**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»**

**ІКНІ**

Кафедра **ПЗ**



**ЗВІТ**

до лабораторної роботи №6

**на тему:** *“Побудова скрінсейвера з використанням графічної бібліотеки OpenGL”*

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

**Лектор:**

проф. каф. ПЗ

Журавчак Л.М.

**Виконав:**

ст. гр. ПЗ-33

Юшкевич. А.І.

**Прийняв:**

проф. каф. ПЗ

Журавчак Л.М.

«\_\_» \_\_\_\_\_\_\_\_\_ 2025р.

∑=\_\_\_\_\_

Львів – 2025

**Тема:** Побудова скрінсейвера з використанням графічної бібліотеки OpenGL.

**Мета:**  Навчитися створювати програми, що імітують роботу скрінсейвера, з можливістю зміни параметрів руху об’єктів у діалозі на основі використання засобів бібліотеки OpenGL.

**Теоретичні відомості**

За допомогою функцій додаткової бібліотеки GLUT в OpenGL можна створювати меню для організації взаємодії між програмою та користувачем. Спливаюче меню створюють за допомогою спеціального оператора **glutCreateMenu(menuFcn)**, параметром якої є ім’я процедури, що буде викликана після вибору однієї з команд меню **void menuFcn(int menuItemNumber)**, де через параметр **menuItemNumber** передають номер вибраної у меню команди.

Для задання команд, перерахованих у меню, використовують функцію **glutAddMenuEntry(charString, menuItemNumber)**, де перший параметр задає текст, який буде відображатись в меню, а другий дає його позицію в меню. Кнопку миші, яка буде використана для вибору команди меню, задають оператором **glutAttachMenu(button)**, де параметр може набувати одну з трьох символьних констант GLUT, що визначають ліву (**GLUT\_LEFT\_BUTTON**), середню чи праву кнопки миші.

Щоб активізувати будь-яке меню для поточного вікна на екрані дисплея, використовують оператор **glutSetMenu(menuID)**, де **menuID**=**glutCreateMenu (menuFcn)**. Видаляють меню за допомогою **glutDestroyMenu(menuID)**.

Отож, у загальному випадку код для створення контекстного меню виглядає так:

**void main()**

**{**

**…**

**glutCreateMenu(menuFcn);**

**glutAddMenuEntry(<назва команди меню 1>, 1);**

**…**

**glutAddMenuEntry(<назва команди меню N>, <N>);**

**glutAttachMenu(GLUT\_RIGHT\_BUTTON);**

**…**

**}**

**void menuFcn(int choice)**

**{**

**…**

**switch(choice)**

**{**

**case 1:     // визначення коду для команди меню 1**

**…;**

**break;**

**…**

**case <N>:   // визначення коду для команди меню <N>**

**…;**

**break;**

**default:**

**…**

**}**

**…**

**}**

# Завдання

Базуючись на завданнях лабораторної роботи № 5, доповнити їх неперервним переміщенням об’єктів за **ламано-прямолінійною тривимірною траєкторією** з відбиванням від **усіх шести обмежувальних площин** сцени.

За допомогою **контекстного меню** організувати діалог з користувачем, у якому надати можливість змінювати швидкості обертання та переміщення об’єктів, вмикати та вимикати джерела освітлення (за замовчуванням сцена має бути освітлена одним джерелом). Передбачити глобальне фонове освітлення, урахування внесків дзеркального і недзеркального кольорів та їх об'єднання.

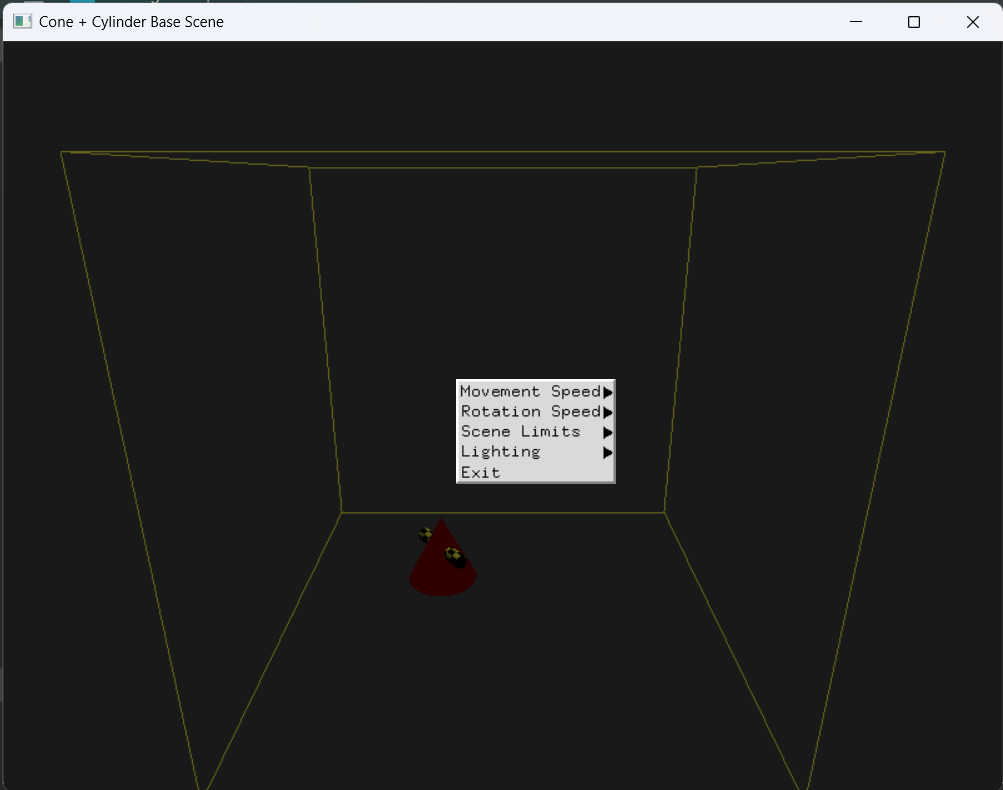
 Завдання повинні бути реалізовані або у **консольному** **вікні** з повноекранним режимом відображення, або як справжні скрінсейвери (файли з розширенням **.scr**).

