Міністерство освіти і науки України

Національний університет «Запорізька політехніка»

кафедра програмних засобів

ЗВІТ

з лабораторної роботи № 4

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

«ІНТЕРПОЛЯЦІЯ, ІНДЕКСИ МАЛЮВАННЯ ТА ПРОЄКЦІЇ»

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 її колір.

2.2 Запишіть коротке відео результату.

2.3 Наведіть коментарі до коду.

2.4 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.

Лабораторна робота складається з двох основних частин – використання індексів замість координат точок та перспектива. Індекси вже були реалізовані раніше, ще в першій лабораторній роботі

Індекси малювання представляють собою масив з індексами трикутників, які будуть потім нарисовані. Це потрібно щоб уникнути дублювання точок, одна точка це 3, або ж навіть 6 якщо з кольором float елементів, при цьому індекс це один int. В код була внесена тільки одна зміна – до кожної точки було додано її колір, який буде

через інтерполяцію створювати градієнт, та був оновлений код шейдерів для підтримки кольору з vertex. Це показано на рисунках 1, 2, 3 та 4.

Також було реалізовано перспективу. Для цього було оновлено vertex шейдер, тепер він приймає три матриці, а саме model, view та projection. Це показано на рисунках 1 та 3.

```
#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;
}
```

Рисунок 1 – Оновлений vertex шейдер

```
// 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,
};
```

Рисунок 2 – Індекси малювання

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

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

    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));
    model = glm::rotate(model, (float)glfwGetTime() * 0.6f, glm::vec3(0.0f, 1.0f, 0.0f));
    model = glm::rotate(model, (float)glfwGetTime() * 0.8f, glm::vec3(0.0f, 0.0f, 1.0f));

    view = glm::translate(view, glm::vec3(0.0f, 0.0f, -3.0f));
    projection = glm::perspective(glm::radians(45.0f), (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();
}
```

Рисунок 3 – Логіка матриць та перспетива

```
*/
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;
}
```

Рисунок 4 – Оновлена логіка створення VAO з коліром

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>
```

```cpp
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);
	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);

		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));
		model = glm::rotate(model, (float)glfwGetTime() * 0.6f, glm::vec3(0.0f, 1.0f,
0.0f));
		model = glm::rotate(model, (float)glfwGetTime() * 0.8f, glm::vec3(0.0f, 0.0f,
1.0f));

		view = glm::translate(view, glm::vec3(0.0f, 0.0f, -3.0f));
```

```
            projection = glm::perspective(glm::radians(45.0f), (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);
      }
```

```
}

/*!
 * 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 та роботи з бібліотекою OpenGL, використовуючи мову програмування C++