|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кафедра |  | O7 |  | Информационные системы и программная инженерия |
|  |  | шифр |  | наименование кафедры, по которой выполняется работа |
| Дисциплина |  | Компьютерная геометрия и графика | | |
|  |  | наименование дисциплины | | |

|  |  |  |
| --- | --- | --- |
| ЛАБОРАТОРНАЯ РАБОТА | 3 |  |
|  | номер (при наличии) |  |
| Использование массивов индексов | | |
| Преобразования координат | | |
|  | | |

при наличии указать тему лабораторной работы и (или) номер варианта

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ОБУЧАЮЩИЙСЯ** | | | | | | |
| группы | | | |  | | И913Б |
|  |  | Кульга Н.И | | | | |
| подпись |  | фамилия и инициалы | | | | |
|  | | | | |
| дата сдачи | | | | |
| **ПРОВЕРИЛ** | | | | | | |
|  | | | | | | | |
| ученая степень, ученое звание, должность | | | | | | | |
|  |  | Мажайцев Е.А | | | | | |
| подпись |  | фамилия и инициалы | | | | | |
| Оценка / балльная оценка | | |  | | | |
|  | | | | |
| дата проверки | | | | |

Задание:

Задание 1. Создать массив вершин, содержащий координаты n вершин. Используя массив индексов нарисовать:

a. Набор из 2\*n различных треугольников;

b. Набор из 3\*n различных линий.

Все объекты должны быть изображены каркасно, каждый своим цветом.

Задание 2. Нарисуйте треугольник, прямоугольник и линию. Получите новое изображение, на котором:

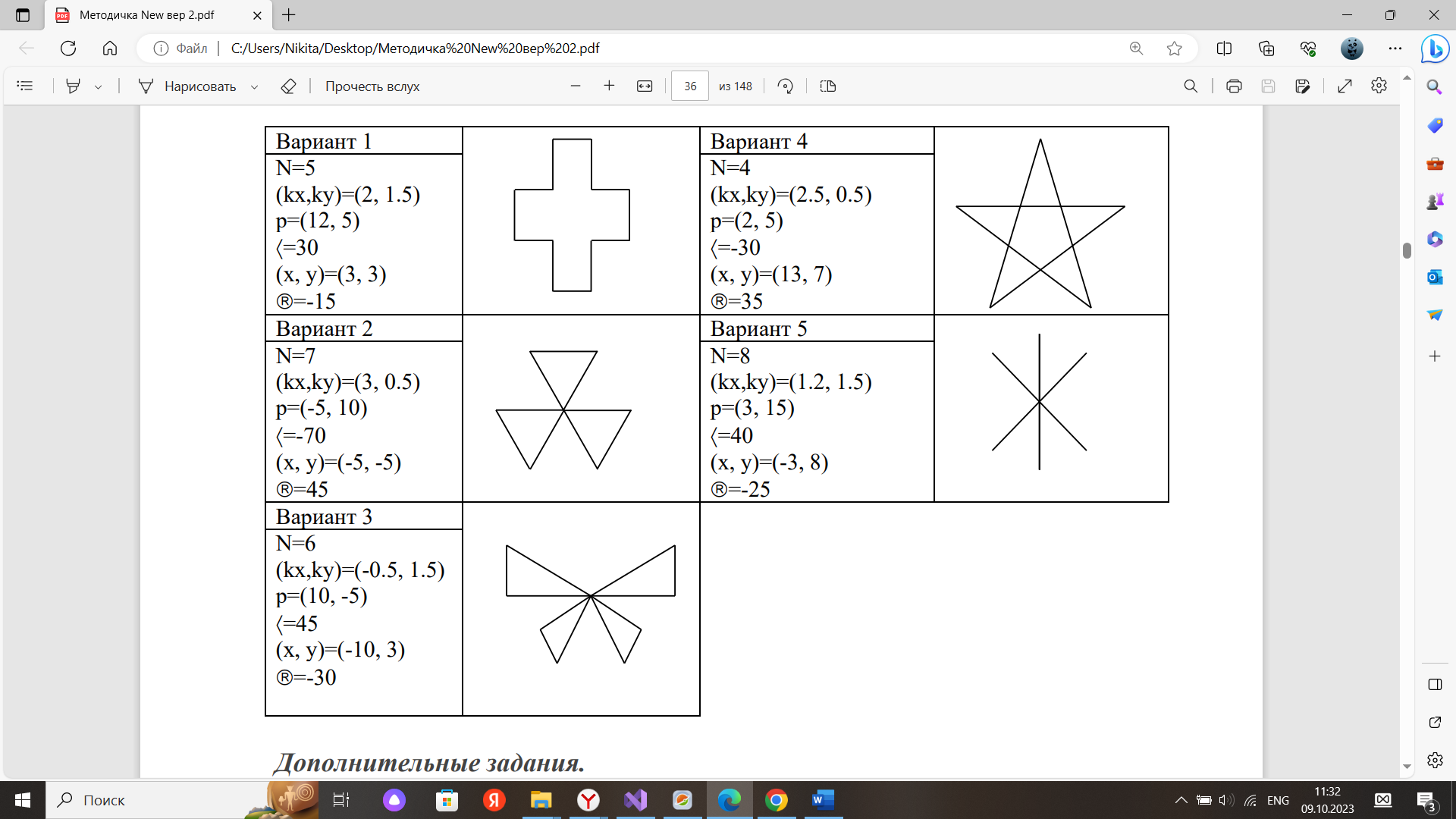
● треугольник следует смасштабировать с коэффициентами (kx, ky) и перенести

