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  *335а*

напряму підготовки (спеціальності):

*122 “Комп’ютерні науки”*

(шифр і назва напряму підготовки / спеціальності)

*Мінаков А.А.*

(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:

Зміст

[Практична робота 1. Основні принципи роботи з OpenGL 3](#_Toc177946558)

[Завдання, варіант № 14 3](#_Toc177946559)

[Системна інформація 3](#_Toc177946560)

[Теоретичні відомості 3](#_Toc177946561)

[Результати виконання практичної роботи](#_Toc177946562) 4

[Практична робота 2. Графічні примітиви OpenGL 8](#_Практична_робота_2.)

[Завдання, варіант № 14 8](#_Завдання,_варіант_№)

[Теоретичні відомості 10](#_Теоретичні_відомості)

[Результати виконання практичної роботи 13](#_Результати_виконання_практичної)

[Практична робота 3. Графік функції однієї змінної 19](#_Практична_робота_3.)

[Завдання, варіант № 14 19](#_Завдання,_варіант_№_1)

[Теоретичні відомості 19](#_Теоретичні_відомості_1)

[Результати виконання практичної роботи 21](#_Результати_виконання_практичної_1)

[Практична робота 4. Криві другого порядку 26](#_Практична_робота_4.)

[Завдання, варіант № 14 26](#_Завдання,_варіант_№_2)

[Теоретичні відомості 26](#_Теоретичні_відомості_2)

[Результати виконання практичної роботи 28](#_Результати_виконання_практичної_2)

[Практична робота 5. Афінні перетворення у просторі 33](#_Практична_робота_5.)

[Завдання, варіант № 14 33](#_Завдання,_варіант_№_3)

[Теоретичні відомості 33](#_Теоретичні_відомості_3)

[Результати виконання практичної роботи 35](#_Результати_виконання_практичної_3)

[Практична робота 6. Візуалізація прямої задачі кінематики 43](#_Практична_робота_6.)

[Завдання, варіант № 14 43](#_Завдання,_варіант_№_4)

[Теоретичні відомості 43](#_Теоретичні_відомості_4)

[Результати виконання практичної роботи 45](#_Результати_виконання_практичної_4)

[Загальний перелік посилань 43](#_Загальний_перелік_посилань)

[Додаток А. Лістинг програми до практичної роботи №1 44](#_Додаток_А._Лістинг)

[Додаток Б. Лістинг програми до практичної роботи №2 48](#_Додаток_Г._Лістинг)

[Додаток В. Лістинг програми до практичної роботи №3 57](#_Додаток_В._Лістинг)

[Додаток Г. Лістинг програми до практичної роботи №4 65](#_Додаток_Г._Лістинг)

[Додаток Ґ. Лістинг програми до практичної роботи №5 76](#_Додаток_Ґ._Лістинг)

[Додаток Д. Лістинг програми до практичної роботи №6 95](#_Додаток_Д._Лістинг)

# Практична робота 1. Основні принципи роботи з OpenGL

## Завдання, варіант № 14

За допомогою інструментальних засобів, зазначених викладачем, створити простий програмний проєкт із підтримкою бібліотеки OpenGL. Розробити програму із застосуванням команд OpenGL, яка встановлює анізотропну систему координат, створює та виводить варіант зображення на екран/у вікно з урахуванням заданих примітивів та координат x1, y1 та x2, y2 . Для рисування координатної сітки необхідно використовувати пунктирні лінії. Контур фігури, осі та координатну сітку зобразити лініями різної товщини. Для парних варіантів точки повинні мати квадратну форму, а для непарних – круглу.

|  |  |  |
| --- | --- | --- |
| 14 | Примітиви:  GL\_POINTS, GL\_LINE\_STRIP  x1 = -2;  x2 = 7 y1 = -4;  y2 = 0 |  |

## Системна інформація

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

Processor AMD Ryzen 5 3600 6-Core Processor 3.60 GHz

RAM 16.0 GB

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

Edition Windows 10 Pro Version 22H2

IDE Microsoft Visual Studio Community 2022 (64-bit) version 17.8.6

## Теоретичні відомості

### Робота з графічними примітивами

Для відображення графічних елементів у OpenGL використовуються декілька команд, серед яких:

* glColor3d() — для задання кольору примітивів;
* glLineWidth() — для задання товщини ліній;
* glEnable() / glDisable() — для активації та деактивації різних функцій, наприклад, пунктирних ліній;
* glLineStipple() — для налаштування пунктирних ліній.

У нашому коді для малювання координатної сітки, осей та фігури використовувалися ці команди.

### Побудова координатної сітки

Для відображення сітки використовувалися пунктирні лінії, які створюються за допомогою функції glLineStipple(). Сітка дозволяє точно визначати позиції точок на площині та задавати контури фігур. Основні кроки для побудови сітки:

1. Встановлення кольору сітки командою glColor3d().
2. Використання функції glLineWidth() для задання товщини ліній.
3. Активація режиму пунктирних ліній командою glEnable(GL\_LINE\_STIPPLE).
4. Використання вектора для зберігання координат вершин сітки, який передається до OpenGL через функцію glVertexPointer().

### Малювання осей координат

Для побудови осей використовуються аналогічні методи з деякими змінами. Осі відображаються за допомогою товстих ліній, для яких задається інший колір і товщина. У нашому випадку осі зображаються жовтим кольором.

### Малювання полігонів та точок

В програмі передбачено малювання полігонів та окремих точок. Для цього були створені спеціальні класи, такі як Point та MyPolygon. Клас Point відповідає за відображення окремих точок на екрані, тоді як MyPolygon — за малювання полігонів, що складаються з кількох точок.

Для кожного примітиву використовується метод draw(), що викликає необхідні OpenGL-функції для відображення об’єктів.

## Результати виконання практичної роботи

### Розв'язання завдання

1. **Використання команд управління параметрами графічних примітивів:**

Для управління параметрами графічних примітивів були використані наступні команди (Додаток А):

* Колір: glColor3d(), рядки: 21, 49, 62, 112 та 127 у файлі Render.cpp.
* Тип: glLineStipple(), glEnable(), glDisable(), рядки: 23, 24 та 45 у файлі Render.cpp.
* Товщина: glLineWidth(), рядки: 22, 50, 63, 74, 87 та 128 у файлі Render.cpp.

1. **Коректне відображення завдання під час змінення розмірів/положення вікна:**

Коректне відображення завдання під час зміни розмірів вікна продемонстровано на рис. 1.1 та 1.2.

1. **Розроблення підпрограм для виключення дублювання коду:**

Підпрограми для уникнення дублювання коду були розроблені у рядках 20–28 файлу Render.cpp.

1. **Застосування циклів для створення зображень:**

Цикли для автоматичного малювання графічних елементів, таких як сітка, осі і відмітки, використовуються у рядках 35–46 та 86-93 файлу Render.cpp.

1. **Формування зображення векторними командами OpenGL:**

Векторні команди OpenGL, такі як glDrawArrays(), були використані для оптимізації рендерингу. У прикладі малювання сітки, осей та додаткових ліній використовується glDrawArrays() замість багатьох викликів окремих вершин. Це можна знайти у рядку 26 файлу Render.cpp.

1. **Використання ООП (розроблення власних класів):**

Використання об’єктно-орієнтованого підходу було реалізовано за допомогою класів Point та MyPolygon, які наведені у файлі Render.cpp (рядки 102 – 132). Клас Point відповідає за відображення точок, а клас MyPolygon — за побудову фігури.

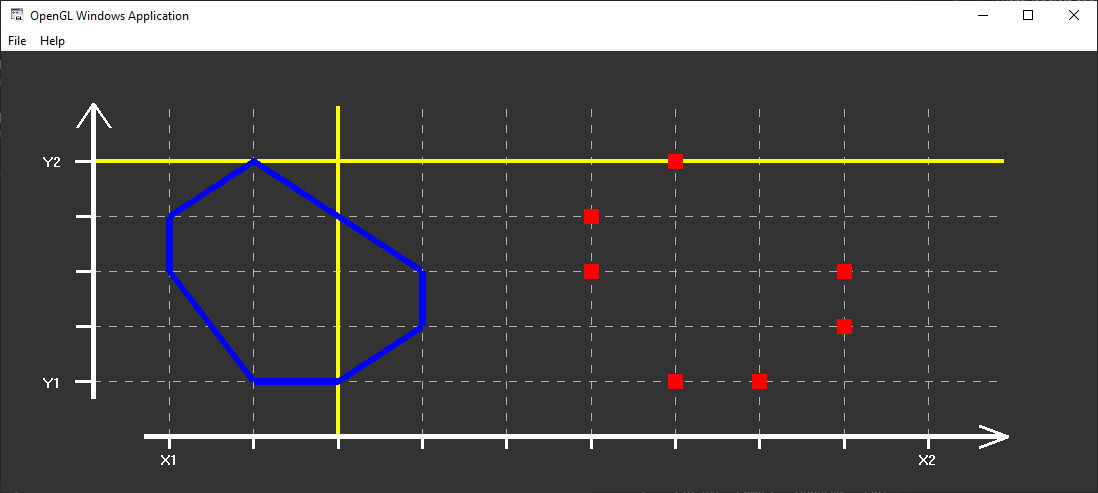


Рисунок 1.1 – Тестування програми при зміні ширини вікна

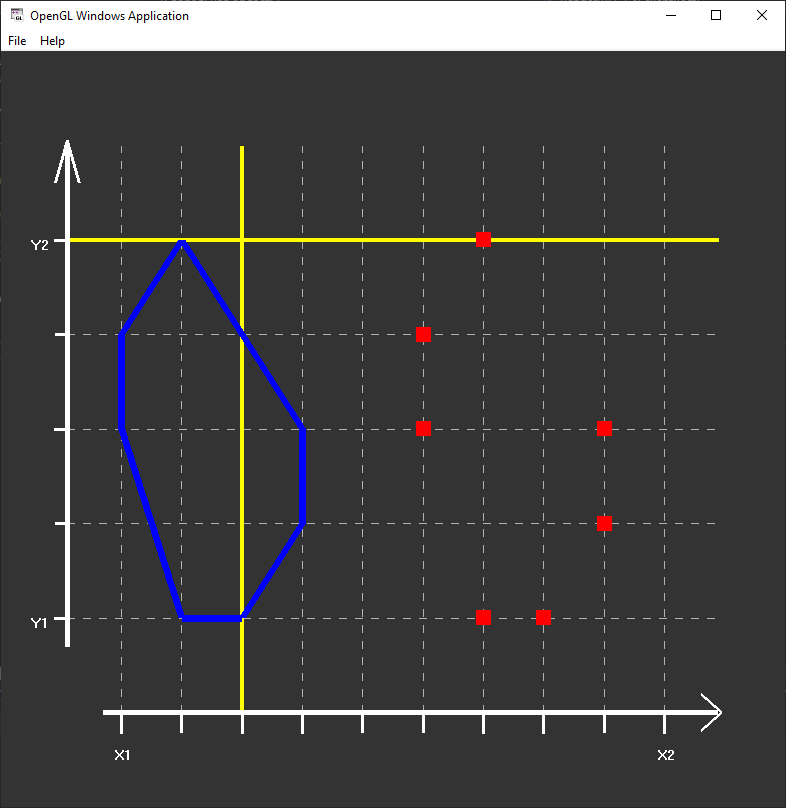


Рисунок 1.2 – Тестування програми при зміні висоти вікна

### Контроль виконання вимог та елементів завдання

В результаті виконання практичної роботи були повністю виконані елементи базового рівня та повністю підвищеного рівня складності, що відображено в таблиці 1.1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблиця 1.1 | | | | |
| № з/п | Складність | Вимоги | Бали | Зроблено |
| 1 | Базовий рівень | Використання команд управління параметрами графічних примітивів (колір, тип, товщина) | 2 | **+** |
| 2 | Коректне відображення завдання під час змінення розмірів/положення вікна | 1 | **+** |
| 3 | Розроблення підпрограм для виключення дублювання коду | 1 | **+** |
| 4 | Застосування циклів для створення зображень | 1 | **+** |
| 5 | Підвищений рівень | Формування зображення векторними командами *OpenGL* (*glDrawArrays* и т.п.) | 1 | **+** |
| 6 | Використання ООП (розроблення власних класів) | 2 | **+** |

### Посилання на GitHub

<https://github.com/Mausipupsi/OpenGL>

# Практична робота 2. Графічні примітиви OpenGL

## Завдання, варіант № 14

|  |  |  |
| --- | --- | --- |
|  | Сторона фігури a = 1000  Примітив(и):   GL\_TRIANGLES, GL\_QUAD\_STRIP | 04-02.PNG |

Використовуючи інструментальні засоби, що вказані викладачем, і беручи до уваги вимого, що наведено в табл. 2.1, створити програмний проєкт з підтримкою *OpenGL*. За допомогою команд ***glOrtho*****/*****gluOrtho2D*** і ***glViewport*** встановити для робочої області ізотропну систему координат з урахуванням розміру фігури, яку задано у варіанті (табл. 2.2). Після старту застосунок повинен відображати у робочій області одну плитку (***tile***). Приклад початкового стану застосунку показано на рис. 2.1.



Рисунок 2.1 – Вигляд застосунку після старту

Усі варіанти заданій основані на правильних багатокутниках, розмір яких визначається величиною одного ребра. Для зафарбування пропонується використовувати шість кольорів: білий, сірий (35 %), червоний, зелений, синій и жовтий.

За допомогою клавіатури або маніпулятора «миша» користувач повинен мати можливість виконати замощення (***tessellation***, ***tilling***) робочої області по горизонталі і вертикалі. При цьому систему координат необхідно скорегувати таким чином, щоб замощена поверхня розташовувалася у центрі робочої області. Приклад замощення робочої області застосунку показано на рис. 2.2.



*а*



*б*

Рисунок 2.2 – Вигляд застосунку під час замощення:   
*а* – тільки по горизонталі; *б* – по горизонталі та вертикалі

Крім цього, користувач повинен мати можливість змінювати режим відображення графічних примітивів *OpenGL*: точкове (тільки вершини фігури), контурне (рис. 2.3) і з заповненням кольором (див. рис. 2.2). Передбачається, що перемикання між режимами виконують за подією від клавіатури і/або маніпулятора «миша». При цьому можна використовувати як стандартні елементи керування, так і власні, що реалізовані та відображені засобами *OpenGL* (для підвищеного рівня складності, див. табл. 2.1).



Рисунок 2.3 – Приклад керування режимом виводу графічних примітивів

## Теоретичні відомості

### Windows Forms (WinForms)

Windows Forms — це графічний інтерфейс користувача (GUI) для створення додатків у середовищі .NET. Він надає набір елементів керування (контролів), таких як кнопки, текстові поля, радіокнопки тощо, які можна використовувати для побудови інтерфейсу користувача.

* InitializeComponent(): Метод, який автоматично генерується дизайнером WinForms. Він ініціалізує всі елементи керування на формі, встановлюючи їхні властивості та розташування.

### Події та Обробники Подій

У програмуванні подійна модель дозволяє реагувати на дії користувача, такі як натискання кнопок, зміна значень контролів тощо.

* ValueChanged: Подія, яка виникає при зміні значення елемента керування, наприклад, NumericUpDown. Обробник цієї події використовується для виконання певних дій у відповідь на зміну значення.
* CheckedChanged: Подія, яка виникає при зміні стану радіокнопки (RadioButton). Використовується для визначення, яка радіокнопка вибрана, та відповідного оновлення режиму рендерингу.
* RenderModeChanged: Користувацька подія, яка сповіщає про зміну режиму рендерингу в контролі рендерингу. Дозволяє синхронізувати стан інтерфейсу з поточним режимом рендерингу.

### OpenGL

OpenGL (Open Graphics Library) — це стандартний API для рендерингу 2D та 3D графіки. Він надає функції для малювання геометричних фігур, управління камерою, освітленням та іншими аспектами графічного відображення.

* glClear: Очищує буфери кольору та глибини, підготовлюючи кадр для нової сцени.
* glViewport: Встановлює розмір в'юпорту — області екрану, куди буде рендеритися сцена.
* glMatrixMode: Встановлює поточну матрицю (наприклад, проекційна або модельно-видова).
* glLoadIdentity: Завантажує одиничну матрицю, скидаючи попередні трансформації.
* glOrtho: Встановлює ортографічну проекцію, яка використовується для 2D відображення без перспективних спотворень.
* glBegin / glEnd: Обгортки для визначення початку та кінця малювання примітивів (трикутників, квадрацій тощо).
* glColor3f: Встановлює колір для наступних вершин.
* glVertex2f: Визначає вершину примітива у 2D просторі.
* glFlush: Забезпечує негайне виконання команд рендерингу.

### Режими Рендерингу

Режими рендерингу визначають спосіб, у який OpenGL відображає геометричні примітиви.

* Fill (Заповнення): Примітиви заповнюються кольором, визначеним за допомогою glColor3f. Використовується для відображення суцільних фігур.
* Line (Лінії): Примітиви відображаються лише контурами (гранями), без заповнення. Застосовується для контурних малюнків.
* Point (Точки): Примітиви відображаються як окремі точки. Використовується для створення точкових зображень або ефектів.

### В'юпорт та Проекція

В'юпорт — це область екрану, куди рендериться сцена. Проекція визначає, як тривимірні координати перетворюються у двовимірні координати екрану.

* Ортографічна Проекція (glOrtho): Тип проекції, де паралельні лінії залишаються паралельними після проекції. Підходить для 2D графіки та технічних креслень.
* Співвідношення Сторін (Aspect Ratio): Відношення ширини до висоти в'юпорту. Важливе для збереження пропорцій об'єктів при зміні розміру вікна.

### Малювання Геометричних Фігур

У коді використовується клас ComplexFigure для малювання складних фігур, які складаються з трикутників та квадрейтрипів.

* Трикутники (GL\_TRIANGLES): Основні примітиви для створення складних форм. Три вершини визначають один трикутник.
* Квадрейтрипи (GL\_QUAD\_STRIP): Використовуються для малювання послідовних чотирикутників, спрощуючи процес створення смуг фігур.

### Сітка Фігур

Сітка фігур дозволяє організувати розташування геометричних об'єктів у певній конфігурації, наприклад, гексагональній сітці.

* Гексагональна Сітка: Тип сітки, де кожна фігура (наприклад, шестиугольник) оточена шістьма іншими фігурами. Забезпечує рівномірне розподілення простору та відсутність пропусків.
* Зміщення Позицій (Offsetting Positions): Використовується для створення ефекту гексагонального розміщення, де кожен ряд змінює вертикальну позицію фігур для досягнення щільного розташування.

### Подвійна Буферизація

Подвійна буферизація — техніка, яка використовує два буфери для рендерингу кадрів: один для відображення поточного кадру, а інший для підготовки наступного. Це дозволяє уникнути мерехтіння та забезпечити плавний візуальний перехід між кадрами.

