Лабораторна робота № 6

Створення та використання шейдерних програм.

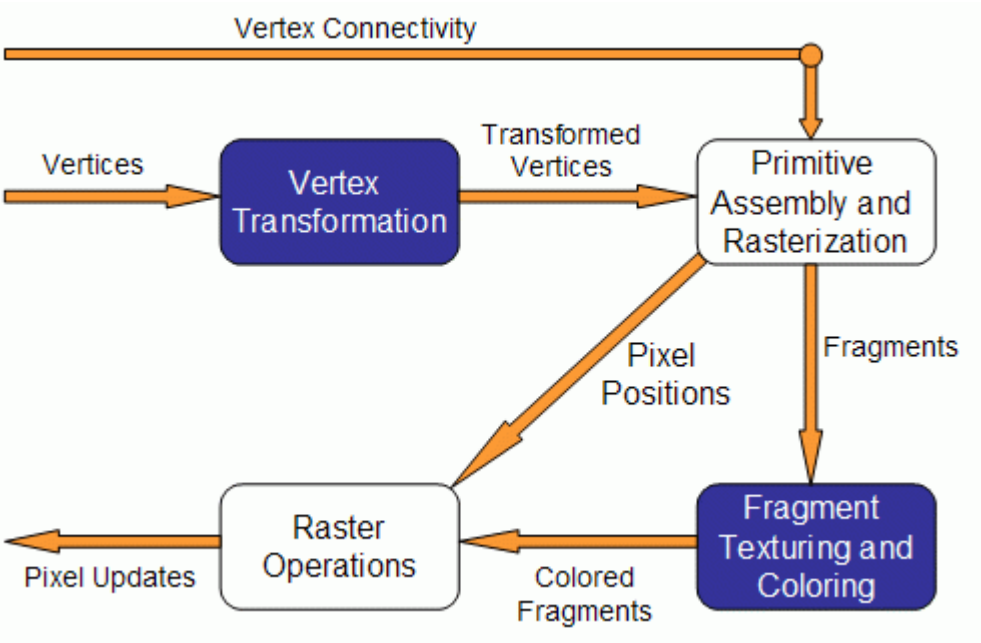
Мета роботи: Вивчити та набути практичних навичок у використанні шейдерних програм.

Завдання до роботи: Розробити програму в якій будуть використані шейдерні програми для зображення об’єктів.

Теоретичні відомості

Шейдери.

Шейдером (англ. Shader) називається програма для однієї із ступенів графічного конвеєра, використовувана для визначення остаточних параметрів об'єкта або зображення. Шейдерні мови містять спеціальні типи даних (матриці, вектори, семплери), а також вбудовані змінні і константи для зручної інтеграції зі стандартною функціональністю 3D API.



Шейдерні мови, що використовуються для завдань професійного рендеринга (кіно, телебачення), такі як RenderMan і Gelato націлені на досягнення максимальної якості візуалізаціїзображення. Вони зазвичай вимагають великих обчислювальних потужностей і не призначені для рендеринга в реальному часі. Потреба в шейдерах для візуалізації в реальному часі виникла з появою графічних прискорювачів, коли розробники

виявилися обмеженими тим набором ефектів, який закладений в графічне апаратне забезпечення. Шейдерні мови, що використовуються для візуалізації в реальному часі, такі як GLSL (Використовується в OpenGL), HLSL (іспользуется в DirectX), Cg націлені на використання обчислювальних потужностей сучасних графічних прискорювачів. Вони дозволяють замінити частину стандартних механізмів графічного конвеєра з обробки вершин і фрагментів, а також (з появою геометричних шейдерів) генерувати нові примітиви на основі даних, відправлених на вхід графічного конвеєра. У даній лабораторній роботі Ви познайомитеся з мовою програмування шейдеров GLSL, а також з інтеграцією шейдерних програм з додатками на OpenGL.

Основи використання мови програмування шейдерів GLSL в OpenGL-додатках.

OpenGL надає програмісту досить гнучкий, але статичний інтерфейс для малювання графіки. Шейдери надають програмісту перевизначити стандартний механізм обробки графіки на певних етапах роботи графічного конвеєра, що дозволяє застосовувати просунуті технології візуалізації в реальному часі.

