**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»**

**ІКНІ**

Кафедра **ПЗ**



**ЗВІТ**

до лабораторної роботи №5

**на тему:** *“Проектування динамічних тривимірних об’єктів на екран монітора з урахуванням освітлення та текстурних шаблонів”*

**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.

«\_\_» \_\_\_\_\_\_\_\_\_ 2024р.

∑=\_\_\_\_\_

Львів – 2024

**Тема:** Проектування динамічних тривимірних об’єктів на екран монітора з урахуванням освітлення та текстурних шаблонів.

**Мета:**  Навчитися будувати тривимірні об’єкти, покриті текстурою, з подальшим їхнім обертанням та масштабуванням на основі застосування базових геометричних перетворень OpenGL.

**Теоретичні відомості**

Для задання різних перетворень об’єктів сцени в OpenGL використовують операції над матрицями, при цьому розрізняють кілька типів матриць, зокрема вигляду і проекції. Перша матриця визначає перетворення об’єкта у світових координатах, такі як паралельний перенос, зміна масштабу і поворот. Матриця проекцій задає, як будуть проектуватися тривимірні об’єкти на площину екрана (у віконні координати). Для того, щоб вибрати матрицю для опрацювання, використовують команду **glMatrixMode(mode),** виклик якої зі значенням параметра **GL\_MODELVIEW** чи **GL\_PROJECTION**вмикає режими роботи з матрицею виду чи проекцій відповідно.

Робота з матрицями проекції та вигляду

Отож, щоб працювати з матрицею проекції, треба активувати відповідний режим командою **glMatrixMode(GL\_PROJECTION)**. Параметри ортогональної проекції вибирають за допомогою функції **glOrtho(…)**, для отримання перспективної проекції сцени використовують функцію **gluPerspective(…)** або **glFrustum(…)**. Якщо команда проектування не задана явно, OpenGL виконує ортогональну проекцію сцени за замовчуванням. Параметри спостереження задають за допомогою функції **gluLookAt(…)**.

Перед роботою з об’єктами необхідно задати режим роботи з матрицею вигляду, виконавши команду **glMatrixMode(GL\_MODELVIEW)**. Прямокутну область виведення результатів визначають функцією **glViewport(…)**. Для правильного відображення сцени у випадку зміни розмірів вікна здебільшого створюють спеціальну функцію, що має такий загальний вигляд:

**void resize(int width,int height)**

**{**

**glViewport(0,0,width,height);**

**glMatrixMode(GL\_PROJECTION);**

**glLoadIdentity();**

**glOrtho(…);**

**gluLookAt(…);**

**glMatrixMode(GL\_MODELVIEW);**

**}**

Тривимірні примітиви

Для створення тривимірних примітивів використовують відповідні функції додаткових бібліотек OpenGL – **gluSphere()**, **gluCylinder()**, **gluDisk()** із GLU, **glutSolidSphere()**, **glutSolidCube()**, **glutSolidCone()** тощо із GLUT, **auxSolidCube()**, **auxSolidBox()**, **auxSolidTorus()**, **auxSolidCylinder()**, **auxSolidCone()** тощо із GLAUX і т.д. Для коректної побудови цих об’єктів необхідно видаляти невидимі лінії та поверхні, увімкнувши відповідний режим командою**glEnable(GL\_DEPTH\_TEST)**.

Накладання текстури

Щоб створити повноцінну тривимірну сцену, часто на тривимірні об’єкти потрібно накладати зображення – для цього використовують текстури. У загальному випадку для накладання текстури на тривимірний об’єкт потрібно:

* увімкнути режим використання текстури (**glEnable(GL\_Texture\_2d)**);
* завантажити графічний файл (наприклад, функцією **auxDIBImageLoad** із GLAUX);
* створити ім’я-ідентифікатор текстури (**glGenTextures**);
* зробити його активним (**glBindTexture**);
* створити саму текстуру в пам’яті (**glTexImage2D**);
* установити параметри текстури (**glTexParameter**);
* установити параметри взаємодії текстури з об’єктом (**glTexEnv**);
* зв’язати координати текстури з об’єктом (**glTexCoord2d** – вручну, **glTexGen** – автоматично).

Освітлення

Важливою деталлю у тривимірних сценах є освітлення: без нього об’єкти здаватимуться плоскими. За замовчуванням освітлення вимкнуто, його потрібно увімкнути командою **glEnable(GL\_LIGHTING)**. Для створення реалістичних зображень необхідно визначити параметри матеріалу, з якого зроблено об’єкт, та кількість і властивості джерел світла. Властивості матеріалу задають за допомогою функції **glMaterial(…)**. Джерело освітлення вмикають командою **glEnable(GL\_LIGHTn)**, де n може набувати значень від 0 до 7, та задають його властивості за допомогою функції **glLight(…)**. Джерел можна використовувати кілька. Для визначення глобальних параметрів освітлення використовують команду **glLightModel(…)**.

