МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського

«Харківський авіаційний інститут»

Факультет програмної інженерії та бізнесу

Кафедра інженерії програмного забезпечення

Практичні роботи

Minor *«Розробник ігрових додатків»*

дисципліна *«Комп’ютерна графіка з OpenGL»*

(назва дисципліни)

Виконав: студент 3 курсу групи  *535а*

напряму підготовки (спеціальності):

*124 системне програмування*

(шифр і назва напряму підготовки / спеціальності)

*Ананєнко Д.В.*

(прізвище й ініціали студента)

Прийняв: *доц. каф 603, к.т.н, Лучшев П.О.*

(посада, науковий ступінь, прізвище й ініціали)

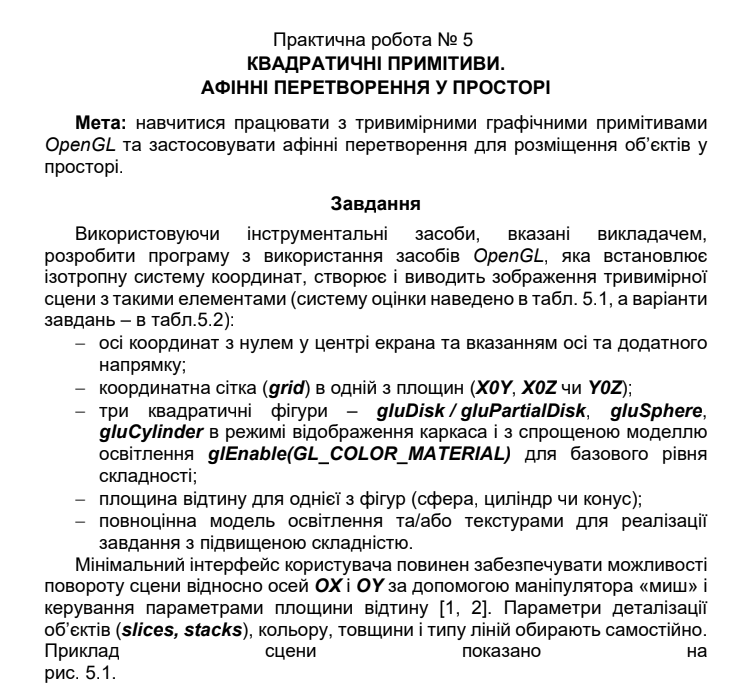
Національна шкала:

Кількість балів:

Оцінка ECTS:

# Практична робота 5. ГРАФІЧНІ ПРИМІТИВИ OPENGL

## Завдання, варіант № 1





## Системна інформація

Для розробки та виконання практичних робіт використовувалися наступні апаратні та програмні засоби:

Processor AMD Ryzen 7 5800X 8-Core Processor 3.80 GHzRAM 16.0 GB (15.9 GB usable)

System type 64-bit operating system, x64-based processor

Edition Windows 10 Pro Version 22H2

IDE Microsoft Visual Studio Enterprise 2024 (64-bit)

**Код**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Diagnostics;

using System.Drawing;

using System.Drawing.Design;

using System.Linq;

using System.Windows.Forms;

namespace Transform3D

{

public partial class RenderControl : OpenGL

{

// Визначення власного кольору LightPurple

private readonly Color LightPurple = Color.FromArgb(200, 150, 255); // Ви можете налаштувати RGB значення за бажанням

// Список фігур для малювання

private List<Figure> figures = new List<Figure>();

public RenderControl()

{

InitializeComponent();

MouseWheel += OnWheel;

// Підписка на події миші

this.MouseDown += OnDown;

this.MouseUp += OnUp;

this.MouseMove += OnMove;

this.Render += OnRender;

// Ініціалізація фігур за варіантом 1

InitializeFiguresVariant1();

