Міністерство освіти і науки України

Національний університет «Запорізька політехніка»

кафедра програмних засобів

ЗВІТ

з лабораторної роботи № 5

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

«КАМЕРА»

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 фігуру та колір.

2.2 Змініть розмір вікна.

2.3 Наведіть коментарі до коду.

2.4 Опищіть алгоритм роботи камери.

2.5 Знайдіть зайве у коді.

2.6 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.

В результаті виконання роботи було додано можливість переміщувати камеру по вертикалі за допомогою клавіш W та S, показано на рисунку 1. Було додано можливість повертати камеру відносно точки, показано на рисунку 2. Було додано можливість повертати камеру через мишку використовуючи pitch та yaw, показано на рисунку 3. Було додано zoom через колесико мишки, показано на рисунку 4.

На рисунку 5 показано використання glm::lookAt. На рисунках 6, 7 та 8 показані різні позиції об’єкта в просторі.

```
void processInput(GLFWwindow* window) {
    if (glfwGetKey(window, GLFW_KEY_ESCAPE) == GLFW_PRESS) {
        glfwSetWindowShouldClose(window, true);
    }

    const float cameraSpeed = 2.5f * deltaTime;

    if (glfwGetKey(window, GLFW_KEY_W) == GLFW_PRESS) {
        cameraPos += cameraSpeed * cameraFront;
    }

    if (glfwGetKey(window, GLFW_KEY_S) == GLFW_PRESS) {
        cameraPos -= cameraSpeed * cameraFront;
    }
```

Рисунок 1 – Зміна позиції камери при натисканні W або S

```
// Rotate aroud center
if (glfwGetKey(window, GLFW_KEY_A) == GLFW_PRESS || glfwGetKey(window, GLFW_KEY_D) == GLFW_PRESS) {
    float difference = 1.0f;

    if (glfwGetKey(window, GLFW_KEY_D) == GLFW_PRESS) {
        difference *= -1;
    }

    glm::mat4 rotationMatrix(1.0f);
    rotationMatrix = glm::rotate(rotationMatrix, glm::radians(difference), cameraUp);
    glm::vec4 rotatedRelativePos = rotationMatrix * glm::vec4(cameraPos, 1.0f);
    cameraPos = glm::vec3(rotatedRelativePos);

    yaw += difference * -1;

    glm::vec3 direction;

    direction.x = cos(glm::radians(yaw)) * cos(glm::radians(pitch));
    direction.y = sin(glm::radians(pitch));
    direction.z = sin(glm::radians(yaw)) * cos(glm::radians(pitch));

    cameraFront = glm::normalize(direction);
}
```

Рисунок 2 – Поворот камери відносно середини екрану

```
void mouse_callback(GLFWwindow* window, double xPos, double yPos) {
    if (firstMouse) {
        lastX = xPos;
        lastY = yPos;
        firstMouse = false;
    }

    const float sensitivity = 0.1f;

    float xOffset = xPos - lastX;
    // reversed, y ranges bottom to top
    float yOffset = lastY - yPos;

    lastX = xPos;
    lastY = yPos;

    xOffset *= sensitivity;
    yOffset *= sensitivity;

    yaw += xOffset;
    pitch += yOffset;

    if (pitch > 89.0f) pitch = 89.0f;
    if (pitch < -89.0f) pitch = -89.0f;

    glm::vec3 direction;

    direction.x = cos(glm::radians(yaw)) * cos(glm::radians(pitch));
    direction.y = sin(glm::radians(pitch));
    direction.z = sin(glm::radians(yaw)) * cos(glm::radians(pitch));

    cameraFront = glm::normalize(direction);
}
```

Рисунок 3 – Зміна позиції камери через мишку

```
void scroll_callback(GLFWwindow* window, double xOffset, double yOffset) {
    fov -= (float)yOffset;

    if (fov < 1.0f) fov = 1.0f;
    if (fov > 45.0f) fov = 45.0f;
}
```