# Завдання

Завдання повинні бути реалізовані у вікні з наявними елементами керування для запуску виконання окремих дій або етапів роботи. Для оформлення зовнішнього вигляду тривимірних об’єктів (окрім однотонного зафарбування) необхідно застосувати накладання текстури. Для формування текстур матеріалів (граніту, мармуру тощо) можна скористатися текстурним заповненням фігур у програмі Word.

Для освітлення сцени почергово передбачити рух чотирьох основних типів джерел світла: фонове освітлення, точкові, прожектори та віддалені. Реалістичність зображення забезпечити за допомогою функції радіального загасання.

Додати ефект туману, врахувавши лінійну та експоненційну функції загасання з глибиною.

**Програма має передбачати такі можливості:**

* виведення зображення вказаних у першій частині варіанта багатогранника або поверхні 2-го порядку як каркасних та з зафарбованими гранями (чи ділянками); у другому випадку передбачити плоске та інтерполяційне зафарбовування;
* формування та проектування на екран монітора вказаної у другій частині варіанта поверхні (Без’є, бі-сплайнової, NURBS) із вирізаним отвором для випадків каркасного мозаїчного та зафарбованого зображення;
* використання для обох фігур ортогональної та перспективної проекції.

Спроектувати на екран монітора NURBS-поверхню.

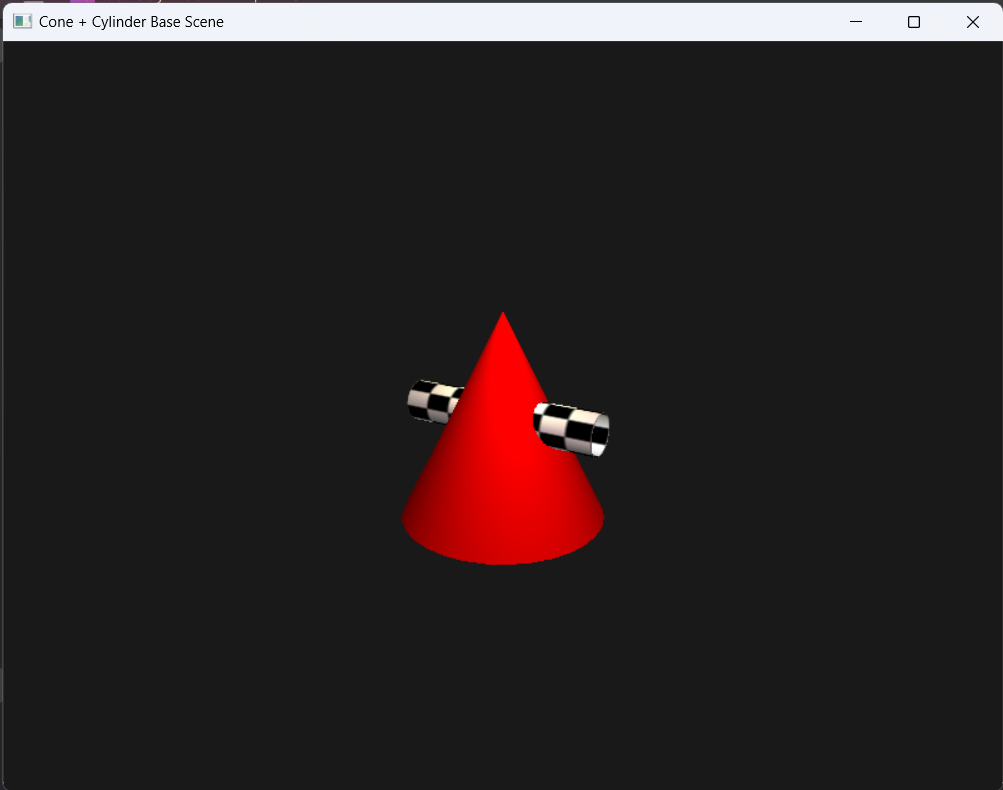
Спроектувати на екран монітора бі-сплайнову поверхню.

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.

