Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Сибирский государственный университет
телекоммуникаций и информатики» (СибГУТИ)

09.03.01 "Информатика и вычислительная техника"
профиль "Программное обеспечение средств
вычислительной техники и автоматизированных систем"

# ОТЧЕТ
по дисциплине «Программирование графических процессоров»
лабораторная работа 9

Выполнил:
Разумов Д. Б.
студент группы ИП-811

Проверил:
преподаватель Малков Е.А.

Новосибирск 2021

# Оглавление

# Задание

**Задание:** настроить среду для разработки OpenGL приложений (см. Лекция 10) и протестировать программу из лекции 10.

**Цель:** получить начальные навыки работы с OpenGL.

# Листинг

## Файл main.cpp

```
#include "constants.h"

void initGL();
int initBuffer();
void display();
void myCleanup();

GLFWwindow* window;

int main(){
	initGL();
	initBuffer(); //функция из util_template.cpp
	do {
		glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
		//glClearColor(0.7,0.7,0.7,1.0);
		glPointSize(6);
		display(); //основная функция
		glfwSwapBuffers(window);
		glfwPollEvents();
	} while (glfwGetKey(window, GLFW_KEY_ESCAPE ) != GLFW_PRESS
&&
		glfwWindowShouldClose(window) == 0 );

	glfwSetInputMode(window, GLFW_STICKY_KEYS, GL_TRUE);
	myCleanup();
```

```
	glfwTerminate();
	return 0;
}

void initGL() {
	//OpenGL - программный интерфейс для написания приложений, использующих двумерную и
	//трёхмерную компьютерную графику. Независимый от языков программирования и ОС
	if( !glfwInit() )
	{
		fprintf( stderr, "Failed to initialize GLFW\n" );
		getchar();
		return;
	}
	//функция glfwWindowHint задает параметры для функции glfwCreateWindow
	glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 4); //задать версию клиентского API,
	glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3); //с которым совместима данная программа
	glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE); //для OpenGL версии 3 и выше
	glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_COMPAT_PROFILE); //для какого профиля

		//OpenGL создавать контекст

	window = glfwCreateWindow( window_width, window_height,
		"Template window", NULL, NULL);
```

```
	if( window == NULL ) {
		fprintf( stderr, "Failed to open GLFW window. \n" );
		getchar();
		glfwTerminate();
		return;
	}

	glfwMakeContextCurrent(window);
	//инициализация GLEW - библиотеки для упрощения загрузки расширений OpenGL
	glewExperimental = true;
	if (glewInit() != GLEW_OK) {
		fprintf(stderr, "Failed to initialize GLEW\n");
		getchar();
		glfwTerminate();
		return;
	}
	return;
}
```

## Файл constants.h

```
#ifndef CONSTANTS_H
#define CONSTANTS_H

#include <stdio.h>
#include <string>
#include <stdlib.h>
```

```
#include <GL/glew.h>
#include <GLFW/glfw3.h>

const unsigned int window_width = 700;
const unsigned int window_height = 700;
const int num_of_verticies = 3; //число вершин

#endif
```

