

**МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ  
 НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ   
імені ІГОРЯ СІКОРСЬКОГО»**

**Навчально-науковий інститут прикладного системного аналізу  
Кафедра системного проектування**

**Звіт**

**про виконання лабораторної роботи №2  
з дисципліни «Комп’ютерна графіка»**

Виконав:  
студент III курсу, групи ДА-22  
Жадько Микита Сергійович

Київ – 2024

ПОСТАНОВКА ЗАДАЧІ

Мета роботи: Отримати навички з побудови тривимірних зображень з базових примітивів та атомарних елементів.

Завдання:

1. Ознайомитися з принципами побудови тривимірної системи координат.
2. Побудувати сцену, що складається з трьох об'єктів: дві базові фігури і площину. Вважаємо, що центр площини - збігається з центром сцени, а фігури розташовані на рівній відстані по обом сторонам від неї.
3. У відповідності до свого варіанту відобразити на екрані базові бібліотечні об’єкти у вигляді каркасу та у вигляді суцільного об’єкту. Задати два різних кольори заповнення.
4. Створити поверхню з вершин за допомогою функцій glVertexPointer та glVertex3f, у відповідності до функції вашого варіанту
5. Використовуючи команду gluLookAt та функції обробки клавіш, створити рухому камеру для зображених об’єктів. (Камеру рухаємо навколо центру сцени - повний оборот в горизонтальній і вертикальній площині)
6. Отримати зображення об’єктів в ортогональній та перспективній проекціях використовуючи функції glOrtho, glFrustum, gluLookAt (перемикання між проекціями запрограмувати на натискання клавіш "P" та "O" відповідно).
7. При цьому необхідно враховувати що матриці об’єктів та відображення мають бути обрані безпосередньо командою glMatrixMode, та попередньо нормовані функцією glLoadIdentity.

Завдання за варіантом:

|  |  |  |  |
| --- | --- | --- | --- |
| Варіант | Контурний примітив | Зафарбований примітив | Функція |
| 9 | Тор | Куб | *Z=sin(x)\*cos(y)* |

