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

Отчет по лабораторной работе № 1

на тему: «Ознакомление с OpenGL»

Выполнил:

Студент группы ПРО-233Б Ермолов И О

Проверил:

канд. техн. наук В.А. Котельников

Уфа - 2023

**Цель работы**

Познакомиться с OpenGL.

**Задание**

Выполнить 1-3 уроки по OpenGL <https://triplepointfive.github.io/ogltutor/>

**Ход работы**

1. Создание графических примитивов. В ходе лабораторной работы я изучил способ создания графических примитивов (точка, треугольник) путём задания их вершин и дальнейшей отрисовки.

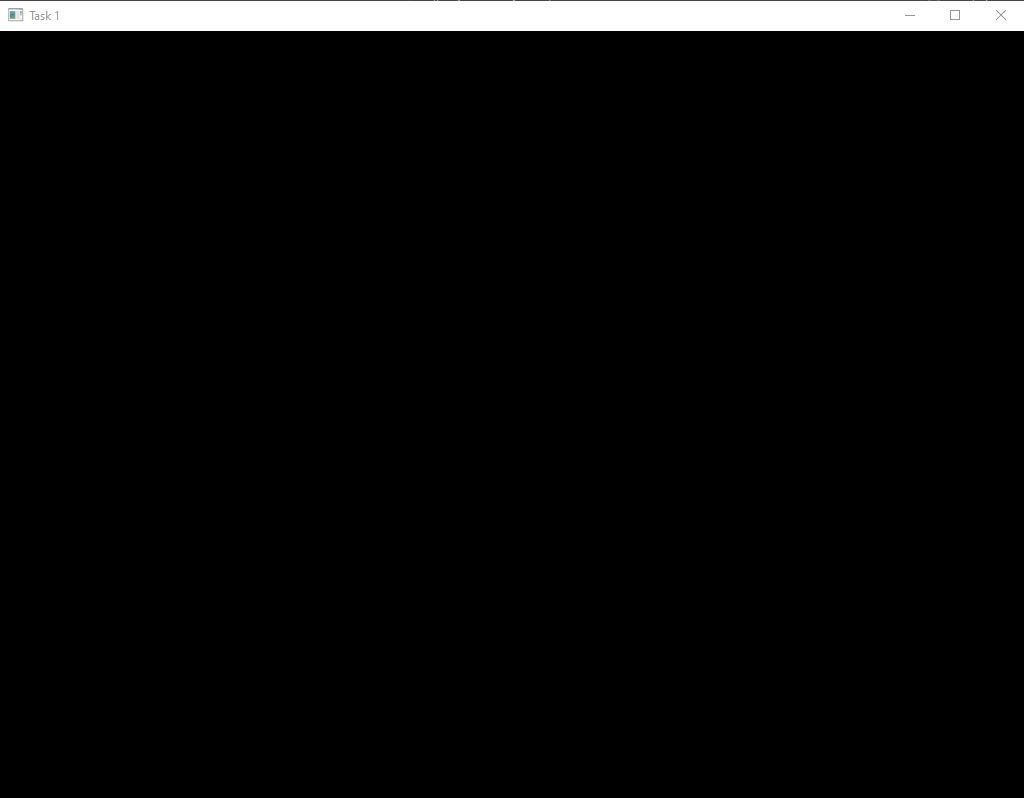


Рисунок 1. Создание окна

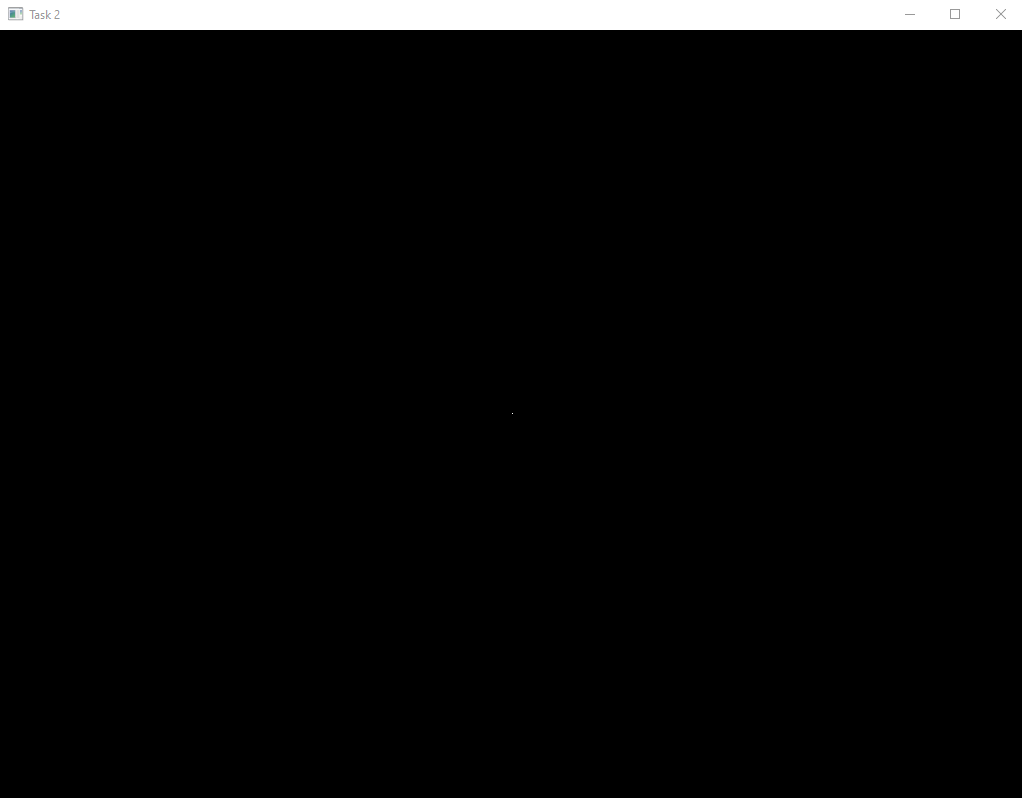


Рисунок 2. Отрисовка точки.

