної реальності (Virtual Reality Modeling Language – VRML) розроблено на хвилі ентузіазму з приводу віртуальної реальності у Всесвітній павутині в 1994 році для створення механізму поширення віртуального світу в Інтернеті з використанням веб-браузерів.

VRML дозволяла специфікацію об'єктів через їхню геометрію і матеріал, з якого їх зроблено, давала можливість накладати на поверхню текстуру, а самі об'єкти переміщувати у тривимірному просторі за допомогою геометричних перетворень, освітлювати сцену. Після того, як файл VRML завантажено у придатний для візуалізації браузер, користувач міг досліджувати описуваний цим файлом тривимірний світ, начебто сам переміщується у ньому.

VRML досягла вершини популярності після виходу в 1997 році VRML 2.0, однак надалі стала поступово занепадати. Хоча саму мову VRML ще продовжують використовувати у деяких галузях, особливо в освітньому та дослідницькому середовищі, де найбільше цінують відкриті специфікації, можна сказати, що сьогодні її витіснив формат X3D.

X3D – це стандарт ISO, призначений для роботи з тривимірною графікою в реальному часі, розширення VRML, що передбачає анімацію гуманоїдних персонажів, NURBS, GeoVRML та інші засоби, дає змогу кодувати сцену згідно із синтаксисом XML, а також використовувати розширений інтерфейс прикладного програмування API.

Особливостями X3D є інтеграція з XML, компонентність, розширюваність, еволюційність (X3D сумісний з VRML97), масштабованість (від суперкомп'ютерів до мобільних телефонів), робота в реальному часі та добра стандартизованість. X3D дає змогу використовувати потужні засоби керування анімацією та морфінгом, складну програмовану плоску та тривимірну графіку, поверхневе аудіо та відео, спроектовані на геометрію сцени; організовувати складну взаємодію з користувачем за допомогою миші, його переміщення по 3D-сцені, динамічну зміну сцен за допомогою програм мовами опису сценаріїв; складати одну X3D-сцену з матеріалів, розташованих у мережі, пов'язувати об'єкти з різних сцен гіперпосиланнями, симулювати фізичні явища і комунікацію в реальному часі.

2.6.2. Віртуальна реальність QuickTime

QuickTime – технологія, розроблена корпорацією Apple для відтворення цифрового відео, звуку, тексту, анімації, музики і панорамних зображень, збережених у різних форматах.

Віртуальна реальність QuickTime (QuickTime Virtual Reality – QTVR) належить до початкового рівня віртуального світу. Є два типи роликів QTVR: *панорамні картини* та *кліпи з об'єктами*. Перші надають можливість огляду сцени на 360°: використовуючи мишу, користувач може перетягувати сцену (інтер'єр кімнати або долину, оточену горами), наче оглядаючись. Кліпи з об'єктами, навпаки, дають змогу користувачу досліджувати (також за допомогою миші) об'єкт із різних боків, ніби ходячи довкола нього. Ролики QTVR будь-якого типу можуть містити *гарячі точки* – активні ділянки, що містять посилання на інші ролики. Основне призначення таких точок – створення дверей, клацаючи на яких, користувач переходить в іншу кімнату. Оскільки QTVR є частиною QuickTime, панорамні картини і кліпи з об'єктами можна поєднувати з аудіо та відео.

QTVR і X3D надто недосконалі з огляду на людські уявлення про глибоку віртуальну реальність, але їхніми перевагами є реалізованість без спеціальних пристроїв чи потужних робочих станцій. Вони пропонують нові підходи до поєднання віртуальної реальності з мультимедіа, основані на організації різних видів і засобів інформації у тривимірному просторі.

2.7. Контрольні питання

1. Що таке анімація і на чому ґрунтується її сприйняття?
2. Назвіть та охарактеризуйте загальні види анімації.
3. Які основні технології традиційної анімації?
4. Які технології використовують сьогодні у цифровій анімації?
5. Що таке ключові та проміжні кадри у традиційній і цифровій анімації?
6. Опишіть ключові елементи технології Flash.
7. Що спільного та відмінного в інтерполяції руху та форми у Flash?
8. Опишіть загальну схему створення веб-анімації засобами HTML 5.
9. Які основні принципи розроблення комп'ютерної тривимірної анімації?
10. У чому полягають особливості створення реалістичної тривимірної анімації?
11. Що таке віртуальна реальність і які засоби її створення?
12. Опишіть основні веб-технології віртуальної реальності.

2.8. Приклади тестових питань

1. Поняття анімації:

   а) це спосіб генерації рухомих зображень із послідовності нерухомих;
   б) це набір технологій роботи з візуальним матеріалом;
   в) у ньому всі кадри створюють за допомогою неперервного знімання.

2. Поняття процедурної анімації:

   а) вона генерує послідовності кадрів у діалоговому режимі;
   б) у ній кожного разу результати можуть бути трохи інші;
   в) у ній аніматор задає ключові кадри та спосіб інтерполяції.

3. Поняття GIF-анімації:

   а) для її створення використовують метод покадрової анімації;
   б) для її створення використовують технологію ключових кадрів;
   в) це найпростіший різновид процедурної анімації.

4. Поняття стандарту X3D:

   в) це стандарт для розроблення інтерфейсів та застосунків у XML-базованих технологіях;
   б) це стандарт для створення інтерактивних мультимедійних застосунків;
   а) його використовують для моделювання віртуальної реальності.

5. Полотно HTML 5:

   в) це прямокутна поверхня для малювання на веб-сторінці, може бути тільки одна;
   б) це прямокутна поверхня для малювання на веб-сторінці, може бути не одна;

а) це поверхня довільної форми для малювання на веб-сторінці, може бути тільки одна.

6. Загальні види анімації:

а) графічна;
б) плоска;
в) інтерполяційна.

7. Основні технології традиційної анімації:

а) пошарова анімація;
б) растрова анімація;
в) технологія проміжних кадрів;
г) технологія інтерполяції.

8. Типові технології створення цифрової анімації:

а) графіка руху;
б) записування руху;
в) комп'ютерна анімація;
г) об'ємна анімація.

9. Сфери застосування процедурної анімації:

а) записування руху;
б) захоплення рухів;
в) створення ключових кадрів;
г) моделювання поведінки систем часток.

10. Символи Flash можуть бути:

а) векторні;
б) кнопкові;
в) фізичні;
г) логічні.

11. Властивості інтерполяції руху у Flash:

а) її об'єкт може змінювати форму;
б) її об'єкт може бути растровим;
в) її об'єкт не може бути символом;
г) інтерполяція руху у Flash є квадратичною.

12. Веб-технології віртуальної реальності:

а) DVR;
б) WAVR;
в) SWRF;
г) VRML.

Розділ 3

ЦИФРОВЕ ВІДЕО

У попередньому розділі розглянуто перший із двох способів генерації рухомих зображень – анімацію, коли всі кадри створюють окремо як нерухомі зображення, а потім послідовно їх відтворюють – "анімують", тобто "оживляють". Другий спосіб – за допомогою відеокамери записати послідовність кадрів реального руху в реальному світі для подальшого відтворення на екрані телевізора чи дисплеї комп'ютера. Для цього випадку використовують термін *відео*.

3.1. Аналогове відео та телебачення і перехід до цифрової форми

Штучно створені двовимірні зображення в русі демонстрували ще у 1860-ті роки за допомогою оптичних приладів, які могли показувати послідовність нерухомих картинок з достатньою для створення ілюзії руху швидкістю. З використанням для фотографії целулоїдної плівки стало можливим захоплювати об'єкти в русі у режимі реального часу. У 1880-ті роки кінокамери вже давали можливість зберігати окремі зображення на одній плівці у рулонах, які стали називати "рухомими зображеннями", звідки й походить англійське слово "movie". Так було започатковано кінематограф.

Кінематограф (від гр. κινεμα – рух та γραφο – писати, зображати) – галузь культури та економіки, що об'єднує всі види професійної діяльності, пов'язаної з виробництвом, поширенням, зберіганням та демонструванням фільмів.

Фільм (кінофільм, кіно) – аудіовізуальний твір кінематографії, що складається з епізодів, поєднаних між собою творчим задумом і зображувальними засобами; у технологічному аспекті – сукупність пов'язаних єдиним сюжетом рухомих зображень (монтажних кадрів), що складаються з послідовності фотографічних або цифрових нерухомих зображень, на яких зафіксовано окремі фази руху.

Спочатку кіно мало лише візуальну компоненту, і тільки від кінця 1920-х років стало можливим додавати до кожного фільму звукову доріжку. Іншим важливим кроком у розвитку кіно було введення кольору: хоча технологічно це

вміли робити ще перед уведенням звуку, однак лише наприкінці 1960-х колір став нормою для всіх кіновиробників.

Іншим шляхом ішло телебачення.

Телебачення (від гр. τήλε – далеко і "бачити") – загальний термін, що охоплює всі аспекти технології та практичної діяльності, пов'язані з передаванням зображень зі звуковим супроводом на далекі відстані.

Уже на ранніх стадіях свого розвитку воно використовувало поєднання оптичних, механічних та електронних технологій для отримання, передавання та відображення візуальних зображень. Ідея використовувати для цього сканування була фактично впроваджена у практику в 1881 році в пантелеграфах, і відтоді сканування використовують майже в кожній телевізійній технології, враховуючи сучасні телевізори.

Подальший розвиток аналогового телебачення в епоху всезагальної комп'ютеризації врешті-решт призвів до появи *цифрового телебачення*.

Цифрове телебачення (англ. Digital Television, DTV) – телевізійна технологія, в якій передавання, обробка та зберігання телевізійного сигналу відбувається у цифровій формі.

Цифрове телебачення впродовж тривалого часу витісняло аналогове, і сьогодні більшість розвинутих країн повністю відмовились від передавання аналогового телевізійного сигналу. В Україні від середини 2018 року повністю відключено аналогове мовлення із заміною його на цифрове.

Наприкінці XX століття дуже популярним стало *стереоскопічне відео*, або *3D-відео*. По всьому світу почали масово з'являтися кінотеатри, які за допомогою тієї чи іншої технології демонстрували "об'ємні" фільми. Для стереовідео потрібно два відеоканали: один для лівого ока, інший для правого: у такий спосіб у глядача виникає відчуття об'ємності, тривимірності відеоматеріалу, підвищується реалістичність відчуття перегляду.

3.2. Стандарти і формати відео

Цифрове відео записують за допомогою пристроїв, які також використовують для одержання відеозображень із метою їхнього відтворення на екранах телевізорів, тому під час виробництва мультимедіа потрібно працювати із сигналами, які відповідають стандартним сигналам керування телевізором і можуть взаємодіяти з устаткуванням, що задовольняє вимоги і стандарти *широкомовного телебачення*.

Широкомовлення (англ. broadcasting) – метод передавання даних у мережах соціального призначення (радіомовлення, телебачення тощо) та комп'ютерних мережах, у якому потік передаваних даних (кожен пакет у випадку пакетного передавання) призначений для приймання всіма учасниками мережі.

Новітні цифрові пристрої повинні бути сумісними зі старим аналоговим обладнанням за такими важливими параметрами, як розмір і частота зміни кадрів, тому для того, щоб добре зрозуміти цифрове відео, спочатку потрібно ознайомитися з його аналоговими варіантами.

3.2.1. Стандарти аналогового широкомовлення

До переходу на цифрове телебачення існувало (і ще дотепер існує) три набори стандартів, які застосовували в аналоговому кольоровому широкомовному телебаченні. Найстарішим серед них є *NTSC* (National Television System Committee) – його використовували у Північній Америці, Японії, Тайвані і більшості країн Латинської Америки. У Західній Європі застосовували стандарт *PAL* (Phase Alternating Line), а у Франції – *SECAM* (Sequential Couleur avec Memoire – колірна система із запам'ятовуванням). Систему PAL використовували також в Австралії, Новій Зеландії та Китаї, а SECAM – у більшості країн колишнього СРСР.

Щоб не було миготіння, екран повинен оновлюватися близько 40 разів за секунду. Таке швидке передавання всього зображення потребує ширини смуги, визнаної недосяжною на момент розроблення стандартів. Замість цього кожен кадр ділили на два *напівкадри,* один із яких складається з непарних, а інший – із парних рядків кадру. Їх передавали почергово, так що кожний кадр (нерухоме зображення) побудований методом *чергування напівкадрів.* Такий спосіб формування відеозображення називають *черезрядковою розгорткою* на відміну від *прогресивної*, де кадр передають і відображають повністю.

Спочатку частоту передавання напівкадрів вибирали відповідно до частоти місцевої мережі змінного струму, тому в Західній Європі відповідно до стандарту PAL частота становила 50 напівкадрів (або 25 кадрів) за секунду; у Північній Америці відповідно до стандарту NTSC – близько 60 (точніше – 59,94) напівкадру за секунду.

Крім рядків, які стосуються зображення, у кожному кадрі додатково містяться рядки з іншою інформацією, зокрема, стосовно синхронізації зображення і звуку. Кадр NTSC складається з 525 рядків, 480 з яких належать до зображення; стандарти PAL і SECAM використовують 625 рядків, з яких 576 належать до зображення. Часто певний стандарт характеризують за допомогою кількості рядків і частоти зміни напівкадрів: наприклад, стандарт NTSC описують як 525/59,94.

3.2.2. Стандарти цифрового відео

Ситуація зі стандартами цифрового відео визначається потребою його сумісності з аналоговим. Через це повинні існувати формати цифрового відео 625/50 і 525/59,94. Крім того, потрібно враховувати перспективні стандарти телебачення високої чіткості HDTV (HighDefinition Television), що зараз розвиваються. Певні спроби уніфікувати наявні формати робилися, однак, на жаль, для користувацького і професійного застосування та передавання було прийняті різні цифрові стандарти.

Подібно до будь-яких аналогових даних для перетворення на цифрову форму відео потрібно дискретизувати. Дискретизацію аналогового відео визначає стандарт CCIR 601. Оскільки відеокадр двовимірний, його потрібно дискретизувати в обох напрямах. Дещо спрощуючи, можна вважати, що кадр NTSC, дискретизований відповідно до стандарту CCIR 601, матиме розміри 720×480 пікселів, а кадр PAL/SECAM – 720×576 пікселів.

Хоча для відображення на моніторах чи екранах телевізорів використовують колірну схему *RGB*, для цифрового відео прийнято іншу модель. Найдрібніший елемент (піксель) відео, дискретизованого відповідно до стандарту CCIR 601, складається з компонента світності та двох компонентів – різниць кольорів. Технічно простір кольорів визначений моделлю $Y'C_BC_R$, і можна вважати, що трьома компонентами є масштабовані світність Y і різниці B-Y та R-Y. На першому етапі зменшення розміру цифрового відео для кожного значення різниці кольорів беруть меншу кількість вибірок, ніж для світності – цей процес називають *субдискретизацією колірності*. Доцільність такого підходу можна пояснити емпіричним спостереженням, що людське око менш чутливе до змін кольорів, ніж до змін яскравості.

Є різні формати субдискретизації колірності, які позначають як співвідношення між трьома частинами X:a:b (наприклад, 4:2:2), що описують кількість вибірок світності і різниць кольорів: X – частота дискретизації світності Y'; a – кількість вибірок різницевих сигналів C_B та C_R у горизонтальному напрямку в першому рядку; b – кількість додаткових вибірок різницевих сигналів у другому рядку.

У стандарті CCIR 601 використано субдискретизацією 4:2:2: у кожному рядку вибірок Y удвічі більше, ніж вибірок B-Y і R-Y.

Дискретизація переводить відеосигнал з аналогової у цифрову форму. Для цього сигнал спочатку потрібно стиснути, а потім сформувати на його основі потік даних для передавання. Щоб задати алгоритм компресії та формат потоку даних, потрібні додаткові стандарти. Було розроблено два набори стандартів, основаних на CCIR 601, але з подальшою субдискретизацією.

- Цифрова відеоапаратура, призначена для користувацького та напівпрофесійного застосування, працює за стандартом цифрового відео **DV**.
- Студійне обладнання, цифрове широкомовне телебачення, DVD, Blu-ray тощо основані на **MPEG2**. Ранній стандарт MPEG1, який так і не став поширеним, є основою для наступних стандартів відео MPEG. Один із них, MPEG4, останнім часом стає одним із лідерів серед форматів потокового відео.

Зараз на зміну традиційним стандартам телебачення приходить *телебачення високої чіткості*.

Телебачення високої чіткості (англ. High-definition television, HDTV) – система трансляції цифрового телебачення з роздільністю, вищою за аналогові системи телебачення (NTSC, SECAM, PAL).

HDTV-передачі транслюють у цифровому форматі, оскільки цифрове телебачення потребує каналів із меншою пропускною спроможністю завдяки відповідному стисненню зображень.

Сьогодні у сфері HDTV найпоширеніші два міжнародні стандарти, перший із яких підтримує розмір зображення 1920×1080 пікселів з різною кадровою частотою, а другий – 1280×720 пікселів. Обидві системи розраховані на співвідношення сторін кадру 16:9 і мають позначення:

- *1080i*: черезрядковий стандарт із частотою 25, 29,97 або 30 кадрів за секунду;
- *1080p*: стандарт з прогресивною розгорткою, який допускає використання частот 24, 25, 30, 50 або 60 кадрів на секунду;
- *720p*: стандарт з прогресивною розгорткою, що допускає використання кадрових частот 50 або 60 кадрів за секунду.

3.2.3. Формати зберігання цифрового відео

Усі формати, які використовують для зберігання цифрового відео, є *мультимедійними контейнерами.*

Мультимедійний контейнер – формат, який може об'єднувати дані різних типів, стиснені різними кодеками, та зберігати аудіо, відео і текстову інформацію в єдиному файлі.

Кодек (англ. *codec* – скорочено від coder/decoder чи compressor/decompressor) – пристрій або програмне забезпечення, що виконує функції кодування та декодування цифрового потоку.

Формат *AVI* (Audio Video Interleave) ґрунтується на давнішому форматі RIFF, від якого він запозичив спосіб записування даних методом їхнього поділу на "шматочки". Файл із розширенням **.avi** може містити відео і аудіодані, стиснуті з використанням різних комбінацій кодеків, що дає змогу синхронно відтворювати відео зі звуком. AVI-файли підтримують багатопотокове аудіо та відео.

Формат *MOV*, що базується на технології *QuickTime*, дає змогу об'єднувати звук, текст, анімацію та відео в одному файлі з розширенням **.mov**. Файли цього формату можна використовувати для поширення відео в Інтернеті.

MP4 або *MPEG-4 Part 14* – формат медіаконтейнера, що є частиною стандарту MPEG-4. Як і більшість сучасних медіаконтейнерів, MP4 надає можливість здійснювати потокове мовлення через Інтернет, додатково до файла передаючи метадані певного стандарту, такі як час створення файла, тривалість, мовний код потоку, назва альбому, правовий статус тощо. Це один із трьох форматів відео, які відтворює стандартний плеєр HTML5.

Формат *MKV* (Matroska або Matrôşka – Матрьошка) є прямою відкритою альтернативою контейнерам AVI, MOV, MP4 та іншим. Файли цього формату з розширенням **.mkv** можуть містити велику кількість потоків аудіо, відео та субтитрів.

OGG (Ogg Theora) – вільний формат відео, що є частиною розробленого Фондом Xiph.Org проекту "Ogg" для інтеграції відеокодека Theora, аудіокодека Vorbis та мультимедійного контейнера Ogg в одне рішення на зразок MPEG-4. Це повністю відкритий, вільний з ліцензійного погляду мультимедіа-формат; його файли, що мають розширення **.ogv**, відтворюються у браузерах за допомогою стандартного плеєра HTML5.

WebM – вільний відкритий мультимедійний контейнер, призначений для доставки аудіовізуальних даних у мережі Інтернет у межах нового стандарту HTML5. Файл WebM складається з відеопотоків, стиснених відеокодеком VP8, та аудіопотоків, стиснених аудіокодеком Vorbis, у контейнері на основі MKV. Сьогодні ці файли вже можуть відтворювати всі найпопулярніші браузери як стандартне відео HTML5.

