

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ  
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

**МЕТОДИЧНІ ВКАЗІВКИ**

до виконання практичних і лабораторних робіт

**«Основи комп'ютерної графіки. Частина 2»**

з курсу «Комп'ютерна графіка»  
для студентів спеціальностей

**151 «Автоматизація та комп'ютерно-інтегровані технології»,  
172 «Телекомунікація та радіотехніка»**

Рекомендовано  
редакційно-видавничою  
радою університету,  
протокол №1 від 19.02.2020р.

Харків НТУ «ХПІ» 2020

Методичні вказівки до виконання практичних і лабораторних робіт «Основи комп'ютерної графіки. Частина 2» з курсу «Комп'ютерна графіка» для студентів спеціальностей 151 «Автоматизація та комп'ютерно-інтегровані технології», 172 «Телекомунікація та радіотехніка» денної та заочної форм навчання / уклад. А. О. Зуєв, О. М. Євсеєнко, В.А. Крилова. – Харків : НТУ «ХПІ». – 46 с.

Укладачі: А. О. Зуєв  
О. М. Євсеєнко  
В.А. Крилова

Рецензент І.В. Григоренко

Кафедра інформаційно-вимірювальних технологій і систем

## ЗМІСТ

5. РОЗРАХУНОК ОСВІТЛЕННЯ ОБ'ЄКТІВ СЦЕНИ	4
5.1 Типи джерел світла	4
5.2 Сприйняття кольору	6
5.3 Моделі освітлення	9
5.4 Задання матеріалу	12
6. АПАРАТНІ ЗАСОБИ ТРИВИМІРНОЇ ГРАФІКИ	14
6.1 Оцінка швидкодії графічних процесорів	16
7. ПРОГРАМНІ ІНТЕРФЕЙСИ ТРИВИМІРНОЇ ГРАФІКИ	19
7.1 Direct3D API	19
7.2 Використання API Direct3D	21
7.2.1 Опис моделі об'єкта в Direct3D	22
7.2.2 Вершинні і фрагментні мікропрограми на мові HLSL	25
7.3 Візуалізація з використанням Direct3D	32
8. МАТЕРІАЛИ І ТЕКСТУРУВАННЯ ПОВЕРХНІ ОБ'ЄКТІВ	34
8.1 Представлення текстур	34
Список літератури	37
Додаток А. Робота додатків в операційній системі Windows	38
Додаток Б. Довідкові відомості з мови HLSL	40
Додаток В. Мікропрограми мовою HLSL для розрахунку освітленості за правилами Ламберта і Фонга	44

## 5. РОЗРАХУНОК ОСВІТЛЕННЯ ОБ'ЄКТІВ СЦЕНИ

Важливим етапом реалістичної візуалізації є розрахунок освітленості кожної точки 3D сцени. Для того щоб провести такий розрахунок, необхідно:

- а) розмістити на сцені джерела світла і вказати їх властивості;
- б) у процесі растеризації для кожного фрагмента, який пройшов Z-тест, провести розрахунок освітлення усіма джерелами, які його освітлюють.

### 5.1 Типи джерел світла

Умовно всі джерела освітлення можна розділити на чотири типи: фонові, спрямовані, точкові і зональні.

**1) Фонове джерело (ambient light) світла.** Освітлює всі об'єкти сцени рівномірно, єдиною його характеристикою є колір освітлення. Такий тип джерел використовується для моделювання джерел непрямого розсіяного світла: світла від небозводу або відбитого від інших об'єктів.

**2) Спрямоване джерело світла (directional light).** У джерела освітлення такого типу немає місця розташування, воно випускає паралельні світлові промені в заданому напрямку  $L$ . Такий тип джерел використовується для моделювання нескінченно віддалених потужних джерел природного світла, наприклад, Сонця і Місяця. Оскільки відстань до будь-якої точки сцени від такого джерела дуже велика, можна вважати, що промені світла приходять з одного напрямку і освітлюють усі об'єкти однаково (рис. 5.1).

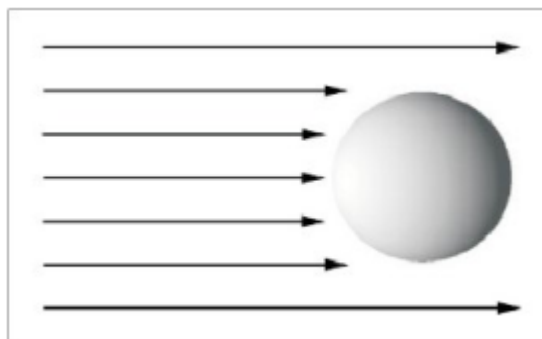


Рисунок 5.1 – Джерело спрямованого світла

**3) Точкове світло (point light).** Джерело такого типу має місце розташування в просторі сцени і випромінює світло в усіх напрямках. Таким чином, залежно від положення точки об'єкта щодо джерела, світло буде приходити з різних напрямків, тобто вектор  $L$  для кожної точки різний (рис. 5.2).

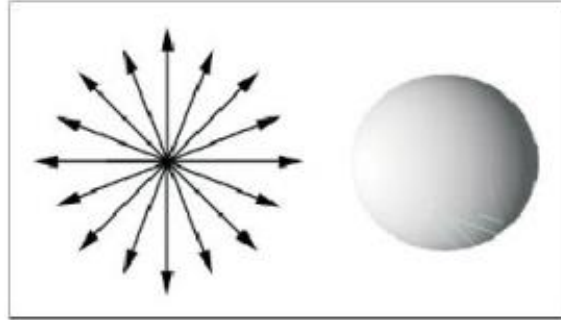


Рисунок 5.2 – Точкове джерело світла

Таке джерело має обмежений радіус дії  $R$ . Об'єкти, що знаходяться за його межами, не освітлюються. Залежно від віддаленості точки від джерела, падає інтенсивність її освітлення (від 1 до 0), зазвичай така залежність від відстані квадратична (5.1).

$$i_r = 1 - \min(1, d/R)^2 \quad (5.1)$$

Такий тип джерел використовується для моделювання локальних джерел світла обмеженої потужності, наприклад, лампочок розжарювання або ліхтарів.

**4) Зональне світло (spot light).** Джерело даного типу схоже на прожектор, у нього крім місця розташування є обмеження на напрямок випускання променів  $D$  (рис. 5.3).

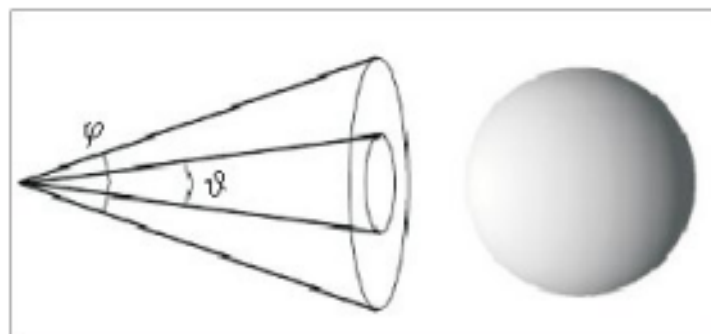


Рисунок 5.3 – Джерело зонального світла

Форма такого джерела світла являє собою світловий конус, який характеризується двома кутами:  $\varphi$  і  $\theta$ . Кут  $\varphi$  задає розмір внутрішнього конуса, а кут  $\theta$  – зовнішнього.

У межах внутрішнього конуса інтенсивність освітлення (5.2) падає тільки з відстанню, а між внутрішнім і зовнішнім конусами падає в залежності від кута. За межами зовнішнього конуса точки не освітлюються.

$$i_z = \max\left(0, 1 - \min\left(1, \frac{a - \theta}{\theta - \varphi}\right)\right) \quad (5.2)$$

Загальна інтенсивність з урахуванням дистанції  $i$  кута обчислюється за формулою (5.3).

$$i = i_z \cdot i_r \quad (5.3)$$

Таке джерело використовується для моделювання локальних спрямованих джерел світла, наприклад, ліхтариків і прожекторів.

Загальна освітленість точки розраховується як сума внесків усіх джерел світла, які її освітлюють. Задля продуктивності доцільно використовувати більш прості моделі джерел.

## 5.2 Сприйняття кольору

**Колір** – це те, про що може говорити людина, яка спостерігає навколишній світ за допомогою органів зору. Око реєструє фізичні властивості електромагнітного випромінювання, яке потрапляє на сітківку, за допомогою елементів колбочок і паличок. Поняття колір нерозривно пов'язане з біологією.

Ньютон, розклавши біле світло за допомогою призми, показав, що воно пов'язане з відносним спектральним розподілом потужності потоку випромінювання.

При цьому колір – це ще і феномен психологічний (відчуття), оскільки ми можемо говорити про колір об'єктів, ніяк безпосередньо не пов'язаний з тим, яке випромінювання прийде від об'єкта до ока.

Таким чином, колір це поняття фізичне, психологічне і біологічне.

**Константне сприйняття кольору** – здатність людини незалежно від

реєстрованого оком випромінювання (і джерела освітлення) оперувати терміном забарвлення об'єкта. Таку властивість дуже складно повторити в системах технічного зору.

**Визначення Шредингера.** Колір – це властивість спектрального складу випромінювань, спільна для всіх випромінювань, візуально не помітних для людини. Це те загальне, що є у всіх спектрів, які викликають один і той же відгук. Якби колір відповідав тільки випромінюванню, таке визначення було б вичерпним.

**Забарвлення** – об'єктивна властивість поверхні об'єкта, на відміну від кольору.

**Яскравість** – характеристика потужності випромінювання. Світлість – характеристика потужності кольору об'єкта.

Білий об'єкт може існувати, а білого випромінювання існувати не може, бо немає обмеження зверху для потужності випромінювання.

**Насиченість** – наскільки колір далекий від градацій сірого кольору. Максимальну насиченість має випромінювання лазера.

**Колірний тон** – те, що залишається від кольору після відокремлення яскравості і насиченості (веселка).

**Кольоровість** – двокомпонентна частина кольору, за винятком потужнісної характеристики (яскравості).

Якщо два забарвлення  $\Phi_1(\lambda)$  и  $\Phi_2(\lambda)$  виглядають однаково при фіксованому випромінюванні, це не означає, що при іншому випромінюванні вони теж будуть виглядати однаково. Гарантувати це можна тільки при  $\Phi_1(\lambda) = \Phi_2(\lambda)$ .

