Міністерство освіти і науки України

Національний університет «Запорізька політехніка»

кафедра програмних засобів

ЗВІТ

з лабораторної роботи № 3

з дисципліни «Комп'ютерна графіка та обробка зображень» на тему:

«ПЕРЕМІЩЕННЯ, ОБЕРТАННЯ ТА МАСШТАБУВАННЯ»

Виконав:

ст. гр. КНТ-113сп Іван ЩЕДРОВСЬКИЙ

Прийняв:

Асистент Артем ТУЛЕНКОВ

2025

1 Мета роботи

Отримати практичні навички роботи з бібліотекою OpenGL, використовуючи мову програмування C++. Навчитися малювати 3D об’єкти, виконувати базові маніпуляції з ними, створювати вікна, керувати камерою, працювати з освітленням, створювати прості шейдери та зрозуміти як працюють бібіліотеки GLEW, GLFW, GLM.

2 Завдання до лабораторної роботи

2.1 Змініть колір фону та моделі.

2.2 Змініть фігуру на будь-яку іншу.

2.3 Виконайте декілька різних трансформацій з фігурою.

2.4 Наведіть коментарі до коду.

2.5 Внесіть індивідуальні зміни до коду.

3 Виконання лабораторної роботи

Для виконання лабораторної роботи було використано Visual Studio 17 2022

Лабораторна робота виконувалась, в тому числі, з взаємодією з книгою “Learn OpenGL – Graphics Programming” від Joey de Vries 2020 року, яку можна безкоштовно знайти на сайті learnopengl.com.

Для початку виконання роботи було встановлено бібліотеку OpenGL Mathematics, або ж GLM. Оскільки у мене є налаштована система inludes та libraries мені було достатньо додати header файли до папки з includes

Далі був створений новий vertex шейдер “transform”, який приймає uniform параметр transform та помножує його на vec4 координати точки для отримання нової позиції vertex. Код цього шейдеру показаний на рисунку 1.

```
#version 330 core
layout (location = 0) in vec3 aPos;

uniform mat4 transform;

void main() {
    gl_Position = transform * vec4(aPos, 1.0);
}
```

Рисунок 1 – Vertex шейдер з transform

Наступним етапом було якось сказати шейдеру значення цієї змінної. Для цього в класс Shader було додано метод setMat4, який встановлює змінну з типом glm::mat4. Інтерфейс та реалізація цього методу показані на рисунках 2 та 3 відповідно

```
    /*!
     * Utils for set `uniform` variables in shaders
     */
    void setVec2(const std::string& name, float x, float y) const;
    void setMat4(const std::string& name, glm::mat4 matrix) const;
private:
```

Рисунок 2 – Новий метод в класі

```
// TODO: Add GLM version?

void Shader::setVec2(const std::string& name, float x, float y) const {
    glUniform2f(glGetUniformLocation(Shader::ID, name.c_str()), x, y);
}

void Shader::setMat4(const std::string& name, glm::mat4 matrix) const {
    glUniformMatrix4fv(glGetUniformLocation(Shader::ID, name.c_str()), 1, GL_FALSE, glm::value_ptr(matrix));
}
```

Рисунок 3 – Реалізія методу для встановлення значення матриці

Після цього можна було вже працювати з трансформуванням об’єктів

Для створення деякої анімації повороту зображення код створення матриць був доданий безпосередньо до циклу рендера.

До всіх точок обох фігур були застосовані операції scale, rotation та translate.