ХІД РОБОТИ

1. **Блок-схема дерева проекту.**

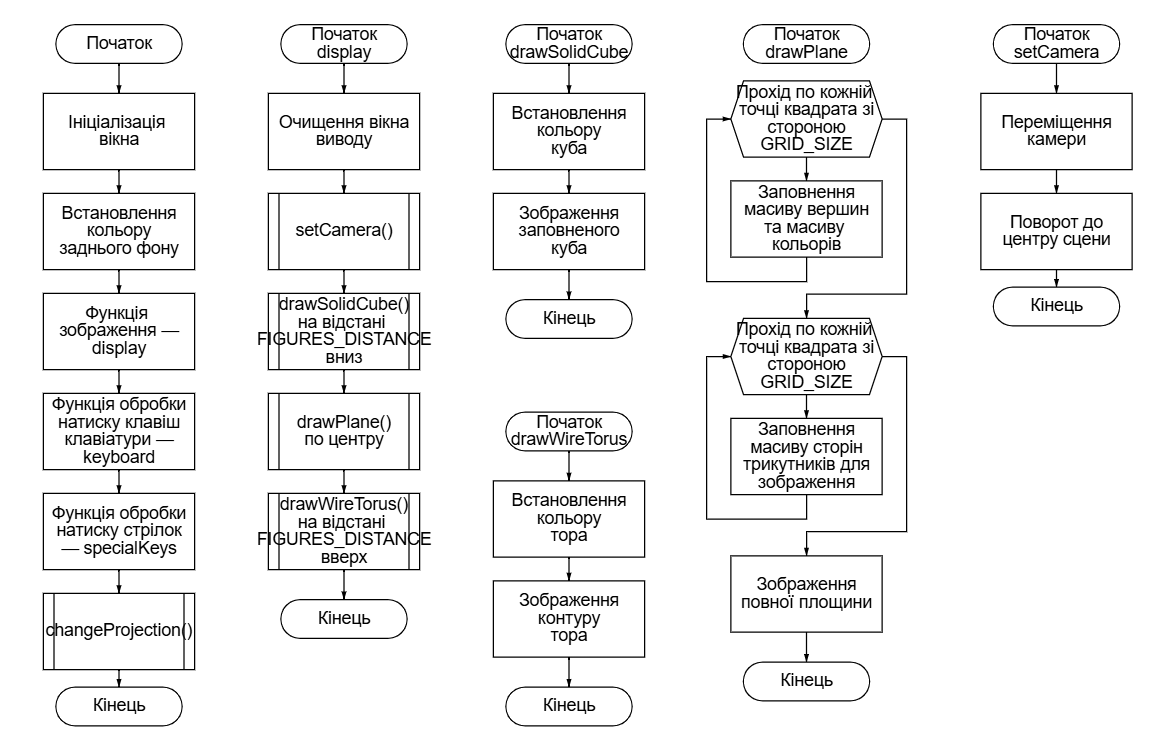


Рисунок 1 — Блок-схема дерева проекту (частина 1).

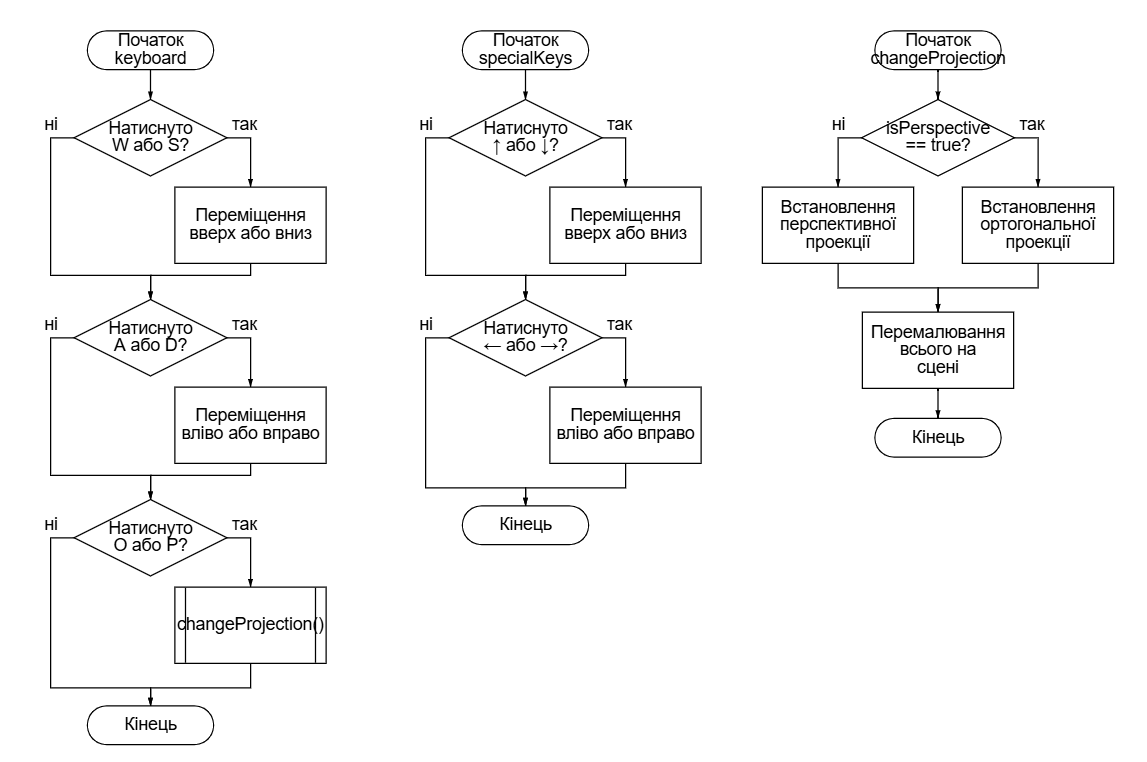


Рисунок 2 — Блок-схема дерева проекту (частина 2).

