**Лабораторная работа №6**

Использование перспективной проекции. Создание трехмерных объектов. Матрица обзора. Методы затенения Гуро и Фонга.

Цель работы: Рассмотреть применение OpenGL для визуализации трехмерной сцены. Изучить и применить матрицу обзора. Применение перспективной проекции. Алгоритмы затенения Гуро и Фонга.

*Теоретические сведения*

Матрица обзора – матрица, задающая точку, из которой ведется обзор, и направление обзора.

Перспективная проекция – проекция. при которой размеры объектов визуально изменяются в зависимости от дальности обзора.

Матрица обзора может задаваться несколькими способами. Рассмотрим создание матрицы обзора на примере матрицы LookAt:

Вектор U (up vector) задает вертикальную ось обзора.

Вектор D (direction vector) задает направление обзора.

Вектор P (position vector) задает позицию обзора.

Вектор R (right vector) получается в результате векторного произведения векторов U и D.

**Метод затенения по Фонгу**

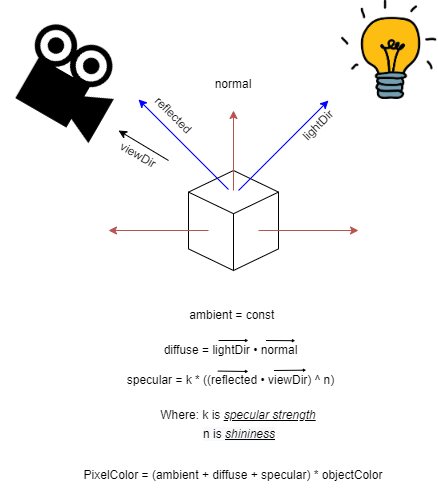


Рис. 1. Затенение по Фонгу

Применение затенения по Фонгу подразумевает представление освещения в виде трех компонентов:

1. Ambient – компонент, отвечающий за интенсивность освещения при отдалении источников света на большое расстояние. Принимает постоянные значения.
2. Diffuse – компонент, рассчитывающий влияние рассеянной энергии источника света на цвет объекта.
3. Specular – компонент, рассчитывающий влияние отраженной энергии источника света на цвет объекта.

Рассмотрим вектора, использующиеся для расчетов:

1. Вектор lightDir задается следующей формулой:

Где: LightPosition – точка, в которой находится источник света. PixelPosition – точка с координатами пикселя, который визуализирует освещаемую поверхность. При расчете алгоритма затенения по Фонгу вектор lightDir рассчитывается для каждого пикселя.

1. Вектор viewDir задается следующей формулой:

Где: CameraPosition – точка, из которой происходит обзор сцены. PixelPosition – точка с координатами пикселя, который визуализирует освещаемую поверхность. При расчете алгоритма затенения по Фонгу вектор viewDir рассчитывается для каждого пикселя.

1. Вектор normal является нормалью к освещаемой плоскости.
2. Вектор reflected является отраженным вектором (-lightDir) относительно нормали normal.

Для расчета рассеянной (diffuse) составляющей света используется вектор направления освещения (lightDir) и нормаль (normal), проведенная от освещаемой поверхности. Скалярное произведение векторов lightDir и normal определяют рассеянную (diffuse) составляющую света. Скалярное произведение векторов viewDir и reflected определяет отраженную (specular) составляющую света.

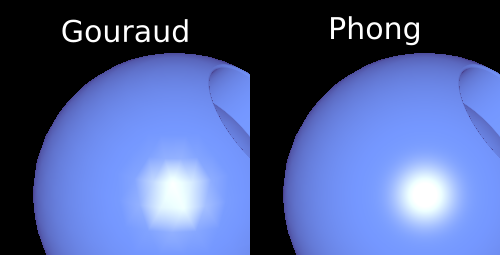


Рис. 2. Сравнение методов Гуро и Фонга

Освещение методом Гуро отличается от освещения методом Фонга тем, что вектора lightDir и viewDir рассчитываются для каждой вершины фигуры, а не для каждого пикселя. Таким образом, при использовании метода Гуро освещение рассчитывается в вершинном шейдере, а при использовании метода Фонга – в пиксельном.