Порядок операцій відіграє велику роль, оскільки множення матриць не є комутативним, тобто від зміни порядку змінюється результат. А також порядок матриць читається з права на ліво, або ж з кінця до початку

При виконані роботі матриці було застосовані в порядку в коді: translate, rotation та scale, що в реальності буде рівно навпаки. Приклад коду таких змін показаний на рисунку 4

```
glm::mat4 trans = glm::mat4(1.0f);

trans = glm::translate(trans, glm::vec3(0.5f, 0.2f, 0.0f));
trans = glm::rotate(trans, (float)glfwGetTime() * 0.9f, glm::vec3(0.0f, 1.0f, 0.0f));
trans = glm::rotate(trans, (float)glfwGetTime() * 0.6f, glm::vec3(1.0f, 0.0f, 0.0f));
trans = glm::rotate(trans, (float)glfwGetTime() * 0.8f, glm::vec3(0.0f, 0.0f, 1.0f));
trans = glm::scale(trans, glm::vec3(0.5f, 0.75f, 0.2f));

greenShader.use();
greenShader.setVec2("u_resolution", WIDTH, HEIGHT);
greenShader.setMat4("transform", trans);
glBindVertexArray(VAO1);
glDrawElements(GL_TRIANGLES, 3, GL_UNSIGNED_INT, (void*)(3 * sizeof(unsigned int)));

skyShader.use();
skyShader.setVec2("u_resolution", WIDTH, HEIGHT);
skyShader.setMat4("transform", trans);
glBindVertexArray(VAO2);
```

Рисунок 4 – Використання матриць

Приклади запуску програми в різний час без translate показані на рисунку 5 та 6. Приклад програми з translate показаний на рисунку 7