* DoubleBuffered: Властивість контролю WinForms, яка визначає, чи використовувати подвійне буферизування для зменшення мерехтіння при малюванні.

### Класи та Об'єктно-орієнтоване Програмування (ООП)

Код використовує принципи ООП для організації функціональності.

* Клас Button: Представляє кнопку вибору режиму рендерингу. Містить властивості для позиції та режиму, а також методи для малювання та перевірки натискання.
* Клас ComplexFigure: Відповідає за створення та малювання складних геометричних фігур. Використовує методи OpenGL для визначення вигляду фігур залежно від вибраного режиму рендерингу.
* Класи та Інкапсуляція: Використання класів дозволяє інкапсулювати дані та методи, забезпечуючи модульність та повторне використання коду.

### Обробка Подій Миші

Обробка подій миші дозволяє взаємодіяти з користувачем через натискання, переміщення або інші дії миші.

* MouseDown: Подія, яка виникає при натисканні кнопки миші. Використовується для визначення, чи було натиснуто одну з кнопок вибору режиму рендерингу.
* IsPointInside(float x, float y): Метод, який перевіряє, чи знаходиться певна точка (координати миші) всередині області кнопки. Дозволяє визначити, яка кнопка була натиснута.

## Результати виконання практичної роботи

### Розв'язання завдання

1. **Під час запуску застосунку зображення відповідає варіанту завдання з однією плиткою:**

При запуску застосунку за замовчуванням встановлюється кількість плиток по горизонталі та вертикалі рівною 1. Це забезпечує відображення єдиної плитки, яка відповідає початковому варіанту завдання. Файл MainForm.cs, рядки 10-20. Продемонстровано на рис. 2.1 та 2.2.

1. **Багаторазове замощення плиткою. Кратність замощення задається користувачем під час роботи застосунку:**

Користувач може змінювати кількість плиток по горизонталі та вертикалі за допомогою елементів NumericUpDown. Зміни значень викликають оновлення кількості плиток у контролі рендерингу. Багаторазове замощення плиткою продемонстровано на рис. 2.5 та 2.6. Файл MainForm.cs, рядки 22-40 та файл RenderControl.cs, рядки 182-213.

1. **Коректне відображення завдання під час зміни як розмірів/положення вікна, так і параметрів замощення:**

Контроль рендерингу реагує на події зміни розміру вікна та змін параметрів замощення, перерисовуючи зображення відповідно до нових параметрів. Продемонстровано на рис. 2.1 - 2.6. Файл RenderControl.cs, рядки 124-132 та 154-174.

1. **Організація взаємодії з користувачем одним зі стандартних засобів (клавіатура, «миша» та ін.):**

Взаємодія з користувачем здійснюється через мишу. Користувач може натискати на кнопки режиму рендерингу, використовуючи події миші для визначення натискання на відповідні області. Файл RenderControl.cs, рядки 63-82.

1. **Застосування мінімальної (у рамках варіанту) кількості графічних примітивів для виконання завдання:**

Для створення необхідних фігур використовується мінімальна кількість графічних примітивів OpenGL, таких як трикутники та квадрати. Файл RenderControl.cs, рядки 332-357.

1. **Створення власних елементів інтерфейсу за допомогою OpenGL:**

Власні елементи інтерфейсу, такі як кнопки вибору режиму рендерингу, створені та намальовані вручну за допомогою команд OpenGL. Файл RenderControl.cs, рядки 228-301.

1. **Використання ООП (розробка власних класів):**

Об'єктно-орієнтований підхід застосовується через створення власних класів Button та ComplexFigure, які інкапсулюють властивості та методи, необхідні для їхньої функціональності. Файл RenderControl.cs, рядки 228-401.

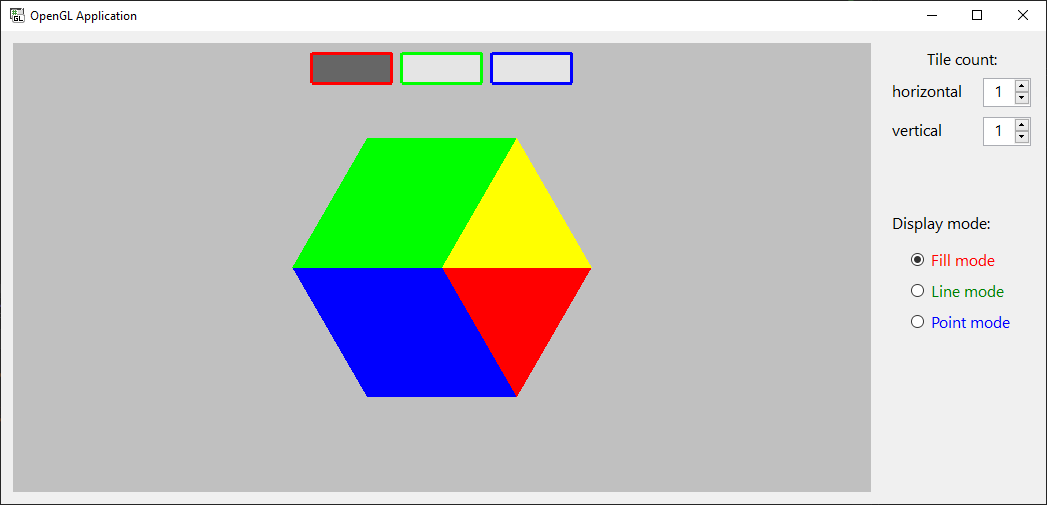


Рисунок 2.1 – Тестування програми при зміні ширини вікна та режимі Fill

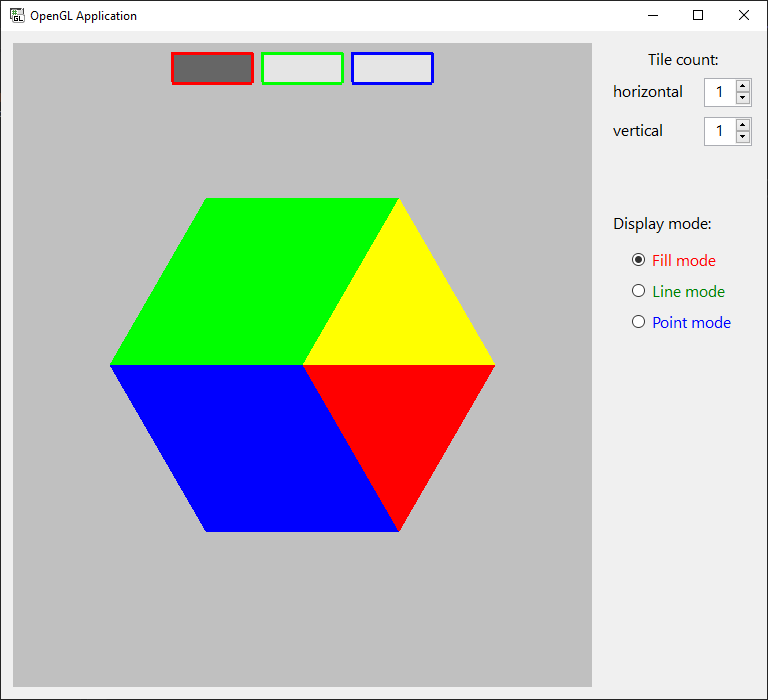


Рисунок 2.2 – Тестування програми при зміні висоти вікна та режимі Fill

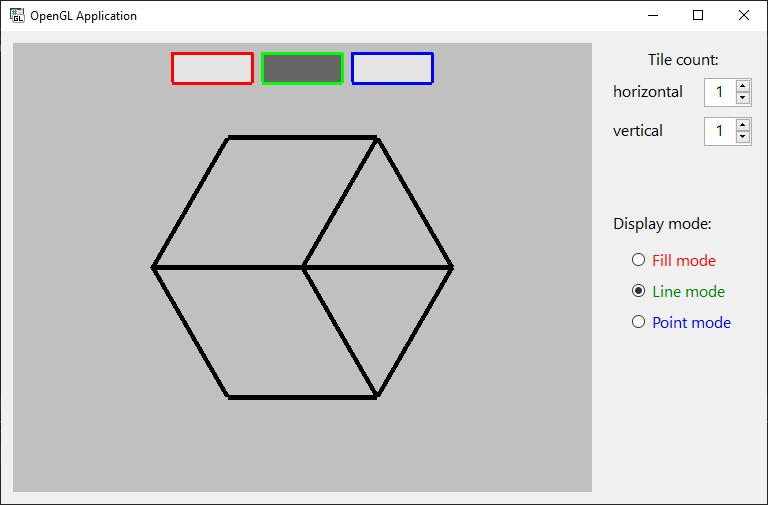


Рисунок 2.3 – Тестування програми в режимі Line

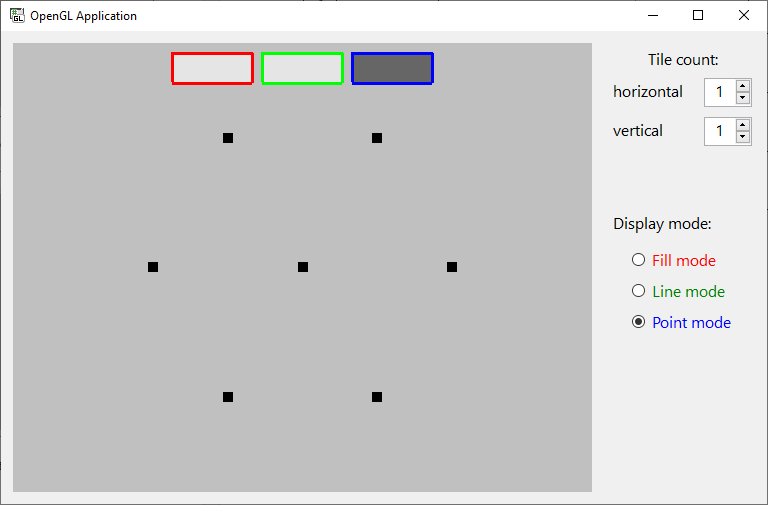


Рисунок 2.4 – Тестування програми в режимі Point

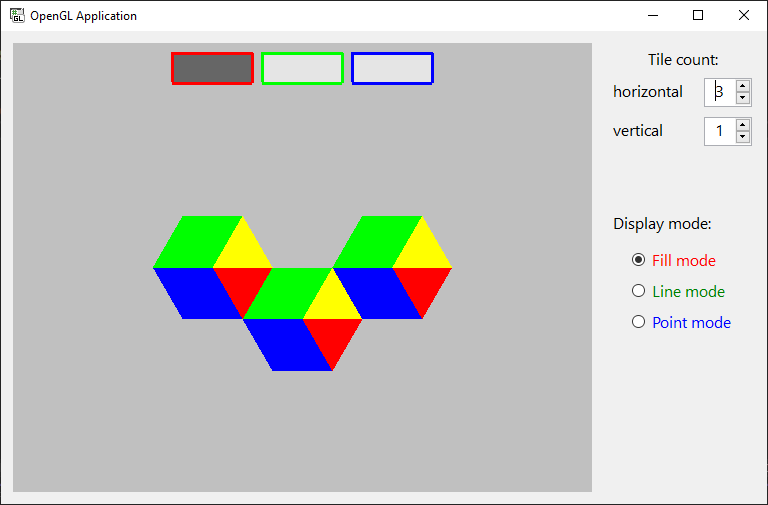


Рисунок 2.5 – Тестування програми при додаванні фігур по горизонталі

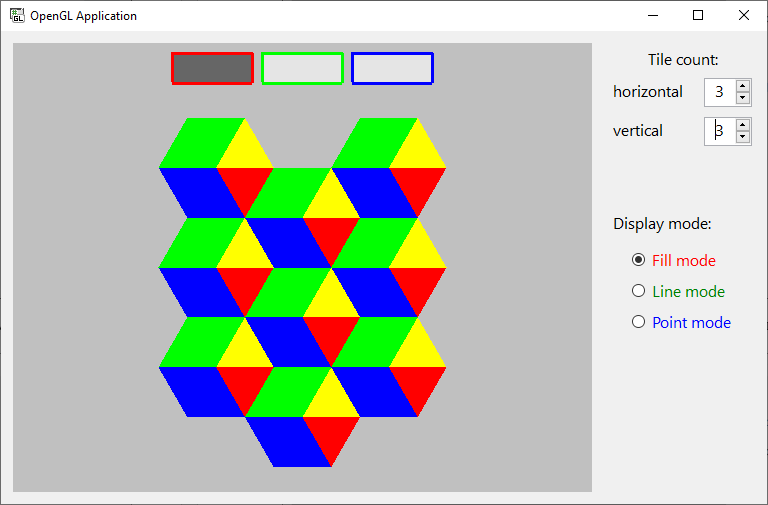


Рисунок 2.6 – Тестування програми при додаванні фігур по вертикалі

### Контроль виконання вимог та елементів завдання

В результаті виконання практичної роботи були повністю виконані елементи базового рівня та повністю підвищеного рівня складності, що відображено в таблиці 2.1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблиця 2.1 | | | | |
| № з/п | Складність | Вимоги | Бали | Зроблено |
| 1 | Базовий рівень | Під час запуску застосунку зображення відповідає варіанту завдання з однією плиткою (див. рис. 2.1) | 1 | **+** |
| 2 | Багаторазове замощення плиткою (див. рис.2.2). Кратність замощення задається користувачем під час роботи застосунку. | 1 | **+** |
| 3 | Коректне відображення завдання під час зміни як розмірів/положення вікна, так і параметрів замощення | 1 | **+** |
| 4 | Організація взаємодії з користувачем одним зі стандартних засобів (клавіатура, «миша» та ін.) | 1 | **+** |
| 5 | Застосування мінімальної (у рамках варіанту) кількості графічних примітивів для виконання завдання | 1 | **+** |
| 6 | Підвищений рівень | Створення власних елементів інтерфейсу за допомогою OpenGL | 2 | **+** |
| 7 | Використання ООП (розробка власних класів) | 1 | **+** |

### Посилання на GitHub

<https://github.com/Mausipupsi/OpenGL>

# Практична робота 3. ГРАФІК ФУНКЦІЇ ОДНІЄЇ ЗМІННОЇ

## Завдання, варіант № 14

| Номер варіанта | Функція ***f1(x)*** |
| --- | --- |
|  |  |

| Номер варіанта | Функція ***f2(x)*** |
| --- | --- |
|  |  |

Використовуючи інструментальні засоби, що вказані викладачем, розробити програму для побудови графіка функції виду на довільному інтервалі від ***Xmin*** до ***Xmax*** і відображення точок перетину функції з віссю абсцис. Крім того, програма повинна мати такі можливості(табл.3.1):

* дозволяти користувачу задавати інтервал від ***Xmin*** до***Xmax*** з перевіркою***Xmin < Xmax***;
* виконувати для завданого користувачем інтервалу від ***Xmin*** до ***Xmax*** автоматичне масштабування за віссю ***Y*** (додатково допускається наявність ручного режиму встановлення ***Ymin*** і ***Ymax***);
* відображати осі координат (та/або координатну сітку) з виводом значень меж видимої області ***Xmin***, ***Xmax***, ***Ymin*** і ***Ymax***, при цьому система координат повинна бути анізотропною;
* відображати усі точки, де , якщо вони є на завданому інтервалі від ***Xmin*** до ***Xmax***.

Приклад інтерфейсу користувача наведено на рис. 3.1. Варіанти функції базового рівня складності наведені в табл. 3.2

Для підвищеного рівня складності необхідно додатково реалізувати коректне виведення функції з урахуванням області визначення функції (варіанти наведені в табл. 3.3) і відобразити лінії розриву (рис. 3.2).

## Теоретичні відомості

### Побудова графіків функцій

Для побудови графіків функцій у програмі використовуються інструменти OpenGL, які забезпечують відображення координатної сітки, осей, графіків функцій та точок нулів. Основні функції OpenGL, що використовуються в програмі:

* glColor3d() — задає колір об'єкта (сітка, осі, графік).
* glLineWidth() — встановлює товщину ліній.
* glEnable() / glDisable() — активує та деактивує функції, наприклад, режим пунктирних ліній.
* glLineStipple() — налаштовує пунктирні лінії, що використовуються для позначення розривів.

### Відображення координатної сітки

Сітка на площині дозволяє візуально оцінити положення графіка функції. Її побудова реалізована наступними кроками:

* Задання кольору сітки за допомогою glColor3d().
* Використання glLineWidth() для встановлення товщини сіткових ліній.
* Малювання горизонтальних і вертикальних ліній у вигляді лінійних примітивів за допомогою функцій OpenGL.

### Побудова осей координат

Координатні осі є базовими орієнтирами для графіка. Для їх відображення використовуються такі прийоми:

* Осі відображаються чорним кольором за допомогою glColor3d().
* Для підкреслення важливості осей використовується товстіша лінія, яка задається через glLineWidth().
* Лінії осей малюються вздовж горизонталі та вертикалі через початок координат.

### Відображення розривів функції

Для функцій, які мають точки розриву (наприклад, *f*2(x)), ці розриви позначаються пунктирними лініями:

* Активація режиму пунктирних ліній за допомогою glEnable(GL\_LINE\_STIPPLE).
* Налаштування пунктирного малюнка через glLineStipple().
* Малювання вертикальних ліній у місцях розривів.

### Побудова графіка функції

Графік функції створюється шляхом побудови лінійного примітиву GL\_LINE\_STRIP, де кожна точка графіка задається функцією:

* Обчислення значень функції в заданому діапазоні Xmin​ до Xmax​.
* Перевірка на наявність допустимих значень (NaN та ∞).
* Розрив графіка при знаходженні точок розриву.

### Відображення нулів функції

Точки перетину графіка з віссю абсцис (f(x)=0) визначаються шляхом аналізу зміни знаку функції:

* Використання лінійної інтерполяції для точного визначення положення нуля.
* Відображення нулів на графіку у вигляді синіх точок із заданим розміром через glPointSize().

### Основні класи програми

* FunctionBase - абстрактний клас для опису функцій.
* Function1 та Function2 - похідні класи, що реалізують конкретні функції.
* RenderControl - клас для малювання графіків, сітки, осей та позначення точок розриву.

## Результати виконання практичної роботи

### Розв'язання завдання

1. **Виведення осей координат і графіка функції *f*1(x) на заданому користувачем інтервалі:**

При зміні значень Xmin​, Xmax​, Ymin​, Ymax​, обраних через NumericUpDown, графік функції *f*1(x) оновлюється відповідно до цих значень. Осі координат автоматично підлаштовуються під новий інтервал.  
Файл MainForm.cs, рядки 67-147. Продемонстровано на рис. 2.1.