1. **Результати виконання лабораторної роботи.**

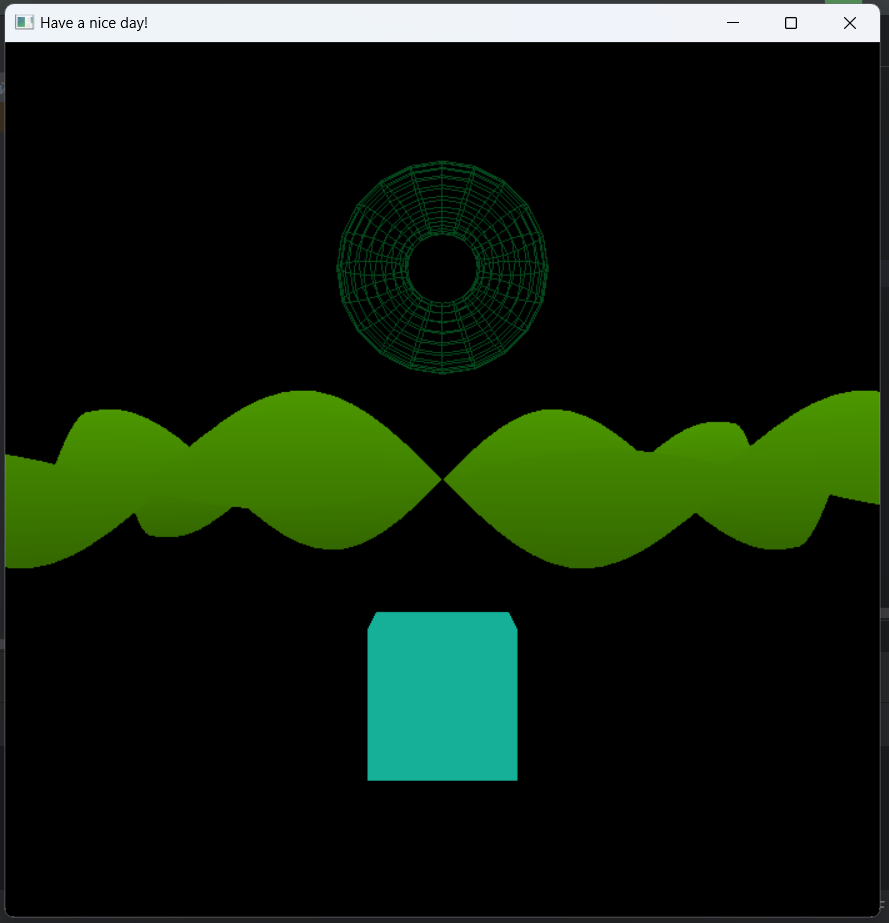


Рисунок 3 — Завантажена програма.