Таке явище називається метамерія – властивість зору, при якій світло різного спектрального складу може викликати відчуття однакового кольору.

**Комбінації кольорів.** На практиці неможливо отримати будь-які комбінації основних кольорів RGB, оскільки спектри реєстраторів (колбочок) пересікаються, і неможливо збудити один тип колбочок, не збудивши інший.

Таким чином, усі можливі спектри випромінювань в нескінченновимірному просторі утворюють конус, тобто коли вектор належить конусу, то помножений на невід'ємне число, він також належить конусу (рис. 5.4).

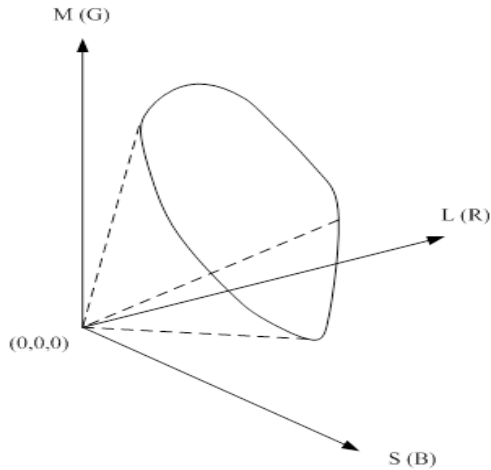


Рисунок 5.4 – Усі можливі спектри випромінювань в нескінченновимірному просторі

У просторі RGB усі допустимі реакції теж будуть утворювати конус. Конус є опуклим. Це означає, що зважена сума векторів, яка належить конусу з невід'ємними коефіцієнтами, теж належить конусу. Якщо задати площину колірності, що не проходить через 0, і за допомогою центральної проєкції спроектувати на неї конус (центр проєкції в точці чорного кольору), то всі кольори різної інтенсивності «схлопнуться» і вийде плоска фігура (рис. 5.5).

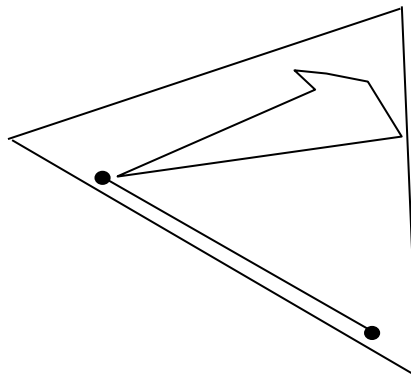


Рисунок 5.5 – Колірний "трикутник"

Ця фігура називається колірним "трикутником", у якого два кути. В одному з кутів знаходиться ультрафіолетова частина, а в іншому – інфрачервона, при цьому обидві точки знаходяться в точці проєкції. Кожній точці на дузі "трикутника" можна зіставити певну довжину хвилі.



### 5.3 Моделі освітлення

На результат освітлення, крім типу джерела, впливають також: напрямок світлових променів, позиція спостерігача і характеристики поверхні: матеріал та його кривизна. Можна сказати, що процес освітлення – це отримання кольору точки на екрані в результаті впливу на матеріал поверхні світлового потоку від джерела світла. Для моделювання такого впливу застосовуються різні більш-менш точні моделі освітлення. Таким чином, щоб провести розрахунок освітлення, необхідно:

- а) задати властивості матеріалів поверхні;
- б) розрахувати нормаль до поверхні в точці;
- в) визначити джерела світла, які освітлюють поверхню;
- г) вибрати модель розрахунку освітлення;
- д) визначити внесок кожного джерела.

Розглянемо найпростішу модель освітлення. Усе світло, яке падає на об'єкти, або поглинається або відбивається їхньою поверхнею, у другому випадку поверхня об'єкта є, по суті, вторинним джерелом відбитого світла. Таким чином, можна прийняти, що світло, яке випускається джерелами, має три складові: фонову, розсіяну і дзеркальну.

**1) Фонове світло (ambient light).** Такий тип освітлення моделює світло, яке, відбиваючись від інших поверхонь, потрапляє в точку, для якої ведеться розрахунок. Наприклад, у сцені часто висвітлені ті частини об'єктів, які не перебувають у прямій видимості джерела світла. Фонове світло визначає мінімальний рівень освітленості об'єктів сцени в тіні і характеризується тільки кольором джерела (5.4).

$$f_a = p_a, \quad (5.4)$$

де  $p_a$  – інтенсивність фонового освітлення.

**2) Розсіяне світло (diffuse light)** – пряме світло від джерела. Взаємодіючи з матеріалом поверхні, воно розсіюється рівномірно в усіх напрямках. Тому інтенсивність світла, яке досягло очей спостерігача, не залежить від точки, з якої розглядаються об'єкти сцени, і місце розташування глядача можна не враховувати. Отже, при розрахунку розсіяного освітлення треба враховувати тільки напрямок світлових променів  $L$  і нормаль до поверхні  $N$  (рис. 5.6). Це основна складова світла,

яке відбиває поверхня.

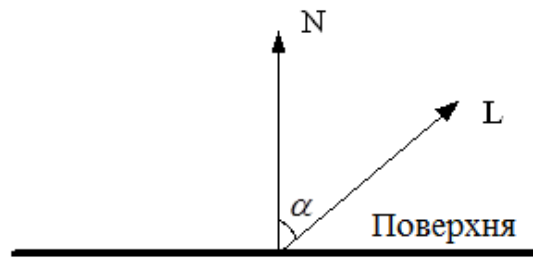


Рисунок 5.6 – Характеристики поверхні для моделі розсіяного освітлення

Розрахунок інтенсивності такого типу освітлення можна провести згідно з правилом Ламберта: інтенсивність освітлення пропорційна косинусу кута між нормаллю до поверхні  $N$  і напрямом на джерело світла  $L$  (5.5).

$$f_d(\alpha) = p_d \cdot \cos(\alpha), \quad (5.5)$$

де  $p_d$  – інтенсивність розсіяного освітлення в точці;  $\alpha$  – кут між вектором напрямку на джерело світла  $L$  і нормаллю до поверхні  $N$ .

Визначення кута між двома векторами – складна задача для обчислення, але можна визначити косинус між ними, використовуючи одну з властивостей скалярного векторного добутку. Для цього вектори  $L$  і  $N$  необхідно нормалізувати і розрахувати їх скалярний добуток, він дасть результат пошуку. Оскільки інтенсивність світла не може бути від'ємною величиною, додатково обмежимо отримане значення 0 (5.6).

$$f_d(\alpha) = p_d \cdot \max(0, \|N\| \|L\|), \quad (5.6)$$

"·" – «точка» (від англ. **dot product**) – операція скалярного векторного добутку,  $\| \|$  – операція нормалізації вектора.

**3) Відбите (дзеркальне) світло (specular light)** – це світло, яке, відбившись від поверхні об'єкта, поширюється в напрямку спостерігача. Стикаючись з поверхнею, воно відбивається, формуючи відблиски (відображення джерела світла на поверхні об'єкта), видимі лише при

погляді на об'єкт під певним кутом. При розрахунку дзеркального світла необхідно брати до уваги місце розташування і орієнтацію камери, які задаються через напрям на спостерігача  $V$ , напрямок світлових променів  $L$  і нормаль до поверхні  $N$  (рис. 5.7).

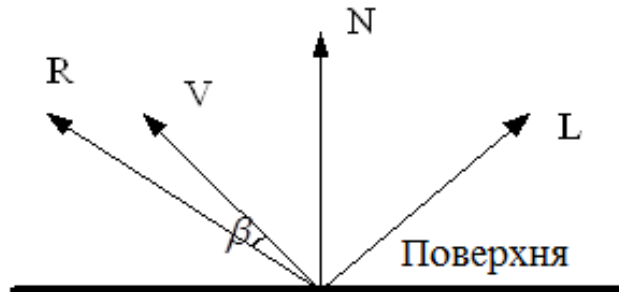


Рисунок 5.7 – Характеристики поверхні для дзеркального освітлення

Для розрахунку такого типу освітлення можна використовувати правило Фонга: інтенсивність відблиску пропорційна косинусу кута між вектором відображення  $R$  і напрямом на спостерігача  $V$ , зведеного в ступінь  $n$ , яка характеризує шорсткість поверхні (розмір відблиску). Чим менший кут  $\beta$  між вектором спостереження і вектором відображення, тим яскравіший відблиск.

Інтенсивність відблиску в даній моделі обчислюється по (5.7).

$$f_s(b, d) = p_s \cdot \cos^n(\beta), \quad (5.7)$$

$p_s$  – відбивальна здатність матеріалу точки;

$\beta$  – кут між вектором відображення  $R$  і напрямом на спостерігача  $V$ ;

$R = L - 2 \cdot \cos(\alpha) \cdot N$  – вектор відбиття світла;  $n$  – параметр, який задає відносний розмір відблиску.

Зі зростанням параметра  $n$  відображення стає інтенсивнішим і концентрується вздовж напрямку вектора відображення  $R$ . Використовуючи властивості скалярного векторного добутку, отримаємо (5.8).

$$f_s(b, a) = p_s \cdot \max(0, \|V\| \cdot \|R\|)^n. \quad (5.8)$$

Моделювання відблисків дозволяє істотно збільшити реалістичність

зображення, яке отримується системою візуалізації.

Підсумувавши внесок від усіх трьох компонент освітлення, можна отримати результивний внесок джерела світла в формування кольору фрагмента (5.9), (5.10).

$$c = c_1 \cdot (f_s(b, a) + c_m \cdot (f_d(a) + p_a)), \quad (5.9)$$

$$c = c_1 \cdot (p_s \cdot \max(0, \|V\|, \|R\|)^n + c_m \cdot (p_d \cdot \max(0, \|N\|, \|L\|) + p_a)), \quad (5.10)$$

де  $c_m$  – колір матеріалу поверхні,  $c_1$  – колір джерела світла.

Як можна помітити, на колір відблиску впливає тільки колір джерела світла (для діелектриків), а колір матеріалу впливає тільки на фонове і розсіяне освітлення.

Існує множина різних, більш точних і комплексних моделей освітленості, наприклад, модель Орена-Наяра, Куку-Торренса, Блінна, Варда та інші. Залежно від наявності обчислювальних ресурсів і необхідної якості освітлення можна вибрати більш-менш точну модель.

