**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»**

**ІКНІ**

Кафедра **ПЗ**



**ЗВІТ**

до лабораторної роботи №4

**на тему:** *“Проектування статичних тривимірних об’єктів на екран монітора за допомогою графічних бібліотек OpenG”*

**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 статичних тривимірних об’єктів на екран монітора за допомогою графічних бібліотек OpenGL

**Мета:** Навчитися будувати тривимірні об’єкти, проектувати їх на екран монітора, застосовуючи функції 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)**.

**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)**. 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.

**Завдання**

Завдання повинні бути реалізовані у вікні з наявними елементами керування для запуску виконання окремих дій або етапів роботи. За замовчуванням сцена має бути освітлена одним джерелом світла. У всіх важливих місцях програми необхідно давати коментарі з описом призначення поданого далі фрагмента коду.

**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.

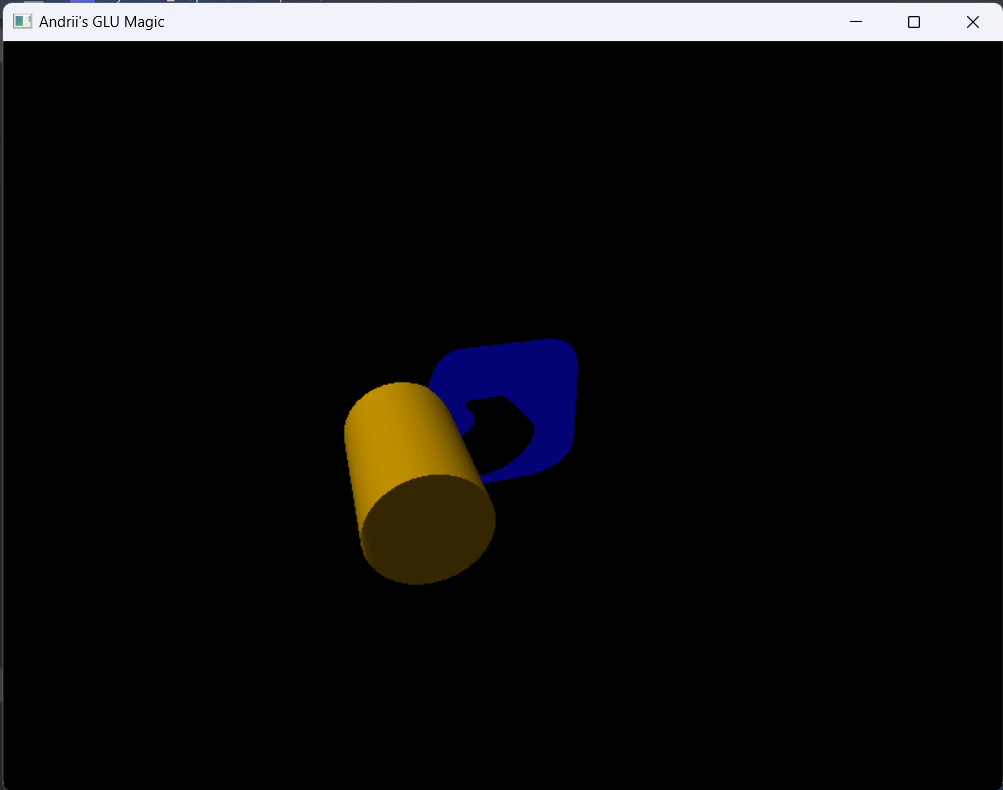
Спроектувати на екран монітора бі-сплайнову поверхню.

12. Згенерувати за допомогою функцій бібліотеки GLU циліндр. У бі-сплайновій поверхні вирізати отвір, сформований NURBS-кривою.