}

double size = 1.1;

double AspectRatio { get => (double)Width / Height; }

double xMin { get => (AspectRatio > 1) ? -size \* AspectRatio : -size; }

double xMax { get => (AspectRatio > 1) ? +size \* AspectRatio : +size; }

double yMin { get => (AspectRatio < 1) ? -size / AspectRatio : -size; }

double yMax { get => (AspectRatio < 1) ? +size / AspectRatio : +size; }

double zMin { get => -size \* 1.2; }

double zMax { get => +size \* 1.2; }

double rx = +20;

double ry = -30;

double m = 1.0;

public bool DepthTest { set; get; } = true;

/// <summary>

/// Ініціалізація фігур для варіанту 1

/// </summary>

private void InitializeFiguresVariant1()

{

// Варіант 1: Y0Z

// Фігури:

// 1. Сфера ∥ 0X -1.5 -0.5 +2.5 1.0 - - - -

figures.Add(new Figure

{

Type = FigureType.Sphere,

Axis = "∥",

AlignmentAxis = "0X",

x0 = -1.5,

y0 = -0.5,

z0 = +2.5,

R = 1.0,

r = 0.0,

h = 0.0,

AngleStart = 0.0,

AngleSweep = 360.0,

Slices = 36,

Stacks = 18

});

// 2. Циліндр ⇅ 0Y +3.5 +1.5 +4.0 1.5 1.5 1.5 - -

figures.Add(new Figure

{

Type = FigureType.Cylinder,

Axis = "⇅",

AlignmentAxis = "0Y",

x0 = +3.5,

y0 = +1.5,

z0 = +4.0,

R = 1.5,

r = 1.5, // Для циліндра r = R

h = 1.5,

AngleStart = 0.0,

AngleSweep = 360.0,

Slices = 36,

Stacks = 0 // Не використовується для циліндра

});

// 3. Диск ∥ 0Z +2.0 -2.0 -3.5 1.5 0.0 - -

figures.Add(new Figure

{

Type = FigureType.Disk,

Axis = "∥",

AlignmentAxis = "0Z",

x0 = +2.0,

y0 = -2.0,

z0 = -3.5,

R = 1.5,

r = 0.0, // Внутрішній радіус диска

h = 0.0,

AngleStart = 0.0,

AngleSweep = 360.0,

Slices = 36

});

}

private void OnRender(object sender, EventArgs e)

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glLoadIdentity();

glViewport(0, 0, Width, Height);

glOrtho(xMin, xMax, yMin, yMax, zMin, zMax);

glRotated(rx, 1, 0, 0);

glRotated(ry, 0, 1, 0);

glColor(Color.Black);

if (DepthTest)

glEnable(GL\_DEPTH\_TEST);

Axis(0.9);

glScaled(m, m, m);

// Ітерація по всім фігурам та їх малювання

foreach (var figure in figures)

{

switch (figure.Type)

{

case FigureType.Sphere:

DrawSphere(figure);

break;

case FigureType.Cylinder:

DrawCylinder(figure);

break;

case FigureType.Disk:

DrawDisk(figure);

break;

default:

break;

}

}

glDisable(GL\_DEPTH\_TEST);

}

/// <summary>

/// Метод для малювання сфери на основі параметрів фігури

/// </summary>

/// <param name="figure">Фігура типу сфери</param>

private void DrawSphere(Figure figure)

{

glPushMatrix();

glTranslated(figure.x0, figure.y0, figure.z0);

glColor(Color.Green);

for (int i = 0; i < figure.Stacks; i++)

{

double lat0 = Math.PI \* (-0.5 + (double)(i) / figure.Stacks);

double z1 = figure.R \* Math.Sin(lat0);

double zr1 = figure.R \* Math.Cos(lat0);

double lat1 = Math.PI \* (-0.5 + (double)(i + 1) / figure.Stacks);

double z2 = figure.R \* Math.Sin(lat1);

double zr2 = figure.R \* Math.Cos(lat1);

glBegin(GL\_QUAD\_STRIP);

for (int j = 0; j <= figure.Slices; j++)

{

double lng = 2.0 \* Math.PI \* (double)(j) / figure.Slices;

double x = Math.Cos(lng);

double y = Math.Sin(lng);

glVertex3d(x \* zr1, y \* zr1, z1);

glVertex3d(x \* zr2, y \* zr2, z2);