#### 5.4 Задання матеріалу

Колір об'єктів, який людина бачить у реальному світі, визначається кольором відображуваного ними світла. Наприклад, червона куля виглядає червоною, тому що вона поглинає весь спектр світла, крім червоного. Червоне світло відбивається від кулі і потрапляє на сітківку ока спостерігача, викликаючи відгук відповідних рецепторів. Дане явище моделюється шляхом задання кольору матеріалу для поверхні об'єкта  $c_m$ .

Матеріал визначає спектр світла, яке відбивається. Взаємодія світла джерела і кольору матеріалу моделюється шляхом покомпонентного перемноження відповідних векторів. Таким чином, якщо ми створюємо джерело світла, що випускає тільки синю частину спектра, і освітлюємо ним кулю – його поверхня буде виглядати чорною, оскільки куля повністю поглинає промені всього спектра, крім червоного, у тому числі і синій, а промені червоного кольору на нього не потрапляють. Якщо об'єкт поглинає всі падаючі на нього промені світла, то він виглядає чорним. А якщо об'єкт повністю відображає всі промені (червоний, синій і зелений), то він виглядає білим у випадку освітлення його джерелом білого світла.

Розмір відблиску  $n$  для моделі Фонга і відображальна здатність поверхні об'єкта теж відносяться до властивостей матеріалу. Задавати властивості можна з точністю як для вершин, так і для всіх точок поверхні. В останньому випадку властивості матеріалу зображаються у вигляді набору текстур, які покривають поверхню об'єкта.

Процеси перетворення координат, проєціювання, особливо растеризації і розрахунку освітлення вимагають виконання великого обсягу обчислень, які доцільно проводити з використанням прискорювачів тривимірної графіки. Кінцевим результатом етапу растеризації є двовимірне зображення, що виводиться на екран монітора.

### Контрольні питання

1. Для чого необхідно розраховувати освітленість об'єктів?
2. Типи джерел світла.
3. Що таке фонове джерело світла?
4. Що таке направлене джерело світла?
5. Що таке точкове джерело світла?
6. Чим відрізняється зональне джерело світла від точкового?
7. Призначення кутів  $\phi$  і  $\theta$  зонального джерела світла.
8. Які моделі освітлення ви знаєте?
9. Що таке колір, забарвлення, яскравість?
10. Що називається насиченістю, колірним тоном, кольоровістю?
11. Що таке константне сприйняття кольору?
12. Принцип розрахунку освітлення.
13. Чи можливо отримати на практиці будь-які комбінації кольорів RGB? Чому?
14. Алгоритм розрахунку освітлення.
15. Що називається метамерією?
16. Призначення колірного трикутника.
17. Що таке фонове світло?
18. Що таке розсіяне світло?
19. Що таке відбите світло?
20. Правило Ламберта.
21. Правило Фонга.
22. Як виглядає загальна формула розрахунку освітлення?
23. Що таке матеріал?

## 6. АПАРАТНІ ЗАСОБИ ТРИВИМІРНОЇ ГРАФІКИ

Сучасні графічні прискорювачі мають колосальні обчислювальні можливості і досить складну архітектуру, яка дозволяє ефективно використовувати їх при вирішенні як традиційних завдань обробки графіки, так і обчислювальних завдань загального призначення. Графічний прискорювач складається з трьох основних частин:

- а) графічного процесора;
- б) швидкодіючої пам'яті;
- в) інтерфейсу RAMDAC, який пов'язує його з пристроєм виведення (монітором);
- г) системи охолодження;
- д) ПЗУ;
- е) шини даних і різних інтерфейсів.

**Графічний процесор (англ. Graphics processing unit, GPU)** – окремий пристрій у складі обчислювальної системи, який виконує обробку графічних даних. Сучасні графічні процесори ефективно обробляють і відображають комп'ютерну графіку завдяки спеціалізованій конвеєрній архітектурі. Вони набагато ефективніші в обробці графічної інформації і в цілому в обчисленнях, ніж центральний процесор.

Графічний процесор, на відміну від універсального, володіє меншим набором виконуваних команд, але більшою продуктивністю. Сучасний графічний процесор складається з тисяч простіших за архітектурою й енергоефективних ядер, створених для обробки декількох завдань одночасно.

Таким чином, CPU краще працює з послідовними завданнями, а при великому обсязі оброблюваної інформації перевагу має GPU, за умови, що обчислювальна задача може бути вирішена паралельно.

Конвеєр графічного процесора складається з наступних блоків:

- а) вершинного процесора;
- б) геометричного процесора;
- в) фрагментного процесора;
- г) блока растеризації.

На рис. 6.1 показано процес обробки даних за допомогою графічного конвеєра.

У графічний процесор надходять дані від CPU про сцену, яку

необхідно візуалізувати. Спочатку дані надходять до вершинного процесора, який займається розрахунком геометрії сцени і визначає положення вершин, здійснюючи матричні перетворення систем координат. Тут також можуть здійснюватися різні операції з вершинами. Цей процесор керується спеціальною мікропрограмою – вершинним шейдером, написаною спеціалізованою мовою, наприклад, **Cg** або **HLSL**.

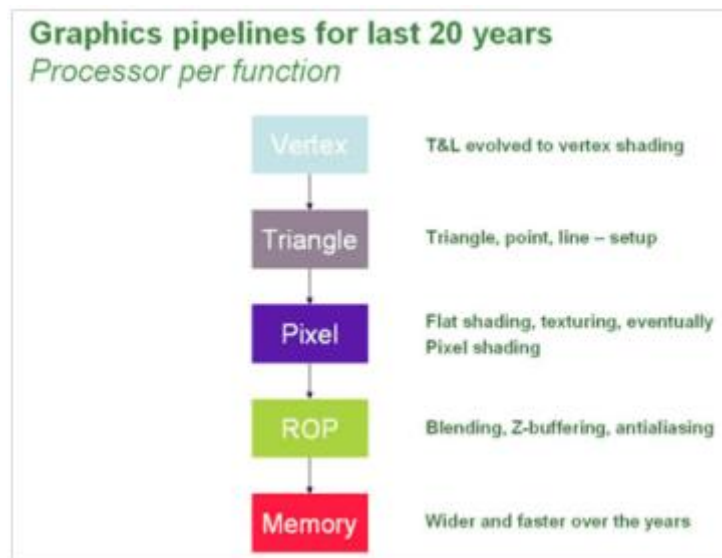


Рисунок 6.1 – Процес обробки даних за допомогою графічного конвеєра

Після вершинного процесора дані надходять до геометричного процесора, де відбувається складання вершин тривимірної моделі в трикутники з використанням інформації про з'єднання вершин (індексів). У сучасних прискорювачах на даному етапі також може використовуватися мікропрограма, яка буде видаляти або додавати геометрію. На цьому етапі відбувається проектування трикутників і визначення їх координат на екрані.

Після проектування дані про трикутники передаються до фрагментного процесора, де відбувається операція зафарбовування і растеризація кожного пікселя зображення, текстурування, розрахунок освітлення і затінення, створення процедурних текстур, постобробка кадру і накладення різних спецефектів, наприклад, туману. Це робиться також за допомогою мікропрограми – фрагментного шейдера. У разі використання антиаліасингу (AA) фрагментний процесор розраховує кожен використовуваний для формування кольору пікселя фрагмент.

Потім дані потрапляють до блоку растрових операцій, де з використанням буфера глибини (Z-буфера) визначаються і відкидаються ті фрагменти, яких не буде видно. Тут реалізується також ефект напівпрозорості і AA. Після розрахунку підсумкового кольору пікселя проводиться операція змішування поточного значення кольору, збереженого в буфері кадру, і нового значення кольору. Потім результат зберігається в буфері кадру.

Після формування всього зображення і збереження його в буфері проводиться обмін активного і додаткового буферів, щоб уникнути мерехтіння. Відеосигнал про сформовану картинку з активного буфера передається в монітор, для чого застосовується спеціальний перетворювач **RAMDAC** (RAM Digital-to-Analog Converter – цифро-аналоговий конвертор). Він безперервно читає активний кадровий буфер і формує сигнал, який передається через відеовихід відеокарти. В сучасних GPU є кілька RAMDAC, що дозволяє одночасно виводити кілька відеосигналів на кілька моніторів.

## 6.1 Оцінка швидкодії графічних процесорів

Зазвичай графічний процесор складається з декількох конвеєрів, які працюють паралельно, і чим більше конвеєрів, тим вища продуктивність GPU. Це відбувається за рахунок того, що вершини і пікселі обробляються на різних конвеєрах паралельно.

Найважливішою характеристикою GPU є кількість вироблених операцій з плаваючою комою за певний час (як правило, за 1 секунду). Такий показник називається FLOPS (Floating point Operations Per Second – операції з плаваючою комою в секунду). Ця величина визначається за допомогою тестових програм, наприклад, для вирішення систем лінійних алгебраїчних рівнянь.

Також важливою характеристикою GPU є пропускна здатність шини даних, оперативної і кеш-пам'яті.

На рис. 6.2 наведено порівняльну характеристику деяких GPU. Інтенсивний розвиток і ускладнювання GPU, продуктивність і простота програмування - сприяли тому, що їх стали застосовувати не тільки для обробки графіки, але також для реалізації розрахункових алгоритмів і методів загального призначення. Наприклад, ресурсоемних наукових розрахунків (**GPGPU** – General-Purpose Computation on GPUs). Причому



графічний процесор може виконувати такі розрахунки в кілька разів (до 10 і більше) швидше, ніж найпродуктивніший CPU.

	G70 (GeForce 7800)	R520 (Radeon X1800)	G71 (GeForce 7900)	R580 (Radeon X1900)	G80 (GeForce 8800)	R600 (Radeon HD2900)
Пиксельных блоков (шт.)	24	16	24	48	128	320
Вершинных блоков (шт.)	8	8	8	8	128	320
Количество TMU (текст. за такт)	24	16	24	16	8 (32)	4 (16)
Количество ROP (пикс. за такт)	16	16	16	16	6 (24)	4 (16)
Частота ядро (МГц)	430	625	650	650	575	742
Частота вершинного блока/унифиц. проц. (МГц)	470		700		1350	
Техпроцесс (нм)	110	90	90	90	90	80
Площадь кристалла (мм <sup>2</sup> )	333	288	196	352	480	425
Количество транзисторов (млн. шт.)	302	305	279	384	681	700
Объем памяти (Мб)	256	256	512	512	768	512
Частота памяти, номин/факт (МГц)	600/1200	750/1500	800/1600	775/1550	900/1800	828/1660
Ширина шины памяти (бит)	256	256	256	256	384	512
Тип памяти	GDDR3	GDDR3	GDDR3	GDDR3	GDDR3	GDDR3
Производительность (Гигафлопс)	165	83	230	360	518	475
Пропускную способность памяти (Гб/с)	38,4	48	51,2	49,6	86,3	106
Энергопотребление (Вт)	110	100	70-80	120	145	215

Рисунок 6.2 – Порівняльна характеристика сучасних GPU

Порівняльне зростання обчислювальної потужності GPU і CPU за 10 років наведено на рис. 6.3.