12. Намалювати **об’єкт, складений із червоного конуса, бічна поверхня якого пронизана циліндром, покритим текстурою шахової дошки**. Реалізувати обертання об’єкта навколо осі, що проходить через основу та вершину конуса.

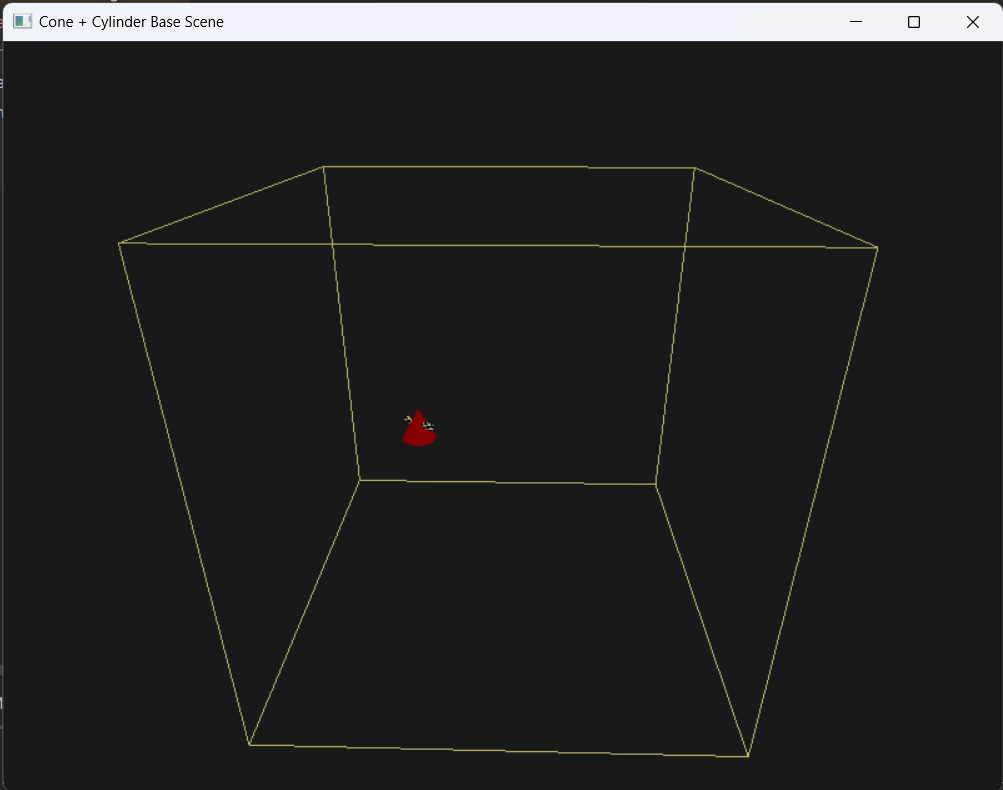
# Код програми

#include <GL/glut.h>  
#include <cmath>  
#include <iostream>  
#include <ostream>  
  