Багато сучасних мобільних телефонів мають функції запису і перегляду аудіо та відео у форматі *3GP*. У ньому для зберігання відео використано MPEG-4 або H.263. Готові відеоролики в 3GP мають невеликий розмір порівняно з іншими форматами, однак платою за це є доволі низька якість.

3.2.4. Технологія QuickTime

Отож, є багато різноманітних схем стискання відео та форматів даних: DV, MPEG1, MPEG2 і MPEG4, кілька різновидів MJPEG, нестиснуте відео тощо. Сподівання на розроблення універсального формату відеофайлів не виправдались. Більш прийнятний підхід до стандартизації міг би ґрунтуватися на архітектурному каркасі, визначеному на достатньо абстрактному рівні, щоб урахувати всю різноманітність конкретних подань відео. Було запропоновано кілька таких підходів, але де-факто стандартом став *QuickTime*.

QuickTime – технологія, яку розробила корпорація Apple для відтворення цифрового відео, звуку, тексту, анімації, музики і панорамних зображень, збережених у різних форматах.

Об'єкти, якими маніпулює QuickTime, називають *фільмами* (*movie*). Фільми можна вважати абстракцією відеоряду, хоча фактично вони можуть містити й інші види інформації. Сам фільм, по суті, є просто основою для організації даних, забезпечення доступу до них і маніпулювання даними, які насправді є кадрами відео й можуть зберігатися окремо від самого фільму.

QuickTime має власний формат (файлове розширення **.mov**), який дає можливість доволі гнучко зберігати відео та інші види інформації.

QuickTime також використовують для потокової обробки відео. Є також способи інтеграції сервера потокової обробки QuickTime із засобами записування, що дає змогу передавати через Internet "живе" відео.

3.3. Одержання цифрового відео

Коли йдеться про цифрове відео, потрібно завжди пам'ятати про розмір одержуваних даних. Для 24-бітового кольорового зображення системи PAL з розміром 768×576 пікселів і частотою відтворення 25 кадрів за секунду одержуємо 31 Мбайтів для запису однієї секунди і 1,85 Гбайтів для запису однієї хвилини відео. За таких цифр записування/відтворення відео на запам'ятовувальних пристроях є нереальним, також видається неможливим передавати відео через мережі. Отож, використання відео в такій формі неможливе для звичайних споживачів.

Великий обсяг кожного кадру пояснюється вимогами до растрових зображень. Однак цей обсяг можна зменшити за допомогою стискання кадрів. Далі не розглядатимемо можливість роботи з нестиснутим відео, оскільки сьогодні це реально лише на дуже потужному обладнанні.

3.3.1. Способи оцифрування відео

Цифрове відео можна записувати або безпосередньо за допомогою камери, або за допомогою відеомагнітофона, або із широкомовного телесигналу. Наявні технології дають змогу здійснювати оцифрування і стискання безпосередньо у камері або на комп'ютері.

Якщо оцифрування виконують на комп'ютері, аналоговий відеосигнал, узгоджений із певним стандартом широкомовного відео, надходить на вхід *плати оцифрування відеозображень,* підключеної до комп'ютера. У цій платі аналоговий сигнал перетворюється на цифрову форму.

Якщо ці процеси виконуються з використанням фізичних схем, вбудованих у камеру, суто цифровий сигнал (або *потік даних)* передається на комп'ютер. Сьогодні звичними є цифрові відеокамери, які приймають сигнал, перетворюють його на цифрову форму, обробляють і зберігають у самому апараті у вигляді файла. Більшості цифрових відеокамер використовують флеш-пам'ять, але є також камери, де використовують оптичні диски Blu-ray або інші сучасні носії інформації. Відео туди записують у форматах, основаних на MPEG-2.

Хоча суб'єктивна якість цифрового відео у звичайних відеоформатах достатньо висока, всі вони є стиснутими, а компресія може породжувати *артефакти* (помилки) і перешкоджати подальшій обробці та відновленню стиснутого зображення. Тому *цифрові кінокамери* – професійні відеокамери високої роздільності, призначені для знімання кінофільмів за безплівковою цифровою технологією, – надають можливість отримання відеоданих у нестиснутому форматі.

3.3.2. Кодеки

Цифрова відеоапаратура та плати оцифрування відеозображень зазвичай виконують не тільки оцифрування і стискання, але також і зворотні операції – відновлення та цифро-аналогове перетворення. Пристрої, що стискають і відновлюють сигнали, називають *кодеками.* Це же термін використовують і для відповідного програмного забезпечення.

Використовуючи *апаратний кодек* (кодек цифрової відеокамери або плату оцифрування), можна оцифрувати відеосигнали, записати їх на комп'ютері, а потім відтворити на зовнішньому моніторі (телевізорі), підключеному до аналогового виходу відеокамери або відеокарти.

Однак під час поширення мультимедійної продукції наперед невідомо, чи будуть у глядачів апаратні кодеки, а якщо й будуть, то які саме. Також зазвичай потрібно показувати відео на комп'ютерному моніторі. Тому для відтворення мультимедіа необхідний *програмний кодек* – програма, що виконує ті самі функції, що й спеціалізовані апаратні кодеки, і гарантує, що глядачі зможуть переглядати відео на звичайних комп'ютерних моніторах.

3.4. Принципи стискання цифрового відео

Усі алгоритми відеокомпресії працюють з оцифрованим відео, яке складається з послідовності растрових зображень. Є два підходи до зменшення розміру такої послідовності.

3.4.1. Просторова і часова компресія

Сьогодні розроблено дві загальні технології стискання відеозображень: або окремо стискати кожне нерухоме зображення, або записувати відмінності між послідовними кадрами. Їх зазвичай називають відповідно *просторовим* і *часовим* стисканням, або *внутрішньокадровим* (intra-frame) та *міжкадровим* (inter-frame). Треба зазначити, що цілком природно використовувати обидва підходи одночасно.

Просторова компресія – це насправді лише стискання зображень; у будь-якому разі вона дає надто малий ступінь стискання відеоданих, тому ключовим для цифрового відео є все ж компресія часова.

Зрозуміти принцип дії алгоритмів часового стискання просто. У відеоряді виділяють кадри послідовності, які називають *ключовими.* Часто ключові кадри вибирають із постійним інтервалом (наприклад, кожен шостий кадр). Такі ключові кадри або не стискають узагалі, або піддають тільки просторовому стисканню. Усі кадри між парою ключових заміняють на *різницеві кадри,* на яких записано тільки різницю між кадром, що раніше був на цьому місці, і попередніми кадром або ключовим кадром. Для більшості послідовностей різниця буде ненульовою лише на маленькій частині зображення (рис. 3.1). Отже, всі різницеві кадри міститимуть набагато менше інформації, ніж повний кадр, і її можна подати набагато компактніше, ніж інформацію про весь кадр.

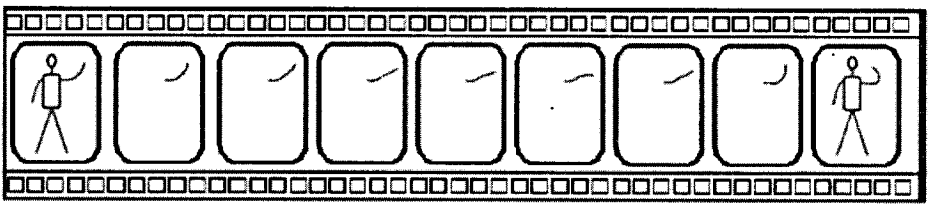


Рис. 3.1. Ключові (по краях) та різницеві кадри

3.4.2. Основні схеми стискання відео

Деякі схеми стискання відео основані подібно до компресії зображень JPEG, на використанні дискретного косинус-перетворення. Це прямолінійний спосіб застосування схеми JPEG до кожного кадру без часового стискання.

3.4.2.1. MJPEG

Технологію просторового стискання відеопослідовностей за допомогою застосування компресії JPEG до кожного кадру називають *Motion JPEG* (MJPEG). MJPEG використовують у більшості аналогових плат записування відео, у яких реалізовано спеціалізоване апаратне забезпечення, що дає змогу достатньо швидко (щоб встигати за вхідним відеосигналом) виконувати складні обчислення алгоритму JPEG. Утім, треба зазначити, що внаслідок розвитку цифрових форматів, що приходять на зміну аналоговому відео, стискання MJPEG застосовують усе рідше.

MJPEG і сьогодні широко застосовують у деяких галузях, зокрема для нелінійного відеомонтажу, в IP-камерах, непрофесійних відеокамерах та системах відеоспостереження.

3.4.2.2. DV

У просторовій технології стискання DV також використовують дискретне косинус-перетворення із подальшим квантуванням, що зменшує обсяг даних у відеопотоці.

Через деякі особливості цієї схеми стискання у разі постійної швидкості передавання 25 Мбітів/с (яку використовують у більшості форматів сімейства DV) отримують якісніше зображення, ніж для цієї самої швидкості може дати MJPEG.

Втім, треба зазначити, що формати DV розробляли для систем відеозапису побутового призначення на відеокасети. Для обміну відеоданими між пристроями формату DV та комп'ютерами використовують цифровий інтерфейс Digital DV. Відео з плівкових пристроїв передається потоком, на комп'ютері його захоплює спеціальна програма і зберігає в контейнер (зазвичай AVI).

3.4.2.3. MPEG

У 1988 році Міжнародна організація стандартизації (ISO) організувала спеціальну групу – Motion Pictures Expert Group (MPEG). Перед нею поставили завдання розробити методи стиснення і відновлення цифрового відеосигналу в межах стандарту, що дає змогу об'єднати потоки відео-, аудіо- та іншої цифрової інформації. Результатом багаторічних досліджень у галузі цифрового кодування сигналів зображення і звуку стало створення сімейства міжнародних стандартів MPEG для компресії цифрового відео. Усі стандарти MPEG ґрунтуються на базовому стандарті цифрового відео CCIR-601.

Група MPEG уніфікувала такі стандарти стиснення і допоміжні стандарти:

- MPEG-1 – вихідний стандарт відео й аудіокомпресії, пізніше застосований як стандарт для Video CD, також містить популярний формат MP3;
- MPEG-2 – транспортні, відео- і аудіостандарти для широкомовного телебачення, які також використовують у цифровому телебаченні, цифрових супутникових телеслужбах, цифровому кабельному телебаченні, у DVD;

- MPEG-4 – розширює MPEG-1 для підтримки відео- та аудіо"об'єктів", 3D-контенту і засобів керування цифровими правами.

Для мультимедіа найважливішим зі стандартів MPEG є MPEG4, але використана в ньому схема обробки відео ґрунтується на MPEG1.

MPEG1

Компресія MPEG1 поєднує просторове стискання, основане (як JPEG та DV) на квантуванні й кодуванні частотних коефіцієнтів, отриманих після застосування до відеоданих дискретного косинус-перетворення, та часове, що ґрунтується на *компенсації руху*.

Ключові кадри MPEG називають *І-кадрами* або *І-зображеннями* (від "intra" – "внутрішні"). Їх стискають винятково просторово (внутрішньокадровою компресією). Різницеві кадри, що використовують дані із попередніх кадрів, мають назву *Р-кадри* (від "predicted" – "передбачені"). Р-кадри будують за **попередніми** І- або Р-кадрами. У цьому питанні розробники MPEG пішли далі й допустили, що кадри можна передбачати і за наступними кадрами; у підсумку було визначено *В-кадри* (від "bi-directional prediction" – "двоспрямоване передбачення"). У В-кадрах можна використовувати компенсацію руху з урахуванням не тільки попередніх, але й **наступних** І- та Р-кадрів.

Відеокліп можна закодувати у стиснутій формі як послідовність І-, Р- і В-кадрів. Вона необов'язково повинна бути регулярною, але у схемах кодування зазвичай використовують повторювану послідовність, яку називають *групою зображень* (Group of Pictures – GOP), що завжди починається з І-зображення. Типовий приклад такої схеми наведено на рис. 3.2. Стрілки зображають пряме і двоспрямоване передбачення.

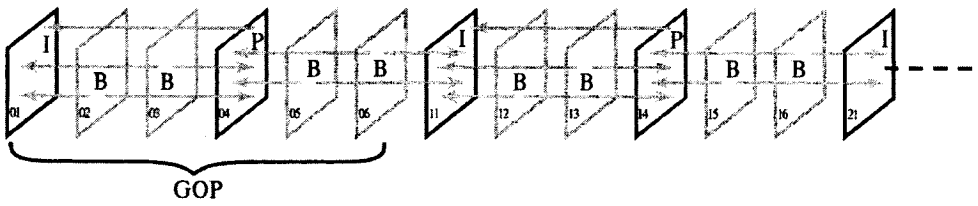


Рис. 3.2. Послідовність кадрів MPEG1 у порядку відображення

Якщо говорити про декодери, то очевидно, що опрацювання В-зображень становить проблему: інформація, необхідна для відновлення відповідного кадру, міститься в І- або Р-кадрах, що з'являться пізніше. Проблему вирішують за допомогою перевпорядкування послідовності. Кажуть, що відеоряд, який відповідає реальному порядку кадрів, іде "у порядку відображення"; для передавання його потрібно зобразити в "порядку потоку бітів". На рис. 3.3 у порядку потоку бітів зображено послідовність, показану на рис. 3.2 у порядку відображення. Тепер усі стрілки, що показують передбачення, спрямовані справа наліво, тобто всі передбачувані кадри розташовані після кадрів, від яких вони залежать.

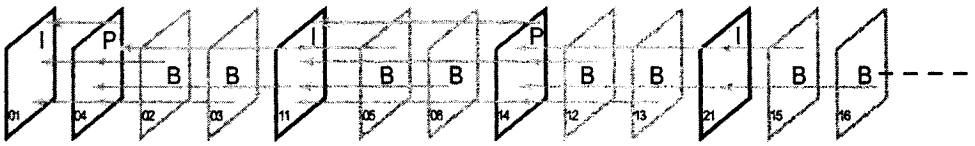


Рис. 3.3. Послідовність кадрів MPEG1 у порядку потоку бітів

З наведених пояснень зрозуміло, що стискання і відновлення MPEG1 є обчислювально громіздкими завданнями.

MPEG2

Стандарту MPEG1 як найстарішому з сімейства MPEG властиві певні недоліки, зокрема підтримка тільки прогресивної (порядкової) розгортки відео. Через це свого часу доволі швидко набув визнання універсальніший стандарт MPEG2, який використовують для стиснення інформації для цифрових відеодисків DVD, кодування аудіо- й відеоданих у теле- та радіомовленні, включаючи супутникове мовлення і кабельне телебачення. MPEG2 підтримує відео і в прогресивній, і в черезрядковій розгортці. Схема компресії, яка використана в ньому, ґрунтується на MPEG1.

MPEG4

Цю саму базову схему використано й у MPEG4 – стандарті, що визначає кодування мультимедійних потоків, які складаються з різних типів об'єктів (відео, нерухомих зображень, анімації, текстури, тривимірних моделей тощо), і пропонує спосіб формування сцен за окремо переданими відображеннями об'єктів. За такого підходу кожен тип об'єктів подано оптимально. Це дає змогу не лише досягти більшого стискання, але також полегшує взаємодію зі сформованою сценою, оскільки об'єкти зберігають власні сутності.

MPEG4 визначає для відеоданих набір профілів різного рівня. У вищих профілях використано метод поділу сцени на *відеооб'єкти* довільної форми (наприклад, людей та інтер'єр, на тлі якого відбуваються події), які можна стискати окремо. Найкращий метод для стискання фонового зображення може відрізнятися від найкращого методу стискання людини, тому, розмежувавши ці різнотипні псевдооб'єкти, можна підвищити загальну ефективність компресії.

3.5. Потокове відео

Принципово відтворення записаного відео з жорсткого диска, DVD чи флеш-пам'яті (тобто в разі автономного доставлення) є однаковим. Зовсім інша справа – його відтворення з мережі, тобто у випадку неавтономного доставлення з віддаленого сервера відеопотоку, який демонструється в міру надходження даних (на відміну від завантаження всього відеофайлу на диск із його подальшим відтворенням).

3.5.1. Завантаження відео з мережі на комп'ютер

Щоб краще зрозуміти природу того, що називають *справжнім потоковим передаванням даних* (true streaming), слід розглянути приклади альтернативних методів доставлення відео, які використовують у Всесвітній павутині.

Найпростішою технологією є впроваджене відео, коли файл фільму передається із сервера на комп'ютер користувача, де й починає відтворюватися повністю безпосередньо після надходження. Удосконалення цього методу називають *прогресивним завантаженням* або *потоковим передаванням через HTTP*. За такого способу доставлення файл передається на диск користувача, але починає відтворюватися не після завершення передавання, а коли певна його частина з'явиться на диску. Момент початку відтворення розраховано так, щоб надходження останнього байта файла збіглося із закінченням відтворення. Після завершення відтворення файл фільму здебільшого залишається на жорсткому диску користувача (принаймні, у кеші веб-браузера).

Останнім часом почали з'являтися практичні реалізації нової розробки – *адаптивного потокового мовлення*. Це гібридний метод доставлення контенту, який функціонує як потік, але при цьому оснований на прогресивному HTTP-завантаженні. Це дуже передова методика, що використовує HTTP як транспортний протокол і завантажує медіа у вигляді послідовності дуже маленьких прогресивних завантажень, а не як одне велике прогресивне завантаження.

3.5.2. Відтворення відео як потоку

На відміну від різних методів завантаження відеофайлів справжнє *потокове відео* ніколи не зберігається на диску користувача. Для згладжування ймовірних затримок можна використати невеликий буфер, але кожен кадр потоку відтворюється практично відразу ж після надходження з мережі. Це означає, що справжнє потокове передавання даних дає змогу відтворювати "живе" відео. Довільний доступ до заданих точок потоку можливий, крім випадку передавання даних у реальному часі.

Потокове відео відкриває можливість доставлення живого відеозображення (у режимі реального часу). Водночас ця сфера вже сьогодні пішла далі від звичайного широкомовного телебачення: кожний належно обладнаний комп'ютер може діяти і як приймач, і як передавач, тому користувачі кількох комп'ютерів можуть спілкуватися візуально, беручи участь у так званих *відеоконференціях*.

Відеоконференція – телекомунікаційна технологія, яка забезпечує одночасне двостороннє передавання, обробку, перетворення та відображення інтерактивної відео- та аудіоінформації на відстані в режимі реального часу за допомогою апаратно-програмних засобів обчислювальної техніки для забезпечення взаємодії між людьми, що спільно працюють над однією проблемою.

Фундаментальним обмежувальним фактором потокового відео є *ширина смуги* (швидкість передавання даних або *бітрейт*). Навіть дуже стиснуте відеозображення зі зменшеним розміром кадрів потребує швидкості порядку мегабітів за секунду, є проблема розсинхронізації відео- та аудіопотоків, котрі доставляються незалежно. Проте сьогодні досягнуто таких параметрів з'єднання з Інтернетом, що потокове відео вже успішно конкурує із широкомовним телебаченням.

3.6. Робота з цифровим відео

У результаті знімання і записування відео одержують тільки початковий (сирий) матеріал. Для створення власне фільму необхідно виконати додаткову роботу.

3.6.1. Редагування і компонування відео

Процес створення завершеного відеопродукту з набору частин називають *редагуванням*. У це поняття входять вибір, вирізання й упорядкування відзнятого матеріалу, а також поєднання зображення зі звуком (якщо він є). Між кадрами можуть бути створені *переходи* (плавні напливи наступних кадрів), однак знятий матеріал не змінюється.

На відміну від цього під час *компонування* матеріал змінюють або додають. Багато змін, виконуваних на цьому етапі, є узагальненням операцій над растровими зображеннями – корекції кольору та контрасту, розмивання чи збільшення різкості тощо. У процесі компонування часто комбінують або накладають елементи з різних кадрів в одну складну послідовність (наприклад, накладають фігури людей на окремо зняте тло), анімують елементи, причому анімацію можна поєднати з живою дією, що часто використовують для створення спецефектів.

