Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное

образовательное учреждение высшего образования

«Уфимский университет науки и технологий»

Факультет информатики и робототехники

Кафедра Информатики

Отчёт по лабораторной работе №3

По дисциплине “Инженерная и компьютерная графика”

Выполнил:

Студент группы ПРО-231 К.И. Арменшин

Проверил:

доцент кафедры ВМиК Котельников В.А.

Уфа – 2023

**Цель:**

Изучение управления камерой и освещения в OpenGL

**Ход работы:**

**Уроки 14-15.**

Реализовано движение камеры с помощью четырёх клавиш и мыши (Рисунок 1. Управление камерой).

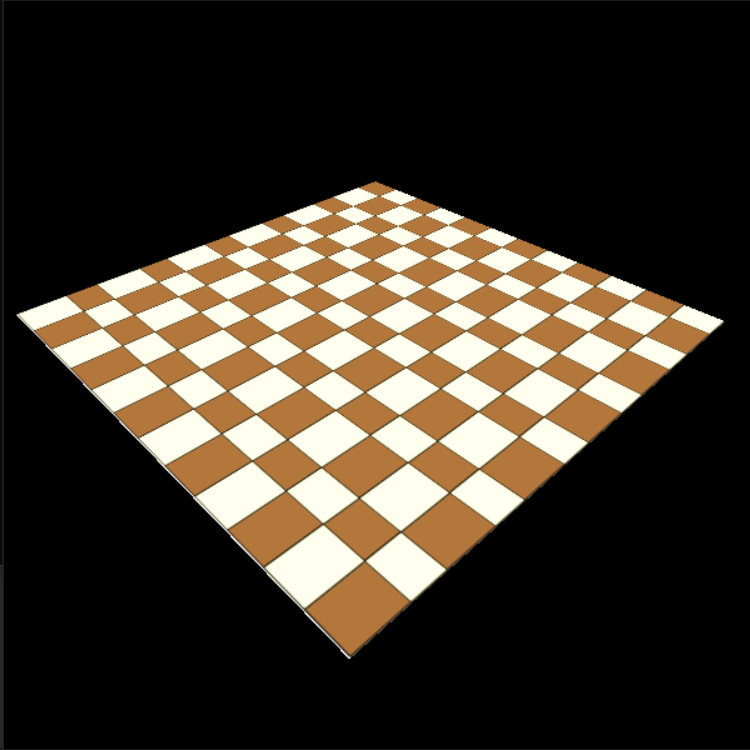
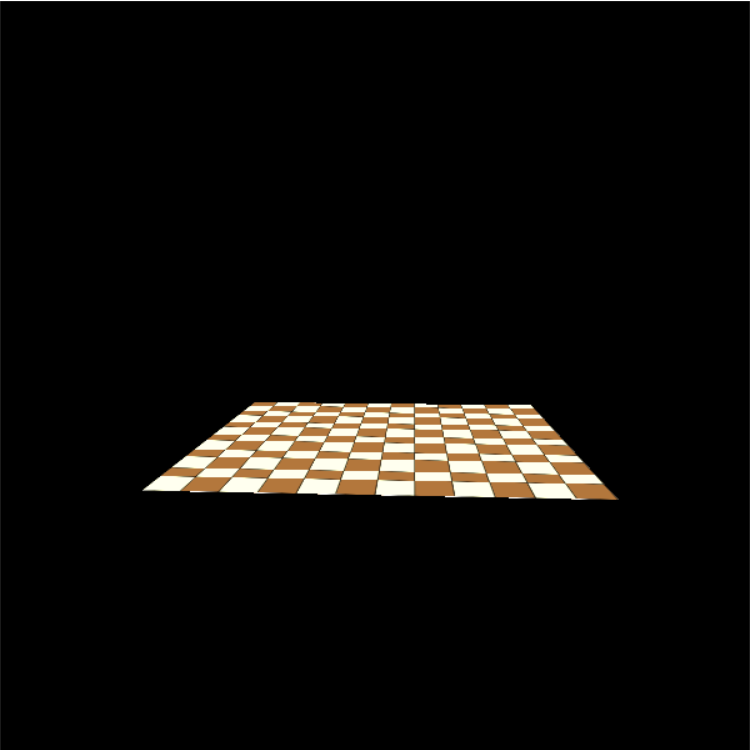
 