на вектор p;

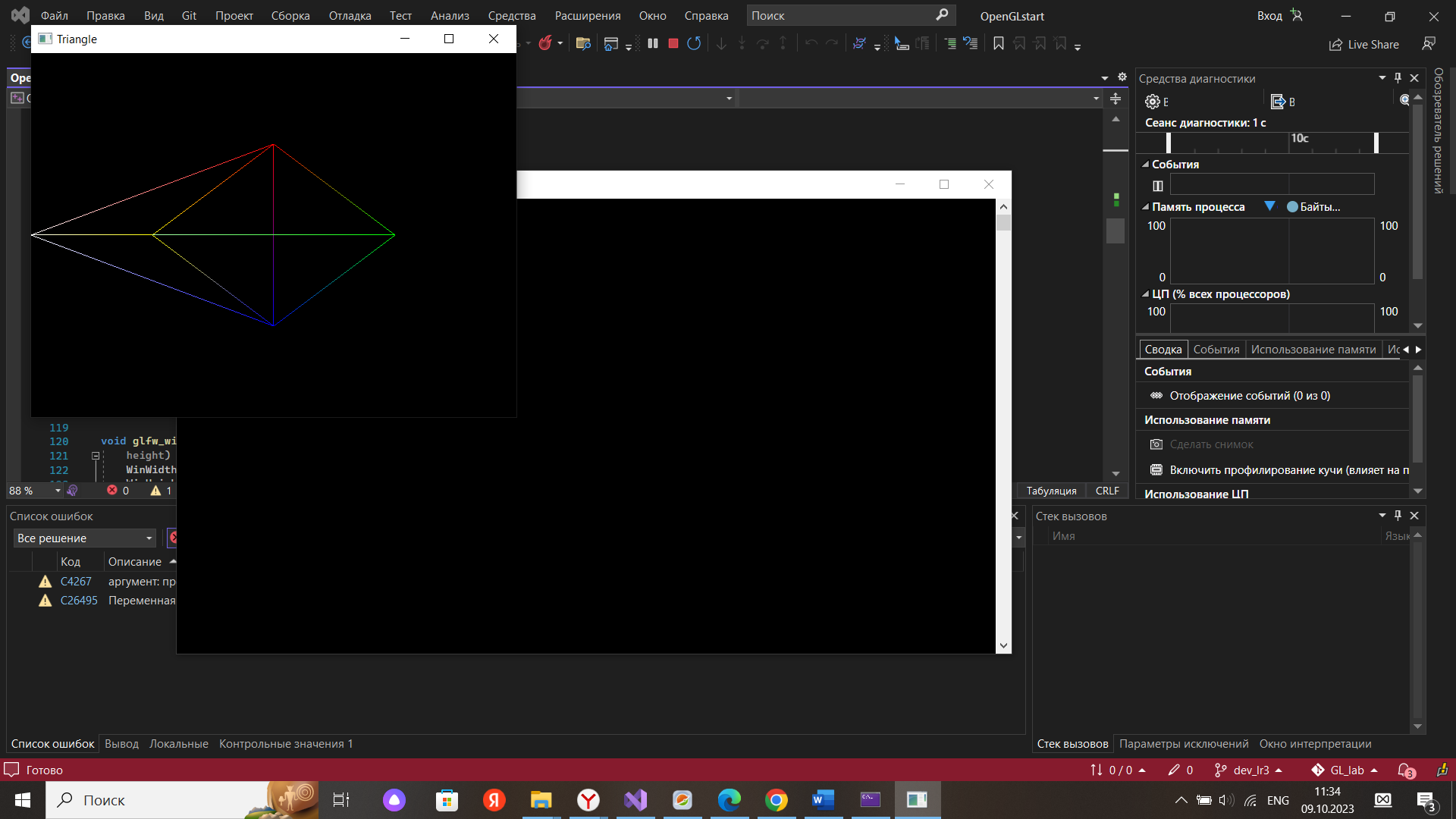
● линию повернуть на угол относительно начала координат;

