ФГБОУ ВО

Уфимский университет науки и технологии

Кафедра ВМиК

Отчет

по лабораторной работе №1

по дисциплине «Инженерная и компьютерная графика»

Выполнил: ст.гр. ПРО-231

Самиков Д.З.

Принял: доцент

Котельников В.А.

Уфа 2023 г

**Цель работы**: познакомиться с OpenGL

Задание: выполнить 1-3 уроки по OpenGL

**Ход работы:**

[**Создание окна**](http://ogldev.atspace.co.uk/www/tutorial01/tutorial01.html)**:**

1) Инициализируем библиотеку GLUT и передаем ей аргументы командной строки.

3) Задаем режим отображения окна, в данном случае используется двойная буферизация и цветовая модель RGBA.

5) Устанавливаем размеры окна.

6) Устанавливаем позицию окна на экране и имя заголовка окна.

7) Устанавливаем цвет очистки экрана.

8) Задаем оконный режим работы программы.

9) Задаем функции рендера — это очистка буфера кадра

10) Компилируем запускаем.

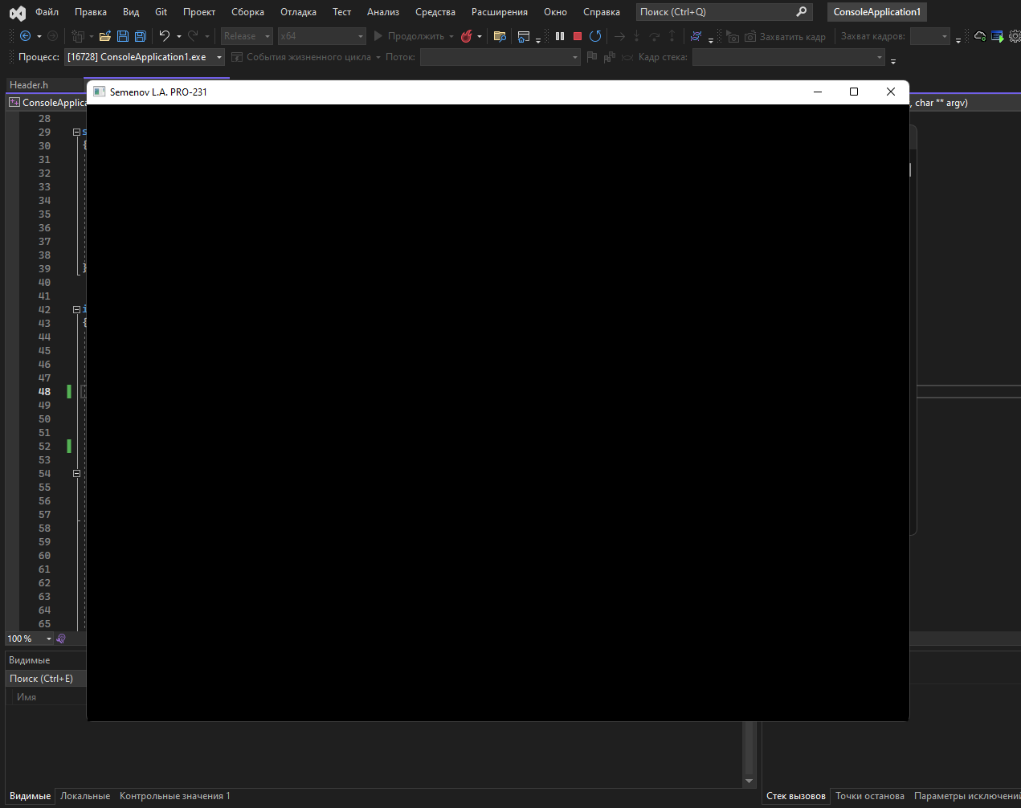


Рис 1. Вид созданного окна в приложении.

