Національний технічний університет України «КПІ ім. Ігоря Сікорського»

Факультет Інформатики та Обчислювальної Техніки

Кафедра Автоматизованих Систем Обробки Інформації та Управління

**Лабораторна робота №3**

**з дисципліни "Комп’ютерна графіка та мультимедія"**

на тему

«**Основи роботи з OpenGL**»

Виконав:

студент гр. ІС-91

Хмелiнiн Андрiй

Перевірив:

Старший викладач Родіонов П. Ю.

Київ – 2019

**Завдання 1**

**Запустіть приклади програм, що описано в даній лабораторній роботі.**

#include "C:\Users\AdmiN\source\repos\khmelinin\ComputerGraphic\Lab3\include\glad/glad.h"

#include <C:\Users\AdmiN\Desktop\glfw-3.3.2\include\GLFW/glfw3.h>

#include <iostream>

#include <stdlib.h>

#include <stdio.h>

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

void processInput(GLFWwindow\* window);

// settings

const unsigned int SCR\_WIDTH = 800;

const unsigned int SCR\_HEIGHT = 600;

int main()

{

// glfw: initialize and configure

// ------------------------------

glfwInit();

glfwWindowHint(GLFW\_CONTEXT\_VERSION\_MAJOR, 3);

glfwWindowHint(GLFW\_CONTEXT\_VERSION\_MINOR, 3);

glfwWindowHint(GLFW\_OPENGL\_PROFILE,

GLFW\_OPENGL\_CORE\_PROFILE);

#ifdef \_\_APPLE\_\_

glfwWindowHint(GLFW\_OPENGL\_FORWARD\_COMPAT, GL\_TRUE); //

uncomment this statement to fix compilation on OS X

#endif

// glfw window creation

// --------------------

GLFWwindow\* window = glfwCreateWindow(SCR\_WIDTH, SCR\_HEIGHT,

"LearnOpenGL", NULL, NULL);

if (window == NULL)

{

std::cout << "Failed to create GLFW window" << std::endl;

glfwTerminate();

return -1;

}

glfwMakeContextCurrent(window);

glfwSetFramebufferSizeCallback(window, framebuffer\_size\_callback);

// glad: load all OpenGL function pointers

// ---------------------------------------

if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress))

{

std::cout << "Failed to initialize GLAD" << std::endl;

return -1;

}

// render loop

// -----------

while (!glfwWindowShouldClose(window))

{

// input

// -----

processInput(window);

// render

// ------

glClearColor(0.2f, 0.3f, 0.3f, 1.0f);

glClear(GL\_COLOR\_BUFFER\_BIT);

// glfw: swap buffers and poll IO events (keys pressed/released, mouse moved etc.)

// -------------------------------------------------------------------------------

glfwSwapBuffers(window);

glfwPollEvents();

}

// glfw: terminate, clearing all previously allocated GLFW resources.

// ------------------------------------------------------------------

glfwTerminate();

return 0;

}

// process all input: query GLFW whether relevant keys are pressed/released this frameand react accordingly

// ----------------------------------------------------------------------------------------------------

void processInput(GLFWwindow \* window)

{

if (glfwGetKey(window, GLFW\_KEY\_ESCAPE) == GLFW\_PRESS)

glfwSetWindowShouldClose(window, true);

}

// glfw: whenever the window size changed (by OS or user resize) this callback function executes

// ---------------------------------------------------------------------------------------------

void framebuffer\_size\_callback(GLFWwindow\* window, int width, int height)