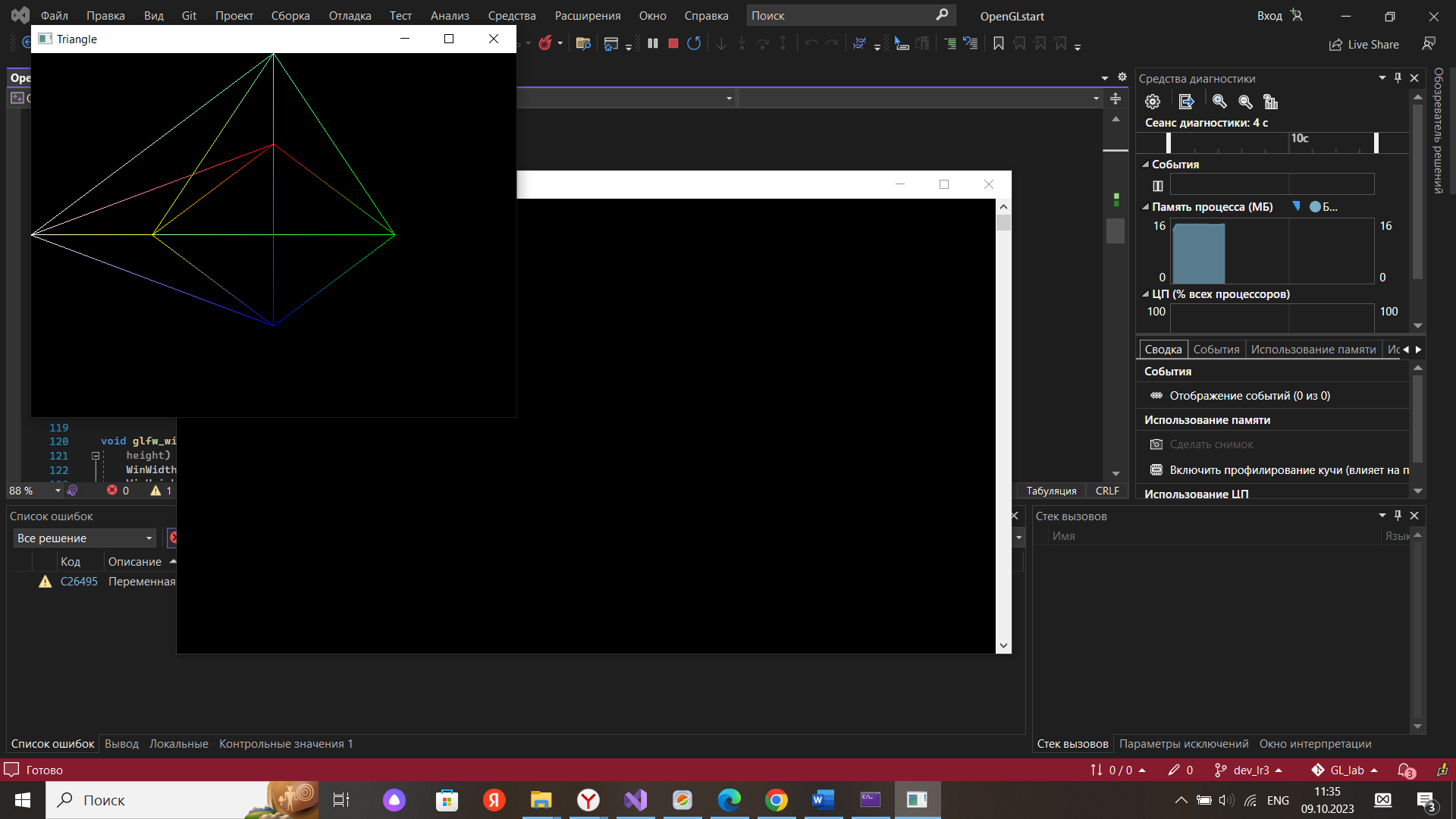
● прямоугольник повернуть относительно точки с координатами (x, y) на угол.

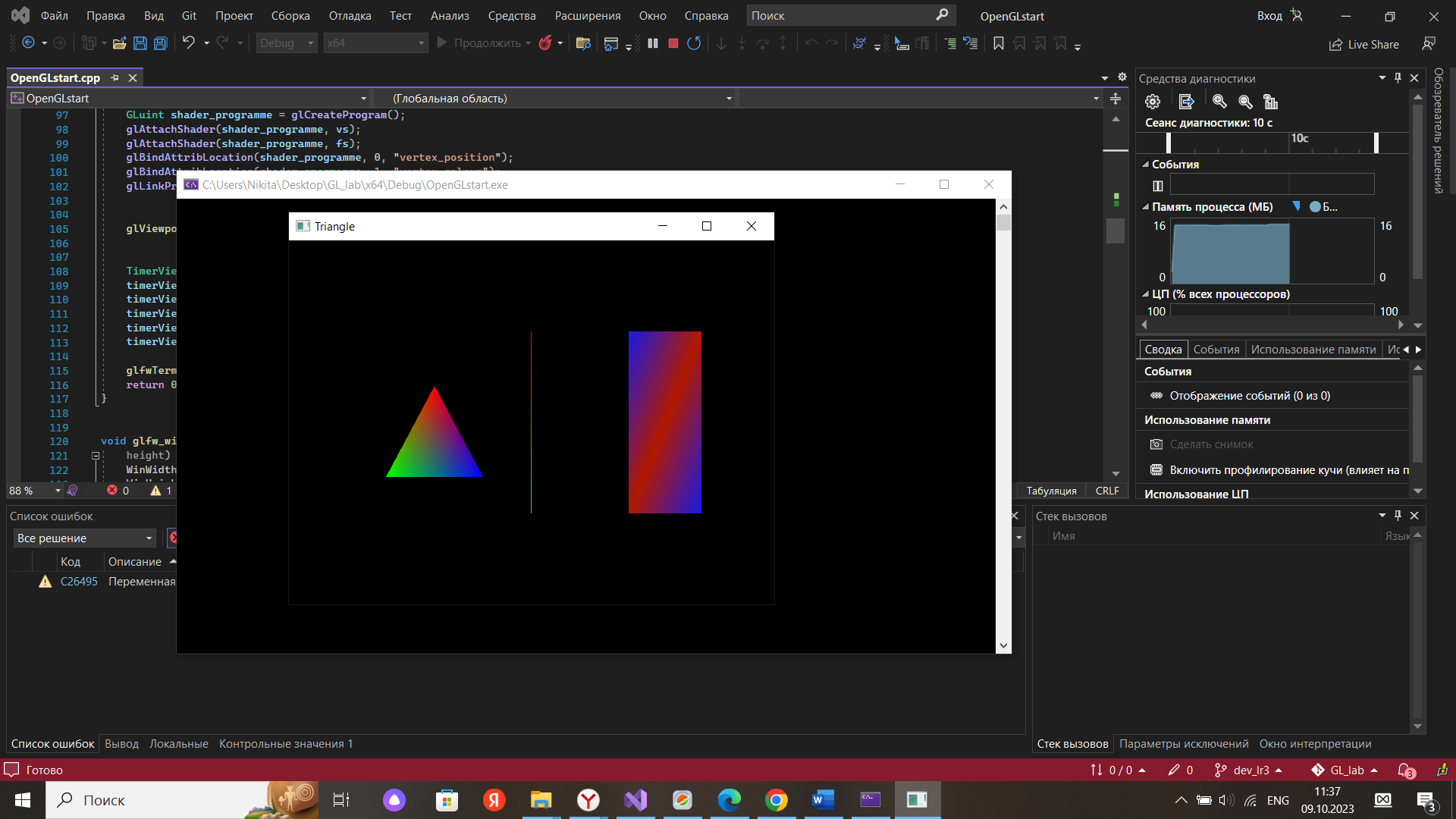
Задание 3. Используя примитив для рисования линий и операции геометрических преобразований изобразить фигуру, показанную на рисунке.