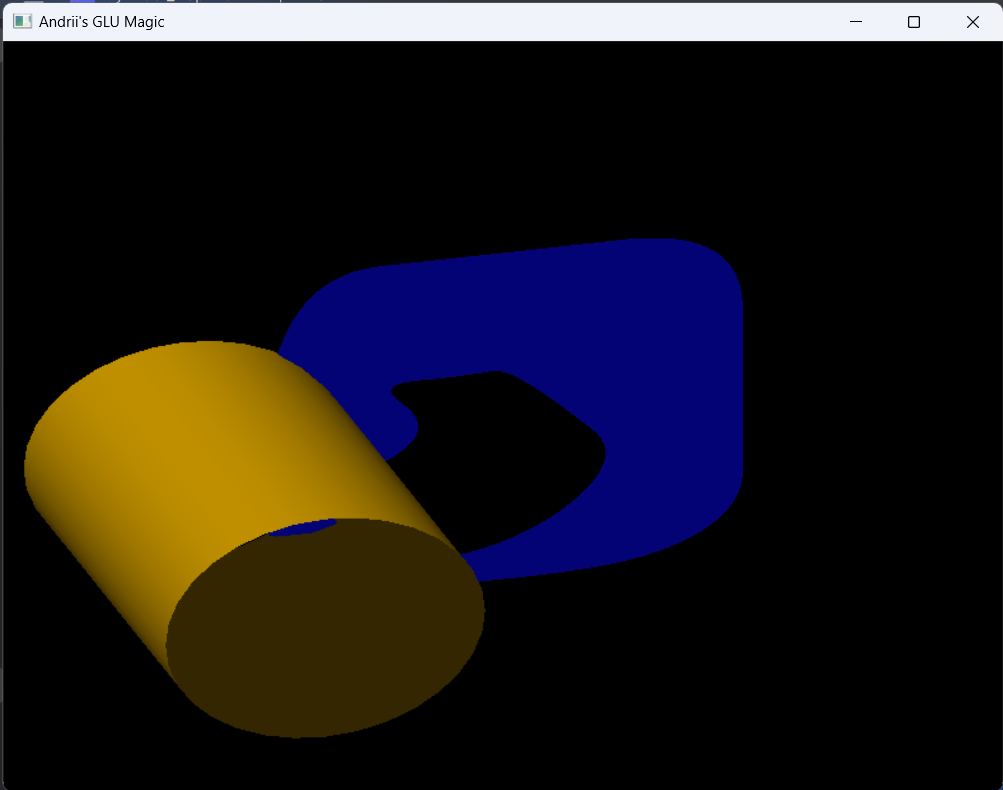
**Код програми**

#include <cstdio>  
#include <iostream>  
#include <GL/glut.h>  
#include <GL/glu.h>  
#include <cmath>  
  
// Початкові параметри  
bool wireframe = false; // режим відображення  
bool perspective = true; // тип проекції  
bool useFlatShading = false; // Плоске або інтерполяційне зафарбування  
  
// Render mode для NURBS (заливка, каркас тощо)  
enum NurbsMode { FILL, OUTLINE\_POLYGON, OUTLINE\_PATCH };  
NurbsMode renderMode = FILL;  
  
// Матеріал/освітлення для NURBS-поверхні: базовий колір (можна міняти клавішею 'c')  
GLfloat surfaceColor[3] = { 0.02f, 0.015f, 0.67f };  
  
// Глобальні об'єкти GLU  
GLUquadric\* quad;  
GLUnurbsObj\* nurbs;  
  
// Контрольні точки для бі-сплайнової поверхні (розмір 4x4)  
GLfloat controlPoints[4][4][3];  
  
// Knot vectors для поверхні (4x4 => 8 елементів)  
GLfloat uKnot[] = { 0, 0, 0, 0, 1, 1, 1, 1 };  
GLfloat vKnot[] = { 0, 0, 0, 0, 1, 1, 1, 1 };  
  
struct transform {  
 GLdouble yaw = 0.0f;  
 GLdouble pitch = 20.0f;  
 GLdouble zoom = 10.0f;  
 int lastX = 0, lastY = 0;  
 bool isDragging = false;  
} SceneTransform;  
  
// Анімування однієї контрольної точки (наприклад, [1][1])  
// Зберігаємо її базове значення:  
GLfloat baseZ = 3.0f;  
  
// GLU\_SAMPLING\_TOLERANCE – можна змінювати клавішею 't'  
GLfloat samplingTolerance = 25.0f;  
  