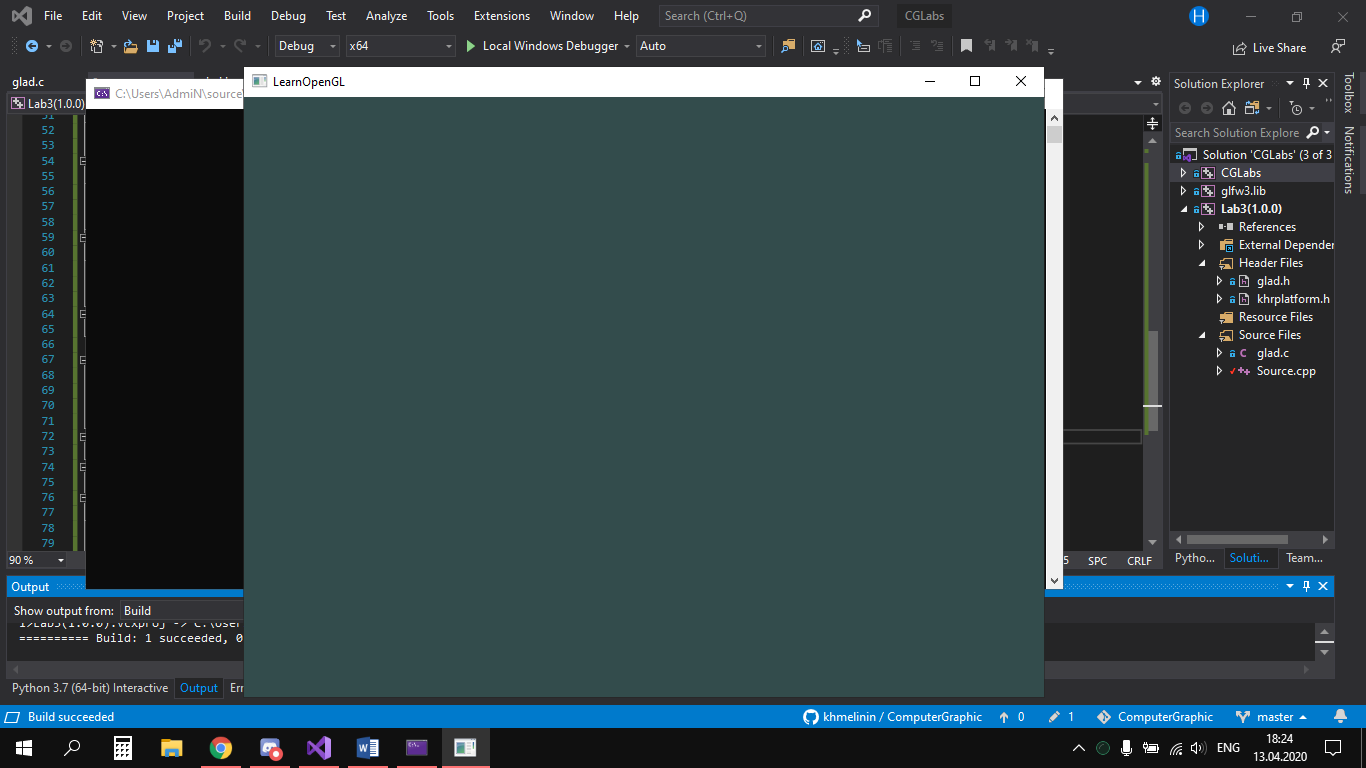
{

// make sure the viewport matches the new window dimensions; note that width and

// height will be significantly larger than specified on retina displays.

glViewport(0, 0, width, height);

}

****

#include "C:\Users\AdmiN\source\repos\khmelinin\ComputerGraphic\Lab3\include\glad/glad.h"

#include <C:\Users\AdmiN\Desktop\glfw-3.3.2\include\GLFW/glfw3.h>

#include <iostream>

#include <stdlib.h>

#include <stdio.h>

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

void processInput(GLFWwindow\* window);

// settings

const unsigned int SCR\_WIDTH = 800;

const unsigned int SCR\_HEIGHT = 600;

const char\* vertexShaderSource = "#version 330 core\n"

"layout (location = 0) in vec3 aPos;\n"

"void main()\n"

"{\n"

" gl\_Position = vec4(aPos.x, aPos.y, aPos.z, 1.0);\n"

"}\0";

const char\* fragmentShaderSource = "#version 330 core\n"

"out vec4 FragColor;\n"

"void main()\n"

"{\n"

" FragColor = vec4(1.0f, 0.5f, 0.2f, 1.0f);\n"

"}\n\0";

int main()

{

// glfw: initialize and configure

// ------------------------------

glfwInit();

glfwWindowHint(GLFW\_CONTEXT\_VERSION\_MAJOR, 3);

glfwWindowHint(GLFW\_CONTEXT\_VERSION\_MINOR, 3);

glfwWindowHint(GLFW\_OPENGL\_PROFILE,

GLFW\_OPENGL\_CORE\_PROFILE);

#ifdef APPLE

glfwWindowHint(GLFW\_OPENGL\_FORWARD\_COMPAT, GL\_TRUE); //

uncomment this statement to fix compilation on OS X

#endif

// glfw window creation

// --------------------

GLFWwindow\* window = glfwCreateWindow(SCR\_WIDTH, SCR\_HEIGHT,

"LearnOpenGL", NULL, NULL);

if (window == NULL)

{

std::cout << "Failed to create GLFW window" << std::endl;

glfwTerminate();

return -1;

}

glfwMakeContextCurrent(window);

glfwSetFramebufferSizeCallback(window, framebuffer\_size\_callback);

// glad: load all OpenGL function pointers

// ---------------------------------------

if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress))

{

std::cout << "Failed to initialize GLAD" << std::endl;

return -1;

}

// build and compile our shader program

// ------------------------------------

// vertex shader

int vertexShader = glCreateShader(GL\_VERTEX\_SHADER);

glShaderSource(vertexShader, 1, &vertexShaderSource, NULL);

glCompileShader(vertexShader);

// check for shader compile errors

int success;

char infoLog[512];

glGetShaderiv(vertexShader, GL\_COMPILE\_STATUS, &success);

if (!success)

{

glGetShaderInfoLog(vertexShader, 512, NULL, infoLog);

std::cout << "ERROR::SHADER::VERTEX::COMPILATION\_FAILED\n" <<

infoLog << std::endl;

}

// fragment shader

int fragmentShader = glCreateShader(GL\_FRAGMENT\_SHADER);

glShaderSource(fragmentShader, 1, &fragmentShaderSource, NULL);

glCompileShader(fragmentShader);

// check for shader compile errors

glGetShaderiv(fragmentShader, GL\_COMPILE\_STATUS, &success);

if (!success)

{

glGetShaderInfoLog(fragmentShader, 512, NULL, infoLog);

std::cout << "ERROR::SHADER::FRAGMENT::COMPILATION\_FAILED\n"

<< infoLog << std::endl;

}

// link shaders

int shaderProgram = glCreateProgram();

glAttachShader(shaderProgram, vertexShader);

glAttachShader(shaderProgram, fragmentShader);

glLinkProgram(shaderProgram);