**Создание точки:**

1. Подключаем заголовок GLEW'a.
2. Добавляем вспомогательные структуру вектор.
3. Инициализируем GLEW и проверяем на ошибки
4. Создаем массив из одного экземпляра структуры Vector3f (этот тип объявлен в math\_3d.h) и задаём XYZ по нулям. Так задается точка в середине экрана.
5. Назначаем GLuint в качестве глобальной переменной для хранения указателя на буфер вершин
6. После связывания нашего объекта, мы наполняем его данными (название цели, размер данных в байтах, адрес массива вершин, и флаг, который обозначает использование паттернов для этих данных)
7. Привязываем наш буфер, приготавливая его для отрисовки. В этой маленькой программе мы имеем только 1 буфер вершин
8. Вызываем функцию для отрисовки.

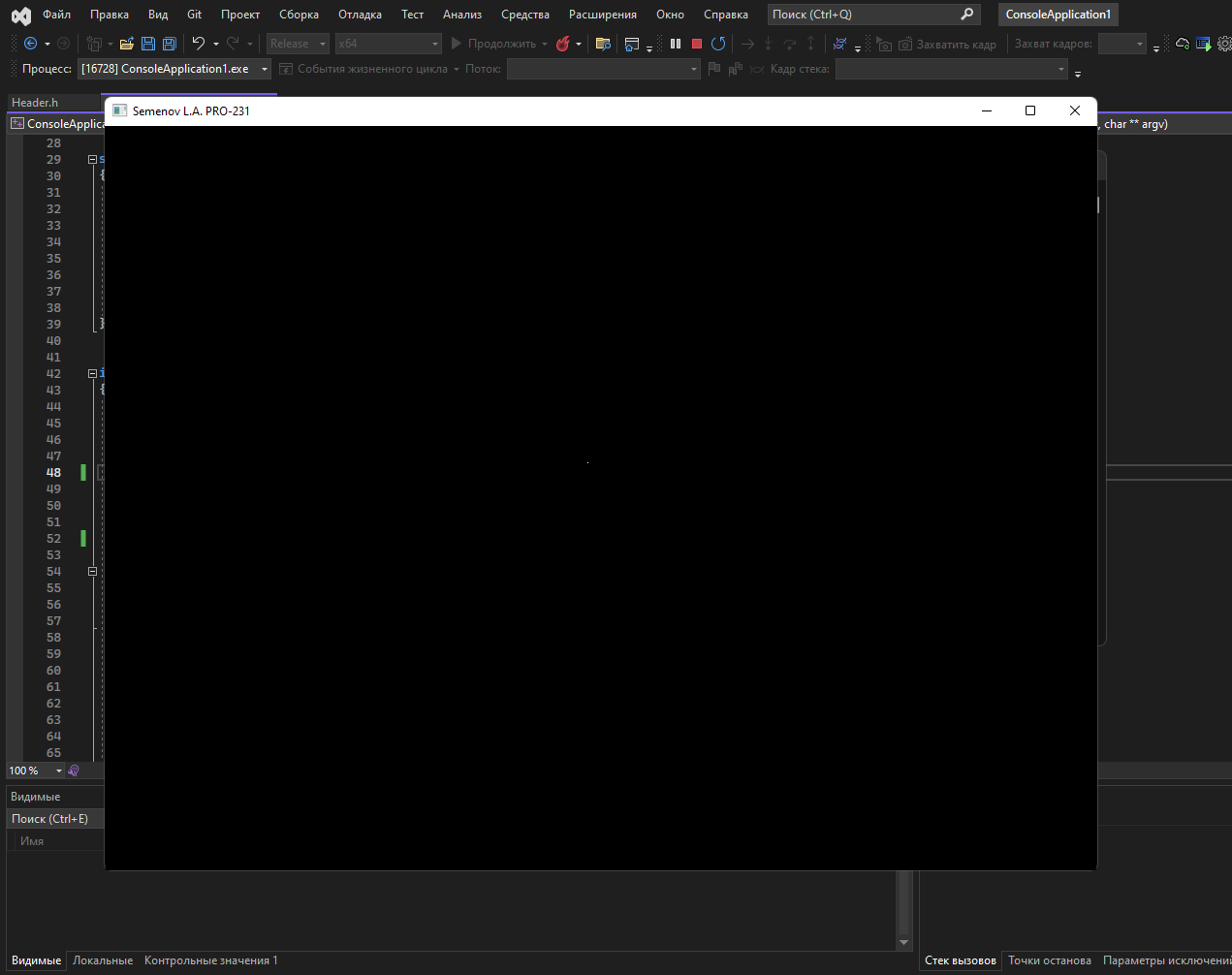


Рис 2. Работа программы с добавлением точки.