//  
// Ініціалізація  
//  
void init() {  
 glEnable(GL\_DEPTH\_TEST);  
 glEnable(GL\_LIGHTING);  
 glEnable(GL\_LIGHT0);  
 glEnable(GL\_COLOR\_MATERIAL);  
  
 // Налаштування матеріалу (блиск і specular)  
 GLfloat mat\_specular[] = {1.0, 1.0, 1.0, 1.0};  
 GLfloat mat\_shininess[] = {50.0};  
 glMaterialfv(GL\_FRONT, GL\_SPECULAR, mat\_specular);  
 glMaterialfv(GL\_FRONT, GL\_SHININESS, mat\_shininess);  
  
 // Створення об'єктів GLU  
 quad = gluNewQuadric();  
 nurbs = gluNewNurbsRenderer();  
  
 // Початкові параметри NURBS  
 gluNurbsProperty(nurbs, GLU\_DISPLAY\_MODE, GLU\_FILL);  
 gluNurbsProperty(nurbs, GLU\_SAMPLING\_TOLERANCE, samplingTolerance);  
  
 // Ініціалізація контрольних точок для бі-сплайнової поверхні  
 for (int u = 0; u < 4; ++u) {  
 for (int v = 0; v < 4; ++v) {  
 // Простір від -1.5 до +1.5 по X та Y (масштаб 3.0)  
 controlPoints[u][v][0] = 3.0f \* (static\_cast<float>(u) / 3.0f - 0.5f);  
 controlPoints[u][v][1] = 3.0f \* (static\_cast<float>(v) / 3.0f - 0.5f);  
 // В центрі піднята, решта на 0  
 controlPoints[u][v][2] = (u == 1 && v == 1) ? baseZ : 0.0f;  
 }  
 }  
}  
  
//  
// Малювання NURBS-поверхні з вирізаним отвором  
//  
void drawSurfaceWithHole() {  
 // Встановлення матеріалу (колір)  
 glColor3fv(surfaceColor);  
  
 gluNurbsProperty(nurbs, GLU\_SAMPLING\_TOLERANCE, samplingTolerance);  
 // Режим рендерингу можна перемикати  
 switch (renderMode) {  
 case FILL:  
 gluNurbsProperty(nurbs, GLU\_DISPLAY\_MODE, GLU\_FILL);  
 break;  
 case OUTLINE\_POLYGON:  
 gluNurbsProperty(nurbs, GLU\_DISPLAY\_MODE, GLU\_OUTLINE\_POLYGON);  
 break;  
 case OUTLINE\_PATCH:  
 gluNurbsProperty(nurbs, GLU\_DISPLAY\_MODE, GLU\_OUTLINE\_PATCH);  
 break;  
 }  
 glShadeModel(useFlatShading ? GL\_FLAT : GL\_SMOOTH);  
  
 gluBeginSurface(nurbs);  
  
 gluNurbsSurface(nurbs,  
 8, uKnot, 8, vKnot,  
 4 \* 3, 3,  
 &controlPoints[0][0][0],  
 4, 4,  
 GL\_MAP2\_VERTEX\_3);  
  
 // === Outer boundary in (u,v), from (0,0) to (1,1) ===  
 GLfloat outerPoints[10][2] = {  
 {0.5f, 0.0f},  
 {1.0f, 0.0f},  
 {1.0f, 0.5f},  
 {1.0f, 1.0f},  
 {0.5f, 1.0f},  
 {0.0f, 1.0f},  
 {0.0f, 0.5f},  
 {0.0f, 0.1f},  
 {0.1f, 0.0f},  
 {0.5f, 0.0f}  
 };  
  
 GLfloat outerKnot[] = {0, 0, 0, 1, 2, 3, 4, 5, 5, 6, 7, 7, 7};  
  
 gluBeginTrim(nurbs);  
 gluNurbsCurve(nurbs, 13, outerKnot, 2, &outerPoints[0][0], 3, GLU\_MAP1\_TRIM\_2);  
 gluEndTrim(nurbs);  
  