// check for linking errors

glGetProgramiv(shaderProgram, GL\_LINK\_STATUS, &success);

if (!success) {

glGetProgramInfoLog(shaderProgram, 512, NULL, infoLog);

std::cout << "ERROR::SHADER::PROGRAM::LINKING\_FAILED\n" <<

infoLog << std::endl;

}

glDeleteShader(vertexShader);

glDeleteShader(fragmentShader);

// set up vertex data (and buffer(s)) and configure vertex attributes

// ------------------------------------------------------------------

float vertices[] = {

0.5f, 0.5f, 0.0f, // top right

0.5f, -0.5f, 0.0f, // bottom right

-0.5f, -0.5f, 0.0f, // bottom left

-0.5f, 0.5f, 0.0f // top left

};

unsigned int indices[] = { // note that we start from 0!

0, 1, 3, // first Triangle

1, 2, 3 // second Triangle

};

unsigned int VBO, VAO, EBO;

glGenVertexArrays(1, &VAO);

glGenBuffers(1, &VBO);

glGenBuffers(1, &EBO);

// bind the Vertex Array Object first, then bind and set vertex buffer(s), and then

//configure vertex attributes(s).

glBindVertexArray(VAO);

glBindBuffer(GL\_ARRAY\_BUFFER, VBO);

glBufferData(GL\_ARRAY\_BUFFER, sizeof(vertices), vertices,

GL\_STATIC\_DRAW);

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

glBufferData(GL\_ELEMENT\_ARRAY\_BUFFER, sizeof(indices), indices,

GL\_STATIC\_DRAW);

glVertexAttribPointer(0, 3, GL\_FLOAT, GL\_FALSE, 3 \* sizeof(float), (void\*)0);

glEnableVertexAttribArray(0);

// note that this is allowed, the call to glVertexAttribPointer registered VBO as the

// vertex attribute's bound vertex buffer object so afterwards we can safely unbind

glBindBuffer(GL\_ARRAY\_BUFFER, 0);

// remember: do NOT unbind the EBO while a VAO is active as the bound element

//buffer object IS stored in the VAO; keep the EBO bound.

//glBindBuffer(GL\_ELEMENT\_ARRAY\_BUFFER, 0);

// You can unbind the VAO afterwards so other VAO calls won't accidentally

//modify this VAO, but this rarely happens.Modifying other

// VAOs requires a call to glBindVertexArray anyways so we generally don't

//unbind VAOs(nor VBOs) when it's not directly necessary.

glBindVertexArray(0);

// uncomment this call to draw in wireframe polygons.

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

// render loop

// -----------

while (!glfwWindowShouldClose(window))

{

// input

// -----

processInput(window);

// render

// ------

glClearColor(0.2f, 0.3f, 0.3f, 1.0f);

glClear(GL\_COLOR\_BUFFER\_BIT);

// draw our first triangle

glUseProgram(shaderProgram);

glBindVertexArray(VAO); // seeing as we only have a single VAO there's no

//need to bind it every time, but we'll do so to keep things a bit more organized

//glDrawArrays(GL\_TRIANGLES, 0, 6);

glDrawElements(GL\_TRIANGLES, 6, GL\_UNSIGNED\_INT, 0);

// glBindVertexArray(0); // no need to unbind it every time

// glfw: swap buffers and poll IO events (keys pressed/released, mouse moved

//etc.)

// -------------------------------------------------------------------------------

glfwSwapBuffers(window);

glfwPollEvents();

}

// optional: de-allocate all resources once they've outlived their purpose:

// ------------------------------------------------------------------------

glDeleteVertexArrays(1, &VAO);

glDeleteBuffers(1, &VBO);

glDeleteBuffers(1, &EBO);

// glfw: terminate, clearing all previously allocated GLFW resources.

// ------------------------------------------------------------------

glfwTerminate();

return 0;

}

// process all input: query GLFW whether relevant keys are pressed/released this

//frameand react accordingly

void processInput(GLFWwindow\* window)

{

if (glfwGetKey(window, GLFW\_KEY\_ESCAPE) == GLFW\_PRESS)

glfwSetWindowShouldClose(window, true);

}

// glfw: whenever the window size changed (by OS or user resize) this callback

//function executes

void framebuffer\_size\_callback(GLFWwindow\* window, int width, int height)