Функції стандартного конвеєра OpenGL здійснюють над кожною вершиною ряд перетворень:

- Трансформація координат вершин і нормалей в систему координат спостерігача
- Розрахунок кольору вершини на основі матеріалу і джерел світла
- Проектування вершини і перспективне проектування
- Генерування текстурних координат

Вершинний процесор це програмований модуль, який виконує операції над вхідними значеннями вершин та іншими пов'язаними з ними даними. Вершинний процесор виконує:

- перетворення вершин;
- перетворення і нормалізація нормалей;
- генерування і перетворення текстурних координат; налаштування освітлення;
- накладення кольору на матеріал

Фрагментний процесор це програмований модуль, що виконує операції над фрагментами та іншими пов'язаними з ними даними. Фрагментний процессор виконує такі стандартні операції:

- операції над інтерпольованої значеннями;
- доступ до текстур;
- накладення текстур;
- створення ефекту туману;
- змішування кольорів.

Фрагментний процесор не замінює наступні операції:

- перевірка на видимість;
- відсікання по прямокутнику (scissors test);
- тест трафарету;
- тест прозорості;
- тест глибини;
- відсікання по трафарету;
- змішування кольорів;
- визначення видимості площин.

Функції для роботи з шейдерами.

Для створення шейдера скористаємося функцією glCreateShader, передавши їй тип шейдера як іменовану константу GL_VERTEX_SHADER або GL_FRAGMENT_SHADER. Для створення програмного об'єкта скористаємося функцією glCreateProgram. Після його створення можна приєднати до нього шейдер за допомогою функції glAttachShader.

На даному етапі нам необхідно сформувати початковий код шейдера в пам'яті у вигляді масиву символів і передати його OpenGL за допомогою функції glShaderSource. Після того, як в шейдер буде завантажений вихідний код, його можна відкомпілювати за допомогою функції glCompileShader. Процес компіляції може здійснюватися паралельно з роботою програми. Для отримання статусу останньої операції компіляції, а також отримання іншої інформації про шейдери, слід скористатися функцією glGetShader. У тому випадку, якщо компіляція шейдера була завершена невдачею, програма перерве свою роботу, згенерувавши виняткову ситуацію.

Після успішного завершення компіляції шейдеров, приєднаних до програми, можна виконати її компонування за допомогою функції glLinkProgram. Компонування шейдерной програми також може відбуватися

паралельно з роботою програми. Для отримання статусу останньої операції компонування можна скористатися функцією glGetProgram. Після того, як програма скомпонована, можна видалити пов'язані з нею шейдери, попередньо від'єднавши їх за допомогою фукнція glDetachShader. Для її використання шейдери не потрібні.

Функція glGetShaderInfoLog дозволяє одержати вміст «інформаційного журналу» шейдера - рядок з діагностичною інформацією, помилками та попередженнями, виявленими в процесі компіляції шейдера. Дана інформація може бути корисною в процесі розробки вихідного коду шейдера, так як на її основі програмісту буде легше виявити і виправити в своєму коді помилки та інші «підозрілі» фрагменти.

Функція glValidateProgram здійснює перевірку можливості виконання шейдерної програми в поточному стані OpenGL. Інформація про валідність програми зберігається всередині стану програми та може буде отримана за допомогою функції glGetProgram з аргументом GL_VALIDATE_STATUS.

Компіляція шейдера може виконуватися OpenGL асинхронно (паралельно з роботою програми). Для того, щоб дізнатися, чи успішно закінчилася остання компіляція шейдера, слід скористатися функцією glGetShader, передавши їй в якості ідентифікатора параметра шейдера значення GL_COMPILE_STATUS.

Снтаксис мови GLSL

Оголошення змінних аналогічно C, тобто можуть оголошуватися в будь-якому місці програми, та їх видимість залежить від області, де вони були оголошені. Крім цього, в GLSL введені спеціальні типи змінних для зв'язку шейдера з зовнішнім світом (uniform), фрагментного шейдера з вершинним шейдером (varying) і змінні-атрибути вершина (attribute). Такі змінні повинні мати глобальну область видимості.

У GLSL є цикли - for, while, do .. while; оператори переходу - continue, break, return, discard (заборона виведення фрагмента, тільки в fs). Синтаксис аналогічний C. Оголошення звичайних (не спеціальних) змінних відбувається наступним чином:

```
type variableName;
```

де type:

- float - тип числа з плаваючою точкою одинарної точності;
- int - цілочисельний тип;

- bool - логічний тип (приймає значення true, false);
- vec2, vec3, vec4 - float-вектор (розмірність 2, 3, 4 соотв.);
- ivec2, ivec3, ivec4 - int-вектор;
- bvec2, bvec3, bvec4 - bool-вектор;
- mat2, mat3, mat4 - float-матриця (розмірність 2x2, 3x3, 4x4 соотв.);

Крім того, type можна використовувати і як функцію перетворення типів (конструктор):

```
float f = 0.9;
vec4 v4 = vec4 (f, 0.5, f, 1); / / v4 = [0.9, 0.5, 0.9, 1]
vec3 v3 = vec3 (v4); / / v3 = [0.9, 0.5, 0.9]
```

Також типом змінної може бути структура:

```
struct structName {... } VariableName1, variableName2, ... ;
```

Ця декларація оголошує новий тип structName, яким можна користуватися так само, як і вбудованими типами.