Рисунок . Управление камерой

**Урок 16.**

Реализовано наложение текстур на объекты с помощью библиотеки ImageMagick (Рисунок 2. Наложение текстур).

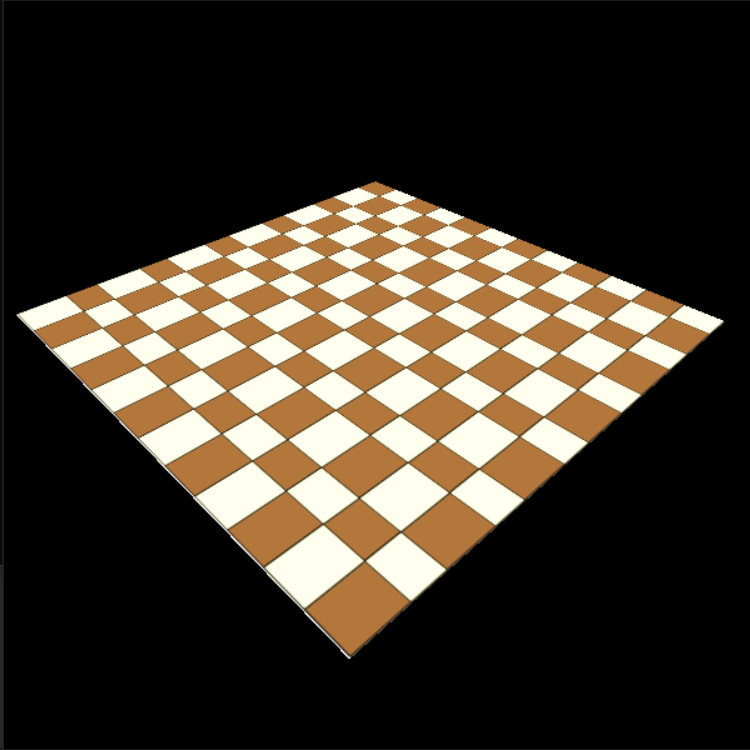


Рисунок . Наложение текстур

**Уроки 17-18.**

Реализовано фоновое и рассеянное освещение (Рисунок 3. Освещение).