1. **Автоматичні обчислення Ymin​ і Ymax​:**

Програма автоматично розраховує межі осі Y на основі значень функції *f*1(x) у заданому користувачем інтервалі [Xmin, Xmax]. Значення Ymin​ та Ymax​ обчислюються під час проходження по точках функції.  
Файл RenderControl.cs, рядки 54-58. Приклад відображено на рис. 2.2.

1. **Обчислення та відображення точок *f*1(x)=0:**

Для обчислення точок нулів використовується лінійна інтерполяція між сусідніми точками, де функція змінює знак. Ці точки позначаються синіми маркерами на графіку.  
Файл RenderControl.cs, рядки 165-189. Демонстрація показана на рис. 2.1, 2.2 та 2.3.

1. **Коректне виведення графіка *f*2​(x):**

Графік *f*2​(x) будується з урахуванням її області визначення. Точки розриву визначаються за формулою періодичності та відображаються пунктирними вертикальними лініями. Точки розриву не враховуються як точки нулів. Файл Function2.cs, та RenderControl.cs, рядки 123-163. Приклад показано на рис. 2.3.

1. **Використання ООП:**

У програмі застосовується об’єктно-орієнтований підхід для роботи з функціями:

* Створено абстрактний клас FunctionBase, який визначає базовий метод Evaluate() для обчислення значення функції та віртуальний метод GetDiscontinuities() для обробки точок розриву.
* Метод GetDiscontinuities() реалізований як віртуальний, що дозволяє перевизначити його в похідних класах, таких як Function2. Наприклад, у Function2 цей метод використовується для обчислення періодичних точок розриву функції *f*2​(x), тоді як у Function1 він не перевизначається, оскільки функція *f*1​(x) не має розривів.

Файл FunctionBase.cs, рядки 9-16 (визначення віртуального методу), Function2.cs, рядки 15-32 (перевизначення методу).

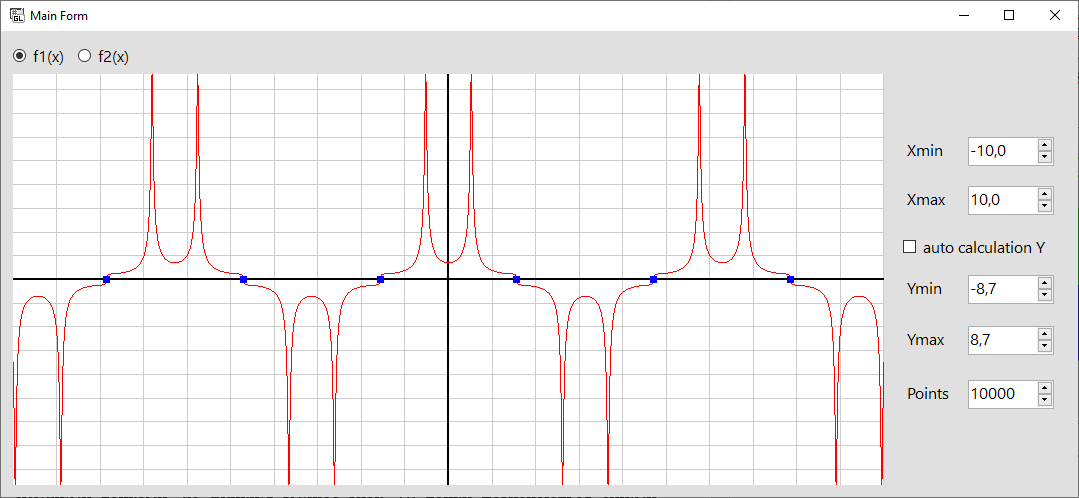


Рисунок 3.1 – Тестування програми при зміні ширини вікна та Y заданий користувачем

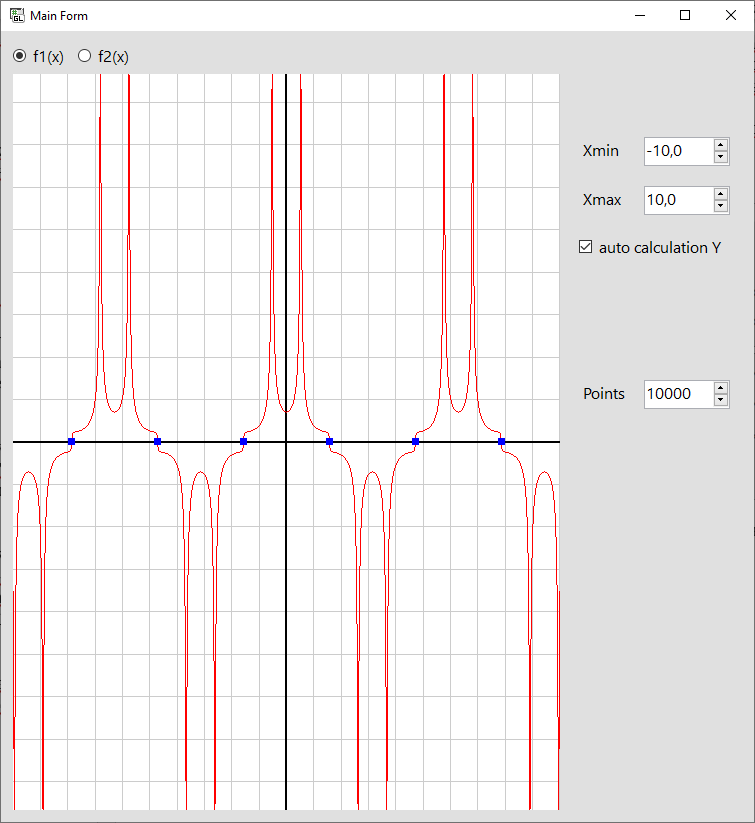


Рисунок 3.2 – Тестування програми при зміні висоти вікна та автоматичне обчислення Y

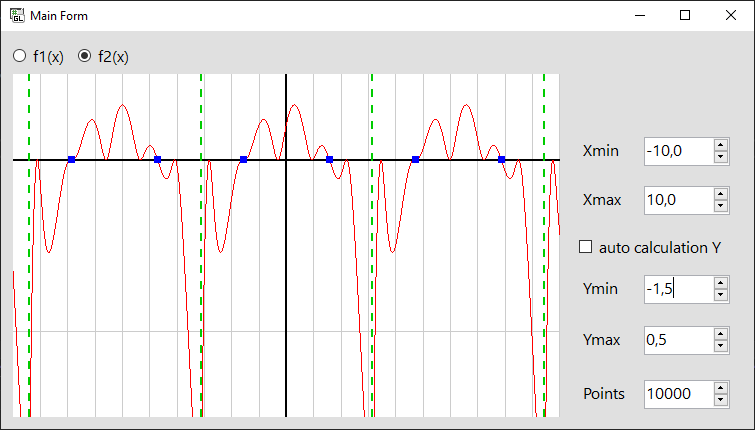


Рисунок 3.3 – Тестування програми з другою функцією та розривами

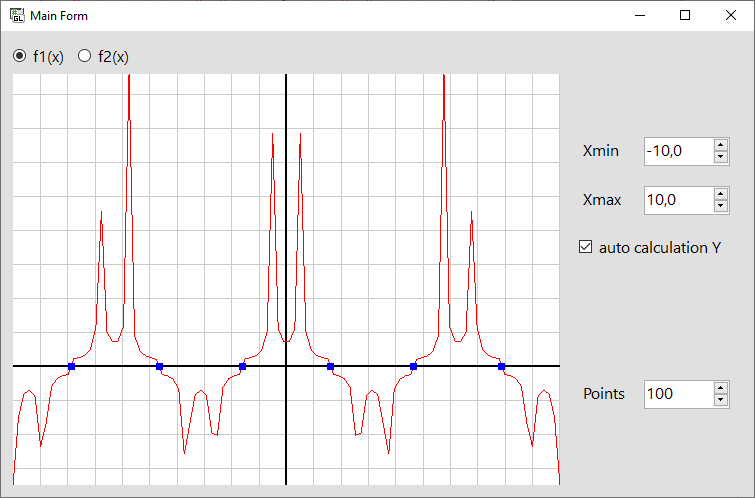


Рисунок 3.4 – Тестування програми при зміні points

### Контроль виконання вимог та елементів завдання

В результаті виконання практичної роботи були повністю виконані елементи базового рівня та повністю підвищеного рівня складності, що відображено в таблиці 3.1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблиця 3.1 | | | | |
| № з/п | Складність | Вимоги | Бали | Зроблено |
| 1 | Базовий рівень | Осі координат і графік функції ***f1(x)*** виводяться на заданому користувачем інтервалі від *Xmin* до *Xmax* і від *Ymin* до *Ymax* | 1 | **+** |
| 2 | Автоматичні обчислення *Ymin* і *Ymax* на завданому інтервалі від *Xmin* до *Xmax* функції ***f1(x)*** | 2 | **+** |
| 3 | Обчислення і виведення на екран точок *f1(x)* = 0 | 2 | **+** |
| 4 | Підвищений рівень | Коректне виведення графіка ***f2(x)***(без хибного виводу точок розриву як точок перетину з віссю абсцис) і з відображенням ліній розриву функції | 2 | **+** |
| 5 | Використання ООП (наслідування, використання віртуальних і абстрактних методів) | 1 | **+** |

### Посилання на GitHub

<https://github.com/Mausipupsi/OpenGL>

# Практична робота 4. КРИВІ ДРУГОГО ПОРЯДКУ

## Завдання, варіант № 14

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Номер варіанта | Окружність | | Еліпс | | Гіпербола | | Парабола | |
| Подання | | Подання | | Подання | | Подання | |
| явне | парамет­ричне | явне | парамет­ричне | явне | парамет­ричне | явне | парамет­ричне |
| 14 |  |  | + |  |  |  |  | ++ |

Використовуючи інструментальні засоби, вказані викладачем, розробити програму для виведення кривих другого порядку на екран   
(у вікно *Windows*) за допомогою відрізків. Систему оцінювання наведено в табл. 4.1, а варіанти завдань – в табл. 4.2. Для кривих, які у варіанті відмічені «++», знайти та вивести на екран точки перетину, якщо такі є, з довільним відрізком, координати якого задає користувач.

## Теоретичні відомості

### Криві другого порядку

Криві другого порядку – це множини точок, які задовольняють рівнянням другого ступеня. Вони включають еліпси, параболи, гіперболи та окремі випадки, такі як коло та пари прямих.

### Визначення та властивості

Еліпс – це геометричне місце точок, сума відстаней від яких до двох заданих точок (фокусів) є сталою. Рівняння еліпса у канонічній формі:



де:

* h , k – координати центру;
* a – довжина великої півосі;
* b – довжина малої півосі.

Основні властивості еліпса:

Основні властивості еліпса:

* Якщо a = b, еліпс стає колом.
* Фокуси розташовані на великій осі на відстані ​ від центру.
* Еліпс є замкнутою кривою, симетричною відносно осей.

Парабола – це множина точок, рівновіддалених від заданої прямої (директриси) та точки (фокуса). Її канонічне рівняння:



де A,B,C – коефіцієнти, які визначають форму та положення.

Основні властивості параболи:

* Вершина параболи – це точка, у якій напрямок змінюється. Її координати:



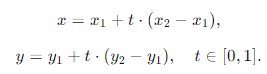
* Симетрія відносно осі параболи.
* Використовується для моделювання руху тіл під дією гравітації, світлових і звукових хвиль.

### Пошук точок перетину з відрізком

Для знаходження точок перетину кривої другого порядку з довільним відрізком використовується параметричний метод.

Рівняння відрізка:

Відрізок визначається двома точками (x1,y1) і (x2,y2​) у параметричній формі:



Методика розв’язання:

* Підставляємо рівняння відрізка в рівняння кривої.
* Отримуємо рівняння відносно параметра t.
* Розв’язуємо це рівняння (квадратне для еліпса та параболи).
* Перевіряємо, чи належать розв’язки інтервалу t∈[0,1].

Критерій існування точок перетину:  
Точки перетину існують, якщо дискримінант рівняння щодо ttt є невід’ємним (D ≥ 0).

### Візуалізація кривих

Для відображення кривих другого порядку використовується метод побудови кривих з набору відрізків. Криві малюються з використанням OpenGL:

* **Еліпс**: обчислюються координати точок за допомогою параметричного рівняння:



* **Парабола**: обчислюються точки для рівняння на заданому інтервалі.

Для кожної кривої виділяються точки перетину з відрізком, які відображаються червоним кольором.

## Результати виконання практичної роботи

### Розв'язання завдання

1. **Установлення ізотропної системи координат для вікна з змінюваними розмірами:**

Виконання: Система координат налаштовується у методі RenderControl\_Render класу RenderControl.

Файл: RenderControl.cs. Рядки: 34–51

Налаштовується видова область (glViewport) відповідно до розміру вікна. Встановлюється ортогональна проєкція (glOrtho), яка забезпечує ізотропність, тобто однаковий масштаб по осях X та Y. Обчислення ширини і висоти світу залежно від пропорцій вікна.

1. **Виведення кривих другого порядку відповідно до варіанту завдання​:**

Виконання: Відображення еліпса та параболи реалізовано в методі RenderControl\_Render.

Файл: RenderControl.cs. Рядки:

- Для еліпса: 91–110

- Для параболи: 122–140

Еліпс будується за параметричними рівняннями x = h + a cos(theta), y = k + b sin(theta). Парабола будується за рівнянням y = Ax^2 + Bx + C у вигляді ламаної лінії з відрізків. Обидві криві відповідають варіанту завдання.

1. **Виведення відрізка та обчислення його точок перетину з кривою другого порядку відповідно до варіанту:**

Виконання: Відрізок малюється як лінія між точками (AX, AY) і (BX, BY). Точки перетину обчислюються окремими методами для еліпса та параболи.

Файл: RenderControl.cs. Рядки:

- Малювання відрізка: 72–77

- Обчислення точок перетину:

- Для еліпса: Ellipse.GetIntersectionPoints (файл Ellipse.cs, рядки 20–71).

- Для параболи: Parabola.GetIntersectionPoints (файл Parabola.cs, рядки 24–84).

Для обчислення точок перетину відрізка з кривими використовується параметричний метод. Точки перетину перевіряються на належність відрізку (параметр t є [0, 1]).

1. **Вказання положення точок, що формують відрізок, в області графічного виведення програми за допомогою маніпулятора «миш»:**

Виконання: Координати точок відрізка задаються кліком миші.

Файл: RenderControl.cs. Рядки: 155–182

При натисканні миші координати перетворюються з екранних у світові за допомогою методу ScreenToWorld. Перша дія встановлює точку A, друга – точку B. Після оновлення точок відрізка контроль перемальовується.

1. **Використання ООП:**

Виконання: У програмі реалізовано принципи об’єктно-орієнтованого програмування.

Файли:

- Ellipse.cs: Клас Ellipse представляє еліпс.

- Parabola.cs: Клас Parabola представляє параболу.

- Segment.cs: Клас Segment представляє відрізок.

- RenderControl.cs: Клас RenderControl обробляє рендеринг та взаємодію з користувачем.

Кожна крива та відрізок представлені окремими класами з властивостями та методами. Інкапсуляція реалізована через використання публічних методів для доступу до даних. Поліморфізм застосовується у методах рендерингу та обчислення точок перетину.