Редагування і компонування виконують практично однаково незалежно від того, готується завершене відео для мультимедійного доставлення, звичайного передавання чи записування на відеострічку. Для мультимедійних продуктів потрібен ще додатковий етап – підготовки матеріалу для доставлення. На цьому етапі потрібно йти на компроміси й вирішувати, чим можна пожертвувати, щоб задовольнити вимоги середовища, де буде відтворюватися мультимедіа (наприклад, на малопотужному комп'ютері).

Чим же можна пожертвувати? Розміром кадру, частотою зміни кадрів, насиченістю кольорів та якістю зображення.

3.6.2. Програмне забезпечення для роботи з відео

Для створення й обробки відео є чимало професійних та користувацьких програм: від дорогих ліцензійних до цілком вільних і безкоштовних. Розглянемо найпоширеніші з них.

Adobe Premiere Pro – професійна програма нелінійного відеомонтажу компанії Adobe Systems. Підтримує високоякісне редагування відео роздільності 4K×4K і вище, з 32-бітовими кольорами у колірних просторах RGB та YUV.

Adobe After Effects – програмне забезпечення компанії Adobe Systems для редагування відео та динамічних зображень, розроблення відеокомпозицій, анімації та створення різних ефектів.

Final Cut Pro – програмний комплекс для професійного відеомонтажу, розроблений корпорацією Apple для операційної системи Mac OS. В основу програми покладено гнучку часову шкалу Magnetic Timeline, яка пропонує новий підхід до редагування відео без доріжок.

Windows Movie Maker – програма для створення і редагування відео. Входила до складу клієнтських версій Microsoft Windows аж до Windows Vista. Програма дає змогу виконувати такі операції, як одержання відео із цифрової відеокамери, обрізання або склеювання відео, накладення звукової доріжки, додавання заголовків і титрів, створення переходів між фрагментами відео, додавання простих ефектів тощо. Після випуску Vista роботу над програмою припинено. Як її заміну зараз пропонують програму **Кіностудія Windows**, що входить до складу безкоштовно завантажуваного із сайту Microsoft пакета Windows Live.

3.7. Контрольні питання

1. Що спільного та відмінного між відео та анімацією?
2. Перелічіть стандарти аналогового широкомовлення, цифрового відео і телебачення високої чіткості.
3. Що називають мультимедійним контейнером і які його найпоширеніші формати?
4. Що таке субдискретизація колірності та де і як її використовують?
5. Розкрийте значення поняття відеокодека та назвіть типи відеокодеків.
6. Охарактеризуйте два загальні підходи до стискання відео.
7. Які основні схеми стискання відео використовують сьогодні?
8. Опишіть загальні принципи відеокомпресії MPEG1.
9. Назвіть методи завантаження відео з Інтернету та їхні особливості.
10. Яка різниця між редагуванням і компонуванням відео?

3.8. Приклади тестових питань

1. Відеокодека – це:

   а) пристрій, що виконує компресію аналогового та цифрового відео;
   б) програмне забезпечення, що виконує лише декодування відеопотоку;
   в) пристрій або програмне забезпечення для кодування/декодування відеопотоку.

2. Відеоконференція – це:

   а) технологія записування і відтворення відео;
   б) дія, під час якої дані передаються одночасно в обох напрямках;
   в) дія, під час якої учасники можуть спілкуватися "наживо" лише через модератора.

3. Мультимедійний контейнер – це:

   а) зовнішній пристрій для зберігання мультимедійних даних;
   б) пристрій для об'єднання аналогових та цифрових мультимедійних даних;
   в) пристрій, що може об'єднувати мультимедійні дані, стиснені різними кодеками.

4. Поняття широкомовлення:

   а) це метод передавання даних у мережах соціального призначення;
   б) це метод передавання даних у захищених мережах;
   в) широкомовлення може бути тільки аналоговим.

5. Стандарт MPEG4:

   а) він визначає метод поділу відеопотоків на окремі профілі;
   б) за ним застосовують компресію JPEG до кожного кадра;
   в) за ним пропонують спосіб формування сцени з окремих відеооб'єктів.

6. Методи доставлення відео в Інтернеті:

   б) повне завантаження (файл не зберігається на диску користувача);
   в) потокове передавання (файл зберігається на диску користувача);
   д) прогресивне завантаження (файл зберігається на диску користувача);
   е) прогресивне завантаження (файл не зберігається на диску користувача).

7. Підходи до стискання відео:

   а) просторове стискання;
   б) двовимірне стискання;
   д) позакадрова компресія;
   е) компресія з випередженням.

8. Властивості відеокодека:

   б) відеокодек стискає сигнал;
   в) відеокодек підсилює сигнал;
   г) відеокодек буває аналоговим;
   е) відеокодек буває цифровим.

9. Стандарти цифрового відео:

   б) NTSC;
   в) SECAM;
   г) PAL;
   д) CCIR 601.

10. Стандарти телебачення високої чіткості:

   а) 1080i;
   б) CCIR 601;
   в) HDTV;
   г) SECAM.

Розділ 4

ЦИФРОВЕ АУДІО

Звук – коливальний рух частинок пружного середовища, що поширюється у вигляді хвиль у газі, рідині чи твердому тілі. У вузькому значенні терміном "звук" здебільшого означають коливання, які поширюються в повітрі і сприймаються органами чуттів людей і тварин. Більшість явищ у природі супроводжуються характерними звуками, які живі істоти сприймають, розпізнають і використовують для орієнтування та спілкування.

4.1. Виникнення, сприймання та збереження звуку

Як було вже сказано, цифровий звук у формі аудіозапису поряд із відео та анімацією є важливою складовою мультимедіа. Виникнувши як фізичні коливання повітря, звук проходить непростий шлях перетворень, аж поки не досягне слуху сучасного споживача мультимедійної продукції.

4.1.1. Фізична природа звуку

Якщо різко вдарити камертоном по твердій поверхні, його ніжки будуть вібрувати з точно заданою частотою. Під час їхнього переміщення взад-вперед повітря внаслідок вібрації стискається й розріджується в часі. Взаємодія сусідніх молекул повітря призводить до того, що ці періодичні флуктуації тиску поширюються хвилеподібно. Коли звукова хвиля досягає вуха, вона змушує барабанні перетинки вібрувати з такою самою частотою. Потім вібрація передається через середнє вухо і перетворюється на нервові імпульси, які людина інтерпретує як звук чистого тону, породжений камертоном.

Всі звуки виникають завдяки перетворенню енергії в коливання повітря або якогось іншого пружного середовища. У загальному випадку весь процес може мати кілька етапів перетворення енергії на різні форми.

Ніжки камертона чисто вібрують на єдиній частоті, а от більшість інших джерел звуку вібрують набагато складніше, породжуючи різноманітні знайомі нам звуки й шуми. Одна нота (наприклад, породжена гітарною струною) складається з кількох компонентів із частотами, кратними фундаментальній висоті *(тону)* ноти. Для деяких звуків ударних і більшості природних звуків неможливо навіть

ідентифікувати фундаментальну частоту, хоча такі звуки можна розкласти на набір (часто дуже складний) частотних компонентів.

4.1.2. Сприйняття звуку людиною

Сигнали та фізика звуку є лише частиною загальної картини. Звук насправді існує як відчуття мозку, і сприйняття звуку – це не просто реєстрація фізичних характеристик хвиль, що надходять у вуха. Наприклад, більшість людей із добрим слухом можуть почути власне ім'я, вимовлене на протилежній стороні шумної кімнати, навіть якщо інша частина розмови нерозбірлива, або спокійно спілкуватися зі співрозмовником, гучність мови якого тихіша за фоновий шум.

Однією з найкорисніших ілюзій у сприйнятті звуку є стереофонія. Мозок визначає джерело звуку на основі різниці амплітуди і фази сигналів, отриманих лівим і правим вухами. Якщо для запису звуку використати пару мікрофонів, одержати два монофонічні сигнали, а потім подати їх на два динаміки, розділені певною відстанню, то уявне розташування джерела звуку залежатиме від відносної інтенсивності двох каналів: якщо вони однакові, джерело звуку буде начебто розміщене посередині між динаміками; якщо лівий канал голосніший, джерело звуку здаватиметься розташованим ліворуч і т.д.

4.1.3. Відображення звукових сигналів

Французький математик Фур'є (1768–1830 рр.) довів, що будь-яку періодичну функцію можна розкласти в ряд – суму косинусів і синусів із деякими коефіцієнтами, який тепер називають тригонометричним рядом Фур'є. Алгоритм такого перетворення – *швидкого перетворення Фур'є* – широко використовують у всіх галузях науки і техніки. Зокрема, будь-який звуковий сигнал, навіть найскладніший за формою, можна зобразити у вигляді суми найпростіших синусоїдальних коливань певних частот та амплітуд. На рис. 4.1 показано реальну *звукову хвилю* (графік залежності амплітуди звуку від часу) в різних часових масштабах.

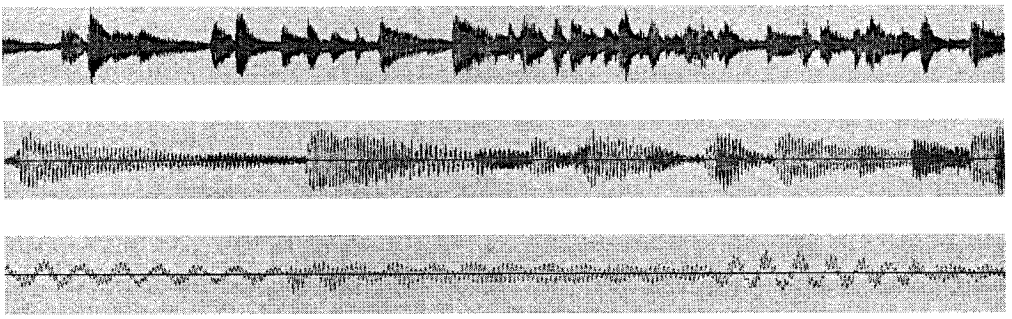


Рис. 4.1. Форма реальної звукової хвилі в різних часових масштабах

Наведемо приклади сигналів, що відображають звуки різних типів. На рис. 4.2, *а* показано мовлення: основний співрозмовник двічі повторює фразу "Feisty teenager", і йому відповідає віддалений голос. На рис. 4.2, *б* показано інший сигнал, що відповідає музичному твору для скрипки, віолончелі й фортепіано. Тут неможливо виділити окремі інструменти, хоча під час прослуховування музики це зробити дуже легко.

а *б*

Рис. 4.2. Форми звукових сигналів мовлення та музики: "Feisty teenager" (а); скрипка, віолончель і фортепіано (б)

Саме ці два типи звуку – музику й мову – найбільше використовують у мультимедійній продукції. Культурний статус музики та лінгвістичний зміст мови означають, що ці два різновиди звуку діють інакше, ніж інші звуки та шуми, і відіграють важливу роль у засобах інформації.

Крім звукової хвилі, є й інші форми опису звукових сигналів, зокрема, *частотний спектр* – зображення амплітуд частотних компонентів звуку. Узагальнену інформацію про звук дає тривимірна *спектрограма*, яка показує взаємозв'язок амплітуди, частоти і часу.

4.1.4. Аналогове записування звуку

Найстаріший із відомих звукозаписів зробив 9 квітня 1860 паризький винахідник де Мартенвіль за допомогою пристрою, названого *фоноавтографом*. У 1877 році американський винахідник Томас Едісон побудував *фонограф* із записом звуку на валу, обернутому олов'яною фольгою, яку надалі замінили воском.

Механічний звукозапис на грамофонних платівках набув поширення через простоту і зручність відтворення звуку в домашніх умовах на грамофонах. У процесі відтворення програвачем грамофонна голка, рухаючись по звивині канавки, повторює ці коливання і передає їх або мембрані, що випромінює звук через рупор, або електромеханічному перетворювачу звукознімача, що виробляє електричні сигнали.

За електромеханічним способом записувані звукові коливання мікрофон перетворює на відповідні електричні струми, що діють після їхнього підсилення на електромеханічний перетворювач – рекордер, який перетворює змінні електричні струми за допомогою магнітного поля на відповідні механічні коливання різця.

Під час фотографічного звукозаписування у такт зі звуковими коливаннями змінюється (модулюється) сила або форма світлового променя, що падає на рухому

кіноплівку, внаслідок чого звук виявляється "сфотографованим". Цей звукозапис застосовують переважно у кіно.

У разі магнітного записування в такт зі звуковими коливаннями намагнічуються окремі ділянки носія, що рухається через магнітне поле, яке створює магнітна головка, через обмотку якої проходять посилені електричні струми мікрофона. Під час відтворення відбувається зворотне перетворення: рухома магнітна фонограма створює в магнітній головці електричні сигнали.

4.2. Оцифрування звуку

Перетворення звукового сигналу на цифрову форму полягає у вимірюванні миттєвих значень його амплітуди через однакові проміжки часу і поданні отриманих значень, які ще називають *відліками*, у вигляді послідовності чисел. Ця процедура називається аналого-цифровим перетворенням, а пристрій для її реалізації – аналого-цифровим перетворювачем (АЦП).

4.2.1. Дискретизація і квантування звукового сигналу

Частоту дискретизації вибирають так, щоб вона щонайменше зберігала повний діапазон чутних частот, якщо, звичайно, необхідне високоякісне відтворення. Для цього використовують спеціальну теорему відліків.

Теорема відліків (Віттакера–Найквіста–Котельникова–Шеннона): щоб відновити аналоговий сигнал без втрат, необхідно, щоб частота дискретизації була принаймні удвічі більша за максимальну частоту вхідного сигналу.

Якщо взяти до уваги, що максимальна чутна частота становить 20 КГц, з теореми випливає, що мінімальною необхідною частотою дискретизації є 40 КГц. Для записування аудіокомпакт-дисків використовують частоту дискретизації 44,1 КГц. Для записування цифрового звуку у форматах, використовуваних у miniDV, digital TV, DVD, DAT, фільмах та професійному аудіо, стандартною є частота 48 КГц.

Дискретизація грунтується на дуже точних синхронізуючих імпульсах, які визначають інтервал між вибірками значень звукового сигналу. Якщо синхронізуючі імпульси починають зміщуватися, то такі часові зміни називають *дрожем*. Наслідком дрожу є внесення шуму у відновлюваний сигнал. За високих частот дискретизації, необхідних для роботи зі звуком, вимоги до дрожу в аналого-цифровому перетворювачі дуже суворі: для звуку CD-якості він не повинен перевищувати 200 пікосекунд (200×10^{-12} с).

На стадії квантування звуку кількість рівнів зазвичай вибирають із міркувань зручної для кодування кількості бітів. Для звуку найчастіше розмір пам'яті для зберігання дискретного значення (*розмір вибірки* або *розрядність*) становить 16 бітів, що дає 65536 рівнів квантування. Мінімально прийнятним є 8-бітовий звук, але вже у ньому проявляється відчутний шум квантування (помилки, що виникають

під час оцифрування аналогового сигналу), тому його можна використовувати тільки там, де певне спотворення прийнятне, наприклад, у застосунках для голосового зв'язку. Якщо необхідно високоякісне відтворення, іноді для зберігання аудіовибірок беруть 24 біти, але в цьому випадку потрібна висока точність схем аналого-цифрового перетворення.

4.2.2. Якість цифрового звуку

Основними параметрами, що впливають на якість оцифрування звуку, є розрядність, частота дискретизації та дрож. Після кодування цифрового звуку його важливою характеристикою стає *швидкість цифрового потоку* (*бітрейт*).

Наприклад, потік даних зі звичайного аудіокомпакт-диска формату AUDIO-CD має швидкість 1411,2 Кбітів/с за частоти дискретизації 44,1 КГц. У форматі DVD-Audio, де дані стиснуті без втрат, при розрядності 24 біти і частоті дискретизації 192 КГц бітрейт може досягати 9216 Кбітів/с. Найпопулярніший формат MP3 стискає дані із втратами: частина звукової інформації, яку вухо людини сприйняти не може, із запису видаляється. Ступінь стиснення можна варіювати, інтервал можливих значень бітрейту становить 8–320 Кбітів/с.

Швидкість потоку даних визначає якість звучання аудіозапису, зокрема суб'єктивну оцінку якості стиснутого звуку, а також розмір файла. Тривалий час була поширена думка, що аудіозапис із бітрейтом 128 Кбітів/с підходить для музичних творів, призначених для прослуховування більшістю людей, забезпечуючи якість звучання AUDIO-CD. Сьогодні вважають, що звучання, яке неможливо відрізнити від оригінального (у разі використання правильно вибраного і налаштованого кодека), зазвичай можна досягти з бітрейтом від 160 Кбітів/с і вище – залежно від початкового аудіофайла, слухача і його аудіосистеми.

4.3. Формати аудіофайлів

Щодо використання цифрового звуку в комп'ютерах, то тут на сьогодні розроблено багато несумісних запатентованих форматів і спеціальних стандартів. Три основні платформи (Windows, MacOS і Unix) мають власні формати звукових файлів, але їхня підтримка є звичною справою для застосунків усіх трьох платформ.

4.3.1. Формати для зберігання звуку

Формат файла визначає структуру й особливості подання звукових даних для зберігання на запам'ятовувальному пристрої комп'ютера. Для усунення надмірності в аудіоданих використовують аудіокодеки, за допомогою яких цифровий звук стискають. Виділяють три групи звукових форматів файлів:

- аудіоформати без стискання, такі як WAV, AIFF тощо;
- аудіоформати зі стисканням без втрат (FLAC, DVD-Audio);
- аудіоформати із застосуванням стискання із втратами (MP3, AAC).

Хоча сьогодні є велика кількість аудіоформатів, для записування і відтворення цифрового звуку найчастіше використовують формати WAV, AIFF, WMA, MP3, AAC та Ogg.

WAV (Waveform Audio Format) – формат аудіофайла, розроблений компаніями Microsoft та IBM. Прийнятий як стандарт для звукового супроводу роботи Microsoft Windows і комп'ютерних ігор. WAV-дані зберігаються у файлах з розширенням **.wav** у нестиснутому вигляді, що призводить до великих обсягів файла. Іншим недоліком файла є обмеження обсягу до 4 ГБ.

AIFF (Audio Interchange File Format) – це популярний формат на платформі Macintosh. Звукові дані у стандартному файлі формату AIFF (з розширенням **.aif** або **.aiff**) зберігаються без стискання. Також існує і стиснута версія формату AIFF, яку називають AIFC.

WMA (Windows Media Audio) – ліцензований формат файла, розроблений компанією Microsoft для зберігання і передавання аудіоінформації. Спочатку формат WMA рекламували як альтернативу MP3, але сьогодні Microsoft на цю роль просуває формат AAC. В останніх версіях формату, починаючи з WMA 9.1, передбачено стискання без втрати якості (*lossless*), багатоканальне кодування об'ємного звуку та кодування голосу.

Формат *MP3* (MPEG-1 Audio Layer-3) має високий регульований ступінь стискання даних і дає змогу створювати файли невеликого розміру. Втім, MP3 – це передусім кодування, а не файловий формат, і MP3-дані можуть зберігатися в інших типах файлів. Зокрема, QuickTime може містити аудіотреки, закодовані MP3, а в роликах SWF – використовувати MP3 для стискання звуку.

AAC (Advanced Audio Coding) – запатентований формат аудіофайла з меншою втратою якості після кодування, ніж MP3, за однакових розмірів. Формат від початку створювали як наступника MP3 з поліпшеною якістю кодування. Формат AAC є одним з найякісніших серед тих, що використовують стискання із втратами, однак поки що поширений значно менше, ніж MP3 та інші альтернативні рішення.

OGG (Ogg Vorbis) – вільний формат стисненого звуку, який використовує власну психоакустичну модель для досягнення високого ступеня компресії даних. За замовчуванням Ogg використовує змінний бітрейт (до 1000 Кбітів/с), підтримує до 255 окремих каналів із частотою дискретизації до 192 КГц і розрядністю до 32 бітів. Файли цього формату, які мають розширення **.ogg**, підтримують усі відомі платформи.