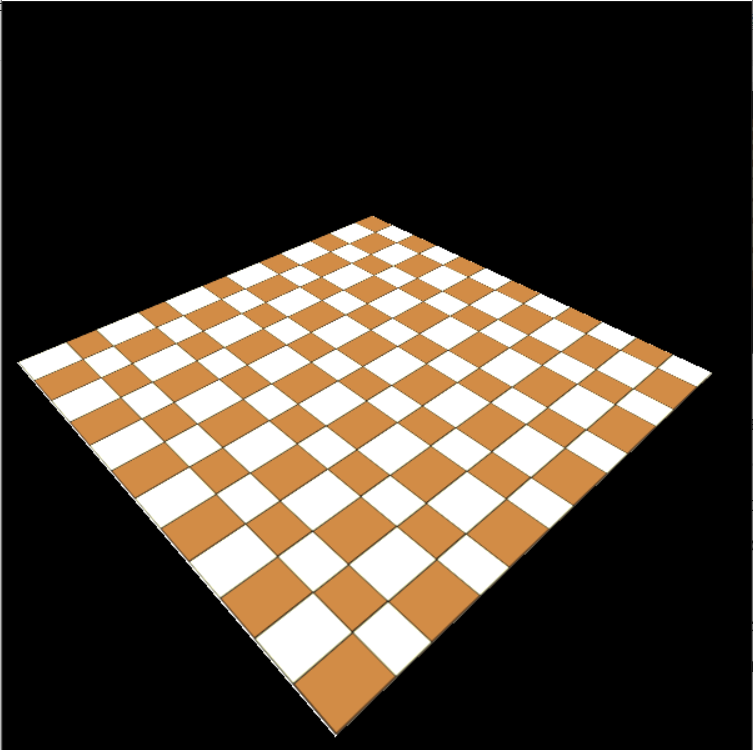
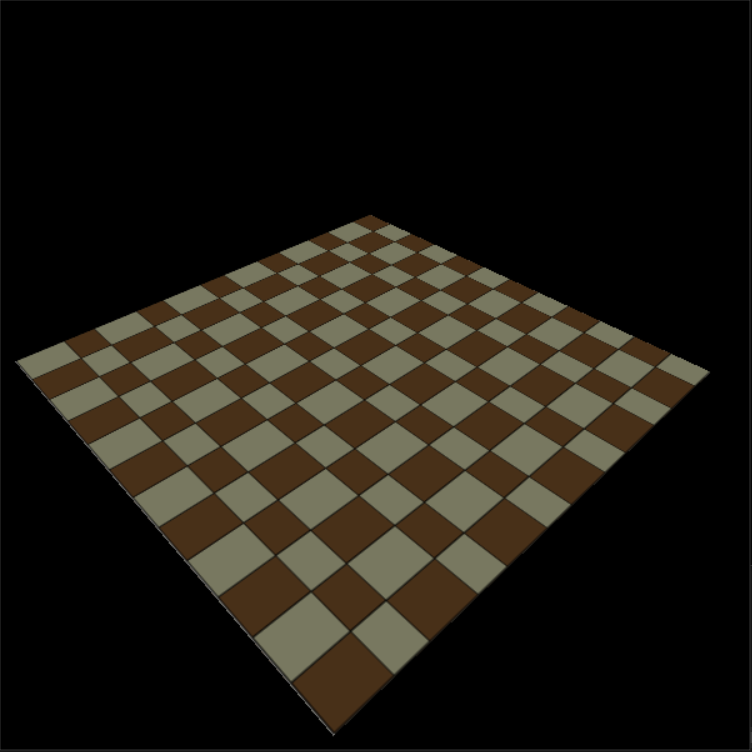


Рисунок . Освещение

**Урок 19.**

Реализовано отражение света от объектов (Рисунок 4. Отражение света.).

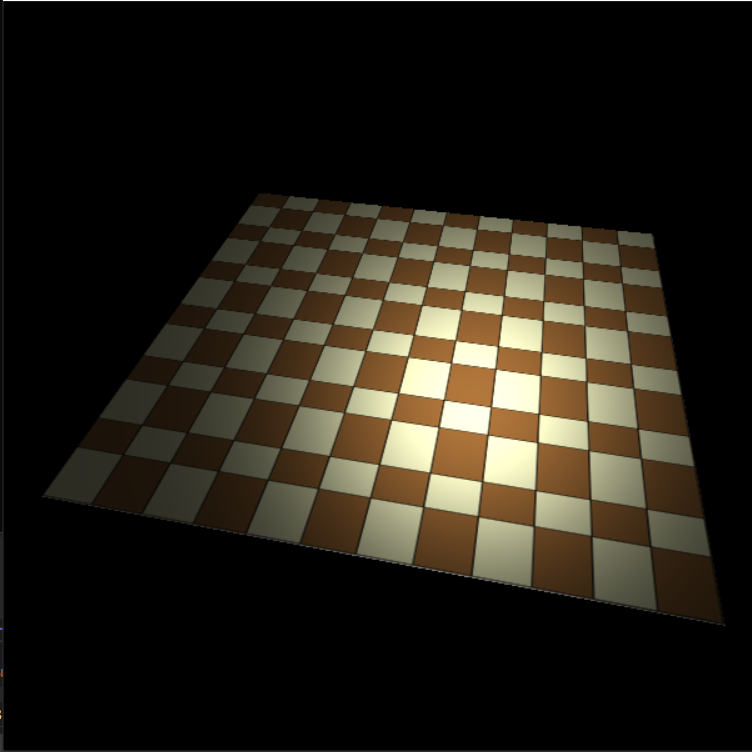


Рисунок . Отражение света.