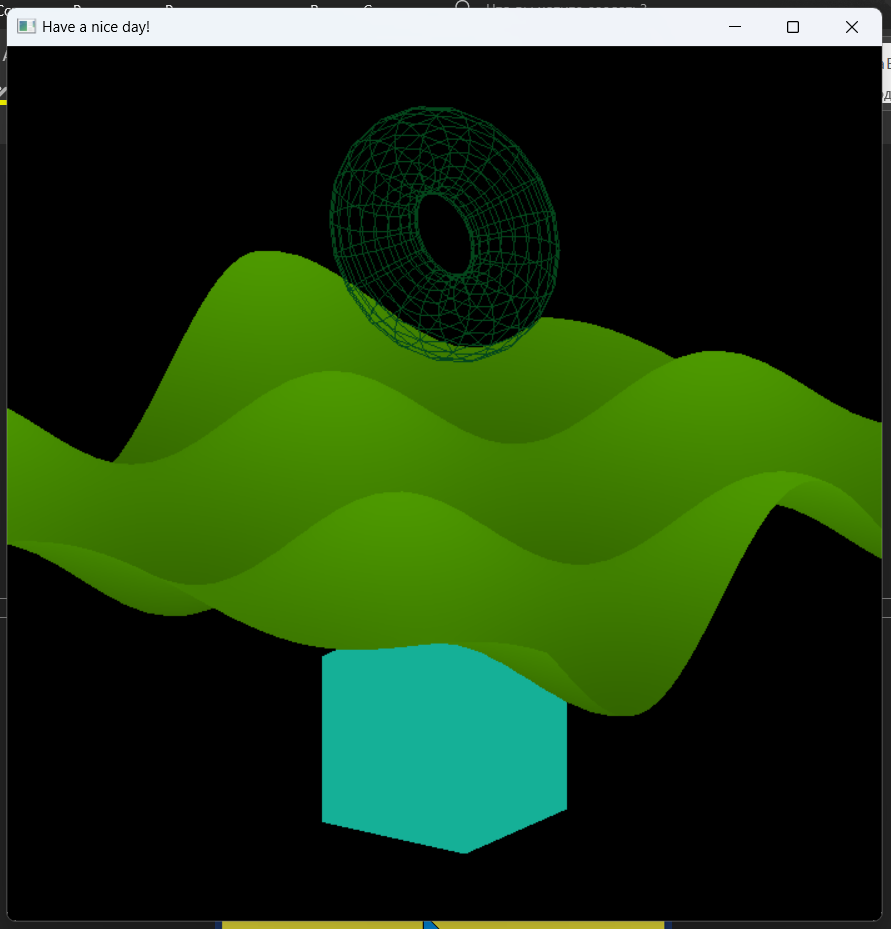


Рисунок 4 — Ортогональна проекція + інший кут огляду.

1. **Лістинг коду.**

#include <GL/glut.h>  
#include <cmath>  
#include <vector>  
  
using namespace std;  
  
const int WINDOW\_WIDTH = 800;  
const int WINDOW\_HEIGHT = 800;  
const float MOVE\_SPEED = 0.08f;  
  
const float INNER\_THORUS\_R = 0.5;  
const float OUTER\_THORUS\_R = 1;  
const float CUBE\_SIDE\_SIZE = 2;  
const float FIGURES\_DISTANCE = 3;  
  
const int GRID\_SIZE = 50;  
const float GRID\_STEP = 0.2f;  
  
float cameraAngleX = 0.0f;  
float cameraAngleY = 0.0f;  
float cameraDistance = 15.0f;  
bool isPerspective = true;  
  
vector planeColor = {  
 128,  
 255,  
 0,  
};  
  
vector thorusColor = {  
 4,  
 71,  
 28,  
};  
  
vector cubeColor = {  
 21,  
 176,  
 151,  
};  
  
float normRGB(const int RGBvalue) {  
 return RGBvalue / 255.0f;  
}  
  
void drawPlane() {  
 glEnableClientState(GL\_VERTEX\_ARRAY);  
 glEnableClientState(GL\_COLOR\_ARRAY);  
  
 vector<float> vertices;  
 vector<float> colors;  
 vector<int> indices;  
  
 for (int x = 0; x <= GRID\_SIZE; ++x) {  
 for (int y = 0; y <= GRID\_SIZE; ++y) {  
 float posX = (x - GRID\_SIZE / 2.0f) \* GRID\_STEP;  
 float posY = (y - GRID\_SIZE / 2.0f) \* GRID\_STEP;  
 float posZ = sin(posX) \* cos(posY);  
  
 vertices.push\_back(posX);  
 vertices.push\_back(posZ);  
 vertices.push\_back(posY);  
  
 float t = (posZ + (GRID\_SIZE / 2.0f) \* GRID\_STEP) / (GRID\_SIZE \* GRID\_STEP);  
 colors.push\_back(normRGB(planeColor[0]) \* t);  
 colors.push\_back(normRGB(planeColor[1]) \* t);  
 colors.push\_back(normRGB(planeColor[2]) \* t);  
 }  
 }  
  
 for (int x = 0; x < GRID\_SIZE; ++x) {  
 for (int y = 0; y < GRID\_SIZE; ++y) {  
 int topLeft = x \* (GRID\_SIZE + 1) + y;  
 int topRight = topLeft + 1;  
 int bottomLeft = topLeft + GRID\_SIZE + 1;  
 int bottomRight = bottomLeft + 1;  
  
 indices.push\_back(topLeft);  
 indices.push\_back(bottomLeft);  
 indices.push\_back(bottomRight);  
  
 indices.push\_back(topLeft);  
 indices.push\_back(bottomRight);  
 indices.push\_back(topRight);  
 }  
 }  
  
 glVertexPointer(3, GL\_FLOAT, 0, vertices.data());  
 glColorPointer(3, GL\_FLOAT, 0, colors.data());  
 glDrawElements(GL\_TRIANGLES, indices.size(), GL\_UNSIGNED\_INT, indices.data());  
  