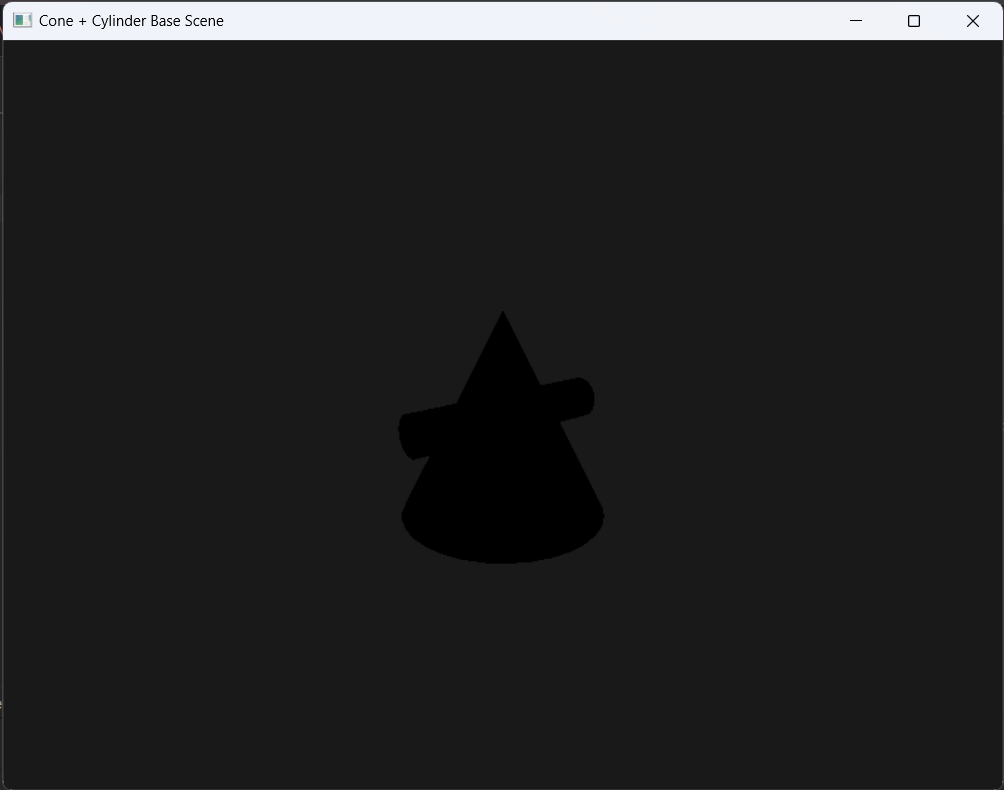
# Код програми

#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;  
  
  
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 = true;  
bool directional = true;  
bool point = true;  
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, 500.0f); // Для лінійного туману  
 glFogf(GL\_FOG\_END, 1000.0f); // 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, 3.0f, 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[] = { 1.0f, 1.0f, 0.6f, 1.0f };  
 GLfloat lightPos[] = { spotLightX, 3.0f, 0.0f, 1.0f };  
 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() {  
 GLUquadric\* quad = gluNewQuadric();  
 gluQuadricNormals(quad, GLU\_SMOOTH);  
  
 glPushMatrix();  
 glColor3f(1.0f, 0.0f, 0.0f); // Червоний  
 glRotatef(-90, 1, 0, 0); // Щоб стояв вертикально  
 gluCylinder(quad, 1.0, 0.0, 2.0, 32, 32);  
 glPopMatrix();  
  
 gluDeleteQuadric(quad);  
}  
  
void drawCylinder() {  
 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(0.0f, 1.0f, -1.0f);  
 glRotatef(90, 0, 0, 1);  
  
 gluCylinder(quad, 0.2, 0.2, 2.0, 32, 32);  
  
 glDisable(GL\_TEXTURE\_2D);  
  
 glPopMatrix();  
 gluDeleteQuadric(quad);  
}  
  
  
  
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();  
  
 setupAmbientLight();  
 setupDirectionalLight();  
 setupPointLight();  
 setupSpotlight();  
  
 glRotatef(rotationAngle, 0.0f, 1.0f, 0.0f); // Обертання об’єкта  
  
 drawCone();  
 drawCylinder();  
  
 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 += 0.5f;  
 if (rotationAngle > 360.0f)  
 rotationAngle -= 360.0f;  
  
 glutPostRedisplay();  
 glutTimerFunc(16, timer, 0); // ~60 FPS  
}  
  
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();  
 glutDisplayFunc(display);  
 glutReshapeFunc(reshape);  
 glutKeyboardFunc(keyboard);  
 glutSpecialFunc(specialKeys);  
 glutTimerFunc(0, timer, 0);  
 glutMouseFunc(mouse);  
 glutMotionFunc(motion);  
  