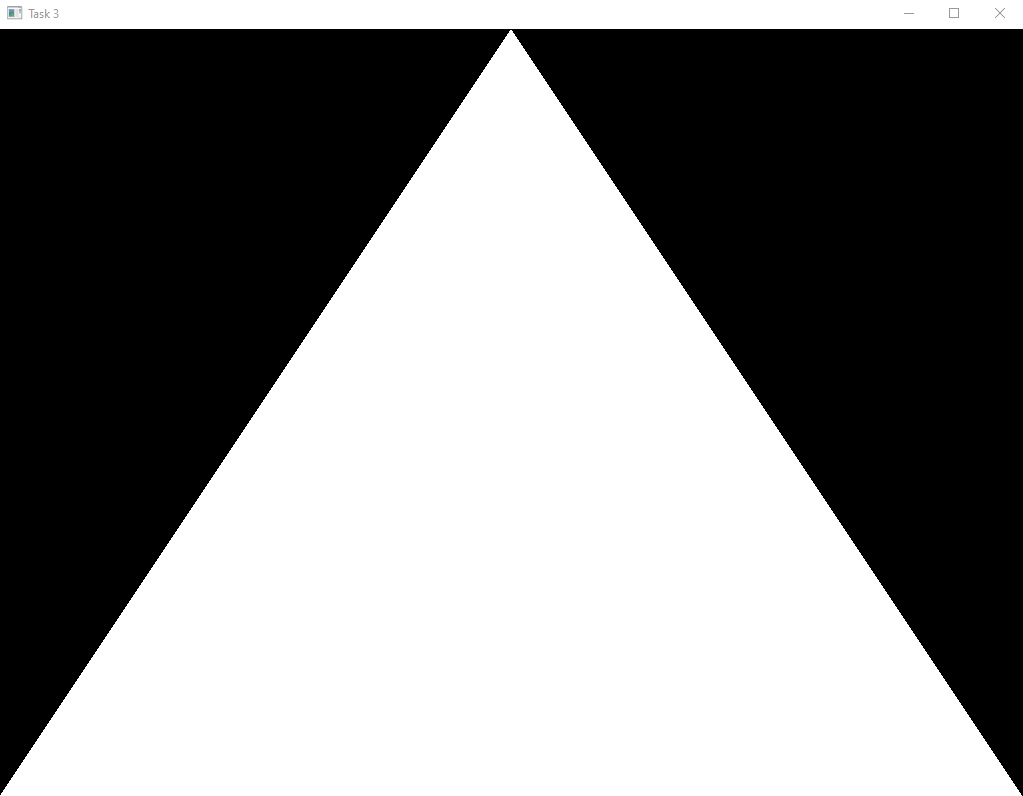


Рисунок 3. Отрисовка треугольника.

**Выводы к работе**

При выполнении данной лабораторной работы я познакомился с преобразованием трёхмерных объектов и изменением свойств камеры. Полный код программ по ссылке: [https://github.com/Ivan122727/EKG/tree/master/LW1](https://github.com/VerVyle/LW_ECG/tree/master/LW1)

**Приложение 1**

**math\_3d.cpp**

#include "math\_3d.h"

Vector3f Vector3f::Cross(const Vector3f& v) const

{

const float \_x = y \* v.z - z \* v.y;

const float \_y = z \* v.x - x \* v.z;

const float \_z = x \* v.y - y \* v.x;

return Vector3f(\_x, \_y, \_z);

}

Vector3f& Vector3f::Normalize()

{

const float Length = sqrtf(x \* x + y \* y + z \* z);

x /= Length;

y /= Length;

z /= Length;

return \*this;

}

void Matrix4f::InitScaleTransform(float ScaleX, float ScaleY, float ScaleZ)

{

m[0][0] = ScaleX; m[0][1] = 0.0f; m[0][2] = 0.0f; m[0][3] = 0.0f;

m[1][0] = 0.0f; m[1][1] = ScaleY; m[1][2] = 0.0f; m[1][3] = 0.0f;

m[2][0] = 0.0f; m[2][1] = 0.0f; m[2][2] = ScaleZ; m[2][3] = 0.0f;

m[3][0] = 0.0f; m[3][1] = 0.0f; m[3][2] = 0.0f; m[3][3] = 1.0f;

}

void Matrix4f::InitRotateTransform(float RotateX, float RotateY, float RotateZ)

{

Matrix4f rx, ry, rz;

const float x = ToRadian(RotateX);

const float y = ToRadian(RotateY);

const float z = ToRadian(RotateZ);

rx.m[0][0] = 1.0f; rx.m[0][1] = 0.0f; rx.m[0][2] = 0.0f; rx.m[0][3] = 0.0f;

rx.m[1][0] = 0.0f; rx.m[1][1] = cosf(x); rx.m[1][2] = -sinf(x); rx.m[1][3] = 0.0f;

rx.m[2][0] = 0.0f; rx.m[2][1] = sinf(x); rx.m[2][2] = cosf(x); rx.m[2][3] = 0.0f;

rx.m[3][0] = 0.0f; rx.m[3][1] = 0.0f; rx.m[3][2] = 0.0f; rx.m[3][3] = 1.0f;

ry.m[0][0] = cosf(y); ry.m[0][1] = 0.0f; ry.m[0][2] = -sinf(y); ry.m[0][3] = 0.0f;

ry.m[1][0] = 0.0f; ry.m[1][1] = 1.0f; ry.m[1][2] = 0.0f; ry.m[1][3] = 0.0f;