}

glEnd();

}

glPopMatrix();

}

/// <summary>

/// Метод для малювання циліндра на основі параметрів фігури

/// </summary>

/// <param name="figure">Фігура типу циліндра</param>

private void DrawCylinder(Figure figure)

{

glPushMatrix();

glTranslated(figure.x0, figure.y0, figure.z0);

// Вирівнювання циліндра за заданою віссю

switch (figure.AlignmentAxis)

{

case "0X":

// Орієнтація вздовж X-осі (за замовчуванням)

break;

case "0Y":

glRotated(-90, 1, 0, 0); // Обертання для орієнтації вздовж Y

break;

case "0Z":

glRotated(90, 1, 0, 0); // Обертання для орієнтації вздовж Z

break;

default:

break;

}

glColor(Color.Orange);

glBegin(GL\_QUAD\_STRIP);

for (int i = 0; i <= figure.Slices; i++)

{

double theta = 2.0 \* Math.PI \* i / figure.Slices;

double cosTheta = Math.Cos(theta);

double sinTheta = Math.Sin(theta);

double xTop = figure.R \* cosTheta;

double yTop = figure.R \* sinTheta;

double zTop = 0;

double xBase = figure.r \* cosTheta;

double yBase = figure.r \* sinTheta;

double zBase = figure.h;

glVertex3d(xTop, yTop, zTop);

glVertex3d(xBase, yBase, zBase);

}

glEnd();

// Верхня основа циліндра

glColor(Color.DarkOrange);

glBegin(GL\_TRIANGLE\_FAN);

glVertex3d(0, 0, 0); // Центр верхньої основи

for (int i = 0; i <= figure.Slices; i++)

{

double theta = 2.0 \* Math.PI \* i / figure.Slices;

double x = figure.R \* Math.Cos(theta);

double y = figure.R \* Math.Sin(theta);

glVertex3d(x, y, 0);

}

glEnd();

// Нижня основа циліндра

glColor(Color.DarkOrange);

glBegin(GL\_TRIANGLE\_FAN);

glVertex3d(0, 0, figure.h); // Центр нижньої основи

for (int i = figure.Slices; i >= 0; i--)

{

double theta = 2.0 \* Math.PI \* i / figure.Slices;

double x = figure.r \* Math.Cos(theta);

double y = figure.r \* Math.Sin(theta);

glVertex3d(x, y, figure.h);

}

glEnd();

glPopMatrix();

}

/// <summary>

/// Метод для малювання диска на основі параметрів фігури

/// </summary>

/// <param name="figure">Фігура типу диска</param>

private void DrawDisk(Figure figure)

{

glPushMatrix();

glTranslated(figure.x0, figure.y0, figure.z0);

// Вирівнювання диска за заданою віссю

switch (figure.AlignmentAxis)

{

case "0X":

// Орієнтація вздовж X-осі

glRotated(-90, 0, 1, 0);

break;

case "0Y":

// Орієнтація вздовж Y-осі

glRotated(90, 1, 0, 0);

break;

case "0Z":

// Орієнтація вздовж Z-осі (за замовчуванням)

break;

default:

break;

}

glColor(Color.Purple);

glBegin(GL\_TRIANGLE\_FAN);

glVertex3d(0, 0, 0); // Центр диска

int slices = figure.Slices;

double angleStartRad = figure.AngleStart \* Math.PI / 180.0;

double angleSweepRad = figure.AngleSweep \* Math.PI / 180.0;

for (int i = 0; i <= slices; i++)

{

double theta = angleStartRad + (angleSweepRad \* i / slices);

double x = figure.R \* Math.Cos(theta);

double y = figure.R \* Math.Sin(theta);

glVertex3d(x, y, 0);

}

glEnd();

// Якщо диск має внутрішній радіус, малюємо внутрішню частину

if (figure.r > 0)

{

glColor(LightPurple); // Використання власного LightPurple

glBegin(GL\_TRIANGLE\_FAN);

glVertex3d(0, 0, 0); // Центр внутрішнього кола

for (int i = 0; i <= slices; i++)

{

double theta = angleStartRad + (angleSweepRad \* i / slices);

double x = figure.r \* Math.Cos(theta);

double y = figure.r \* Math.Sin(theta);

glVertex3d(x, y, 0);

}

glEnd();

}

glPopMatrix();

}

/// <summary>

/// Метод для малювання координатних осей.