## Файл util_template.cpp

```
#include "constants.h"

#include <glm/vec3.hpp> // glm::vec3
#include <glm/vec4.hpp> // glm::vec4
#include <glm/mat4x4.hpp> // glm::mat4
#include <glm/gtc/matrix_transform.hpp>

GLuint bufferID;
GLuint progHandle;
GLuint genRenderProg();

int initBuffer() { //выделяем память, делаем ее текущей, инициализируем
	glGenBuffers(1, &bufferID); //выделение памяти(кол-во буферов, массив идентификаторов)
	//буфер - это область памяти
	glBindBuffer(GL_ARRAY_BUFFER, bufferID); //делаем буфер текущим
	static const GLfloat vertex_buffer_data[] = { //инициализируем массив
```

```
        -0.9f, -0.9f, -0.0f, 1.0f, 0.0f, 0.0f,
        0.0f, 0.0f, 0.0f, 0.0f, 1.0f, 0.0f,
        0.9f, -0.5f, 0.0f, 0.0f, 0.0f, 1.0f,
    };
    glBufferData( //выделить память и скопировать туда созданный массив
        GL_ARRAY_BUFFER, //тип памяти
        6 * num_of_verticies*sizeof(float), //размер памяти
        vertex_buffer_data, //указатель на хосте
        GL_STATIC_DRAW); //только для чтения
    //glBindBuffer(GL_ARRAY_BUFFER, inA);
    return 0;
}

void camera() {
    glm::mat4 Projection = glm::perspective(glm::radians(60.0f),
        (float) window_width / (float)window_height, 0.1f, 0.0f);
    glm::mat4 View = glm::lookAt( //местонахождение камеры
        glm::vec3(0,1,1), // Камера находится в точке (x,y,z)
        glm::vec3(0,0,0), // и направлена на начало координат.
        glm::vec3(0,1,0) // Ось Y направлена вверх, ( 0,-1,0) - вниз.
    );

    glm::mat4 Model = glm::mat4(1.0f); //единичная матрица
    glm::mat4 mvp = Projection * View * Model; //матрица конечная - как и откуда смотрим

    GLuint MatrixID = glGetUniformLocation(progHandle, "MVP"); //передать матрицу шейдеру
```

```
	glUniformMatrix4fv(MatrixID, 1, GL_FALSE, &mvp[0][0]); //передаем только для чтения
}

void display() { //здесь выводится изображение
	progHandle = genRenderProg(); //см. sh_template.cpp
	glUseProgram(progHandle); //делаем программу текущей
	camera();
	GLint posPtr = glGetAttribLocation(progHandle, "pos"); //получить указатель на программу
	//pos связывается с буфером (массивом чисел, где координаты и цвета)
		//по имени "pos"
	glVertexAttribPointer( //атрибуты
		posPtr, //переменная-вектор
		3, //3 значения пропускать для каждого элемента в массиве из initBuffer()
		GL_FLOAT, //тип = вещественные числа
		GL_FALSE,
		24, //размерность масива
		0); //0 = с самого начала буфера
	glEnableVertexAttribArray(posPtr); //делаем указатель активным

	GLint colorPtr = glGetAttribLocation(progHandle, "color"); //цвет
	glVertexAttribPointer( //передать цвет
		colorPtr, //цвет
		3, //по 3 элемента передаем данные
		GL_FLOAT, //веществены числа
		GL_FALSE,
```

```
		24, //размер в байтах
		(const GLvoid*)12); //смещение
	glEnableVertexAttribArray(colorPtr);
	glDrawArrays( //запустить программу
		GL_TRIANGLES, //интерпретировать данные как вершины треугольника
		0, //начинать с нуля байт
		num_of_verticies); //количество вершин
	glDisableVertexAttribArray(posPtr); //сделать неактивным указатель
	glDisableVertexAttribArray(colorPtr);
	//"Vertex" в функциях обозначает вершинные шейдеры
}

void myCleanup() { //освобождение ресурсов - сделать неактивным
	glDeleteBuffers(1, &bufferID);
	glDeleteProgram(progHandle);
}
```

## Файл sh_template.cpp

```
#include "constants.h"

void checkErrors(std::string desc) {
	GLenum e = glGetError();
	if (e != GL_NO_ERROR) {
		fprintf(stderr, "OpenGL error in \"%s\": %s (%d)\n", desc.c_str(),
		gluErrorString(e), e);
		exit(20);
```

```
	}
}

GLuint genRenderProg() {
	//создаем шейдер - код в виде строки комплириуем и компонуем
	//преобразования выполняются на хосте, шейдеры выполняются на устройстве
	//возвращает айди программы
	GLuint progHandle = glCreateProgram(); //создать программу, получить дескриптор
	GLuint vp = glCreateShader(GL_VERTEX_SHADER); //создать вершинный шейдер
	GLuint fp = glCreateShader(GL_FRAGMENT_SHADER); //создать фрагментный шейдер

	//строки ниже:
		//версия вершинного шейдера
		//переменные (pos, color) длины 3
		//out = переменная будет передана фрагментному шейдеру
		////матрица конечная - как и откуда смотрим
		//void main()
			//gl_Poition - это встроенная переменная; расширяем позицию до 4-мерного
			//4 координата для сдвижения в пространстве
			//инициализировать цвет
	const char *vpSrc[] = {
		"#version 430\n",
		"layout(location = 0) in vec3 pos;\
		layout(location = 1) in vec3 color;\
```

```
        out vec4 vs_color;\
        uniform mat4 MVP;\
        void main() {\
                gl_Position = MVP*vec4(pos,1);\
                vs_color=vec4(color,1.0);\
        }"
};
const char *fpSrc[] = {
        "#version 430\n",
        "in vec4 vs_color;\
        out vec4 fcolor;\
        void main() {\
                fcolor = vs_color;\
        }"
};

glShaderSource( //связать дескриптор шейдера и описание
        vp, //дескриптор шейдера
        2, //сколько строк
        vpSrc, //массив строк
        NULL); //массив длин строк
glShaderSource(fp, 2, fpSrc, NULL);

glCompileShader(vp); //комплируем шейдер
int rvalue; //return value
glGetShaderiv(vp, GL_COMPILE_STATUS, &rvalue);
```

```
    if (!rvalue) { //обработка ошибки
        fprintf(stderr, "Error in compiling vp\n");
        exit(30);
    }
    glAttachShader(progHandle, vp); //включить шейдер в программу

    //все то же самое с фрагментным шейдером:
    glCompileShader(fp);
    glGetShaderiv(fp, GL_COMPILE_STATUS, &rvalue);
    if (!rvalue) {
        fprintf(stderr, "Error in compiling fp\n");
        exit(31);
    }
    glAttachShader(progHandle, fp);

    glLinkProgram(progHandle); //компонуем программу
    glGetProgramiv(progHandle, GL_LINK_STATUS, &rvalue);
    if (!rvalue) {
        fprintf(stderr, "Error in linking sp\n");
        exit(32);
    }
    checkErrors("Render shaders");

    return progHandle; //вернуть дескриптор программы
}
```

