**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 804 «Теория вероятностей и компьютерное моделирование»

Дисциплина: «Компьютерная графика»

**Лабораторная работа № 6**

Тема: Создание шейдерных анимационных эффектов в OpenGL 2.1

Студент: Шевчук П.В.

Группа: 80-304

Преподаватель: Чернышов Л.Н.

Дата: 11.12.18

Оценка:

Москва, 2018

**1. Постановка задачи**

Для поверхности, созданной в л.р. №5, обеспечить выполнение шейдерного эффекта.

Вариант эффекта: Анимация. Вращение относительно направления на источник света.

**2. Решения задачи**

ЯП: C++

ОС: Linux Mint 18.2

Среда разработки: Qt Creator 4.7.2

Библиотеки: QOpenGLFunctions, QOpenGLShaderProgram

Выбранный язык удобен для решения задачи, так как он обладает удобными библиотеками для построения фигур и вывода их на экран.

Основные функции:

* Функции initializeGL, resizeGL и paintGL перекрывают методы

родительского класса QGLWidget. Обработчики событий от мыши перекрывают обработчики, унаследованные от QWidget.

* В конструкторе вызывается QGLWidget::setFormat, чтобы

задать контекст устройства отображения OpenGL, и инициализируются приватные переменные-члены класса.

* Функция initializeGL вызывается один раз, перед вызовом

paintGL. Здесь выполняется настройка контекста отображения.

* Функция resizeGL вызывается один раз, перед paintGL, но после

того, как будет вызвана функция initializeGL. Здесь настраивается область просмотра (viewport), проекция и прочие настройки, которые зависят от размера виджета.

* Функция paintGL вызывается всякий раз, когда возникает

необходимость перерисовать содержимое виджета

* Функции mousePressEvent и mouseMoveEvent позволяют

пользователю вращать фигуру и перемещать его по поверхности экрана. Левой кнопкой мыши выполняется вращение по осям x и y, правой по осям x и z.

Ход работы:

Реализована наивная модель освещения, когда степень освещенности точки обратно пропорциональна расстоянию до источника света. Также используется модель Фонга. Эта модель разбивает свет на 3 составляющих: диффузный свет, свет окружения, зеркальный свет. Диффузный свет - результат прямого попадания лучей света на объект, свет окружения - свет рассеивания, наполняющий пространство, зеракальный свет отвечает за блики и визуально отличается от диффузного тем, что он ближе по цветовой гамме к источнику света. Для того, чтобы правильно рассчитать отражающие свойства объекта, требуется вычислить нормали. Для простоты наблюдатель является источником света.

Все вычисления освещения происходят во фрагментном шейдере. Вершинный шейдер преобразовывает вершины фигуры, а фрагментный уже обрабатывает стороны (по пикселям), образуемые вершинами и рисует цвета.

Чтобы выполнить вращение относительно направления света понадобилось рассчитать нормаль к вектору направления света и прибавить полученное вращение к текущему вращению.

**3. Руководство по использованию программы**

В окне вывода пользователь может менять степень аппроксимации, которая задается с помощью поля «Approximation accuracy». Поставим значение 10 (рис.1).

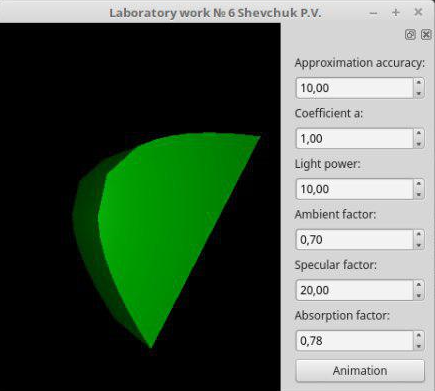


Рис 1. Степень аппроксимации равна 10

Пользователь может изменять значение первого коэффициента «a» уравнения параболы, а также выключать анимацию(рис.2).

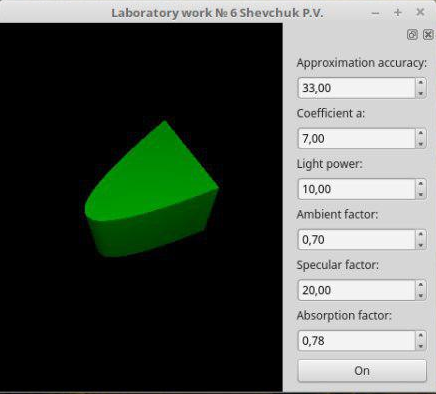


Рис 2. Параметр а = 7

Изменять мощность падающего света можно с помощью параметра «Light power» (рис. 3). Фоновое освещение(ambient factor) можно считать как более однотонное, а рассеянное(absorption factor) позволяет увидеть разную степень освещения той или грани. Степень рассеивания регулируется с помощью «Specular factor».