**Создание треугольника.**

1) Инициализируем библиотеку GLUT и передаем ей аргументы командной строки.

3) Задаем режим отображения окна, в данном случае используется двойная буферизация и цветовая модель RGBA.

5) Устанавливаем размеры окна.

6) Устанавливаем позицию окна на экране и имя заголовка окна.

7) Устанавливаем цвет очистки экрана.

8) Задаем оконный режим работы программы.

9) Задаем функции рендера — это очистка буфера кадра

10) Подключаем заголовок GLEW'a.

11) Добавляем вспомогательные структуру вектор.

12) Инициализируем GLEW и проверяем на ошибки

13) Создаем массив из одного экземпляра структуры Vector3f (этот тип объявлен в math\_3d.h) и задаём XYZ.

14) Увеличиваем массив, что бы он мог содержать 3 вершины.

15) Назначаем GLuint в качестве глобальной переменной для хранения указателя на буфер вершин

16) После связывания нашего объекта, мы наполняем его данными (название цели, размер данных в байтах, адрес массива вершин, и флаг, который обозначает использование паттернов для этих данных)

17) Задаем функцию и теперь мы рисуем треугольники вместо точек и принимаем 3 вершины вместо 1.

18) Привязываем наш буфер, приготавливая его для отрисовки.

19) Компилируем запускаем.

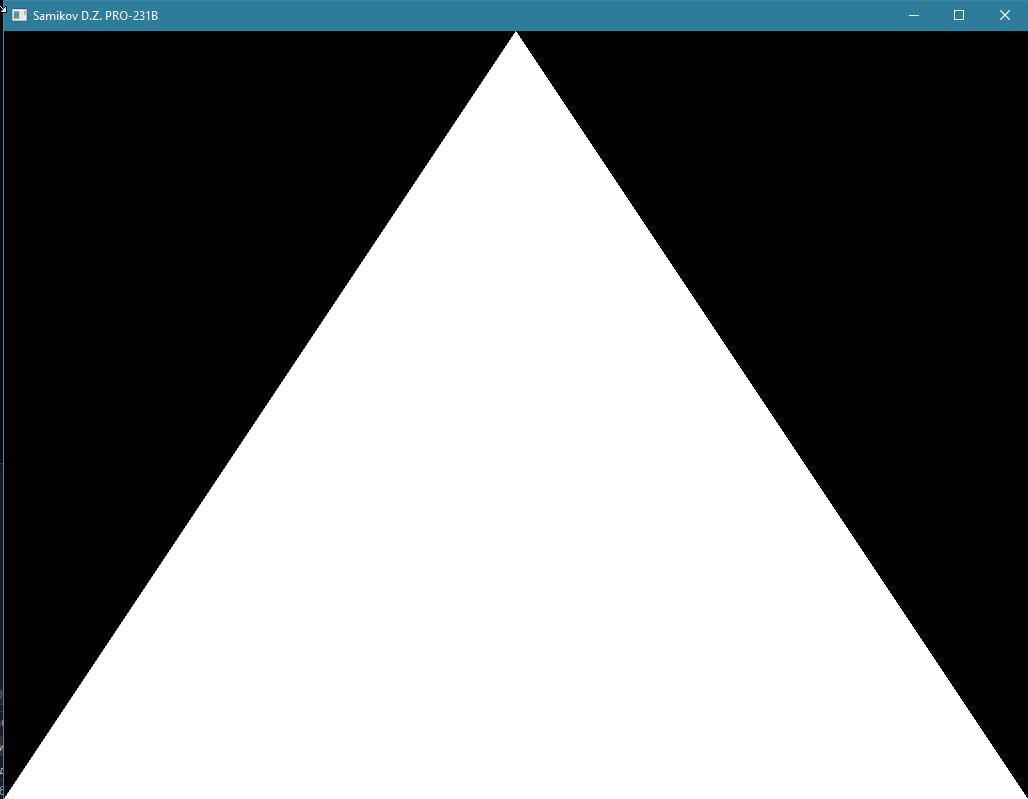


Рис 3. Работа программы с реализованным треугольником.