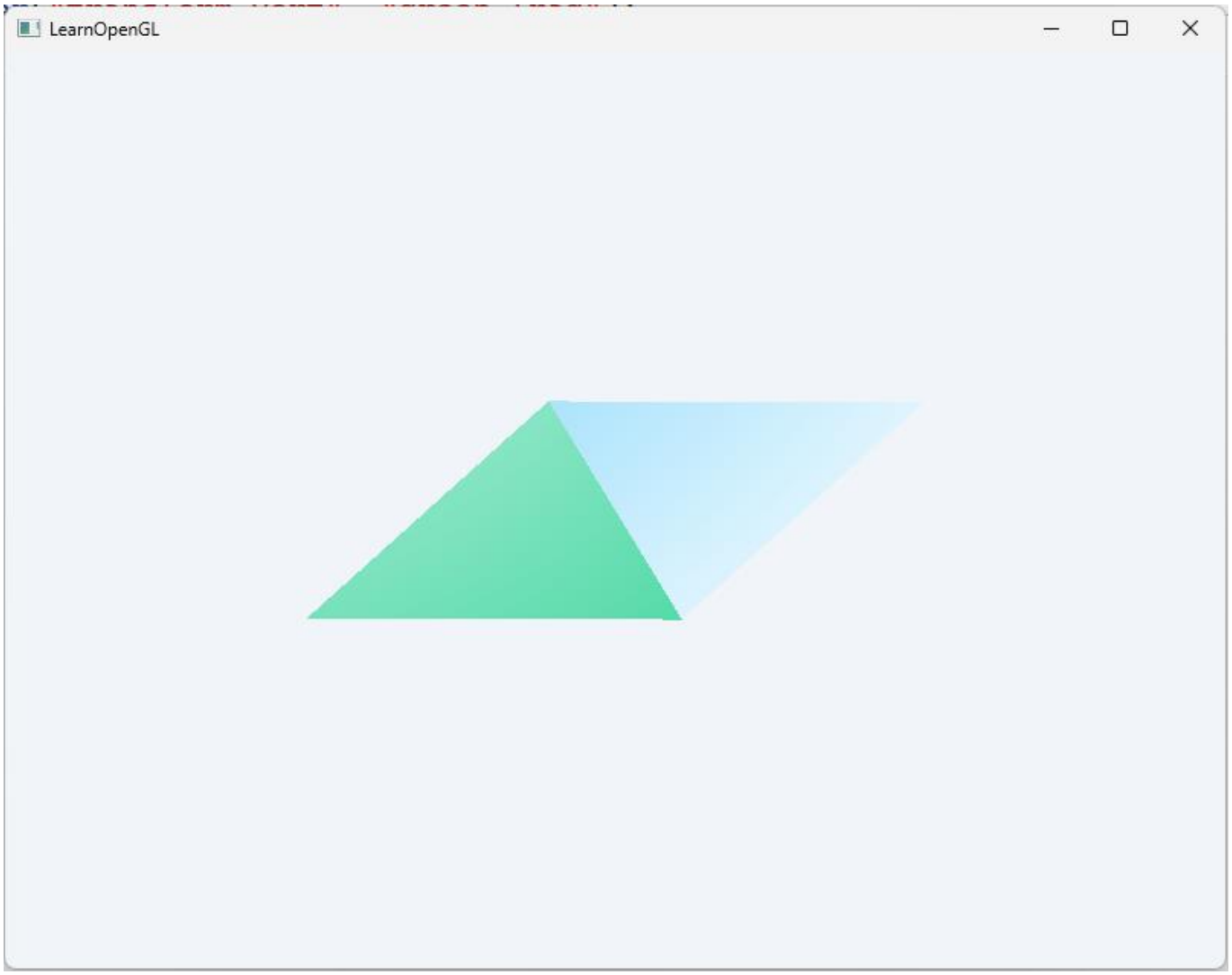

Рисунок 5 – Перший приклад роботи програми

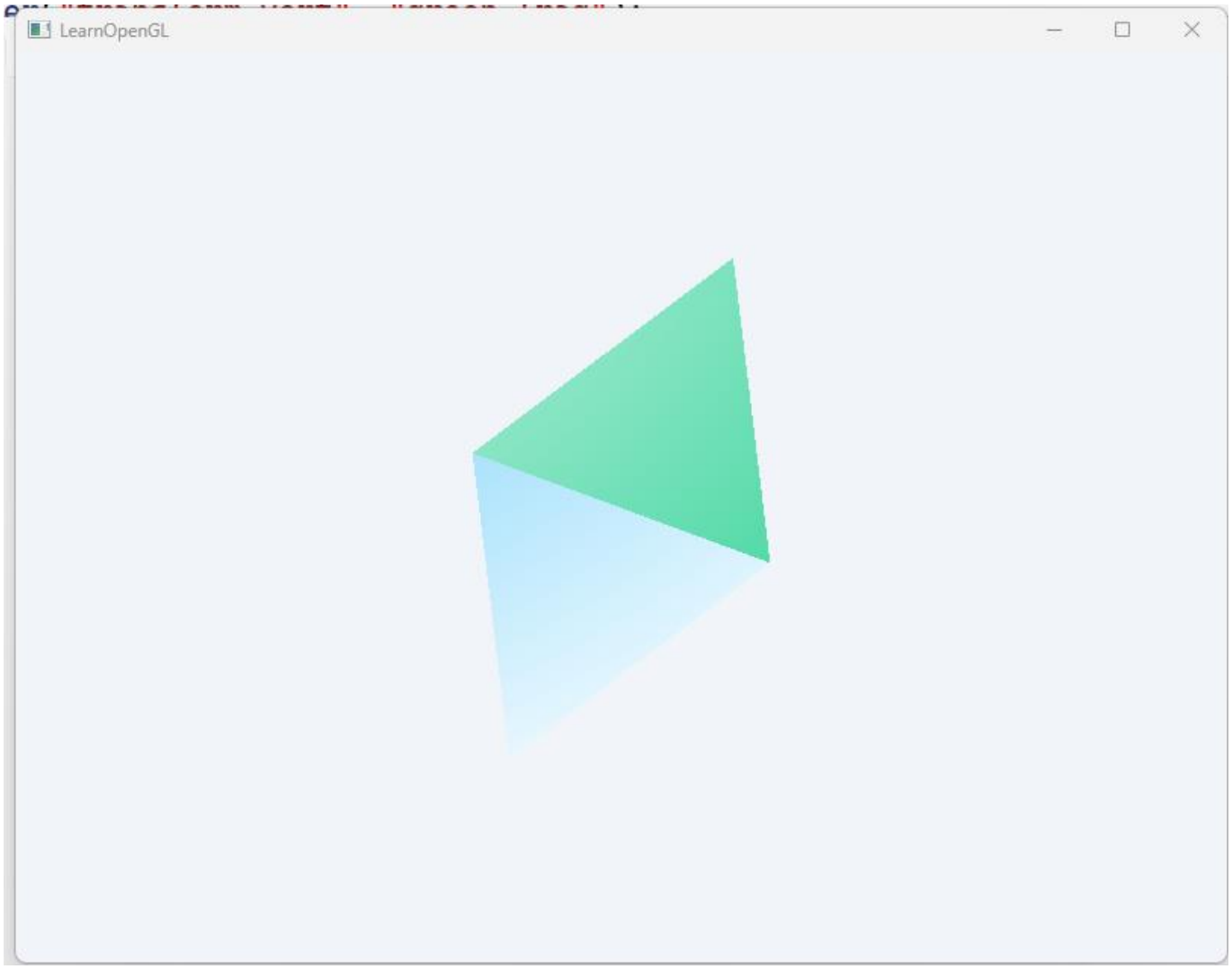

Рисунок 6 – Другий приклад роботи програми

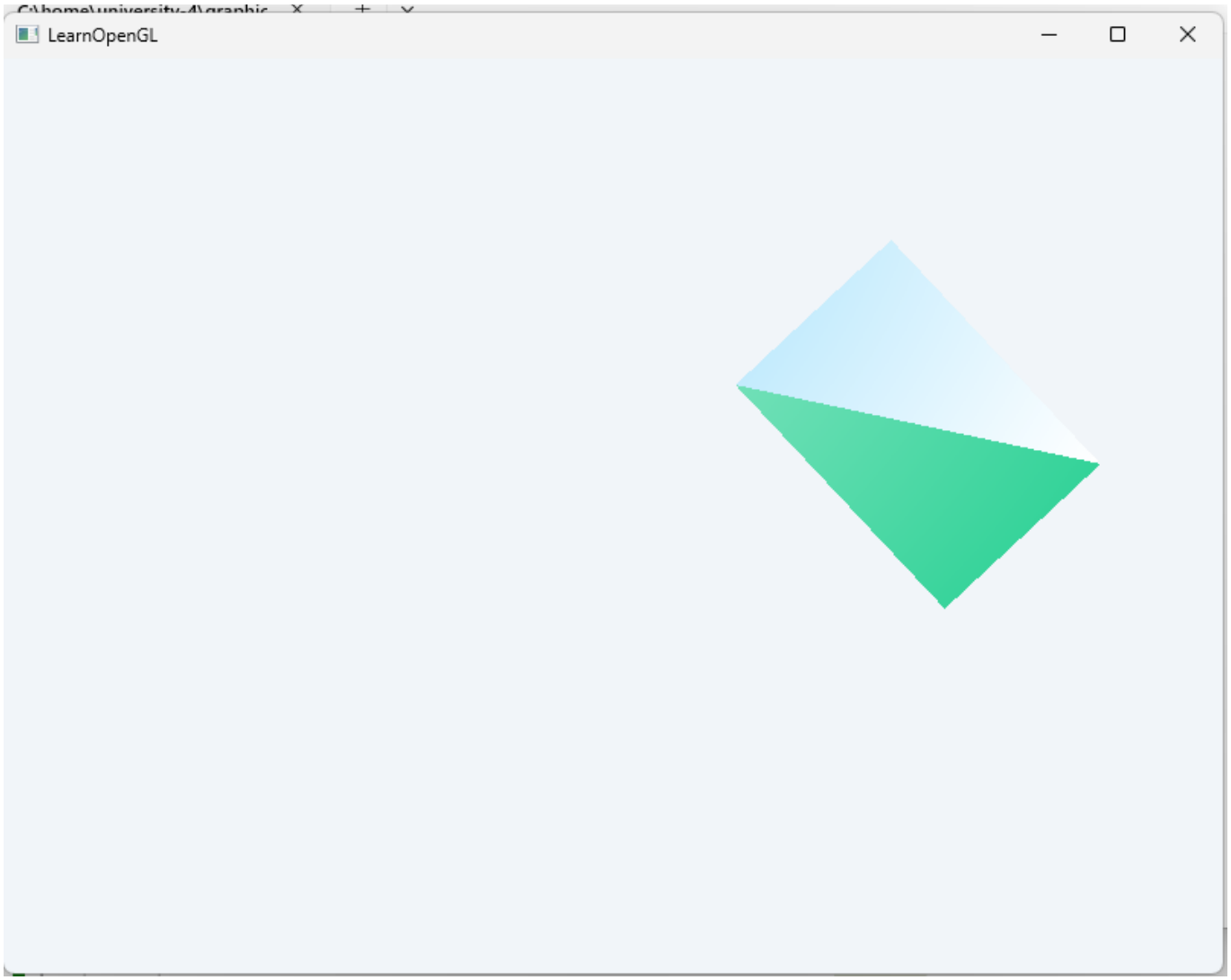

Рисунок 7 – Приклад роботи програми з переміщенням

4 Тест розробленої програми

```
+ shader.hpp
#ifndef SHADER_H
#define SHADER_H

#include <string>
#include <glad/glad.h>

#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>

/*!
 * OpenGL Shader program wrapper for work with GLSL shader files
 */
class Shader {
public:
	/*!
	* Create Shader program from two shader-files
```

```
	*
	* @param vertexPath - Path to vertex shader file in file system
	* @param fragmentPath - Path to vertex shader file in file system
	*/
	Shader(const char* vertexPath, const char* fragmentPath);

	/*!
	 * Use current shader
	 */
	void use();

	/*!
	 * Utils for set `uniform` variables in shaders
	 */
	void setVec2(const std::string& name, float x, float y) const;
	void setMat4(const std::string& name, glm::mat4 matrix) const;
private:
	GLuint ID;

	enum Error {
		PROGRAM,
		VERTEX,
		FRAGMENT
	};

	const std::string readShaderFile(std::string path);
	void checkOnErrors(GLuint shader, Shader::Error type);
};
#endif
```