{

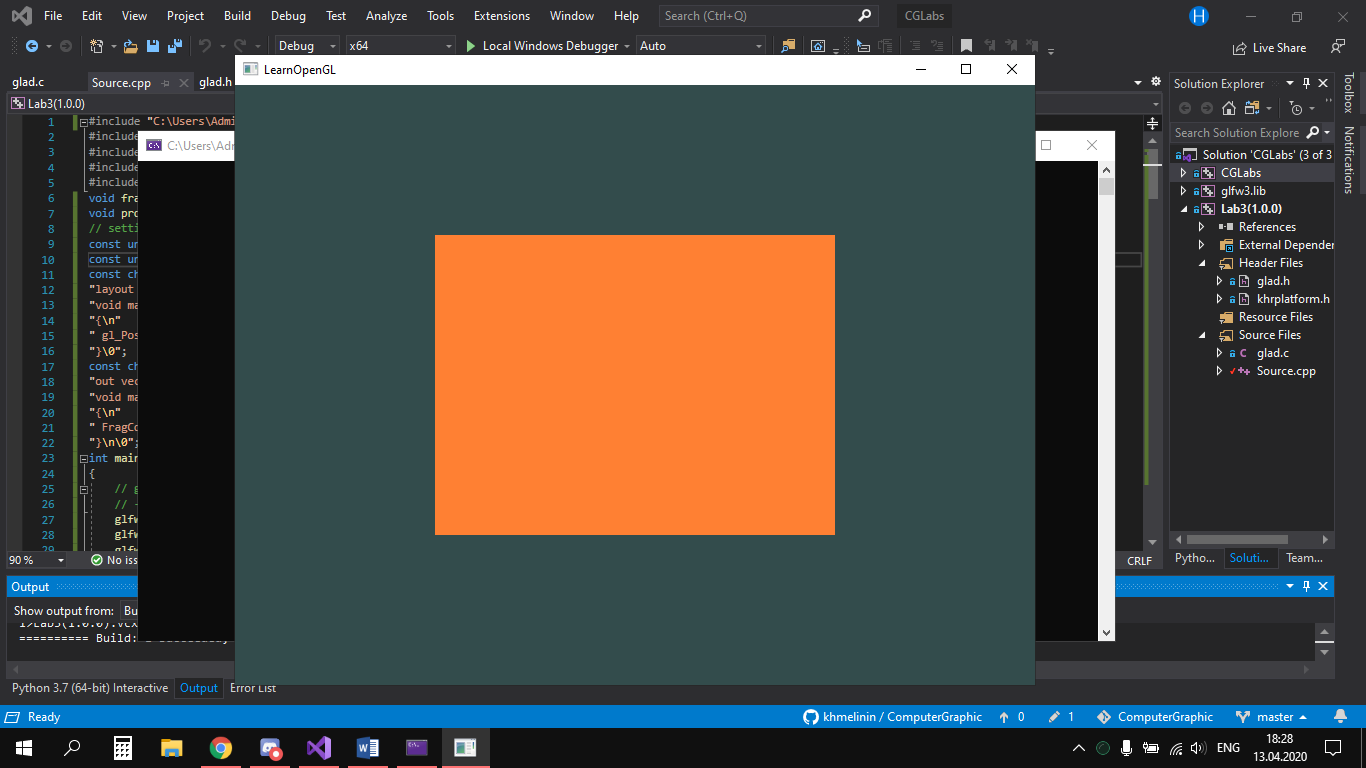
// make sure the viewport matches the new window dimensions; note that width

//and

// height will be significantly larger than specified on retina displays.

glViewport(0, 0, width, height);

}

****

//#include "C:\Users\AdmiN\source\repos\khmelinin\ComputerGraphic\Lab3\include\glad/glad.h"

#include <C:\Users\AdmiN\Desktop\glfw-3.3.2\include\GLFW/glfw3.h>

//#include <iostream>

#include <stdlib.h>

#include <stdio.h>

static void error\_callback(int error, const char\* description)

{

fputs(description, stderr);

}

static void key\_callback(GLFWwindow\* window, int key, int scancode, int action,

int mods)

{

if (key == GLFW\_KEY\_ESCAPE && action == GLFW\_PRESS)

glfwSetWindowShouldClose(window, GL\_TRUE);

}

int main(void)

{

GLFWwindow\* window;

glfwSetErrorCallback(error\_callback);

if (!glfwInit())

exit(EXIT\_FAILURE);

window = glfwCreateWindow(640, 480, "Simple example", NULL, NULL);

if (!window)

{

glfwTerminate();

exit(EXIT\_FAILURE);

}

glfwMakeContextCurrent(window);

glfwSetKeyCallback(window, key\_callback);

while (!glfwWindowShouldClose(window))

{

float ratio;

int width, height;

glfwGetFramebufferSize(window, &width, &height);

ratio = width / (float)height;

glViewport(0, 0, width, height);

glClear(GL\_COLOR\_BUFFER\_BIT);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(-ratio, ratio, -1.f, 1.f, 1.f, -1.f);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glRotatef((float)glfwGetTime() \* 50.f, 0.f, 0.f, 1.f);

glBegin(GL\_TRIANGLES);

glColor3f(1.f, 0.f, 0.f);

glVertex3f(-0.6f, -0.4f, 0.f);

glColor3f(0.f, 1.f, 0.f);

glVertex3f(0.6f, -0.4f, 0.f);

glColor3f(0.f, 0.f, 1.f);

glVertex3f(0.f, 0.6f, 0.f);

glEnd();

glfwSwapBuffers(window);

glfwPollEvents();