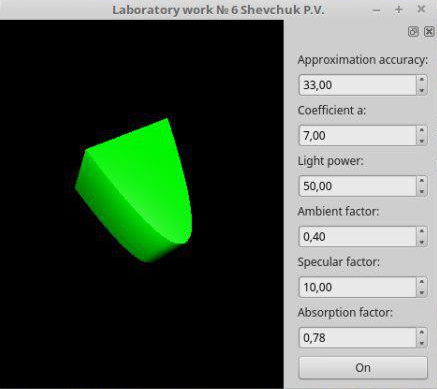


Рис 3 . Увеличиваем мощность падающего света

**4. Листинг программы**

#include "view.h"

#include "panel.h"

#include "simpleobject3d.h"

#include <QOpenGLContext>

// сигнал

View::View(QWidget \*parent) : QOpenGLWidget(parent), pan(nullptr) {

}

const Panel \*View::controlPanel() const { return pan; }

void View::setControlPanel(const Panel \*p) {

pan = p;

update();

}

// инициализация графика

void View::initializeGL() {

glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_CULL\_FACE);

// glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE);

glLineWidth(10.0f);

scale = -4.0f;

timer = startTimer(30);

animation = true;

lightPosition = QVector4D(0.0, 0.0, 0.0, 1.0);

initShaders();

initFigure(1.0f);

}

// настраивается область просмотра

void View::resizeGL(int w, int h) {

float aspect = w / static\_cast<float>(h);

projectionMatrix.setToIdentity();

projectionMatrix.perspective(45, aspect, 0.1f, 20.0f);

}

// перерисовка содержимого

void View::paintGL() {

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

QMatrix4x4 viewMatrix;

viewMatrix.setToIdentity();

viewMatrix.translate(0.0f, 0.0f, scale);

viewMatrix.rotate(rotation);

m\_program.bind();

m\_program.setUniformValue("u\_projectionMatrix", projectionMatrix);

m\_program.setUniformValue("u\_viewMatrix", viewMatrix);

m\_program.setUniformValue("u\_lightPosition", lightPosition);

m\_program.setUniformValue("u\_lightPower", static\_cast<float>(pan->getLightpower()));

m\_program.setUniformValue("u\_ambientFactor", static\_cast<float>(pan->getAmbientfactor()));

m\_program.setUniformValue("u\_specularFactor", static\_cast<float>(pan->getSpecularfactor()));

m\_program.setUniformValue("u\_absorptionFactor", static\_cast<float>(pan->getAbsorptionfactor()));

for (auto &object : objects) {

object->draw(&m\_program, context()->functions());

}

}

// инициализация шейдеров

void View::initShaders() {

if (!m\_program.addShaderFromSourceFile(QOpenGLShader::Vertex,

":/vshader.vsh")) {

close();

}

if (!m\_program.addShaderFromSourceFile(QOpenGLShader::Fragment,

":/fshader.fsh")) {

close();

}

if (!m\_program.link()) {

close();

}

}

// соблюдение анимации

void View::toggleTimer()

{

if (animation)

killTimer(timer);

else

timer = startTimer(30);

animation = not animation;

}

// инициализация фигуры

void View::initFigure(float width)

{

if (!objects.empty()) {

objects[0]->~SimpleObject3D();

objects.pop\_back();

}

float width\_div\_2 = width / 2.0f;

QVector<VertexData> vertexes;

auto cnt = static\_cast<float>(pan->getAccuracy());

float step = width / cnt;

auto a = static\_cast<float>(pan->getA());

vertexes.append(VertexData(QVector3D(0, -width\_div\_2, width\_div\_2), QVector2D(0.0, 0.0), QVector3D(0.0, -1.0, 0.0)));

for (int i = 1; i <= cnt; ++i) {

float z = i \* step;

vertexes.append(VertexData(QVector3D(0, -width\_div\_2, width\_div\_2 - z), QVector2D(1.0, 0.0), QVector3D(0.0, -1.0, 0.0)));

vertexes.append(VertexData(QVector3D(sqrt(z / a), -width\_div\_2, width\_div\_2 - z), QVector2D(0.0, 1.0), QVector3D(0.0, -1.0, 0.0)));

vertexes.append(VertexData(QVector3D(-sqrt(z / a), -width\_div\_2, width\_div\_2 - z), QVector2D(1.0, 1.0), QVector3D(0.0, -1.0, 0.0)));