4.3.2. Інтерфейс MIDI

Музику можна передати користувачам двома способами. По-перше, безпосередньо записати звук і відіслати запис; по-друге, записати музику, застосовуючи деякі умовні позначки (ноти), і передати партитуру, щоб користувач міг сам її зіграти.

У цифровій реальності є подібний вибір у доставленні музики. Досі йшлося про способи передавання оцифрованого звуку, тобто еквівалента запису. Однак існує й еквівалент передавання партитури, тобто вказівок про те, як одержати музику.

Основи другого способу доставлення музики пропонує *MIDI* (Musical Instruments Digital Interface – цифровий інтерфейс музичних інструментів). MIDI дає змогу автоматично контролювати інструменти за допомогою пристроїв, які можна запрограмувати на передавання послідовностей MIDI-команд. Комп'ютер також можна обладнати MIDI-інтерфейсом для відсилання необхідних сигналів іншим MIDI-пристроям, і тому комп'ютерні цифрові синтезатори доволі швидко стали популярними. Програмний синтезатор вимагає, щоб MIDI-послідовності зберігалися у файлах, і ця вимога призвела до розроблення стандартного формату MIDI-файлів (з розширенням **.mid**). Такі файли можна передавати між комп'ютерами, які обладнано відповідним програмним забезпеченням, а також вбудовувати в мультимедійну продукцію.

Для відтворення MIDI-файлів необхідний інструмент, що розуміє формат MIDI, але комп'ютер, обладнаний відповідним програмним або апаратним забезпеченням, сам може бути таким інструментом. Звуки можна або синтезувати у звуковій карті, або зберігати на диску у формі аудіовибірок, відтворюваних у відповідь на команду MIDI. Отже, файли MIDI – це засоби музичного повідомлення. Оскільки вони не містять аудіоданих, то можуть бути набагато компактнішими від реальних звукових файлів. Утім, з цієї самої причини вони не можуть гарантувати потрібної точності відтворення.

4.4. Схеми стискання звуку

Складна й непередбачувана природа звукових сигналів рідко дає змогу ефективно стискати їх без втрат – зазвичай потрібна яка-небудь схема стискання із втратами.

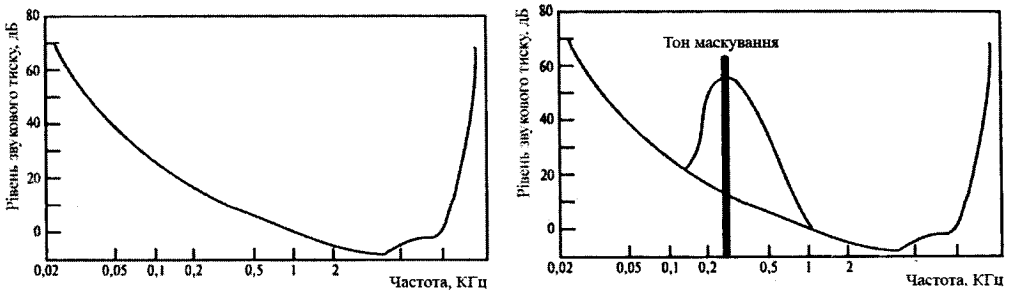
Принципи аудіостискання із втратами відрізняються від принципів компресії відео через те, що ці види інформації люди сприймають по-різному. Зокрема, високі частоти, пов'язані зі швидкими змінами кольорів на зображенні, можна спокійно відкинути, але високі частоти, пов'язані зі швидкими змінами звуку, дуже важливі, тому, щоб вирішити, якими саме аудіоданими можна пожертвувати під час стискання, потрібні інші принципи.

4.4.1. Стискання на основі сприйняття

Секрет ефективного стискання із втратами полягає у визначенні даних, що не мають значення (тобто не впливають на сприйняття сигналу), і їхньому відкиданні. Якщо аудіосигнал оцифровувати прямолінійно, до нього можуть потрапити дані, що відповідають нечутним звукам.

Мінімальний рівень гучності, за якого можна почути звук, називають *порогом чутності.* Цей поріг нелінійно залежить від частоти (рис. 4.3, *а*). На-

приклад, щоб звук дуже низької або дуже високої частоти був почутий, він повинен бути значно голоснішим, ніж тон середньої висоти. Під час стискання аудіо немає сенсу залишати звуки, які не досягають порога чутності, тому алгоритми компресії повинні відкидати відповідні дані. Щоб реалізувати цю ідею на практиці, алгоритм мусить використовувати *психоакустичну модель* – математичний опис сприйняття звуків вухом і мозком.



а *б*

Рис. 4.3. Поріг чутності (а) та маскування (б)

Голосні тони можуть заглушати тихіші, якщо всі вони звучать одночасно. Фактично вони можуть також перекривати слабкі звуки, що пролунали трохи пізніше або (як це не дивно) трохи раніше. Це явище називають *маскуванням,* і його можна описати як модифікацію кривої порога чутності в околі голосного тону (рис. 4.3, *б*). Піднятий фрагмент (або *крива маскування*) нелінійний та асиметричний і швидше росте, ніж спадає. Будь-який звук, що перебуває всередині кривої маскування, буде нечутний, навіть якщо він піднімається над немодифікованим порогом чутності. Це дає додаткову можливість відкинути деякі дані.

4.4.2. Часова компресія аудіо

Важливою технікою, спочатку розробленою для телекомунікаційної галузі (і яку широко використовують у ній і сьогодні), є *адаптивна диференціальна імпульсно-кодова модуляція* (Adaptive Differential Pulse Code Modulation). Ця схема пов'язана з часовою компресією відео, оскільки основана на зберіганні різниць послідовних вибірок, а не їхніх абсолютних значень.

Запис різниць даватиме стискання тільки в тому випадку, коли їх можна записувати за допомогою меншої кількості бітів, ніж вибірки. Аудіосигнали можуть змінюватися швидко, на відміну від послідовних відеокадрів, тому немає особливих підстав сподіватися, що різниця обов'язково буде набагато меншою від значення. Отож у базовій *диференціальній імпульсно-кодовій модуляції* (Differential Pulse Code Modulation) на основі попередніх вибірок обчислюють **передбачуване значення вибірки** і зберігають **різницю між ним і реальним значенням**. Якщо передбачення вдале, різниця буде маленькою.

Адаптивна модуляція додатково збільшує стискання за допомогою динамічної зміни кроку, який використовують для подання квантованих різниць.

4.4.3. Аудіо MPEG

Кращі розроблені алгоритми пов'язані зі стисканням аудіо в контексті стандартів MPEG. MPEG-1 та MPEG-2 є переважно стандартами відео, проте, оскільки відео здебільшого супроводжується звуком, туди також включили аудіокомпресію. Вона виявилась настільки вдалою, що її часто використовують виключно для стискання звуку, особливо музики.

Стандарт MPEG-1 визначає три рівні стискання аудіо. Складність кодування підвищується з переходом від рівня 1 до рівня 3, а в підсумку зменшується бітрейт аудіоданих. Аудіо MPEG-1 рівня 3, або MP3, як його зазвичай називають, пропонує стискання з коефіцієнтом приблизно 10:1, зберігаючи високу якість. На рівні 3 можна використовувати вищі бітрейти, які дають, відповідно, кращу якість. Крім того, тут може бути застосовано *кодування зі змінним бітрейтом* (Variable Bit Rate – VBR), за допомогою якого фрагменти, котрі важко стискати, кодують з більшою швидкістю.

Частина стандарту MPEG-2, пов'язана з аудіо, містить усі кодування, ідентичні аудіо MPEG-1, за винятком кількох доповнень, пов'язаних з об'ємним звучанням. Стандарт MPEG-2 також визначає новий аудіокодек *Advanced Audio Coding* (AAC). Розробники AAC досягли більшого стискання для менших бітрейтів, ніж у MP3: наприклад, аудіо AAC при 96 Кбітах/с на слух здається вищої якості, ніж MP3 при 128 Кбітах/с.

Стискання із втратами завжди сприймають із певними сумнівами: як можна відкинути інформацію, не вплинувши на її якість? Стосовно аудіо MPEG використання компресії із втратами можна пояснити тим, що тут відкинуто інформацію, якої не чути.

4.5. Редагування та синхронізація аудіо

Сьогодні немає єдиного застосунка для роботи зі звуком, який мав би дефакто статус міжплатформного настільного стандарту (як, наприклад, у своїх галузях мають Photoshop і Flash). На аматорському та напівпрофесійному рівні роботи з цифровим звуком достатньо засобів пропонує **Audacity** – вільний багатоплатформний аудіоредактор. Серед надаваних програмою функцій – редагування звукових файлів найпоширеніших форматів, записування та оцифрування звуку, робота з кількома доріжками, зміна параметрів звукового файла, накладення треків і застосування найважливіших звукових ефектів.

У пакети для редагування відео зазвичай інтегровані певні засоби редагування, оброблення та записування звуку. Цих засобів може бути достатньо для створення мультимедіа в разі відсутності спеціальних програм для роботи зі звуком.

4.5.1. Редагування звуку

Можна виділити кілька класів операцій, застосовних до цифрового аудіо. Більшість із них має аналоги у сфері редагування відео й має подібні цілі.

Редагування – це прямий монтаж, комбінування і перевпорядкування аудіофрагментів. Часовий характер звуку природно наштовхує на думку про використання інтерфейсу редагування на основі часової шкали. Типове вікно програми для редагування звуків поділено на *доріжки* (*треки*), звук кожної доріжки зазвичай відображається у вигляді сигналу; масштаб осей часу й амплітуди можна змінювати, щоб бачити звуковий сигнал з різними ступенями деталізації.

Один зі способів редагування полягає у вирізанні та вставленні (або перетягуванні) виділених фрагментів доріжки. У процесі редагування можна комбінувати звуки з різних записів. Під час комбінування відносні рівні окремих доріжок часто переналаштовують, наприклад, щоб збалансувати звучання різних інструментів.

Найпоширенішою операцією з коригування звуку є видалення небажаних шумів. Часто доводиться видаляти фонові шуми, які неминуче захоплює мікрофон під час записування на відкритому просторі. Тут використовують грубий інструмент, відомий як *схема усунення шуму*, що видаляє всі значення, нижчі за певний поріг. До шуму, що лежить у відомому діапазоні частот, можна застосувати фільтри, які видаляють певні частотні смуги. Щоб забрати свист, використовують *фільтри нижніх частот*, які пропускають низькі частоти і видаляють високі; щоб усунути гул (низькочастотний шум, породжений механічними вібраціями), застосовують *фільтри верхніх частот*, що, навпаки, залишають високі частоти і блокують низькі. *Вузькосмуговий режекторний фільтр* видаляє одну вузьку смугу частот.

4.5.2. Звукові ефекти

Розглядаючи ефекти, що впливають на якість звуку, треба зазначити, що сьогодні розроблено їхній повний діапазон – від тих, що мінімально прикрашають запис, до засобів, які радикально змінюють звук або створюють нові звуки на основі оригіналу. Наприклад, ефект *штучної луни*, створюваний цифровими засобами, полягає в додаванні до сигналу запізнілих у часі копій оригіналу зі зменшеною інтенсивністю. Ефект *графічного вирівнювання* змінює спектральні властивості (тембр) звуку, використовуючи блок фільтрів, кожен із яких впливає на вузьку смугу частот. Ефект *тремоло* дає періодичні осциляції амплітуди від нуля до максимального значення.

Двома тісно пов'язаними ефектами, що добре підходять для цифрового звуку, є *розтягування в часі* та *зміна тону*. В аналоговому записі зміну тривалості звуку одержують лише зміною швидкості його відтворення, що відразу впливає на тон звуку. У разі цифрової обробки звуку тривалість можна змінювати, не змінюючи тону, а просто додаючи або видаляючи вибірки. І, навпаки, тон звуку можна змінювати, залишаючи його колишню тривалість.

Розтягування в часі потрібно для синхронізації аудіо з відео або іншим звуком. Якщо, наприклад, голос за кадром говорить надто довго і не вміщається в межі описуваної ним сцени, звукову доріжку можна стиснути в часі, не підвищивши тон голосу диктора.

4.5.3. Об'єднання звуку та зображення

Звук часто використовують як частину відео або анімаційної продукції. У цьому випадку вкрай необхідна синхронізація звуку і зображення. Як характерний приклад можна навести зображення людини, мову якої записано на звуковій доріжці. Якщо синхронізація недостатня, результат буде дезорієнтувати глядача; якщо синхронізація відсутня, результат у найкращому разі буде кумедним, а найімовірніше – безглуздим. Голос за кадром повинен відповідати зображенню, яке він описує, музика часто пов'язана з монтажними переходами, а події на екрані можуть супроводжуватися природними звуками, які мають бути узгодженими.

Щоб встановити синхронізацію, необхідно вміти виділяти потрібні моменти часу. Звичайну кіно- чи відеоплівку поділено на кадри, які природно визначають часові моменти. Звукову стрічку також можна постачити часовим кодом і використати його для синхронізації аудіо- та відеомагнітофонів. Поки часові коди обох пристроїв узгоджені, пристрої синхронізовані.

Звукова доріжка у програмі редагування цифрового відео фізично незалежна від зображення, яке вона супроводжує, тому звук можна легко переміщувати уздовж часової шкали. Звичайно, так чинити небажано, якщо звук і зображення записано одночасно. Проте якщо звукову доріжку створено незалежно від зображення (наприклад, голос за кадром або музичний акомпанемент), звук необхідно узгодити з картинкою.

Таким способом синхронізацію можна виконати в програмі редагування відео, але потім її необхідно підтримувати під час відтворення (можливо, через мережу) відео та звукової доріжки до нього. Якщо вони фізично незалежні (наприклад, передаються різними мережевими маршрутами), можлива розсинхронізація, і це є явищем, якого неможливо уникнути. Зазвичай повторна синхронізація потребує відкидання деяких відеокадрів, що дозволяє зображенню "наздогнати" звук, причому що вищий бітрейт відео, то імовірніше, що воно буде відставати від звуку.

Коли відео й аудіо програють з локального жорсткого диска, синхронізацію підтримувати легше, хоча гарантувати її також вдається не завжди через різну швидкість складових. Дуже короткі кліпи можна цілком завантажити у пам'ять до початку відтворення, що усуває потенційні затримки, пов'язані з читанням із диска, однак цей підхід непрактичний для фільмів значної тривалості. Для них звичним є чергування аудіо та відео, тобто аудіо поділено на фрагменти, які вставлені між відеокадрами.

4.6. Контрольні питання

1. Опишіть процеси породження, передавання і сприймання звуку людиною.
2. Які є способи графічного відображення звукових сигналів?
3. Опишіть процеси дискретизації і квантування звуку під час оцифрування.
4. Охарактеризуйте поняття бітрейту в цифровому аудіо і його вплив на якість звуку.
5. Назвіть найпопулярніші формати для зберігання цифрового звуку.
6. Що таке поріг чутності та психоакустична модель?
7. Опишіть принципи диференціальної імпульсно-кодової модуляції звуку.
8. У чому полягають особливості аудіокомпресії на основі стандарту MPEG-1?
9. Що таке звукові ефекти і які з них є найважливішими?
10. Поясніть причини розсинхронізації аудіо та відео під час відтворення.

4.7. Приклади тестових питань

1. Математична основа зображення звукових сигналів – це сума найпростіших:

   а) експоненціальних кривих різних частот;
   б) експоненціальних коливань різних фаз та амплітуд;
   в) синусоїдальних коливань певних частот та амплітуд.

2. Основне положення теореми відліків:

   а) частота дискретизації має бути менша за максимальну частоту вхідного сигналу;
   б) частота дискретизації має дорівнювати максимальній частоті вхідного сигналу;
   в) частота дискретизації має бути більша за максимальну частоту вхідного сигналу.

3. Властивості формату MP3:

   а) формат MP3 базується на стандарті MPEG-1;
   б) формат MP3 базується на стандарті MPEG-3;
   в) MP3 – це формат стискання цифрового звуку без втрат.

4. Властивості аудіокодека AAC:

   а) аудіокодек AAC базується на стандарті MPEG-1;
   б) аудіокодек AAC базується на стандарті MPEG-2;
   в) аудіокодек AAC сумісний із попередніми стандартами MPEG.

5. Суть ефекту розтягування цифрового звуку в часі:

   а) ефект змінює тривалість звуку, тон залишається незмінним;
   б) ефект змінює тон звуку, тривалість залишається незмінною;
   в) ефект залишає незмінним і тон звуку, і його тривалість.

6. Відображення звукових сигналів може бути:

а) у вигляді звукової гістограми;
б) як графік залежності амплітуди звуку від часу;
б) у формі графіка залежності частоти від часу;
г) у вигляді значка гучномовця.

7. Звукові ефекти:

а) стищування;
б) колірне вирівнювання;
в) штучна луна;
г) відсікання.

8. Особливості аудіокомпресії на основі стандарту MPEG-1:

а) має 1 рівень стискання аудіо;
б) має 3 рівні стискання аудіо;
в) визначає формат AAC;
г) визначає формат WMA.

9. Параметри оцифрування, що впливають на якість цифрового звуку:

а) гучність;
б) відсікання;
в) розрядність;
г) роздільність.

10. Властивості фільтрів для видалення небажаних шумів:

а) схема усунення шуму видаляє всі значення, нижчі за певний поріг;
б) схема усунення шуму видаляє всі значення, вищі за певний поріг;
в) фільтр нижніх частот видаляє низькі частоти;
г) для усунення свисту застосовують фільтри верхніх частот.

Розділ 5

МУЛЬТИМЕДІА І ГІПЕРТЕКСТ

Один із найважливіших компонентів мультимедіа – цифровий текст, але, крім того, в мультимедійній продукції інтенсивно використовують *гіпертекст*, що містить посилання на інші фрагменти чи навіть інші продукти мультимедіа. З іншого боку, гіпертекстові документи, скажімо, веб-сторінки у Всесвітній павутині, можуть містити зображення, звуки, відео й анімацію, вбудовані в них, і програмні модулі, пов'язані з ними для реалізації взаємодії з користувачем. Така мережа елементів різних видів інформації, об'єднаних між собою посиланнями, є прикладом *гіпермедіа*.

Гіпермедіа – це гіпертекст, який містить мультимедійні елементи для формування основи нелінійного середовища інформації.

Частковим випадком гіпермедіа є гіпертекст, у якому присутній тільки один вид інформації – текст. Незважаючи на додаткову технічну складність роботи з багатьма типами інформації, узагальнення гіпертексту до гіпермедіа ідейно є простим та інтуїтивно зрозумілим.

Перші програми-переглядачі були по суті лише текстовими, здатними відображати тільки літери і цифри і сформовані ні їхній основі гіперпосилання. Потім з'явилися браузери, що були здатні відображати картинки, розміщені на веб-сторінках, підтримувати вбудовані у веб-сторінки інтерактивні об'єкти, а також таблиці і форми. Сьогодні уже неможливо уявити собі браузер, що не відтворює мультимедійних ресурсів Інтернету.

5.1. Всесвітня павутина і мультимедіа до HTML 5

Якщо браузер здатний візуалізувати нетекстові дані без сторонньої допомоги, зображення таких даних можна інтегрувати з іншими елементами веб-сторінок. У результаті матимемо режим подання мультимедіа, який ґрунтується на моделі *макета сторінки*. Ця модель розвинулася з багатого досвіду друкованих засобів інформації, пов'язаного з об'єднанням тексту і графіки.

Ці практичні знання можна природно поширити на відео й анімацію, розташовуючи їх на веб-сторінці як ілюстрації, але з додатковою властивістю –

можливістю відтворення. Набагато складніше розташовувати на сторінці звуки – суто невізуальну інформацію. Часто звук зображають за допомогою піктограм або набору елементів керування, які можна розглядати як графічні елементи, а потім, активізуючи їх, відтворювати звук так само, як відео.