*Методические указания*

Модифицируем вершинный шейдер для загрузки вершины состоящей из трех значений координат, а также загрузки нормалей. Добавим новую uniform переменную view, которая будет содержат матрицу обзора.

...

layout(location = 0) in vec3 vPos; //Координаты вершины примитива

layout(location = 1) in vec3 vColor; //Цвет вершины примитива

layout(location = 2) in vec2 vUV; //UV координаты

layout(location = 3) in vec3 vNormal; //Нормаль

...

uniform mat4 view = mat4(1.0f); //Матрица обзора

...

void main() {

...

gl\_Position = projection \* view \* vec4(vPos, 1.0f);

...

Вершинный шейдер

Проведем небольшой рефакторинг кода, добавив функцию для создания массива данных вершин:

//Массив вершин плоскости

std::array<float, 11 \* 6> CreatePlaneArray() {

static float uv\_max = 1.0f;

static float uv\_min = 0.0f;

//Массив вершин

static std::array<float, 11 \* 6> vertices = {

//координаты цвет UV нормаль

0.5f, 0.5f, 0.0f, 1, 0, 0, uv\_max, uv\_max, 0, 0, 1, //------------

0.5f, -0.5f, 0.0f, 0, 1, 0, uv\_max, uv\_min, 0, 0, 1, //Трегольник 1

-0.5f, -0.5f, 0.0f, 0, 0, 1, uv\_min, uv\_min, 0, 0, 1, //------------

-0.5f, -0.5f, 0.0f, 0, 0, 0, uv\_min, uv\_min, 0, 0, 1, //------------

-0.5f, 0.5f, 0.0f, 0, 1, 0, uv\_min, uv\_max, 0, 0, 1, //Треугольник 2

0.5f, 0.5f, 0.0f, 0, 0, 1, uv\_max, uv\_max, 0, 0, 1 //------------

};

return vertices;

}

Подпрограмма создания массива данных

Также, в целях рефакторинга, добавим процедуру для выставления матрицы преобразований в шейдере:

void SetTransform(GLuint program, glm::vec3 position, glm::vec3 rotation, glm::vec3 scale) {

//Матрица переноса

glm::mat4 translationMat = glm::translate(glm::mat4(1.0f), position);

//Матрица масштабирования

glm::mat4 scaleMat = glm::scale(glm::mat4(1.0f), scale);

//Матрица вращения

glm::mat4 rotateMatX = glm::rotate(glm::radians(rotation.x), glm::vec3(1, 0, 1));

//Матрица вращения

glm::mat4 rotateMatY = glm::rotate(glm::radians(rotation.y), glm::vec3(0, 1, 0));

//Матрица вращения

glm::mat4 rotateMatZ = glm::rotate(glm::radians(rotation.z), glm::vec3(0, 0, 1));

//Матрица преобразований

glm::mat4 modelMat = translationMat \* (rotateMatX \* rotateMatY \* rotateMatZ) \* scaleMat;

glUseProgram(program);//Выбираем активную программу

//Передача матрицы преобразований в шейдер

glUniformMatrix4fv(

glGetUniformLocation(program, "model"),

1,

GL\_FALSE,

glm::value\_ptr(modelMat)

);

}

Процедура создания и передачи матрицы преобразований

Функция SetTransform принимает в себя следующие параметры:

1. program – программа, скомпонованная из шейдеров.
2. position – координаты позиции фигуры.
3. rotation – углы поворота фигуры в градусах.
4. scale – значения масштабирования.

Для каждой вершины плоскости добавляется нормаль: (0, 0, 1).

Введем изменения в код создания буфера:

//Массив вершин

auto vertices = CreatePlaneArray();

...

