ФГБОУ ВО

Уфимский университет науки и технологии

Кафедра ВМиК

Отчет

по лабораторной работе №5

по дисциплине «Инженерная и компьютерная графика»

Выполнил: ст.гр. ПРО-231

Семенов Г.Д.

Принял: доцент

Котельников В.А.

Уфа 2023 г

# Цель работы

Ознакомиться с OpenGL, установить необходимые библиотеки, выполнить задания с сайта <https://triplepointfive.github.io/ogltutor/>

# Задание

Выполнить 27, 28, 32, 33 уроки по OpenGL.

# Ход работы

Урок 27:

Используем метод billboarding, то есть делаем так, чтобы все объекты всегда смотрели в камеру.

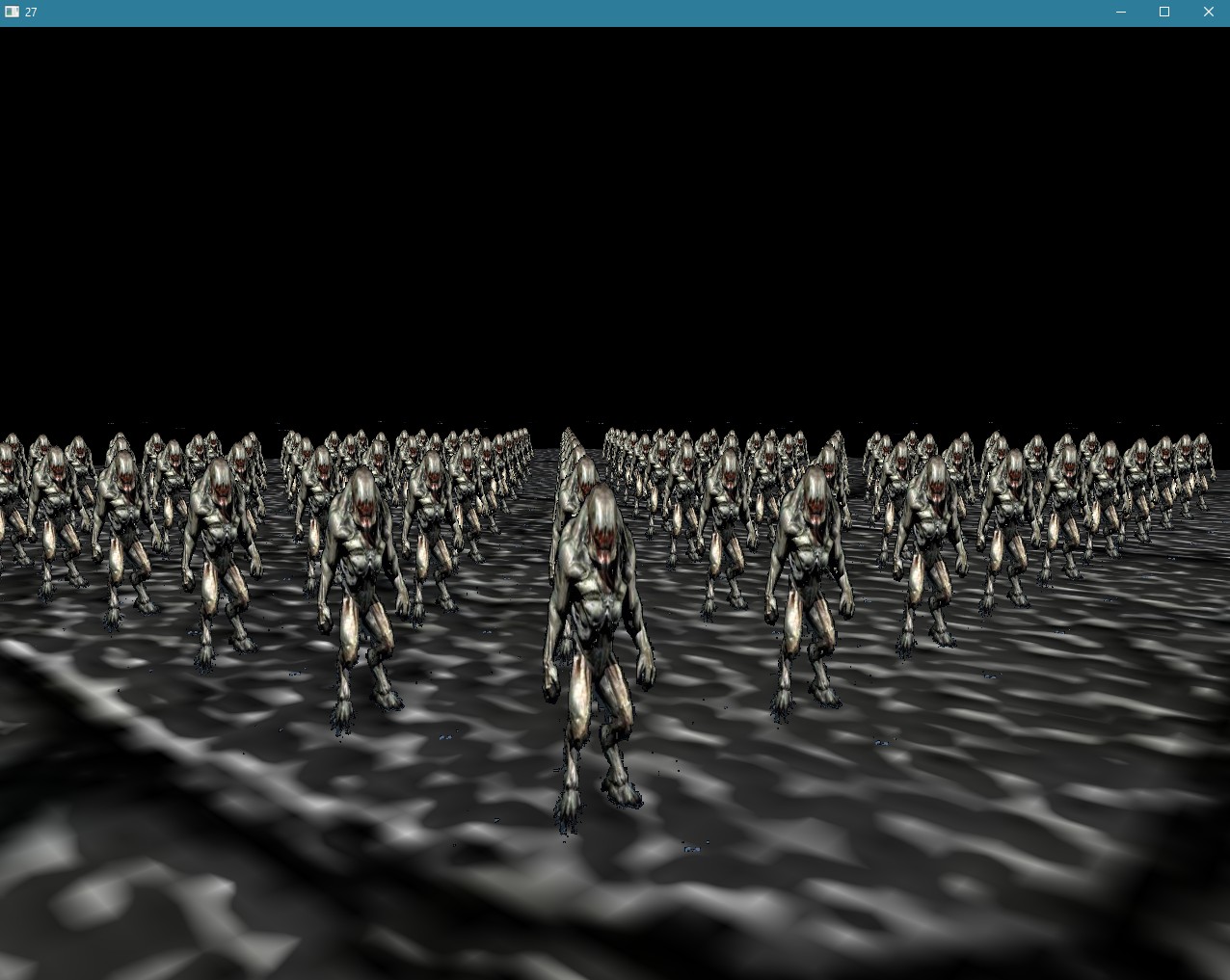


Рисунок 1 Метод billboarding

Урок 32:

Создаем Vertex Array Objects.



Рисунок 2 Vertex Array Objects

Урок 33:

Используем дублирующий рендер.