/// </summary>

/// <param name="sz">Довжина осей.</param>

private void Axis(double sz)

{

double a = sz / 10.0;

glBegin(GL\_LINES);

// X-ось

glColor(Color.Red);

glVertex3d(-a, 0, 0); glVertex3d(+sz, 0, 0);

// Y-ось

glColor(Color.Green);

glVertex3d(0, -a, 0); glVertex3d(0, +sz, 0);

// Z-ось

glColor(Color.Blue);

glVertex3d(0, 0, -a); glVertex3d(0, 0, +sz);

glEnd();

DrawText("+X", sz + a, 0, 0);

DrawText("+Y", 0, sz + a, 0);

DrawText("+Z", 0, 0, sz + a);

}

bool fDown = false;

double lastX, lastY;

private void OnDown(object sender, MouseEventArgs e)

{

if (e.Button == MouseButtons.Left)

{

fDown = true;

lastX = e.X;

lastY = e.Y;

}

}

private void OnUp(object sender, MouseEventArgs e)

{

if ((e.Button == MouseButtons.Left) && fDown)

fDown = false;

}

private void OnMove(object sender, MouseEventArgs e)

{

if (fDown)

{

ry -= (lastX - e.X) / 2.0;

rx -= (lastY - e.Y) / 2.0;

lastX = e.X;

lastY = e.Y;

Invalidate();

}

}

private void OnWheel(object sender, MouseEventArgs e)

{

m += e.Delta / 2000.0;

Invalidate();

}

/// <summary>

/// Клас для представлення фігури

/// </summary>

private class Figure

{

public FigureType Type { get; set; }

public string Axis { get; set; } // ∥, ⇈, ⇅

public string AlignmentAxis { get; set; } // 0X, 0Y, 0Z

public double x0 { get; set; }

public double y0 { get; set; }

public double z0 { get; set; }

public double R { get; set; } // Радіус

public double r { get; set; } // Внутрішній радіус або другий радіус для циліндра

public double h { get; set; } // Висота

public double AngleStart { get; set; } // Початковий кут для диска

public double AngleSweep { get; set; } // Кут розгортки для диска

// Для сфер і циліндрів

public int Slices { get; set; } = 36;

public int Stacks { get; set; } = 18;