**Урок 20.**

Добавлены точечные источники света (Рисунок 5. Точечные источники света).

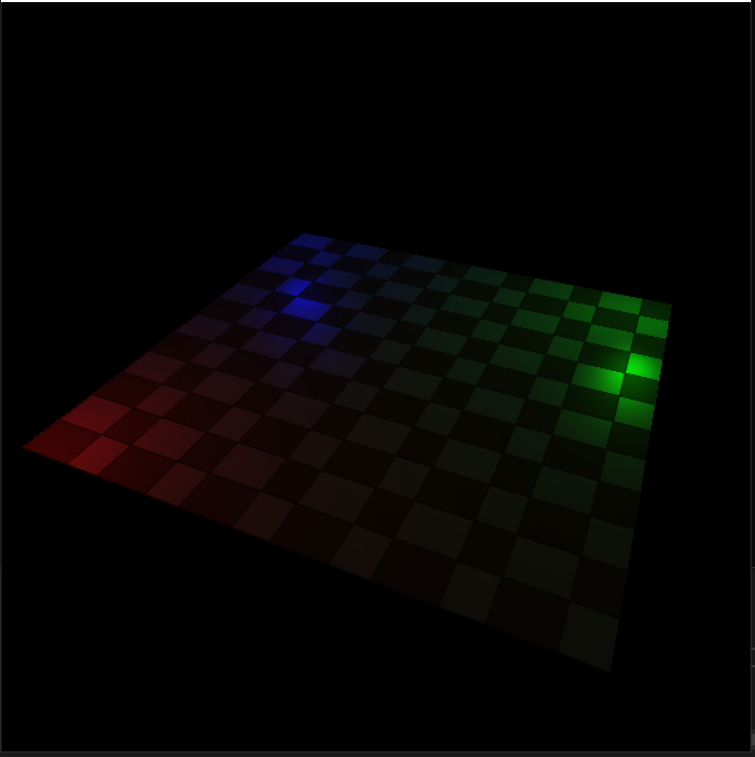


Рисунок . Точечные источники света

**Урок 21.**

Добавлены прожекторы (Рисунок 6. Прожекторы).