Можна задавати константи та функції:

```
  const float ex1 = 1.5;
const vec4 ex2 = vec4 (1, 0, 0, 0);
const vec4 ex2 = vec4 (1, 2, vec2 (0, 1));
```

Параметри функції оголошуються як

```
[Const] convention paramName
```

де convention:

in - створюється локальна копія параметра для читання-запису (обрано за умовчанням), якщо присутній const, то запис заборонена;

inout - локальна копія не створюється, читання / запис відбувається з зовнішньої змінної (як за посиланням);

out - так само, як і inout, тільки читання параметра заборонено;

```
vec4 myFunc1 (const in vec4 v1) {... };
void myFunc2 (in vec4 v1, out float f1) {... };
```

У GLSL існує безліч операторів: алгебраїчні, дереференсірованіе масиву, структури, бітові, логічні ... всі вони розбиті по пріоритетах, докладніше дивіться у специфікації [1]. Більшість алгебраїчних операторів виконуються покомпонентно (component-wise), але є і виключення - множення вектора на матрицю, матрицю на вектор і матриці на матрицю виконуються як у лінійній алгебрі.

Існують також покомпонентний функції порівняння векторів (lessThan, lessThanEqual, notEqual, тощо), вони, на відміну від стандартних операцій порівняння <, <=,> =,>, ==,! =, Які повертають boolean-значення, повертають boolean-вектор з розмірністю, рівної розмірності аргументів.

Uniform, Varying, Attribute змінні шейдерів.

Uniform-змінні. Використовуються для передачі рідко змінюваних даних як до вершиного, так і до фрагментного рейдерів. Uniform-змінні не можуть задаватися між викликами glBegin () і glEnd (). OpenGL підтримує як вбудовані, так і визначені користувачем uniform-змінні Для передачі значення uniform-змінної програма має спочатку визначити розташування даної змінної (індекс) за іменем. При оголошенні uniform змінних з однаковими іменами у вершинному та фрагментному шейдері вони вважаються спільними для даних шейдерів.

Типами uniform-змінних можуть бути type або samplerType (samplerType може використовуватися тільки під фрагментного шейдери):

```
uniform (type | samperType) uniformName;
```

де samplerType:

`sampler1D, sampler2D, sampler3D, samplerCube, sampler1DShadow, sampler2DShadow` - семплери, використовуються для читання з текстури;

Для читання з текстури існує кілька функцій:

`texture [123] D [Proj, Lod, ProjLod]` - читання з текстури. при постфікс Lod потрібно вказувати mip-рівень, при Proj tc діляться на їх останню компоненту;

`textureCube [Lod]` - читання з cubemap-текстури;

`shadow [12] D` - читання з shadowmap-текстури, порівняння відбувається автоматично при читанні;

Наприклад:

```
vec4 t0 = texture2D(uniform sampler2D mySampler2D, gl_TexCoords [0]);
vec4 t1 = textureCube(uniform samplerCube mySamplerCUBE, gl_TexCoords [1]);
```

Varying-змінні. Дані в varying-змінних передаються з вершинного шейдера до фрагментного. Бувають як вбудованими, так і визначеними користувачем. Для кожної вершини існує власне значення відповідної varying-змінної. В процесі растеризації відбувається інтерполяція значень varying-змінних з урахуванням перспективи. Писати в varying-змінну можна тільки в вершинному шейдері. Коли читаємо varying в фрагментному

шейдері видається інтерпольоване між вершинами значення. Типом varying-змінної може бути тільки float-тип: число, вектор, матриця або масив.
Наприклад:

```
varying vec2 my_vec2;
varying float my_f4 [4];
```

Attribute-змінні вершинного шейдера представляють собою дані, передані вершинного шейдера від програми Можуть задавати значення атрибутів або між glBegin () / glEnd (), або за допомогою функцій, що працюють з вершинними масивами.