#define TEX\_WIDTH 64  
#define TEX\_HEIGHT 64  
GLubyte checkerTexture[TEX\_HEIGHT][TEX\_WIDTH][3]; // 3 канали (RGB)  
GLuint textureID;  
  
  
float rotationAngle = 0.0f;  
  
float sceneLimit = 10.0f; // межі +- по кожній осі  
float rotationSpeed = 0.5f;  
float movementSpeed = 0.05f;  
  
struct MovingObject {  
 float x, y, z;  
 float dx, dy, dz;  
 float r, g, b;  
  
 void update() {  
 x += dx \* movementSpeed;  
 y += dy \* movementSpeed;  
 z += dz \* movementSpeed;  
  
 if (x > sceneLimit || x < -sceneLimit) dx \*= -1;  
 if (y > sceneLimit || y < -sceneLimit) dy \*= -1;  
 if (z > sceneLimit || z < -sceneLimit) dz \*= -1;  
 }  
  
 void applyColor() {  
 glColor3f(r, g, b);  
 }  
};  
  
MovingObject coneObj = {0, 0, 0, 1, 1, 0.5f, 1, 0, 0};  
MovingObject cylObj = {0, 0, 0, 1, 1, 0.5f, 1, 1, 1};  
  
struct transform {  
 GLdouble yaw = 0.0f;  
 GLdouble pitch = 20.0f;  
 GLdouble zoom = 10.0f;  
 int leftX = 0, leftY = 0;  
 int rightX = 0, rightY = 0;  
 int dx = 0, dy = 1;  
 bool isDraggingLeft = false;  
 bool isDraggingRight = false;  
 float camX = 3;  
 float camY = 3;  
 float camZ = 5;  
 float panOffsetX = 0.0f;  
 float panOffsetY = 0.0f;  
 float panOffsetZ = 0.0f;  
} SceneTransform;  
  
bool ambient = false;  
bool directional = false;  
bool point = false;  
bool spot = true;  
  
float pointLightX = 2.0f;  
float spotLightX = 0.0f;  
  
bool enableFog = true;  
GLint fogMode = GL\_LINEAR; // Можна змінювати на GL\_EXP або GL\_EXP2  
GLfloat fogColor[4] = {0.1f, 0.1f, 0.1f, 1.0f};  
  
void generateCheckerTexture() {  
 for (int i = 0; i < TEX\_HEIGHT; i++) {  
 for (int j = 0; j < TEX\_WIDTH; j++) {  
 int checker = ((i / 8) % 2) ^ ((j / 8) % 2); // чергуємо квадрати  
 GLubyte color = checker ? 255 : 0; // чорний/білий  
 checkerTexture[i][j][0] = color;  
 checkerTexture[i][j][1] = color;  
 checkerTexture[i][j][2] = color;  
 }  
 }  
  
 glGenTextures(1, &textureID);  
 glBindTexture(GL\_TEXTURE\_2D, textureID);  
  
 glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);  
 glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);  
  
 glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGB, TEX\_WIDTH, TEX\_HEIGHT, 0,  
 GL\_RGB, GL\_UNSIGNED\_BYTE, checkerTexture);  
}  
  
void initLighting() {  
 glEnable(GL\_LIGHTING);  
 glEnable(GL\_COLOR\_MATERIAL); // Дозволяє фарбувати через glColor  
 glColorMaterial(GL\_FRONT, GL\_AMBIENT\_AND\_DIFFUSE);  
  
 glEnable(GL\_NORMALIZE); // Нормалізує нормалі для коректного освітлення  
}  
  
void initFog() {  
 glEnable(GL\_FOG);  
 glFogi(GL\_FOG\_MODE, fogMode); // Лінійний або експоненційний  
 glFogfv(GL\_FOG\_COLOR, fogColor); // Колір туману — як фон  
 glFogf(GL\_FOG\_DENSITY, 0.4f); // Для EXP/EXP2  
 glHint(GL\_FOG\_HINT, GL\_NICEST); // Найвища якість  
 glFogf(GL\_FOG\_START, 0.4f); // Для лінійного туману  
 glFogf(GL\_FOG\_END, 0.0f); // Для лінійного туману  
}  
  
void init() {  
 glEnable(GL\_DEPTH\_TEST);  
 glClearColor(0.1f, 0.1f, 0.1f, 1.0f); // Темний фон  
  
 generateCheckerTexture();  
 glEnable(GL\_TEXTURE\_2D); // Увімкнути текстури  
  
 initFog();  
  
 initLighting();  
}  
  
void setupAmbientLight() {  
 if (ambient) {  
 GLfloat ambientColor[] = { 0.3f, 0.3f, 0.3f, 1.0f };  
 glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, ambientColor);  
 } else {  
 GLfloat ambientColor[] = { 0.0f, 0.0f, 0.0f, 1.0f };  
 glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, ambientColor);  
 }  
}  
  
  
void setupDirectionalLight() {  
 if (directional) {  
 glEnable(GL\_LIGHT0);  
 GLfloat lightColor[] = { 0.9f, 0.9f, 0.9f, 1.0f };  
 GLfloat lightDir[] = { -1.0f, -1.0f, -1.0f, 0.0f };  
  
 glLightfv(GL\_LIGHT0, GL\_DIFFUSE, lightColor);  
 glLightfv(GL\_LIGHT0, GL\_POSITION, lightDir);  
 } else {  
 glDisable(GL\_LIGHT0);  
 }  
}  
  
void setupPointLight() {  
 if (point) {  
 glEnable(GL\_LIGHT1);  
 GLfloat lightColor[] = { 1.0f, 0.8f, 0.8f, 1.0f };  
 GLfloat lightPos[] = { pointLightX, 10.0f, 2.0f, 1.0f };  
  
 glLightfv(GL\_LIGHT1, GL\_DIFFUSE, lightColor);  
 glLightfv(GL\_LIGHT1, GL\_POSITION, lightPos);  
  