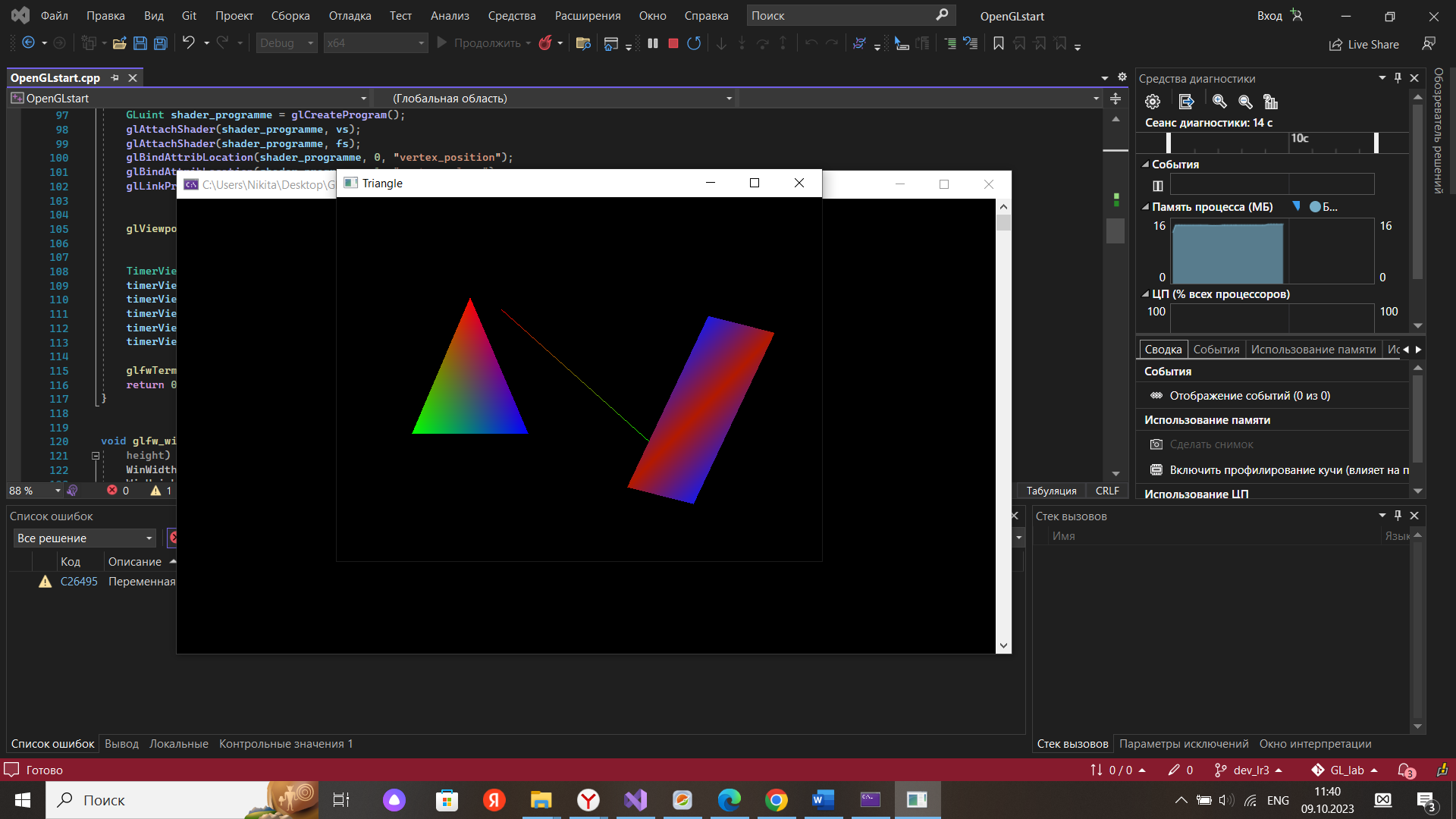


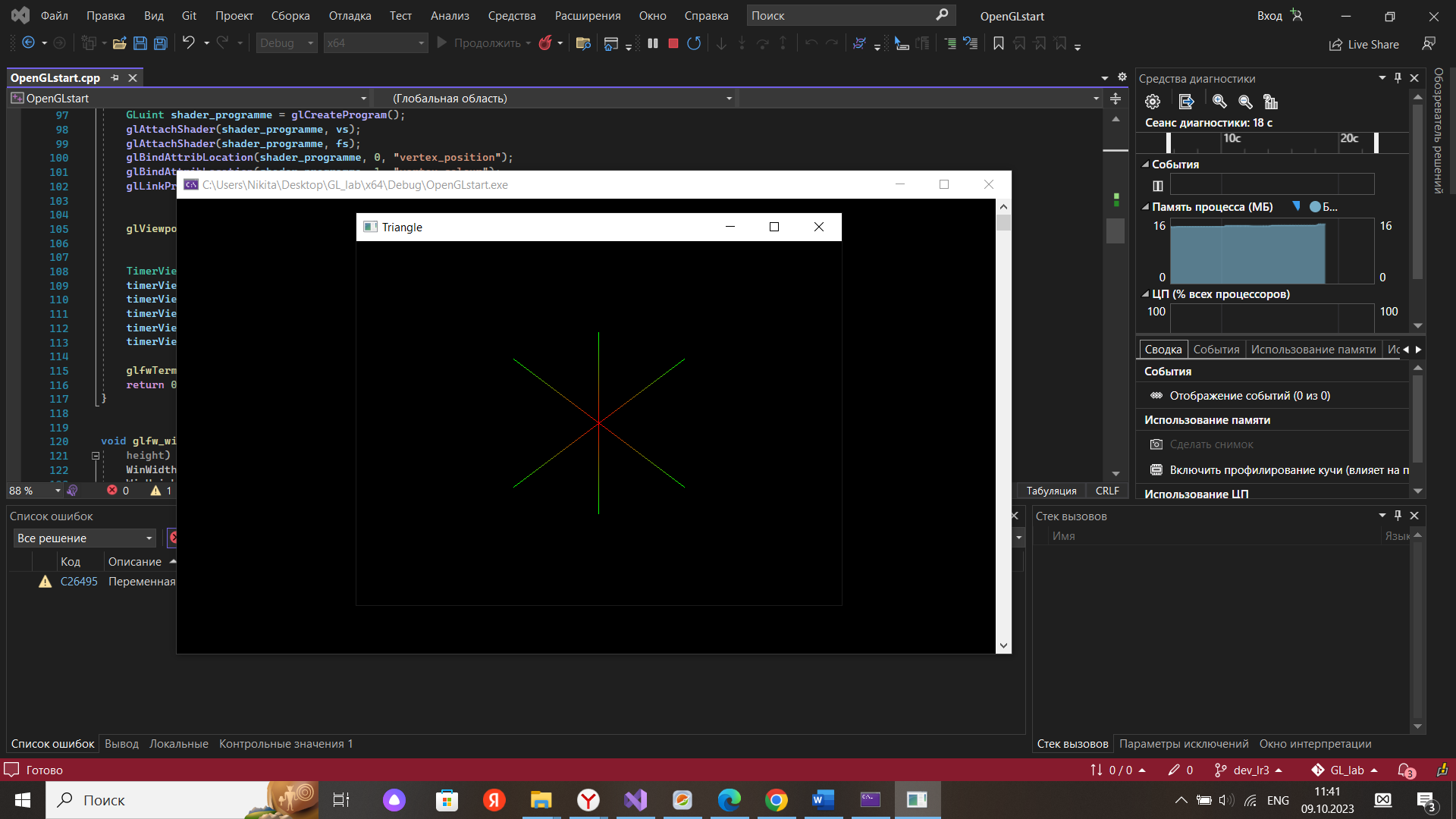
Результат работы программы:











Код программы:

Main.cpp:

#include "glew.h" // подключение GLEW

#include <glfw3.h>

#include <iostream>

#include "ShaderFuncs.h"

#include "glm/glm.hpp"

#include "glm/gtc/matrix\_transform.hpp"

#include <vector>

#include <chrono>

#include <functional>

#include <thread>

class TimerView {

public:

TimerView(GLuint shader\_programme, GLFWwindow\* window) : shader\_programme(shader\_programme),window(window) {}

void add(std::chrono::milliseconds delay\_ms, std::function<void(GLuint shader\_programme)> callback) {

// Создаем объект std::chrono::milliseconds для задержки

glfwPollEvents();

glClear(GL\_COLOR\_BUFFER\_BIT);

glUseProgram(shader\_programme);

callback(shader\_programme); // Здесь передаем shader\_programme в callback

glfwSwapBuffers(window);

std::chrono::milliseconds delay(delay\_ms);

std::this\_thread::sleep\_for(delay);

}

void add(std::chrono::milliseconds delay\_ms, std::function<void(GLuint shader\_programme, bool trueOrFalse)> callback, bool trueOrFalseFlag=false) {