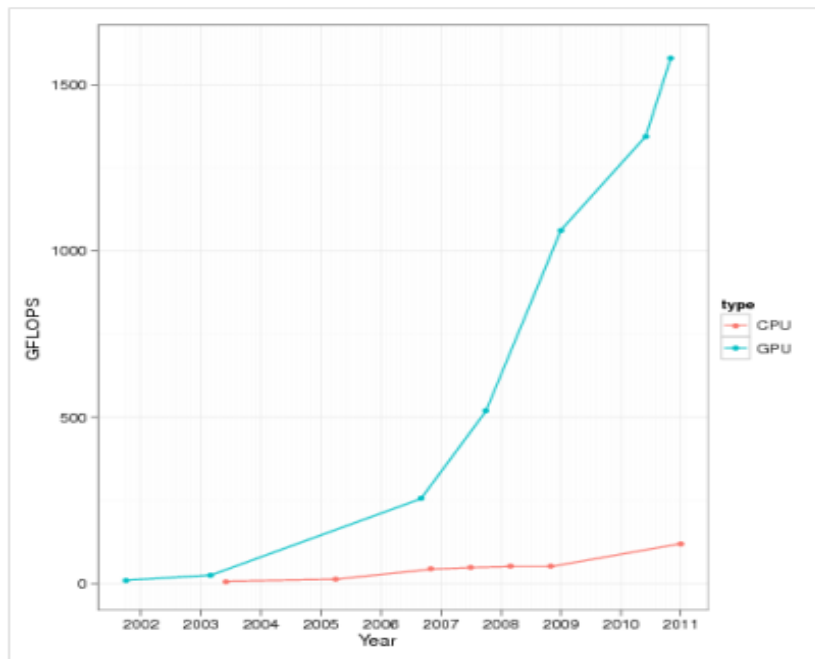


Рисунок 6.3 – Зростання обчислювальної потужності GPU і CPU за часом

Для програмування GPU можливе використання багатьох технологій, у тому числі найпоширенішої технології CUDA, відкритого стандарту OpenCL і OpenACC.

### Контрольні питання

1. Що таке графічний процесор?
2. З яких частин складається графічний прискорювач?
3. У чому відмінності графічного процесора від центрального процесора обчислювальної системи?
4. В яких випадках краще використовувати графічний, а в яких центральний процесор?
5. З яких частин складається конвеєр графічного процесора?
6. Що робить вершинний процесор?
7. Що таке шейдер?
8. Для чого призначений фрагментний процесор?
9. На якій стадії конвеєра здійснюється "збірка" трикутників?
10. Що таке фрагментний шейдер?
11. Що таке растрові операції?
12. Для чого призначений буфер кадру?
13. Який механізм дозволяє уникнути мерехтіння зображення при візуалізації?
14. Для чого використовується RAMDAC?
15. Який показник застосовується для оцінки швидкодії графічного процесора?
16. Для чого крім графіки може використовуватися графічний процесор?

## 7. ПРОГРАМНІ ІНТЕРФЕЙСИ ТРИВИМІРНОЇ ГРАФІКИ

**API** – це інтерфейс прикладного програмування (англ. *Application programming interface*) – набір процедур, функцій, класів, структур і констант, які надає програма. Також API – це сервіс для використання в інших програмних продуктах.

На даний час існує два основних графічних API: OpenGL і DirectX.

**Графічна бібліотека OpenGL** розроблена компанією SGI для роботи з апаратним забезпеченням своїх високопродуктивних графічних станцій. Додатки, які працюють на платформі SGI, використовували для генерації тривимірних сцен графічну бібліотеку з назвою GL (Graphics Library). SGI відкрила цей стандарт для вільного ліцензування, відповідним чином змінивши його назву на OpenGL.

Програми, написані за допомогою OpenGL, теоретично можна переносити на будь-які платформи – будь то графічна станція, або ПК. Якщо пристрій підтримує якусь функцію, то вона виконується апаратно, а якщо ні, то бібліотека виконує її програмно.

Велика кількість додатків (в основному графічні пакети для створення 3D графіки) написана з використанням OpenGL. Вона підтримується багатьма платформами, починаючи від звичайних IBM-сумісних комп'ютерів і закінчуючи RISC-машинами.

**Бібліотека Microsoft DirectX** та її частина, що відповідає за 3D графіку, –Direct3D набула поширення на персональних комп'ютерах. Набір DirectX входить до стандартного комплекту поставки ОС Windows, поставляється також багатьма виробниками програмного і апаратного забезпечення разом з відеоадаптерами і дистрибутивами додатків.

### 7.1 Direct3D API

**Direct3D** – це низькорівневий графічний API (програмний інтерфейс для додатків), який дозволяє відображати тривимірні сцени, використовуючи апаратні прискорювачі 3D графіки. Direct3D можна сприймати як посередника між додатком і графічним пристроєм (апаратурою тривимірної графіки). Наприклад, щоб наказати графічному пристрою очистити екран, додаток повинен викликати метод Direct3D IDirect3DDevice9::Clear. Взаємодію між додатком, Direct3D і апаратурою комп'ютера показано на рис. 7.1.

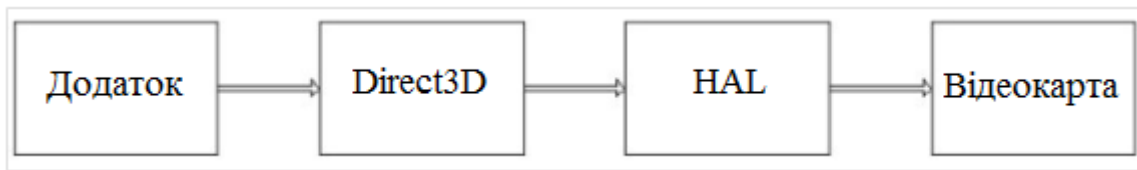


Рисунок 7.1 – Взаємозв'язок між додатком, Direct3D і апаратурою

На рис. 7.1 блок з назвою Direct3D подає набір документованих інтерфейсів і функцій, які охоплюють увесь перелік функціональних можливостей, пропонованих даною версією Direct3D.

**HAL (англ. Hardware Abstraction Layer)** – рівень абстрагування від апаратури, проміжна стадія між Direct3D і графічним пристроєм. Direct3D не взаємодіє з апаратурою безпосередньо, оскільки існують сотні типів різних відеокарт від різних виробників, кожна з яких відрізняється набором підтримування і способом реалізації функцій. Тому Direct3D вимагає, щоб виробники обладнання реалізували рівень абстрагування від устаткування (HAL), який представляє собою апаратно залежний код, що вказує пристрою, як виконувати ті чи інші операції. Це дозволяє Direct3D не знати специфіку конкретних пристроїв, і його специфікації не залежать від використаного обладнання.

Виробники відеокарт вбудовують підтримку всіх пропонованих їх обладнанням можливостей в HAL. Ті можливості, які пропонує Direct3D, але які не підтримує пристрій, в HAL не реалізовані, і спроба їх використання призведе до помилки, за винятком тих випадків, коли необхідна функція може бути відтворена програмно, як, наприклад, програмна обробка вершин.

Для емуляції всіх можливостей Direct3D до складу бібліотеки входить так званий допоміжний растеризатор (reference rasterizer REF), який може повністю підмінити HAL для налагоджувальних цілей. Це дозволяє писати і перевіряти код, який використовує можливості Direct3D, непідтримуваного апаратурою (фрагментні шейдери). У цьому растеризаторі проводиться програмна емуляція всіх функцій бібліотеки з використанням центрального процесора, і його швидкодія дуже низька. Потрібно розуміти, що пристрій REF призначений тільки для розробників, він присутній тільки в DirectX SDK і не може бути наданий користувачам програми.

DirectX поширюється у двох варіантах:

1) *DirectX Redistributable* – тільки бібліотеки, необхідні для роботи, використовується на комп'ютерах користувачів.

2) *DirectX Software Development Kit (DX SDK)* – набір прикладів і вихідних кодів, які використовуються для розробки програм. Для використання основних і додаткових функцій бібліотек необхідно підключити до програми відповідні бібліотеки (d3d9.lib і d3dx9.lib) та заголовочні файли (d3d9.h і d3dx9.h).

## 7.2 Використання API Direct3D

Для розробки додатків з використанням функцій Direct3D необхідно виконати підключення бібліотеки Direct3D, яка оснований на застосуванні COM-інтерфейсів. Модель компонентних об'єктів (англ. Component Object Model, COM) – це технологія, яка дозволяє бібліотеці бути незалежною від мови програмування і сумісною з усіма попередніми версіями. Для отримання покажчика на COM-інтерфейс необхідно викликати спеціальну функцію або метод іншого COM-інтерфейса (зазвичай назва такої функції починається зі слова Create). Крім того, завершивши роботу з COM-інтерфейсом, слід викликати його метод `Release()`. COM-об'єкти самостійно здійснюють управління пам'яттю. У кодї для позначення COM-інтерфейсів використовується велика літера I. Наприклад, COM-інтерфейс, який представляє пристрій візуалізації (відеоадаптер), називається `IDirect3DDevice9`.

Процес ініціалізації та підготовки Direct3D складається з:

1) отримання покажчика на інтерфейс `IDirect3D9`. Цей інтерфейс застосовується для отримання інформації про встановлені в комп'ютері пристрої та створення інтерфейсу `IDirect3DDevice9` – апаратного пристрою, що використовується для виведення тривимірної графіки.

2) перевірки властивостей пристрою (`D3DCAPS9`), щоб дізнатися, які можливості надає 3D-прискорювач.

3) заповнення примірника структури `D3DPRESENT_PARAMETERS`, яка містить ряд параметрів, що дозволяють нам задати необхідні характеристики інтерфейсу `IDirect3DDevice9`. У цій структурі задаються розмір і формат відображуваного буфера, кількість буферів, частота оновлення екрана, використання апаратного прискорення, додаткові буфери, наприклад, Z-буфер, тип антиаліасингу.