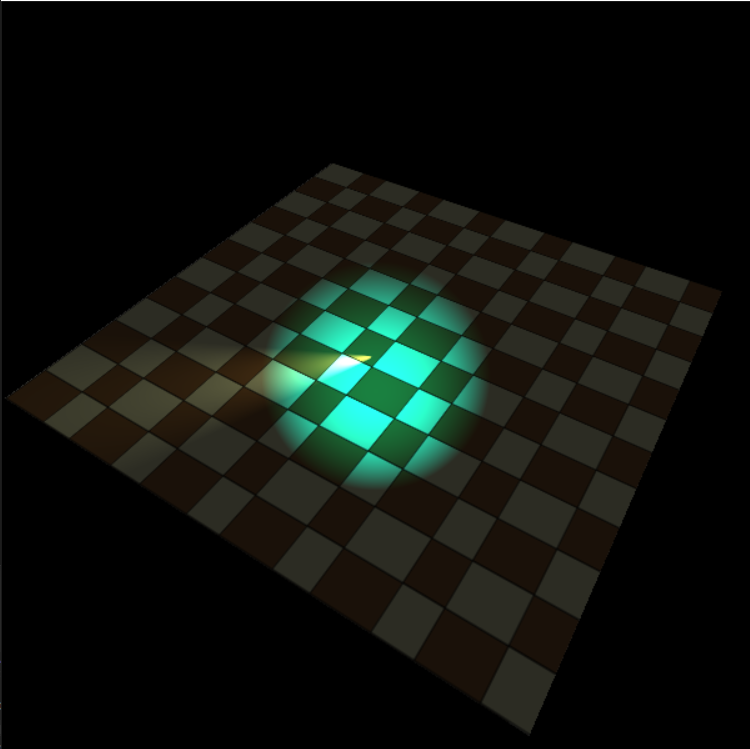


Рисунок . Прожекторы

**Вывод**

В результате выполнения данной лабораторной работы были выполнены уроки 14-21, в ходе которых была разработана программа, демонстрирующая движение камеры, наложение текстур, освещение, точечные источники света и прожекторы.

Полный код находится в Github:

https://github.com/karimarm/Labs\_ECG.git

**Приложение 1**

L3\_ECG.cpp

#include <math.h>

#include <GL/glew.h>

#include <GL/freeglut.h>

#include "pipeline.h"

#include "camera.h"

#include "texture.h"

#include "lighting\_technique.h"

#include "glut\_backend.h"

#include "util.h"

#define WINDOW\_WIDTH 600

#define WINDOW\_HEIGHT 600

// Координаты вершин и текстуры

struct Vertex

{

Vector3f m\_pos;

Vector2f m\_tex;

Vector3f m\_normal;

Vertex() {}

Vertex(Vector3f pos, Vector2f tex)

{

m\_pos = pos;

m\_tex = tex;

m\_normal = Vector3f(0.0f, 0.0f, 0.0f);

}

};

class Main : public ICallbacks

{

public:

Main()

{

m\_pGameCamera = NULL;

m\_pTexture = NULL;

m\_pEffect = NULL;

m\_scale = 0.0f;

// Параметры рассеяноого освещения

m\_directionalLight.Color = Vector3f(1.0f, 1.0f, 1.0f);

m\_directionalLight.AmbientIntensity = 0.1f;

m\_directionalLight.DiffuseIntensity = 0.1f;

m\_directionalLight.Direction = Vector3f(-1.0f, -1.0f, 0.0f);

}

~Main()

{

delete m\_pEffect;

delete m\_pGameCamera;

delete m\_pTexture;

}

bool Init()

{

// Позиция камеры

Vector3f Pos(-10.0f, 10.0f, -10.0f);

Vector3f Target(1.0f, 0.0f, 1.0f);

Vector3f Up(0.0, 1.0f, 0.0f);

m\_pGameCamera = new Camera(WINDOW\_WIDTH, WINDOW\_HEIGHT, Pos, Target, Up);

// Массив индексов вершин

unsigned int Indices[] = { 0, 2, 1,

0, 3, 2 };

CreateIndexBuffer(Indices, sizeof(Indices));

CreateVertexBuffer(Indices, ARRAY\_SIZE\_IN\_ELEMENTS(Indices));

m\_pEffect = new LightingTechnique();

if (!m\_pEffect->Init())

{

printf("Error initializing the lighting technique\n");

return false;

}

m\_pEffect->Enable();

m\_pEffect->SetTextureUnit(0);

m\_pTexture = new Texture(GL\_TEXTURE\_2D, "123.jpg");

if (!m\_pTexture->Load()) {

return false;

}

return true;

}

void Run()

{

GLUTBackendRun(this);

}

virtual void RenderSceneCB()

{

m\_pGameCamera->OnRender();

glClear(GL\_COLOR\_BUFFER\_BIT); // Очистка буфера

m\_scale += 0.01f;