// Создаем объект std::chrono::milliseconds для задержки

glfwPollEvents();

glClear(GL\_COLOR\_BUFFER\_BIT);

glUseProgram(shader\_programme);

callback(shader\_programme, trueOrFalseFlag); // Здесь передаем shader\_programme в callback

glfwSwapBuffers(window);

std::chrono::milliseconds delay(delay\_ms);

std::this\_thread::sleep\_for(delay);

}

private:

GLuint shader\_programme; // Добавьте поле для хранения shader\_programme

GLFWwindow\* window;

};

struct dataPoint {

std::vector<glm::vec3> coordinates;

std::vector<glm::vec3> colors;

size\_t sizeDataPoints;

void addCoordinates(glm::vec3 coordinate) {

coordinates.push\_back(coordinate);

sizeDataPoints= (coordinates.size()) == 0 ? 0 : 3 \* coordinates.size();

}

void addColors(glm::vec3 color) {

colors.push\_back(color);

}

// Метод для получения указателя на данные координат

glm::vec3\* getCoordinatesData() {

return coordinates.data();

}

// Метод для получения указателя на данные цветов

glm::vec3\* getColorsData() {

return colors.data();

}

};

int WinWidth = 640;

int WinHeight = 480;

void glfw\_window\_size\_callback(GLFWwindow\* window, int width, int height);

void drawSmth(dataPoint dataPoints, std::vector<GLuint> indexes, glm::mat4 transformMatrix, GLuint shader\_programme, GLenum type);

void task1(GLuint shader\_programme);

void task\_1\_a(GLint shader\_programme);

void task\_1\_b(GLint shader\_programme);

void task\_2(GLint shader\_programme, bool trueOrFalse);

void task\_3(GLint shader\_programme);

int main() {

glfwInit();

GLFWwindow\* window = glfwCreateWindow(WinWidth, WinHeight, "Triangle", NULL, NULL);

glfwMakeContextCurrent(window);

glewInit();

glfwSetWindowSizeCallback(window, glfw\_window\_size\_callback);

GLuint vs = LoadShader("VertexShader.glsl", GL\_VERTEX\_SHADER);

GLuint fs = LoadShader("FragmentShader.glsl", GL\_FRAGMENT\_SHADER);

GLuint shader\_programme = glCreateProgram();

glAttachShader(shader\_programme, vs);

glAttachShader(shader\_programme, fs);

glBindAttribLocation(shader\_programme, 0, "vertex\_position");

glBindAttribLocation(shader\_programme, 1, "vertex\_colour");

glLinkProgram(shader\_programme);

glViewport(0, 0, WinWidth, WinHeight);

TimerView timerView(shader\_programme,window);

timerView.add(std::chrono::milliseconds(4000), task\_1\_a);

timerView.add(std::chrono::milliseconds(4000), task\_1\_b);

timerView.add(std::chrono::milliseconds(4000), task\_2);

timerView.add(std::chrono::milliseconds(4000), task\_2,true);

timerView.add(std::chrono::milliseconds(4000), task\_3);

glfwTerminate();

return 0;

}

void glfw\_window\_size\_callback(GLFWwindow\* window, int width, int

height) {

WinWidth = width;

WinHeight = height;

}

void drawSmth(dataPoint dataPoints, std::vector<GLuint> indexes, glm::mat4 transformMatrix, GLuint shader\_programme, GLenum type) {

GLuint stm = glGetUniformLocation(shader\_programme, "transform");

glUniformMatrix4fv(stm, 1, GL\_FALSE, &transformMatrix[0][0]);

//БУФЕРЫ

GLuint coords\_vbo = 0;

glGenBuffers(1, &coords\_vbo);

glBindBuffer(GL\_ARRAY\_BUFFER, coords\_vbo);

glBufferData(GL\_ARRAY\_BUFFER, dataPoints.sizeDataPoints \* sizeof(GLfloat), dataPoints.getCoordinatesData(), GL\_STATIC\_DRAW);

GLuint colors\_vbo = 0;

glGenBuffers(1, &colors\_vbo);

glBindBuffer(GL\_ARRAY\_BUFFER, colors\_vbo);

glBufferData(GL\_ARRAY\_BUFFER, dataPoints.sizeDataPoints \* sizeof(GLfloat),dataPoints.getColorsData(), GL\_STATIC\_DRAW);

GLuint elementbuffer;

glGenBuffers(1, &elementbuffer);

glBindBuffer(GL\_ELEMENT\_ARRAY\_BUFFER, elementbuffer);

glBufferData(GL\_ELEMENT\_ARRAY\_BUFFER, indexes.size() \* sizeof(GLuint), indexes.data(), GL\_STATIC\_DRAW);

// end БУФЕРЫ

//Массив

GLuint vao = 0;

glGenVertexArrays(1, &vao);

glBindVertexArray(vao);

glBindBuffer(GL\_ARRAY\_BUFFER, coords\_vbo);

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

glBindBuffer(GL\_ARRAY\_BUFFER, colors\_vbo);

glVertexAttribPointer(1, 3, GL\_FLOAT, GL\_FALSE, 0, NULL);

glEnableVertexAttribArray(0);

glEnableVertexAttribArray(1);

glBindBuffer(GL\_ELEMENT\_ARRAY\_BUFFER, elementbuffer);