4) отримання інтерфейсу для роботи з прискорювачем 3D графіки `IDirect3DDevice9`, ґрунтуючись на заповненій структурі `D3DPRESENT_PARAMETERS`. Для цього викликається метод `IDirect3D9::CreateDevice()`.

Після успішного створення `IDirect3DDevice9` можна виводити тривимірну графіку з апаратним прискоренням. Для цього необхідно завантажити та (або) створити всі необхідні для візуалізації ресурси (текстури, моделі і т.д.) перед початком процесу візуалізації.

На кожному кадрі потрібно очистити буфер для малювання та допоміжні буфери. Потім установити настройки графічного конвеєра і вивести об'єкти сцени з відповідними їм параметрами освітлення і матеріалів. Після чого перемкнути екранні буфери, щоб показати зображення на екрані.

Після закінчення роботи програми необхідно звільнити отримані інтерфейси `IDirect3D9` і `IDirect3DDevice9`, використовуючи методи `Release()` у кожного з інтерфейсів.

### 7.2.1 Опис моделі об'єкта в Direct3D.

Геометрія об'єктів сцени подається за допомогою сітки трикутників (triangle mesh). Точка, в якій зустрічаються два ребра трикутника, називається вершиною (vertex). У Direct3D біля вершини, крім її координат, можуть бути присутніми додаткові властивості. Наприклад, може зберігатися колір, нормаль до поверхні, текстурні координати. Direct3D дозволяє конструювати довільні формати опису вершин.

Щоб створити формат вершини, необхідно повідомити структуру, яка буде містити поля різних типів, з необхідними даними, наприклад:

```
struct s_vertex
{
    float x, y, z; //координати
};

struct s_vertex_norm
{
    float x, y, z; //координати
    float nx, ny, nz; //нормаль
};
```

Після повідомлення такої структури необхідно описати формат зберігання цих даних у структурі. Це можна зробити за допомогою

комбінації прапорів формату вершин (**flexible vertex format, FVF**), або структури опису вершин `Vertex Declaration`, маніпуляції з якою здійснюються за допомогою інтерфейсу `IDirect3DVertexDeclaration9`. Опис створюється за допомогою методу `IDirect3DDevice9::CreateVertexDeclaration()`.

**Опис вершин** надається у вигляді масиву структур `D3DVERTEXELEMENT9`, у кінці якого йде спеціальна структура `D3DDECL_END`.

Кожен елемент опису складається з:

- а) номера потоку;
- б) зміщення поля від початку структури в байтах;
- в) типу поля (`FLOAT`, `FLOAT2`, `FLOAT3`, ...);
- г) типу інтерпретації даних;
- д) призначення даних (позиція, нормаль, текстурні координати), яке дозволяє зв'язати їх з вхідними параметрами вершинної мікропрограми;
- е) номера даних (використовується, якщо є дані з однаковим призначенням).

Для представленої вище структури `s_vertex_norm` з двома векторами з трьох 32-бітних елементів (`FLOAT3`) необхідно задати наступний опис формату:

```
const D3DVERTEXELEMENT9 decl [] =
{
    {0, 0, D3DDECLTYPE_FLOAT3, D3DDECLMETHOD_DEFAULT,
     D3DDECLUSAGE_POSITION, 0},

    {0, 12, D3DDECLTYPE_FLOAT3, D3DDECLMETHOD_DEFAULT,
     D3DDECLUSAGE_NORMAL, 0},

    D3DDECL_END()
};
```

Перший опис говорить про те, що структура даних вершини містить відомості про місце знаходження (`D3DDECLUSAGE_POSITION`), а в другому додається нормаль (`D3DDECLUSAGE_NORMAL`).

Трикутники є основними будівельними блоками тривимірних об'єктів. Щоб сконструювати об'єкт, необхідно створити список трикутників, які описують його форму і топологію (рис. 7.2).

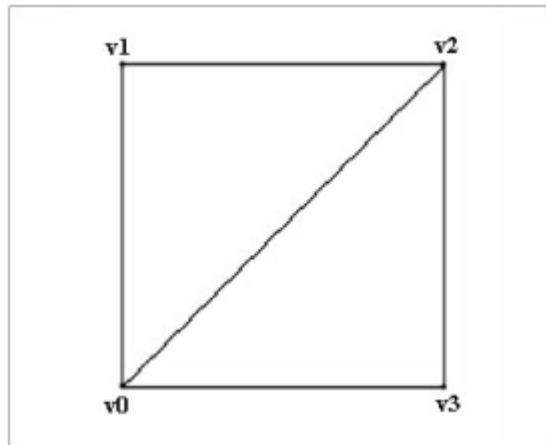


Рисунок 7.2 – Розбиття об'єкта на трикутники

Трикутники, які утворюють тривимірний об'єкт, задаються набором індексів (indices) – номерів вершин. У списку (масиві) вершин перераховуються всі унікальні вершини у вигляді набору структур, а список індексів містить послідовність номерів (індексів) вершин зі списку вершин, що показує, як об'єднуються вершини в трикутники:

```
WORD indexes[6] =  
{  
    0, 1, 2,    //трикутник 1  
    0, 2, 3    //трикутник 2  
};
```

Таким чином, для опису моделі об'єкта необхідно заповнити два масиви – масив властивостей вершин і масив індексів.

Перед змалюванням необхідно встановити коректний опис структури вершин за допомогою методу `IDirect3DDevice9::SetVertexDeclaration()`, а потім передати масиви вершин та індексів при виклику методу `IDirect3DDevice9::DrawIndexedPrimitiveUP()`, щоб намалювати об'єкт, опис якого знаходиться в основному ОЗП обчислювальної системи.

Якщо потрібна максимальна швидкість, доцільно помістити опис об'єкта (вершини і індекси) в вершинні й індексні буфери, які попередньо створені і знаходяться в ОЗП прискорювача, і проводити рендеринг з використанням цих буферів.



### 7.2.2 Вершинні і фрагментні мікропрограми на мові HLSL.

Для опису мікропрограм у бібліотеці Direct3D використовується спеціальна високорівнева мова **HLSL** (англ. High Level Shading Language). Вона дозволяє управляти роботою графічного процесора на етапах обробки вершин і растеризації зображення, а в новітніх прискорювачах – і на етапі складання трикутників. Кожна мікропрограма описується у вигляді функції і називається шейдером.

HLSL підтримує різноманітні типи даних: скалярні (табл. 7.1) і складові типи (табл. 7.2) – вектори і матриці. До кожного з складових типів можна звертатися, використовуючи поля: *x*, *y*, *z*, *w* або *r*, *g*, *b*, *a*, які еквівалентні елементам від 0 до 3 відповідно.

Змінна може бути оголошена як *static* або *extern*. Будь-яка нестатична змінна (записується з приставкою *extern* або без приставки), оголошена за межами шейдера, може бути змінена через API.

Таблиця 7.1 – Скалярні типи

Тип	Значення
<code>bool</code>	<code>true</code> або <code>false</code>
<code>int</code>	32-bit signed integer
<code>half</code>	16-bit floating point value
<code>float</code>	32-bit floating point value
<code>double</code>	64-bit floating point value

Таблиця 7.2 – Складові типи

Тип	Значення
<code>float?</code>	? 32-bit floats
<code>half?</code>	? 16-bit floats
<code>float?x?</code>	?x? 32-bit floats
<code>half?x?</code>	?x? 16-bit floats

Статична змінна використовується тільки шейдером і не керується API.

```
extern float a;
static float c;
float d;
```

Ініціалізація змінних проводиться, як у мові C / C++.

```
float2x2 mat =
{
    1.0f, 0.0f,    //ряд 1
    1.0f, 2.0f    //ряд 2
};
float4 pos = { 1.0f, 0.5f, 12.0f, 1.0f };
float f = 0.01f;
```

Основні операції показані в табл. 7.3.

Таблиця 7.3 – Основні операції

Операції	Оператори
арифметичні	-, +, *, /, %
інкремент, декремент	++, -
логічні	&&,   , ?:
унарні	!, -, +
порівняння	<, >, <=, >=, ==, !=
присвоєння	=, -=, +=, *=, /=
приведення типів	(тип)
перерахування	,
доступ до поля структури	.
доступ до елемента масиву	[індекс]

Оператор вилучення остачі від ділення % працює як з цілими так і дійсними числами. Порівняння векторів проводиться покомпонентно.

Для розгалуження застосовується оператор `if(A) {B;} else {C;}`,

або тернарний оператор  $x = A ? B : C$ .

Слід пам'ятати, що через особливості архітектури графічних прискорювачів розгалуження в більшості випадків буде замінено арифметичним виразом: розрахуються обидві гілки, а потім буде обрана одна з них шляхом множення на 0 результатів протилежної.

Для організації циклів за аналогією з мовою C використовуються оператори:

```
do { D; } while (A);  
while( A ) { D; }  
for( B; A; C ) { D; }
```

Циклічне повторення коду може уповільнити виконання прошивки і збільшити її розмір, оскільки у більшості випадків цикли будуть розгорнуті компілятором у набір послідовних команд.

Виклик і визначення функцій також зроблені аналогічно мові C++, за винятком того, що не підтримує рекурсія. Параметри функції можуть бути семантично пов'язані з даними.

Деякі математичні функції, які можна застосовувати в HLSL, наведені в таблиці Б.1 (див. додаток Б).

Великі масиви однотипних даних подаються у вигляді текстур. Для використання текстур у фрагментного шейдера необхідно оголосити *extern* змінну типу **sampler**. Вона визначає номер масиву (текстури), з якого можна читати дані (зазвичай колір) у фрагментного шейдера.

Наприклад, оголошення такого семплера, який прив'язує його до текстури з номером 0, виглядає наступним чином:

```
sampler tex: register( s0 );
```

Функції, які повертають колір текстури, виходячи із заданих параметрів, наведені в таблиці Б.2 (див. додаток Б).

Також необхідно визначити вхідні і вихідні параметри вершинного і фрагментного шейдерів, за аналогією з аргументами і результатами, які повертаються функціями в мові C.

Вершинні і фрагментні шейдери мають два типи вхідних даних: **varying** і **uniform**.