 glDisableClientState(GL\_VERTEX\_ARRAY);  
 glDisableClientState(GL\_COLOR\_ARRAY);  
}  
  
  
void drawSolidCube() {  
 glColor3f(normRGB(cubeColor[0]), normRGB(cubeColor[1]), normRGB(cubeColor[2]));  
 glutSolidCube(CUBE\_SIDE\_SIZE);  
}  
  
void drawWireTorus() {  
 glColor3f(normRGB(thorusColor[0]), normRGB(thorusColor[1]), normRGB(thorusColor[2]));  
 glutWireTorus(INNER\_THORUS\_R, OUTER\_THORUS\_R, 20, 20);  
}  
  
void setCamera() {  
 float cameraX = cameraDistance \* sin(cameraAngleX) \* cos(cameraAngleY);  
 float cameraY = cameraDistance \* sin(cameraAngleY);  
 float cameraZ = cameraDistance \* cos(cameraAngleX) \* cos(cameraAngleY);  
  
 gluLookAt(cameraX, cameraY, cameraZ, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);  
}  
  
void changeProjection() {  
 glMatrixMode(GL\_PROJECTION);  
 glLoadIdentity();  
  
 float sceneSize = FIGURES\_DISTANCE + max(CUBE\_SIDE\_SIZE, OUTER\_THORUS\_R \* 2);  
 if (isPerspective) {  
 gluPerspective(45.0, 1.0, 0.1, sceneSize \* 4);  
 } else {  
 glOrtho(-sceneSize, sceneSize, -sceneSize, sceneSize, -sceneSize \* 2, sceneSize \* 4);  
 }  
  
 glMatrixMode(GL\_MODELVIEW);  
 glutPostRedisplay();  
}  
  
void display() {  
 glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);  
 glLoadIdentity();  
 setCamera();  
  
 glPushMatrix();  
 glTranslatef(0, -FIGURES\_DISTANCE, 0);  
 drawSolidCube();  
 glPopMatrix();  
  
 glPushMatrix();  
 drawPlane();  
 glPopMatrix();  
  
 glPushMatrix();  
 glTranslatef(0, FIGURES\_DISTANCE, 0);  
 drawWireTorus();  
 glPopMatrix();  
  
 glutSwapBuffers();  
}  
  
void keyboard(const unsigned char key, int x, int y) {  
 if (key == 'p' || key == 'P') {  
 isPerspective = true;  
 changeProjection();  
 } else if (key == 'o' || key == 'O') {  
 isPerspective = false;  
 changeProjection();  
 }  
  
 if (key == 'a' || key == 'A')  
 cameraAngleX -= MOVE\_SPEED;  
 if (key == 'w' || key == 'W')  
 cameraAngleY = min(cameraAngleY + MOVE\_SPEED, 1.5f);  
 if (key == 'd' || key == 'D')  
 cameraAngleX += MOVE\_SPEED;  
 if (key == 's' || key == 'S')  
 cameraAngleY = max(cameraAngleY - MOVE\_SPEED, -1.5f);  
 glutPostRedisplay();  
}  
  
void specialKeys(const int key, int x, int y) {  
 if (key == GLUT\_KEY\_LEFT)  
 cameraAngleX -= MOVE\_SPEED;  
 if (key == GLUT\_KEY\_RIGHT)  
 cameraAngleX += MOVE\_SPEED;  
 if (key == GLUT\_KEY\_UP)  
 cameraAngleY = min(cameraAngleY + MOVE\_SPEED, 1.5f);  
 if (key == GLUT\_KEY\_DOWN)  
 cameraAngleY = max(cameraAngleY - MOVE\_SPEED, -1.5f);  
 glutPostRedisplay();  
}  
  
int main(int argc, char\*\* argv) {  
 glutInit(&argc, argv);  
 glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);  
 glutInitWindowSize(WINDOW\_WIDTH, WINDOW\_HEIGHT);  
 glutCreateWindow("Have a nice day!");  
  
 glEnable(GL\_DEPTH\_TEST);  
  
 glutDisplayFunc(display);  
 glutKeyboardFunc(keyboard);  
 glutSpecialFunc(specialKeys);  
  
 changeProjection();  
 glutMainLoop();  
  
 return 0;  
}