// Прожекторы

SpotLight sl[2];

sl[0].DiffuseIntensity = 15.0f;

sl[0].Color = Vector3f(1.0f, 1.0f, 0.7f);

sl[0].Position = Vector3f(-0.0f, -1.9f, -0.0f);

sl[0].Direction = Vector3f(sinf(m\_scale), 0.0f, cosf(m\_scale));

sl[0].Attenuation.Linear = 0.1f;

sl[0].Cutoff = 20.0f;

sl[1].DiffuseIntensity = 5.0f;

sl[1].Color = Vector3f(0.0f, 1.0f, 1.0f);

sl[1].Position = m\_pGameCamera->GetPos();

sl[1].Direction = m\_pGameCamera->GetTarget();

sl[1].Attenuation.Linear = 0.1f;

sl[1].Cutoff = 10.0f;

m\_pEffect->SetSpotLights(2, sl);

// Точечное освещение

PointLight pl[3];

pl[0].DiffuseIntensity = 0.5f;

pl[0].Color = Vector3f(1.0f, 0.0f, 0.0f);

pl[0].Position = Vector3f(-9.0f, 1.0f, -9.0f);

pl[0].Attenuation.Linear = 0.1f;

pl[1].DiffuseIntensity = 0.5f;

pl[1].Color = Vector3f(0.0f, 1.0f, 0.0f);

pl[1].Position = Vector3f(9.0f, 1.0f, 9.0f);

pl[1].Attenuation.Linear = 0.1f;

pl[2].DiffuseIntensity = 0.5f;

pl[2].Color = Vector3f(0.0f, 0.0f, 1.0f);

pl[2].Position = Vector3f(-9.0f, 1.0f, 9.0f);

pl[2].Attenuation.Linear = 0.1f;

//m\_pEffect->SetPointLights(3, pl);

// Создаёт объект конвейера и настраивает его

Pipeline p;

p.Rotate(0.0f, 0.0f, 0.0f);

p.WorldPos(0.0f, 0.0f, 1.0f);

p.SetCamera(m\_pGameCamera->GetPos(), m\_pGameCamera->GetTarget(), m\_pGameCamera->GetUp());

p.SetPerspectiveProj(60.0f, WINDOW\_WIDTH, WINDOW\_HEIGHT, 0.1f, 100.0f);

m\_pEffect->SetWVP(p.GetWVPTrans());

const Matrix4f& WorldTransformation = p.GetWorldTrans();

m\_pEffect->SetWorldMatrix(WorldTransformation);

m\_pEffect->SetDirectionalLight(m\_directionalLight);

m\_pEffect->SetEyeWorldPos(m\_pGameCamera->GetPos());

m\_pEffect->SetMatSpecularIntensity(1.0f);

m\_pEffect->SetMatSpecularPower(32);

glEnableVertexAttribArray(0);// Атрибуты вершин для координат вершин

glEnableVertexAttribArray(1); // Атрибуты вершин для координат текстур

glEnableVertexAttribArray(2);

glBindBuffer(GL\_ARRAY\_BUFFER, m\_VBO);// Привязка буфера

glVertexAttribPointer(0, 3, GL\_FLOAT, GL\_FALSE, sizeof(Vertex), 0);// Восприятие данных внутри буфера

glVertexAttribPointer(1, 2, GL\_FLOAT, GL\_FALSE, sizeof(Vertex), (const GLvoid\*)12); // Восприятие данных внутри буфера

glVertexAttribPointer(2, 3, GL\_FLOAT, GL\_FALSE, sizeof(Vertex), (const GLvoid\*)20); // Восприятие данных внутри буфера

glBindBuffer(GL\_ELEMENT\_ARRAY\_BUFFER, m\_IBO);

m\_pTexture->Bind(GL\_TEXTURE0); // Привязываем текстуры к модулю

glDrawElements(GL\_TRIANGLES, 6, GL\_UNSIGNED\_INT, 0); // Отрисовка

glDisableVertexAttribArray(0); // Отключить атрибуты

glDisableVertexAttribArray(1); // Отключить атрибуты

glDisableVertexAttribArray(2); // Отключить атрибуты

glutSwapBuffers(); // Меняет буферы местами

}

virtual void IdleCB()