Рисунок 4 – Зміна зуму камери через скролл

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

float currentFrame = glfwGetTime();
deltaTime = currentFrame - lastFrame;
lastFrame = currentFrame;

glm::mat4 model = glm::mat4(1.0f);
glm::mat4 view = glm::mat4(1.0f);
glm::mat4 projection = glm::mat4(1.0f);

model = glm::rotate(model, glm::radians(-55.0f), glm::vec3(1.0f, 0.0f, 0.0f));

view = glm::lookAt(cameraPos, cameraPos + cameraFront, cameraUp);

projection = glm::perspective(glm::radians(fov), (float)WIDTH / (float)HEIGHT, 0.1f, 100.0f);
```

Рисунок 5 – Використання lookAt

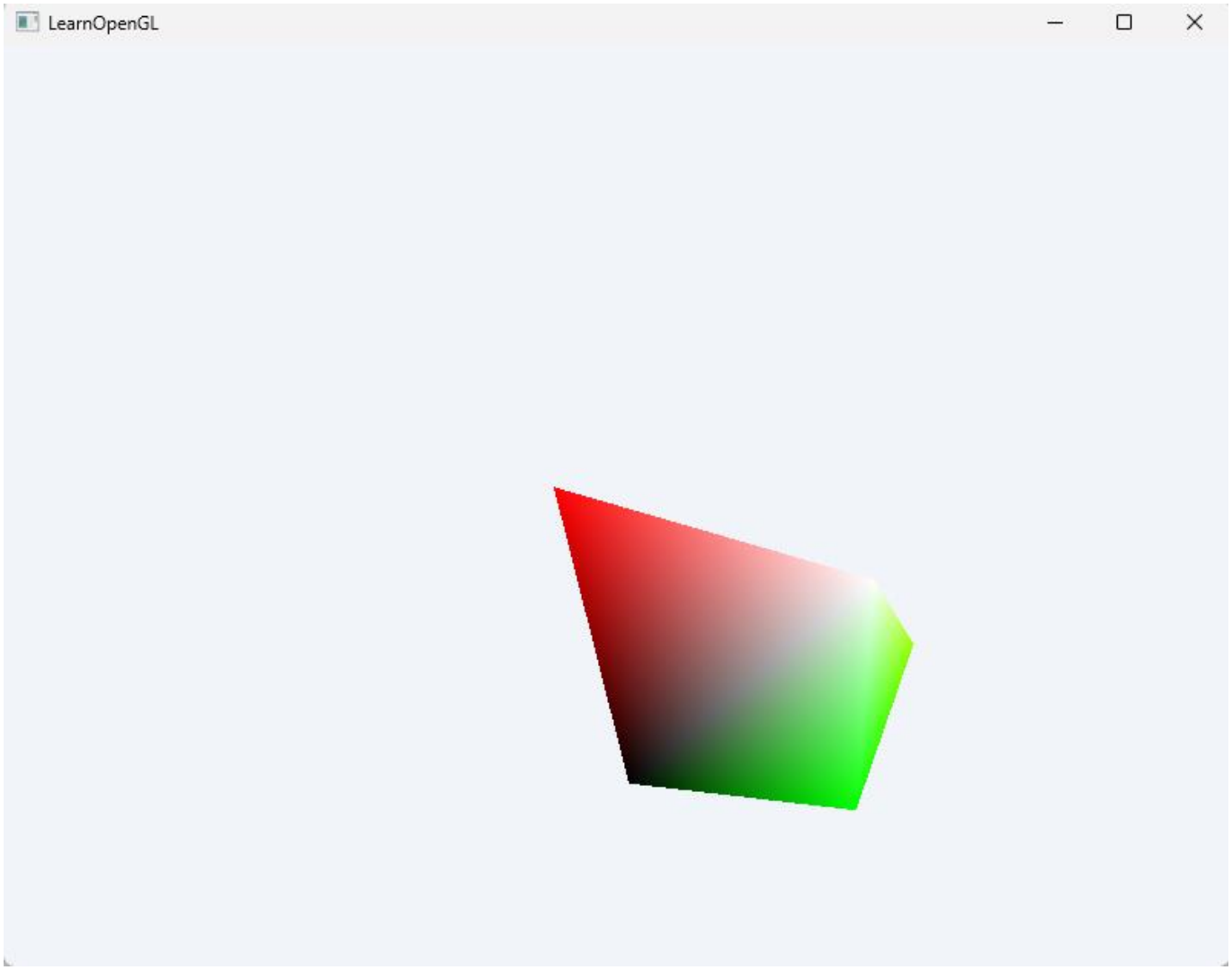

Рисунок 6 – Приклад 1

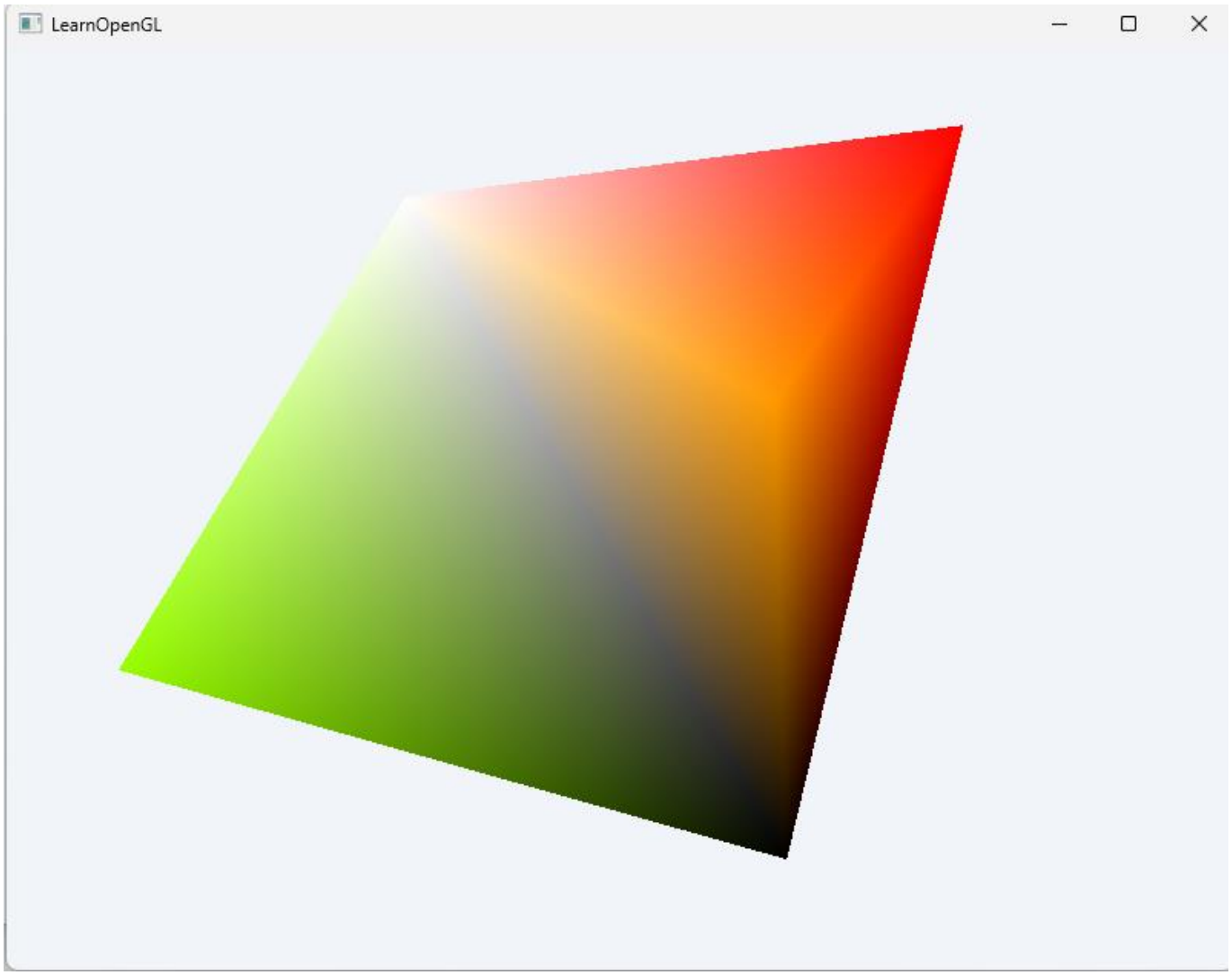

Рисунок 7 – Приклад 2

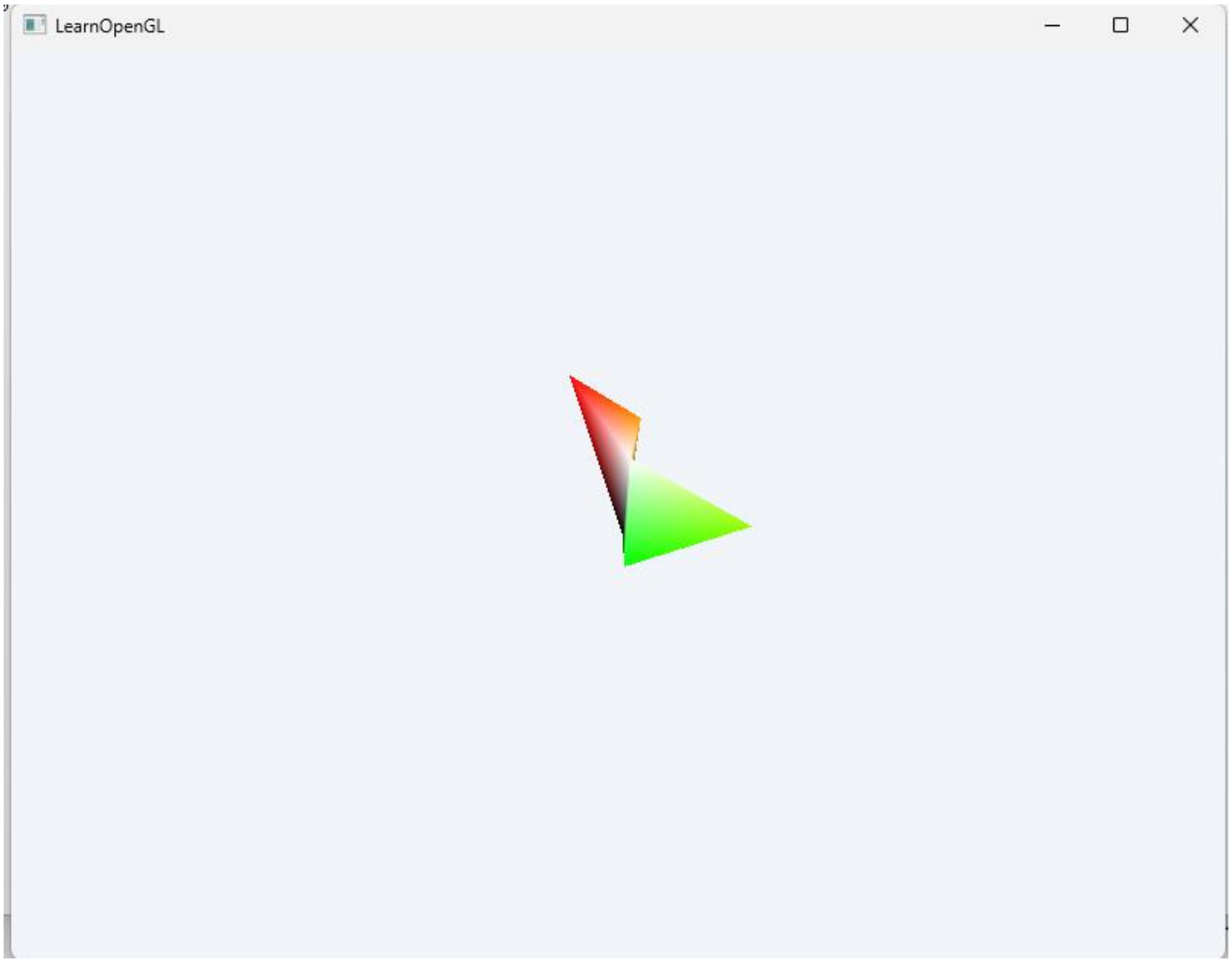

Рисунок 8 – Приклад 3

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);
void mouse_callback(GLFWwindow* window, double xPos, double yPos);
void scroll_callback(GLFWwindow* window, double xOffset, double yOffset);