 glutMainLoop();  
 return 0;  
}

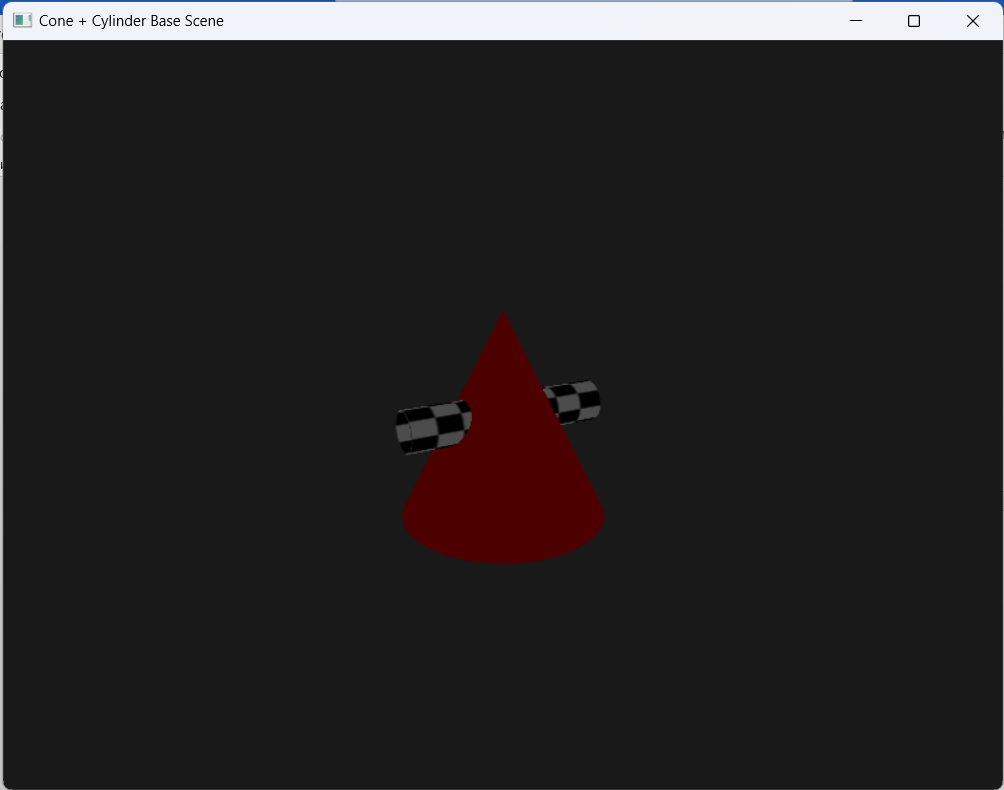
# Результат виконання програми

**

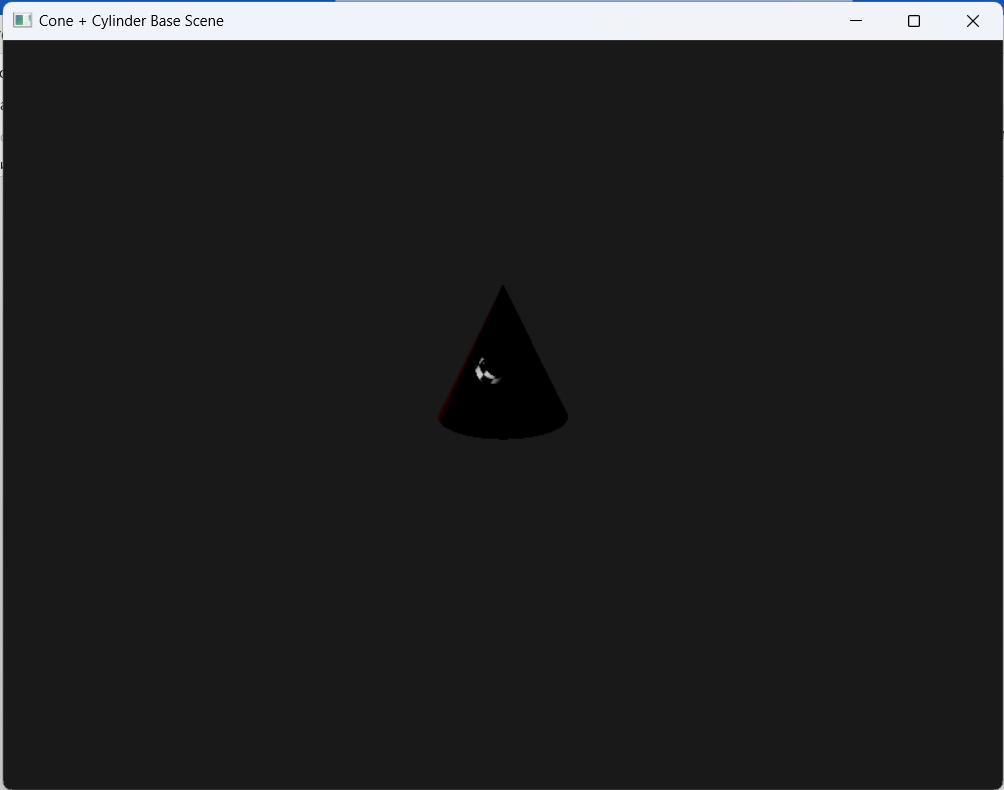
*Рис. 1. Конус червоного кольору, який перетинай циліндр із шаховою текстурою.*