ry.m[2][0] = sinf(y); ry.m[2][1] = 0.0f; ry.m[2][2] = cosf(y); ry.m[2][3] = 0.0f;

ry.m[3][0] = 0.0f; ry.m[3][1] = 0.0f; ry.m[3][2] = 0.0f; ry.m[3][3] = 1.0f;

rz.m[0][0] = cosf(z); rz.m[0][1] = -sinf(z); rz.m[0][2] = 0.0f; rz.m[0][3] = 0.0f;

rz.m[1][0] = sinf(z); rz.m[1][1] = cosf(z); rz.m[1][2] = 0.0f; rz.m[1][3] = 0.0f;

rz.m[2][0] = 0.0f; rz.m[2][1] = 0.0f; rz.m[2][2] = 1.0f; rz.m[2][3] = 0.0f;

rz.m[3][0] = 0.0f; rz.m[3][1] = 0.0f; rz.m[3][2] = 0.0f; rz.m[3][3] = 1.0f;

\*this = rz \* ry \* rx;

}

void Matrix4f::InitTranslationTransform(float x, float y, float z)

{

m[0][0] = 1.0f; m[0][1] = 0.0f; m[0][2] = 0.0f; m[0][3] = x;

m[1][0] = 0.0f; m[1][1] = 1.0f; m[1][2] = 0.0f; m[1][3] = y;

m[2][0] = 0.0f; m[2][1] = 0.0f; m[2][2] = 1.0f; m[2][3] = z;

m[3][0] = 0.0f; m[3][1] = 0.0f; m[3][2] = 0.0f; m[3][3] = 1.0f;

}

void Matrix4f::InitCameraTransform(const Vector3f& Target, const Vector3f& Up)

{

Vector3f N = Target;

N.Normalize();

Vector3f U = Up;

U.Normalize();

U = U.Cross(N);

Vector3f V = N.Cross(U);

m[0][0] = U.x; m[0][1] = U.y; m[0][2] = U.z; m[0][3] = 0.0f;

m[1][0] = V.x; m[1][1] = V.y; m[1][2] = V.z; m[1][3] = 0.0f;

m[2][0] = N.x; m[2][1] = N.y; m[2][2] = N.z; m[2][3] = 0.0f;

m[3][0] = 0.0f; m[3][1] = 0.0f; m[3][2] = 0.0f; m[3][3] = 1.0f;

}

void Matrix4f::InitPersProjTransform(float FOV, float Width, float Height, float zNear, float zFar)

{

const float ar = Width / Height;

const float zRange = zNear - zFar;

const float tanHalfFOV = tanf(ToRadian(FOV / 2.0f));

m[0][0] = 1.0f / (tanHalfFOV \* ar); m[0][1] = 0.0f; m[0][2] = 0.0f; m[0][3] = 0.0;

m[1][0] = 0.0f; m[1][1] = 1.0f / tanHalfFOV; m[1][2] = 0.0f; m[1][3] = 0.0;

m[2][0] = 0.0f; m[2][1] = 0.0f; m[2][2] = (-zNear - zFar) / zRange; m[2][3] = 2.0f \* zFar \* zNear / zRange;

m[3][0] = 0.0f; m[3][1] = 0.0f; m[3][2] = 1.0f; m[3][3] = 0.0;

}

**task1.cpp**

#include <GL/freeglut.h>

#include "tasks.h"

static void RenderSceneCB() {

glClear(GL\_COLOR\_BUFFER\_BIT);

glutSwapBuffers();

}

static void InitializeGlutCallbacks() {

glutDisplayFunc(RenderSceneCB);

}

void invoke1(int argc, char\*\* argv) {

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGBA);

glutInitWindowSize(1024, 768);

glutInitWindowPosition(100, 100);

glutCreateWindow("Task 1");

InitializeGlutCallbacks();

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

glutMainLoop();

}

**task2.cpp**

#include "tasks.h"