glm::vec3 cameraPos = glm::vec3(0.0f, 0.0f, 3.0f);
glm::vec3 cameraFront = glm::vec3(0.0f, 0.0f, -1.0f);
glm::vec3 cameraUp = glm::vec3(0.0f, 1.0f, 0.0f);

float deltaTime = 0.0f; // Time between current frame and last frame
float lastFrame = 0.0f; // Time of last frame

float lastX = WIDTH / 2;
float lastY = HEIGHT / 2;

// Camera up/down
float pitch = 0.0f;
// Camera right/left
float yaw = -90.0f;

bool firstMouse = true;
float fov = 45.0f;

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);
glfwSetScrollCallback(window, scroll_callback);
glfwSetCursorPosCallback(window, mouse_callback);
glfwSetInputMode(window, GLFW_CURSOR, GLFW_CURSOR_DISABLED);
glEnable(GL_DEPTH_TEST);

// First 3 is pos, second 3 is color in RGB
float vertices[] = {
	// Base, shared for both
	-0.5f, 0.5f, 0.3f,		1.0f, 1.0f, 1.0f, // A aka top-left
	0.5f, -0.5f, -0.5f,		0.0f, 0.0f, 0.0f, // B aka bottom-right

	// First
	0.7f, 0.7f, 0.5f,		1.0f, 0.0f, 0.0f, // C aka top-right
	0.5f, 0.0f, 0.5f,		1.0f, 0.6f, 0.0f, // D new for 3d