5.1.1. Розміщення мультимедіа на веб-сторінках

Для вбудовування елементів мультимедіа у веб-сторінку потрібна спеціальна розмітка. В HTML для розміщення на сторінці інформації будь-яких типів застосовують тег **object**, достатньо гнучкий і такий, що підходить для вставлення різнорідного вмісту. Ранні версії HTML пропонували тільки підтримку растрових зображень за допомогою тегу **img**, який став настільки поширеним, що для зображень його навряд чи замінить **object**.

Крім того, ще в середині 1990-х років компанія Netscape реалізувала у своїх браузерах тег **embed** (згодом також прийнятий Microsoft в Internet Explorer) для вставлення відео, звуку та інших типів інформації. Хоча **embed** ніколи не мав офіційного статусу, його широко використовували і навіть процедурно генерували програми створення HTML-документів.

Проте найкращим (тобто рекомендованим Консорціумом W3C) засобом вбудовування мультимедіа і виконуваного вмісту у веб-сторінки до впровадження HTML 5 був тег **object**.

5.1.2. Відтворення мультимедіа в HTML 4.01

У HTML посилання мають атрибут **href**, значення якого – URL-адреса, що вказує на місце призначення. Спочатку малося на увазі, що URL вказує на HTML-сторінку. А якщо це не так? Для протоколу HTTP це не має значення. Якщо серверу дано команду відкрити ресурс, ідентифікований його URL-адресою, він це зробить. Крім даних, що стосуються ресурсу, протокол зазначає тип інформації, і, якщо це не **text/html**, браузер або намагається впоратися з цією проблемою самостійно, або звертається за допомогою до іншої програми.

Щоб, крім відформатованого тексту, браузер міг правильно відображати інші типи інформації, потрібно було відповідно розширити його можливості і надати додаткові теги HTML для контролю над компонуванням сторінок із вбудованою графікою, відео та звуком. Очікувати, що браузер зможе впоратися з абсолютно усіма можливими типами даних, було нереально: для деяких із них найкращим вибором завжди буде допоміжна програма. З іншого боку, окремі типи, особливо зображення і текстові документи, наприклад, у форматі PDF, стали настільки поширеними, що було розумно очікувати їхнього відображення від самого браузера.

Такі види інформації, як відео й аудіо, вже давно стали повсюдно вживаними, але до HTML 5 витрати на реалізацію нових функцій браузерів, які давали б їм змогу самостійно опрацьовувати й відтворювати мультимедіа, вважали економічно невиправданими. Для вирішення цієї проблеми використовували спеціальні програмні модулі – *плагіни* (*plug-ins*) – допоміжні застосунки, які користувач міг

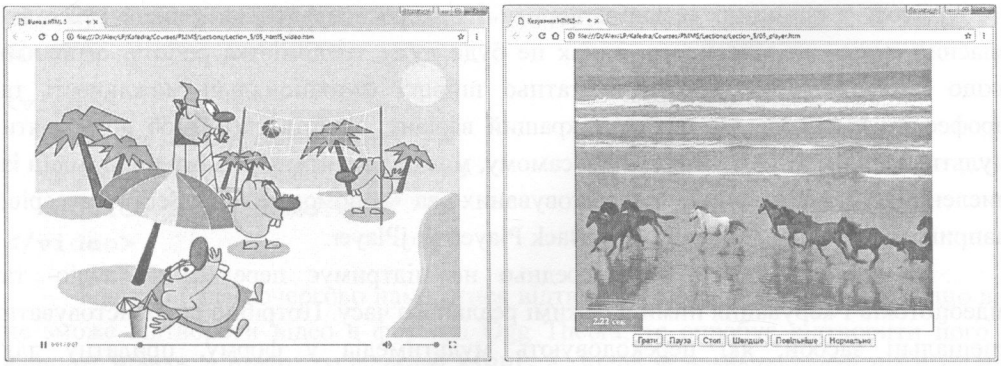
інсталювати самостійно, якщо вони йому потрібні. Плагіни завантажувалися під час запуску браузера й підвищували його функціональні можливості, зокрема пов'язані з обробкою додаткових типів інформації. Прикладом популярного плагіна є Flash Player, який переносить усі функціональні можливості однойменного самодостатнього застосунка у кожний розвинутий браузер.

5.2. Гіпермедіа в HTML 5

Сьогодні вже завершено роботу над специфікацією HTML 5, яка спрямована на скорочення використання заснованих на плагінах технологій, таких як Adobe Flash та Microsoft Silverlight. У жовтні 2014 року консорціум W3C оголосив про надання набору специфікацій HTML 5 статусу "рекомендованого стандарту".

5.2.1. Теги audio та video

HTML 5 вводить низку нових тегів і атрибутів, які відображають типову структуру сучасних веб-сторінок. Однак новації не обмежуються тільки розміткою і містять низку веб-технологій, котрі формують відкриту веб-платформу – інтегрований набір засобів для крос-платформових застосунків, здатних взаємодіяти з обладнанням і підтримувати роботу з мультимедіа – відео, графікою та анімацією. Зокрема, теги **audio** та **video** забезпечують нові функціональні можливості для розміщення і відтворення через стандартизований інтерфейс звукових файлів чи відеозаписів.



а *б*

Рис. 5.1. Відтворення вбудованого відео за допомогою стандартного (а) і модифікованого (б) плеєра HTML 5

У HTML 5 для того, щоб розмістити на веб-сторінці відео- або аудіо-файли, достатньо скористатись відповідними тегами. Найпростіший спосіб вбудувати відео – це використати тег **video** і дозволити браузеру відобразити інтерфейс для відтворення за замовчуванням (рис. 5.1, *а*). Необов'язковий атрибут **poster** можна використовувати для задання зображення, яке буде відображатися до того, як відео

почне програватися. Атрибут **source** визначає альтернативні відео- та аудіофайли, щоб браузер міг вибрати той, який підходить до підтримуваного медіатипу або кодеків; якщо цей атрибут використано, то не слід вживати **src**, інакше **source** буде проігноровано.

Так само просто підключити й звуковий файл, використовуючи тег **audio**, у якого більшість атрибутів спільні з **video**.

5.2.2. Використання модифікованих HTML5-плеєрів

Стандартний програвач HTML 5 має фіксований набір функцій: запуск/пауза, зміна гучності, вхід/вихід з повноекранного режиму (для відео) і переходи за часовою шкалою. Проте всі мультимедійні елементи мають доступ через HTML5 Media API до об'єктної моделі JavaScript, за допомогою якої можна керувати відтворенням за допомогою сценаріїв і, більш того, реалізувати додаткові функціональні можливості.

HTML 5 надає розробникам можливість у разі необхідності самостійно створювати власні аудіо- та відеопрогравачі. Основна ідея полягає в тому, щоб відключити елементи керування стандартного HTML5-плеєра, залишивши тільки саме вікно відтворення, і додати свої кнопки для керування ним. А щоб ці кнопки функціонували, необхідно записати відповідний JavaScript-код. Приклад найпростішого мультимедійного програвача, створеного таким способом, наведено на рис. 5.1, *б*.

Отож у принципі можна створити і використовувати на своєму веб-сайті власний аудіо- чи відеоплеєр, однак це буде дуже трудомістка робота, особливо якщо необхідно забезпечити достатньо широкі функціональні можливості та професійний дизайн. На щастя, є кращий варіант: замість того, щоб розробляти мультимедійний HTML5-програвач самому, можна завантажити з Інтернету один із численних безкоштовних, налаштовуваних за допомогою JavaScript плеєрів, наприклад, YouTube, Video.js, LeanBack Player чи jPlayer.

На жаль, HTML 5 безпосередньо не підтримує передавання аудіо- та відеопотоків і керування ними в режимі реального часу. Потрібно використовувати спеціальні засоби, які перекодовують мультимедіа у форму, придатну для потокового передавання, і лише тоді відтворюють із відчутною затримкою у кількадцять чи й кількадесят секунд. На щастя, ці проблеми практично невідчутні для звичайних власників невеликих сайтів, користувачів соціальних мереж чи контент-менеджерів, яким потрібно розмістити потокове мультимедіа на своїх веб-сторінках, – для цього достатньо скористатись уже готовими розробками та інструментами. Зокрема, за допомогою служби YouTube можна без проблем організувати на своєму сайті відеотрансляцію в онлайн-режимі, а сервіс Hangouts On Air соцмережі Google+ дає змогу проводити вебінари та відеоконференції.

5.2.3. Підтримувані мультимедійні формати

Важливим для HTML 5 є питання аудіо- та відеоформатів, у контексті якого іноді навіть говорять про "війну мультимедійних форматів". Якщо раніше у браузерах можна було відтворювати аудіо та відео різних форматів, встановивши відповідні плагіни, то тепер відмова від плагінів потребує визначення форматів, підтримуваних стандартним плеєром HTML 5.

Сьогодні до переліку основних форматів, підтримуваних у HTML 5, входять чотири аудіоформати: MP3 (рекомендоване розширення **.mp3**), AAC (**.aac**), Ogg Vorbis (**.ogg**) та WAV (**.wav**) – і три відеоформати – MPEG-4 (H.264, розширення **.mp4**), Ogg Theora (**.ogv**) та WebM (**.webm**). Вживання саме рекомендованого розширення важливе для правильної ідентифікації використаних аудіо- та відеокодеків і контейнера, інакше браузер може відмовитися відтворювати мультимедійний файл. На практиці ж ситуація ще складніша, оскільки контейнер певного формату здатен підтримувати кілька різних аудіо- і відеостандартів.

Інший аспект проблеми – підтримка різними браузерами різних форматів мультимедіа. Наприклад, найновіші (на початок 2017 року) версії браузерів Chrome, Firefox та Opera загалом підтримують всі три зазначені відеоформати, Internet Explorer і Safari підтримують лише MPEG-4 (H.264), а Microsoft Edge не підтримує Ogg Theora. Підтримка ж цих форматів мобільними браузерами взагалі є окремими питанням.

Для вирішення проблеми форматів у HTML 5 власне й було розроблено систему задання альтернативних аудіо- та відеофайлів за допомогою атрибута **source**:

```
<video>
  <source src="demo.ogv" />
  <source src="demo.webm" />
  <source src="demo.mp4" />
</video>
```

Браузер буде почергово намагатися відтворити файл кожного типу: якщо він не зможе відтворити відео в форматі Ogg Theora, то спробує відтворити його у форматі WebM, а потім – у форматі MPEG-4. Якщо ж розраховувати ще й на користувачів із браузерами, що не визнають HTML 5 (наприклад, Internet Explorer 8), то варто додати до ресурсів тегу **video** вбудований об'єкт, який використовуватиме для свого відтворення, скажімо, Flash-плеєр.

5.3. XML і мультимедіа

Сьогодні є чимало мов, розроблених для створення часових засобів інформації та векторної графіки. Багато з них базовані на XML і є відкритими стандартами W3C із текстовим поданням.

XML (Extensible Markup Language, розширювана мова розмітки) – рекомендований Консорціумом WWW текстовий формат для зберігання ієрархічно структурованих даних, стандарт для побудови спеціалізованих мов розмітки для обміну інформацією між різними застосунками.

Вивчаючи структуру та вміст документів, написаних цими мовами, легко зрозуміти, яким чином такі концепції, як синхронізація чи зафарбовування фігур, можна подати у вигляді, придатному для опрацювання комп'ютерною програмою.

5.3.1. HTML, SGML та XML

Найкращий спосіб організувати інформацію у Всесвітній павутині з урахуванням її неоднорідної природи – це відокремити вміст і структуру від зовнішнього вигляду, задаючи за допомогою CSS і HTML структурні та візуальні властивості. Однак множина тегів, які пропонує HTML, обмежена, що, своєю чергою, обмежує можливості зазначених засобів.

Хоча тегів HTML достатньо для розмітки простого, переважно текстового, документа (для чого, власне, й було створено цю мову), вони не дають змоги охопити всі типи матеріалів, які трапляються у WWW. Будь-яка спроба додати до HTML нові теги, що враховували б весь можливий спектр документів, приречена на фіаско. Набагато краще рішення – запропонувати веб-дизайнерам засіб, що давав би їм змогу визначати власні теги. Вирішення цієї проблеми кілька років проіснувало у формі мови *SGML* (Standard Generalized Markup Language – стандартна узагальнена мова розмітки), яку й зараз використовують у видавничій справі.

Робота з адаптації SGML до Інтернету призвела до визначення підмножини цієї мови, названої Extensible Markup Language – XML. Вона дає змогу веб-дизайнерам визначати власні набори тегів для будь-якого типу документів, звільняючи веб-сторінки від обмежень HTML. Формальне визначення набору тегів та їхніх атрибутів, а також умови і способи їхнього об'єднання подають у *визначенні типу документа* (Document Type Definition – DTD) XML. Інакше кажучи, XML не використовують безпосередньо для розмічування веб-сторінок, а розробляють DTD-шаблони XML, які визначають нові мови розмітки для виконання конкретних завдань.

Процес визначення нових мов на базі XML почався давно. З DTD-шаблонів, що набули поширення, можна виділити: *SMIL* (Synchronized Multimedia Integration Language – мова інтеграції синхронізованих засобів мультимедіа), *SVG* (Scalable Vector Graphics – масштабована векторна графіка), *MathML* (Math Markup Language – мова розмітки математики) і *XFDL* (eXtensible Forms Description Language – мова опису розширюваних форм). Визначенням мови HTML з використанням DTD-шаблона XML є *XHTML*.

5.3.2. Мультимедійні засоби мови SMIL

SMIL (Synchronized Multimedia Integration Language – синхронізована мова інтеграції мультимедіа) – базована на XML тегова мова, призначена для визначення часової структури мультимедійної презентації.

Першу специфікацію SMIL 1.0 було прийнято як Рекомендацію Консорціуму WWW у червні 1998 року, а в грудні 2008 року прийнято останню версію SMIL 3.0. Однак 1 квітня 2012 року було оголошено про закриття робочої групи SYMM Консорціуму, яка працювала над розробленням специфікації SMIL.

Сьогодні сфера використання SMIL є дуже вузькою; основна цінність мови полягає в тому, що її текстова форма розкриває сутність синхронізації у відтворенні часових мультимедійних елементів, які зазвичай приховані використанням інструмента часової шкали. Крім того, у SMIL було вперше розроблено принципи декларативної анімації, які зараз широко використовують в інших XML-базованих мовах.

5.3.2.1. Синхронізація у SMIL

На найвищому рівні структура документа SMIL подібна до структури HTML-файла: весь документ міститься всередині тегу **smil**, який має теги **head** та **body**. Реальний вміст документа SMIL закладено в його тіло (тег **body**). У тілі можна розміщувати так звані *теги синхронізації,* за допомогою яких визначають часові властивості мультимедійних об'єктів.

Тегів синхронізації є два: **par** (від parallel – паралельний) і **seq** (від sequence – послідовний). Кожен тег може містити *теги мультимедійних об'єктів*, які задають реальні зображення, відеокліпи, звук тощо, використані у презентації. Щоб описати складні взаємозв'язки синхронізовуваних об'єктів, теги синхронізації можна вкладати один в одний. Елементи, розміщені всередині **par**, можуть відтворюватися одночасно, тоді як елементи із **seq** відображаються послідовно.

Будь-який елемент синхронізації можна відтворити певну кількість разів, задавши її за допомогою атрибута **repeatCount**; його значенням також може бути **indefinite**, тобто необмежена кількість разів. Атрибут **repeatCount** можна застосовувати до **par** або **seq**, а також безпосередньо до мультимедійних елементів.

Хоча, на перший погляд, теги синхронізації SMIL і часові атрибути видаються трохи незграбними, їх достатньо, щоб дати змогу точно задавати будь-який можливий часовий зв'язок між окремими елементами презентації.

5.3.2.2. Анімація

Основою механізму анімації у SMIL є тег **animate**, атрибути якого задають властивості елемента, який буде анімовано, і принципи зміни його параметрів у процесі анімації. За замовчуванням анімацію застосовують до батьківського стосовно **animate** тегу. Наприклад, щоб анімувати зображення, **animate** потрібно розмістити всередині тегу **img**.

Часові зміни анімованої властивості можна задавати кількома способами. У найпростішому випадку використовують два атрибути тегу **animate** (**from** і **to**),

за допомогою яких задають початкове та кінцеве значення властивості. Зазначивши, що саме змінювати (**attributeName**) і в яких межах (**from** і **to**), необхідно лише задати, коли і як довго повинні тривати зміни: для цього використовують атрибути **dur** (тривалість), **begin** (час початку) та **end** (час завершення).

Розглянутий механізм анімації можна проілюструвати на простому прикладі. Припустимо, потрібно збільшити зображення, змінивши висоту і ширину елемента **img**. Для цього можна скористатись таким кодом:

```
<img src="Images/logo.jpg" type="image/jpeg" >
 <animate attributeName="width" from="10" to="100" dur="9s" />
 <animate attributeName="height" from="10" to="100" dur="9s" />
</img>
```

Почавши з невеликого розміру (квадрата зі стороною 10 пікселів), зображення рівномірно збільшується упродовж дев'яти секунд, досягнувши значення сторін 100 пікселів.

Описаний синтаксис із використанням атрибутів **from** і **to** є частковим випадком більш загального опису анімації набором значень, яких набуває змінювана властивість у процесі анімації. Ці значення задають у вигляді списку і присвоюють його атрибуту **values**, наприклад: **values="10;20;40;80;160"**. Атрибут **calcMode** задає різновид інтерполяції значень: **linear** – лінійну, **discrete** – постійні значення, що дискретно змінюються, **paced** – значення змінюються з постійною швидкістю.

Засоби анімації, які пропонує SMIL, є вкрай примітивними, і створення анімації вручну за їхньою допомогою буде дуже стомливим. Тим не менш будь-який різновид цього процесу можна виразити за допомогою тегу **animate** із застосуванням синхронізуючих тегів.

5.3.3. Векторна графіка SVG

SVG (Scalable Vector Graphics – масштабована векторна графіка) – базована на XML тегова мова, призначена для опису двовимірної векторної і векторно-растрової графіки, а також формат файлів для статичної, анімованої та інтерактивної графіки.

У специфікації SVG сказано: "SVG розуміє три типи графічних об'єктів: форми векторної графіки (наприклад, траєкторії, що складаються із прямих і кривих ліній), зображення і текст". Із такими самими типами об'єктів працюють і редактори векторної графіки, такі як CorelDRAW і Adobe Illustrator, і зазвичай SVG-документи створюють експортуванням із таких програм. Подальший короткий опис цієї мови потрібен не для того, щоб навчитися самостійно писати SVG-код, а щоб показати, як об'єкти і перетворення векторної графіки можна подати у форматі, зрозумілому і для комп'ютера, і для людини.

5.3.3.1. Опис об'єктів векторної графіки

Проілюструємо на тривіальному прикладі, як за допомогою конструкцій XML у SVG можна зобразити об'єкти векторної графіки.

```
<svg xmlns="http://www.w3.org/2000/svg" baseProfile="full">
  <rect x="120" y="120" width="126" height="126"
        fill="green" stroke="magenta" stroke-width="4" />
</svg>
```

На відміну від HTML і SMIL, документи SVG не поділено на заголовок і тіло. У тегові **svg** міститься вся суть документа – зображення графічних об'єктів. У наведеному прикладі об'єкт лише один – кольоровий квадрат. Ця форма описана єдиним тегом **rect**, атрибути якого надають усю інформацію, необхідну для зображення квадрата в обраному місці.

Кожен представник множини простих форм (прямокутники, кола, еліпси, лінії, ламані лінії і багатокутники) має власний тег, кожен із яких має власні характерні атрибути для задання форми і розташування. Форми є частковим випадком траєкторій, що складаються з послідовності прямих і кривих ліній; у такому загальному випадку використовують тег SVG **path**.

Контури і заповнення можна зробити частково прозорими, для чого передбачені атрибути **stroke-opacity** та **fill-opacity**: за значення 0 кольори повністю прозорі, значення 1 позначає повну непрозорість, проміжні значення дають часткову прозорість. SVG підтримує також градієнтне заповнення лінійного та радіального типу, яке задають тегами **linearGradient** або **radialGradient**.