{

RenderSceneCB();

}

virtual void SpecialKeyboardCB(int Key, int x, int y)

{

m\_pGameCamera->OnKeyboard(Key);

}

virtual void KeyboardCB(unsigned char Key, int x, int y)

{

switch (Key) {

case 'q':

glutLeaveMainLoop();

break;

case 'a':

m\_directionalLight.AmbientIntensity += 0.05f;

break;

case 's':

m\_directionalLight.AmbientIntensity -= 0.05f;

break;

case 'z':

m\_directionalLight.DiffuseIntensity += 0.05f;

break;

case 'x':

m\_directionalLight.DiffuseIntensity -= 0.05f;

break;

}

}

virtual void PassiveMouseCB(int x, int y)

{

m\_pGameCamera->OnMouse(x, y);

}

private:

void CalcNormals(const unsigned int\* pIndices, unsigned int IndexCount,

Vertex\* pVertices, unsigned int VertexCount) {

for (unsigned int i = 0; i < IndexCount; i += 3) {

unsigned int Index0 = pIndices[i];

unsigned int Index1 = pIndices[i + 1];

unsigned int Index2 = pIndices[i + 2];

Vector3f v1 = pVertices[Index1].m\_pos - pVertices[Index0].m\_pos;

Vector3f v2 = pVertices[Index2].m\_pos - pVertices[Index0].m\_pos;

Vector3f Normal = v1.Cross(v2);

Normal.Normalize();

pVertices[Index0].m\_normal += Normal;

pVertices[Index1].m\_normal += Normal;

pVertices[Index2].m\_normal += Normal;

}

for (unsigned int i = 0; i < VertexCount; i++) {

pVertices[i].m\_normal.Normalize();

}

}

void CreateVertexBuffer(const unsigned int\* pIndices, unsigned int IndexCount)

{

Vertex Vertices[4] = { Vertex(Vector3f(-10.0f, -2.0f, -10.0f), Vector2f(0.0f, 0.0f)),

Vertex(Vector3f(10.0f, -2.0f, -10.0f), Vector2f(1.0f, 0.0f)),

Vertex(Vector3f(10.0f, -2.0f, 10.0f), Vector2f(1.0f, 1.0f)),

Vertex(Vector3f(-10.0f, -2.0f, 10.0f), Vector2f(0.0f, 1.0f)) };

unsigned int VertexCount = ARRAY\_SIZE\_IN\_ELEMENTS(Vertices);

CalcNormals(pIndices, IndexCount, Vertices, VertexCount);

glGenBuffers(1, &m\_VBO);

glBindBuffer(GL\_ARRAY\_BUFFER, m\_VBO);

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

}

void CreateIndexBuffer(const unsigned int\* pIndices, unsigned int SizeInBytes)

{

glGenBuffers(1, &m\_IBO);

glBindBuffer(GL\_ELEMENT\_ARRAY\_BUFFER, m\_IBO);

glBufferData(GL\_ELEMENT\_ARRAY\_BUFFER, SizeInBytes, pIndices, GL\_STATIC\_DRAW);

}

GLuint m\_VBO;

GLuint m\_IBO;

LightingTechnique\* m\_pEffect;

Texture\* m\_pTexture;

Camera\* m\_pGameCamera;

float m\_scale;

DirectionalLight m\_directionalLight;

};

int main(int argc, char\*\* argv)

{

Magick::InitializeMagick(\*argv);

GLUTBackendInit(argc, argv);

if (!GLUTBackendCreateWindow(WINDOW\_WIDTH, WINDOW\_HEIGHT, 32, false, "OpenGL tutors")) {

return 1;

}

Main\* pApp = new Main();

if (!pApp->Init()) {

return 1;

}

pApp->Run();

delete pApp;

return 0;

}