 GLfloat holePoints[9][2] = {  
 {0.2f, 0.2f},  
 {0.02f, 0.4f},  
 {0.2f, 0.5f},  
 {0.3f, 0.6f},  
 {0.2f, 0.7f},  
 {0.4f, 0.7f},  
 {0.5f, 0.7f},  
 {0.8f, 0.3f},  
 {0.2f, 0.2f},  
 };  
 // === Hole (inner trim loop) ===  
 GLfloat holeKnot[] = {0, 0, 0, 1, 2, 3, 4, 5, 6, 7, 7, 7};  
  
 gluBeginTrim(nurbs);  
 gluNurbsCurve(nurbs, 12, holeKnot, 2, &holePoints[0][0], 3, GLU\_MAP1\_TRIM\_2);  
 gluEndTrim(nurbs);  
  
 gluEndSurface(nurbs);  
}  
  
//  
// Малювання циліндра (зліва від поверхні)  
//  
void drawCylinder() {  
 glColor3f(1.f, 0.75f, 0.f);  
 if (wireframe)  
 gluQuadricDrawStyle(quad, GLU\_LINE);  
 else  
 gluQuadricDrawStyle(quad, GLU\_FILL);  
  
 glPushMatrix();  
 glTranslatef(-2.0f, 0.0f, 0.0f);  
 gluCylinder(quad, 1.0, 1.0, 3.0, 32, 32);  
 glPopMatrix();  
}  
  
//  
// Оновлення камери (орбітна камера)  
//  
void updateCamera() {  
 glLoadIdentity();  
  
 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);  
  
 gluLookAt(camX, camY, camZ,  
 0.0, 0.0, 0.0,  
 0.0, 1.0, 0.0);  
}  
  
//  
// Анімаційна функція – оновлюється кожен кадр  
//  
void idleFunc() {  
 // Використовуємо час для анімації (зміна контролюючої точки [1][1])  
 float time = glutGet(GLUT\_ELAPSED\_TIME) / 1000.0f; // секунди  
 controlPoints[1][1][2] = baseZ + sinf(time) \* 1.0f; // коливання ±1  
 glutPostRedisplay();  
}  
  
//  
// Малювання сцени  
//  
void display() {  
 glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);  
 updateCamera();  
  
 // Малюємо поверхню з отвором  
 glPushMatrix();  
 drawSurfaceWithHole();  
 glPopMatrix();  
  
 // Малюємо циліндр  
 drawCylinder();  
  
 glutSwapBuffers();  
}  
  
//  
// Обробка зміни розміру вікна  
//  
void reshape(int w, int h) {  
 glViewport(0, 0, w, h);  
 glMatrixMode(GL\_PROJECTION);  
 glLoadIdentity();  
 if (perspective)  
 gluPerspective(60.0, static\_cast<float>(w) / h, 1.0, 100.0);  
 else  
 glOrtho(-3, 3, -3, 3, 1, 100);  
 glMatrixMode(GL\_MODELVIEW);  
}  
  
//  
// Обробка клавіатури  
//  
void keyboard(unsigned char key, int x, int y) {  
 switch (key) {  
 case 'l':  
 case 'L':  
 wireframe = !wireframe;  
 break;  
 case 'p':  
 case 'P':  
 perspective = !perspective;  
 reshape(glutGet(GLUT\_WINDOW\_WIDTH), glutGet(GLUT\_WINDOW\_HEIGHT));  
 break;  
 // Зміна кольору поверхні: натисни 'c'  
 case 'c':  
 case 'C': {  
 // Проста ротація між кількома кольорами  
 static int colIndex = 0;  
 colIndex = (colIndex + 1) % 4;  
 switch (colIndex) {  
 case 0:  
 surfaceColor[0] = 0.02f; surfaceColor[1] = 0.015f; surfaceColor[2] = 0.67f; break;  
 case 1:  
 surfaceColor[0] = 0.67f; surfaceColor[1] = 0.15f; surfaceColor[2] = 0.02f; break;  
 case 2:  
 surfaceColor[0] = 0.02f; surfaceColor[1] = 0.67f; surfaceColor[2] = 0.15f; break;  
 case 3:  
 surfaceColor[0] = 0.5f; surfaceColor[1] = 0.5f; surfaceColor[2] = 0.5f; break;  
 }  
 break;  
 }  
 // Зміна GLU\_SAMPLING\_TOLERANCE (натисни 't' для зменшення, 'T' для збільшення)  
 case 't':  
 samplingTolerance = (samplingTolerance > 1.0f) ? samplingTolerance - 1.0f : samplingTolerance;  
 break;  
 case 'T':  
 samplingTolerance += 1.0f;  
 break;  
 // Перемикання режимів рендерингу для NURBS-поверхні (натисни 'm')  
 case 'm':  
 case 'M':  
 renderMode = static\_cast<NurbsMode>((renderMode + 1) % 3);  
 break;  
 case 'f':  
 case 'F':  
 useFlatShading = !useFlatShading;  
 break;  
 case 27:  
 exit(0);  
 }  
 glutPostRedisplay();  
}  
  