	// Second
	-0.7f, -0.7f, 0.3f,		0.6f, 1.0f, 0.0f, // K aka bottom left
	-0.5f, 0.0f, -0.5f,		0.0f, 1.0f, 0.0f, // G new for 3d
};

unsigned int indices[] = {
	// First
	0, 1, 2,
	0, 1, 3,
	1, 2, 3,
	0, 2, 3,

	// Second
	0, 1, 4,
	0, 1, 5,
	1, 4, 5,
	0, 4, 5,
};

Shader greenShader("base.vert", "base.frag");
Shader skyShader("base.vert", "base.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 | GL_DEPTH_BUFFER_BIT);

	float currentFrame = glfwGetTime();
	deltaTime = currentFrame - lastFrame;
	lastFrame = currentFrame;

	glm::mat4 model = glm::mat4(1.0f);
	glm::mat4 view = glm::mat4(1.0f);
	glm::mat4 projection = glm::mat4(1.0f);

	model = glm::rotate(model, glm::radians(-55.0f), glm::vec3(1.0f, 0.0f, 0.0f));

	view = glm::lookAt(cameraPos, cameraPos + cameraFront, cameraUp);
```

```
            projection = glm::perspective(glm::radians(fov), (float)WIDTH / (float)HEIGHT,
0.1f, 100.0f);

            skyShader.use();
            skyShader.setVec2("u_resolution", WIDTH, HEIGHT);
            skyShader.setMat4("model", model);
            skyShader.setMat4("view", view);
            skyShader.setMat4("projection", projection);
            glBindVertexArray(VAO2);

            glDrawElements(GL_TRIANGLES, 12, GL_UNSIGNED_INT, (void *)0);

            greenShader.use();
            greenShader.setVec2("u_resolution", WIDTH, HEIGHT);
            greenShader.setMat4("model", model);
            greenShader.setMat4("view", view);
            greenShader.setMat4("projection", projection);
            glBindVertexArray(VAO1);
            glDrawElements(GL_TRIANGLES, 12, GL_UNSIGNED_INT, (void*)(12 * sizeof(unsigned
int)));

            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, 6 * sizeof(float), (void*)0);
      glEnableVertexAttribArray(0);

      glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)(3 *
sizeof(float)));
      glEnableVertexAttribArray(1);

      return VAO;
}

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