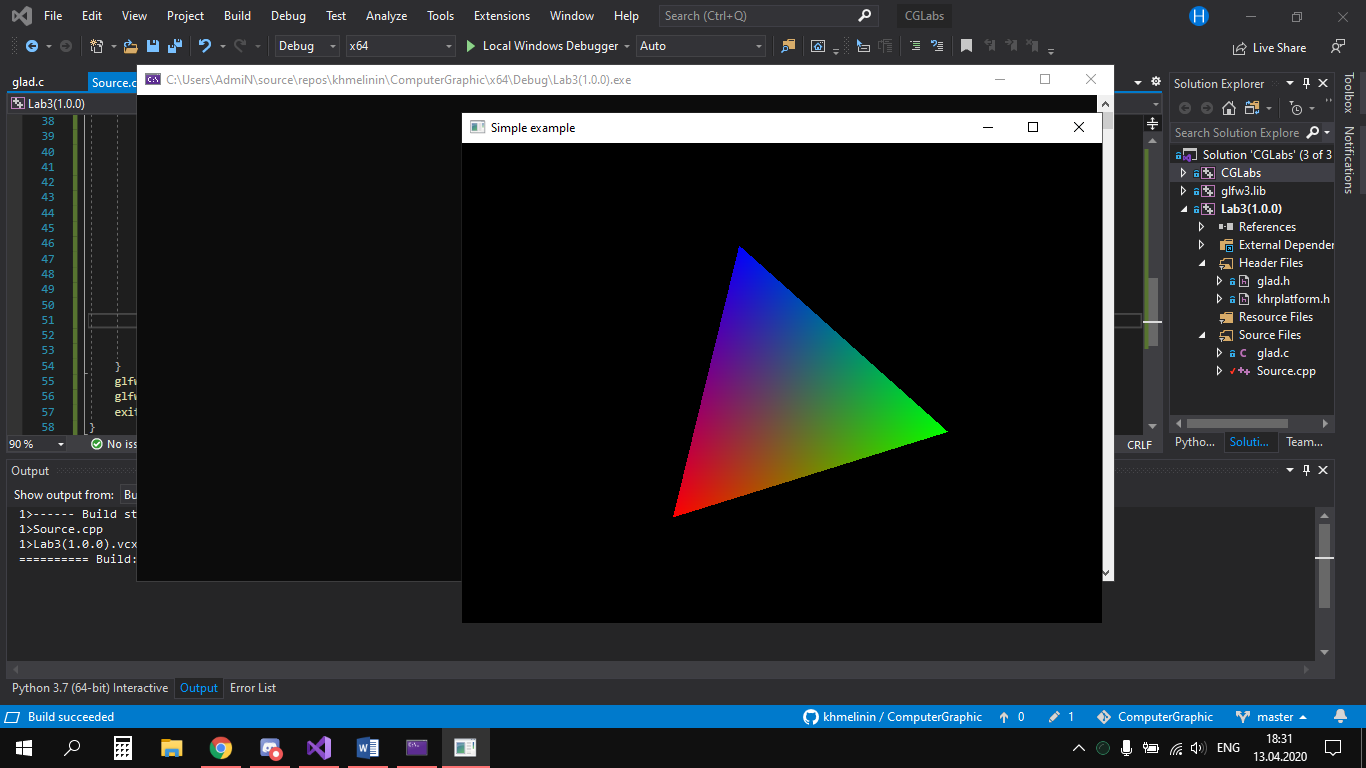
}

glfwDestroyWindow(window);

glfwTerminate();

exit(EXIT\_SUCCESS);

}

****

**Завдання 2**

**Створіть два трикутника, розмістивши один за іншим, використовуючи**

**glDrawArrays, додаючи більшу кількість вершин. Створіть два трикутника за**

**допомогою різних VAO та VBO.**

#include "C:\Users\AdmiN\source\repos\khmelinin\ComputerGraphic\Lab3\include\glad/glad.h"

#include <C:\Users\AdmiN\Desktop\glfw-3.3.2\include\GLFW/glfw3.h>

#include <iostream>

#include <stdlib.h>

#include <stdio.h>

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

void processInput(GLFWwindow\* window);

const unsigned int SCR\_WIDTH = 800;

const unsigned int SCR\_HEIGHT = 600;

const char\* vertexShaderSource = "#version 330 core\n"

"layout (location = 0) in vec3 aPos;\n"

"void main()\n"

"{\n"

" gl\_Position = vec4(aPos.x, aPos.y, aPos.z, 1.0);\n"

"}\0";

const char\* fragmentShaderSource = "#version 330 core\n"

"out vec4 FragColor;\n"

"void main()\n"

"{\n"

" FragColor = vec4(1.0f, 0.5f, 0.2f, 1.0f);\n"

"}\n\0";

int main()

{

glfwInit();

glfwWindowHint(GLFW\_CONTEXT\_VERSION\_MAJOR, 3);

glfwWindowHint(GLFW\_CONTEXT\_VERSION\_MINOR, 3);

glfwWindowHint(GLFW\_OPENGL\_PROFILE, GLFW\_OPENGL\_CORE\_PROFILE);

#ifdef \_\_APPLE\_\_

glfwWindowHint(GLFW\_OPENGL\_FORWARD\_COMPAT, GL\_TRUE);

#endif

GLFWwindow\* window = glfwCreateWindow(SCR\_WIDTH, SCR\_HEIGHT, "LearnOpenGL", NULL, NULL);

if (window == NULL)

{

std::cout << "Failed to create GLFW window" << std::endl;

glfwTerminate();

return -1;

}

glfwMakeContextCurrent(window);

glfwSetFramebufferSizeCallback(window, framebuffer\_size\_callback);

if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress))

{

std::cout << "Failed to initialize GLAD" << std::endl;

return -1;