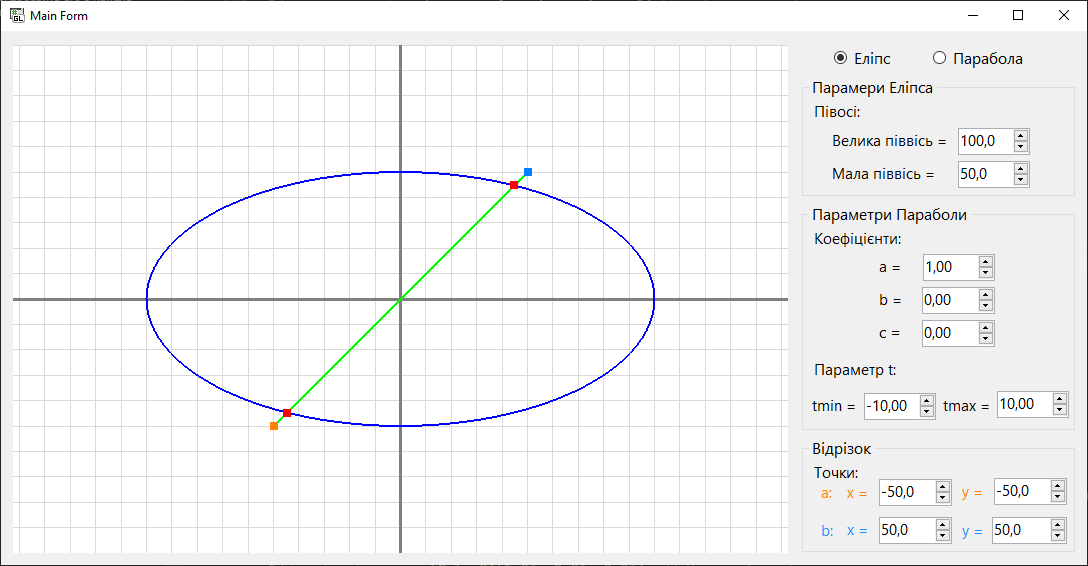


Рисунок 4.1 – Тестування програми при зміні ширини вікна та Еліпс

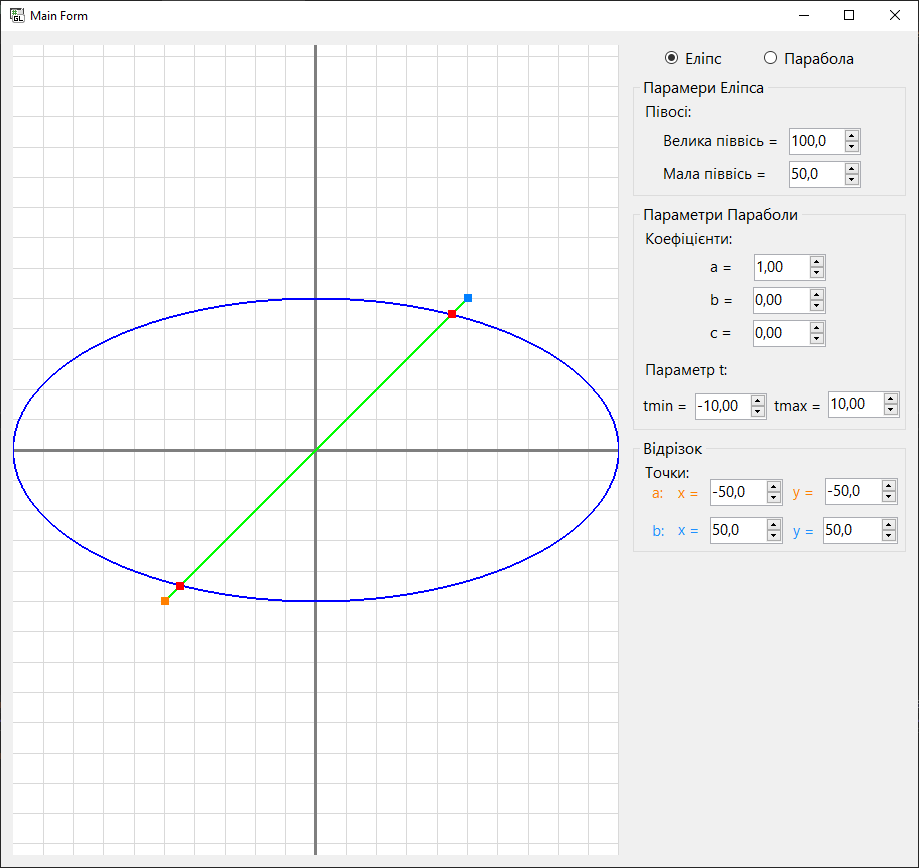


Рисунок 4.2 – Тестування програми при зміні висоти вікна та Еліпс

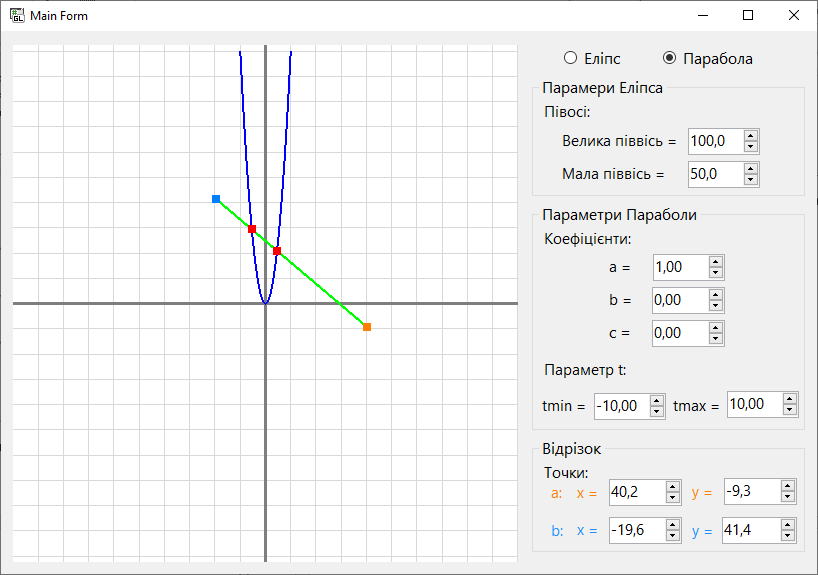


Рисунок 4.3 – Тестування програми з параболою

### Контроль виконання вимог та елементів завдання

В результаті виконання практичної роботи були повністю виконані елементи базового рівня та повністю підвищеного рівня складності, що відображено в таблиці 4.1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблиця 4.1 | | | |  |
| № п/п | Складність | Вимоги | Бали | Зроблено |
| 1 | Базовий рівень | Установлення ізотропної системи координат для вікна з змінюваними розмірами | 1 | **+** |
| 2 | Виведення кривих другого порядку відповідно до варіанту завдання | 2 | **+** |
| 3 | Виведення відрізка та обчислення його точок перетину з кривою другого порядку відповідно до варіанту | 2 | **+** |
| 4 | Підвищений рівень | Вказання положення точок, що формують відрізок, в області графічного виведення програми за допомогою маніпулятора «миш» | 2 | **+** |
| 5 | Використання ООП | 1 | **+** |

### Посилання на GitHub

<https://github.com/Mausipupsi/OpenGL>

# Практична робота 5. КвадратичНІ примІтивИ. АфІНнІ ПЕРЕТВОРЕННЯ У простОрІ

## Завдання, варіант № 14

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Номер  варіанта | *Grid* | Фігура | Параметри квадратичних фігур | | | | | | | | |
| ось | *x0* | *y0* | *z0* | *R* | *r* | *h* | *∠start* | *∠sweep* |
| 14 | X0Z | сфера | ∥ 0Z | +2.5 | +1.0 | -2.5 | 1.5 | - | - | - | - |
| усічений конус | ⇈ 0Y | -3.0 | -2.5 | -4.5 | 3.0 | 1.0 | 2.5 | - | - |
| частинний диск | ∥ 0X | -3.5 | +0.5 | +2.0 | 2.0 | 0.0 | - | 225° | 180° |

Використовуючи інструментальні засоби, вказані викладачем, розробити програму з використання засобів *OpenGL*, яка встановлює ізотропну систему координат, створює і виводить зображення тривимірної сцени з такими елементами (систему оцінки наведено в табл. 5.1, а варіанти завдань – в табл.5.2):

* осі координат з нулем у центрі екрана та вказанням осі та додатного напрямку;
* координатна сітка (***grid***) в одній з площин (***X0Y***, ***X0Z*** чи ***Y0Z***);
* три квадратичні фігури – ***gluDisk / gluPartialDisk***, ***gluSphere***, ***gluCylinder*** в режимі відображення каркаса і з спрощеною моделлю освітлення ***glEnable(GL\_COLOR\_MATERIAL)*** для базового рівня складності;
* площина відтину для однієї з фігур (сфера, циліндр чи конус);
* повноцінна модель освітлення та/або текстурами для реалізації завдання з підвищеною складністю.

Мінімальний інтерфейс користувача повинен забезпечувати можливості повороту сцени відносно осей ***OX*** і ***OY*** за допомогою маніпулятора «миш» і керування параметрами площини відтину [1, 2]. Параметри деталізації об’єктів (***slices, stacks***), кольору, товщини і типу ліній обирають самостійно. Приклад сцени показано на   
рис. 5.1.

## Теоретичні відомості

### Квадратичні Примітиви

Квадратичні примітиви є геометричними об'єктами, визначеними квадратичними рівняннями у тривимірному просторі. Вони широко використовуються в комп'ютерній графіці для моделювання складних форм завдяки їхній математичній простоті та можливості точного опису криволінійних поверхонь. Основні квадратичні примітиви включають:

1. **Сфера**: Описується рівнянням (x−x0​)2+(y−y0​)2+(z−z0​)2=R2, де (x0​,y0​,z0​) — центр сфери, а R — радіус. У OpenGL сфера створюється за допомогою функції gluSphere, яка генерує вершини на поверхні сфери, розбиті на сектори (slices) та стеки (stacks) для контролю деталізації.
2. **Конус**: Усічений конус або циліндр визначається рівняннями, що включають радіуси верхньої та нижньої основ та висоту. У OpenGL для створення усіченого конуса використовується функція gluCylinder, де параметри R та r представляють радіуси основ, а h — висоту.
3. **Диск**: Частковий або повний диск описується радіусом та кутовими параметрами початку та розгортки (∠start​ та ∠sweep). У OpenGL диск створюється за допомогою функції gluPartialDisk, яка дозволяє визначати сектор диска за допомогою кутів.

### Афінні Перетворення в Просторі

Афінні перетворення — це клас лінійних перетворень, які включають комбінації масштабування, повороту, перенесення та зсуву об'єктів у тривимірному просторі. Вони зберігають колінеарність та паралельність ліній, що дозволяє точно позиціонувати та орієнтувати геометричні примітиви у сцені. Основні види афінних перетворень:

* **Перенесення (Трансляція)**: Зсуває об'єкт на певний вектор (x0,y0,z0). У OpenGL реалізується командою glTranslatef(x0, y0, z0), яка змінює положення об'єкта у просторі без зміни його орієнтації чи масштабу.
* **Поворот**: Обертає об'єкт навколо певної осі на заданий кут. Використовується команда glRotatef(angle, x, y, z), де (x,y,z) визначає вісь обертання, а angle — кут повороту.
* **Масштабування**: Змінює розміри об'єкта за осями X, Y та Z. Реалізується командою glScalef(sx, sy, sz), де sx, sy, sz — коефіцієнти масштабування по відповідних осях.

### Впровадження Афінних Перетворень для Розміщення Квадратичних Примітивів

Для коректного розміщення квадратичних примітивів у тривимірній сцені необхідно застосувати відповідні афінні перетворення, щоб встановити їхні координати прив'язки (x0,y0,z0) та орієнтацію відповідно до заданої осі. Процес включає:

1. **Перенесення**: За допомогою glTranslatef(x0, y0, z0) переміщаємо об'єкт так, щоб його точка прив'язки відповідала заданим координатам.
2. **Поворот**: Якщо вісь об'єкта не збігається з однією з координатних осей, застосовується поворот для вирівнювання його осі з необхідною координатною віссю (наприклад, паралельність осі Z для сфери).
3. **Масштабування**: Налаштовує розміри об'єкта відповідно до заданих радіусів, висоти чи інших параметрів.

### Ізотропна Система Координат та Проекції

Ізотропна система координат забезпечує однакове масштабування за всіма осями, що дозволяє уникнути спотворень при зміні розмірів вікна перегляду. В OpenGL це досягається за допомогою налаштування області перегляду (glViewport) та матриці проекції (glOrtho для ортографічної та gluPerspective для перспективної проекцій).

* **Ортографічна Проекція**: Забезпечує паралельне зображення об'єктів без перспективного спотворення. Використовується для технічних малюнків та сцен, де важлива точність розмірів.
* **Перспективна Проекція**: Відображає об'єкти з перспективним спотворенням, де об'єкти, що знаходяться далі, виглядають меншими. Це створює більш реалістичне зображення сцени.

### Освітлення та Матеріали

Освітлення відіграє ключову роль у візуалізації тривимірних сцен, забезпечуючи глибину та реалістичність об'єктів. У OpenGL використовується модель освітлення, яка включає різні типи світла (напрямлене, точкове, розсіяне) та матеріальні властивості об'єктів (колір, відбивна здатність, блиск).

* **Матеріали**: Визначають, як об'єкти взаємодіють зі світлом. Встановлюються за допомогою команд glMaterialfv та glMaterialf, які задають параметри відбиття для амбієнтного, дифузного та спекулярного компонентів.
* **Джерела Світла**: Визначають положення та властивості освітлення сцени. Наприклад, GL\_LIGHT0 може бути налаштований на позицію (5.0,10.0,5.0) з амбієнтним, дифузним та спекулярним освітленням.

### Списки Відображення (Display Lists)

Списки відображення — це механізм оптимізації рендерингу у OpenGL, який дозволяє попередньо зберігати команди малювання об'єктів для повторного використання без повторного обчислення геометрії. Це підвищує продуктивність, особливо при роботі зі складними сценами або великою кількістю примітивів.

## Результати виконання практичної роботи

### Розв'язання завдання

1. **Коректне (ізотропне) відображення завдання (під час зміни розмірів вікна) у ортографічній проекції:**

**Файл**: RenderControl.cs

**Рядки коду**: 88-107

**Як реалізовано**: В методі RenderControl\_Render встановлюється ізотропна система координат за допомогою функції glViewport, яка визначає робочу область як квадрат, використовуючи мінімальне значення між шириною та висотою вікна (рядки 88-92). Потім, залежно від вибору проекції, застосовується ортографічна проекція через glOrtho (рядки 98-102).

1. **Під час запуску застосунку відображаються осі OX, OY, OZ, координатна сітка і каркас квадратичних об’єктів​:**

**Файл**: RenderControl.cs

**Рядки коду**: 126-140

**Як реалізовано**: У методі RenderControl\_Render після налаштування проекції викликаються методи DrawCoordinateAxes() та DrawGrid() для відображення координатних осей та сітки (рядки 127-130). Каркасні об'єкти малюються за допомогою списків відображення у методі DrawFigures().

1. **Інтерфейс керування параметрами площини відсічення:**

**Файл**: MainForm.cs

**Рядки коду**: 57-173

**Як реалізовано**: У файлі MainForm.cs реалізовано інтерфейс для керування параметрами площини відсічення через чекбокс та числові поля (numericUpDownA, numericUpDownB, numericUpDownC, numericUpDownD) (рядки 57-173). Обробники подій (checkBox\_Clip\_CheckedChanged, numericUpDownA\_ValueChanged тощо) дозволяють користувачу увімкнути/вимкнути площину відсічення та змінювати її параметри.

1. **Використання джерел світла для освітлення об’єктів сцени сумісно з командою glColorMaterial:**

**Файл**: RenderControl.cs

**Рядки коду**: 142-285

**Як реалізовано**: У методі RenderControl\_Render налаштовується освітлення сцени. Використовується джерело світла GL\_LIGHT0 з параметрами позиції, амбієнтного, дифузного та спекулярного освітлення (рядки 147-285). Хоча glColorMaterial вимкнено (glDisable(GL\_COLOR\_MATERIAL) на рядку 145), налаштування матеріалів через glMaterialfv забезпечує взаємодію з джерелами світла.

1. **Використання списків відображення (Display Lists):**

**Файл**: RenderControl.cs

**Рядки коду**: 162-214

**Як реалізовано**: Списки відображення створюються у методі InitializeDisplayLists (рядки 162-214), де генеруються ідентифікатори списків (glGenLists), визначаються стилі малювання (gluQuadricDrawStyle), та записуються команди малювання фігур (gluSphere, gluCylinder, gluPartialDisk). Під час рендерингу фігури відображаються за допомогою glCallList (рядки 313-320).

1. **Створення зображення сцени в перспективній проекції:**

**Файл**: RenderControl.cs

**Рядки коду**: 103-107, 113-117, 177-183 (в MainForm.cs)

**Як реалізовано**: У методі RenderControl\_Render встановлюється перспективна проекція за допомогою gluPerspective, якщо режим проекції не ортографічний (рядки 103-107). У MainForm.cs реалізовано перемикання між ортографічною та перспективною проекцією через радіокнопки (рядки 177-191).

1. **Накладення текстури на поверхню завданих у варіанті фігур:**

**Файл**: RenderControl.cs

**Рядки коду**: 23-25

**Як реалізовано**: У нас в коді є FillMode, який закрашує фігури повністю кольором.

1. **Застосування команди glMaterial для налаштування параметрів відбиття поверхонь об’єктів сцени:**

**Файл**: RenderControl.cs

**Рядки коду**: 301-371

**Як реалізовано**: У методі DrawFigures() для кожного примітиву встановлюються матеріальні властивості за допомогою glMaterialfv та glMaterialf (рядки 301-371). Це включає налаштування амбієнтного, дифузного та спекулярного компонентів, а також коефіцієнта блиску для сфери, конуса та диска.