```
	const float cameraSpeed = 2.5f * deltaTime;

	if (glfwGetKey(window, GLFW_KEY_W) == GLFW_PRESS) {
		cameraPos += cameraSpeed * cameraFront;
	}

	if (glfwGetKey(window, GLFW_KEY_S) == GLFW_PRESS) {
		cameraPos -= cameraSpeed * cameraFront;
	}

	// Rotate aroud center
	if (glfwGetKey(window, GLFW_KEY_A) == GLFW_PRESS || glfwGetKey(window, GLFW_KEY_D) ==
GLFW_PRESS) {
		float difference = 1.0f;

		if (glfwGetKey(window, GLFW_KEY_D) == GLFW_PRESS) {
			difference *= -1;
		}

		glm::mat4 rotationMatrix(1.0f);
		rotationMatrix = glm::rotate(rotationMatrix, glm::radians(difference),
cameraUp);
		glm::vec4 rotatedRelativePos = rotationMatrix * glm::vec4(cameraPos, 1.0f);
		cameraPos = glm::vec3(rotatedRelativePos);

		yaw += difference * -1;

		glm::vec3 direction;

		direction.x = cos(glm::radians(yaw)) * cos(glm::radians(pitch));
		direction.y = sin(glm::radians(pitch));
		direction.z = sin(glm::radians(yaw)) * cos(glm::radians(pitch));

		cameraFront = glm::normalize(direction);
	}
}

void mouse_callback(GLFWwindow* window, double xPos, double yPos) {
	if (firstMouse) {
		lastX = xPos;
		lastY = yPos;
		firstMouse = false;
	}

	const float sensitivity = 0.1f;

	float xOffset = xPos - lastX;
	// reversed, y ranges bottom to top
	float yOffset = lastY - yPos;

	lastX = xPos;
	lastY = yPos;

	xOffset *= sensitivity;
	yOffset *= sensitivity;

	yaw += xOffset;
	pitch += yOffset;

	if (pitch > 89.0f) pitch = 89.0f;
	if (pitch < -89.0f) pitch = -89.0f;

	glm::vec3 direction;
```

```
        direction.x = cos(glm::radians(yaw)) * cos(glm::radians(pitch));
        direction.y = sin(glm::radians(pitch));
        direction.z = sin(glm::radians(yaw)) * cos(glm::radians(pitch));

        cameraFront = glm::normalize(direction);
}

void scroll_callback(GLFWwindow* window, double xOffset, double yOffset) {
        fov -= (float)yOffset;

        if (fov < 1.0f) fov = 1.0f;
        if (fov > 45.0f) fov = 45.0f;
}

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

```
+ base.vert
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aColor;

out vec3 outColor;

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

void main() {
        gl_Position = projection * view * model * vec4(aPos, 1.0);
        outColor = aColor;
}
```

```
+ base.frag
#version 330 core

in vec3 outColor;

void main() {
        gl_FragColor = vec4(outColor, 1.0f);
}
```

5 Висновки

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