}

int vertexShader = glCreateShader(GL\_VERTEX\_SHADER);

glShaderSource(vertexShader, 1, &vertexShaderSource, NULL);

glCompileShader(vertexShader);

int success;

char infoLog[512];

glGetShaderiv(vertexShader, GL\_COMPILE\_STATUS, &success);

if (!success)

{

glGetShaderInfoLog(vertexShader, 512, NULL, infoLog);

std::cout << "ERROR::SHADER::VERTEX::COMPILATION\_FAILED\n" << infoLog << std::endl;

}

int fragmentShader = glCreateShader(GL\_FRAGMENT\_SHADER);

glShaderSource(fragmentShader, 1, &fragmentShaderSource, NULL);

glCompileShader(fragmentShader);

glGetShaderiv(fragmentShader, GL\_COMPILE\_STATUS, &success);

if (!success)

{

glGetShaderInfoLog(fragmentShader, 512, NULL, infoLog);

std::cout << "ERROR::SHADER::FRAGMENT::COMPILATION\_FAILED\n" << infoLog << std::endl;

}

int shaderProgram = glCreateProgram();

glAttachShader(shaderProgram, vertexShader);

glAttachShader(shaderProgram, fragmentShader);

glLinkProgram(shaderProgram);

glGetProgramiv(shaderProgram, GL\_LINK\_STATUS, &success);

if (!success) {

glGetProgramInfoLog(shaderProgram, 512, NULL, infoLog);

std::cout << "ERROR::SHADER::PROGRAM::LINKING\_FAILED\n" << infoLog << std::endl;

}

glDeleteShader(vertexShader);

glDeleteShader(fragmentShader);

float vertices[] = {

-0.9f, -0.5f, 0.0f,

-0.0f, -0.5f, 0.0f,

-0.45f, 0.5f, 0.0f,

0.0f, -0.5f, 0.0f,

0.9f, -0.5f, 0.0f,

0.45f, 0.5f, 0.0f

};

unsigned int VBO, VAO;

glGenVertexArrays(1, &VAO);

glGenBuffers(1, &VBO);

glBindVertexArray(VAO);

glBindBuffer(GL\_ARRAY\_BUFFER, VBO);

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

glVertexAttribPointer(0, 3, GL\_FLOAT, GL\_FALSE, 3 \* sizeof(float), (void\*)0);

glEnableVertexAttribArray(0);

glBindBuffer(GL\_ARRAY\_BUFFER, 0);

glBindVertexArray(0);

while (!glfwWindowShouldClose(window))

{

processInput(window);

glClearColor(0.2f, 0.3f, 0.3f, 1.0f);

glClear(GL\_COLOR\_BUFFER\_BIT);

glUseProgram(shaderProgram);

glBindVertexArray(VAO);

glDrawArrays(GL\_TRIANGLES, 0, 6);

glfwSwapBuffers(window);

glfwPollEvents();

}

glDeleteVertexArrays(1, &VAO);

glDeleteBuffers(1, &VBO);

glfwTerminate();

return 0;

}

void processInput(GLFWwindow\* window)

{

if (glfwGetKey(window, GLFW\_KEY\_ESCAPE) == GLFW\_PRESS)

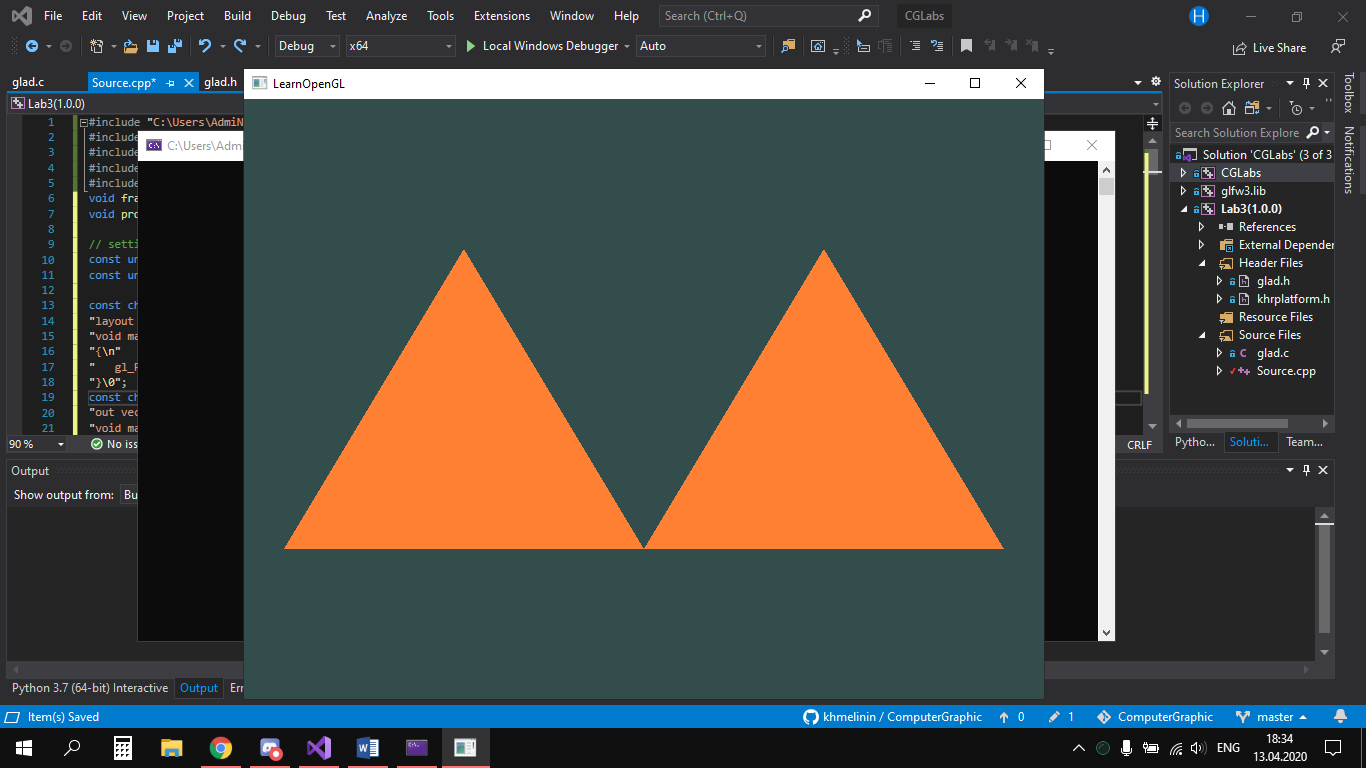
glfwSetWindowShouldClose(window, true);