}

QVector<GLuint> indexes;

indexes.append(1);

indexes.append(2);

indexes.append(0);

indexes.append(1);

indexes.append(0);

indexes.append(3);

for (GLuint i = 1; i < 3 \* (cnt - 1) + 1; i += 3) {

indexes.append(i + 3);

indexes.append(i + 4);

indexes.append(i);

indexes.append(i + 1);

indexes.append(i);

indexes.append(i + 4);

indexes.append(i);

indexes.append(i + 2);

indexes.append(i + 5);

indexes.append(i);

indexes.append(i + 5);

indexes.append(i + 3);

}

GLuint offset = vertexes.size();

vertexes.append(VertexData(QVector3D(0, width\_div\_2, width\_div\_2), QVector2D(0.0, 0.0), QVector3D(0.0, 1.0, 0.0)));

for (GLuint i = 1; i <= cnt; ++i) {

float z = i \* step;

vertexes.append(VertexData(QVector3D(0, width\_div\_2, width\_div\_2 - z), QVector2D(0.0, 0.0), QVector3D(0.0, 1.0, 0.0)));

vertexes.append(VertexData(QVector3D(sqrt(z / a), width\_div\_2, width\_div\_2 - z), QVector2D(0.0, 0.0), QVector3D(0.0, 1.0, 0.0)));

vertexes.append(VertexData(QVector3D(-sqrt(z / a), width\_div\_2, width\_div\_2 - z), QVector2D(0.0, 0.0), QVector3D(0.0, 1.0, 0.0)));

}

indexes.append(offset + 1);

indexes.append(offset + 0);

indexes.append(offset + 2);

indexes.append(offset + 1);

indexes.append(offset + 3);

indexes.append(offset + 0);

for (GLuint i = 1; i < 3 \* (cnt - 1) + 1; i += 3) {

indexes.append(offset + i + 3);

indexes.append(offset + i);

indexes.append(offset + i + 4);

indexes.append(offset + i + 1);

indexes.append(offset + i + 4);

indexes.append(offset + i);

indexes.append(offset + i);

indexes.append(offset + i + 5);

indexes.append(offset + i + 2);

indexes.append(offset + i);

indexes.append(offset + i + 3);

indexes.append(offset + i + 5);

}

vertexes.append(VertexData(QVector3D(0, -width\_div\_2, width\_div\_2), QVector2D(0.0, 0.0), QVector3D(0.0, 0.0, 1.0)));

for (GLuint i = 1; i <= cnt; ++i) {

float z = i \* step;

vertexes.append(VertexData(QVector3D(0, -width\_div\_2, width\_div\_2 - z), QVector2D(0.0, 0.0), QVector3D(0.0, -1.0, 0.0)));

float dx1 = sqrt(z / a) - vertexes[(i == 1 ? vertexes.size() - 2 : vertexes.size() - 3)].position.x();

float dz1 = width\_div\_2 - z - vertexes[(i == 1 ? vertexes.size() - 2 : vertexes.size() - 3)].position.z();

vertexes.append(VertexData(QVector3D(sqrt(z / a), -width\_div\_2, width\_div\_2 - z), QVector2D(0.0, 0.0), QVector3D(-dz1, 0.0, dx1)));

float dx2 = -sqrt(z / a) - vertexes[vertexes.size() - 3].position.x();

float dz2 = width\_div\_2 - z - vertexes[vertexes.size() - 3].position.z();

vertexes.append(VertexData(QVector3D(-sqrt(z / a), -width\_div\_2, width\_div\_2 - z), QVector2D(0.0, 0.0), QVector3D(dz2, 0.0, -dx2)));