Attribute-змінні існують лише у вершинного шейдера, це аналоги ATTRIB-змінних в ARB_vertex_program, однак, є й відмінність - якщо в ARBvp ми жорстко ставимо номер generic-атрибуту, по якому будуть розташовуватися дані, то в GLSL компілятор сам вирішує, які generic-атрибути для чого використовувати. Типом attribute-змінної можуть бути тільки float, vec2, vec3, vec4, mat2, mat3 і mat4. Масиви та структури не допускаються.
Наприклад:

```
attribute vec4 my_vec4;
attribute mat4 my_mat4;
```

Крім того, в шейдерах присутня велика кількість вбудованих атрибутів самих різних типів, які представляють собою поточний OpenGL state (наприклад, gl_Color).

Вхідні та вихідні дані вершинного процесора.

Вбудовані attribute змінні: gl_Color, gl_Normal, gl_Vertex, gl_MultiTexCoord0 тощо.

Вбудовані uniform-змінні: gl_ModelViewMatrix, gl_FrontMaterial, gl_LightSource [0 .. n], gl_Fog тощо.

Вбудовані varying-змінні: gl_FrontColor, gl_BackColor, gl_FogFragCoord тощо.

Спеціальні вихідні змінні: gl_Position, gl_PointSize, gl_ClipVertex

Вхідні і вихідні дані фрагментного процесора.

Вбудовані varying-змінні: gl_Color, gl_SecondaryColor, gl_TexCoord [0 .. n], gl_FogFragCoord.

Спеціальні вхідні змінні: gl_FragCoord, gl_FrontFacing.

Спеціальні вихідні змінні: gl_FragColor, gl_FragDepth

Вбудовані uniform-змінні: gl_ModelViewMatrix, gl_FrontMaterial, gl_LightSource [0 .. n], gl_Fog і т.п.

Карти текстур.

Передача параметрів шейдерній програмі через uniform-змінні

Шейдер може отримати доступ до стану OpenGL через вбудовані uniform-змінні, починаються з префікса «gl_». Наприклад, до поточної матриці моделювання-вигляду можна звернутися по імені gl_ModelViewMatrix. Програма також може визначати свої uniform- змінні і використовувати спеціальні команди OpenGL для встановлення їх значень. Загальна кількість вбудованих uniform-змінних, доступних вершинами і фрагментного процесорам, не може бути більше деякого встановленого реалізацією OpenGL максимуму, що задається в компонентах розміру float. Наприклад, тип vec2 складається з двох компонентів типу float.

Для того, щоб передати до шейдеру значення uniform-змінної, необхідно після компіляції та лінковки шейдерної програми викликати функцію glGetUniformLocation, передавши їй дескриптор програмного об'єкта та рядок з іменем заданої uniform-змінної. В результаті такого виклику функції отримаємо ціле число, що задає розташування даної uniform-змінної в програмі. Якщо такої змінної немає серед активних uniform-змінних програми, або ім'я починається з зарезервованого префікса «gl_», функція поверне значення –1. Інформацію про активну uniform-змінну з заданим індексом можна отримати у скомпонованого програмного об'єкту за допомогою функції glGetActiveUniform. Uniform-змінна є активною, якщо її значення використовується в рейдері, компілятор може відкинути деякі змінні якщо в процесі компіляції з'ясується, що їх значення не впливає на роботу шейдера. Встановити або отримати значення uniform-змінної можна шляхом використання фінкцій glUniform[1234]{ifd}{v} та glGetUniform відповідно.

Порядок виконання лабораторної роботи

1. Створити консольну програму що використовує для малування об’єктів.

2. Зобразити фігуру з набору моделей приведених у папці моделі з використанням механізму розширення буферів (додаткові бали) або без цього механізму (без додаткових балів, за вибором студента).
3. Створити, відкомпілювати та виконати простий вершинний рейдер, що виконує модельне та перспективне перетворення координат.
4. Створити та відкомпілювати та виконати простий фрагмент ний шейдер рейдер, що виконує формування вихідного кольору зображення.
5. Використати спільну для обох рейдерів uniform-змінну що відповідає за модельний час та впливає на вихідне зображення.

Контрольні питання

1. Вершинний шейдер, вхідні та вихідні параметри, синтаксис, задачі.
2. Фрагментний шейдер, вхідні та вихідні параметри, синтаксис, задачі.
3. Поєдннння рейдерів у програму, компіляція та лінковка, перевірка стану шейдера та отримання статуса компіляції.
4. Varaing, attribute, uniform змінні. Призначення різних типів змінних.
5. Синтаксис мови GLSL та базові типи даних.
6. Отримання та встановлення значень uniform змінних.