// end массивы

//Отрисовка

glDrawElements(type, indexes.size(), GL\_UNSIGNED\_INT, 0);

// end Отрисовка

//Освобождение памяти

glDeleteVertexArrays(1, &vao);

glDeleteBuffers(1, &coords\_vbo);

glDeleteBuffers(1, &colors\_vbo);

glDeleteBuffers(1, &elementbuffer);

// end Освобождение памяти

}

void task\_1\_a(GLint shader\_programme) {

std::vector<GLuint> indexes = {

0,1,2,

0,1,3,

0,1,4,

0,2,3,

0,2,4,

0,3,4,

1,2,3,

1,2,4,

1,3,4,

2,3,4,

};

dataPoint dataPoints;

dataPoints.addCoordinates({ 0.0f, 0.5f, 0.0f });

dataPoints.addCoordinates({ 0.5f, 0.0f, 0.0f });

dataPoints.addCoordinates({ 0.0f, -0.5f, 1.0f });

dataPoints.addCoordinates({ -0.5f, 0.0f, 0.0f });

dataPoints.addCoordinates({ -1.0f, 0.0f, 0.0f });

dataPoints.addColors({ 1.0f, 0.0f, 0.0f });

dataPoints.addColors({ 0.0f, 1.0f, 0.0f });

dataPoints.addColors({ 0.0f, 0.0f, 1.0f });

dataPoints.addColors({ 1.0f, 1.0f, 0.0f });

dataPoints.addColors({ 1.0f, 1.0f, 1.0f });

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE); // отрисовка только граний

glm::mat4 transformMatrix = glm::mat4(1.0f);

drawSmth(dataPoints, indexes, transformMatrix, shader\_programme, GL\_TRIANGLES);

}

void task\_1\_b(GLint shader\_programme) {

std::vector<GLuint> indexes = {

0,1,

0,2,

0,3,

0,4,

0,5,

1,2,

1,3,

1,4,

1,5,

2,3,

2,4,

2,5,

3,4,

3,5,

4,5

};

dataPoint dataPoints;

dataPoints.addCoordinates({ 0.0f, 0.5f, 0.0f });

dataPoints.addCoordinates({ 0.5f, 0.0f, 0.0f });

dataPoints.addCoordinates({ 0.0f, -0.5f, 1.0f });

dataPoints.addCoordinates({ -0.5f, 0.0f, 0.0f });

dataPoints.addCoordinates({ -1.0f, 0.0f, 0.0f });

dataPoints.addCoordinates({ 0.0f, 1.0f, 0.0f });

dataPoints.addColors({ 1.0f, 0.0f, 0.0f });

dataPoints.addColors({ 0.0f, 1.0f, 0.0f });

dataPoints.addColors({ 0.0f, 0.0f, 1.0f });

dataPoints.addColors({ 1.0f, 1.0f, 0.0f });

dataPoints.addColors({ 1.0f, 1.0f, 1.0f });

dataPoints.addColors({ 0.4f, 1.0f, 0.7f });

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE); // отрисовка только граний

glm::mat4 transformMatrix = glm::mat4(1.0f);

drawSmth(dataPoints, indexes, transformMatrix, shader\_programme, GL\_LINES);

}

void task\_2(GLint shader\_programme, bool trueOrFalse) {

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_FILL);

dataPoint dataPoints;

dataPoints.addCoordinates({ -0.4f,0.2f,0.0f });

dataPoints.addCoordinates({ -0.6f,-0.3f,0.0f });

dataPoints.addCoordinates({ -0.2f,-0.3f,0.0f });

dataPoints.addColors({ 1.0f,0.0f,0.0f });

dataPoints.addColors({ 0.0f,1.0f,0.0f });

dataPoints.addColors({ 0.0f,0.0f,1.0f });

std::vector<GLuint> indexes1 = { 0,1,2 };

glm::mat4 transformMatrix = glm::mat4(1.0f);

glm::vec3 vector\_P = { 0.03f,0.15f,0.0f };

glm::vec3 scale = { 1.2f,1.5f,0.0f };

if (trueOrFalse) {

//зд: треугольник следует смасштабировать с коэффициентами(kx, ky) и перенести на вектор p;

transformMatrix = glm::translate(transformMatrix, vector\_P); // перенос на вектор p

transformMatrix = glm::scale(transformMatrix, scale); // маштабирование

}

drawSmth(dataPoints, indexes1, transformMatrix, shader\_programme, GL\_TRIANGLES);

dataPoint dataPoints2;

dataPoints2.addCoordinates({ 0.0f,0.5f,0.0f });

dataPoints2.addCoordinates({ 0.0f,-0.5f,0.0f });

dataPoints2.addColors({ 1.0f,0.0f,0.0f });

dataPoints2.addColors({ 0.0f,1.0f,0.0f });

std::vector<GLuint> indexes2 = { 0,1 };

transformMatrix = glm::mat4(1.0f);

if (trueOrFalse) transformMatrix = glm::rotate(transformMatrix, glm::radians(40.0f), glm::vec3(0.0, 0.0, 1.0)); //поворачивает 3D вектор на заданный угол вокруг заданной оси(представленной орт - вектором) :

drawSmth(dataPoints2,indexes2, transformMatrix, shader\_programme, GL\_LINES);