//  
// 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;  
 glutPostRedisplay();  
 } else if (button == 4 && state == GLUT\_DOWN) { // scroll down  
 SceneTransform.zoom += 0.5f;  
 if (SceneTransform.zoom > 100.f) SceneTransform.zoom = 100.f;  
 glutPostRedisplay();  
 }  
 if (button == GLUT\_LEFT\_BUTTON) {  
 if (state == GLUT\_DOWN) {  
 SceneTransform.isDragging = true;  
 SceneTransform.lastX = x;  
 SceneTransform.lastY = y;  
 } else {  
 SceneTransform.isDragging = false;  
 }  
 }  
}  
  
//  
// Обробка руху миші (drag-to-rotate)  
//  
void motion(int x, int y) {  
 if (SceneTransform.isDragging) {  
 float dx = x - SceneTransform.lastX;  
 float dy = y - SceneTransform.lastY;  
  
 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.lastX = x;  
 SceneTransform.lastY = y;  
 glutPostRedisplay();  
 }  
}  
  
//  
// Main function  
//  
int main(int argc, char\*\* argv) {  
 glutInit(&argc, argv);  
 glutInitDisplayMode(GLUT\_RGB | GLUT\_DOUBLE | GLUT\_DEPTH);  
 glutInitWindowSize(800, 600);  
 glutCreateWindow("Andrii's GLU Magic");  
  
 init();  
 glutDisplayFunc(display);  
 glutReshapeFunc(reshape);  
 glutKeyboardFunc(keyboard);  
 glutMouseFunc(mouse);  
 glutMotionFunc(motion);  
 glutIdleFunc(idleFunc); // Анімація поверхні  
  