//Заполнить буфер данными

glBufferData(GL\_ARRAY\_BUFFER, sizeof(vertices), &vertices, GL\_STATIC\_DRAW);

GLsizei stride = 11;

//Определить интерпретацию считывания из буфера для идентификатора 0 в шейдере

glVertexAttribPointer(0, 3, GL\_FLOAT, GL\_FALSE, stride \* sizeof(GL\_FLOAT), (void\*)0);

//Определить интерпретацию считывания из буфера для идентификатора 1 в шейдере

glVertexAttribPointer(1, 3, GL\_FLOAT, GL\_FALSE, stride \* sizeof(GL\_FLOAT), (void\*)(sizeof(float) \* 3));

//Определить интерпретацию считывания из буфера для идентификатора 2 в шейдере

glVertexAttribPointer(2, 2, GL\_FLOAT, GL\_FALSE, stride \* sizeof(GL\_FLOAT), (void\*)(sizeof(float) \* 6));

//Определить интерпретацию считывания из буфера для идентификатора 3 в шейдере

glVertexAttribPointer(3, 3, GL\_FLOAT, GL\_FALSE, stride \* sizeof(GL\_FLOAT), (void\*)(sizeof(float) \* 8));

...

glEnableVertexAttribArray(3);

Модифицированный буфер

Необходимо интерпретировать данные из буфера включая нормали, поэтому воспользуемся функциями glVertexAttribPointer и glEnableVertexAttribArray. Так как мы передаем теперь три координаты вершины, то нужно учесть это во всех вызовах функций glVertexAttribPointer.

Рассмотрим создание матрицы обзора:

...

glm::vec3 cameraPos = {2.5f, 2.5f, 2.5f};

view\_mat = glm::lookAt(cameraPos, glm::vec3(0, 0, 0), glm::vec3(0, 0, 1));

projection\_mat = glm::perspective(glm::radians(90.0f), 800.f/600.f, 0.1f, 1000.f);

...

//Пока окно не закрыто

while (!glfwWindowShouldClose(window))

{

...

//Передача матрицы обзора в шейдер

glUniformMatrix4fv(

glGetUniformLocation(program, "view"),

1,

GL\_FALSE,

glm::value\_ptr(view\_mat)

);

...

SetTransform(program, { 0, 0, 0 }, { 0, 0, 0 }, { 2, 2, 2 });

glDrawArrays(GL\_TRIANGLES, 0, vertices.size() / stride); //Отрисовать vertices.size/11 вершин

Создание матрицы обзора

Функция glm::lookAt создает матрицу обзора, описанную в теоретической справке.

Параметры функции glm::lookAt:

1. Точка из которой ведется обзор.
2. Точка, формирующая направление обзора (точка, в которую “смотрит” камера).
3. Вертикальный вектор (ось Z).

Функция glm::perspective создает матрицу проецирования для применения перспективной проекции.

Параметры функции glm::perspective:

1. Угол поля зрения (Field Of View).
2. Соотношение сторон окна.
3. Ближняя плоскость отсечения.
4. Дальняя плоскость отсечения.

В результате проделанных операций получим результат:

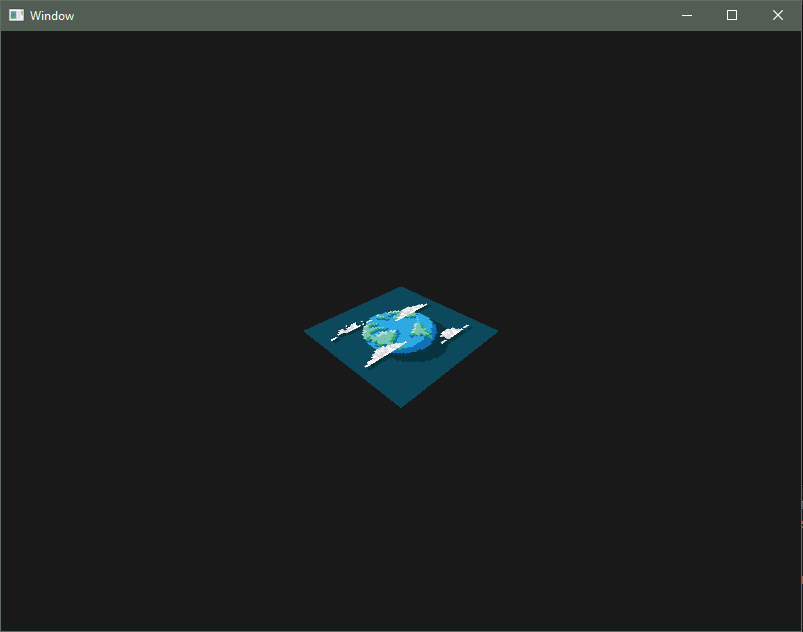


Рис. 3. Визуализация плоскости в трехмерном пространстве

Если мы изменим массив данных, добавив вершины куба, то получим следующий результат:

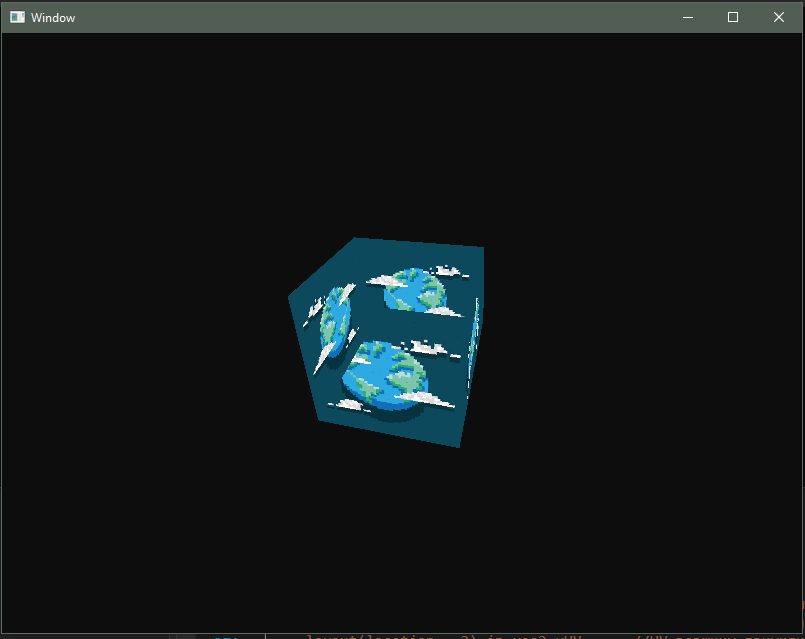


Рис. 4. Визуализация куба без теста глубины

На рис. 4 показана визуализация куба без теста глубины. Тот факт, что вершины могут перекрывать другие при визуализации не учитываются. Поэтому воспользуемся тестом глубины:

...

//Тест глубины

glEnable(GL\_DEPTH\_TEST);

//Пока окно не закрыто

while (!glfwWindowShouldClose(window))

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT); //Очистить экран

...

Тест глубины

Тест глубины включается с помощью функции glEnable. Для теста глубины используется буфер глубины, который хранит значения глубин (дальностей) пикселей. На этапе теста глубины значения глубин сравниваются, и пиксель с меньшей глубиной визуализируется.