**uniform** – дані, постійні для кожного використання в шейдері. Об'яву **uniform** даних в HLSL можна зробити двома способами:

1) Передати дані як extern змінну:

```
float f;  
float4 main(): COLOR  
{  
    return f;  
}
```

2) Передати дані через визначник uniform:

```
float4 main( uniform float4 f ): COLOR  
{  
    return f;  
}
```

Uniform змінні задаються через таблицю констант. Таблиця констант містить усі регістри, які постійно використовуються в шейдері, їх кількість обмежена невеликим значенням, залежним від покоління відеокарт.

**varying** – дані, які є унікальними для кожного виклику шейдера: позиція, нормаль і т. д. У вершинному шейдері така семантика описує varying дані, які передаються з вершинного буфера, заданого в основній програмі, а у фрагментного шейдера – інтерпольовані дані, отримані з вершинного шейдера. Основні семантичні типи для вхідних значень наведені в таблиці Б.3 (див. додаток Б).

Використання varying даних у фрагментному шейдері визначає стан одного фрагмента. Основні вхідні семантичні типи наведені в табл. 7.4.

Таблиця 7.4 – Основні вхідні семантичні типи

COLOR $n$	Колір
TEXCOORD $n$	Текстурні координати

$n$  – визначає номер семантичного типу.

Наприклад: TEXCOORD1, NORMAL0.

Вихідні дані шейдера визначають спосіб лінкування з вхідними даними наступного етапу графічного конвеєра. Наприклад, вихідна семантика для вершинного шейдера пов'язує вихідні дані з інтерполяторами в растеризаторі, який генерує вхідні дані для вершинного (табл. 7.5) і фрагментного шейдера (табл. 7.6).

Таблиця 7.5 – Вихідні дані для вершинного шейдера

POSITION	Позиція
PSIZE	Розмір точки
FOG	Коефіцієнт "туманності" для вершини
COLOR $n$	Колір
TEXCOORD $n$	Текстурні координати

Таблиця 7.6 – Вихідні дані для фрагментного шейдера:

COLOR $n$	Колір
DEPTH	Значення глибини

Приклад найпростішого вершинного шейдера:

```
//Зовнішня матриця для загального перетворення вершин.
//Задається з основної програми
float4x4 mvp;
//Структура для вихідних даних.
//Ці дані будуть передаватися в фрагментний шейдер.
struct s_vs_out
{
    float4 pos: POSITION;
    float2 tex0: TEXCOORD0;
};

//Для вхідних даних потрібна тільки позиція об'єкта,
//яка задається відразу в вершинному буфері.
s_vs_out main( float4 ipos: POSITION )
{
    s_vs_out o;
    //Перетворення координат
    o.pos = mul( mvp, ipos );
    //Формування текстурних координат з екранних
    o.tex0 = normalize( pos.xy );
    return o;
}
```

Фрагментний шейдер, який малює концентричні кільця на екрані за формулою  $R^2 = (x+a)^2 + (y+b)^2$ . Для зручності приймемо  $a = 0$ ,  $b = 0$ :

```

//Зовнішні змінні, в яких задано:
//час і кількість кілець
float time, rings = 5;
float4 main( float2 tex0: TEXCOORD0 ): COLOR
{
    float a = atan2( tex0.x, tex0.y );
    float rad = dot( tex0, tex0 );
    return 0.5 * ( 1.0 + sin( a + rings * rad + time ) );
};

```

Більш складний фрагментний шейдер для розрахунку освітлення за правилами Ламберта і Фонга наведено в додатку В.

Зазвичай в Direct3D шейдери задаються у вигляді файлів спецефектів *.fx*, які завантажуються і компілюються на старті програми, використовуючи функцію `D3DXCreateEffectFromFile()`. Робота з шейдерами ведеться за допомогою інтерфейсу `ID3DXEffect` – допоміжної бібліотеки `D3DX`. Крім компіляції шейдерів цей інтерфейс дозволяє встановлювати змінні й управляти графічним конвеєром прискорювача (через `Direct3D`).

Кожен файл ефектів складається з опису змінних, вершинних і фрагментних шейдерів, а також кількох технік рендерингу, позначених ключовим словом `technique`. Кожна техніка, в свою чергу, складається з одного або декількох етапів, позначених ключовим словом `pass`. На кожному етапі задаються настройки конвеєра: використання `Z`-буфера (`ZEnable`, `ZFunc`), тип змішування й застосування прозорості пікселів (`AlphaBlendEnable`, `AlphaTestEnable`), вид рендерингу (`FillMode`), тип відсікаючихся граней (`CullMode`) і т. д. Також задаються вершинний і фрагментний шейдери (`VertexShader`, `PixelShader`).

Перед початком рендерингу необхідно знайти за ім'ям і перевірити на можливість роботи з обраним пристроєм усі необхідні техніки рендерингу з використанням методів `ID3DXEffect::GetTechniqueByName()` і `ID3DXEffect::ValidateTechnique()`. Потім отримати дескриптори всіх змінних, які використовувались у мікропрограмі і для яких будуть задаватися значення методом `ID3DXEffect::GetParameterByName()`.

При рендерингу через інтерфейс `ID3DXEffect` вибирається необхідна техніка методом `ID3DXEffect::SetTechnique()`, у цей метод передається дескриптор техніки, отриманий після виклику методу `ID3DXEffect::GetTechniqueByName()`. Далі запускається процес рендерингу техніки `ID3DXEffect::Begin()`. Для неї в циклі перебираються і запускаються всі етапи, з яких вона складається, за допомогою функцій

ID3DXEffect::BeginPass(), ID3DXEffect::EndPass(). У кінці рендерингу викликається метод ID3DXEffect::End(). Приклад виклику техніки hTec для ефекту pEffect:

```
void RenderScene( IDirect3DDevice9* pDev, D3DXHANDLE hTec )
{
    pEffect->SetTechnique( hTec );

    UINT passes = 0;
    if( pEffect->Begin( &passes, 0 ) != D3D_OK )
        return;

    for( UINT i = 0; i < passes; ++i )
    {
        if( pEffect->BeginPass( i ) != D3D_OK )
            break;

        //функції установки параметрів рендерингу і малювання
        RenderObjects();
        pEffect->EndPass();
    }

    pEffect->End();
}
```

На початковому етапі необхідно встановити формат опису вершин, змінні шейдера (матриці, кольори, вектори), застосувати їх до ефекту методом ID3DXEffect::CommitChanges() і обрисувати моделі IDirect3DDevice9::DrawIndexedPrimitiveUP().

Для установки матриць 4x4 використовується метод ID3DXEffect::SetMatrix(), векторів розмірністю 1x4 – ID3DXEffect::SetVector(), а одиничних чисел з плаваючою комою – ID3DXEffect::SetFloat(). До кожної з функцій передаються дескриптори параметрів, отримані на підготовчому етапі у виклику методу ID3DXEffect::GetParameterByName(), і нові значення змінних.

Таким чином, інтерфейс ID3DXEffect дозволяє істотно спростити роботу з мікропрограмами і конвеєром графічного прискорювача. Він надає весь необхідний функціонал для пошуку і вибору технік рендерингу і задання змінних, які використовуються у мікропрограмах.

### 7.3 Візуалізація з використанням Direct3D

Процес візуалізації кадру зазвичай відбувається при обробці повідомлення WM\_PAINT, яке приходить від операційної системи (див. додаток А).

Спочатку перевіряється доступність пристрою за допомогою методу `IDirect3DDevice9::TestCooperativeLevel()`. Якщо пристрій недоступний – рендеринг не проводиться. Сигналом для початку рендерингу кадру є виклик методу `IDirect3DDevice9::BeginScene()`. Потім необхідно очистити місце виведення певним кольором, для чого використовується метод `IDirect3DDevice9::Clear()`, в якому вказується колір і значення Z для очищення Z-буфера.

На першому кроці рендерингу розраховуються матриці виду і перспективної проекції відповідно до параметрів спостерігача і обраного типу проектування за допомогою функцій: `D3DXMatrixLookAtLH()` і `D3DXMatrixPerspectiveFovLH()`, а також допоміжні матриці для мікропрограм, наприклад, інверсна матриця виду за допомогою функції `D3DXMatrixInverse()`. При необхідності матриці перемножуються для отримання загального перетворення за допомогою функції `D3DXMatrixMultiply()`. Розраховані матриці записуються у відповідні змінні для використання мікропрограмами з використанням методу `ID3DXEffect::SetMatrix()`.

Далі встановлюються необхідні техніки, проводиться перемикання всіх вхідних стадій. На кожній стадії встановлюється опис вершин, параметри мікропрограм: кольори, матриці об'єктів і їх частин, вектори, опис матеріалів об'єкта і джерел світла. Застосовуються зроблені зміни в ефекті `ID3DXEffect::CommitChanges()` і рендериться геометрія потрібних об'єктів з використанням методу `IDirect3DDevice::DrawIndexedPrimitiveUP()`.

Закінченням рендерингу кадру служить виклик `IDirect3DDevice9::EndScene()`. Після закінчення рендерингу проводиться перемикання активного і вторинного буферів за допомогою методу `IDirect3DDevice9::Present()`.



## Контрольні питання

1. Які основні графічні API ви знаєте?
2. Для чого призначений графічний API?
3. Що таке DirectX?
4. В якому вигляді може поширюватися DirectX?
5. Що таке Direct3D?
6. Яким чином відбувається взаємодія програми і графічного прискорювача за допомогою Direct3D?
7. Що таке допоміжний растеризатор?
8. Для чого використовується модель компонентних об'єктів?
9. Процес ініціалізації та підготовки Direct3D.
10. Яким чином в Direct3D подається модель об'єкта?
11. Як відбувається опис формату вершини в Direct3D?
12. Що таке вершинні та індексні буфери?
13. Для чого потрібна мова HLSL?
14. Які типи даних підтримуються в мові HLSL?
15. Чим можна замінити розгалуження у мікропрограмі?
16. Що відбувається з циклами всередині мікропрограми?
17. Для чого застосовуються текстури?
18. Що таке семплер?
19. Що таке uniform дані?
20. Що таке varying дані?
21. Для чого використовується інтерфейс ID3DXEffect?
22. З чого складається файл опису ефектів?

## 8. МАТЕРІАЛИ І ТЕКСТУРУВАННЯ ПОВЕРХНІ ОБ'ЄКТІВ

**Текстурування** – це надання певних властивостей поверхням об'єктів. Текстурування є одним з найважливіших етапів роботи над моделлю. Воно дозволяє зробити так, щоб змодельований, штучно створений об'єкт виглядав, як справжній.