 glutMainLoop();  
 return 0;  
}

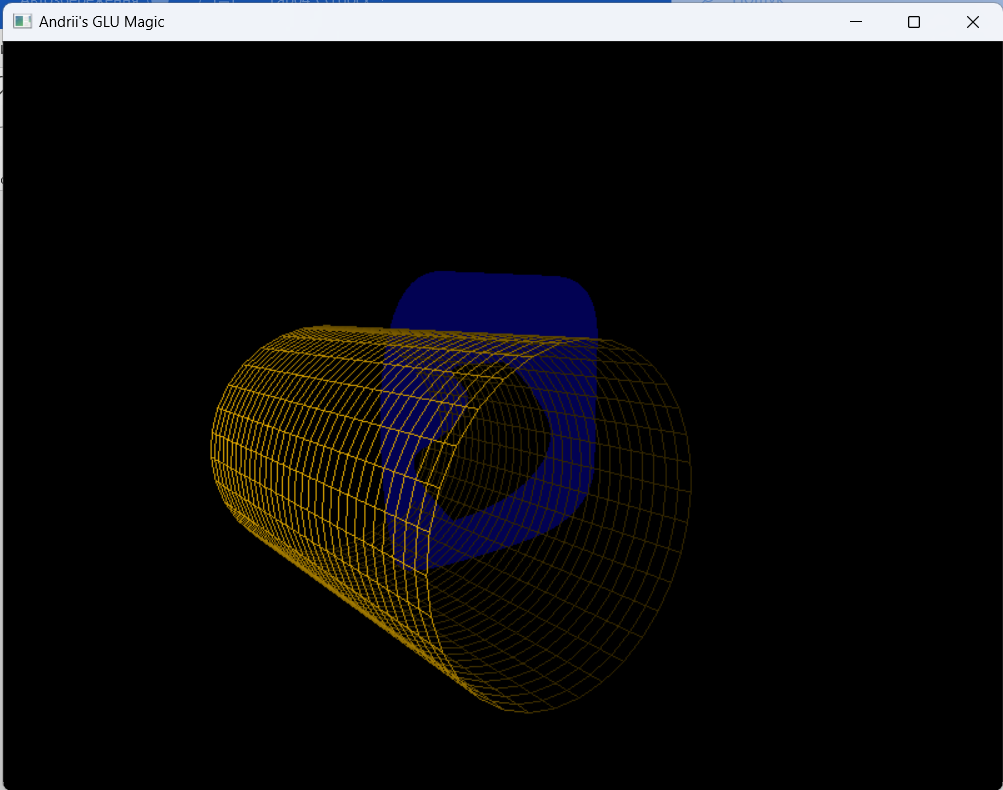
# Результат виконання програми

**

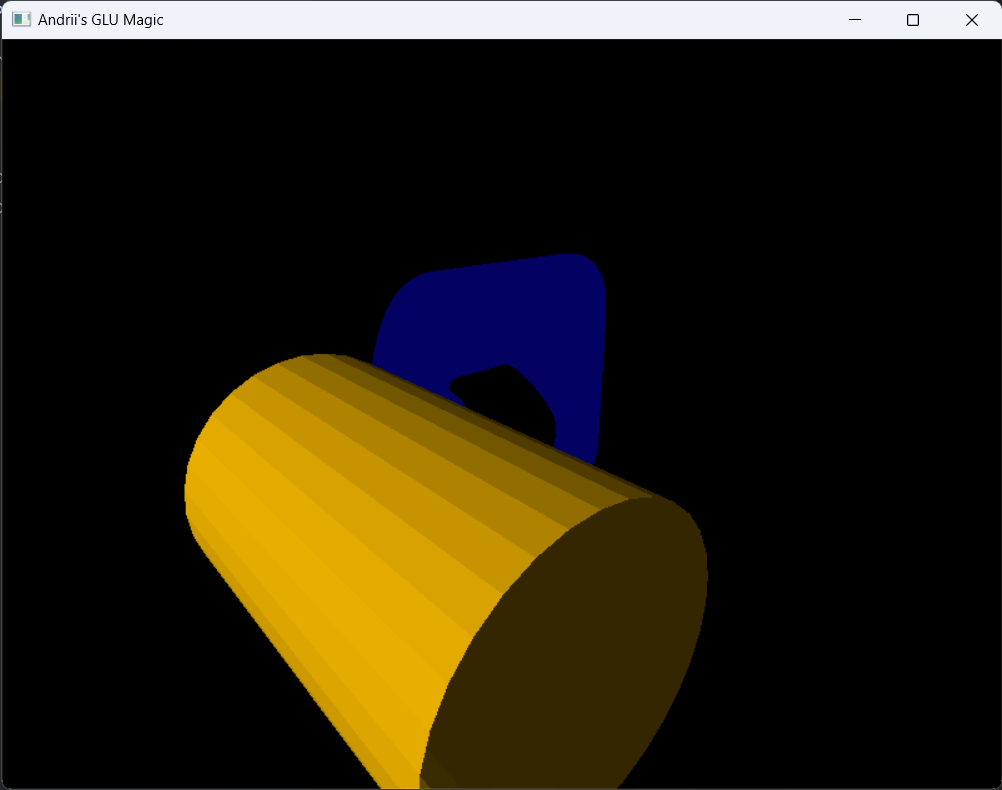
*Рис. 1. Циліндр та поверхня з отвором всередині. Перспектива*

