**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»**

**ІКНІ**

Кафедра **ПЗ**



**ЗВІТ**

до лабораторної роботи №2

**на тему:** *“Генерування динамічних двовимірних зображень за допомогою бібліотеки OpenGL”*

**з дисципліни** *“Засоби програмування комп'ютерної графіки”*

**Лектор:**

асис. каф. ПЗ

Журавчак Л.М.

**Виконав:**

ст. гр. ПЗ-33

Юшкевич. А.І.

**Прийняв:**

ст. викл. каф. ПЗ

Журавчак Л.М.

«\_\_» \_\_\_\_\_\_\_\_\_ 2024р.

∑=\_\_\_\_\_

Львів – 2024

**Тема:** Генерування динамічних двовимірних зображень за допомогою бібліотеки OpenGL.

**Мета:** Навчитися будувати двовимірні зображення з подальшим їхнім переміщенням, обертанням та масштабуванням на основі використання базових геометричних перетворень OpenGL.

**Теоретичні відомості**

Для створення динамічних зображень, які змінюють у часі своє положення, орієнтацію та розміри, в OpenGL використовують основні геометричні перетворення – перенесення, обертання і масштабування.

Більшість фігур наперед визначеної форми (особливо це стосується тривимірних об’єктів) можна намалювати тільки на початку координат, тобто в точці (0,0,0). Для того, щоб зобразити фігуру в точці (x0,y0,z0), треба перемістити початок координат у цю точку, тобто перейти до нових координат. Часто буває дуже зручно, крім того, повернути координати на необхідний кут. Для переходу до нових координат в OpenGL є дві функції:

**glTranslate(dx,dy,dz)**

**glRotate(ang,x0,y0,z0)**

Перша функція зміщує початок системи координат на (*dx,dy,dz*), друга – повертає на кут *ang* проти годинникової стрілки навколо вектора (*x0,y0,z0*), який виходить з початку координат (0,0,0). Ще дві дуже важливі функції –

**glPushMatrix()**

**glPopMatrix()**

призначені для збереження й відновлення поточних координат. Отож, для того, щоб намалювати фігуру не на початку координат, треба:

1. зберегти поточні координати (**glPushMatrix**);
2. змістити (**glTranslated**) та повернути (**glRotated**) систему координат;
3. намалювати те, що треба;
4. повернутися до старих координат (**glPopMatrix**).

Виклики **glPushMatrixglPopMatrix**можуть бути вкладеними, тобто:

**glPushMatrix();**

**...**

**glPushMatrix();**

**glPopMatrix();**

**...**

**glPopMatrix();**

Зрозуміло, що кількість викликів **glPopMatrix**повинне відповідати кількості викликів **glPushMatrix**, інакше вся сцена полетить у наперед невідомому напрямку.

Подібним чином функція **glScale(kx,ky,kz)** масштабує об’єкт (стискає чи розтягує його) на задані коефіцієнти по відповідних осях. Вона підходить також і для генерації дзеркального відображення – для цього коефіцієнтам надають від’ємних значень.

Отож, для створення засобами OpenGL сцени із динамічно змінюваними об’єктами треба організувати цикл, на кожній ітерації якого будуть обчислені значення зміщень та кутів повороту потрібних об’єктів (і, якщо необхідно, коефіцієнтів масштабування) та виконано пункти *a)–d)* процедури малювання об’єктів.

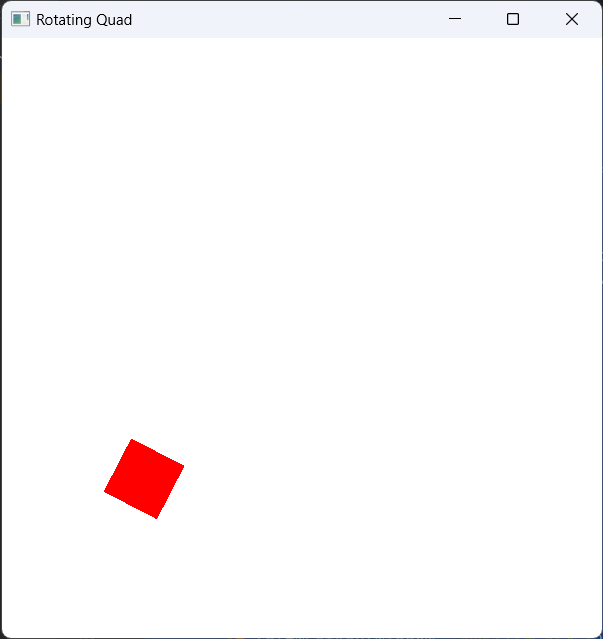
**Завдання**

12. Задано **чотирикутник**. Реалізувати неперервний рух фігури за траєкторією ***y=ax***в обох напрямках з одночасним обертанням навколо заданої вершини.