dataPoint dataPoints3;

dataPoints3.addCoordinates({ 0.4f,0.5f,0.0f });

dataPoints3.addCoordinates({ 0.7f,0.5f,0.0f });

dataPoints3.addCoordinates({ 0.4f,-0.5f,0.0f });

dataPoints3.addCoordinates({ 0.7f,-0.5f,0.0f });

dataPoints3.addColors({ 0.1f,0.1f,0.9f });

dataPoints3.addColors({ 0.7f,0.1f,0.0f });

dataPoints3.addColors({ 0.7f,0.1f,0.0f });

dataPoints3.addColors({ 0.1f,0.1f,0.9f });

std::vector<GLuint> indexes3 = { 0,1,2,2,1,3 };

transformMatrix = glm::mat4(1.0f);

if (trueOrFalse) transformMatrix = glm::rotate(transformMatrix, glm::radians(-25.0f), glm::vec3(-0.3, -0.8, 1.0)); //поворачивает 3D вектор на заданный угол вокруг заданной оси(представленной орт - вектором) :

drawSmth(dataPoints3, indexes3, transformMatrix, shader\_programme, GL\_TRIANGLES);

}

void task\_3(GLint shader\_programme) {

dataPoint dataPoints;

dataPoints.addCoordinates({ 0.0f,0.0f,0.0f });

dataPoints.addCoordinates({ 0.0f,0.5f,0.0f });

dataPoints.addColors({ 1.0f,0.0f,0.0f });

dataPoints.addColors({ 0.0f,1.0f,0.0f });

std::vector<GLuint> indexes = { 0,1 };

glm::mat4 transformMatrix = glm::mat4(1.0f);

drawSmth(dataPoints, indexes, transformMatrix, shader\_programme, GL\_LINES);

transformMatrix = glm::rotate(transformMatrix, glm::radians(45.0f), glm::vec3(0.0, 0.0, 1.0)); //поворачивает 3D вектор на заданный угол вокруг заданной оси(представленной орт - вектором) :

drawSmth(dataPoints, indexes, transformMatrix, shader\_programme, GL\_LINES);

transformMatrix = glm::rotate(transformMatrix, glm::radians(90.0f), glm::vec3(0.0, 0.0, 1.0)); //поворачивает 3D вектор на заданный угол вокруг заданной оси(представленной орт - вектором) :

drawSmth(dataPoints, indexes, transformMatrix, shader\_programme, GL\_LINES);

transformMatrix = glm::rotate(transformMatrix, glm::radians(45.0f), glm::vec3(0.0, 0.0, 1.0)); //поворачивает 3D вектор на заданный угол вокруг заданной оси(представленной орт - вектором) :

drawSmth(dataPoints, indexes, transformMatrix, shader\_programme, GL\_LINES);

transformMatrix = glm::rotate(transformMatrix, glm::radians(45.0f), glm::vec3(0.0, 0.0, 1.0)); //поворачивает 3D вектор на заданный угол вокруг заданной оси(представленной орт - вектором) :

drawSmth(dataPoints, indexes, transformMatrix, shader\_programme, GL\_LINES);

transformMatrix = glm::rotate(transformMatrix, glm::radians(90.0f), glm::vec3(0.0, 0.0, 1.0)); //поворачивает 3D вектор на заданный угол вокруг заданной оси(представленной орт - вектором) :

drawSmth(dataPoints, indexes, transformMatrix, shader\_programme, GL\_LINES);

}

Shaderfuncs.h:

#pragma once

#include <iostream>

#include <string>

#include <GL/glew.h>

#include <GLFW/glfw3.h>

GLuint LoadShader(std::string filename, GLenum type);

Shaderfuncs.cpp:

#include "ShaderFuncs.h"

#include <fstream>

#include <vector>

GLuint LoadShader(std::string filename, GLenum type)

{

std::ifstream fin;

fin.open(filename);

if (fin) {

std::string textshader;

char c= fin.get();

while (c!=-1) {

textshader += c;

c = fin.get();

}

fin.close();

GLuint s = glCreateShader(type);

char\* buf = new GLchar[textshader.length() + 1];

strcpy\_s(buf, textshader.length()+1 , textshader.c\_str());

glShaderSource(s, 1, &buf, NULL);

glCompileShader(s);

GLint isCompiled = 0;

glGetShaderiv(s, GL\_COMPILE\_STATUS, &isCompiled);

if (isCompiled == GL\_FALSE)

{

GLint maxLength = 0;

glGetShaderiv(s, GL\_INFO\_LOG\_LENGTH, &maxLength);

std::vector<GLchar> errorLog(maxLength);

glGetShaderInfoLog(s, maxLength, &maxLength, &errorLog[0]);

for (GLchar c : errorLog)

std::cout << c;

}

delete[] buf;

return s;

}

else {

std::cout << "ERROR: No such file";

return -1;

}

}

FragmentShader.glsl:

#version 400

in vec3 colour;

out vec4 frag\_colour;

void main () {

frag\_colour = vec4 (colour, 1.0);

}

VertexShaner.glsl:

#version 400

in vec3 vertex\_position;

in vec3 vertex\_colour;

uniform mat4 transform;

out vec3 colour;

void main () {

colour = vertex\_colour;

gl\_Position = transform \* vec4 (vertex\_position, 1.0);

}