+ shader.cpp

```
#include "shader.hpp"
#include <glad/glad.h>

#include <string>
#include <fstream>
#include <sstream>
#include <iostream>
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>

Shader::Shader(const char* vertexPath, const char* fragmentPath) {
	const std::string vertexCodeRaw = Shader::readShaderFile(vertexPath);
	const std::string fragmentCodeRaw = Shader::readShaderFile(fragmentPath);

	const char* vertexCode = vertexCodeRaw.c_str();
	const char* fragmentCode = fragmentCodeRaw.c_str();

	GLuint vertexShader = glCreateShader(GL_VERTEX_SHADER);
	glShaderSource(vertexShader, 1, &vertexCode, NULL);
	glCompileShader(vertexShader);
	Shader::checkOnErrors(vertexShader, Shader::Error::VERTEX);

	GLuint fragmentShader = glCreateShader(GL_FRAGMENT_SHADER);
	glShaderSource(fragmentShader, 1, &fragmentCode, NULL);
	glCompileShader(fragmentShader);
	Shader::checkOnErrors(fragmentShader, Shader::Error::FRAGMENT);

	Shader::ID = glCreateProgram();

	glAttachShader(Shader::ID, vertexShader);
	glAttachShader(Shader::ID, fragmentShader);
	glLinkProgram(Shader::ID);
	Shader::checkOnErrors(Shader::ID, Shader::Error::PROGRAM);
```

```
	glDeleteShader(vertexShader);
	glDeleteShader(fragmentShader);
}

void Shader::use() {
	glUseProgram(Shader::ID);
}

// TODO: Add GLM version?

void Shader::setVec2(const std::string& name, float x, float y) const {
	glUniform2f(glGetUniformLocation(Shader::ID, name.c_str()), x, y);
}

void Shader::setMat4(const std::string& name, glm::mat4 matrix) const {
	glUniformMatrix4fv(glGetUniformLocation(Shader::ID, name.c_str()), 1, GL_FALSE,
glm::value_ptr(matrix));
}

const std::string Shader::readShaderFile(std::string path) {
	std::string code;
	std::ifstream file;

	file.exceptions(std::ifstream::failbit | std::ifstream::badbit);

	try {
		file.open(path);

		std::stringstream stream;
		stream << file.rdbuf();

		file.close();

		code = stream.str();
	} catch (std::ifstream::failure e) {
		std::cout << "ERROR::SHADER::FILE_NOT_SUCCESFULLY_READ" << std::endl;
	}

	return code;
}

void Shader::checkOnErrors(GLuint shader, Shader::Error type) {
	GLint success;
	GLchar infoLog[1024];

	if (type == Shader::Error::PROGRAM) {
		glGetProgramiv(shader, GL_LINK_STATUS, &success);

		if (!success) {
			glGetProgramInfoLog(shader, 512, NULL, infoLog);

			std::cout << "ERROR::SHADER::LINK_FAILED\n" <<
				infoLog << std::endl;
		}

		return;
	}

	glGetShaderiv(shader, GL_COMPILE_STATUS, &success);

	if (!success) {
		glGetShaderInfoLog(shader, 512, NULL, infoLog);
```

```
            std::cout << "ERROR::SHADER::SHADER_COMPILATION_ERROR in type:" << type <<
"\n" <<
                  infoLog << std::endl;
      }
}
```

```
+ main.cpp
#include <glad/glad.h>
#include <glfw/glfw3.h>
#include <iostream>

#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>

#include "shader.hpp"

const unsigned int WIDTH = 800;
const unsigned int HEIGHT = 600;

void framebuffer_size_callback(GLFWwindow* window, int width, int height);
void processInput(GLFWwindow* window);
GLuint initVAO(float vertices[], size_t verticesSize, unsigned int indices[], size_t
indicesSize);

int main() {
      if (!glfwInit()) {
            std::cout << "GLFW failed to start" << std::endl;
            glfwTerminate();
            return -1;
      }

      glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
      glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
      glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);

      GLFWwindow* window = glfwCreateWindow(WIDTH, HEIGHT, "LearnOpenGL", NULL, NULL);

      if (window == NULL) {
            std::cout << "Failed to create GLFW window" << std::endl;
            glfwTerminate();
            return -1;
      }

      glfwMakeContextCurrent(window);

      if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress)) {
            std::cout << "Failed to initialize GLAD" << std::endl;
            return -1;
      }

      glfwSetFramebufferSizeCallback(window, framebuffer_size_callback);

      float vertices[] = {
            0.7f, 0.7f, 0.0f, // top right
            0.5f, -0.5f, 0.0f, // bottom right
            -0.7f, -0.7f, 0.0f, // bottom left
            -0.5f, 0.5f, 0.0f, // top left
      };

      unsigned int indices[] = {
            0, 1, 3,
            1, 2, 3
      };
```

```
	Shader greenShader("transform.vert", "green.frag");
	Shader skyShader("transform.vert", "sky.frag");

	GLuint VAO1 = initVAO(vertices, sizeof(vertices), indices, sizeof(indices));
	GLuint VAO2 = initVAO(vertices, sizeof(vertices), indices, sizeof(indices));

	while (!glfwWindowShouldClose(window)) {
		processInput(window);

		glClearColor(0.945f, 0.961f, 0.976f, 1.0f);
		glClear(GL_COLOR_BUFFER_BIT);

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

		trans = glm::translate(trans, glm::vec3(0.5f, 0.2f, 0.0f));
		trans = glm::rotate(trans, (float)glfwGetTime() * 0.9f, glm::vec3(0.0f, 1.0f,
0.0f));
		trans = glm::rotate(trans, (float)glfwGetTime() * 0.6f, glm::vec3(1.0f, 0.0f,
0.0f));
		trans = glm::rotate(trans, (float)glfwGetTime() * 0.8f, glm::vec3(0.0f, 0.0f,
1.0f));
		trans = glm::scale(trans, glm::vec3(0.5f, 0.75f, 0.2f));

		greenShader.use();
		greenShader.setVec2("u_resolution", WIDTH, HEIGHT);
		greenShader.setMat4("transform", trans);
		glBindVertexArray(VAO1);
		glDrawElements(GL_TRIANGLES, 3, GL_UNSIGNED_INT, (void*)(3 * sizeof(unsigned
int)));

		skyShader.use();
		skyShader.setVec2("u_resolution", WIDTH, HEIGHT);
		skyShader.setMat4("transform", trans);
		glBindVertexArray(VAO2);
		glDrawElements(GL_TRIANGLES, 3, GL_UNSIGNED_INT, (void *)0);

		glfwSwapBuffers(window);
		glfwPollEvents();
	}

	glfwTerminate();
	return 0;
}

/*!
 * Create Vertex Array Object from all vertices and indices
 */
GLuint initVAO(float vertices[], size_t verticesSize, unsigned int indices[], size_t
indicesSize) {
	GLuint VAO;
	glGenVertexArrays(1, &VAO);
	glBindVertexArray(VAO);

	GLuint VBO;
	glGenBuffers(1, &VBO);
	glBindBuffer(GL_ARRAY_BUFFER, VBO);
	glBufferData(GL_ARRAY_BUFFER, verticesSize, vertices, GL_STATIC_DRAW);

	GLuint EBO;
	glGenBuffers(1, &EBO);
	glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
	glBufferData(GL_ELEMENT_ARRAY_BUFFER, indicesSize, indices, GL_STATIC_DRAW);

	glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)0);
```

```
	glEnableVertexAttribArray(0);

	return VAO;
}

/*!
 * Process interaction with user
 */
void processInput(GLFWwindow* window) {
	if (glfwGetKey(window, GLFW_KEY_ESCAPE) == GLFW_PRESS) {
		glfwSetWindowShouldClose(window, true);
	}
}

/*!
 * Auto update OpenGL viewport on resize
 */
void framebuffer_size_callback(GLFWwindow* window, int width, int height) {
	glViewport(0, 0, width, height);
}
```