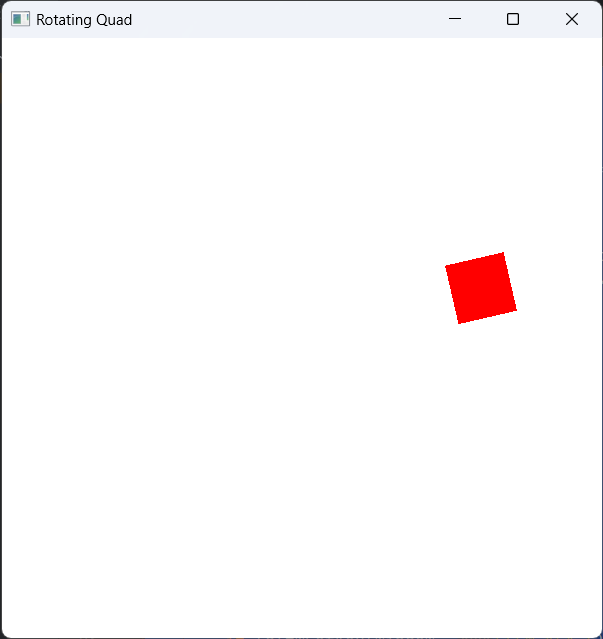
**Код програми**

#include <GL/glew.h>  
#include <GLFW/glfw3.h>  
#include <iostream>  
#include <cmath>  
  
struct Vertex {  
 float x, y;  
};  
  
float angle = 0.05f;  
float speed = 0.0002f;  
float slope = 0.5f;  
Vertex rotationVertex;  
  
Vertex rectangle[] = {  
 Vertex{-0.1f, -0.1f},  
 Vertex{0.1f, -0.1f},  
 Vertex{0.1f, 0.1f},  
 Vertex{-0.1f, 0.1f},  
};  
  
void updateRectangle(float angleRad, float dx) {  
 for (int i = 0; i < 4; i++) {  
 // Рух уздовж прямої y = ax  
 rectangle[i].x += dx;  
 rectangle[i].y += slope \* dx;  
 }  
 rotationVertex.x += dx;  
 rotationVertex.y += slope \* dx;  
 for (int i = 0; i < 4; i++) {  
 float rel\_x = rectangle[i].x - rotationVertex.x;  
 float rel\_y = rectangle[i].y - rotationVertex.y;  
  
 float rotated\_x = rel\_x \* cos(angleRad) - rel\_y \* sin(angleRad);  
 float rotated\_y = rel\_x \* sin(angleRad) + rel\_y \* cos(angleRad);  
  
 rectangle[i].x = rotated\_x + rotationVertex.x;  
 rectangle[i].y = rotated\_y + rotationVertex.y;  
 }  
}  
  
void drawQuad() {  
 glBegin(GL\_QUADS);  
 glColor3f(1.0f, 0.0f, 0.0f);  
 for (int i = 0; i < 4; i++) {  
 glVertex2f(rectangle[i].x, rectangle[i].y);  
 }  
 glEnd();  
}  
  
void render() {  
 glClear(GL\_COLOR\_BUFFER\_BIT);  
 drawQuad();  
 glFlush();  
}  
  
int main() {  
 if (!glfwInit()) {  
 std::cerr << "Failed to initialize GLFW" << std::endl;  
 return -1;  
 }  
  
 GLFWwindow\* window = glfwCreateWindow(600, 600, "Rotating Quad", nullptr, nullptr);  
 if (!window) {  
 std::cerr << "Failed to create GLFW window" << std::endl;  
 glfwTerminate();  
 return -1;  
 }  
 glfwMakeContextCurrent(window);  
 glewInit();  
  
 glClearColor(1, 1, 1, 1);  
 rotationVertex = rectangle[0];  
  
 while (!glfwWindowShouldClose(window)) {  
 float angleRad = angle \* M\_PI / 180.0f;  
 updateRectangle(angleRad, speed);  
 for (int i = 0; i < 4; i++) {  
 if (rectangle[i].x < -0.8f || rectangle[i].x > 0.8f  
 || rectangle[i].y < -0.8f || rectangle[i].y > 0.8f) {  
 speed = -speed;  
 angle = -angle;  
 }  
 }  
  
 render();  
 glfwSwapBuffers(window);  
 glfwPollEvents();  
  
 if (glfwGetKey(window, GLFW\_KEY\_1) == GLFW\_PRESS) rotationVertex = rectangle[0];  
 if (glfwGetKey(window, GLFW\_KEY\_2) == GLFW\_PRESS) rotationVertex = rectangle[1];  
 if (glfwGetKey(window, GLFW\_KEY\_3) == GLFW\_PRESS) rotationVertex = rectangle[2];  
 if (glfwGetKey(window, GLFW\_KEY\_4) == GLFW\_PRESS) rotationVertex = rectangle[3];  
 }  
  
 glfwDestroyWindow(window);  
 glfwTerminate();  
 return 0;  
}

**Результати виконання роботи**



*Рис. 1. Чотирикутник, що обертається навколо однієї зі своїх першин та рухається вздовж осі.*

****

*Рис. 2. Чотирикутник, що обертається навколо однієї зі своїх першин та рухається вздовж осі.*

**Висновки:** Під час виконання лабораторної роботи я навчився будувати двовимірні зображення з подальшим їхнім переміщенням, обертанням та масштабуванням на основі використання базових геометричних перетворень OpenGL.