}

/// <summary>

/// Перелічення типів фігур

/// </summary>

private enum FigureType

{

Sphere,

Cylinder,

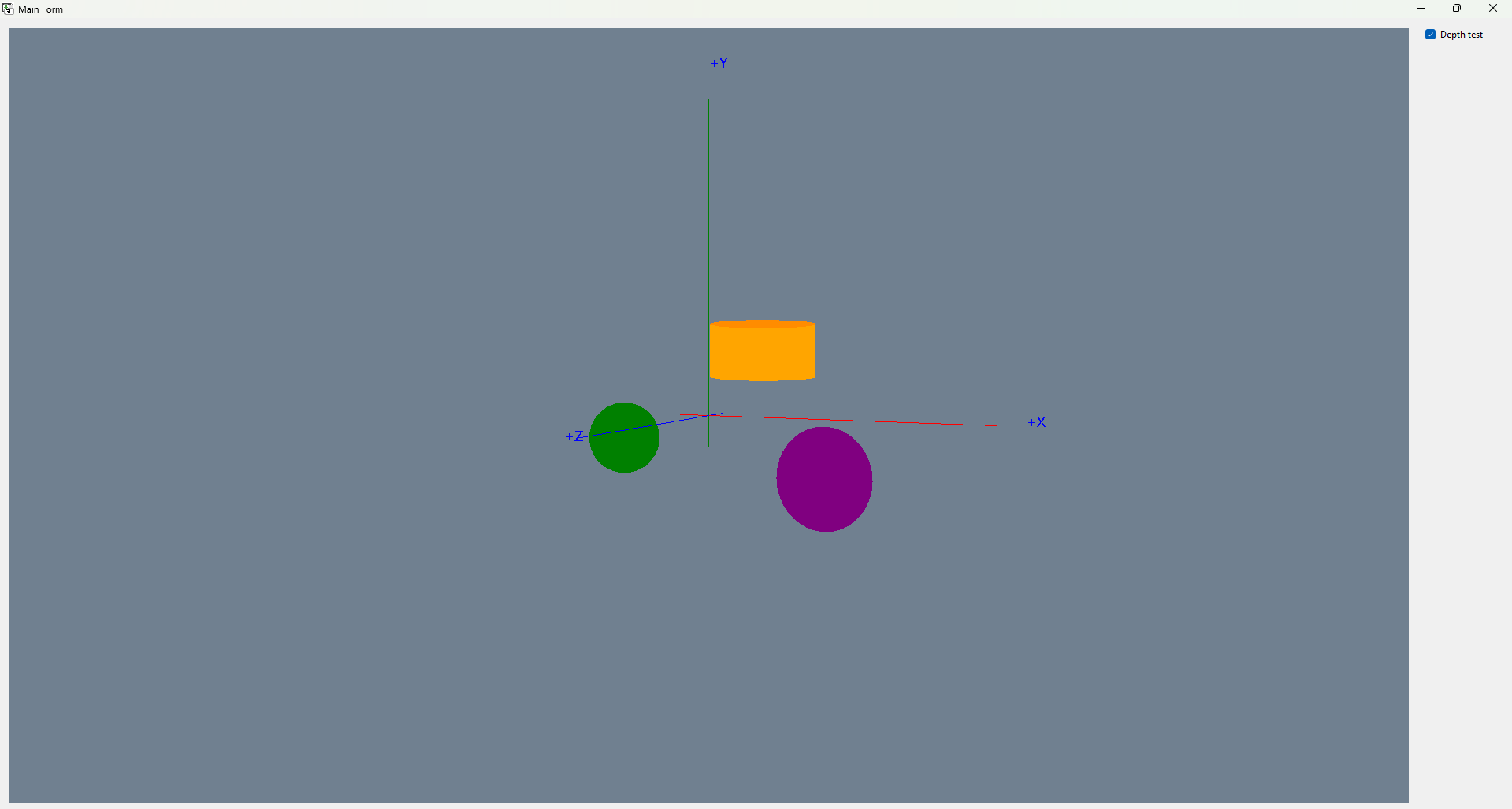
Disk

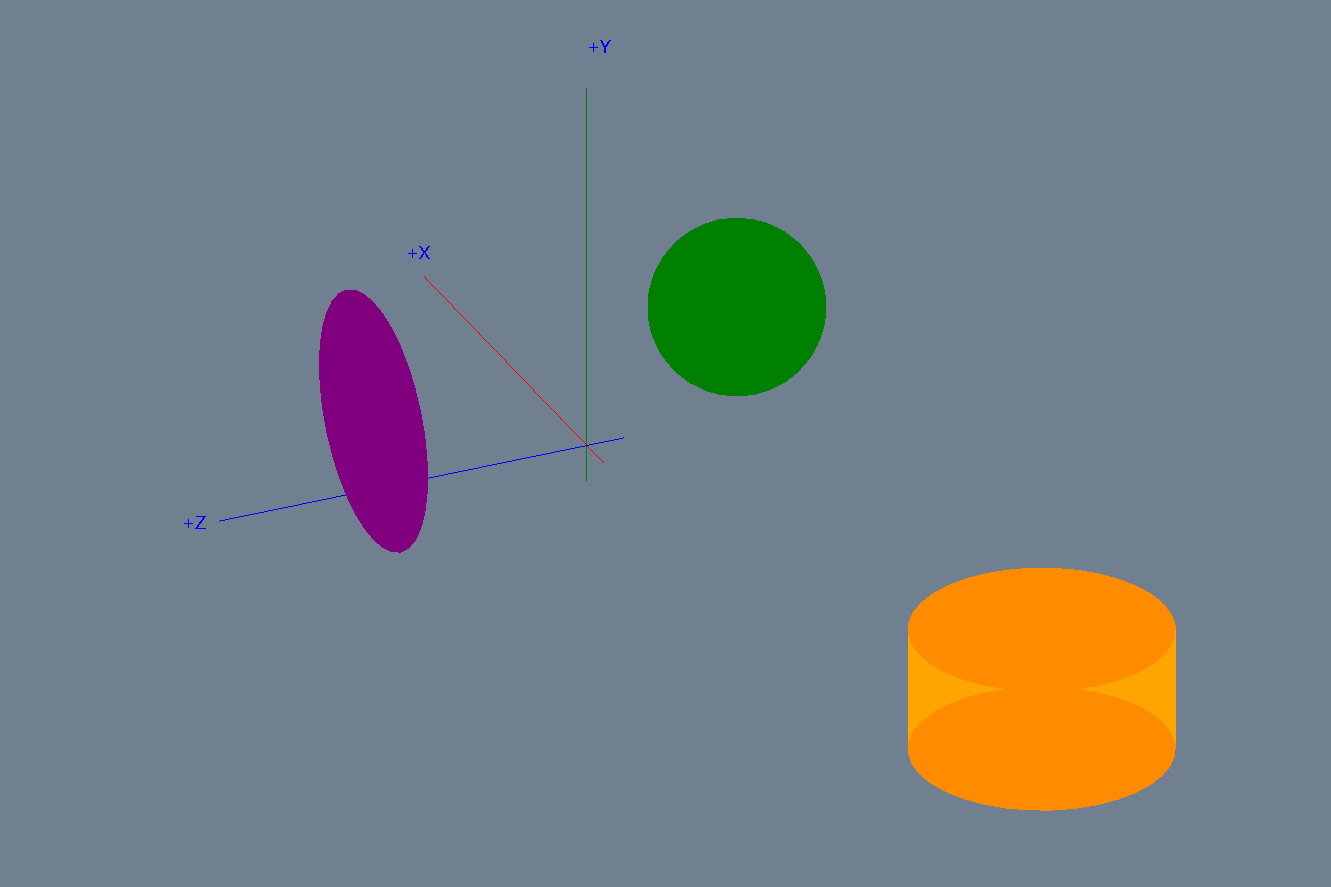