**

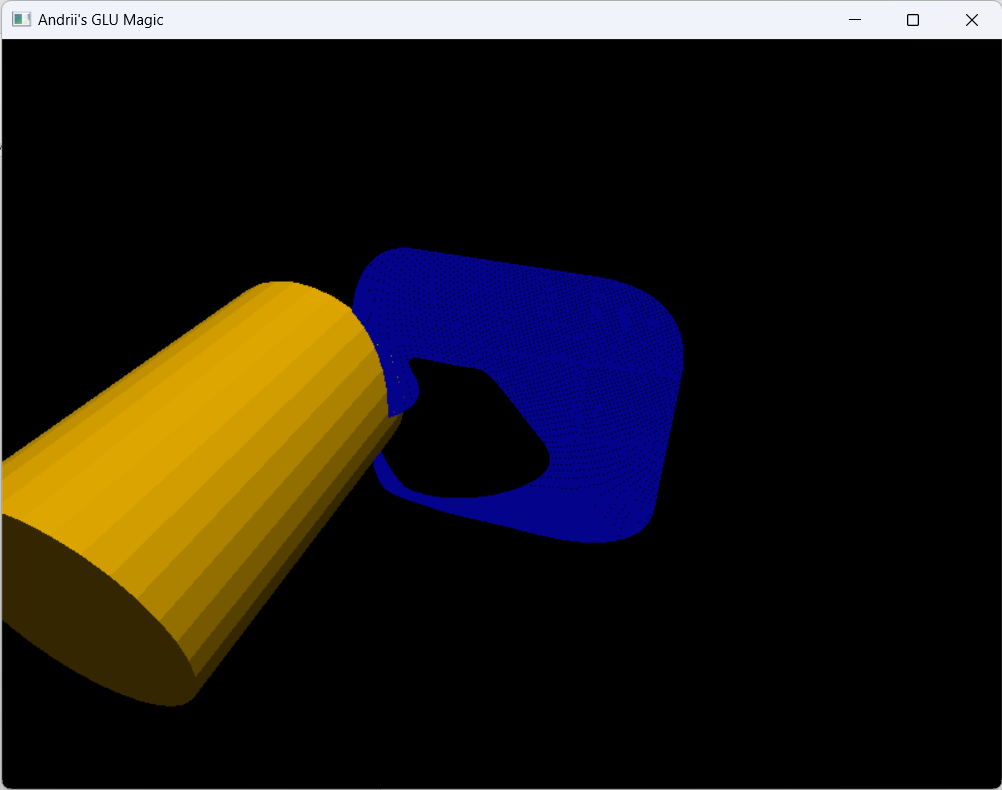
*Рис. 2. Циліндр та поверхня з отвором всередині. Ортогональне представлення*