**

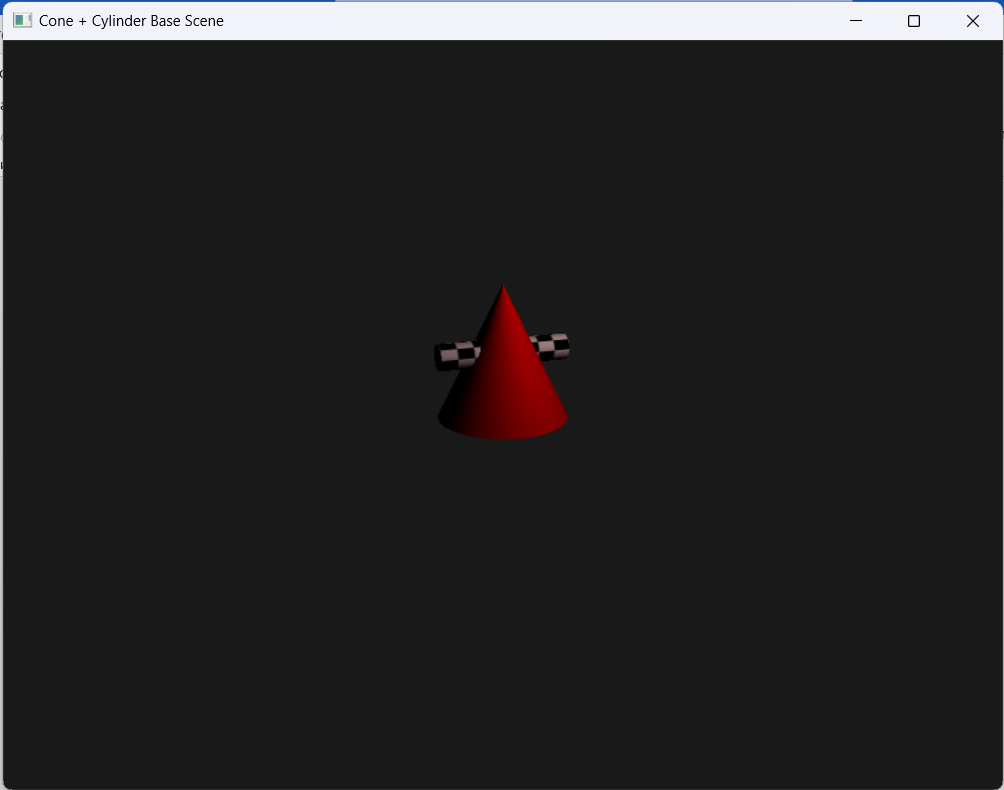
*Рис. 2. Відсутнє освітлення*

**

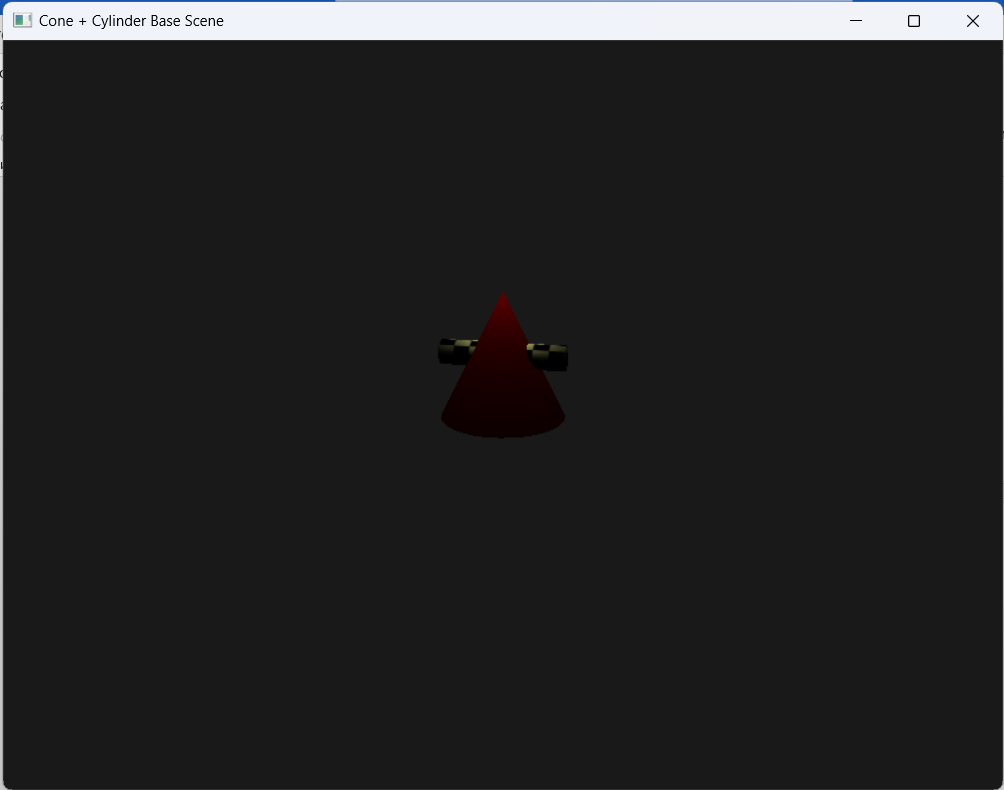
*Рис. 3. Присутнє лише фонове освітлення*