 // Реалістичне загасання (наприклад, до 50 юнітів)  
 glLightf(GL\_LIGHT1, GL\_CONSTANT\_ATTENUATION, 1.0f);  
 glLightf(GL\_LIGHT1, GL\_LINEAR\_ATTENUATION, 0.09f);  
 glLightf(GL\_LIGHT1, GL\_QUADRATIC\_ATTENUATION, 0.032f);  
 } else {  
 glDisable(GL\_LIGHT1);  
 }  
}  
  
  
void setupSpotlight() {  
 if (spot) {  
 glEnable(GL\_LIGHT2);  
 GLfloat lightColor[] = { 0.4f, 0.4f, 0.1f, 0.01f };  
 GLfloat lightPos[] = { spotLightX, 10.0f, 0.0f, 0.0f };  
 GLfloat spotDir[] = { 0.0f, -1.0f, 0.0f };  
  
 glLightfv(GL\_LIGHT2, GL\_DIFFUSE, lightColor);  
 glLightfv(GL\_LIGHT2, GL\_POSITION, lightPos);  
 glLightfv(GL\_LIGHT2, GL\_SPOT\_DIRECTION, spotDir);  
 glLightf(GL\_LIGHT2, GL\_SPOT\_CUTOFF, 30.0f);  
 glLightf(GL\_LIGHT2, GL\_SPOT\_EXPONENT, 10.0f);  
  
 // Більш реалістичне загасання  
 glLightf(GL\_LIGHT2, GL\_CONSTANT\_ATTENUATION, 1.0f);  
 glLightf(GL\_LIGHT2, GL\_LINEAR\_ATTENUATION, 0.09f);  
 glLightf(GL\_LIGHT2, GL\_QUADRATIC\_ATTENUATION, 0.032f);  
 } else {  
 glDisable(GL\_LIGHT2);  
 }  
}  
  
  
void drawCone(MovingObject& obj) {  
 GLUquadric\* quad = gluNewQuadric();  
 gluQuadricNormals(quad, GLU\_SMOOTH);  
  
 glPushMatrix();  
 glColor3f(1.0f, 0.0f, 0.0f); // Червоний  
  
 glTranslatef(obj.x, obj.y, obj.z);  
 glRotatef(-90, 1, 0, 0); // Щоб стояв вертикально  
 gluCylinder(quad, 1.0, 0.0, 2.0, 32, 32);  
 glPopMatrix();  
  
 gluDeleteQuadric(quad);  
}  
  
void applyLightning() {  
 setupAmbientLight();  
 setupDirectionalLight();  
 setupPointLight();  
 setupSpotlight();  
  
 GLfloat diffuse[] = {0.7f, 0.7f, 0.7f, 1.0f};  
 GLfloat specular[] = {1.0f, 1.0f, 1.0f, 1.0f};  
 glMaterialfv(GL\_FRONT, GL\_DIFFUSE, diffuse);  
 glMaterialfv(GL\_FRONT, GL\_SPECULAR, specular);  
 glMaterialf(GL\_FRONT, GL\_SHININESS, 30.0f);  
}  
  
void drawCylinder(MovingObject& obj) {  
 GLUquadric\* quad = gluNewQuadric();  
 gluQuadricNormals(quad, GLU\_SMOOTH);  
 gluQuadricTexture(quad, GL\_TRUE); // Дозволити текстуру  
  
 glPushMatrix();  
  
 glEnable(GL\_TEXTURE\_2D);  
 glBindTexture(GL\_TEXTURE\_2D, textureID);  
  
 glColor3f(1.0f, 1.0f, 1.0f);  
  
 glTranslatef(obj.x, obj.y + 1.0f, obj.z - 1.0f);  
 glRotatef(90, 0, 0, 1);  
 glTranslatef(0.0f, 0.0f, 1.0f);  
  
 glRotatef(rotationAngle, 1.0f, 0.0f, 0.0f); // Обертання об’єкта  
 glTranslatef(0.0f, 0.0f, -1.0f);  
  
 gluCylinder(quad, 0.2, 0.2, 2.0, 32, 32);  
  
 glDisable(GL\_TEXTURE\_2D);  
  