#include <stdio.h>

#include <GL/glew.h>

#include <GL/freeglut.h>

#include "math\_3d.h"

GLuint VBO;

static void RenderSceneCB(){

glClear(GL\_COLOR\_BUFFER\_BIT);

glEnableVertexAttribArray(0);

glBindBuffer(GL\_ARRAY\_BUFFER, VBO);

glVertexAttribPointer(0, 3, GL\_FLOAT, GL\_FALSE, 0, 0);

glDrawArrays(GL\_POINTS, 0, 1);

glDisableVertexAttribArray(0);

glutSwapBuffers();

}

static void InitializeGlutCallbacks(){

glutDisplayFunc(RenderSceneCB);

}

static void CreateVertexBuffer(){

Vector3f Vertices[1];

Vertices[0] = Vector3f(0.0f, 0.0f, 0.0f);

glGenBuffers(1, &VBO);

glBindBuffer(GL\_ARRAY\_BUFFER, VBO);

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

}

void invoke2(int argc, char\*\* argv){

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGBA);

glutInitWindowSize(1024, 768);

glutInitWindowPosition(100, 100);

glutCreateWindow("Task 2");

InitializeGlutCallbacks();

// Must be done after glut is initialized!

GLenum res = glewInit();

if (res != GLEW\_OK) {

fprintf(stderr, "Error: '%s'\n", glewGetErrorString(res));

return;

}

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

CreateVertexBuffer();

glutMainLoop();

}

**task3.cpp**

#include "tasks.h"

#include <stdio.h>

#include <GL/glew.h>

#include <GL/freeglut.h>

#include "math\_3d.h"

GLuint VBO1;

static void RenderSceneCB() {

glClear(GL\_COLOR\_BUFFER\_BIT);

glEnableVertexAttribArray(0);

glBindBuffer(GL\_ARRAY\_BUFFER, VBO1);

glVertexAttribPointer(0, 3, GL\_FLOAT, GL\_FALSE, 0, 0);

glDrawArrays(GL\_TRIANGLES, 0, 3);

glDisableVertexAttribArray(0);

glutSwapBuffers(); // отрисовка

}

static void InitializeGlutCallbacks() {

glutDisplayFunc(RenderSceneCB); // Эта функция регулярно вызывается GLUT'ом.

}

static void CreateVertexBuffer() {

Vector3f Vertices[3];

Vertices[0] = Vector3f(-1.0f, -1.0f, 0.0f);

Vertices[1] = Vector3f(1.0f, -1.0f, 0.0f);

Vertices[2] = Vector3f(0.0f, 1.0f, 0.0f);

glGenBuffers(1, &VBO1);

glBindBuffer(GL\_ARRAY\_BUFFER, VBO1);

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

}

void invoke3(int argc, char\*\* argv) {

glutInit(&argc, argv); // Инициализация OpenGL utility library с параметрами из командной строки

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGBA); // GLUT\_DOUBLE включает двойную буферизацию, GLUT\_RGBA - буфер цвета, непрерывно используемого для конечной цели рендеринга (т.е. на экране).

glutInitWindowSize(1024, 768);

glutInitWindowPosition(100, 100);

glutCreateWindow("Task 3"); // Эти функции задают параметры окна и создают его. Так же мы даем ему заголовок.

InitializeGlutCallbacks(); // функция обратного вызова.

GLenum res = glewInit();

if (res != GLEW\_OK) {

fprintf(stderr, "Error: '%s'\n", glewGetErrorString(res));

return;

} // Теперь мы инициализируем GLEW и проверяем на ошибки. GLUT необходимо инициализировать перед этим.

glClearColor(0.0f, 0.0f, 0.0f, 0.0f); // устанавливает цвет, который будет использован во время очистки буфера кадра.

CreateVertexBuffer(); // создание треугольника.

glutMainLoop(); // передача управления GLUT.

}