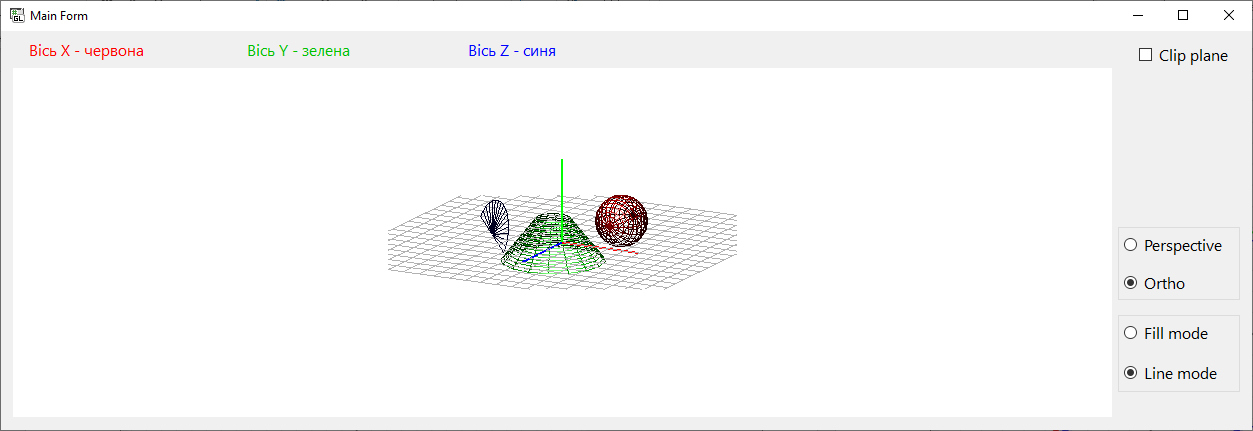


Рисунок 5.1 – Тестування програми при зміні ширини вікна

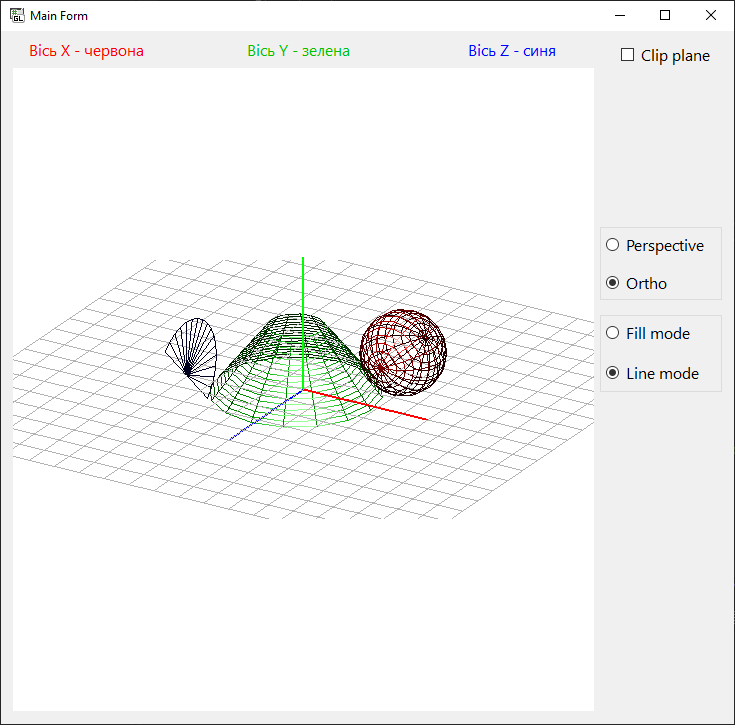


Рисунок 5.2 – Тестування програми при зміні висоти вікна

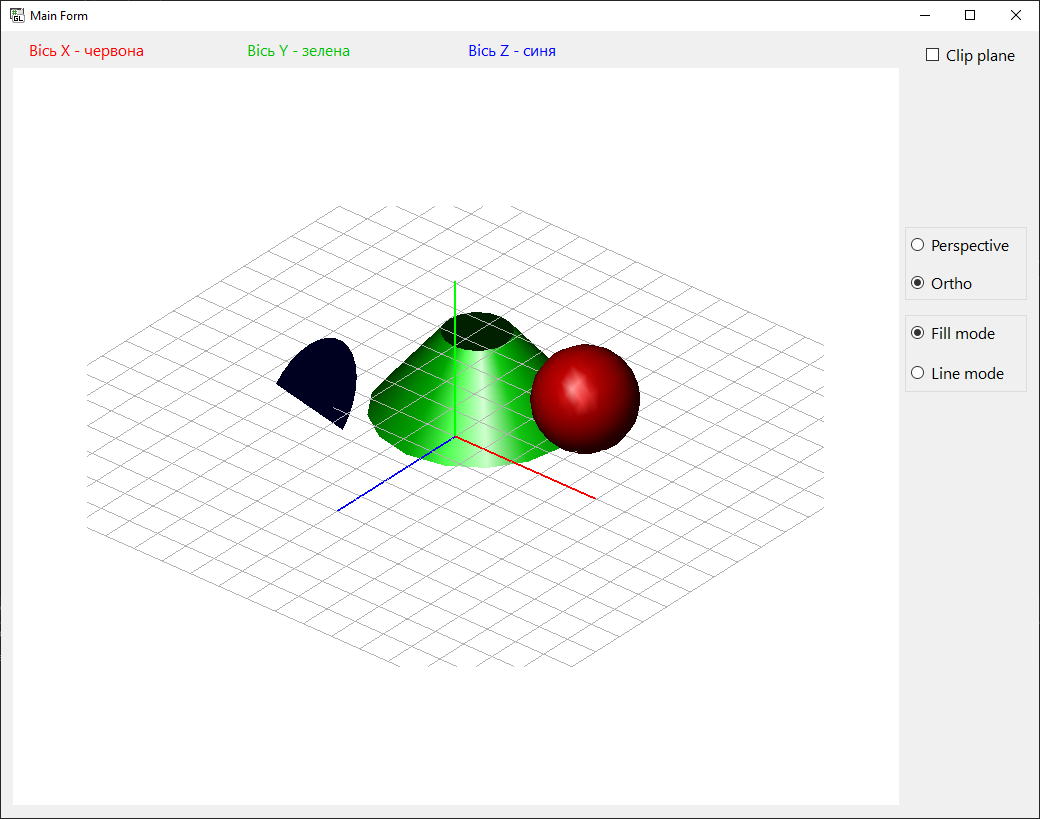


Рисунок 5.3 – Тестування програми з Fill mode

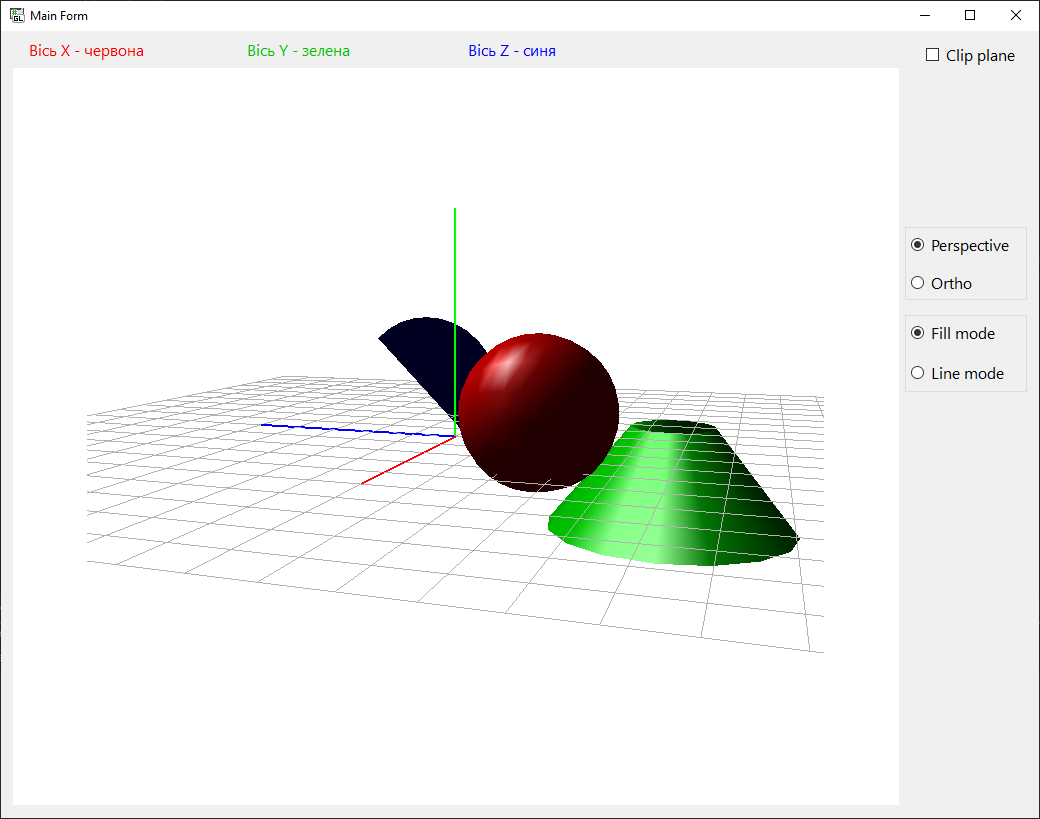


Рисунок 5.4 – Тестування програми з Perspective

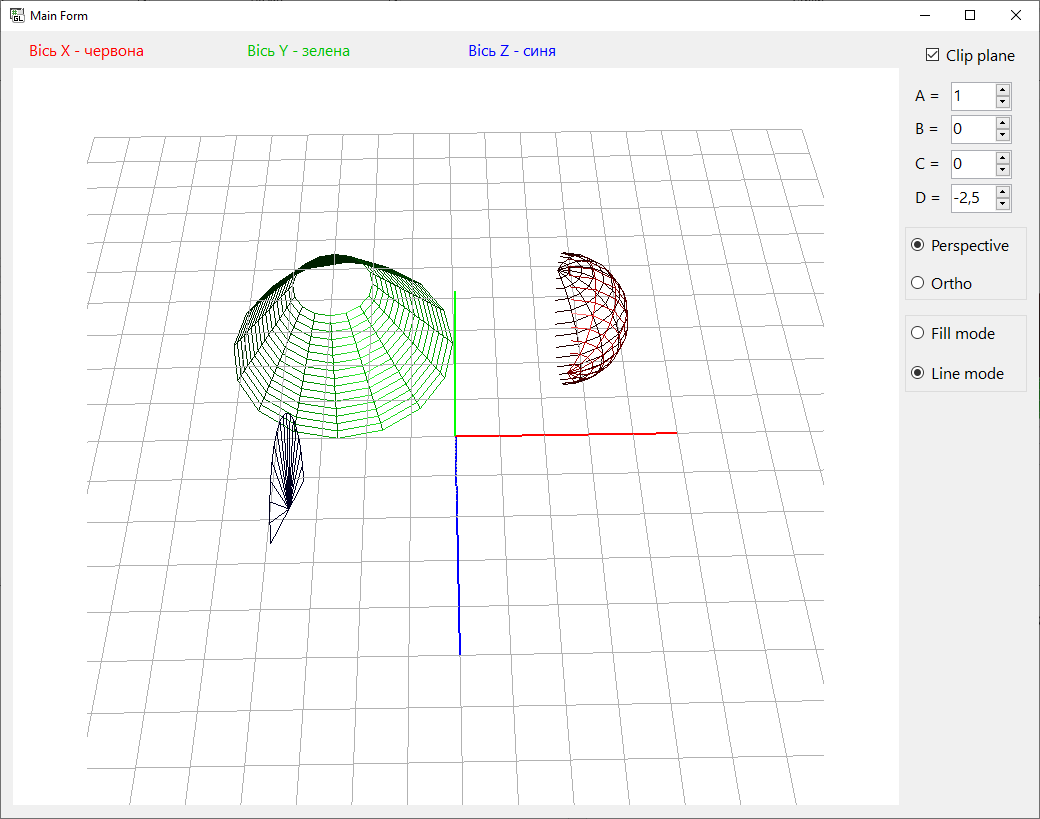


Рисунок 5.4 – Тестування програми з відсіканням сфери

### Контроль виконання вимог та елементів завдання

В результаті виконання практичної роботи були повністю виконані елементи базового рівня та повністю підвищеного рівня складності, що відображено в таблиці 5.1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблиця 5.1 | | | |  |
| №  п/п | Складність | Вимоги | Бали | Зроблено |
| 1 | Базовий рівень | Коректне (ізотропне) відображення завдання (під час зміни розмірів вікна) у ортографічній проекції | 1 | **+** |
| 2 | Під час запуску застосунку відображаються осі ***0X***, ***0Y***, ***0Z***, координатна сітка і каркас квадратичних об’єктів | 1 | **+** |
| 3 | Інтерфейс керування параметрами площини відтину | 1 | **+** |
| 4 | Використання джерел світла для освітлення об’єктів сцени сумісно з командою ***glColorMaterial*** | 1 | **+** |
| 5 | Використання списків відображення (***Display Lists***) | 1 | **+** |
| 6 | Підвищений рівень | Створення зображення сцени в перспективній проекції | 1 | + |
| 7 | Накладення текстури на поверхню завданих у варіанті фігур | 1 | + |
| 8 | Застосування команди ***glMaterial*** для налаштування параметрів відбиття поверхонь об’єктів сцени | 1 | + |

### Посилання на GitHub

<https://github.com/Mausipupsi/OpenGL>

# Практична робота 6. **ВІЗУАЛІЗАЦІЯ ПРЯМОЇ ЗАДАЧІ КІНЕМАТИКИ**

## Завдання, варіант № 14

|  |  |  |
| --- | --- | --- |
| Номер варіанта | Параметри | Кінематична модель |
| 14 | а = 0.46  b = 0.82  c = 0.6 | v14-00%20copy |

Використовуючи інструментальні засоби, запропоновані викладачем, створити додаток для виведення на екран моделі маніпулятора за даною кінематичною схемою (оцінка складності наведена в табл. 6.1). Для управління моделлю та точкою спостереження необхідно використовувати клавіатуру та/або маніпулятор «миша», за допомогою яких змінюють значення параметрів, які відповідають руху, наприклад кути ****** , ****** , ****** , відстань ***S*** (див. варіанти завдань у табл. 6.2).

## Теоретичні відомості

### Пряма задача кінематики

Пряма задача кінематики полягає у визначенні координат кінцевої точки маніпулятора (ефектора) у просторі на основі заданих параметрів системи, таких як довжини ланок і кути обертання. Це базова задача в робототехніці, яка використовується для управління маніпуляторами та їх моделювання.

Основою вирішення прямої задачі кінематики є геометричні залежності між ланками маніпулятора. Маніпулятор є послідовністю ланок, з'єднаних рухомими шарнірами, рух яких визначається кутами обертання ****** , ****** , ****** та, у деяких випадках, лінійними зміщеннями, такими як параметр ***S***.

**Геометрична основа**

Кожна ланка маніпулятора може бути описана як вектор, що характеризується довжиною та напрямком. Для розрахунку положення кожної точки маніпулятора використовуються:

* Тригонометричні функції (синуси та косинуси) для обчислення проєкцій ланок на осі координат.
* Теорема Піфагора для розрахунку відстаней.
* Геометричні співвідношення для врахування обмежень на довжини ланок.

### Кінематична модель

Для побудови моделі маніпулятора використовується така послідовність розрахунків:

1. Визначення положення базової точки O у фіксованій системі координат.
2. Обчислення координат точки B як функції лінійного зміщення S вздовж осі y.
3. Визначення точки C, яка з'єднана з точкою B шарніром. Розрахунки включають використання кола, радіус якого дорівнює довжині ланки.
4. Розрахунок кінцевої точки A маніпулятора, положення якої задається напрямком (кут ψ) і довжиною ланки.

**Перевірка коректності**

Для забезпечення коректності моделі виконуються перевірки:

* Фізичної досяжності заданого положення (відстані між точками не перевищують сумарної довжини ланок).
* Відсутності самоперетинів (перетину ліній між точками).

### Інструменти візуалізації

Для реалізації візуалізації використовується бібліотека OpenGL, яка забезпечує:

1. Відображення маніпулятора в режимі реального часу з використанням тривимірної графіки.
2. Можливість інтерактивного управління камерою (масштабування, обертання, переміщення).
3. Управління параметрами маніпулятора (кути ψ\psiψ, ϕ\phiϕ, лінійне зміщення SSS) за допомогою клавіатури або миші.

Особливості візуалізації:

* Ланки маніпулятора зображаються у вигляді тривимірних об'єктів (кубів або циліндрів).
* Присутня інтерактивність для зміни параметрів моделі.
* Реалізовані освітлення та відкидання тіней для покращення візуального сприйняття.

## Результати виконання практичної роботи

### Розв'язання завдання

1. **Реалізація програми двовимірної моделі маніпулятора відповідно до варіанта:**

**Файл**: RenderControl.cs

**Рядки коду**: 39-213

**Як реалізовано**: Клас RobotArm (рядки 39-213) реалізує двовимірну модель маніпулятора з параметрами ланок та методами для оновлення положення маніпулятора на основі заданих параметрів S та psi.

1. **Доопрацювання програми до тривимірної моделі (обертання точки спостереження, масштаб)​:**

**Файл**: RenderControl.cs

**Рядки коду**: 222-299

**Як реалізовано**: У класі RenderControl реалізовано параметри ax, ay для обертання сцени та m для масштабування (рядки 222-224). Методи Mouse\_Move (рядки 611-620) дозволяють обертати модель за допомогою миші, а масштабування здійснюється через змінні m.

1. **Управління моделлю та точкою спостереження маніпулятором «миша» та/або клавіатурою:**

**Файл**: RenderControl.cs

**Рядки коду**: 245-249, 598-641

**Як реалізовано**: Події миші (MouseDown, MouseUp, MouseMove) реалізовані на рядках 245-247 та 598-620. Подія клавіатури обробляється методом Key\_Down (рядки 623-641), що дозволяє змінювати параметри S та psi маніпулятора.

1. **Використання квадратичних примітивів для відображення кінематичної схеми:**

**Файл**: RenderControl.cs

**Рядки коду**: 230-231, 570-577

**Як реалізовано**: Використання квадратичних примітивів реалізовано за допомогою об'єкта quadric (рядки 230-231) та методів DrawSphere (рядки 570-577), gluSphere для малювання сферичних елементів кінематичної схеми.

1. **Використання освітлення та визначення матеріалів командою glColorMaterial(...):**

**Файл**: RenderControl.cs

**Рядки коду**: 322-340, 438-449

**Як реалізовано**: Налаштування освітлення здійснено у методі SetupLighting (рядки 322-340). Використано glColorMaterial для визначення матеріалів об'єктів, що дозволяє інтегрувати кольори об'єктів з освітленням сцени.

1. **Вміст звіту відповідає прикладу оформлення:**

Вміст звіту відповідає прикладу оформлення.

1. **Використання ООП (розробка власних класів):**

**Файл**: RenderControl.cs

**Рядки коду**: 10-36, 39-213, 215-642

**Як реалізовано**: У нас в коді є FillMode, який закрашує фігури повністю кольором.

1. **Використання текстур для елементів кінематичної схеми:**

**Файл**: RenderControl.cs

**Рядки коду**: 362-385

**Як реалізовано**: Малюємо сам маніпулятор та його елементи.

1. **Визначення матеріалів командою glMaterial(...), використання прозорості:**

**Файл:** RenderControl.cs

**Рядки:** 438-449

**Опис:** Метод SetMaterial (рядки 438-449) визначає матеріали об'єктів за допомогою glMaterialfv, включаючи параметри прозорості alpha.

1. **Використання перспективної проекції для відображення моделі маніпулятора:**

**Файл**: RenderControl.cs

**Рядки коду**: 283-294

**Як реалізовано**: Перспективна проекція налаштована у методі OnRender (рядки 283-294) за допомогою gluPerspective, що забезпечує тривимірне відображення моделі.

1. **Реалізація освітлення з тінню від моделі маніпулятора:**

**Файл**: RenderControl.cs

**Рядки коду**: 342-409

**Як реалізовано**: Освітлення налаштовано у методі SetupLighting (рядки 322-340). Метод DrawShadow (рядки 388-409) реалізує відображення тіней маніпулятора на підлозі за допомогою створення матриці тіні та її застосування до об'єктів сцени.