}

vertexes.append(VertexData(QVector3D(0, width\_div\_2, width\_div\_2), QVector2D(0.0, 0.0), QVector3D(0.0, 0.0, 1.0)));

for (GLuint i = 1; i <= cnt; ++i) {

float z = i \* step;

vertexes.append(VertexData(QVector3D(0, width\_div\_2, width\_div\_2 - z), QVector2D(0.0, 0.0), QVector3D(0.0, 1.0, 0.0)));

float dx1 = sqrt(z / a) - vertexes[(i == 1 ? vertexes.size() - 2 : vertexes.size() - 3)].position.x();

float dz1 = width\_div\_2 - z - vertexes[(i == 1 ? vertexes.size() - 2 : vertexes.size() - 3)].position.z();

vertexes.append(VertexData(QVector3D(sqrt(z / a), width\_div\_2, width\_div\_2 - z), QVector2D(0.0, 0.0), QVector3D(-dz1, 0.0, dx1)));

float dx2 = -sqrt(z / a) - vertexes[vertexes.size() - 3].position.x();

float dz2 = width\_div\_2 - z - vertexes[vertexes.size() - 3].position.z();

vertexes.append(VertexData(QVector3D(-sqrt(z / a), width\_div\_2, width\_div\_2 - z), QVector2D(0.0, 0.0), QVector3D(dz2, 0.0, -dx2)));

}

for (GLuint i = 3; i <= cnt \* 3; i += 3) {

indexes.append(i + offset \* 3);

indexes.append(i + offset \* 2);

indexes.append(i - 3 + offset \* 2);

indexes.append(i + offset \* 3);

indexes.append(i - 3 + offset \* 2);

indexes.append(i - 3 + offset \* 3);

}

indexes.append(offset \* 3);

indexes.append(offset \* 2);

indexes.append(offset \* 3 + 2);

indexes.append(offset \* 2);

indexes.append(offset \* 2 + 2);

indexes.append(offset \* 3 + 2);

for (GLuint i = 5; i <= cnt \* 3; i += 3) {

indexes.append(i + offset \* 3);

indexes.append(i - 3 + offset \* 2);

indexes.append(i + offset \* 2);

indexes.append(i + offset \* 3);

indexes.append(i + offset \* 3 - 3);

indexes.append(i - 3 + offset \* 2);

}

vertexes.append(VertexData(QVector3D(sqrt(width / a), width\_div\_2, -width\_div\_2), QVector2D(0.0, 0.0), QVector3D(0.0, 0.0, -1.0)));

vertexes.append(VertexData(QVector3D(sqrt(width / a), -width\_div\_2, -width\_div\_2), QVector2D(0.0, 0.0), QVector3D(0.0, 0.0, -1.0)));

vertexes.append(VertexData(QVector3D(-sqrt(width / a), width\_div\_2, -width\_div\_2), QVector2D(0.0, 0.0), QVector3D(0.0, 0.0, -1.0)));

vertexes.append(VertexData(QVector3D(-sqrt(width / a), -width\_div\_2, -width\_div\_2), QVector2D(0.0, 0.0), QVector3D(0.0, 0.0, -1.0)));

indexes.append(vertexes.size() - 4);

indexes.append(vertexes.size() - 3);

indexes.append(vertexes.size() - 2);

indexes.append(vertexes.size() - 3);

indexes.append(vertexes.size() - 1);

indexes.append(vertexes.size() - 2);

objects.append(new SimpleObject3D(vertexes, indexes, QImage(":/cube.jpg")));

}

void View::timerEvent(QTimerEvent\*)

{

float angle = 2.0f;

QVector3D axis = QVector3D(0.0, 0.0, 1.0);

rotation = QQuaternion::fromAxisAndAngle(axis, angle) \* rotation;

update();

}

// мыш

void View::mousePressEvent(QMouseEvent \*event) {

if (event->buttons() == Qt::LeftButton) {

mousePosition = QVector2D(event->localPos());

}

event->accept();

}

void View::mouseMoveEvent(QMouseEvent \*event) {

if (event->buttons() != Qt::LeftButton) {

return;

}

QVector2D diff = QVector2D(event->localPos()) - mousePosition;

mousePosition = QVector2D(event->localPos());

float angle = diff.length() / 2.0f;

QVector3D axis = QVector3D(diff.y(), diff.x(), 0);

rotation = QQuaternion::fromAxisAndAngle(axis, angle) \* rotation;

update();

}

void View::wheelEvent(QWheelEvent \*event) {

scale += (event->angleDelta().y() / 200.0f);

scale = std::min(scale, -1.0f);

scale = std::max(scale, -20.0f);

update();

}

**5. Вывод**

Благодаря проделанной работе удалось освоить 3D-графику на С++ в среде разработки QT, методы закраски, отрисовки плоскостей и способы анимаций с помощью технологий OpenGL.

**Список литературы**

1. Графика OpenGL[Электронный ресурс].

URL: <https://www.opennet.ru/docs/RUS/qt3_prog/x4697.html>

2. learn opengl. Shaders [Электронный ресурс].

URL: <https://habr.com/post/313380/>