 glPopMatrix();  
 gluDeleteQuadric(quad);  
}  
  
void drawCube() {  
 glPushMatrix();  
 glBegin(GL\_LINE\_LOOP);  
 glVertex3f(sceneLimit, -sceneLimit, -sceneLimit);  
 glVertex3f(sceneLimit, -sceneLimit, sceneLimit);  
 glVertex3f(sceneLimit, sceneLimit, sceneLimit);  
 glVertex3f(sceneLimit, sceneLimit, -sceneLimit);  
 glEnd();  
 glBegin(GL\_LINE\_LOOP);  
 glVertex3f(-sceneLimit, -sceneLimit, -sceneLimit);  
 glVertex3f(-sceneLimit, -sceneLimit, sceneLimit);  
 glVertex3f(-sceneLimit, sceneLimit, sceneLimit);  
 glVertex3f(-sceneLimit, sceneLimit, -sceneLimit);  
 glEnd();  
 glBegin(GL\_LINE\_LOOP);  
 glVertex3f(-sceneLimit, sceneLimit, -sceneLimit);  
 glVertex3f(-sceneLimit, sceneLimit, sceneLimit);  
 glVertex3f(sceneLimit, sceneLimit, sceneLimit);  
 glVertex3f(sceneLimit, sceneLimit, -sceneLimit);  
 glEnd();  
 glBegin(GL\_LINE\_LOOP);  
 glVertex3f(-sceneLimit, -sceneLimit, -sceneLimit);  
 glVertex3f(-sceneLimit, -sceneLimit, sceneLimit);  
 glVertex3f(sceneLimit, -sceneLimit, sceneLimit);  
 glVertex3f(sceneLimit, -sceneLimit, -sceneLimit);  
 glEnd();  
 glBegin(GL\_LINE\_LOOP);  
 glVertex3f(-sceneLimit, -sceneLimit, sceneLimit);  
 glVertex3f(sceneLimit, -sceneLimit, sceneLimit);  
 glVertex3f(sceneLimit, sceneLimit, sceneLimit);  
 glVertex3f(-sceneLimit, sceneLimit, sceneLimit);  
 glEnd();  
 glBegin(GL\_LINE\_LOOP);  
 glVertex3f(-sceneLimit, -sceneLimit, -sceneLimit);  
 glVertex3f(sceneLimit, -sceneLimit, -sceneLimit);  
 glVertex3f(sceneLimit, sceneLimit, -sceneLimit);  
 glVertex3f(-sceneLimit, sceneLimit, -sceneLimit);  
 glEnd();  
 glPopMatrix();  
}  
  
void updateCamera() {  
 glLoadIdentity();  
  
 float centerX = SceneTransform.camX;  
 float centerY = SceneTransform.camY;  
 float centerZ = SceneTransform.camZ;  
  
 float targetX = SceneTransform.dx;  
 float targetY = SceneTransform.dy;  
 float targetZ = 0.0f;  
  
 // Враховуємо панорамування  
 centerX += SceneTransform.panOffsetX;  
 centerY += SceneTransform.panOffsetY;  
 centerZ += SceneTransform.panOffsetZ;  
 targetX += SceneTransform.panOffsetX;  
 targetY += SceneTransform.panOffsetY;  
 targetZ += SceneTransform.panOffsetZ;  
  
 gluLookAt(centerX, centerY, centerZ,  
 targetX, targetY, targetZ,  
 0.0, 1.0, 0.0);  
}  
  
  
void display() {  
 glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);  
 glLoadIdentity();  
  
 updateCamera();  
  
 applyLightning();  
  
 drawCone(coneObj);  
 drawCylinder(cylObj);  
 drawCube();  
  
 glutSwapBuffers();  
}  
  
void reshape(int w, int h) {  
 glViewport(0, 0, w, h);  
 glMatrixMode(GL\_PROJECTION);  
 glLoadIdentity();  
 gluPerspective(60.0, static\_cast<float>(w) / h, 1.0, 100.0);  
 glMatrixMode(GL\_MODELVIEW);  
}  
  
void timer(int value) {  
 rotationAngle += rotationSpeed;  
 if (rotationAngle > 360.0f)  
 rotationAngle -= 360.0f;  
  
 coneObj.update();  
 cylObj.update();  
  
 glutPostRedisplay();  
 glutTimerFunc(16, timer, 0); // ~60 FPS  
}  
  
void menu(int item) {  
 switch (item) {  
 case 1: movementSpeed = 0.01f; break;  
 case 2: movementSpeed = 0.05f; break;  
 case 3: movementSpeed = 0.1f; break;  
 case 4: rotationSpeed = 1.0f; break;  
 case 5: rotationSpeed = 0.5f; break;  
 case 6: rotationSpeed = 0.0f; break;  
 case 7: sceneLimit = 5.0f; break;  
 case 8: sceneLimit = 10.0f; break;  
 case 9: sceneLimit = 20.0f; break;  
 case 11: ambient = !ambient; break;  
 case 12: directional = !directional; break;  
 case 13: point = !point; break;  
 case 14: spot = !spot; break;  
 case 10: exit(0); break;  
 }  
 glutPostRedisplay();  
}  
  