**

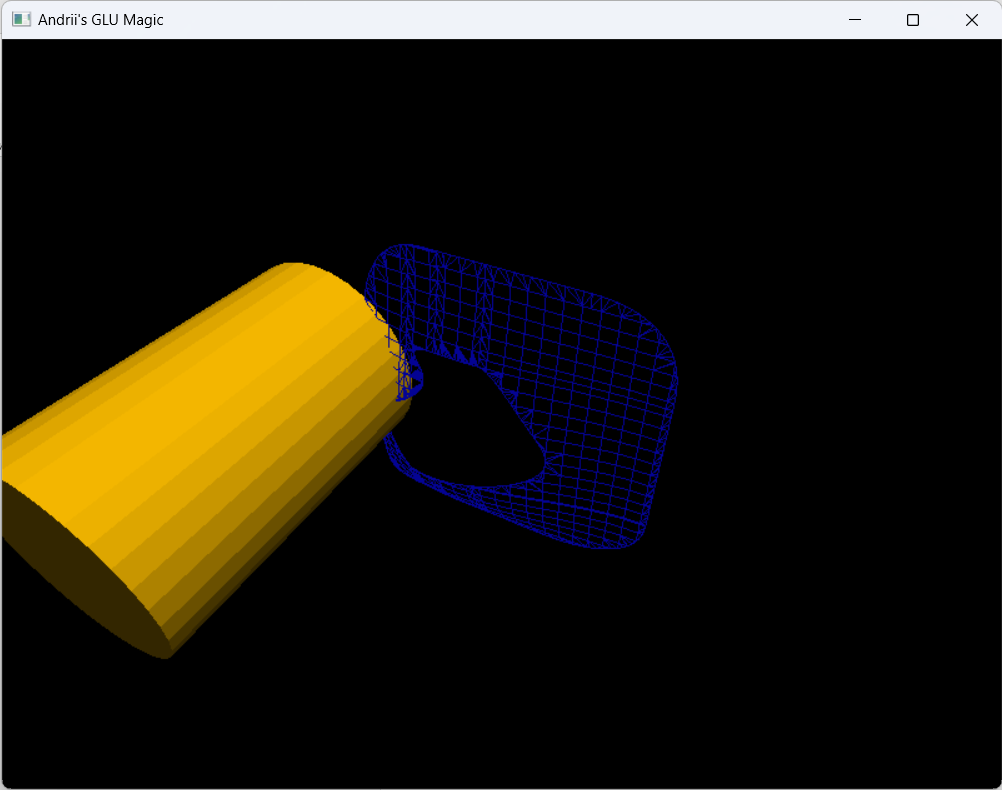
*Рис. 3. Циліндр каркасний вигляд*

**

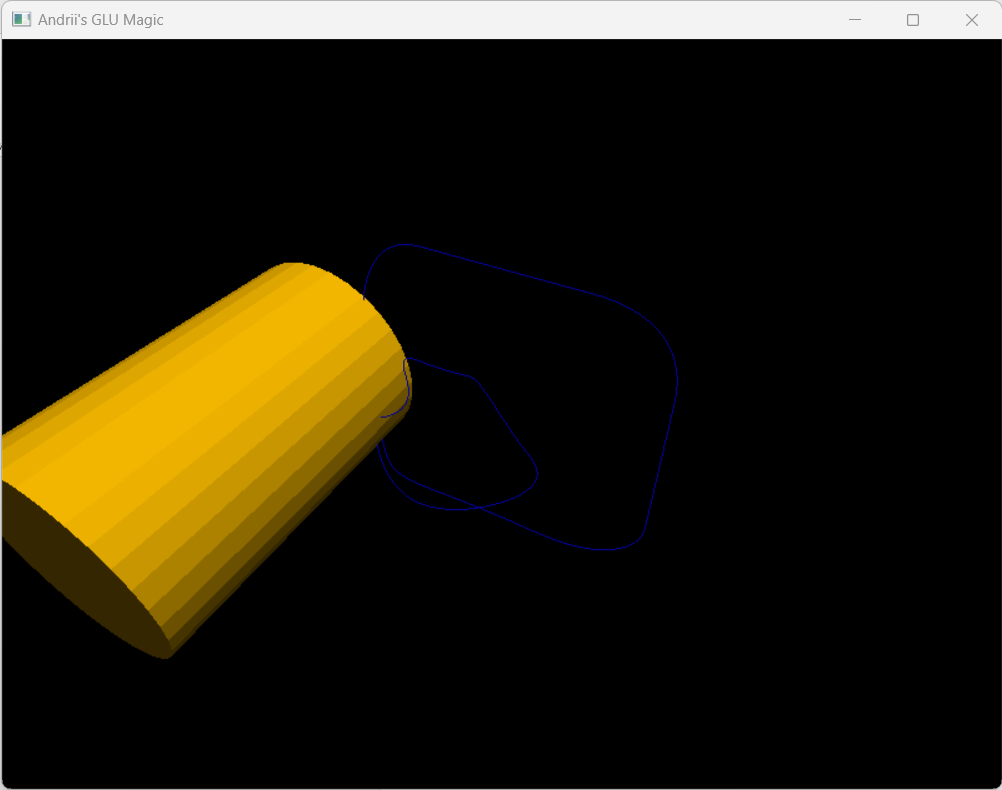
*Рис. 4. Плоске зафарбування*



*Рис. 5. Дрібна теселяція NURBS поверхні*

**

*Рис. 6. Крупна теселяція NURBS поверхні*

**

*Рис. 7. Обведені границі NURBS поверхні*

**Висновки:** Під час виконання лабораторної роботи я навчився будувати тривимірні об’єкти, проектувати їх на екран монітора, застосовуючи функції OpenGL