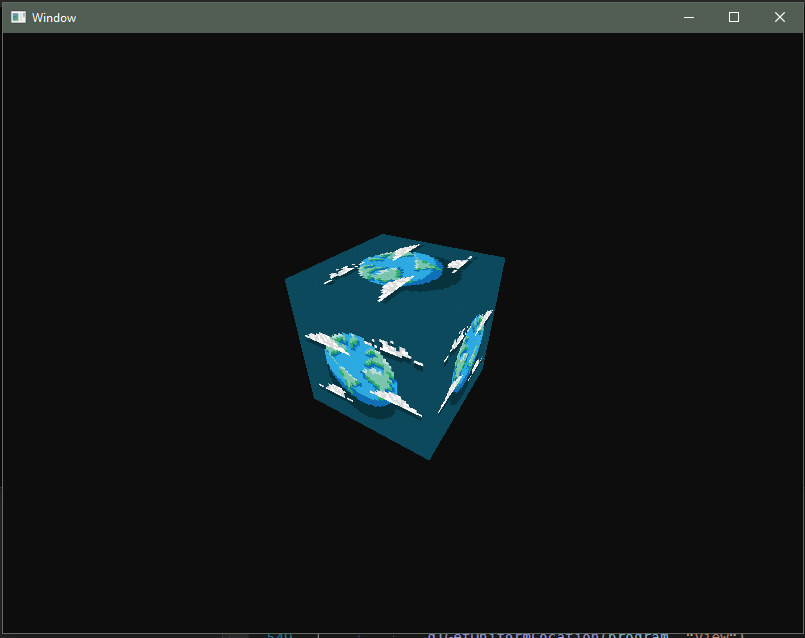


Рис. 5. Визуализация куба с учетом глубины

Приступим к реализации метода освещения Фонга:

...

//Выходные данные вершинного шейдера

out VS\_OUT{

vec3 outColor;

vec2 outUV;

vec3 outNormal;

vec3 fragPos;

}vs\_out;

void main() {

//Позиция вершины после применения матрицы преобразований

vs\_out.fragPos = vec3(model \* vec4(vPos.x, vPos.y, vPos.z, 1.0f));

gl\_Position = projection \* view \* vec4(vs\_out.fragPos, 1.0f);

//Применение преобразований (перемещения, вращения, масштабирования)

//к нормалям

vs\_out.outNormal = mat3(transpose(inverse(model))) \* vNormal;

vs\_out.outColor = vColor;

vs\_out.outUV = vUV;

}

Вершинный шейдер

Рассмотрим две новые выходные переменные вершинного шейдера: fragPos и outNormal.

fragPos – координаты пикселя объекта, после применения преобразований. При передаче данных из вершинного шейдера в пиксельный, данные интерполируются для каждого пикселя. Таким образом, при использовании переменной fragPos в пиксельном шейдере, получим позицию каждого пикселя, покрывающего объект.

outNormal – нормаль визуализированной плоскости. В пиксельный шейдер передается произведение транспонированной обратной матрицы преобразования на входную нормаль. Это произведение позволяет применить геометрические преобразования к исходной нормали.

Произведем расчеты освещения в пиксельном шейдере:

...

//Входные данные пиксельного шейдера

in VS\_OUT{

vec3 outColor;

vec2 outUV;

vec3 outNormal;

vec3 fragPos;

}fs\_in;

out vec4 fragColor; //Выходной цвет пикселя

uniform sampler2D texture\_1;

void main() {

vec3 texture\_color = vec3(texture(texture\_1, fs\_in.outUV));

//Позиция камеры

vec3 cameraPos = vec3(2.5, 2.5, 2.5);

//Направление обзора

vec3 viewDir = normalize(cameraPos - fs\_in.fragPos);

//Позиция источника света

vec3 lightPos = vec3(-1, -1, 2);

//Направления луча света для текущего пикселя

vec3 lightDir = normalize(lightPos - fs\_in.fragPos );

//Цвет источника света

vec3 lightColor = vec3(1, 1, 1);

//Нормирование значений координат нормали

//в промежутке [0, 1]

vec3 nNormal = normalize(fs\_in.outNormal);

vec3 ambient = 0.1f \* lightColor;

//Рассеивание энергии

vec3 diffuse = max(dot(lightDir, nNormal), 0.0f) \* lightColor;

//Отражение энергии

vec3 reflectedLightDir = reflect(-lightDir, nNormal);

vec3 specular = 2 \* pow(max(dot(reflectedLightDir, viewDir), 0.0f), 32) \* lightColor;

//Изменение значения выходного цвета пикселя

fragColor = vec4((ambient + diffuse + specular) \* texture\_color, 1.0f);

}

Пиксельный шейдер

В первую очередь необходимо рассчитать вектора lightDir и viewDir (см. теоретическую справку). Перед расчетом векторов определим точку, из которой ведется обзор cameraPos, и позицию источника света lightPos. Также определим цвет источника света lightColor. Вектора lightDir и viewDir рассчитываются как разности точек (см. теоретическую справку). Полученные значения нормируются.