void createMenu() {  
 int moveSpeedMenu = glutCreateMenu(menu);  
 glutAddMenuEntry("0.01", 1);  
 glutAddMenuEntry("0.05", 2);  
 glutAddMenuEntry("0.1", 3);  
  
 int rotSpeedMenu = glutCreateMenu(menu);  
 glutAddMenuEntry("1.0", 4);  
 glutAddMenuEntry("0.5", 5);  
 glutAddMenuEntry("0.0", 6);  
  
 int sceneSizeMenu = glutCreateMenu(menu);  
 glutAddMenuEntry("Cube +-5", 7);  
 glutAddMenuEntry("Cube +-10", 8);  
 glutAddMenuEntry("Cube +-20", 9);  
  
 int lightMenu = glutCreateMenu(menu);  
 glutAddMenuEntry("Toggle Ambient", 11);  
 glutAddMenuEntry("Toggle Directional", 12);  
 glutAddMenuEntry("Toggle Point", 13);  
 glutAddMenuEntry("Toggle Spot", 14);  
  
 glutCreateMenu(menu);  
 glutAddSubMenu("Movement Speed", moveSpeedMenu);  
 glutAddSubMenu("Rotation Speed", rotSpeedMenu);  
 glutAddSubMenu("Scene Limits", sceneSizeMenu);  
 glutAddSubMenu("Lighting", lightMenu);  
 glutAddMenuEntry("Exit", 10);  
  
 glutAttachMenu(GLUT\_RIGHT\_BUTTON);  
}  
  
void updateCoordinates() {  
 float camX = SceneTransform.zoom \* cosf(SceneTransform.pitch \* M\_PI / 180.f) \* sinf(SceneTransform.yaw \* M\_PI / 180.f);  
 float camY = SceneTransform.zoom \* sinf(SceneTransform.pitch \* M\_PI / 180.f);  
 float camZ = SceneTransform.zoom \* cosf(SceneTransform.pitch \* M\_PI / 180.f) \* cosf(SceneTransform.yaw \* M\_PI / 180.f);  
  
 SceneTransform.camX = camX;  
 SceneTransform.camY = camY;  
 SceneTransform.camZ = camZ;  
  
 SceneTransform.dx = 0.0f; // Центр обертання — (0, 0, 0)  
 SceneTransform.dy = 0.0f;  
}  
  
  
  
//  
// Обробка миші  
//  
void mouse(int button, int state, int x, int y) {  
 if (button == 3 && state == GLUT\_DOWN) { // scroll up  
 SceneTransform.zoom -= 0.5f;  
 if (SceneTransform.zoom < 2.f) SceneTransform.zoom = 2.f;  
 updateCoordinates();  
 glutPostRedisplay();  
 } else if (button == 4 && state == GLUT\_DOWN) { // scroll down  
 SceneTransform.zoom += 0.5f;  
 if (SceneTransform.zoom > 100.f) SceneTransform.zoom = 100.f;  
 updateCoordinates();  
 glutPostRedisplay();  
 }  
 if (button == GLUT\_LEFT\_BUTTON) {  
 if (state == GLUT\_DOWN) {  
 SceneTransform.isDraggingLeft = true;  
 SceneTransform.leftX = x;  
 SceneTransform.leftY = y;  
 } else {  
 SceneTransform.isDraggingLeft = false;  
 }  
 }else if (button == GLUT\_RIGHT\_BUTTON) {  
 if (state == GLUT\_DOWN) {  
 SceneTransform.isDraggingRight = true;  
 SceneTransform.rightX = x;  
 SceneTransform.rightY = y;  
 } else {  
 SceneTransform.isDraggingRight = false;  
 }  
 }  
}  
  
//  
// Обробка руху миші (drag-to-rotate)  
//  
void motion(int x, int y) {  
 if (SceneTransform.isDraggingLeft) {  
 float dx = x - SceneTransform.leftX;  
 float dy = y - SceneTransform.leftY;  
  
 SceneTransform.yaw -= dx \* 0.5f;  
 SceneTransform.pitch += dy \* 0.5f;  
 if (SceneTransform.pitch > 89.0f) SceneTransform.pitch = 89.0f;  
 if (SceneTransform.pitch < -89.0f) SceneTransform.pitch = -89.0f;  
  
 SceneTransform.leftX = x;  
 SceneTransform.leftY = y;  
  
 updateCoordinates();  
 glutPostRedisplay();  
 }  
 else if (SceneTransform.isDraggingRight) {  
 float dx = x - SceneTransform.rightX;  
 float dy = y - SceneTransform.rightY;  
  
 float panSpeed = 0.005f \* SceneTransform.zoom;  
  