5.3.3.2. Перетворення

Існує набір перетворень, які легко застосовні до об'єктів векторної графіки: переміщення, масштабування, поворот, скошування. Усі перетворення виконують за допомогою атрибута **transform**, який можна використовувати з будь-яким тегом форми.

Значення **transform** – це рядок зі специфікацій перетворення, розділених пробілами або комами. Кожна специфікація складається з назви функції перетворення, за якою в дужках записують потрібні аргументи. Застосовують такі назви функцій: **translate**, **scale**, **rotate**, **skewX** та **skewY**, значення яких очевидні.

Іноді потрібно виконати одне й те саме перетворення відразу для кількох об'єктів. Це можна зробити, згрупувавши об'єкти, а потім застосувавши перетворення до групи. Тег **g** формує групу, а його вмістом є всі елементи групи. Цей тег може мати атрибут **transform**, і тоді перетворення буде застосовано до всіх об'єктів групи.

5.3.3.3. Анімація

SVG має простий, але потенційно потужний засіб анімації, який ґрунтується на анімації SMIL. Зокрема будь-який графічний елемент може містити тег **animate**

з кількома атрибутами, що задають, як значення параметрів об'єкта повинні змінюватися з часом. Наприклад, кольори описаного раніше прямокутника композиції можна анімувати так:

```
<rect x="120" y="120" width="126" height="126"
  fill="lime" stroke="magenta" stroke-width="4">
  <animate attributeName="fill" values="green;magenta;green"
           dur="2s" calcMode="linear"
repeatCount="indefinite" />
</rect>
```

У цьому коді записано зміну кольору прямокутника з початкового зеленого на пурпуровий, яким намальовано контур, а потім знову на зелений. Цикл триває дві секунди. Оскільки режим **caclMode** має значення **linear**, зміни кольорів виглядають плавними. Практично всі атрибути об'єктів допускають таку анімацію, тому SVG можна розглядати як мову реалізації векторної анімації для Інтернету (рис. 5.2).

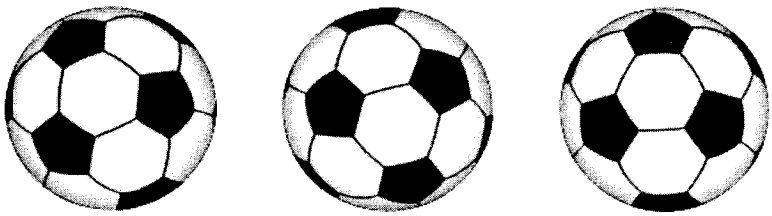


Рис. 5.2. Кадри SVG-анімації: обертання м'яча

5.4. Технологія WPF

WPF (Windows Presentation Foundation) – графічна підсистема у складі .NET Framework, призначена для створення клієнтських застосунків із візуально привабливими можливостями взаємодії з користувачем.

WPF – це високорівневий об'єктно-орієнтований фреймворк (готовий до використання комплекс програмних рішень), що дає змогу розробляти двовимірні та тривимірні інтерфейси. За допомогою WPF можна створювати широкий спектр як автономних, так і веб-орієнтованих застосунків.

В основу WPF покладено векторну систему візуалізації, що не залежить від роздільності пристрою виведення і враховує можливості сучасного графічного устаткування. WPF надає засоби для створення візуального інтерфейсу, зокрема мову XAML, елементи керування, макети, стилі, шаблони, двомірну і тривимірну графіку, анімацію, мультимедійні засоби.

5.4.1. Мова XAML

XAML (Extensible Application Markup Language – розширювана мова розмітки застосунків) є мовою розмітки, яку використовують для створення екземплярів об'єктів .NET. Хоча мова XAML – це технологія, яку можна застосувати до багатьох різних предметних галузей, її головне призначення – конструювання інтерфейсів користувачів WPF.

Документи XAML визначають розташування панелей, кнопок та інших елементів керування у вікнах застосунків WPF. Користувач не пише код XAML вручну – він просто використовує інструмент, що його генерує.

5.4.2. Вікна непрямокутної форми

Вікна незвичної форми часто є товарним знаком ультрасучасних популярних застосунків. Зробити це за допомогою засобів WPF доволі просто, хоча насправді створення привабливих вікон незвичайної форми потребує чималих дизайнерських зусиль (рис. 5.3).

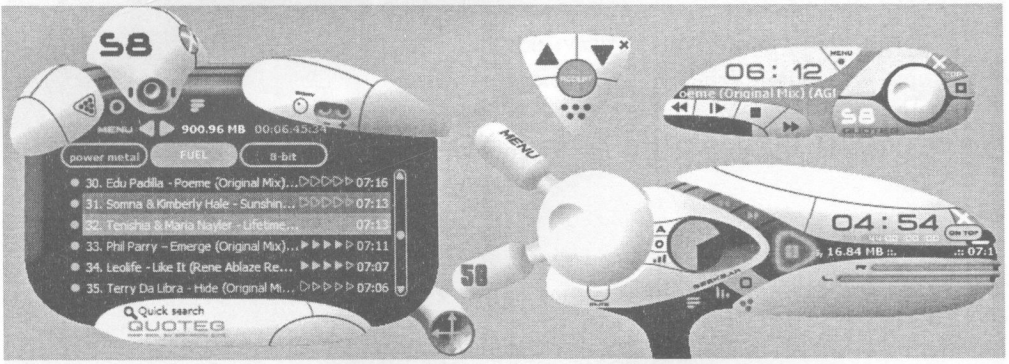


Рис. 5.3. Нестандартні вікна для обкладинки Quoteg плеєра AIMP: це можна зробити засобами WPF

Базова процедура для створення вікна нестандартної форми передбачає виконання таких кроків.

1. Встановити для властивості вікна **AllowsTransparency** значення **True**.
2. Задати для властивості вікна **WindowStyle** значення **None**, щоб сховати рядок заголовка.
3. Визначити для тла (властивість вікна **Background**) прозорий колір (значення **Transparent**). Інший спосіб – використати для тла зображення, що має прозорі ділянки.

Ці три кроки позбавляють вікно стандартного зовнішнього вигляду. Для забезпечення ефекту незвичної форми необхідно визначити якийсь непрозорий вміст, що має потрібну форму.

За допомогою вікон нестандартної форми у WPF можна створювати ілюзію тривимірності програмних інтерфейсів (див. рис. 5.3).

5.4.3. Анімація

Анімація – важлива частина моделі. Щоб запустити її, не потрібно використовувати таймер і код обробки подій: замість цього анімацію створюють декларативно, аналогічно до засобів SMIL та SVG. Анімація інтегрована у звичайні вікна WPF: наприклад, якщо анімувати кнопку так, щоб вона дрейфувала вікном, вона при цьому залишається кнопкою – її можна натиснути, запустивши звичайний код обробника події. Ефектно може також виглядати анімація у прозорому вікні – поверх вікон інших програм або робочого столу (рис. 5.4).

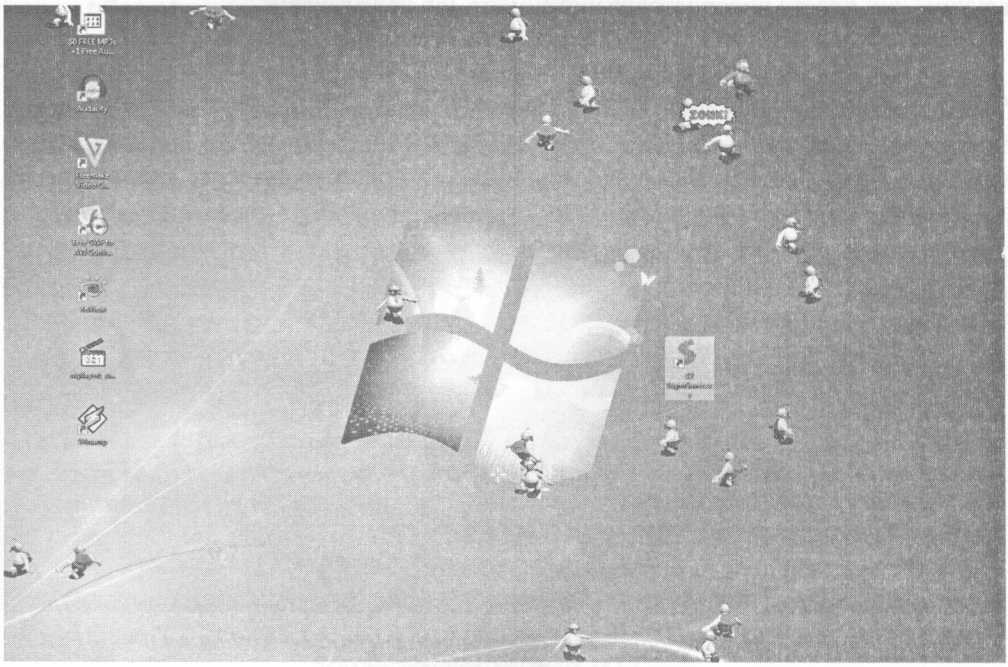


Рис. 5.4. Анімація WPF у прозорому вікні

Ось приклад простої анімації, що змінює розміри кнопки: коли користувач її натискає, вона плавно розширюється. Для цього слід використати анімацію, що модифікує властивість **Width** кнопки. Відповідний код може виглядати так:

```
DoubleAnimation widthAnimation = new DoubleAnimation();
widthAnimation.From = <початкове значення>;
widthAnimation.To = <кінцеве значення>;
widthAnimation.Duration =
TimeSpan.FromSeconds(<тривалість>);
cmdGrow.BeginAnimation(Button.WidthProperty,
widthAnimation);
```

Необхідний мінімум для опису анімації, що використовує лінійну інтерполяцію, становлять три параметри: початкове значення (**From**), кінцеве значення (**To**) і час, за яке анімація повинна виконатись (**Duration**).

5.4.4. Тривимірна графіка

Підтримка тривимірної графіки – одне з важливих нововведень платформи WPF. Попередні високорівневі набори інструментів розроблення, такі як Windows Forms, взагалі обходилися без підтримки 3D-засобів. WPF же надає користувачу надзвичайно прості у використанні й водночас ефективні інструменти для роботи з ними. З одного боку, можна створити складний код, що генерує і модифікує тривимірні сітки із залученням серйозного математичного апарату, водночас можна просто експортувати тривимірні моделі зі спеціалізованого програмного пакета і маніпулювати ними за допомогою простих трансформацій. Ключові засоби забезпечені високорівневими класами, що не потребують глибоких знань.

5.5. Контрольні питання

1. Що таке гіпермедіа?
2. Опишіть засоби для вставлення і відтворення мультимедіа до HTML 5.
3. Які нові засоби для роботи з мультимедіа з'явилися в HTML 5?
4. Перелічіть мультимедійні формати, підтримувані стандартним плеєром HTML 5.
5. Поясніть, що таке XML з погляду гіпермедіа, та опишіть принципи її використання у цій сфері.
6. Які XML-базовані мови використовують у гіпермедіа і яким чином?
7. Назвіть ключові особливості мови SMIL.
8. Поясніть принципи створення анімації в SVG.
9. Які можливості надає користувачу технологія WPF?
10. Опишіть базову схему для створення вікна нестандартної форми у WPF.

5.6. Приклади тестових питань

1. Поняття XML з погляду гіпермедіа:

   а) це об'єктно-орієнтована мова мультимедійних застосунків;
   б) вона базується на мовах, що є стандартами W3C;
   в) це текстовий формат для зберігання ієрархічно структурованих даних.

a) Поняття технології WPF:

   а) WPF – це графічна підсистема у складі Microsoft Silverlight;
   б) WPF – це процедурно-орієнтований функціональний шар;
   в) WPF дає змогу розробляти тривимірні програмні інтерфейси.

b) Поняття гіпермедіа:

   а) це мультимедіа, що містять гіпертекстові елементи;
   б) це гіпертекст, що містить мультимедійні елементи;
   в) це текст, де мультимедійні елементи формують гіперпосилання.

c) Особливості синхронізації у SMIL:

а) синхронізацію об'єктів у SMIL здійснюють за допомогою тегів;
б) для синхронізації об'єктів у SMIL використовують часову шкалу;
в) для синхронізації об'єктів у SMIL використовують часові мітки.

d) Властивості стандартного HTML5-плеєра:

а) він відтворює мультимедійні файли лише певних форматів;
б) він відтворює мультимедійні файли всіх поширених форматів, крім потокових;
в) він відтворює файли всіх поширених форматів і потокове мультимедіа.

e) XML-базовані мови, які використовують у гіпермедіа:

а) DTD;
б) SGML;
в) SVG;
г) WPF.

f) Атрибути і функції для перетворення об'єктів у SVG:

а) атрибут **animate;**
б) атрибут **resize;**
в) функція **resize;**
г) функція **translate.**

g) Засоби для вставлення і відтворення мультимедіа до HTML 5:

а) для розміщення мультимедіа тоді використовували теги **audio** та **video**;
б) для розміщення мультимедіа тоді використовували теги **object** та **embed;**
в) для розміщення мультимедіа тоді використовували тег **multimedia**
г) для відтворення мультимедіа тоді використовували стандартний інтерфейс

h) Властивості мови XAML:

а) у XAML анімацію можна створювати подібно до SMIL;
б) мова XAML базується на SMIL;
в) мова XAML є наступником SMIL;
г) XAML використовують тільки у системі WPF.

i) Підтримувані в HTML 5 відеоформати:

а) AVI;
б) Ogg Vorbis;
в) SWF;
г) WebM.

Розділ 6

МУЛЬТИМЕДІА ТА МЕРЕЖІ

Зв'язки між комп'ютерними мережами і гіпермедіа є суперечливими. Мультимедіа висувають значні вимоги до ресурсів: розміри файлів бувають дуже великі, часто потрібне складне опрацювання (наприклад, відновлення стиснутих даних), час відгуку повинен бути малим, іноді необхідно задовольняти складні умови синхронізації. З погляду цих вимог мережі – дуже невдалий вибір, оскільки обмін через мережу ускладнений внутрішніми обмеженнями наявних технологій.

Водночас мультимедійні дані дуже вигідно зберігати у місцях, доступних через мережу: можна уникнути дублювання тих самих великих файлів, для поширення не потрібні фізичні носії (наприклад, компакт-диски), до того ж такий підхід узгоджується із сучасною тенденцією до використання розподілених систем настільних комп'ютерів, з'єднаних із центральними серверами. До того ж об'єднання мереж і мультимедіа сприяє розвитку таких нових застосунків, як відеоконференції та "живі" мультимедійні презентації. Вигоди від доставляння мультимедіа через мережі зазвичай настільки привабливі, що доводиться миритися із супутніми обмеженнями і намагатися їх подолати.

Розглянемо лише мережі TCP/IP, на яких базується не тільки Інтернет, але й Інтранет, який використовує той самий набір протоколів.

6.1. Протоколи мереж TCP/IP для мультимедіа

Мережевий протокол – базований на стандартах набір правил, який визначає принципи взаємодії програм, вузлів (хостів) та систем у комп'ютерних мережах.

Протоколи – це правила, що керують обміном даними у мережах. Концептуально вони організовані в *рівні*, які розташовані один над одним. Протоколи кожного рівня реалізовано з використанням протоколів нижчого рівня. Ключовими протоколами мереж TCP/IP, які й визначають загальну назву, є мережевий і транспортний протоколи – IP та TCP відповідно.

IP є протоколом Інтернету як за назвою (Internet Protocol – IP), так і в більш фундаментальному сенсі: саме завдяки йому можливий Інтернет як глобальна мережа мереж. IP визначає основну одиницю передавання, яку називають *дейта-*

грамою, і забезпечує механізм її доставлення від джерела до пункту призначення, при цьому навіть не гарантуючи успіху. Для його досягнення поверх рівня протоколу IP поміщено транспортний протокол *TCP* (Transmission Control Protocol – протокол керування передаванням), який забезпечує надійне доставляння послідовних пакетів.

Проте для деяких мережевих мультимедійних застосунків можливість втрати пакетів прийнятніша, ніж накладні витрати від надійної роботи TCP. Основними прикладами є потокове відео та аудіо. Візьмемо, наприклад, відеотрансляцію через мережу у режимі "прямого ефіру". Якби відеодані відсилали з використанням TCP, то всі кадри надійшли б повністю й у правильній послідовності, але час їхнього прибуття був би непередбачуваним, практично неминуче зображення безнадійне відставало б від реального доставлення звукового супроводу. Короткочасні перешкоди, зумовлені епізодичною втратою кадру або фрагмента аудіо, були б у цій ситуації менш помітними, аби тільки забезпечували загалом постійну частоту зміни кадрів. Отже, для передавання даних з подібними характеристиками потрібна альтернатива TCP.

6.1.1. Протоколи UDP та RTP

Протокол дейтаграм користувача UDP (User Datagram Protocol), як і TCP, побудований над рівнем IP, але він набагато простіший і робить лише трохи більше, ніж просто забезпечує передавання дейтаграм після їхнього прибуття на цільовий хост потрібним застосункам. UDP, подібно до IP, намагається лише доставити дейтаграми, не гарантуючи надійного доставлення, яке може забезпечити TCP. Протокол UDP виконує елементарну перевірку на наявність помилок, якої не робить IP, допомагаючи перевірити цілісність пакета. Ці особливості UDP роблять його придатною основою для побудови протоколів доставлення таких даних, як потокове відео та аудіо, для яких обмеження реального часу важливіші, ніж цілком надійне доставляння.

Однак для досягнення поставленої мети самого протоколу UDP недостатньо. Над ним зазвичай запускають *транспортний протокол реального часу RTP* (Real Time Transport Protocol), який додає кілька функцій, необхідних для синхронізації, упорядкування та ідентифікації різних типів даних. Сам по собі протокол RTP також не гарантує доставлення; він допускає прибуття пакетів не один за одним, але дає можливість адресатам визначати потрібну послідовність пакетів і виявляти нестачу, якщо деякі з них відсутні. RTP встановлює з'єднання між застосунками, тому потік даних, що належать певному з'єднанню, можна ідентифікувати. Отож програма-адресат може відновити правильну послідовність пакетів і помістити їх у відповідне місце.

Заголовок пакета RTP також ідентифікує тип корисного навантаження (відео, аудіо тощо), який побічно визначає формат даних, що містяться в тілі пакета. Якщо передається кілька різних типів інформації (наприклад, відео із супровідним звуком), протокол відішле їх в окремих потоках RTP, а отже, потім буде необхідно

синхронізувати після одержання. Для цього в заголовок включена часова мітка, у якій записано момент дискретизації першого байта вмісту пакета. Ці мітки можна зіставляти з мітками інших пов'язаних потоків, щоб гарантувати, що одночасно дискретизовані дані відтворюватимуться одночасно.

6.1.2. Протокол RTSP для мультимедійних застосунків

Мережевий і транспортний протоколи відповідають тільки за доставлення пакетів даних зазначеному адресатові. Над ними повинні бути запущені протоколи вищих рівнів, які пропонують послуги, що підходять для мультимедійних застосунків. Зокрема, протокол HTTP, що є основою Всесвітньої павутини, пропонує служби, необхідні для реалізації розподілених систем гіпертексту. HTTP може працювати із вбудованими зображеннями, звуками та відео, завантажуючи повний файл даних. Однак потокове аудіо і відео потребують інших підходів.