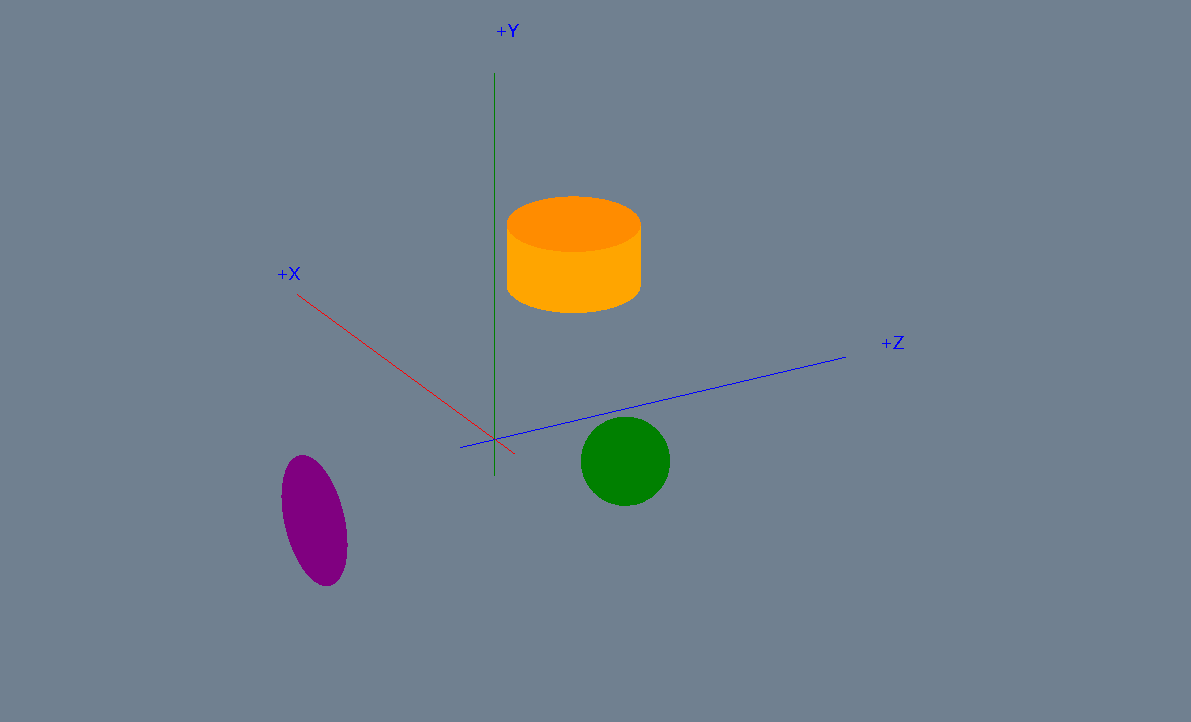
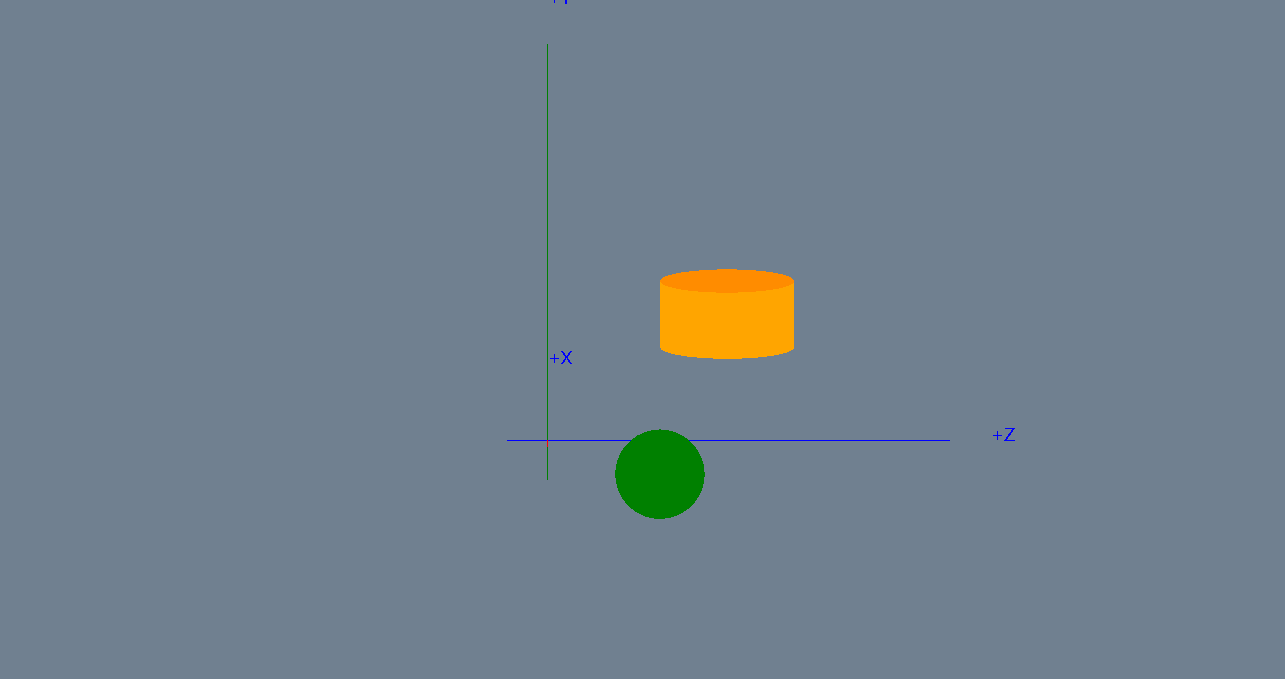
}