**

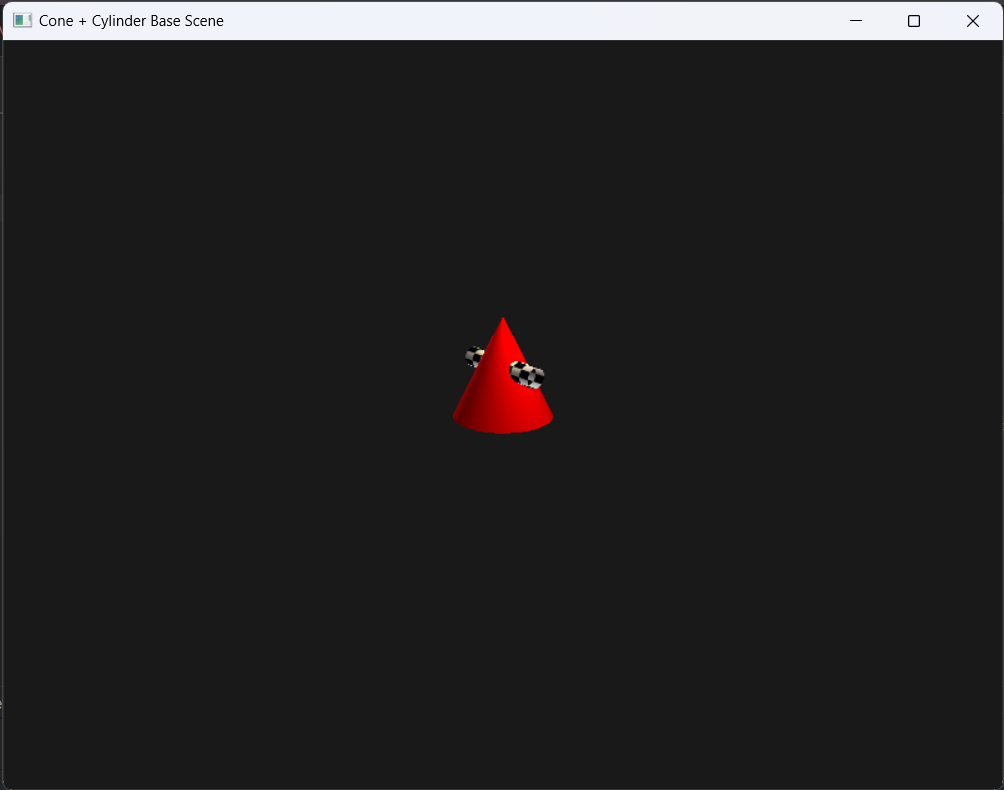
*Рис. 4. Присутнє лише віддалене освітлення*

**

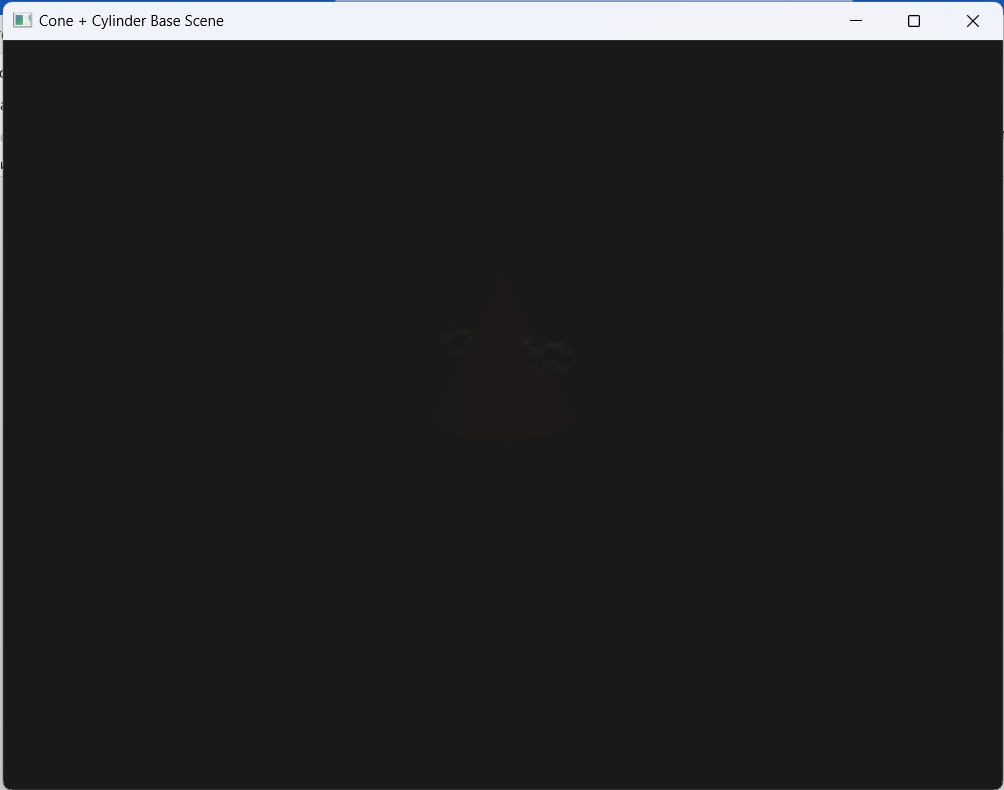
*Рис. 5. Присутнє лише точкове освітлення*