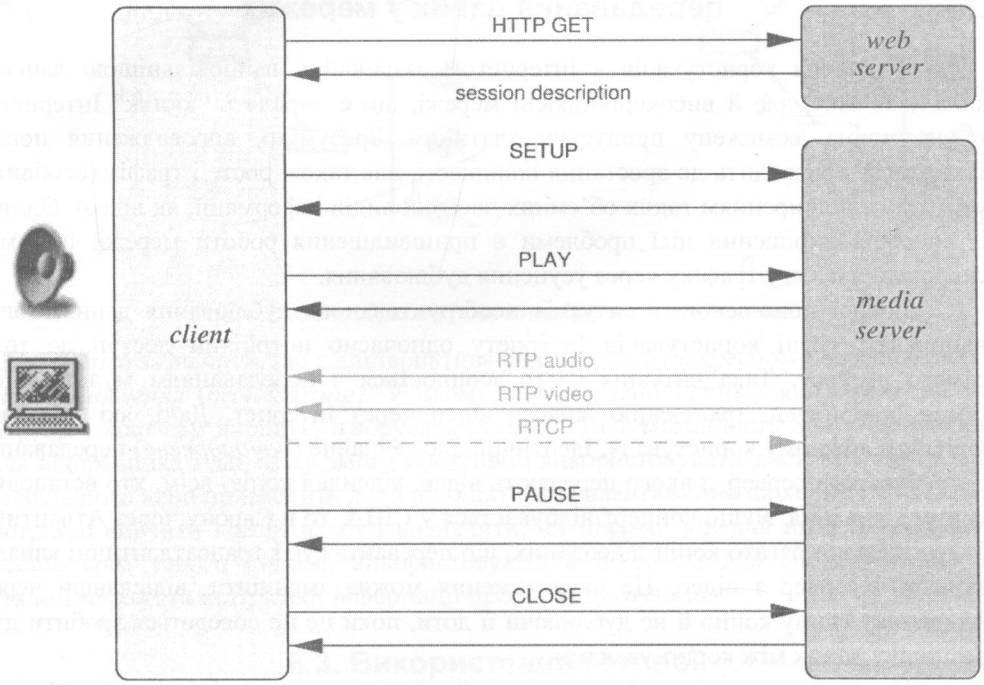


Рис. 6.1. Послуги, надавані протоколами HTTP (організація сеансу), RTP (доставлення мультимедіа) та RTSP (налаштування, запуск, зупинка і завершення відтворення)

Передусім слід пам'ятати, що HTTP запущено над TCP, витрати якого на надійне доставляння є неприйнятними для потокових засобів інформації, які краще обслуговувати менш надійним, але ефективнішим протоколом. Використання TCP також робить HTTP не придатним для багатоадресного передавання. Для роботи з

потоковими даними зручно використовувати описаний раніше протокол RTP, що підходить для групового передавання, але не надає всіх необхідних функціональних можливостей, необхідних потокам мультимедійних даних. Зазвичай бажано мати можливість запустити, зупинити і призупинити відтворення, а також (для потоків, які не передаються "наживо") перейти в деяку часову точку потоку. Всі зазначені послуги надає протокол *RTSP* (Real Time Streaming Protocol – протокол потокового передавання в реальному часі, рис. 6.1). Синтаксично він дуже близький до HTTP, має рядки запиту і стану, заголовки, багато з яких аналогічні використовуваним у HTTP, однак є й певні відмінності.

Потокові дані протокол RTSP доставляє за допомогою RTP, запущеному над UDP, а в інших випадках запускається над TCP, на надійне доставлення якого він покладається.

6.2. Одно- та багатоадресне передавання даних у мережах

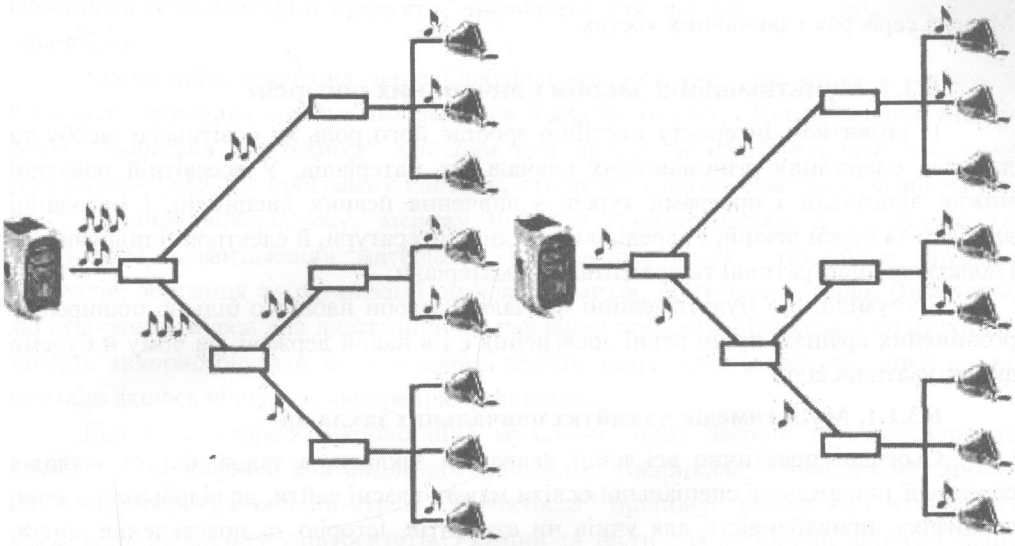
З'єднання користувачів з Інтернетом зазвичай є найповільнішою ланкою каналу зв'язку, але й високошвидкісні мережі, що становлять "кістяк" Інтернету, також мають обмежену пропускну здатність. Зрозуміло, впровадження нових технологій призводить до зростання швидкості, але також росте і трафік (особливо у зв'язку з поширенням таких об'ємних часових видів інформації, як відео). Одним із способів вирішення цієї проблеми є пришвидшення роботи мережі, іншим – скорочення обсягу трафіку через усунення дублювання.

Доволі поширеною є ситуація необґрунтованого дублювання даних, коли, наприклад, групі користувачів Інтернету одночасно потрібний доступ до того самого ресурсу. Така ситуація часто асоціюється з передаванням мультимедіа. Взяти, наприклад, трансляцію живого відео через Інтернет. Дані, які потрібно відіслати кожному користувачу, ідентичні, але звичайне (*одноадресне*) передавання потребує, щоб сервер, з якого передають відео, відсилав копію всім, хто встановив із ним з'єднання. Якщо концерт відбувається у США, то в Європу через Атлантику буде відіслано багато копій відеоданих, що перевантажує і трансатлантичні канали зв'язку, і сервер з відео. Це навантаження можна зменшити, відіславши через Атлантику єдину копію й не дублюючи її доти, поки це не доведеться зробити для розподілу даних між користувачами.

У такому сценарії відображено суть *багатоадресного передавання* (*групового передавання,* або *мультикасту*, англ. *multicast*). На рис. 6.2 продемонстровано різницю між одно- і багатоадресним передаванням.

У першому випадку окремий пакет передається кожному користувачеві; у другому – передається єдиний пакет, що дублюється у кожному випадку розгалуження шляху до різних користувачів. Щоб такий сценарій був можливим, хости необхідно організовувати в *групи хостів,* у яких певні мережеві адреси визначають не окремі хости, а власне їхні групи. Пакет, призначений для групового

передавання, відсилають на підхожу групову адресу. Маршрутизатори повинні вміти працювати із багатоадресними пакетами, дублюючи їх у разі необхідності, а також розраховувати оптимальні маршрути (які можуть відрізнятися від оптимальних маршрутів одноадресного передавання). Усі ці питання, хоча і є складними, мають технічні розв'язання.



а *б*

Рис. 6.2. Одноадресне і багатоадресне передавання

Варто зазначити, що є альтернатива одно- та багатоадресному передаванню – *широкомовлення* (*broadcasting*). У цьому випадку дані відправляють усім хостам мережі, і хост сам вирішує, ігнорувати їх чи ні. Така можливість дуже приваблива для відправника і дає йому змогу ефективно використовувати доступну смугу, але здебільшого вона призводить до неприйнятного навантаження на хости, оскільки їм потрібно вивчити кожен пакет і визначити, чи цікавий він для них. Широкомовлення, крім усього іншого, використовують в Інтернеті для спостереження за станом мережі та підтримки інформації про маршрутизацію.

6.3. Використання готової мультимедійної продукції з Інтернету

Ще донедавна основним видом інформації в Інтернеті був текст, а користувачі активно застосовували надавані браузерами можливості відключення завантаження графіки та анімації на веб-сторінках. Сьогодні ж глобальну мережу неможливо уявити без найширшого використання мультимедіа.

Сучасні технології дають змогу в онлайновому режимі прослуховувати аудіозаписи, переглядати музичні кліпи та фільми, грати в комп'ютерні ігри разом з

партнерами з різних куточків світу, навчатися за допомогою віддалених інтерактивних засобів та іншими способами використовувати мультимедіа. Усе доступнішими стають участь у відеоконференціях, перегляд телепрограм у прямому ефірі, спостереження за "живими" подіями у реальному часі за допомогою веб-камер тощо. Однак найпоширенішим поки що є (і, напевно, буде ще тривалий час) використання вже готових мультимедійних файлів, які зберігаються на підключених до Мережі серверах і звичайних хостах.

6.3.1. Мультимедійні засоби навчальних ресурсів

Із розвитком Інтернету постійно зростає його роль як освітнього засобу та джерела одержання різноманітних навчальних матеріалів. У всесвітній павутині можна відшукати і програми курсів з вивчення певних дисциплін, і методичні вказівки та курси лекцій, і перелік відповідної літератури, й електронні підручники, і додаткові ілюстративні та презентаційні матеріали.

Зрозуміло, що мультимедійні навчальні засоби набагато більше поширені у розвинених країнах, проте певні досягнення є і в нашій державі, на чому й будемо акцентувати надалі.

6.3.1.1. Мультимедіа на сайтах навчальних закладів

Сьогодні практично всі вищі навчальні заклади, а також багато закладів середньої та середньої спеціальної освіти мають власні сайти, де відображено їхню специфіку, привабливість для учнів чи студентів, історію та повсякденне життя. Сайти ВНЗ поступово стають одним із головних джерел інформації для абітурієнтів, адже тут зазвичай подають відомості про факультети та їхні навчальні підрозділи, перелік спеціальностей, за якими виш готує фахівців різної кваліфікації та освітнього рівня, іноді навіть курси, які потрібно прослухати для одержання певної спеціальності, із зазначенням їхньої тривалості та методів оцінювання.

Найпрестижніші університети вже давно намагаються зацікавити вступників за допомогою відеопрезентацій, у яких повідомляють важливу інформацію про себе у динамічній візуальній формі (варто, однак, зазначити, що сьогодні навчальні заклади більшість своїх відеопрезентацій, як і інших відеоматеріалів, розміщують на популярних інтернет-відеосервісах, зазвичай YouTube, а на власних сайтах дають лише посилання на ці матеріали). Все звичнішим стає спілкування представників вишів із майбутніми абітурієнтами у формі днів знайомства в онлайн-режимі за допомогою інструментів, які надають ті самі відеосервіси.

У багатьох вишах поряд із традиційними фотогалереями подають відеогалереї, у яких можна переглянути відеорепортажі про важливі для закладу події. Поступово входить у вжиток виставлення на сайтах ВНЗ відеолекцій, які потім можна переглянути у зручний для себе час.

6.3.1.2. Мультимедійні засоби дистанційного навчання в Інтернеті

Хоча початково важливим елементом дистанційного навчання (яке тоді ще називали просто заочним) була звичайна пошта, віддалене вивчення різноманітних

предметів за допомогою засобів Інтернету використовують уже доволі тривалий час. Суть його полягає в тому, що користувач реєструється на відповідному сайті та одержує доступ до навчальних матеріалів певного навчального курсу, які він повинен вивчити, а в кінці (чи після вивчення окремих розділів) пройти тестування щодо оволодіння необхідними знаннями. Останнім часом усе більшої популярності набувають мультимедійні продукти, призначені для онлайнового вивчення різних дисциплін.

Дистанційне навчання через Глобальну мережу сьогодні використовують у більшості провідних навчальних закладів України та світу. Національний університет "Львівська політехніка" здійснює цей процес через *Віртуальне навчальне середовище* (ВНС) – програмну систему, створену з наголосом са́ме на навчання. ВНС зазвичай використовує мережу Інтернет і надає засоби для оцінювання, комунікації, завантаження матеріалів, оцінювання колег, управління групами студентів, збирання та організації оцінок студентів, опитування тощо. Викладачі, які створюють курси для дистанційного навчання студентів Львівської політехніки, можуть використовувати мультимедійні засоби, однак слід зазначити, поки що це не стало якоюсь відчутно поширеною практикою.

Більш популярне використання мультимедійних засобів у новостворених сучасних платформах дистанційного навчання. Наприклад, громадський проект масових відкритих онлайн-курсів "Prometheus" пропонує відеолекції найкращих викладачів провідних університетів України та тести для перевірки набутих знань за допомогою різноманітних інтерактивних завдань. Український освітній проект для створення онлайн-курсів Educational Era надає, зокрема, через сервіс EdEra Books доступ до онлайн-книг, які складаються з ілюстрованих текстів з інтегрованими відео та візуалізаціями, вбудованими тестами і тренажерами для перевірки здобутих знань, а також до розширених онлайн-курсів.

Такі відеоматеріали можуть бути інтерактивними: реагувати на дії користувача, повідомляти про його вдалі чи невдалі дії, давати підказки, зокрема й у формі відеороликів, оцінювати процес виконання завдання тощо. Все це дає змогу поєднати різні методи сприйняття інформації і підвищити ефективність навчання.

Інші загалом значно потужніші можливості для вивчення конкретних процесів та явищ надають комп'ютерні **симулятори**.

Симулятор – це програмний навчальний засіб, який моделює (симулює) певну реальну або навчальну ситуацію (явище природи, фізичний експеримент або дослід, технічний або технологічний процес тощо), демонструє приклади фізичних явищ у природі та техніці, надає засоби для керування і контролю за цими явищами та процесами.

Сьогодні симулятори доволі активно використовують у дистанційному навчанні для моделювання фізичних експериментів, технічних і технологічних процесів, економічних та фінансових операцій тощо.

Симулятори пропонують користувачам наближене до реальності віртуальне подання об'єктів або процесів за допомогою графічних та мультимедійних засобів. Ретельно спроектовані та реалізовані комп'ютерні симулятори високого рівня можна використати як професійні тренажери, однак для цього переважно потрібно спеціалізоване обладнання.

6.3.2. Відпочинок та розваги онлайн

Користувачі Інтернету віддавна використовують його для того, щоб цікаво провести дозвілля. Спочатку вони шукали тексти і графіку, бавилися у примітивні ігри. Зі зростанням пропускної здатності ліній зв'язку в Мережі почало з'являтися все більше музичних записів і кліпів, які стало можливим завантажувати до себе або й переглядати в режимі онлайн. Сьогодні підключений до Всесвітньої павутини комп'ютер може бути водночас радіоприймачем, телевізором, домашнім кінотеатром, ігровою приставкою та багато чим іншим.

6.3.2.1. Музика та фільми у Глобальній мережі

Ще донедавна одними із найпопулярніших занять в Інтернеті були пошук і прослуховування аудіозаписів, переважно у форматі MP3. Сьогодні також будь-хто може відшукати потрібну пісню, музичний твір чи цілий концерт і прослухати їх або записати на свій комп'ютер. Однак тепер частіше шукають не самі аудіозаписи, а кліпи на окремі пісні або відеозаписи концертів. При цьому слід зазначити, що значна (якщо не більша) частина музики розміщена в Мережі нелегально, з порушенням авторських прав виконавців. Водночас багато файлів розміщують початківці, які використовують Всесвітню павутину як дешевий спосіб легального поширення своїх музичних записів.

Сьогодні завдяки високій швидкості сучасного Інтернету найпопулярнішим видом мультимедійної продукції у Всесвітній павутині стало відео. З'явився спеціальний сервіс **фільми онлайн** (Internet Video on Demand, iVoD (англ) – інтернет-відео за запитом), що полягає у наданні користувачам можливості перегляду кіно-, теле- та відеофільмів через Інтернет у режимі онлайн за допомогою браузера. Такий спосіб дає змогу дивитись відео без потреби використання електронних носіїв (CD чи DVD) або завантаження файла на комп'ютер. Користувач заходить на відповідний сайт, відшукує потрібний фільм і, часто навіть не реєструючись, просто розпочинає перегляд.

Сайти із пропозиціями фільмів онлайн стали з'являтися ще у 2000-ті роки, коли компанія Macromedia почала експериментувати з відео у популярному програвачі Macromedia Flash Player. Першим і досі найпопулярнішим відеосервісом став YouTube.

YouTube – інтернет-служба, що надає послуги розміщення відеоматеріалів звичайним користувачам, які можуть додавати, переглядати і коментувати відеозаписи, а подекуди навіть редагувати їх.

Завдяки простоті та зручності використання YouTube став одним із найпопулярніших місць для розміщення відеофайлів. Від жовтня 2010 року сервіс надає україномовний користувацький інтерфейс. Від січня 2015 року YouTube майже повністю перейшов на HTML5-плеєр. У 2016 році щоденна кількість переглядів відео на сайті становила майже 5 мільярдів.

В Україні порівняно з розвинутими країнами мережеві відеосервіси стали популярними набагато пізніше, що було пов'язано з повільним зростанням швидкості Інтернету для приватних користувачів. Зокрема, перші україномовні сайти з фільмами онлайн з'явилися у 2008 році й відразу стали дуже популярними. Однак, як і у випадку з музикою, на сайтах сервісу часто розміщують нелегальні, "піратські" копії фільмів, що порушує авторські права власників продукції. Через це владні структури різних країн намагаються боротися з "відеопіратами", щоправда, не завжди успішно, про що свідчить закриття на початку 2012 року популярного українського файлообмінника "Експрес файли" (http://www.ex.ua/), подальше швидке його відкриття і нове закриття наприкінці 2016 року. Проте сьогодні в Україні є вже достатньо подібних служб, і закриття конкурентів лише призводить до зростання популярності тих відеосервісів, які працюють.

6.3.2.2. Віртуальні мандрівки

Багато людей, особливо з розвиненої частини світу, люблять під час відпусток подорожувати іншими містами та країнами. Однак не завжди є можливість побувати там, де хочеш, та й цікавих місць на Землі надто багато, щоб встигнути побувати всюди. Та сьогодні завдяки спеціальним інтернет-сервісам можна відвідати далекі міста, знамениті музеї, пам'ятники архітектури мистецтва, не виходячи з дому.

Щороку в Інтернеті з'являється все більше різних онлайнових сервісів, що дають змогу побувати всередині багатьох музеїв, галерей мистецтв, соборів і храмів. Такі віртуальні екскурсії, котрі можна здійснювати не лише з комп'ютера, але й зі смартфона, не потребують великих грошових витрат, які неминучі під час реальних мандрівок по світу. До того ж вони дають змогу побачити експонати, що зберігаються в запасниках, приміщення, які в реальному музеї відвідувачам недоступні, а панорами, виконані з високою роздільністю, відображають усі тонкощі світових шедеврів. Здебільшого такі екскурсії реалізовані у формі *3D-турів*.

3D-тур – інтерактивне відображення за допомогою спеціальних програмних засобів приміщень, їхніх комплексів або певної місцевості, яке дає змогу створити ефект присутності користувача всередині відображуваного об'єкта, переміщуватися по ньому та переходити до інших об'єктів.

Віртуальний 3D-тур організовують переважно як послідовність сферичних панорамних фотографій, об'єднаних між собою інтерактивними посилання-переходами (*хотспотами*). Сюди зазвичай вводять також інші інтерактивні елементи: спливаючі інформаційні вікна, пояснювальні надписи, графічно оформ-

лені клавіші керування тощо, а подекуди ще й звукові доріжки, звичайні фотогалереї, рекламні ролики та інше.

Дуже корисними як для віртуальних, так і для реальних мандрівників є веб-сервіси, які дають змогу здійснювати тривимірні подорожі різними містами світу. Скориставшись ними, можна не лише "прогулятися" вулицями віддалених та екзотичних міст, зазирнути в усі відомі та маловідомі куточки, але й підготуватися до справжньої мандрівки, намітити реальні орієнтири, щоб потім не заблукати у незнайомому місці. Однією з таких служб є широко відомі та загальнодоступні *Карти Google*. Можливість "поблукати" в тривимірній проекції вулиць через Інтернет у вікні звичайного браузера надає користувачам технологія WebGL (веб-Graphics Library – Інтернет-бібліотека графічних функцій). За її допомогою реалізовано плавні переходи між зображеннями під час переміщення карти або зміни масштабу відображення, 3D-перегляд будівель і панорам безпосередньо в Картах Google, плавне масштабування і вигляд під різними кутами з можливістю обертання камери, швидке перемикання між картою і переглядом вулиць.