**Вывод:**

Я изучил примеры работ с OpenGL. Изучил и реализовал программы, которые выводят пустое окно, точки и треугольник на фоне.

Ссылка на GitHub:

https://github.com/SamDanPro/ICG

**Приложение 1.**

Код программы Main.cpp

#include <stdio.h>

#include <GL/glew.h>

#include <GL/freeglut.h>

#include "math\_3d.h"

GLuint VBO; //Объявление переменной-идентификатора буфера вершин.

static void RenderSceneCB() //Функция, которая будет вызываться для отрисовки сцены. Внутри функции:

{

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

glEnableVertexAttribArray(0); //Включение массива атрибутов вершин.

glBindBuffer(GL\_ARRAY\_BUFFER, VBO); //Привязка буфера вершин.

glVertexAttribPointer(0, 3, GL\_FLOAT, GL\_FALSE, 0, 0); //Указание способа интерпретации данных вершин (здесь: 3 координаты, тип данных float).

glDrawArrays(GL\_TRIANGLES, 0, 3);//Вызов функции отрисовки примитивов (здесь: треугольников).

glDisableVertexAttribArray(0);//Отключение массива атрибутов вершин.

glutSwapBuffers();//Переключение буферов (для двойной буферизации).

}

//Функция для установки обратного вызова для отрисовки сцены. Здесь она вызывает функцию RenderSceneCB.

static void InitializeGlutCallbacks()

{

glutDisplayFunc(RenderSceneCB);

}

//Функция для создания буфера вершин. Здесь создаются три вершины, затем создается буфер вершин, привязывается к контексту OpenGL и заполняется данными вершин.

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, &VBO); // генерация буфера вершин.

glBindBuffer(GL\_ARRAY\_BUFFER, VBO); //привязка буфера вершин к текущему контексту.

glBufferData(GL\_ARRAY\_BUFFER, sizeof(Vertices), Vertices, GL\_STATIC\_DRAW); //заполнение буфера вершин данными из массива.

}

int main(int argc, char\*\* argv)// это функция main программы, которая принимает аргументы командной строки и возвращает целочисленное значение.

{

glutInit(&argc, argv); // инициализирует библиотеку GLUT и передает ей аргументы командной строки.

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGBA); //задает режим отображения окна, в данном случае используется двойная буферизация и цветовая модель RGBA.

glutInitWindowSize(1024, 768); //устанавливает размеры окна.

glutInitWindowPosition(100, 100); //устанавливает позицию окна на экране.

glutCreateWindow("Samikov D.Z. PRO-231B"); //создает окно с заголовком

InitializeGlutCallbacks(); //устанавливает функцию обратного вызова для отрисовки сцены.

GLenum res = glewInit(); //инициализирует библиотеку GLEW и проверяет наличие всех необходимых расширений OpenGL.

if (res != GLEW\_OK) //проверяет успешность инициализации GLEW.

{

fprintf(stderr, "Error: '%s'\n", glewGetErrorString(res)); // выводит сообщение об ошибке в поток ошибок.

return 1;

}

glClearColor(0.0f, 0.0f, 0.0f, 0.0f); //устанавливает цвет очистки экрана.

CreateVertexBuffer(); //создает буфер вершин.

glutMainLoop(); //запускает бесконечный цикл обработки сообщений GLUT.

return 0;

}

**Приложение 2.**

Файл Header.h

#ifndef MATH\_3D\_H

#define MATH\_3D\_H

struct Vector3f

{

float x;

float y;

float z;

Vector3f()

{

}

Vector3f(float \_x, float \_y, float \_z)

{

x = \_x;

y = \_y;

z = \_z;

}

};

#endif