 // yaw у радіанах  
 float yawRad = SceneTransform.yaw \* M\_PI / 180.0f;  
  
 // Вектор вправо (по yaw)  
 float rightX = cosf(yawRad);  
 float rightZ = -sinf(yawRad);  
  
 // Вектор вгору — простіше взяти по Y (можна ускладнити через pitch, але поки не треба)  
 float upY = 1.0f;  
  
 // Зсув у світових координатах  
 SceneTransform.panOffsetX += -dx \* panSpeed \* rightX;  
 SceneTransform.panOffsetZ += -dx \* panSpeed \* rightZ;  
 SceneTransform.panOffsetY += dy \* panSpeed \* upY;  
  
 SceneTransform.rightX = x;  
 SceneTransform.rightY = y;  
  
 glutPostRedisplay();  
 }  
}  
  
void keyboard(unsigned char key, int x, int y) {  
 switch (key) {  
 case 'r':  
 SceneTransform.panOffsetX = 0.0f;  
 SceneTransform.panOffsetY = 0.0f;  
 SceneTransform.panOffsetZ = 0.0f;  
 SceneTransform.camX = 3.0f;  
 SceneTransform.camX = 3.0f;  
 SceneTransform.camX = 10.0f;  
 SceneTransform.yaw = 0.0f;  
 SceneTransform.pitch = 20.0f;  
 SceneTransform.zoom = 10.0f;  
  
 updateCoordinates();  
 break;  
 case 'a':  
 ambient = !ambient;  
 std::cout << "ambient = " << ambient << std::endl;  
 break;  
 case 'd':  
 directional = !directional;  
 std::cout << "directional = " << directional << std::endl;  
 break;  
 case 'p':  
 point = !point;  
 std::cout << "point = " << point << std::endl;  
 break;  
 case 's':  
 spot = !spot;  
 std::cout << "spot = " << spot << std::endl;  
 break;  
 case 'f':  
 enableFog = !enableFog;  
 if (enableFog) glEnable(GL\_FOG);  
 else glDisable(GL\_FOG);  
 std::cout << "Fog: " << (enableFog ? "ON" : "OFF") << std::endl;  
 break;  
  
 case 'm':  
 if (fogMode == GL\_LINEAR)  
 fogMode = GL\_EXP;  
 else if (fogMode == GL\_EXP)  
 fogMode = GL\_EXP2;  
 else  
 fogMode = GL\_LINEAR;  
  
 glFogi(GL\_FOG\_MODE, fogMode);  
 std::cout << "Fog mode: " << (fogMode == GL\_LINEAR ? "LINEAR" : (fogMode == GL\_EXP ? "EXP" : "EXP2")) << std::endl;  
 break;  
 case 27:  
 exit(0);  
 }  
 glutPostRedisplay();  
}  
  
void specialKeys(int key, int x, int y) {  
 switch (key) {  
 case GLUT\_KEY\_LEFT:  
 pointLightX -= 0.2f;  
 spotLightX -= 0.2f;  
 break;  
 case GLUT\_KEY\_RIGHT:  
 pointLightX += 0.2f;  
 spotLightX += 0.2f;  
 break;  
 }  
 glutPostRedisplay();  
}  
  
int main(int argc, char\*\* argv) {  
 glutInit(&argc, argv);  
 glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);  
 glutInitWindowSize(800, 600);  
 glutCreateWindow("Cone + Cylinder Base Scene");  
  
 init();  
  
 createMenu();  
  
 glutDisplayFunc(display);  
 glutReshapeFunc(reshape);  
 glutKeyboardFunc(keyboard);  
 glutSpecialFunc(specialKeys);  
 glutTimerFunc(0, timer, 0);  
 glutMouseFunc(mouse);  
 glutMotionFunc(motion);  
  