**

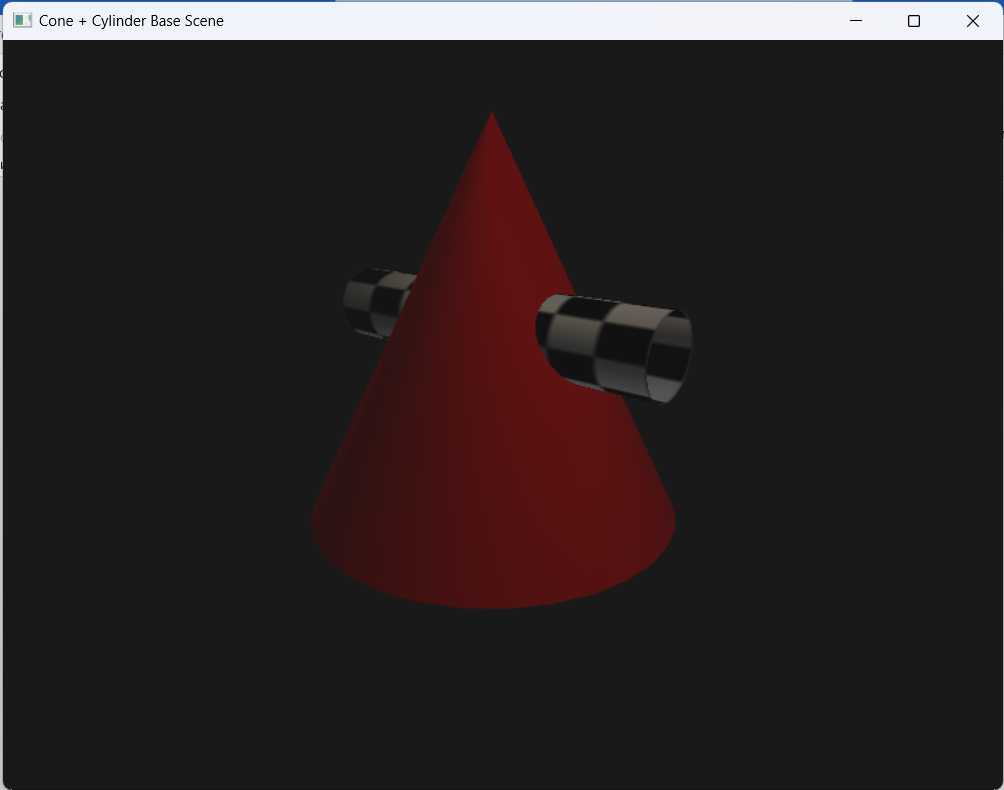
*Рис. 6. Присутнє лише прожекторне освітлення*