}

void framebuffer\_size\_callback(GLFWwindow\* window, int width, int height)

{

glViewport(0, 0, width, height);

}****

**Завдання 3**

**Створіть фрагментний шейдер, що виводить будь-який колір. Нехай**

**другий трикутник буде такого ж самого кольору.**

#include "C:\Users\AdmiN\source\repos\khmelinin\ComputerGraphic\Lab3\include\glad/glad.h"

#include <C:\Users\AdmiN\Desktop\glfw-3.3.2\include\GLFW/glfw3.h>

#include <iostream>

#include <stdlib.h>

#include <stdio.h>

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

void processInput(GLFWwindow\* window);

const unsigned int SCR\_WIDTH = 800;

const unsigned int SCR\_HEIGHT = 600;

const char\* vertexShaderSource = "#version 330 core\n"

"layout (location = 0) in vec3 aPos;\n"

"void main()\n"

"{\n"

" gl\_Position = vec4(aPos.x, aPos.y, aPos.z, 1.0);\n"

"}\0";

const char\* fragmentShader1Source = "#version 330 core\n"

"out vec4 FragColor;\n"

"void main()\n"

"{\n"

" FragColor = vec4(1.0f, 0.5f, 0.2f, 1.0f);\n"

"}\n\0";

const char\* fragmentShader2Source = "#version 330 core\n"

"out vec4 FragColor;\n"

"void main()\n"

"{\n"

" FragColor = vec4(0.0f, 0.0f, 0.0f, 1.0f);\n" //---------------------------- color change -----------------

"}\n\0";

int main()

{

glfwInit();

glfwWindowHint(GLFW\_CONTEXT\_VERSION\_MAJOR, 3);

glfwWindowHint(GLFW\_CONTEXT\_VERSION\_MINOR, 3);

glfwWindowHint(GLFW\_OPENGL\_PROFILE, GLFW\_OPENGL\_CORE\_PROFILE);

#ifdef \_\_APPLE\_\_

glfwWindowHint(GLFW\_OPENGL\_FORWARD\_COMPAT, GL\_TRUE);

#endif

GLFWwindow\* window = glfwCreateWindow(SCR\_WIDTH, SCR\_HEIGHT, "LearnOpenGL", NULL, NULL);

if (window == NULL)

{

std::cout << "Failed to create GLFW window" << std::endl;

glfwTerminate();

return -1;

}

glfwMakeContextCurrent(window);

glfwSetFramebufferSizeCallback(window, framebuffer\_size\_callback);

if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress))

{

std::cout << "Failed to initialize GLAD" << std::endl;

return -1;

}

unsigned int vertexShader = glCreateShader(GL\_VERTEX\_SHADER);

unsigned int fragmentShaderOrange = glCreateShader(GL\_FRAGMENT\_SHADER);

unsigned int fragmentShaderYellow = glCreateShader(GL\_FRAGMENT\_SHADER);

unsigned int shaderProgramOrange = glCreateProgram();

unsigned int shaderProgramYellow = glCreateProgram();

glShaderSource(vertexShader, 1, &vertexShaderSource, NULL);

glCompileShader(vertexShader);

glShaderSource(fragmentShaderOrange, 1, &fragmentShader1Source, NULL);

glCompileShader(fragmentShaderOrange);

glShaderSource(fragmentShaderYellow, 1, &fragmentShader2Source, NULL);

glCompileShader(fragmentShaderYellow);

glAttachShader(shaderProgramOrange, vertexShader);

glAttachShader(shaderProgramOrange, fragmentShaderOrange);

glLinkProgram(shaderProgramOrange);

glAttachShader(shaderProgramYellow, vertexShader);

glAttachShader(shaderProgramYellow, fragmentShaderYellow);

glLinkProgram(shaderProgramYellow);

float firstTriangle[] = {

-0.9f, -0.5f, 0.0f,

-0.0f, -0.5f, 0.0f,

-0.45f, 0.5f, 0.0f,

};

float secondTriangle[] = {

0.0f, -0.5f, 0.0f,

0.9f, -0.5f, 0.0f,

0.45f, 0.5f, 0.0f

};

unsigned int VBOs[2], VAOs[2];

glGenVertexArrays(2, VAOs);

glGenBuffers(2, VBOs);

glBindVertexArray(VAOs[0]);

glBindBuffer(GL\_ARRAY\_BUFFER, VBOs[0]);