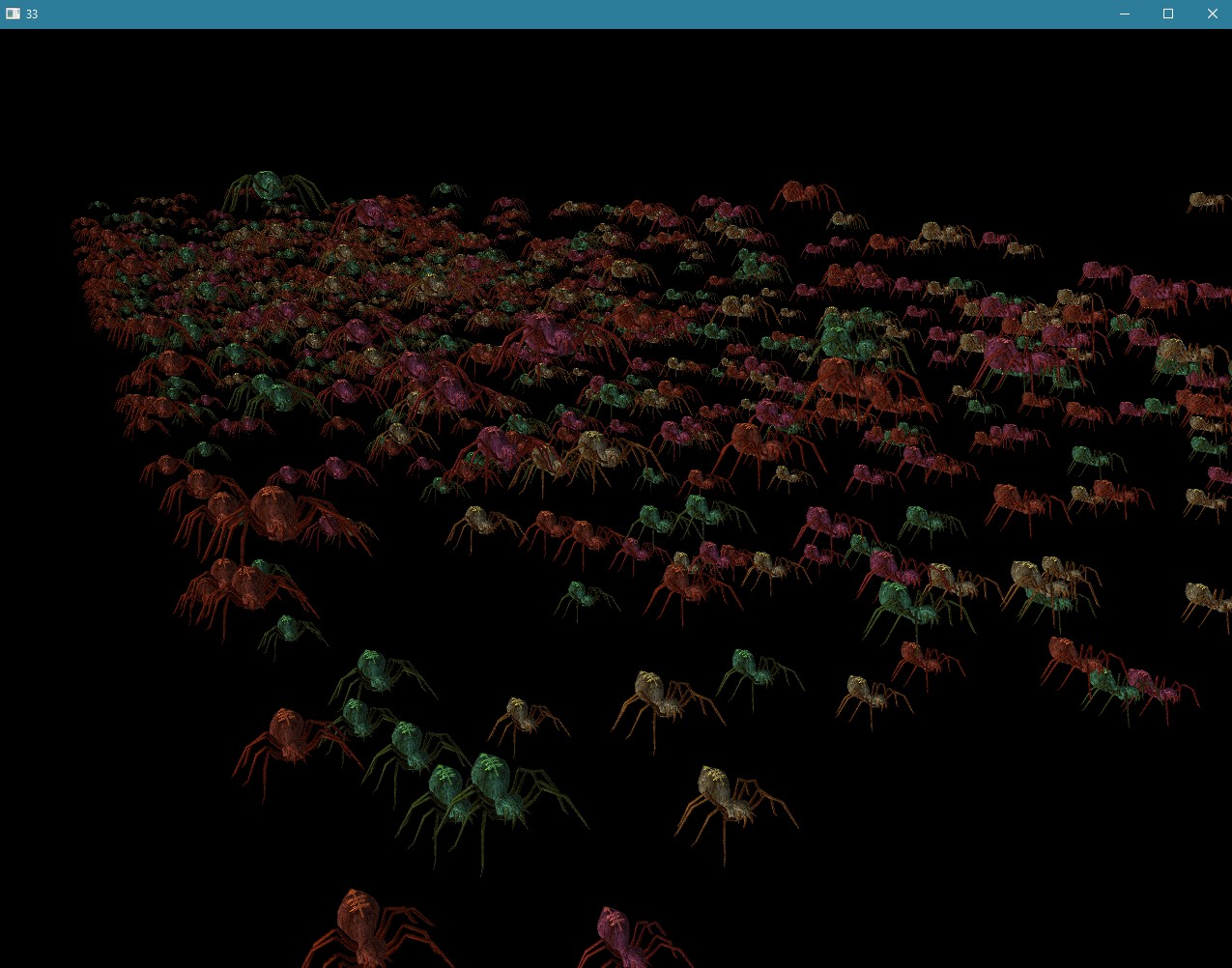


Рисунок 3 Дублирующий рендер

# Вывод

Продолжили изучать OpenGL. Научились работать с методом Billboarding. Изучили метод дублирующего рендера.

Ссылка на GitHub:

https://github.com/SemyonovGleb/Eacg

# Приложение 1

**main.cpp:**

#include <math.h>

#include <GL/glew.h>

#include <GL/freeglut.h>

#include "engine\_common.h"

#include "util.h"

#include "pipeline.h"

#include "camera.h"

#include "texture.h"

#include "lighting\_technique.h"

#include "glut\_backend.h"

#include "mesh.h"

#ifdef FREETYPE

#include "freetypeGL.h"

#endif

#define WINDOW\_WIDTH 1280

#define WINDOW\_HEIGHT 1024

#define NUM\_ROWS 50

#define NUM\_COLS 20

#define NUM\_INSTANCES NUM\_ROWS \* NUM\_COLS

class Tutorial33 : public ICallbacks

{

public:

Tutorial33()

{

m\_pGameCamera = NULL;

m\_pEffect = NULL;

m\_scale = 0.0f;

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

m\_directionalLight.AmbientIntensity = 0.55f;

m\_directionalLight.DiffuseIntensity = 0.9f;