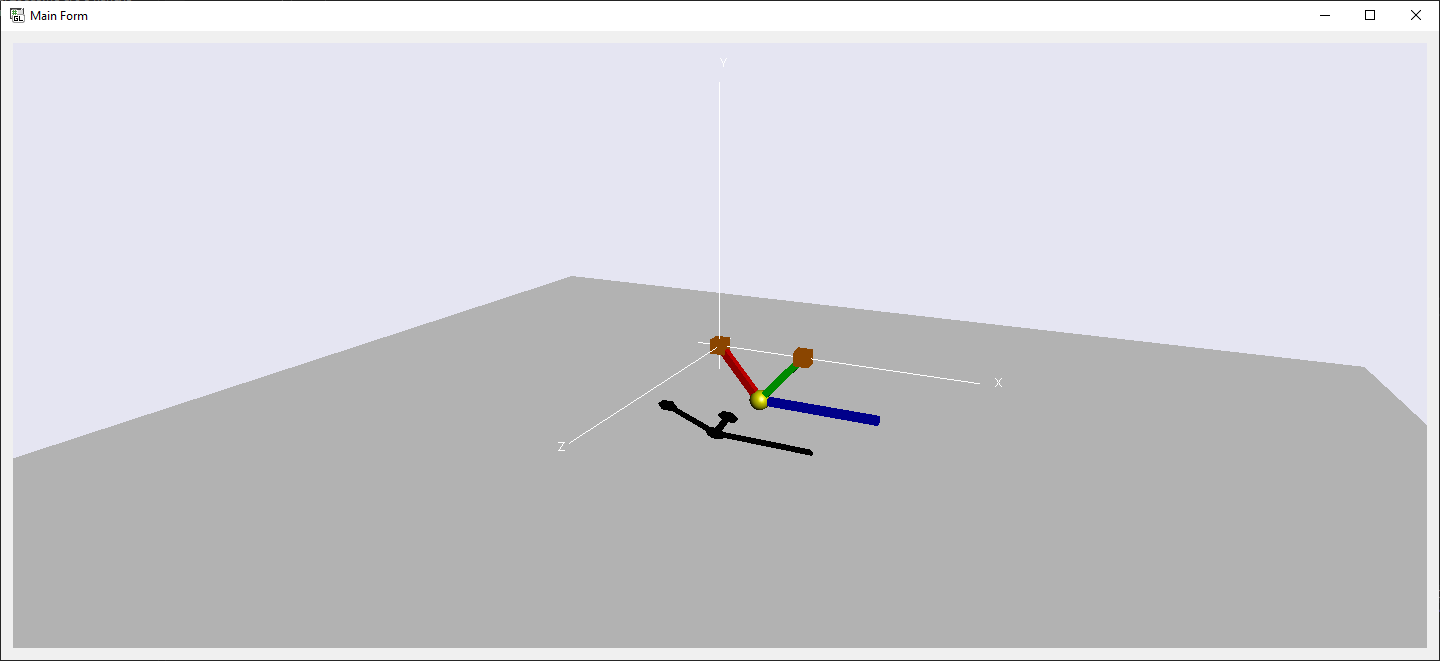


Рисунок 6.1 – Тестування програми при зміні ширини вікна

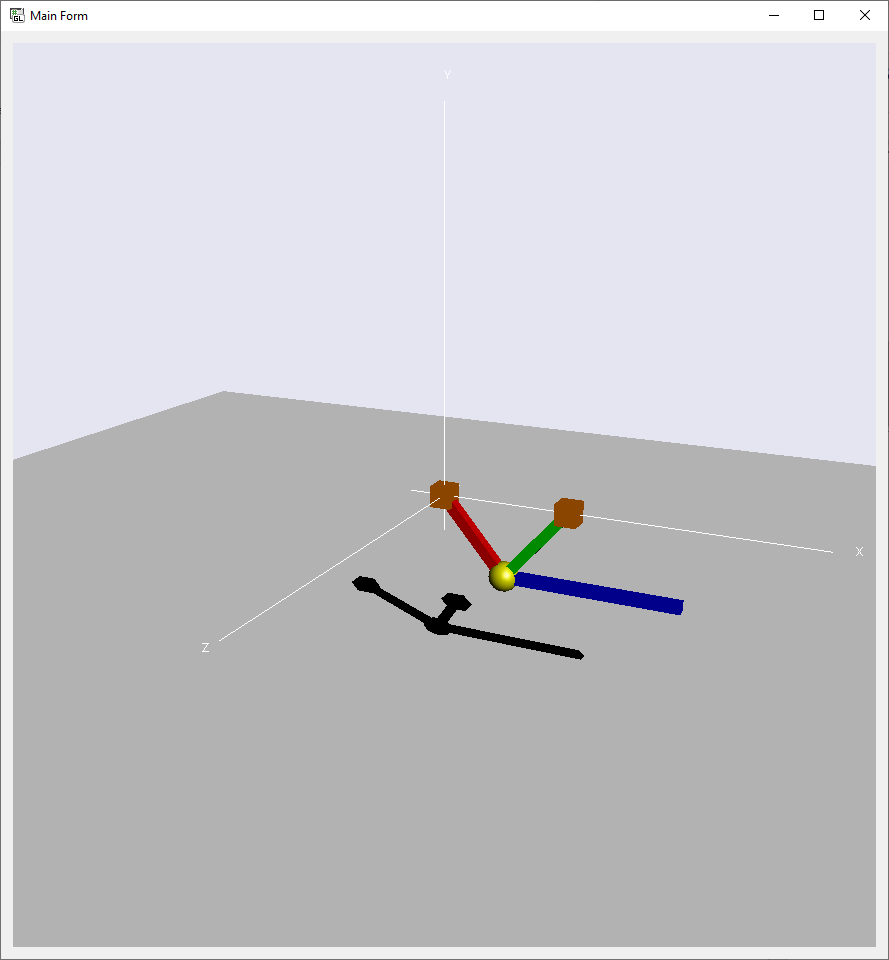


Рисунок 6.2 – Тестування програми при зміні висоти вікна

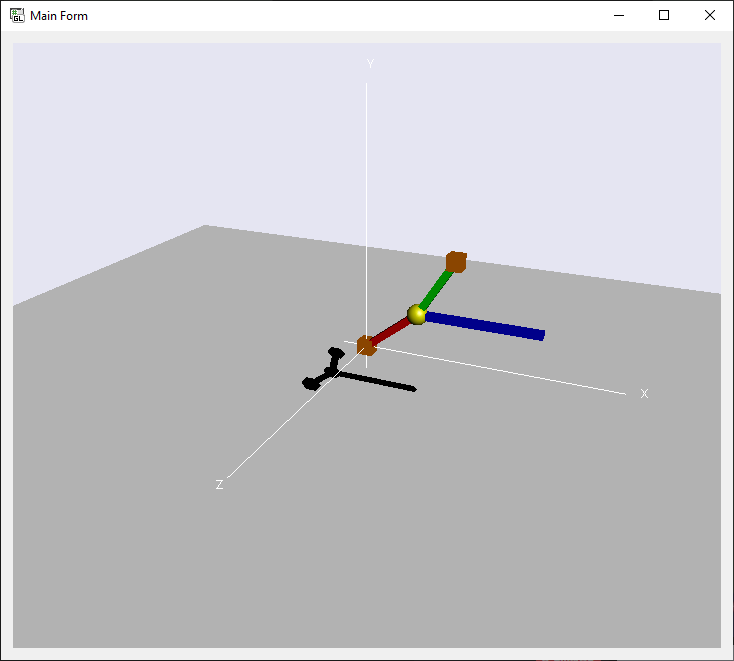


Рисунок 1.3 – Тестування програми з додаванням відстані S

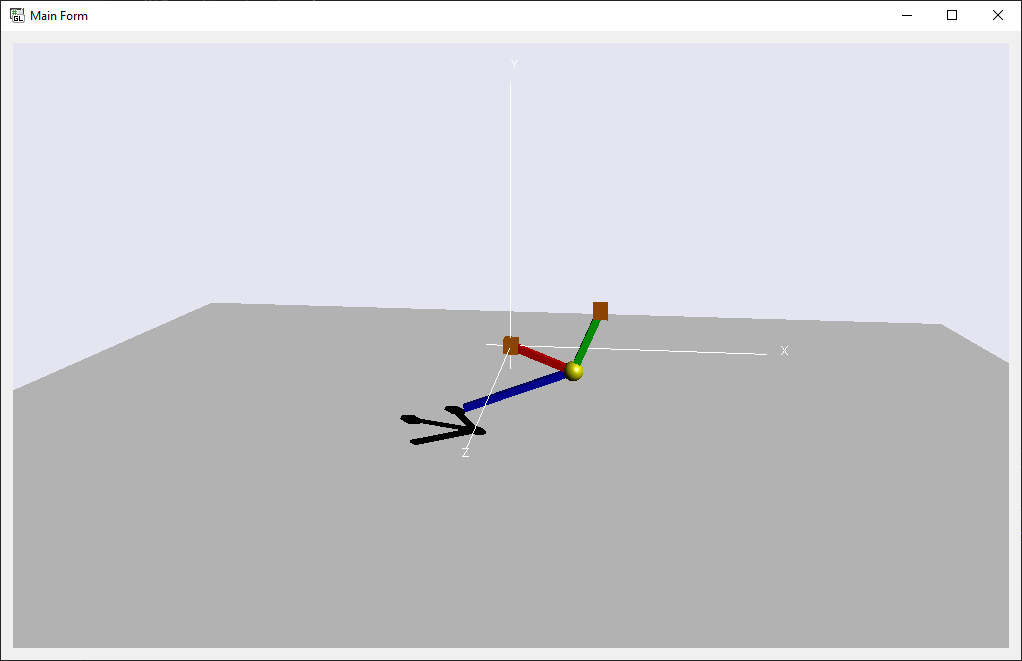


Рисунок 1.4 – Тестування програми з зміною кута Ψ

### Контроль виконання вимог та елементів завдання

В результаті виконання практичної роботи були повністю виконані елементи базового рівня та повністю підвищеного рівня складності, що відображено в таблиці 6.1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблиця 6.1 | | | |  |
| № п/п | Складність | Вимоги | Бали | Зроблено |
| 1 | Базовий рівень | Реалізація програми двовимірної моделі маніпулятора відповідно до варіанта | 5 | **+** |
| 2 | Доопрацювання програми до тривимірної моделі (обертання точки спостереження, масштаб) | 1 | **+** |
| 3 | Управління моделлю та точкою спостереження маніпулятором «миша» та/або клавіатурою | 1 | **+** |
| 4 | Використання квадратичних примітивів для відображення кінематичної схеми | 3 | **+** |
| 5 | Використання освітлення та визначення матеріалів командою *glColorMaterial (...)* | 4 | **+** |
| 6 | Вміст звіту відповідає прикладу оформлення | 6 | + |
| 7 | Підвищений рівень | Використання ООП (розробка власних класів) | 1 | + |
| 8 | Використання текстур для елементів кінематичної схеми | 2 | + |
| 9 | Визначення матеріалів командою *glMaterial* (...), використання прозорості | 2 | + |
| 10 | Використання перспективної проекції для відображення моделі маніпулятора | 1 | + |
| 11 | Реалізація освітлення з тінню від моделі маніпулятора | 4 | + |

### Посилання на GitHub

<https://github.com/Mausipupsi/OpenGL>

# Загальний перелік посилань

1. Microsoft. glDrawArrays function [Електронний ресурс] / Microsoft – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/windows/win32/opengl/gldrawarrays>.
2. Microsoft. glLineStipple function [Електронний ресурс] / Microsoft – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/windows/win32/opengl/gllinestipple>
3. Microsoft. glEnable function [Електронний ресурс] / Microsoft – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/windows/win32/opengl/glenable>
4. Microsoft. glDisable function [Електронний ресурс] / Microsoft – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/windows/win32/opengl/gldisable>
5. Microsoft. glColor3d function [Електронний ресурс] / Microsoft – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/windows/win32/opengl/glcolor3d>
6. Microsoft. glLineWidth function [Електронний ресурс] / Microsoft – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/windows/win32/opengl/gllinewidth>
7. Microsoft. glVertexPointer function [Електронний ресурс] / Microsoft – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/windows/win32/opengl/glvertexpointer>
8. Microsoft. glVertex2d function [Електронний ресурс] / Microsoft – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/windows/win32/opengl/glvertex2d>
9. Microsoft. glMatrixMode function [Електронний ресурс] / Microsoft – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/windows/win32/opengl/glmatrixmode>
10. Microsoft. glPolygonMode function [Електронний ресурс] / Microsoft – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/windows/win32/opengl/glpolygonmode>

# Додаток А. Лістинг програми до практичної роботи №1

### Код файлу (Render.cpp)

1. #include "pch.h"
2. #include "glWinApp.h"
3. #include <vector>
4. using namespace std;
5. extern LPCSTR s1, s2, s3;
6. int LoadWindowDefaultFont() {
7. GLuint id = glGenLists(256);
8. wglUseFontBitmaps(wglGetCurrentDC(), 0, 256, id);
9. return id;
10. }
11. void OutText(LPCSTR str, double x, double y, double z = 0) {
12. glRasterPos3d(x, y, z);
13. glListBase(idFont);
14. glCallLists(static\_cast<GLsizei>(strlen(str)), GL\_UNSIGNED\_BYTE, str);
15. }
16. void SetupAndDrawLines(const vector<double>& vertices, float lineWidth, const double\* color) {
17. glColor3dv(color); // Встановлення кольору ліній
18. glLineWidth(lineWidth); // Встановлення ширини ліній
19. glEnableClientState(GL\_VERTEX\_ARRAY); // Включення масиву вершин
20. glVertexPointer(2, GL\_DOUBLE, 0, vertices.data()); // Встановлення вказівника на масив вершин
21. glDrawArrays(GL\_LINES, 0, static\_cast<GLsizei>(vertices.size() / 2)); // Малювання ліній
22. glDisableClientState(GL\_VERTEX\_ARRAY); // Вимкнення масиву вершин
23. }
24. void DrawGrid(double xMin, double xMax, double yMin, double yMax, double step) {
25. glEnable(GL\_LINE\_STIPPLE); // Включення режиму пунктирних ліній
26. glLineStipple(1, 0x00FF); // Встановлення шаблону пунктирних ліній
27. vector<double> vertices; // Створення вектора для зберігання вершин сітки
28. for (double x = xMin; x <= xMax; x += step) {
29. vertices.push\_back(x); // Додавання координат вершин сітки по осі X
30. vertices.push\_back(yMin - 1); // Додавання координат вершин сітки по осі Y
31. vertices.push\_back(x); // Додавання координат вершин сітки по осі X
32. vertices.push\_back(yMax + 1); // Додавання координат вершин сітки по осі Y
33. }
34. for (double y = yMin; y <= yMax; y += step) {
35. vertices.push\_back(xMin - 0.9); // Додавання координат вершин сітки по осі X
36. vertices.push\_back(y); // Додавання координат вершин сітки по осі Y
37. vertices.push\_back(xMax + 0.9); // Додавання координат вершин сітки по осі X
38. vertices.push\_back(y); // Додавання координат вершин сітки по осі Y
39. }
40. double gridColor[] = { 0.7, 0.7, 0.7 }; // Колір сітки
41. SetupAndDrawLines(vertices, 1, gridColor); // Налаштування та малювання ліній сітки
42. glDisable(GL\_LINE\_STIPPLE); // Вимкнення режиму пунктирних ліній
43. }
44. void DrawAxes(double xMin, double xMax, double yMin, double yMax) {
45. vector<double> vertices = {
46. xMin - 0.9, 0, xMax + 0.9, 0, // Координати вершин осі X
47. 0, yMin - 1, 0, yMax + 1 // Координати вершин осі Y
48. };
49. double axesColor[] = { 1, 1, 0 }; // Колір осей
50. SetupAndDrawLines(vertices, 4, axesColor); // Налаштування та малювання осей
51. vector<double> customLines = {
52. xMin - 0.9, yMin - 0.3, xMin - 0.9, yMax + 1, // Координати додаткових ліній
53. xMin - 0.3, yMin - 1, xMax + 0.9, yMin - 1 // Координати додаткових ліній
54. };
55. double customLinesColor[] = { 1, 1, 1 }; // Колір додаткових ліній
56. SetupAndDrawLines(customLines, 5, customLinesColor); // Налаштування та малювання додаткових ліній
57. glLineWidth(3); // Встановлення ширини ліній
58. glBegin(GL\_LINES); // Початок малювання ліній
59. glVertex2d(xMin - 0.9, yMax + 1.05); // Координати вершин стрілок осі Y
60. glVertex2d(xMin - 1.1, yMax + 0.6); // Координати вершин стрілок осі Y
61. glVertex2d(xMin - 0.9, yMax + 1.05); // Координати вершин стрілок осі Y
62. glVertex2d(xMin - 0.7, yMax + 0.6); // Координати вершин стрілок осі Y
63. glVertex2d(xMax + 0.95, yMin - 1); // Координати вершин стрілок осі X
64. glVertex2d(xMax + 0.6, yMin - 1.2); // Координати вершин стрілок осі X
65. glVertex2d(xMax + 0.95, yMin - 1); // Координати вершин стрілок осі X
66. glVertex2d(xMax + 0.6, yMin - 0.8); // Координати вершин стрілок осі X
67. glEnd(); // Кінець малювання ліній
68. glLineWidth(3); // Встановлення ширини ліній
69. glBegin(GL\_LINES); // Початок малювання ліній
70. for (double y = yMin; y <= yMax; y++) {
71. glVertex2d(xMin - 0.9, y); // Координати вершин поділок осі Y
72. glVertex2d(xMin - 1.12, y); // Координати вершин поділок осі Y
73. }
74. for (double x = xMin; x <= xMax; x++) {
75. glVertex2d(x, yMin - 1.22); // Координати вершин поділок осі X
76. glVertex2d(x, yMin - 1); // Координати вершин поділок осі X
77. }
78. glEnd(); // Кінець малювання ліній
79. OutText("Y1", xMin - 1.5, yMin - 0.1); // Відображення тексту "Y1" на осі Y
80. OutText("Y2", xMin - 1.5, yMax - 0.1); // Відображення тексту "Y2" на осі Y
81. OutText("X1", xMin - 0.1, yMin - 1.5); // Відображення тексту "X1" на осі X
82. OutText("X2", xMax - 0.1, yMin - 1.5); // Відображення тексту "X2" на осі X
83. }
84. class Point {
85. public:
86. double x, y; // Координати точки
87. Point(double x = 0, double y = 0) : x(x), y(y) {} // Конструктор з параметрами за замовчуванням
88. void draw() const {
89. glColor3d(1, 0, 0); // Встановлення червоного кольору для точки
90. glPointSize(15); // Встановлення розміру точки
91. glBegin(GL\_POINTS); // Початок малювання точки
92. glVertex2d(x, y); // Встановлення координат точки
93. glEnd(); // Кінець малювання точки
94. }
95. };
96. class MyPolygon {
97. public:
98. vector<Point> points; // Вектор точок, що складають полігон
99. MyPolygon(const vector<Point>& points) : points(points) {} // Конструктор з ініціалізацією вектора точок
100. void draw() const {
101. glColor3d(0, 0, 1); // Встановлення синього кольору для полігону
102. glLineWidth(7); // Встановлення ширини ліній полігону
103. glBegin(GL\_LINE\_STRIP); // Початок малювання полігону
104. for (const auto& point : points) {
105. glVertex2d(point.x, point.y); // Встановлення координат точок полігону
106. }
107. glEnd(); // Кінець малювання полігону
108. }
109. };
110. void Render(RECT& clientRect) {
111. glClearColor(0.2f, 0.2f, 0.2f, 1.0f); // Встановлення кольору фону
112. glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT | GL\_STENCIL\_BUFFER\_BIT); // Очищення буферів кольору, глибини та трафарету
113. glLoadIdentity(); // Скидання поточної матриці
114. int Height = clientRect.bottom - clientRect.top; // Визначення висоти вікна
115. int Width = clientRect.right - clientRect.left; // Визначення ширини вікна
116. double xMin = -2, xMax = 7, yMin = -4, yMax = 0; // Встановлення меж координатної системи
117. glViewport(0, 0, Width, Height); // Встановлення області перегляду
118. gluOrtho2D(xMin - 2.0, xMax + 2.0, yMin - 2.0, yMax + 2.0); // Встановлення ортогональної проекції
119. DrawGrid(xMin, xMax, yMin, yMax, 1.0); // Малювання сітки
120. DrawAxes(xMin, xMax, yMin, yMax); // Малювання осей
121. vector<Point> polygonPoints = {
122. Point(-1, -4), Point(-2, -2), Point(-2, -1), Point(-1, 0), Point(1, -2), Point(1, -3), Point(0, -4), Point(-1, -4)
123. }; // Визначення точок полігону
124. vector<Point> points = {
125. Point(4, -4), Point(3, -2), Point(3, -1), Point(4, 0), Point(6, -2), Point(6, -3), Point(5, -4), Point(4, -4)
126. }; // Визначення окремих точок
127. MyPolygon polygon(polygonPoints); // Створення об'єкта полігону
128. polygon.draw(); // Малювання полігону
129. for (const auto& point : points) {
130. point.draw(); // Малювання окремих точок
131. }
132. }