6.4. "Живі" інтернет-радіо, телебачення і відео

З розвитком новітніх технологій широкомовлення і завдяки масовому впровадженню високошвидкісного доступу до Інтернету все популярнішим стає використання Глобальної мережі як джерела поширення мультимедійної продукції в режимі "прямого ефіру".

6.4.1. Інтернет-радіо та Інтернет-телебачення

Першим широкої популярності набуло *інтернет-радіо* – завдяки порівняно низьким вимогам до швидкості передавання даних.

> *Інтернет-радіо (веб-радіо)* – група технологій передавання потокових аудіоданих через мережу Інтернет, а також радіостанція, що використовує для мовлення технологію потокового мовлення в Інтернеті.

У технологічну основу інтернет-радіо покладено три елементи:

- **станція** – генерує потік аудіоданих (або із списку звукових файлів, або прямим оцифруванням з аудіокарти, або копіюючи наявний у мережі потік) і спрямовує його до сервера;
- **сервер** (повторювач потоку) – приймає аудіопотік від станції і переспрямовує його копії усім підключеним до сервера клієнтам;
- **клієнт** – приймає потік від сервера і перетворює його на аудіосигнал, який і чує слухач інтернет-радіостанції.

Як клієнт можна використовувати будь-який мультимедійний програвач, що підтримує потокове аудіо і здатний декодувати формат, в якому передає дані радіостанція, а також звичайний браузер. Разом із потоком звукових даних зазвичай

передають також текстові дані для відображення інформації про станцію і поточну композицію.

Інтернет-телебачення ще донедавна здавалося людям, не пов'язаним професійно з ІТ-сферою, дивиною, справою віддаленого майбутнього, однак сьогодні воно набирає колосальних масштабів.

Інтернет-телебачення (телебачення міжмережевого протоколу або *онлайн-TV)* – система, основана на двосторонньому цифровому передаванні телевізійного сигналу через інтернет-з'єднання за допомогою широкосмугового підключення.

Зараз на сайті практично кожного телеканалу є розділ онлайн-трансляції, звідки можна переглядати поточні передачі каналу, а також архіви із записами найпопулярніших передач, фільмів та серіалів. Крім того, у Мережі є величезна кількість сайтів із переліком телеканалів багатьох країн світу, звідки також можна дивитися передачі в режимі онлайн.

6.4.2. IP-телебачення

Дещо інші можливості надає *IP-телебачення*, яке часом необґрунтовано ототожнюють із інтернет-телебаченням.

IP-телебачення (IPTV, IP-TV, англ. Internet Protocol Television) – технологія цифрового інтерактивного телебачення в мережах передавання даних за протоколом IP, нове покоління телебачення.

IP-телебачення, як не парадоксально, – це не телебачення, яке здійснює мовлення через Інтернет. Хоча скорочення IP також походить від "Internet Protocol", це не означає, що можна зайти на потрібну веб-сторінку, щоб подивитися потрібну телепередачу, – тут IP означає лише метод передавання інформації через захищену керовану мережу. Практично ж функціонування IP-телебачення ґрунтується на використанні кількох протоколів: HTTP – для організації інтерактивних сервісів, RTSP – для керування потоками мовлення, UDP та RTP – для доставлення потокового відео, IGMP – для керування мультикаст-потоками.

IPTV – це платформа, яку створює і контролює оператор – постачальник телекомунікаційних структур, з яким безпосередньо взаємодіє споживач. IP-телебачення пропонує такий самий відеопродукт, як і кабельні та супутникові операторами. Головними його перевагами є інтерактивність відеопослуг і наявність широкого набору додаткових сервісів. Під інтерактивністю в цьому випадку розуміють наявність зворотного зв'язку з глядачами, які можуть не тільки самостійно вибирати необхідні пакети каналів, але й формувати "запити на відео" (тобто перегляд відео з електронних каталогів), здійснювати запис майбутніх передач, "відкладений перегляд" (тобто перегляд будь-якого контенту у зручний час) тощо.

Мережа IP-телебачення складається з головної станції, телекомунікаційних мереж, проміжних і приймальних абонентських пристроїв. Головна станція отримує, формує і транслює в мережу відеопотоки контенту, джерелами якого є

супутникові телевізійні канали, аналогове і цифрове відео тощо. Приймати і переглядати програми IP-телебачення можна через комп'ютер (тоді потрібно встановити спеціальну програму-плеєр) або на екрані телевізора. Для старіших моделей телеприймачів необхідно використовувати спеціальні приставки-ресивери, які декодують отриманий відеопотік і виводять його на екран, однак сьогодні більшість телевізорів останнього покоління самостійно підтримують IP-телебачення.

6.4.3. Веб-камери

Ще одним джерелом живих відеозображень в Інтернеті є *веб-камери*.

Веб-камера – невелика цифрова відеокамера, здатна в режимі реального часу фіксувати зображення, призначені для подальшого передавання через мережу Інтернет.

Веб-камери завантажують зображення на веб-сервер або за запитом, або неперервно, або через регулярні проміжки часу. Деякі сучасні моделі мають апаратне і програмне забезпечення, яке дає змогу камері самостійно працювати як веб-сервер, FTP-сервер, FTP-клієнт або відсилати зображення електронною поштою. Такі пристрої, що мають власні IP-адреси, називають IP-камерами.

Стаціонарні веб-камери зазвичай встановлюють у певних місцях: на популярних туристичних маршрутах, на відкритих ландшафтах, на вулицях та площах міст, всередині приміщень тощо. Часто веб-камери використовують для демонстрації якості або умов надаваного комерційного сервісу. Деякими веб-камерами можна віддалено керувати, наприклад, за допомогою кнопок навігації на веб-сторінці – повернути камеру праворуч або ліворуч, змінити кут нахилу, щоб краще розглянути місце зйомки. Є веб-камери, на сторінках яких можна керувати не самою камерою, а пристроєм, який вона показує.

6.5. Мультимедіа та пірингові мережі

Усім відома традиційна *мережева архітектура клієнт-сервер*, у якій лише окрема категорія учасників, які називають серверами, може надавати певні послуги іншим. Однак є й інші типи мереж, які називають *одноранговими*, *децентралізованими* або *пірингові* (від англ. *peer-to-peer, P2P* – рівний до рівного), що базуються на рівноправності своїх учасників. На відміну від архітектури клієнт-сервер, така організація дає змогу зберігати працездатність мережі за будь-якої кількості та будь-якого поєднання доступних вузлів.

6.5.1. Пірингові мережі

Піринг – варіант архітектури системи, в основу якої покладено мережу рівноправних вузлів, що можуть зв'язуватися між собою та обмінюватися даними, де відсутні виділені сервери, а кожен вузол (пір) є одночасно клієнтом і сервером.

У "чистій" піринговій мережі немає поняття клієнтів або серверів, лише рівні вузли, які одночасно функціонують як клієнти та сервери стосовно інших вузлів мережі. Проте використовують також і P2P-мережі, що все ж мають сервери, однак їхня роль полягає лише у координації роботи, пошуку та підтримці інформації щодо сервісів, які надають клієнти мережі.

Одна з найуспішніших галузей застосування технології піринговых мереж – це обмін файлами. Користувачі файлообмінної мережі викладають які-небудь файли у спеціальний каталог, вміст якого доступний для завантаження іншим користувачам. Якщо хтось із них надсилає запит на пошук якого-небудь файла, програма шукає його у клієнтів мережі, показує результат, і користувач може завантажити файл зі знайдених джерел. У сучасних піринговых мережах інформація завантажується відразу з кількох джерел.

Сьогодні найпопулярнішими файлообмінними мережами, що ґрунтуються на технології пірингу, є I2P, Direct Connect, Gnutella та BitTorrent.

6.5.2. Протокол BitTorrent

У мережі BitTorrent (або торрент-мережі) для обміну інформацією використовують відкритий протокол BitTorrent. Його розробляли так, щоби обмін файлами великих розмірів у мережі був полегшений для її учасників. Зокрема, клієнти, які завантажили певну частину великого файла, відразу можуть бути джерелами його розповсюдження.

На практиці для обміну файлами за протоколом BitTorrent використовують спеціальне програмне забезпечення – *BitTorrent-клієнти*, зокрема офіційний клієнт BitTorrent та популярний серед користувачів μTorrent, які в останніх версіях стали фактично однією програмою. Хоча для обміну файлами у торрент-мережах необов'язково користуватися клієнтами: вже успішно функціонують сервіси, які дають змогу завантажувати файли за допомогою самого лише браузера.

6.6. Контрольні питання

1. Що таке мережевий протокол?
2. Які протоколи мереж TCP/IP використовують для роботи з мультимедіа?
3. Яка різниця у функціонуванні протоколів RTP та RTSP?
4. Поясніть суть одноадресного і багатоадресного передавання.
5. Опишіть схему зв'язку між протоколами у сфері мультимедіа.
6. Назвіть два загальні способи використання готової мультимедійної продукції з Інтернету.
7. Які мультимедійні засоби використовують для дистанційного навчання в Інтернеті?
8. Назвіть та поясніть призначення трьох технологічних складових інтернет-радіо.
9. Що є спільне та відмінне між інтернет-телебаченням та IP-телебаченням?
10. У чому особливості піринговых мереж із погляду архітектури?

6.7. Приклади тестових питань

1. Поняття IP-телебачення:

   а) це цифрове телебачення, яке здійснює мовлення через Інтернет;
   б) це цифрове інтерактивне телебачення в захищених керованих мережах;
   в) це система трансляції цифрового телебачення з високою роздільністю.

j) Поняття інтернет-радіо:

   а) це технологія передавання потокових аудіоданих через Інтернет;
   б) це технологія передавання потокових аудіоданих у широкомовних IP-мережах;
   в) це цифрове інтерактивне радіомовлення в захищених керованих мережах.

k) Поняття комп'ютерного симулятора:

   а) це програмований електронний апарат для керування реальним пристроєм;
   б) це програмний засіб для керування реальним пристроєм;
   в) це програмний засіб для моделювання явищ, дослідів, процесів.

l) Поняття мережевого протоколу:

   а) він визначає правила обміну даними у мережах;
   б) він визначає принципи взаємодії між елементами мультимедіа;
   в) він базується на мережевих форматах і стандартах.

m) Властивості 3D-туру:

   а) він дає змогу досліджувати відображуваний об'єкт іззовні;
   б) він створює ефект присутності всередині відображуваного об'єкта;
   в) він створює відчуття реального перебування всередині відображуваного об'єкта.

n) Властивості протоколу RTP:

   а) його використовують для доставлення потокових даних;
   б) його використовують для надійного доставлення даних;
   в) під час своєї роботи він використовує протокол TCP;
   г) під час своєї роботи він не використовує інших протоколів.

o) Властивості протоколу RTSP:

   а) його використовують для доставлення пакетів даних;
   б) його використовують для надійного доставлення потокових даних;
   в) його використовують для керування відтворенням потокових даних;
   г) у своїй роботі він не використовує інших протоколів.

p) Властивості протоколу UDP:

   а) його використовують для надійного доставлення потокових даних;

б) його використовують для керування відтворенням потокових даних;
в) протокол UDP запускають над рівнем протоколу IP;
г) протокол UDP запускають над рівнем протоколу TCP.

ґ) Протоколи для роботи з потоковими даними:

а) FTP;
б) TCP;
в) SMTP;
г) RTP.

г) Особливості піринговых мереж:

а) вони використовують мережеву архітектуру клієнт-сервер;
б) вони базуються на рівноправності своїх учасників;
в) пір – це протокол обміну даними у таких мережах;
г) у них файли завантажуються на комп'ютер користувача із трекерів.

СПИСОК ЛІТЕРАТУРИ

1. Гасов В. М. Трехмерная графика в медиаиндустрии: учебник / В. М. Гасов, А. М. Цыганенко. – М. : Московский гос. ун-т печати, 2010. – 522 с.

2. Глушаков С. Цифровое видео и аудио. Секреты обработки на ПК / С. Глушаков, А. Харьковский. – М. : АСТ, 2008. – 320 с.

3. Михайленко В. Є. Інженерна та комп'ютерна графіка : підручник для студентів ВНЗ / В. Є. Михайленко, В. В. Ванін, С. М. Ковальов ; за ред. В. Є. Михайленка. – К. : Каравела, 2012. – 368 с.

4. Немцова Т. И. Компьютерная графика и Web-дизайн. Практикум по информатике : учеб. пособие / Т. И. Немцова, Ю. В. Назарова. – М. : ИД "Форум", Инфра-М, 2010. – 288 с.

5. Пилгрим М. Погружение в HTML5 / М. Пилгрим. – СПб. : БХВ-Петербург, 2011. – 304 с.

6. Пічугін М. Ф. Комп'ютерна графіка : навч. посібник / М. Ф. Пічугін, І. О. Канкін, В. В. Воротніков. – К. : Центр учбової літератури, 2013. – 346 с.

7. Різник О. Я. Основи комп'ютерної графіки : курс лекцій / О. Я. Різник. – Львів : Видавництво Львівської політехніки, 2012. – 220 с.

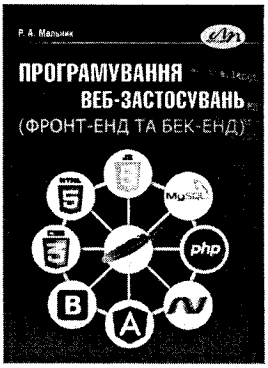
8. Роджерс Д. Математические основы машинной графики : пер. с англ. / Д. Роджерс, Дж. Адамс. – М. : Мир, 2001. – 604 с.

9. Хантер Д. Работа с XML, 4-е изд. / Д. Хантер, Д. Рафтер, Д. Фаусетт и др. – М. : Диалектика, 2009. – 1344 с.

10. Херн Д. Компьютерная графика и стандарт OpenGL. 3-е издание : Пер. с англ. / Д. Херн, П. М. Бейкер. – М. : Изд. дом "Вильямс", 2005. – 1168 с.

11. Чепмен Н. Цифровые технологии мультимедиа. – 2-е изд. / Чепмен Н., Чепмен Дж. – М. : Изд. дом Вильямс, 2006. – 624 с.

Книги для навчання і роботи!



Мельник Р. А.

ПРОГРАМУВАННЯ ВЕБ-ЗАСТОСУВАНЬ (ФРОНТ-ЕНД ТА БЕК-ЕНД)

Навчальний посібник. – 2018. – 248 с.
ISBN 978-966-941-195-2

Викладено матеріал для проектування динамічних web-сторінок. Розділи розкривають мови кодування HTML, HTML5, CSS, JQUERY, W3.CSS, Bootstrap, AngularJS, мови програмування JavaScript та PHP, технологію Ajax, зберігання даних у масивах та базах даних MySQL-сервера. Наведено приклади доступу до них з Web-сторінок. Подано основи проектування Web-сторінок у технологіях. NET та JAVA, зокрема за допомогою класів з бібліотек ASP.NET, класів побудови сервлетів javax.servlet та JSP засобів проектування динамічних web-сторінок. Усі розділи завершуються контрольними запитаннями. Матеріал містить фрагменти програм, що можуть бути корисними студентам під час самостійного розроблення власних програм до лабораторних та курсових завдань.

Для студентів ВНЗ та коледжів, які вивчають сучасні технології створення програмного забезпечення, зокрема за ефективною архітектурою "клієнт-сервер".

Левус Є. В., Мельник Н. Б.

ВСТУП ДО ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Навчальний посібник. – 2018. – 248 с.
ISBN 978-966-941-131-0

Посібник призначений для вивчення навчальної дисципліни "Вступ до інженерії програмного забезпечення". Містить теоретичні відомості, що стосуються означення інженерії програмного забезпечення, життєвого циклу програмного забезпечення, його моделей. Також подано матеріали для самостійного вивчення розділу "Групова динаміка і комунікації" навчальної дисципліни. Важливим засобом підготовки студентів до поточного й семестрового контролю є наведені у посібнику тестові завдання.

Видання призначене для студентів спеціальності "Інженерія програмного забезпечення", а також може використовуватися для інших спеціальностей галузі знань "Інформаційні технології" для вивчення дисциплін, пов'язаних з розробленням програмного забезпечення.

Видавництво Львівської політехніки
вул. Ф. Колесси, 4, корп. 23А, м. Львів, 79013
тел. +380 32 2582146, факс +380 32 2582136, http://vlp.com.ua, vmr@vlp.com.ua

НАВЧАЛЬНЕ ВИДАННЯ

Журавчак Любов Михайлівна
Левченко Олександр Миколайович

ПРОГРАМУВАННЯ КОМП'ЮТЕРНОЇ ГРАФІКИ ТА МУЛЬТИМЕДІЙНІ ЗАСОБИ

Навчальний посібник

Редактор *Ольга Дорошенко*
Коректор *Олеся Косик*
Технічний редактор *Лілія Саламін*
Комп'ютерне верстання *Наталії Максимюк*
Художник-дизайнер *Уляна Келеман*

Здано у видавництво 15.05.2018. Підписано до друку 26.12.2018.
Формат 70×100 1/16. Папір офсетний. Друк офсетний.
Умовн. друк. арк. 20,0. Обл.-вид. арк. 15,9.
Наклад 150 прим. Зам. 180736.

Видавець і виготівник: Видавництво Львівської політехніки.
Свідоцтво субєкта видавничої справи ДК № 4459 від 27.12.2012 р.

Вул. Ф. Колесси, 4, Львів, 79013
Тел. + 380 32 2582146, факс + 380 32 2582136
vlp.com.ua, ел. пошта: vmr@vlp.com.ua

ЖУРАВЧАК Любов Михайлівна
доктор технічних наук, старший науковий співробітник, професор кафедри програмного забезпечення.

Закінчила факультет прикладної математики та механіки Львівського університету імені Івана Франка 1985 року. Понад 20 років працювала у Карпатському відділенні Інституту геофізики ім. С. І. Субботіна Національної академії наук України. У 1996 році отримала спільний ґрант Міжнародної асоціації академій наук та Національної академії наук України для молодих вчених для публікації монографії за результатами досліджень, а в 2014 – премію Львівської облдержадміністрації та Львівської обласної ради для відомих учених і знаних фахівців. Автор понад 130 наукових публікацій в українських та міжнародних фахових виданнях. Читає лекції з курсів «Дослідження операцій», «Програмування мультимедійних систем», «Комп'ютерна дискретна математика».

Наукові інтереси: математичне моделювання процесів теплопровідності, геоелектромагнетизму та лінійного деформування в неоднорідних середовищах складної форми; розроблення нових чисельно-аналітичних підходів, які ґрунтуються на поєднанні методів граничних і приграничних елементів з некласичними скінченними різницями.

ЛЕВЧЕНКО Олександр Миколайович
кандидат технічних наук, асистент кафедри програмного забезпечення, провідний інженер відділу підтримки та розвитку веб-сервісів Центру інформаційного забезпечення.

Закінчив факультет прикладної математики та механіки Львівського державного університету імені Івана Франка. У 1982–2011 роках працював у ЛНУ імені Івана Франка, від 2011 року співробітник Національного університету «Львівська політехніка».

Наукові інтереси: цифрове моделювання рельєфу; методи опрацювання тексту і графіки; інтернет-технології.

Автор близько 90 наукових, науково-популярних та науково-педагогічних праць, зокрема член авторського колективу підручника для вищої школи «Інформатика. Комп'ютерна техніка. Комп'ютерні технології» (К.: Каравела, 2003, 2007, 2011, 2016), співавтор підручників з інформатики для учнів 9 та 10 класів (зокрема мовами національних меншин України), посібників з офісних програм і основ Інтернету, автор посібника «Культура роботи з текстовими документами».

Член Національної спілки письменників України від 2011 року. Автор чотирьох художніх книжок.

ISBN 978-966-941-276-8