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

m\_persProjInfo.FOV = 60.0f;

m\_persProjInfo.Height = WINDOW\_HEIGHT;

m\_persProjInfo.Width = WINDOW\_WIDTH;

m\_persProjInfo.zNear = 1.0f;

m\_persProjInfo.zFar = 100.0f;

m\_pMesh = NULL;

m\_frameCount = 0;

m\_fps = 0.0f;

}

~Tutorial33()

{

SAFE\_DELETE(m\_pEffect);

SAFE\_DELETE(m\_pGameCamera);

SAFE\_DELETE(m\_pMesh);

}

bool Init()

{

Vector3f Pos(7.0f, 3.0f, 0.0f);

Vector3f Target(0.0f, -0.2f, 1.0f);

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

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

m\_pEffect = new LightingTechnique();

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

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

return false;

}

m\_pEffect->Enable();

m\_pEffect->SetColorTextureUnit(COLOR\_TEXTURE\_UNIT\_INDEX);

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

m\_pEffect->SetMatSpecularIntensity(0.0f);

m\_pEffect->SetMatSpecularPower(0);

m\_pEffect->SetColor(0, Vector4f(1.0f, 0.5f, 0.5f, 0.0f));

m\_pEffect->SetColor(1, Vector4f(0.5f, 1.0f, 1.0f, 0.0f));

m\_pEffect->SetColor(2, Vector4f(1.0f, 0.5f, 1.0f, 0.0f));

m\_pEffect->SetColor(3, Vector4f(1.0f, 1.0f, 1.0f, 0.0f));

m\_pMesh = new Mesh();

if (!m\_pMesh->LoadMesh("C:/ogldev/Content/spider.obj")) {

return false;

}

#ifdef FREETYPE

if (!m\_fontRenderer.InitFontRenderer()) {

return false;

}

#endif

m\_time = glutGet(GLUT\_ELAPSED\_TIME);

CalcPositions();

return true;

}

void Run()

{

GLUTBackendRun(this);

}

virtual void RenderSceneCB()

{

CalcFPS();

m\_scale += 0.005f;

m\_pGameCamera->OnRender();

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

m\_pEffect->Enable();

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

Pipeline p;

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

p.SetPerspectiveProj(m\_persProjInfo);

p.Rotate(0.0f, 90.0f, 0.0f);

p.Scale(0.005f, 0.005f, 0.005f);

Matrix4f WVPMatrics[NUM\_INSTANCES];

Matrix4f WorldMatrices[NUM\_INSTANCES];

for (unsigned int i = 0 ; i < NUM\_INSTANCES ; i++) {

Vector3f Pos(m\_positions[i]);

Pos.y += sinf(m\_scale) \* m\_velocity[i];

p.WorldPos(Pos);

WVPMatrics[i] = p.GetWVPTrans().Transpose();

WorldMatrices[i] = p.GetWorldTrans().Transpose();

}

m\_pMesh->Render(NUM\_INSTANCES, WVPMatrics, WorldMatrices);

RenderFPS();

glutSwapBuffers();

}

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;

}

}

virtual void PassiveMouseCB(int x, int y)

{

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

}

virtual void MouseCB(int Button, int State, int x, int y)

{

}

private:

void CalcFPS()

{

m\_frameCount++;

int time = glutGet( GLUT\_ELAPSED\_TIME );

if (time - m\_time > 1000) {

m\_fps = (float)m\_frameCount \* 1000.0f / (time - m\_time);

m\_time = time;

m\_frameCount = 0;

}

}

void RenderFPS()

{

char text[32];

SNPRINTF(text, sizeof(text), "FPS: %.2f", m\_fps);

#ifdef FREETYPE

m\_fontRenderer.RenderText(10, 10, text);

#endif

}

void CalcPositions()

{

for (unsigned int i = 0; i < NUM\_ROWS ; i++) {

for (unsigned int j = 0 ; j < NUM\_COLS ; j++) {

unsigned int Index = i \* NUM\_COLS + j;

m\_positions[Index].x = (float)j;

m\_positions[Index].y = RandomFloat() \* 5.0f;

m\_positions[Index].z = (float)i;

m\_velocity[Index] = RandomFloat();

if (i & 1) {

m\_velocity[Index] \*= (-1.0f);

}

}

}

}

LightingTechnique\* m\_pEffect;

Camera\* m\_pGameCamera;

float m\_scale;

DirectionalLight m\_directionalLight;

Mesh\* m\_pMesh;

PersProjInfo m\_persProjInfo;

#ifdef FREETYPE

FontRenderer m\_fontRenderer;

#endif

int m\_time;

int m\_frameCount;

float m\_fps;

Vector3f m\_positions[NUM\_INSTANCES];

float m\_velocity[NUM\_INSTANCES];

};

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

{

Magick::InitializeMagick(\*argv);

GLUTBackendInit(argc, argv);

if (!GLUTBackendCreateWindow(WINDOW\_WIDTH, WINDOW\_HEIGHT, 32, false, "Tutorial 33")) {

return 1;

}

SRANDOM;

Tutorial33\* pApp = new Tutorial33();

if (!pApp->Init()) {

return 1;

}

pApp->Run();

delete pApp;

return 0;

}