Далее рассчитываются три компоненты света: ambient, diffuse, specular.

Эти три компоненты имеют тип данных vec3, то есть содержат трехмерный вектор.

Все три компоненты умножаются на цвет источника света.

Ambient компонента рассчитывается как произведение константы на цвет источника света.

Diffuse компонента рассчитывается как скалярное произведение луча, проведенного от источника света к пикселю, умноженное на цвет источника света. Функция dot в языке glsl рассчитывает скалярное произведение. Функция max используются для выбора переменной с максимальным значением.

Перед расчетом компоненты specular необходимо рассчитать вектор reflectedDir. Для этого используется встроенная функция reflect, рассчитывающая отраженный вектор относительно нормали.

Скалярное произведение векторов reflectedDir и viewDir определяет компоненту specular.

Компоненты ambient, diffuse, specular суммируются и домножаются на цвет пикселя текстуры, тем самым определяя конечный цвет пикселя.

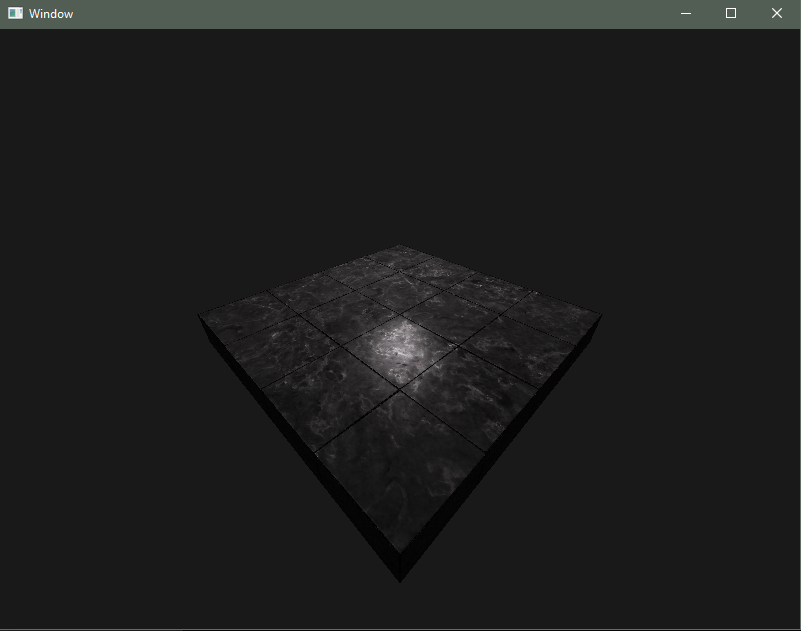


Рис. 6. Применение алгоритма освещения по Фонгу

*Подготовка к работе*

1. Загрузить и скомпилировать исходный код программы примера.
2. С помощью директивы #define USE\_ORTHO применить ортографическую проекцию. Сравнить с результатом без использования директивы.
3. С помощью директивы #define USE\_GOURAUD применить освещение методом Гуро. Сравнить с результатом без использования директивы.

*Выполнение работы*

|  |  |
| --- | --- |
| Вариант | Задание |
| 1 | Реализовать возможность передвижения объектов по сцене следующим образом: по нажатию кнопки “стрелка” камера начинает смотреть на следующий объект, кнопки WASD перемещают объект по сцене (при этом камера следует за объектом). Учесть изменение переменной cameraPos в пиксельном шейдере. |
| 2 | Реализовать загрузку трехмерной сцены из файла (пользовательский формат, либо: JSON (nlohmann/json), XML (zeux/pugixml)). Объекты могут иметь разные координаты, текстуры, содержать разные трехмерные модели (плоскость, куб, пирамида). |
| 3 | Реализовать возможность перемещать источник света и изменять цвет источника используя программу-клиент (C++). Визуализировать позицию источника света с помощью куба. |
| 4\* | Визуализировать куб с разными текстурами на всех гранях. |