+ green.frag

```
#version 330 core
uniform vec2 u_resolution;

void main() {
	vec2 st = gl_FragCoord.xy/u_resolution.xy;

	vec3 color1 = vec3(1.0, 1.0, 1.0);
	vec3 color2 = vec3(0.204f, 0.827f, 0.6f);

	float mixValue = distance(st,vec2(0,1));
	vec3 color = mix(color1,color2, mixValue);

	gl_FragColor = vec4(color, mixValue);
}
```

+ sky.frag

```
#version 330 core
uniform vec2 u_resolution;

void main() {
	vec2 st = gl_FragCoord.xy/u_resolution.xy;

	vec3 color1 = vec3(0.22, 0.741, 0.973);
	vec3 color2 = vec3(1.0, 1.0, 1.0);

	float mixValue = distance(st,vec2(0,1));
	vec3 color = mix(color1,color2,mixValue);

	gl_FragColor = vec4(color, mixValue);
}
```

+ simple.vert

```
#version 330 core
layout (location = 0) in vec3 aPos;

void main() {
	gl_Position = vec4(aPos, 1.0);
}
```

+ transform.vert

```
#version 330 core
layout (location = 0) in vec3 aPos;
```

```
uniform mat4 transform;

void main() {
	gl_Position = transform * vec4(aPos, 1.0);
}
```

5 Висновки

Було отримано практичні навички створення шейдерів, роботи з матрицями, GLM та роботи з бібліотекою OpenGL, використовуючи мову програмування C++