Наприклад, щоб змодельована стіна виглядала, як кам'яна кладка, необхідно зробити наступне:

- а) "пофарбувати" площину стіни в колір каменю;
- б) намалювати карту блиску, щоб каміння блищало так само, як і в реальності;
- в) зробити поверхню нерівною в потрібних місцях;
- г) створити карту відображення, щоб у каміння була невелика відображальна здатність. Мокре каміння буде відображати сильніше, ніж сухе і покрите пилом.

Налаштувавши таким чином властивості матеріалу, ми отримаємо площину, яка буде виглядати, як реальна кам'яна стіна.

Текстури можуть бути створені різними способами:

- а) в 2D редакторі (Photoshop, GIMP і т.д.);
- б) в пакеті для 3D моделювання (Mudbox, 3D-Coat, ZBrush);
- в) на основі процедурних карт;
- г) шляхом комбінації різних 2D, 3D і процедурних технік.

### 8.1 Представлення текстур

Текстури бувають чотирьох видів: одновимірні, двовимірні, кубічні й об'ємні.

Одномірні і двомірні текстури являють собою растри, кубічні – відображення простору на 6-ти площинах, а об'ємні дозволяють зберігати властивості об'єму. За аналогією з пікселем кожен елемент текстури називають тексель.

Кожна площина текстури представлена у вигляді набору MIP рівнів – зменшених копій вихідних текстур.

**MIP-текстурування (англ. MIP mapping)** – метод текстурування, який використовує кілька копій однієї текстури з різною деталізацією (назва походить від лат. *Multum in parvo* – «багато в малому»).

Відомо, що растеризовані об'єкти з текстурами найкраще

виглядають, коли деталізація вибірки з текстурі близька до роздільної здатності екрана. Якщо роздільна здатність екрана висока (текстура занадто маленька або об'єкт дуже близько), виходить розмите зображення. Якщо ж роздільна здатність текстурі занадто високий (текстура занадто велика або об'єкт далеко), при растеризації вибираються, по суті, випадкові пікселі, що призводить до мерехтіння і втрати дрібних деталей.

На практиці краще мати кілька текстур різної деталізації і накладати на об'єкт ту, яка найбільш підходить у даній ситуації.

Для цього створюється набір зображень (MIP-рівнів) з роздільною здатністю від максимального до 1. Наприклад: 256x256, 128 x 128, 64 x 64, 32 x 32 і так далі, до 1x1.

На всіх цих структурах одне і те ж зображення, змінюється тільки роздільна здатність. Таким чином, MIP-текстурування збільшує витрату відеопам'яті на третину (8.1).

$$\sum_{i=0}^n \left(\frac{1}{4}\right)^i = 1\frac{1}{3}. \quad (8.1)$$

При накладенні текстур на об'єкт обчислюється відстань до об'єкту, і номер MIP-рівня, який треба використовувати для візуалізації, знаходиться по (8.2).

$$\text{mip} = \log_2(D / (t \cdot r)) + b, \quad (8.2)$$

де  $r$  – роздільна здатність віртуальної камери (кількість пікселів, яка буде в об'єкті розміром в 1 од., розташованому в 1 од. від камери),  $t$  – розмір текселя в одиницях тривимірного світу,  $D$  – відстань до об'єкта в тих же одиницях,  $b$  – число, яке дозволяє вибирати більш-менш детальну текстуру, ніж дає формула.

Отримане значення округляється до цілого, і рівень з відповідним номером (0 – найдетальніший, 1 – удвічі менший і т. д.) накладається на об'єкт.

MIP-текстурування не вирішує усіх проблем, пов'язаних з текстуруванням. Наприклад, об'єкти, які знаходяться під гострим кутом до спостерігача (наприклад, дорога в автосимуляторі), у яких роздільна здатність по одній осі дуже відрізняється від роздільної здатності по іншій,

будуть явно розмиті по одній осі і буде видно мерехтіння по іншій. Іншою проблемою є чітко помітна межа між MIP-рівнями на моделі.

Для вирішення цих завдань використовуються лінійна й анізотропна фільтрації, які дозволяють змішувати дані текстури з різних MIP-рівнів (лінійна) та індивідуально розраховувати значення MIP рівня по кожній з осей (анізотропна) при розрахунку текселя для текстурованої точки об'єкта.

### **Контрольні питання**

1. Що таке текстурування?
2. Способи створення текстур.
3. Які типи текстур ви знаєте?
4. Що таке MIP-рівні?
5. На скільки використання MIP-рівнів збільшує витрату відеопам'яті?
6. Які види фільтрації текстур ви знаєте?
7. Призначення лінійної та анізотропної фільтрації?
8. У чому відмінність лінійної та анізотропної фільтрації текстур?
9. Для чого використовуються кубічні текстури?

## Список літератури

1. Гилой В. Интерактивная машинная графика : структуры данных, алгоритмы, языки / В. Гилой. — М. : Мир, 1981. — 380 с.
2. Дуда Р. Распознавание образов и анализ сцен : пер. с англ. / Р. Дуда, П. Харт. — М. : Мир, 1976. — Гл. 7, 9.
3. Ньюмен П. Основы интерактивной машинной графики / П. Ньюмен, Р. Спрулл ; пер. с англ. В. М. Грина, О. Н. Родинко. — М. : Мир, 1976. — 573 с.
4. Павлидис Т. Алгоритмы машинной графики и обработки изображений / Т. Павлидис. — М. : Радио и связь, 1986. — 198 с.
5. Порев В.Н. Компьютерная графика : учеб. пособие / Виктор Порев. — СПб.: ВHV-Санкт-Петербург, 2002. — 428 с.
6. Прэтт У. Цифровая обработка изображений : в 2 кн. / У. Прэтт ; пер. с англ. под ред. Д. С. Лебедева. — М. : Мир, 1982. — Гл. 2, 3, 12, 13, 17, 18, 20.
7. Роджерс Д. Алгоритмические основы машинной графики : пер. с англ. / Д. Роджерс. — М. : Мир, 1989. — 503 с.
8. Роджерс Д. Математические основы машинной графики / Д. Роджерс, Дж. Адамс ; пер. со 2-го англ. изд. П. А. Монахова [и др.]. — М.: Мир, 2001. — 604 с.
9. Тихомиров Ю. Программирование трехмерной графики : OpenGL, создание реалистических образов / Ю. Тихомиров. СПб.: ВHV-Санкт-Петербург, 1998. — 245 с.
10. Фоли Дж. Основы интерактивной машинной графики : в 2 кн. / Дж. Фоли. — М.: Мир, 1987. — Кн. 1. — 367 с. ; Кн. 2. — 365 с.
11. Шикин Е. В. Компьютерная графика. Динамика, реалистические изображения / Е. В. Шикин, А. В. Боресков. — М.: ДИАЛОГ-МИФИ, 1995. — 286 с.
12. Шикин Е. В. Компьютерная графика. Полигональные модели / Е. В. Шикин, А. В. Боресков. — М.: Диалог-МИФИ, 2000. — 461 с.
13. Эйнджел Э. Интерактивная компьютерная графика : ввод. курс на базе OpenGL / Эдвард Эйнджел ; пер. с англ. и ред. В. Т. Тертышного. — 2-е изд. — М.: Вильямс, 2001. — 590 с.

## ДОДАТКИ

### Додаток А.

#### Робота додатків в операційній системі Windows

Для взаємодії додатків з ОС Windows використовується універсальний механізм доступу до функцій програмного інтерфейсу Win32 API. Цей механізм є прошарком між програмою користувача і операційною системою. Він забезпечує стандартизований доступ до функцій та інструментів операційної системи. В той же час забезпечується необхідний рівень безпеки операційної системи. Крім того, ОС Windows є багатозадачною операційною системою. Для взаємодії додатків один з одним і з ОС реалізовано механізм на основі подій – обмін повідомленнями. З точки зору додатка, це повідомленням про те, що відбулась якась подія, яка може вимагати виконання певних дій, або носить лише інформаційний характер. Подія може бути наслідком дій користувача, наприклад, переміщення курсора або клацання кнопкою миші, зміни розмірів вікна або вибору пункту меню. Крім того, подія може генеруватися додатком, а також операційною системою. Повідомлення з точки зору програми – це структура даних, що містить наступні елементи:

- а) дескриптор (описувач) вікна, якому адресоване повідомлення;
- б) код (номер) повідомлення;
- в) додаткову інформацію, яка залежить від коду повідомлення.

З моменту старту програми операційна система створює в пам'яті глобальний об'єкт, названий системною чергою повідомлень. Усі повідомлення, що генеруються як апаратурою, так і додатками, поміщаються в цю чергу. ОС періодично опитує цю чергу і якщо вона не порожня, посилає чергове повідомлення потрібному адресату, який визначається за допомогою дескриптора вікна.

Точкою входу в віконний Win32 додаток, створений мовою C++, є функція `winMain()` (`_tWinMain`). Для роботи з Win32 API необхідно підключити заголовний файл `#include <windows.h>`.

Для обробки додатком повідомлень, які поступають до нього, використовується віконна процедура. Вона зазвичай має заголовок зі стандартним синтаксисом:

```
LRESULT CALLBACK ім'я функції( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam ).
```

У цьому визначенні LRESULT – тип значення, що повертається, hWnd – дескриптор вікна, якому адресоване повідомлення, uMsg – код повідомлення, wParam і lParam – параметри повідомлення. Ім'я функції може бути довільним, але для головного вікна програми зазвичай використовується ім'я WndProc.

Неодмінним компонентом усіх Windows-додатків є також цикл обробки повідомлень. Функція WinMain зазвичай містить виклики функцій для ініціалізації і створення вікон (InitInstance, RegisterClass), після чого слідує цикл обробки повідомлень і необхідний код, щоб додатки закрити. Витяг повідомлення з черги здійснює функція PeekMessage(). Якщо чергове повідомлення має код WM\_QUIT, то відбувається вихід з циклу, після чого додаток завершує свою роботу. Якщо чергове повідомлення не є повідомленням WM\_QUIT, то воно передається до функції DispatchMessage(), яка повертає повідомлення назад в ОС Windows. Windows відправляє повідомлення для його обробки відповідній віконній процедурі – іншими словами, Windows викликає віконну процедуру. Після повернення з віконної процедури Windows передає управління оператору, який розміщується після DispatchMessage(), і робота циклу продовжується.