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

glVertexAttribPointer(0, 3, GL\_FLOAT, GL\_FALSE, 3 \* sizeof(float), (void\*)0);

glEnableVertexAttribArray(0);

glBindVertexArray(VAOs[1]);

glBindBuffer(GL\_ARRAY\_BUFFER, VBOs[1]);

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

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

glEnableVertexAttribArray(0);

while (!glfwWindowShouldClose(window))

{

processInput(window);

glClearColor(0.2f, 0.3f, 0.3f, 1.0f);

glClear(GL\_COLOR\_BUFFER\_BIT);

glUseProgram(shaderProgramOrange);

glBindVertexArray(VAOs[0]);

glDrawArrays(GL\_TRIANGLES, 0, 3);

glUseProgram(shaderProgramYellow);

glBindVertexArray(VAOs[1]);

glDrawArrays(GL\_TRIANGLES, 0, 3);

glfwSwapBuffers(window);

glfwPollEvents();

}

glDeleteVertexArrays(2, VAOs);

glDeleteBuffers(2, VBOs);

glfwTerminate();

return 0;

}

void processInput(GLFWwindow\* window)

{

if (glfwGetKey(window, GLFW\_KEY\_ESCAPE) == GLFW\_PRESS)

glfwSetWindowShouldClose(window, true);

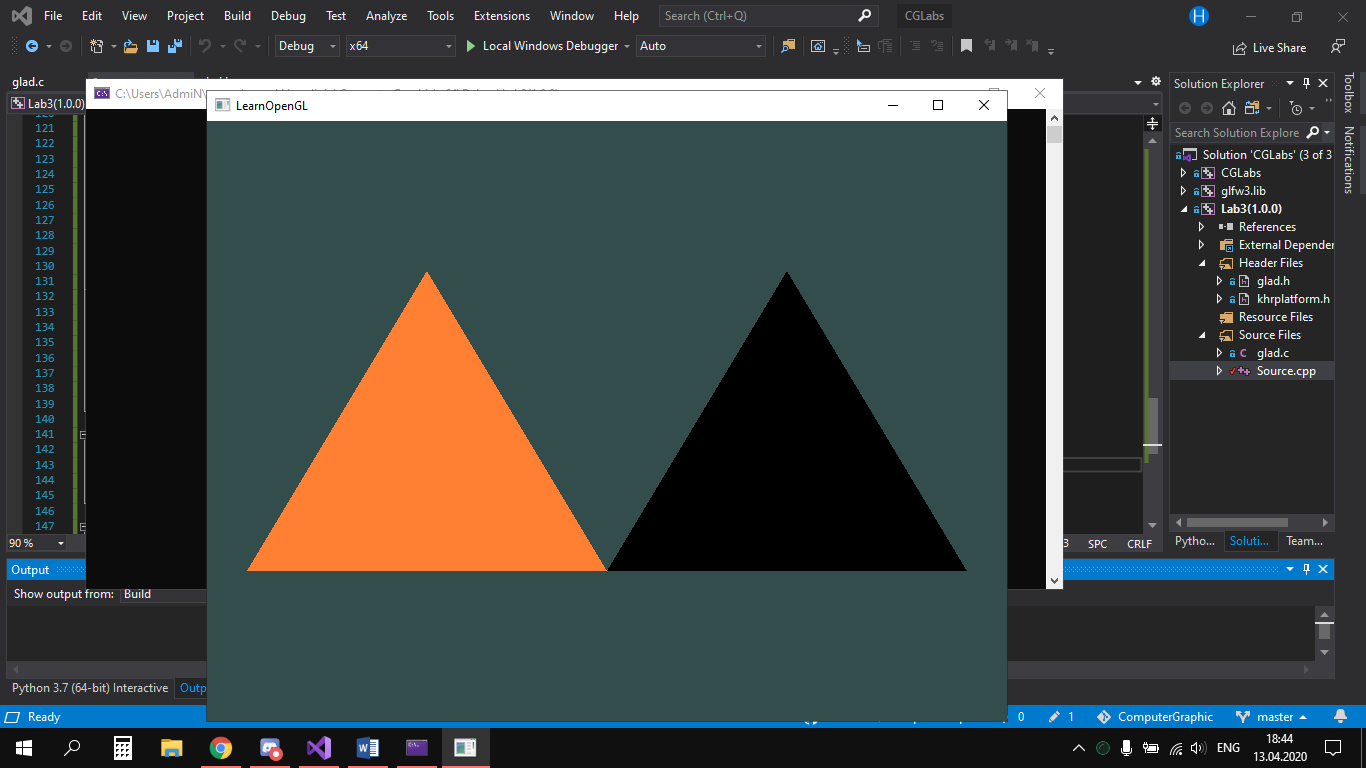
}

void framebuffer\_size\_callback(GLFWwindow\* window, int width, int height)

{

glViewport(0, 0, width, height);

}

****

**Висновки:**

У цій лабораторній роботі я навчився створювати рiзнi графiчнi форми i робити з них об'єкти та анiмацiї.