# Додаток Б. Лістинг програми до практичної роботи №2

### Код файлу (MainForm.cs)

1. using System;
2. using System.Windows.Forms;
3. using static Minakov\_Lab2.OpenGL;
4. namespace Minakov\_Lab2
5. {
6. public partial class MainForm : Form
7. {
8. // Конструктор головної форми
9. public MainForm()
10. {
11. InitializeComponent();
12. // Встановлення початкових значень для числових контролів
13. numericUpDownHorizontal.Value = 1;
14. numericUpDownVertical.Value = 1;
15. // Встановлення радіокнопки "Fill" за замовчуванням
16. radioButtonFill.Checked = true;
17. // Підписка на подію зміни режиму рендерингу
18. renderControl.RenderModeChanged += RenderControl\_RenderModeChanged;
19. }
20. // Обробник зміни значення горизонтальної кількості плиток
21. private void NumericUpDownHorizontal\_ValueChanged(object sender, EventArgs e)
22. {
23. UpdateTileCount();
24. }
25. // Обробник зміни значення вертикальної кількості плиток
26. private void NumericUpDownVertical\_ValueChanged(object sender, EventArgs e)
27. {
28. UpdateTileCount();
29. }
30. // Метод оновлення кількості плиток на основі введених значень
31. private void UpdateTileCount()
32. {
33. int horizontal = (int)numericUpDownHorizontal.Value;
34. int vertical = (int)numericUpDownVertical.Value;
35. renderControl.UpdateTileCount(horizontal, vertical);
36. }
37. // Обробник зміни стану радіокнопки "Fill"
38. private void RadioButtonFill\_CheckedChanged(object sender, EventArgs e)
39. {
40. if (radioButtonFill.Checked)
41. {
42. renderControl.SetRenderMode(RenderControl.RenderMode.Fill);
43. }
44. }
45. // Обробник зміни стану радіокнопки "Line"
46. private void RadioButtonLine\_CheckedChanged(object sender, EventArgs e)
47. {
48. if (radioButtonLine.Checked)
49. {
50. renderControl.SetRenderMode(RenderControl.RenderMode.Line);
51. }
52. }
53. // Обробник зміни стану радіокнопки "Point"
54. private void RadioButtonPoint\_CheckedChanged(object sender, EventArgs e)
55. {
56. if (radioButtonPoint.Checked)
57. {
58. renderControl.SetRenderMode(RenderControl.RenderMode.Point);
59. }
60. }
61. // Обробник події зміни режиму рендерингу в контролі рендерингу
62. private void RenderControl\_RenderModeChanged(object sender, RenderControl.RenderModeChangedEventArgs e)
63. {
64. switch (e.NewRenderMode)
65. {
66. case RenderControl.RenderMode.Fill:
67. radioButtonFill.Checked = true;
68. break;
69. case RenderControl.RenderMode.Line:
70. radioButtonLine.Checked = true;
71. break;
72. case RenderControl.RenderMode.Point:
73. radioButtonPoint.Checked = true;
74. break;
75. }
76. }
77. }
78. }

### Код файлу (RenderControl.cs)

1. using System;
2. using System.Collections.Generic;
3. using System.Windows.Forms;
4. using static Minakov\_Lab2.OpenGL;
5. using static Minakov\_Lab2.RenderControl;
6. namespace Minakov\_Lab2
7. {
8. public partial class RenderControl : OpenGL
9. {
10. // Кількість плиток по горизонталі та вертикалі
11. private int horizontalCount = 1;
12. private int verticalCount = 1;
13. // Поточний режим рендерингу
14. private RenderMode currentMode = RenderMode.Fill;
15. // Перелік можливих режимів рендерингу
16. public enum RenderMode
17. {
18. Fill,
19. Line,
20. Point
21. }
22. // Подія зміни режиму рендерингу
23. public event EventHandler<RenderModeChangedEventArgs> RenderModeChanged;
24. // Клас аргументів для події зміни режиму рендерингу
25. public class RenderModeChangedEventArgs : EventArgs
26. {
27. public RenderMode NewRenderMode { get; private set; }
28. public RenderModeChangedEventArgs(RenderMode newRenderMode)
29. {
30. NewRenderMode = newRenderMode;
31. }
32. }
33. // Список кнопок для вибору режиму рендерингу
34. private readonly List<Button> buttons = new();
35. // Конструктор контролу рендерингу
36. public RenderControl()
37. {
38. InitializeComponent();
39. InitializeButtons();
40. // Підписка на події зміни розміру, натискання миші та малювання
41. this.Resize += new EventHandler(RenderControl\_Resize);
42. this.MouseDown += new MouseEventHandler(RenderControl\_MouseDown);
43. this.Paint += new PaintEventHandler(RenderControl\_Paint);
44. this.DoubleBuffered = false;
45. SetStyle(ControlStyles.Opaque, true);
46. }
47. // Обробник події зміни розміру контролу
48. private void RenderControl\_Resize(object sender, EventArgs e)
49. {
50. InitializeButtons();
51. Invalidate();
52. }
53. // Обробник події натискання миші
54. private void RenderControl\_MouseDown(object sender, MouseEventArgs e)
55. {
56. float mouseX = e.X;
57. float mouseY = Height - e.Y;
58. // Перевірка, чи натиснута кнопка
59. foreach (var button in buttons)
60. {
61. if (button.IsPointInside(mouseX, mouseY))
62. {
63. if (currentMode != button.Mode)
64. {
65. currentMode = button.Mode;
66. Invalidate();
67. OnRenderModeChanged(new RenderModeChangedEventArgs(currentMode));
68. }
69. break;
70. }
71. }
72. }
73. // Метод виклику події зміни режиму рендерингу
74. protected virtual void OnRenderModeChanged(RenderModeChangedEventArgs e)
75. {
76. RenderModeChanged?.Invoke(this, e);
77. }
78. // Обробник події малювання контролу
79. private void RenderControl\_Paint(object sender, PaintEventArgs e)
80. {
81. RenderControl\_Render(sender, e);
82. }
83. // Ініціалізація кнопок вибору режиму рендерингу
84. private void InitializeButtons()
85. {
86. buttons.Clear();
87. float buttonWidth = 80f;
88. float buttonHeight = 30f;
89. float spacing = 10f;
90. float totalWidth = 3 \* buttonWidth + 2 \* spacing;
91. float startX = (Width - totalWidth) / 2f;
92. float startY = Height - buttonHeight - 10f;
93. // Створення трьох кнопок: Fill, Line, Point
94. buttons.Add(new Button(startX, startY, startX + buttonWidth, startY + buttonHeight, "Fill", RenderMode.Fill));
95. buttons.Add(new Button(startX + buttonWidth + spacing, startY, startX + 2 \* buttonWidth + spacing, startY + buttonHeight, "Line", RenderMode.Line));
96. buttons.Add(new Button(startX + 2 \* (buttonWidth + spacing), startY, startX + 3 \* buttonWidth + 2 \* spacing, startY + buttonHeight, "Point", RenderMode.Point));
97. }
98. // Метод рендерингу контролу
99. private void RenderControl\_Render(object sender, EventArgs e)
100. {
101. glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT); // Очищення буферів
102. SetupViewport();
103. DrawButtons(); // Малювання кнопок
104. DrawFigures(); // Малювання фігур
105. glFlush();
106. }
107. // Налаштування в'юпорту та проекції
108. private void SetupViewport()
109. {
110. glViewport(0, 0, Width, Height);
111. glMatrixMode(GL\_PROJECTION);
112. glLoadIdentity();
113. glOrtho(0, Width, 0, Height, -1, 1);
114. glMatrixMode(GL\_MODELVIEW);
115. glLoadIdentity();
116. }
117. // Метод малювання кнопок
118. private void DrawButtons()
119. {
120. glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_FILL);
121. glLineWidth(1f);
122. glPointSize(1f);
123. glDisable(GL\_LINE\_STIPPLE);
124. glDisable(GL\_POINT\_SMOOTH);
125. // Малювання кожної кнопки
126. foreach (var button in buttons)
127. {
128. bool isSelected = button.Mode == currentMode;
129. button.Draw(isSelected);
130. }
131. }
132. // Метод малювання фігур залежно від режиму
133. private void DrawFigures()
134. {
135. glMatrixMode(GL\_PROJECTION);
136. glLoadIdentity();
137. float aspectRatio = (float)Width / Height;
138. float a = 1000;
139. float figureWidth = a \* 2f;
140. float figureHeight = (float)(2f \* Math.Sqrt(a \* a - a \* a / 4f));
141. float horizontalSpacing = a + a / 2f;
142. float totalWidth = figureWidth + (horizontalCount - 1) \* horizontalSpacing;
143. float totalHeight = figureHeight \* verticalCount;
144. float maxDimension = Math.Max(totalWidth, totalHeight);
145. float halfSize = maxDimension \* 0.75f;
146. // Налаштування ортотичної проекції з урахуванням співвідношення сторін
147. if (aspectRatio >= 1.0f)
148. {
149. glOrtho(-halfSize \* aspectRatio, halfSize \* aspectRatio, -halfSize, halfSize, -1, 1);
150. }
151. else
152. {
153. glOrtho(-halfSize, halfSize, -halfSize / aspectRatio, halfSize / aspectRatio, -1, 1);
154. }
155. glMatrixMode(GL\_MODELVIEW);
156. glLoadIdentity();
157. float startX = -totalWidth / 2f;
158. float startY = totalHeight / 2f - figureHeight / 2f;
159. // Малювання сітки фігур
160. for (int row = 0; row < verticalCount; row++)
161. {
162. float xPos = startX;
163. float yPos = startY - row \* figureHeight;
164. for (int col = 0; col < horizontalCount; col++)
165. {
166. var figure = new ComplexFigure(xPos, yPos, a, figureHeight);
167. figure.Draw(currentMode);
168. // Зміщення позиції
169. if (col % 2 == 0)
170. {
171. xPos += horizontalSpacing;
172. yPos -= figureHeight / 2f;
173. }
174. else
175. {
176. xPos += horizontalSpacing;
177. yPos += figureHeight / 2f;
178. }
179. }
180. }
181. }
182. // Метод оновлення кількості плиток та перемалювання
183. public void UpdateTileCount(int horizontal, int vertical)
184. {
185. horizontalCount = horizontal;
186. verticalCount = vertical;
187. Invalidate();
188. }
189. // Метод встановлення нового режиму рендерингу
190. public void SetRenderMode(RenderMode mode)
191. {
192. if (currentMode != mode)
193. {
194. currentMode = mode;
195. Invalidate();
196. OnRenderModeChanged(new RenderModeChangedEventArgs(currentMode));
197. }
198. }
199. }
200. // Клас, що представляє кнопку вибору режиму рендерингу
201. public class Button
202. {
203. public float X1 { get; set; }
204. public float Y1 { get; set; }
205. public float X2 { get; set; }
206. public float Y2 { get; set; }
207. public string Text { get; set; }
208. public RenderControl.RenderMode Mode { get; set; }
209. // Конструктор кнопки
210. public Button(float x1, float y1, float x2, float y2, string text, RenderControl.RenderMode mode)
211. {
212. X1 = x1;
213. Y1 = y1;
214. X2 = x2;
215. Y2 = y2;
216. Text = text;
217. Mode = mode;
218. }
219. // Метод малювання кнопки
220. public void Draw(bool isSelected)
221. {
222. // Встановлення кольору фону кнопки
223. if (isSelected)
224. {
225. glColor3f(0.4f, 0.4f, 0.4f); // Темніший фон для вибраної кнопки
226. }
227. else
228. {
229. glColor3f(0.9f, 0.9f, 0.9f); // Світліший фон для невибраних кнопок
230. }
231. // Малювання прямокутника кнопки
232. glBegin(GL\_QUADS);
233. glVertex2f(X1, Y1);
234. glVertex2f(X2, Y1);
235. glVertex2f(X2, Y2);
236. glVertex2f(X1, Y2);
237. glEnd();
238. // Встановлення кольору обводки в залежності від режиму
239. switch (Mode)
240. {
241. case RenderMode.Fill:
242. glColor3f(1f, 0f, 0f); // Червоний для Fill
243. break;
244. case RenderMode.Line:
245. glColor3f(0f, 1f, 0f); // Зелений для Line
246. break;
247. case RenderMode.Point:
248. glColor3f(0f, 0f, 1f); // Синій для Point
249. break;
250. default:
251. glColor3f(0f, 0f, 0f); // Чорний за замовчуванням
252. break;
253. }
254. // Малювання обводки кнопки
255. glLineWidth(3f);
256. glBegin(GL\_LINE\_LOOP);
257. glVertex2f(X1, Y1);
258. glVertex2f(X2, Y1);
259. glVertex2f(X2, Y2);
260. glVertex2f(X1, Y2);
261. glEnd();
262. }
263. // Метод перевірки, чи знаходиться точка всередині кнопки
264. public bool IsPointInside(float x, float y)
265. {
266. return x >= X1 && x <= X2 && y >= Y1 && y <= Y2;
267. }
268. }
269. // Клас, що представляє фігуру для малювання
270. public class ComplexFigure
271. {
272. public float X { get; set; }
273. public float Y { get; set; }
274. public float A { get; set; }
275. public float FigureHeight { get; set; }
276. // Конструктор фігури
277. public ComplexFigure(float x, float y, float a, float figureHeight)
278. {
279. X = x;
280. Y = y;
281. A = a;
282. FigureHeight = figureHeight / 2f;
283. }
284. // Метод малювання фігури залежно від режиму
285. public void Draw(RenderControl.RenderMode mode)
286. {
287. SetRenderMode(mode);
288. // Встановлення кольору для ліній або точок
289. if (mode == RenderControl.RenderMode.Line || mode == RenderControl.RenderMode.Point)
290. {
291. glColor3f(0f, 0f, 0f); // Чорний колір
292. }
293. // Малювання трикутників
294. glBegin(GL\_TRIANGLES);
295. if (mode == RenderControl.RenderMode.Fill)
296. {
297. DrawTriangle(X + A + A / 2f, Y + FigureHeight, X + A + A, Y, X + A, Y, 1f, 1f, 0f);
298. DrawTriangle(X + A + A / 2f, Y - FigureHeight, X + A + A, Y, X + A, Y, 1f, 0f, 0f);
299. }
300. else
301. {
302. DrawTriangle(X + A + A / 2f, Y + FigureHeight, X + A + A, Y, X + A, Y, 0f, 0f, 0f);
303. DrawTriangle(X + A + A / 2f, Y - FigureHeight, X + A + A, Y, X + A, Y, 0f, 0f, 0f);
304. }
305. glEnd();
306. // Малювання квадрейтрипів
307. glBegin(GL\_QUAD\_STRIP);
308. if (mode == RenderControl.RenderMode.Fill)
309. {
310. DrawQuadStrip(X, Y, X + A / 2f, Y + FigureHeight, X + A, Y, X + A + A / 2f, Y + FigureHeight, 0f, 1f, 0f);
311. DrawQuadStrip(X, Y, X + A, Y, X + A / 2f, Y - FigureHeight, X + A + A / 2f, Y - FigureHeight, 0f, 0f, 1f);
312. }
313. else
314. {
315. DrawQuadStrip(X, Y, X + A / 2f, Y + FigureHeight, X + A, Y, X + A + A / 2f, Y + FigureHeight, 0f, 0f, 0f);
316. DrawQuadStrip(X, Y, X + A, Y, X + A / 2f, Y - FigureHeight, X + A + A / 2f, Y - FigureHeight, 0f, 0f, 0f);
317. }
318. glEnd();
319. }
320. // Допоміжний метод малювання трикутника з кольором
321. private static void DrawTriangle(float x1, float y1, float x2, float y2, float x3, float y3, float r, float g, float b)
322. {
323. glColor3f(r, g, b);
324. glVertex2f(x1, y1);
325. glVertex2f(x2, y2);
326. glVertex2f(x3, y3);
327. }
328. // Допоміжний метод малювання квадрейтрипа з кольором
329. private static void DrawQuadStrip(float x1, float y1, float x2, float y2, float x3, float y3, float x4, float y4, float r, float g, float b)
330. {
331. glColor3f(r, g, b);
332. glVertex2f(x1, y1);
333. glVertex2f(x2, y2);
334. glVertex2f(x3, y3);
335. glVertex2f(x4, y4);
336. }
337. // Метод встановлення режиму рендерингу
338. private static void SetRenderMode(RenderControl.RenderMode mode)
339. {
340. switch (mode)
341. {
342. case RenderControl.RenderMode.Fill:
343. glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_FILL);
344. glLineWidth(1f);
345. glPointSize(1f);
346. break;
347. case RenderControl.RenderMode.Line:
348. glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE);
349. glLineWidth(5f);
350. glPointSize(1f);
351. break;
352. case RenderControl.RenderMode.Point:
353. glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_POINT);
354. glLineWidth(1f);
355. glPointSize(10f);
356. break;
357. }
358. }
359. }
360. }

# Додаток В. Лістинг програми до практичної роботи № 3

### Код файлу (FunctionBase.cs)

1. using System;
2. using System.Collections.Generic;
3. namespace Minakov\_Task03
4. {
5. public abstract class FunctionBase
6. {
7. // Абстрактний метод для обчислення значення функції в точці x
8. public abstract double Evaluate(double x);
9. // Віртуальний метод для отримання точок розриву функції
10. public virtual List<double> GetDiscontinuities(double Xmin, double Xmax)
11. {
12. // За замовчуванням повертаємо порожній список
13. return new List<double>();
14. }
15. }
16. }

### Код файлу (Function1.cs)

1. using System;
2. namespace Minakov\_Task03
3. {
4. public class Function1 : FunctionBase
5. {
6. // Перевизначаємо метод Evaluate для обчислення f1(x)
7. public override double Evaluate(double x)
8. {
9. return Math.Cos(x) / Math.Sqrt(Math.Cos(6 \* x) + 1.01);
10. }
11. }
12. }

### Код файлу (Function2.cs)

1. using System;
2. using System.Collections.Generic;
3. namespace Minakov\_Task03
4. {
5. public class Function2 : FunctionBase
6. {
7. // Перевизначаємо метод Evaluate для обчислення f2(x)
8. public override double Evaluate(double x)
9. {
10. return Math.Log(1 + Math.Cos(x)) \* Math.Log10(2 + Math.Sin(5 \* x));
11. }
12. // Перевизначаємо метод GetDiscontinuities для визначення точок розриву
13. public override List<double> GetDiscontinuities(double Xmin, double Xmax)
14. {
15. List<double> discontinuities = new();
16. int nStart = (int)Math.Ceiling((Xmin - Math.PI) / (2 \* Math.PI));
17. int nEnd = (int)Math.Floor((Xmax - Math.PI) / (2 \* Math.PI));
18. for (int n = nStart; n <= nEnd; n++)
19. {
20. double xDisc = Math.PI + 2 \* Math.PI \* n;
21. if (xDisc >= Xmin && xDisc <= Xmax)
22. {
23. discontinuities.Add(xDisc);
24. }
25. }
26. return discontinuities;
27. }
28. }
29. }