**Додаток Б.**  
**Довідкові відомості з мови HLSL**

Таблиця Б.1 – Математичні функції мови HLSL

<code>abs(x)</code>	Абсолютна величина (per-component).
<code>acos(x)</code>	Повертає арккосинус кожного компонента $x$ . Кожен компонент повинен бути в діапазоні $[-1, 1]$ .
<code>asin(x)</code>	Повертає арксинус кожного компонента $x$ . Кожен компонент повинен бути в діапазоні $[-\pi / 2, \pi / 2]$ .
<code>atan(x)</code>	Повертає арктангенс кожного компонента $x$ . Кожен компонент повинен бути в діапазоні $[-\pi / 2, \pi / 2]$ .
<code>ceil(x)</code>	Повертає найменше ціле число, яке більше ніж, або дорівнює $x$ .
<code>cos(x)</code>	Повертає косинус $x$ .
<code>cosh(x)</code>	Повертає гіперболічний косинус $x$ .
<code>ddx(x)</code>	Повертає часткову похідну $x$ щодо screen-space $x$ -координати.
<code>ddy(x)</code>	Повертає часткову похідну $x$ щодо screen-space $y$ -координати.
<code>degrees(x)</code>	Конвертація $x$ з радіани в градуси.
<code>distance(a, b)</code>	Повертає відстань між двома точками $a$ і $b$ .
<code>dot(a, b)</code>	Повертає dot product двох векторів $a$ і $b$ .
<code>floor(x)</code>	Повертає найбільше ціле число, яке є менше ніж, або рівне $x$ .
<code>fwidth(x)</code>	Повертає $\text{abs}(\text{ddx}(x)) + \text{abs}(\text{ddy}(x))$ .
<code>len(v)</code>	Векторна довжина.
<code>length(v)</code>	Повертає довжину вектора $v$ .



<code>lerp(a, b, s)</code>	Повертає $a + s(b - a)$ .
<code>log(x)</code>	Повертає логарифм $x$ .
<code>log10(x)</code>	Повертає десятковий логарифм $x$ .
<code>mul(a, b)</code>	Робить матричне множення між $a$ і $b$ .
<code>normalize(v)</code>	Повертає нормалізований вектор $v$ .
<code>pow(x, y)</code>	Повертає $x^y$ .
<code>radians(x)</code>	Конвертує $x$ з градусів в радіани.
<code>reflect(i, n)</code>	Повертає вектор відображення.
<code>refract(i, n, eta)</code>	Повертає вектор заломлення.
<code>rsqrt(x)</code>	Повертає $1 / \sqrt{x}$ .
<code>sin(x)</code>	Повертає синус $x$ .
<code>sincos(x, out s, out c)</code>	Повертає синус і косинус $x$ .
<code>sinh(x)</code>	Повертає гіперболічний синус $x$ .
<code>sqrt(x)</code>	Повертає квадратний корінь (per-component).
<code>step(a, x)</code>	Повертає $(x = a) ? 1 : 0$ .
<code>tan(x)</code>	Повертає тангенс $x$ .
<code>tanh(x)</code>	Повертає гіперболічний тангенс $x$ .

Таблиця Б.2 – Функції читання даних з текстур для мови HLSL

<code>tex1D(s, t)</code>	Читання з одновимірної текстури. $s$ - sampler. $t$ - скаляр.
<code>tex1D(s, t, ddx, ddy)</code>	Читання з одновимірної текстури, з похідними. $s$ - sampler. $t$ , $ddx$ , і $ddy$ - скаляри.
<code>tex1Dproj(s, t)</code>	Читання з одновимірної проєктивної текстури. $s$ - sampler. $t$ - 4D вектор, $t$ ділиться на $tw$ перед виконанням функції.

<code>tex1Dbias(s, t)</code>	Читання з одновимірної текстури зі зміщенням, <code>s</code> - sampler, <code>t</code> - 4-х мірний вектор, міп-рівень зміщується на <code>tw</code> до того, як проводиться пошук.
<code>tex2D(s, t)</code>	Читання з двовимірної текстури. <code>s</code> - sampler. <code>t</code> - 2D вектор.
<code>tex2D(s, t, ddx, ddy)</code>	Читання з двовимірної текстури, з похідними. <code>s</code> - sampler. <code>t</code> - 2D текстурні координати. <code>ddx</code> , <code>ddy</code> - 2D вектора.
<code>tex2Dproj(s, t)</code>	Читання з двовимірної проєктивної текстури. <code>s</code> - sampler. <code>t</code> - 4D вектор, <code>t</code> ділиться на <code>tw</code> перед виконанням функції.
<code>tex2Dbias(s, t)</code>	Читання з двовимірної текстури зі зміщенням, <code>s</code> - sampler, <code>t</code> - 4-х мірний вектор, міп-рівень зміщується на <code>tw</code> до того, як проводиться пошук.
<code>tex3D(s, t)</code>	Читання з тривимірної текстури. <code>s</code> - sampler. <code>t</code> - 3D вектор.
<code>tex3D(s, t, ddx, ddy)</code>	Читання з тривимірної текстури, з похідними. <code>s</code> - sampler. <code>t</code> - 3D текстурні координати. <code>ddx</code> , <code>ddy</code> - 3D вектора.
<code>tex3Dproj(s, t)</code>	Читання з тривимірної проєктивної текстури. <code>s</code> - sampler. <code>t</code> - 4D вектор. <code>t</code> ділиться на <code>tw</code> перед виконанням функції.
<code>tex3Dbias(s, t)</code>	Читання з тривимірної текстури зі зміщенням, <code>s</code> - sampler, <code>t</code> - 4-х мірний вектор, міп-рівень зміщується на <code>tw</code> до того, як проводиться пошук.
<code>texCUBE(s, t)</code>	Читання з кубічної текстури. <code>s</code> - sampler, <code>t</code> - 3D текстурні координати.
<code>texCUBE(s, t, ddx, ddy)</code>	Читання з кубічної текстури. <code>s</code> - sampler. <code>t</code> - 3D текстурні координати. <code>ddx</code> , <code>ddy</code> - 3D вектора.

<code>texCUBEproj(s, t)</code>	Читання з кубічної проєктивної текстури. <code>s</code> - sampler, <code>t</code> - 4D вектор, <code>t</code> ділиться на <code>tw</code> перед виконанням функції.
<code>texCUBEbias(s, t)</code>	Читання з кубічної текстури. <code>s</code> - sampler, <code>t</code> - 4D вектор, міп-рівень зміщується на <code>tw</code> до того, як проводиться пошук.

`t` – текстурні координати. `ddx`, `ddy` – похідні.

Таблиця Б.3 – Основні семантичні типи для вхідних параметрів вершиного шейдеру

<code>POSITIONn</code>	Позиція
<code>BLENDWEIGHTn</code>	Ваговий коефіцієнт
<code>BLENDINDICESn</code>	Індекс вагової матриці
<code>NORMALn</code>	Нормаль
<code>PSIZEn</code>	Розмір точки
<code>COLORn</code>	Колір
<code>TEXCOORDn</code>	Текстурні координати
<code>TANGENTn</code>	Тангент
<code>BINORMALn</code>	Бінормаль
<code>TESSFACTORn</code>	Фактор тесселяції

## Додаток В

### Мікропрограми мовою HLSL для розрахунку освітленості за правилами Ламберта і Фонга

```
//матриці перетворень: світова, зворотна видова і загальна
float4x4 world,  iview,  mvp;
//напрямок падаючих променів світла
float4  ldir;

//колір прямого і фонового джерел світла
float4  lcolor,  acolor;

//інтенсивність і розмір (ступінь) відблиску
float  slevel,  spower;
//колір матеріалу
float4  color;

//вхідні дані вершинного шейдера: позиція і нормаль
struct s_input
{
    float4  pos: POSITION;
    float3  nor: NORMAL;
};

//вихідні дані вершинного шейдера:
//позиція, нормаль, вектор напрямку на спостерігача
struct s_output
{
    float4  pos: POSITION;
    float3  nor: TEXCOORD0;
    float3  eye: TEXCOORD1;
};

//вершинний шейдер
s_output vs_simple( s_input i )
{
    s_output o;
    o.pos = mul( i.pos, mvp );
    //нормаль у світовому просторі
    o.nor = mul( i.nor, (float3x3)world );

    //напрямок на спостерігача (уздовж осі Z)
    //з видового в світовий простір
    o.eye = mul( float3(0, 0, -1), (float3x3)iview );

    return o;
}
```

```

//правило Ламберта L.N
float lambert_law( float3 L, float3 N )
{
    return max( 0, dot ( -L, N ) );
}

//правило Фонга (R.V)^n
float phong_law( float3 L, float3 N, float3 V )
{
    float3 R = reflect( L, N );

    return slevel * pow( max( 0, dot(V, R) ), spower );
}

//Фрагментний шейдер
float4 ps_phong( s_output2 i ): COLOR
{
    float3 color = v_obj_color.rgb;

    //усі вектори для розрахунків беруться
    //у нормалізованому вигляді
    float3 N = normalize( i.nor );
    float3 V = normalize( i.eye );
    float3 L = normalize( ldir.xyz );

    //розрахунок 3-х компонент освітлення:
    //фонового, розсіяного і дзеркального
    float3 ambient = acolor.rgb;
    float3 diffuse = lambert_law( L, N ) * lcolor.rgb;
    float3 specular = phong_law( L, N, V ) * lcolor.rgb;

    return float4( (ambient + diffuse) * color + specular, 1 );
}

```

Навчальне видання

ЗУЄВ Андрій Олександрович  
ЄВСЕЄНКО Олег Миколайович  
КРИЛОВА Вікторія Анатоліївна

ОСНОВИ КОМП'ЮТЕРНОЇ ГРАФІКИ.  
ЧАСТИНА 2

**Методичні вказівки**

до виконання практичних та лабораторних робіт  
для студентів спеціальностей  
151 «Автоматизація та комп'ютерно-інтегровані технології»  
172 «Телекомунікація та радіотехніка»

Відповідальний за випуск Качанов П.О.  
Роботу до друку рекомендував О.В. Дудник

Редактор Самініні О.С.

План 2020 р., Поз. 57

Підписано до друку . Формат 60'84 1/16. Папір друк. № 2.  
Друк - ризографія. Гарнітура Times New Roman. Умов. друк. арк. 3,2.  
Обл.-вид. арк. 2,7. Наклад 100 прим. Зам. № . Ціна договірна.

---