# Скриншоты

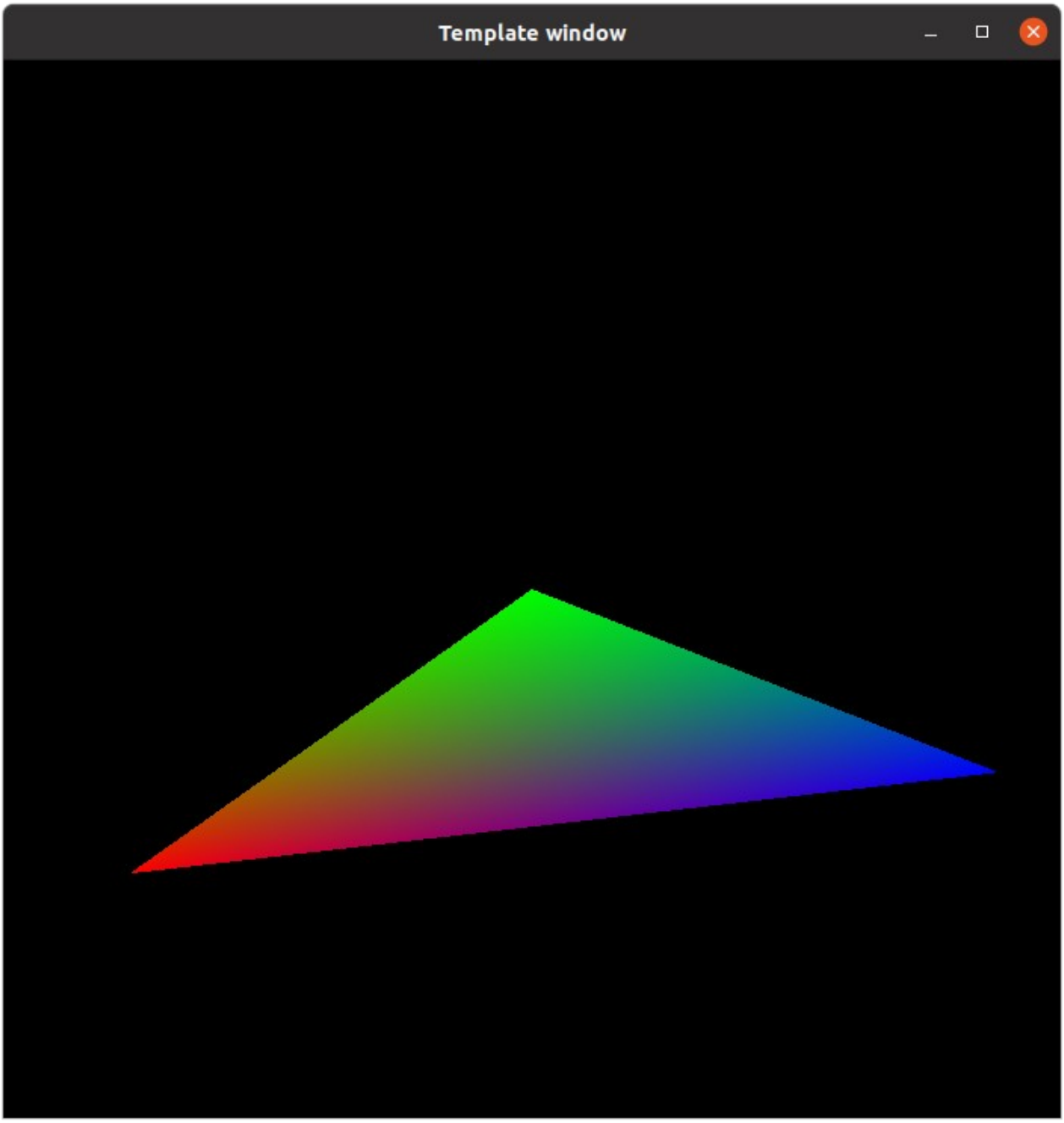


Рисунок 1 — созданное окно

# Об используемом GPU

Device name: GeForce MX350

Global memory available on device: 2099904512 (2002 MByte)

Shared memory available per block: 49152

Count of 32-bit registers available per block: 65536

Warp size in threads: 32

Maximum pitch in bytes allowed by memory copies: 2147483647

Maximum number of threads per block: 1024

Maximum size of each dimension of a block[0]: 1024

Maximum size of each dimension of a block[1]: 1024

Maximum size of each dimension of a block[2]: 64

Maximum size of each dimension of a grid[0]: 2147483647

Maximum size of each dimension of a grid[1]: 65535

Maximum size of each dimension of a grid[2]: 65535

Clock frequency in kilohertz: 1468000

totalConstMem: 65536

Major compute capability: 6

Minor compute capability: 1

Number of multiprocessors on device: 5

Count of cores: 640

Id current device: 0

Id nearest device to 1.3: 0

# Заключение

OpenGL — это программный интерфейс, который используется для написания приложений, использующих двумерную и трёхмерную компьютерную графику. OpenGL использует драйверы видеокарт Nvidia, Intel, AMD.