}

}

**Скріншоти**





### **Отчет о реализации 3D-графического рендеринга с использованием OpenGL**

### **Основные компоненты**

Проект разделен на следующие ключевые части:

* **Фигуры**:
* Моделирование сфер, цилиндров и дисков с использованием параметров для их точного позиционирования и настройки.
* **Перспективная проекция**:
* Использование функции gluPerspective для создания реалистичного 3D-вида.
* **Рендеринг сцены**:
* Применение OpenGL для отрисовки объектов и управления глубиной.
* **Интерактивность**:
* Реализация управления сценой с помощью мыши (обзор, масштабирование).
* **Координатные оси**:
* Отображение координатных осей для упрощения ориентации в 3D-пространстве.

### **Подробное описание функций**

#### **Конструктор RenderControl**

Инициализирует объект класса и настраивает:

* Подписку на события мыши для управления сценой:
* MouseWheel — масштабирование.
* MouseDown, MouseMove, MouseUp — обертание сцены.
* Вызов функции InitializeFiguresVariant1 для загрузки параметров фигур.

#### **InitializeFiguresVariant1**

Создает список фигур на основе их параметров:

* **Сфера**: Позиция (−1.5, −0.5, +2.5), радиус 1.0.
* **Цилиндр**: Позиция (+3.5, +1.5, +4.0), радиус 1.5, высота 1.5.
* **Диск**: Позиция (+2.0, −2.0, −3.5), радиус 1.5, внутренний радиус 0.0.

#### **OnRender**

Основной метод для отрисовки сцены:

* Очищает буферы цвета и глубины.
* Настраивает перспективную проекцию через gluPerspective.
* Смещает камеру назад для обзора сцены.
* Применяет повороты сцены (углы rx, ry).
* Масштабирует сцену.
* Отрисовывает координатные оси (Axis).
* Последовательно отрисовывает все фигуры из списка figures с помощью функций DrawSphere, DrawCylinder, DrawDisk.

#### **gluPerspective**

Настраивает параметры перспективной проекции:

* Поле зрения (угол обзора) — 45 градусов.
* Соотношение сторон экрана.
* Плоскости отсечения (ближняя и дальняя).

#### **DrawSphere**

Моделирует сферу:

* Использует широты и долготы для генерации вершин.
* Строит полосу сферической поверхности через GL\_QUAD\_STRIP.

#### **DrawCylinder**

Моделирует цилиндр:

* Строит боковую поверхность с помощью GL\_QUAD\_STRIP.
* Верхняя и нижняя основы отрисовываются через GL\_TRIANGLE\_FAN.
* Оринетируется вдоль одной из осей (X, Y, Z).

#### **DrawDisk**

Создает диск:

* Отрисовывает заполненный круг через GL\_TRIANGLE\_FAN.
* Если внутренний радиус больше 0, добавляет внутреннюю часть другого цвета.

#### **Управление камерой**

* **Обертание**: Реализовано через события OnDown, OnMove, OnUp.
* ЛКМ фиксирует начальную позицию.
* Перемещение мыши изменяет углы обертания rx, ry.
* **Масштабирование**: Реализовано через колесо мыши (OnWheel), которое изменяет коэффициент масштаба m.

#### **Axis**

Отрисовывает координатные оси:

* Оси X, Y, Z обозначены разными цветами (красный, зеленый, синий).
* Добавляет текстовые подписи (+X, +Y, +Z).

### **Висновок**

Код створює інтерактивний 3D контрол для відображення різних геометричних фігур (сфера, циліндр, диск) з можливістю їх обертання та масштабування за допомогою миші. Використання OpenGL забезпечує ефективний рендеринг та гнучкість у налаштуванні відображуваних об'єктів. Структура коду дозволяє легко додавати нові типи фігур та розширювати функціональність контролу.