 glutMainLoop();  
 return 0;  
}

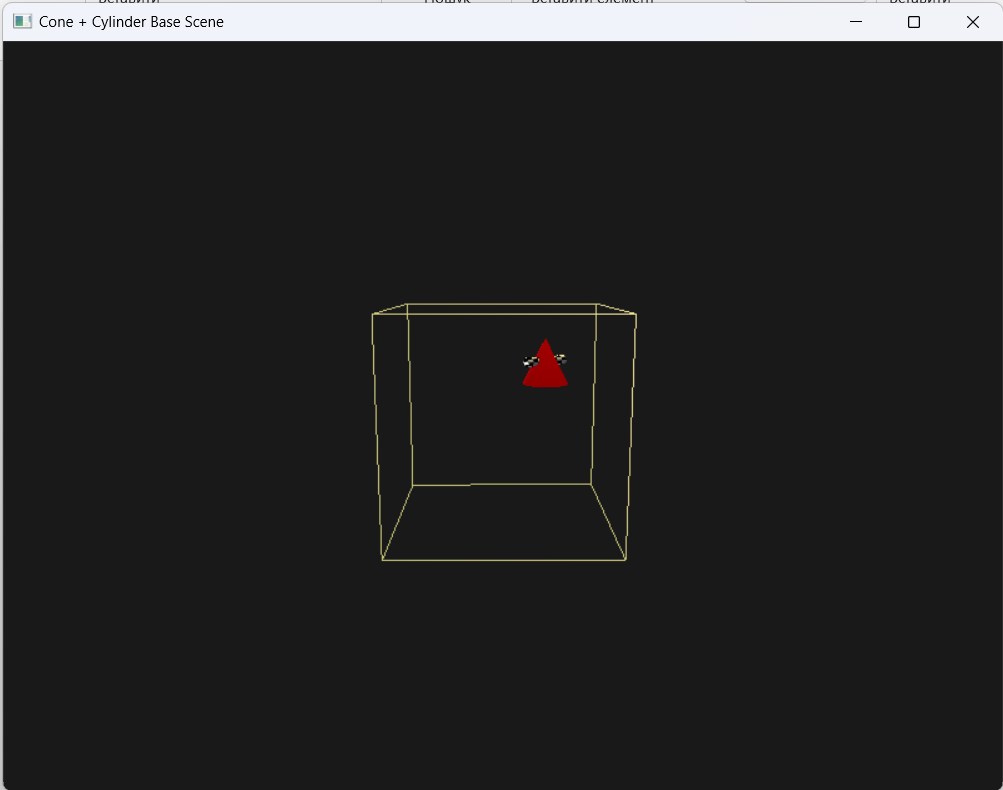
# Результат виконання програми

**

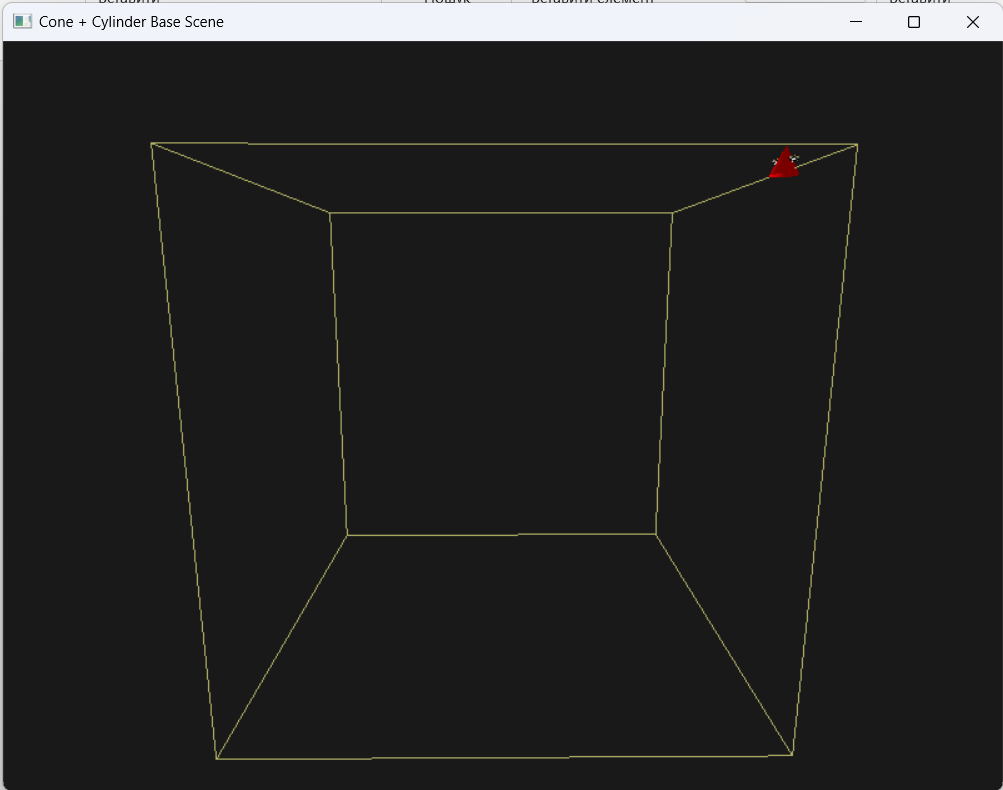
*Рис. 1. Рух конуса в межах квадрата. Виклик меню на праву кнопку миші*

**

*Рис. 2. Присутнє все освітлення*

**

*Рис. 3. Вдвічі менші кордони*

**

*Рис. 4. Вдвічі більші кордони*

**Висновки:** Під час виконання лабораторної роботи я навчився створювати програми, що імітують роботу скрінсейвера, з можливістю зміни параметрів руху об’єктів у діалозі на основі використання засобів бібліотеки OpenGL.