**

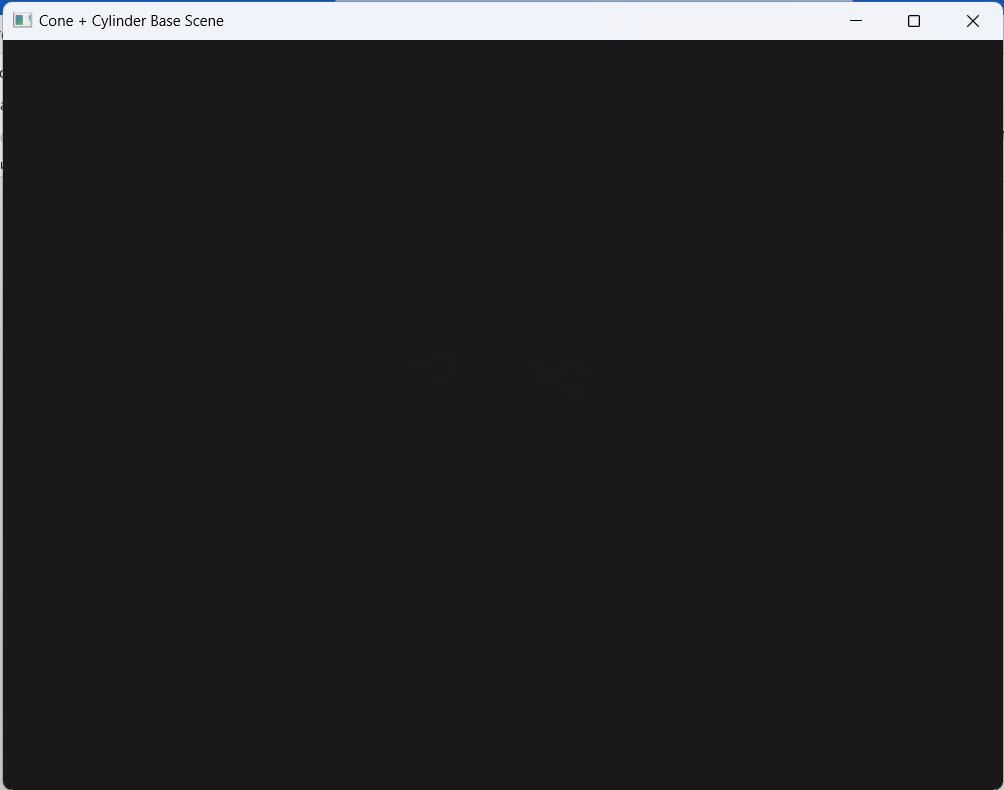
*Рис. 7. Лінійний туман*

**

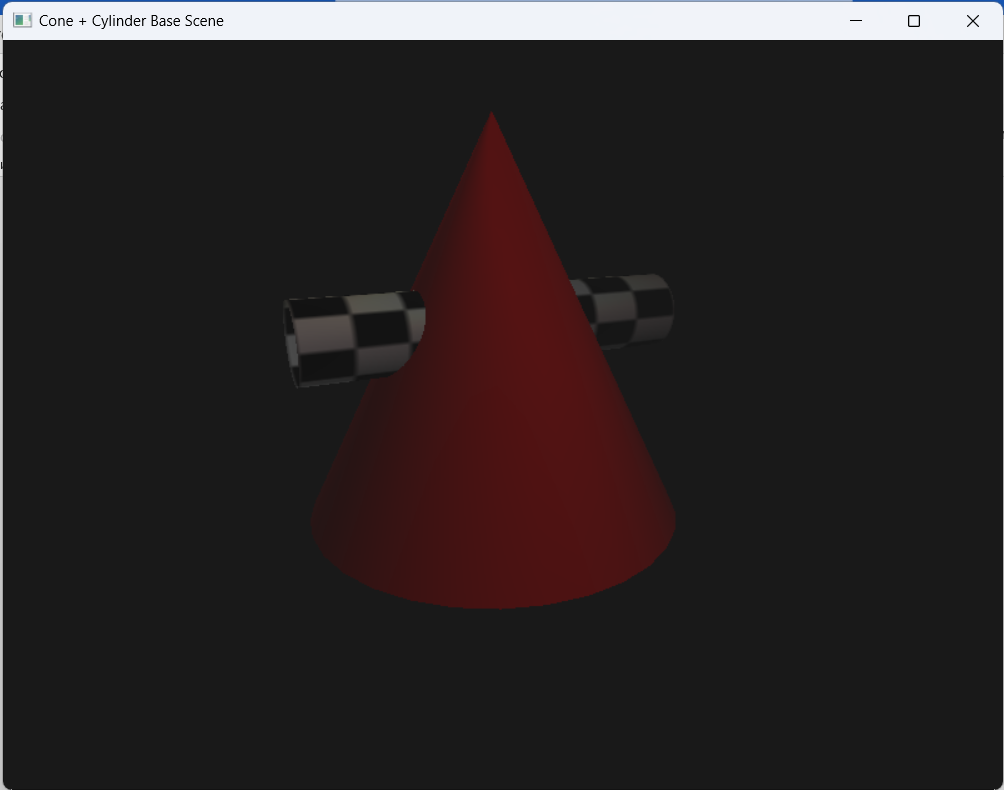
*Рис. 8. Експоненціальний туман на середній дистанції*

**

*Рис. 9. Експоненціальний туман на близькій дистанції*

**

*Рис. 10. Експоненціальний квадратичний туман на середній дистанції*

**

*Рис. 11. Експоненціальний квадратичний туман на близькій дистанції*

**Висновки:** Під час виконання лабораторної роботи я навчився будувати тривимірні об’єкти, покриті текстурою, з подальшим їхнім обертанням та масштабуванням на основі застосування базових геометричних перетворень OpenGL