### Код файлу (MainForm.cs)

1. using System;
2. using System.Windows.Forms;
3. namespace Minakov\_Task03
4. {
5. public partial class MainForm : Form
6. {
7. // Поточний об'єкт функції
8. private FunctionBase currentFunction;
9. // Попередні значення для Xmin, Xmax, Ymin, Ymax
10. private decimal prevXmin;
11. private decimal prevXmax;
12. private decimal prevYmin;
13. private decimal prevYmax;
14. // Автоматичні Ymin та Ymax
15. private double autoYmin;
16. private double autoYmax;
17. public MainForm()
18. {
19. InitializeComponent();
20. // Встановлюємо функцію за замовчуванням та початкові значення
21. currentFunction = new Function1();
22. numericUpDown\_Xmin.Value = -10;
23. numericUpDown\_Xmax.Value = 10;
24. numericUpDown\_Points.Value = 10000;
25. checkBox\_autoY.Checked = true;
26. radioButton\_f1.Checked = true;
27. // Ініціалізуємо попередні значення
28. prevXmin = numericUpDown\_Xmin.Value;
29. prevXmax = numericUpDown\_Xmax.Value;
30. prevYmin = numericUpDown\_Ymin.Value;
31. prevYmax = numericUpDown\_Ymax.Value;
32. // Встановлюємо допустимі мінімальні та максимальні значення для Y
33. numericUpDown\_Ymin.Minimum = decimal.MinValue;
34. numericUpDown\_Ymin.Maximum = decimal.MaxValue;
35. numericUpDown\_Ymax.Minimum = decimal.MinValue;
36. numericUpDown\_Ymax.Maximum = decimal.MaxValue;
37. }
38. // Обробник для вибору функції f1
39. private void radioButton\_f1\_CheckedChanged(object sender, EventArgs e)
40. {
41. if (radioButton\_f1.Checked)
42. {
43. currentFunction = new Function1();
44. renderControl.Invalidate();
45. }
46. }
47. // Обробник для вибору функції f2
48. private void radioButton\_f2\_CheckedChanged(object sender, EventArgs e)
49. {
50. if (radioButton\_f2.Checked)
51. {
52. currentFunction = new Function2();
53. renderControl.Invalidate();
54. }
55. }
56. // Перевірка та оновлення значення Xmin
57. private void numericUpDown\_Xmin\_ValueChanged(object sender, EventArgs e)
58. {
59. if (numericUpDown\_Xmin.Value >= numericUpDown\_Xmax.Value)
60. {
61. MessageBox.Show("Xmin не може бути більшим або рівним Xmax.", "Неприпустиме значення", MessageBoxButtons.OK, MessageBoxIcon.Warning);
62. numericUpDown\_Xmin.Value = prevXmin;
63. }
64. else
65. {
66. prevXmin = numericUpDown\_Xmin.Value;
67. renderControl.Invalidate();
68. }
69. }
70. // Перевірка та оновлення значення Xmax
71. private void numericUpDown\_Xmax\_ValueChanged(object sender, EventArgs e)
72. {
73. if (numericUpDown\_Xmax.Value <= numericUpDown\_Xmin.Value)
74. {
75. MessageBox.Show("Xmax не може бути меншим або рівним Xmin.", "Неприпустиме значення", MessageBoxButtons.OK, MessageBoxIcon.Warning);
76. numericUpDown\_Xmax.Value = prevXmax;
77. }
78. else
79. {
80. prevXmax = numericUpDown\_Xmax.Value;
81. renderControl.Invalidate();
82. }
83. }
84. // Обробка автоматичного масштабування Y
85. private void checkBox\_autoY\_CheckedChanged(object sender, EventArgs e)
86. {
87. bool isChecked = checkBox\_autoY.Checked;
88. numericUpDown\_Ymin.Visible = !isChecked;
89. numericUpDown\_Ymax.Visible = !isChecked;
90. labelYmin.Visible = !isChecked;
91. labelYmax.Visible = !isChecked;
92. if (!isChecked)
93. {
94. // При вимкненні авто Y встановлюємо значення з авто діапазону
95. numericUpDown\_Ymin.Value = (decimal)autoYmin;
96. numericUpDown\_Ymax.Value = (decimal)autoYmax;
97. // Оновлюємо попередні значення
98. prevYmin = numericUpDown\_Ymin.Value;
99. prevYmax = numericUpDown\_Ymax.Value;
100. }
101. renderControl.Invalidate();
102. }
103. // Перевірка та оновлення значення Ymin
104. private void numericUpDown\_Ymin\_ValueChanged(object sender, EventArgs e)
105. {
106. if (numericUpDown\_Ymin.Value >= numericUpDown\_Ymax.Value)
107. {
108. MessageBox.Show("Ymin не може бути більшим або рівним Ymax.", "Неприпустиме значення", MessageBoxButtons.OK, MessageBoxIcon.Warning);
109. numericUpDown\_Ymin.Value = prevYmin;
110. }
111. else
112. {
113. prevYmin = numericUpDown\_Ymin.Value;
114. renderControl.Invalidate();
115. }
116. }
117. // Перевірка та оновлення значення Ymax
118. private void numericUpDown\_Ymax\_ValueChanged(object sender, EventArgs e)
119. {
120. if (numericUpDown\_Ymax.Value <= numericUpDown\_Ymin.Value)
121. {
122. MessageBox.Show("Ymax не може бути меншим або рівним Ymin.", "Неприпустиме значення", MessageBoxButtons.OK, MessageBoxIcon.Warning);
123. numericUpDown\_Ymax.Value = prevYmax;
124. }
125. else
126. {
127. prevYmax = numericUpDown\_Ymax.Value;
128. renderControl.Invalidate();
129. }
130. }
131. // Оновлення при зміні кількості точок
132. private void numericUpDown\_Points\_ValueChanged(object sender, EventArgs e)
133. {
134. renderControl.Invalidate();
135. }
136. // Метод для встановлення автоматичного діапазону Y
137. public void SetAutoYRange(double minY, double maxY)
138. {
139. autoYmin = minY;
140. autoYmax = maxY;
141. }
142. // Властивості для доступу до значень з RenderControl
143. public double Xmin => (double)numericUpDown\_Xmin.Value;
144. public double Xmax => (double)numericUpDown\_Xmax.Value;
145. public double Ymin => (double)numericUpDown\_Ymin.Value;
146. public double Ymax => (double)numericUpDown\_Ymax.Value;
147. public int Points => (int)numericUpDown\_Points.Value;
148. public bool AutoY => checkBox\_autoY.Checked;
149. public FunctionBase CurrentFunction => currentFunction;
150. }
151. }

### Код файлу (RenderControl.cs)

1. using System;
2. using System.Collections.Generic;
3. using System.Windows.Forms;
4. using static Minakov\_Task03.OpenGL;
5. namespace Minakov\_Task03
6. {
7. public partial class RenderControl : OpenGL
8. {
9. public RenderControl()
10. {
11. InitializeComponent();
12. }
13. private void RenderControl\_Render(object sender, EventArgs e)
14. {
15. if (this.FindForm() is not MainForm mainForm)
16. return;
17. double Xmin = mainForm.Xmin;
18. double Xmax = mainForm.Xmax;
19. int Points = mainForm.Points;
20. bool AutoY = mainForm.AutoY;
21. FunctionBase function = mainForm.CurrentFunction;
22. double[] xValues = new double[Points];
23. double[] yValues = new double[Points];
24. double step = (Xmax - Xmin) / (Points - 1);
25. double calculatedYmin = double.MaxValue;
26. double calculatedYmax = double.MinValue;
27. for (int i = 0; i < Points; i++)
28. {
29. double x = Xmin + step \* i;
30. double y = function.Evaluate(x);
31. xValues[i] = x;
32. yValues[i] = y;
33. if (!double.IsNaN(y) && !double.IsInfinity(y))
34. {
35. if (y < calculatedYmin) calculatedYmin = y;
36. if (y > calculatedYmax) calculatedYmax = y;
37. }
38. }
39. // Завжди оновлюємо автоматичний діапазон Y
40. mainForm.SetAutoYRange(calculatedYmin, calculatedYmax);
41. double Ymin, Ymax;
42. if (AutoY)
43. {
44. Ymin = calculatedYmin;
45. Ymax = calculatedYmax;
46. }
47. else
48. {
49. Ymin = mainForm.Ymin;
50. Ymax = mainForm.Ymax;
51. }
52. if (Ymin == Ymax)
53. {
54. Ymin -= 1;
55. Ymax += 1;
56. }
57. glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);
58. glViewport(0, 0, Width, Height);
59. glMatrixMode(GL\_PROJECTION);
60. glLoadIdentity();
61. // Налаштування системи координат
62. glOrtho(Xmin, Xmax, Ymin, Ymax, -1, 1);
63. // Установка модельно-видової матриці
64. glMatrixMode(GL\_MODELVIEW);
65. glLoadIdentity();
66. glColor3d(0.8, 0.8, 0.8); // Світло-сірий колір для сітки
67. glLineWidth(1f);
68. double gridSpacing = 1.0; // Інтервал між лініями сітки
69. // Вертикальні лінії сітки
70. glBegin(GL\_LINES);
71. double xGridStart = Math.Floor(Xmin / gridSpacing) \* gridSpacing;
72. double xGridEnd = Xmax;
73. for (double x = xGridStart; x <= xGridEnd; x += gridSpacing)
74. {
75. glVertex2d(x, Ymin);
76. glVertex2d(x, Ymax);
77. }
78. glEnd();
79. // Горизонтальні лінії сітки
80. glBegin(GL\_LINES);
81. double yGridStart = Math.Floor(Ymin / gridSpacing) \* gridSpacing;
82. double yGridEnd = Ymax;
83. for (double y = yGridStart; y <= yGridEnd; y += gridSpacing)
84. {
85. glVertex2d(Xmin, y);
86. glVertex2d(Xmax, y);
87. }
88. glEnd();
89. // Рисуємо осі
90. glColor3d(0.0, 0.0, 0.0); // Чорний колір
91. glLineWidth(2f);
92. glBegin(GL\_LINES);
93. // Ось X
94. glVertex2d(Xmin, 0);
95. glVertex2d(Xmax, 0);
96. // Ось Y
97. glVertex2d(0, Ymin);
98. glVertex2d(0, Ymax);
99. glEnd();
100. // Знаходимо точки розриву
101. List<double> discontinuities = function.GetDiscontinuities(Xmin, Xmax);
102. if (discontinuities.Count > 0)
103. {
104. glEnable(GL\_LINE\_STIPPLE);
105. glLineStipple(1, 0x00FF); // Шаблон пунктирної лінії
106. glColor3d(0.0, 0.8, 0.0); // Темно-зелений колір
107. glLineWidth(2f);
108. foreach (double xDisc in discontinuities)
109. {
110. glBegin(GL\_LINES);
111. glVertex2d(xDisc, Ymin);
112. glVertex2d(xDisc, Ymax);
113. glEnd();
114. }
115. glDisable(GL\_LINE\_STIPPLE);
116. }
117. // Рисуємо графік функції
118. glColor3d(1.0, 0.0, 0.0); // Червоний колір
119. glLineWidth(1f);
120. glBegin(GL\_LINE\_STRIP);
121. for (int i = 0; i < Points; i++)
122. {
123. double x = xValues[i];
124. double y = yValues[i];
125. if (!double.IsNaN(y) && !double.IsInfinity(y))
126. {
127. glVertex2d(x, y);
128. }
129. else
130. {
131. // Прериваємо лінію при розриві
132. glEnd();
133. glBegin(GL\_LINE\_STRIP);
134. }
135. }
136. glEnd();
137. // Знаходимо та відмічаємо нулі функції
138. glColor3d(0.0, 0.0, 1.0); // Синій колір
139. glPointSize(7.0f); // Розмір точки
140. glBegin(GL\_POINTS);
141. for (int i = 1; i < Points; i++)
142. {
143. double y1 = yValues[i - 1];
144. double y2 = yValues[i];
145. if ((y1 >= 0 && y2 <= 0) || (y1 <= 0 && y2 >= 0))
146. {
147. // Оцінка нуля за допомогою лінійної інтерполяції
148. double x1 = xValues[i - 1];
149. double x2 = xValues[i];
150. double t = y1 / (y1 - y2);
151. double zeroX = x1 + t \* (x2 - x1);
152. double zeroY = 0;
153. // Рисуємо точку в місці перетину з віссю X
154. glVertex2d(zeroX, zeroY);
155. }
156. }
157. glEnd();
158. glFlush();
159. }
160. }
161. }

# Додаток Г. Лістинг програми до практичної роботи № 4

### Код файлу (Segment.cs)

1. // Клас для представлення відрізка з двома точками (A та B)
2. public class Segment
3. {
4. // Координата X точки A
5. public double AX { get; set; } = -50;
6. // Координата Y точки A
7. public double AY { get; set; } = -50;
8. // Координата X точки B
9. public double BX { get; set; } = 50;
10. // Координата Y точки B
11. public double BY { get; set; } = 50;
12. }

### Код файлу (Parabola.cs)

1. using System.Collections.Generic;
2. using System.Drawing;
3. using System;
4. // Клас для представлення параболи та обчислення точок перетину з відрізком
5. public class Parabola
6. {
7. // Коефіцієнт A параболи
8. public double A { get; set; } = 1;
9. // Коефіцієнт B параболи
10. public double B { get; set; } = 0;
11. // Коефіцієнт C параболи
12. public double C { get; set; } = 0;
13. // Мінімальне значення параметра t
14. public double TMin { get; set; } = -10;
15. // Максимальне значення параметра t
16. public double TMax { get; set; } = 10;
17. // Метод для отримання точок перетину параболи з відрізком
18. public List<PointF> GetIntersectionPoints(Segment segment, int numSegments = 100, double epsilon = 1e-6)
19. {
20. List<PointF> intersectionPoints = new List<PointF>();
21. // Параметри параболи
22. double tMin = TMin;
23. double tMax = TMax;
24. double deltaT = (tMax - tMin) / numSegments;
25. // Параметри відрізка
26. double x3 = segment.AX;
27. double y3 = segment.AY;
28. double x4 = segment.BX;
29. double y4 = segment.BY;
30. // Попередня точка на параболі
31. double tPrev = tMin;
32. double xPrev = tPrev;
33. double yPrev = A \* tPrev \* tPrev + B \* tPrev + C;
34. for (int i = 1; i <= numSegments; i++)
35. {
36. double tCurr = tMin + i \* deltaT;
37. double xCurr = tCurr;
38. double yCurr = A \* tCurr \* tCurr + B \* tCurr + C;
39. // Відрізок параболи
40. double x1 = xPrev;
41. double y1 = yPrev;
42. double x2 = xCurr;
43. double y2 = yCurr;
44. // Вектори напрямку
45. double s1\_x = x2 - x1;
46. double s1\_y = y2 - y1;
47. double s2\_x = x4 - x3;
48. double s2\_y = y4 - y3;
49. double denom = (-s2\_x \* s1\_y + s1\_x \* s2\_y);
50. if (Math.Abs(denom) > epsilon)
51. {
52. double s = (-s1\_y \* (x1 - x3) + s1\_x \* (y1 - y3)) / denom;
53. double t = (s2\_x \* (y1 - y3) - s2\_y \* (x1 - x3)) / denom;
54. if (s >= 0 && s <= 1 && t >= 0 && t <= 1)
55. {
56. // Перетин існує
57. double xi = x1 + t \* s1\_x;
58. double yi = y1 + t \* s1\_y;
59. intersectionPoints.Add(new PointF((float)xi, (float)yi));
60. }
61. }
62. // Оновлюємо попередню точку
63. tPrev = tCurr;
64. xPrev = xCurr;
65. yPrev = yCurr;
66. }
67. return intersectionPoints;
68. }
69. }

### Код файлу (Ellipse.cs)

1. using System.Collections.Generic;
2. using System.Drawing;
3. using System;
4. // Клас для представлення еліпса та обчислення точок перетину з відрізком
5. public class Ellipse
6. {
7. // Координата X центру еліпса
8. public double CenterX { get; set; } = 0;
9. // Координата Y центру еліпса
10. public double CenterY { get; set; } = 0;
11. // Довжина великої піввісі
12. public double SemiMajorAxis { get; set; } = 100;
13. // Довжина малої піввісі
14. public double SemiMinorAxis { get; set; } = 50;
15. // Метод для отримання точок перетину еліпса з відрізком
16. public List<PointF> GetIntersectionPoints(Segment segment, double epsilon = 1e-6)
17. {
18. List<PointF> intersectionPoints = new();
19. double h = CenterX;
20. double k = CenterY;
21. double a = SemiMajorAxis;
22. double b = SemiMinorAxis;
23. double x1 = segment.AX;
24. double y1 = segment.AY;
25. double x2 = segment.BX;
26. double y2 = segment.BY;
27. double dx = x2 - x1;
28. double dy = y2 - y1;
29. // Параметричне рівняння прямої: x = x1 + t\*dx, y = y1 + t\*dy
30. // Підставляємо в рівняння еліпса та вирішуємо щодо t
31. double A = (dx \* dx) / (a \* a) + (dy \* dy) / (b \* b);
32. double B = 2 \* (dx \* (x1 - h) / (a \* a) + dy \* (y1 - k) / (b \* b));
33. double C = ((x1 - h) \* (x1 - h)) / (a \* a) + ((y1 - k) \* (y1 - k)) / (b \* b) - 1;
34. double discriminant = B \* B - 4 \* A \* C;
35. if (discriminant >= -epsilon)
36. {
37. if (discriminant < 0) discriminant = 0; // Урахування погрішності
38. double sqrtDiscriminant = Math.Sqrt(discriminant);
39. double t1 = (-B + sqrtDiscriminant) / (2 \* A);
40. double t2 = (-B - sqrtDiscriminant) / (2 \* A);
41. // Перевіряємо t1
42. if (t1 >= 0 && t1 <= 1)
43. {
44. double xi = x1 + t1 \* dx;
45. double yi = y1 + t1 \* dy;
46. intersectionPoints.Add(new PointF((float)xi, (float)yi));
47. }
48. // Перевіряємо t2
49. if (t2 >= 0 && t2 <= 1)
50. {
51. double xi = x1 + t2 \* dx;
52. double yi = y1 + t2 \* dy;
53. intersectionPoints.Add(new PointF((float)xi, (float)yi));
54. }
55. }
56. return intersectionPoints;
57. }
58. }

### Код файлу (MainForm.cs)

1. using System.Windows.Forms;
2. using System;
3. namespace Minakov\_Task04
4. {
5. public partial class MainForm : Form
6. {
7. // Флаг для визначення, чи оновлюються точки відрізка
8. private bool isUpdatingSegmentPoints = false;
9. public MainForm()
10. {
11. InitializeComponent();
12. // Встановлюємо початкові значення елементів інтерфейсу
13. radioButtonEllipse.Checked = true;
14. numericUpDownHalfA.Value = 100;
15. numericUpDownHalfB.Value = 50;
16. numericUpDownA.Value = 1;
17. numericUpDownB.Value = 0;
18. numericUpDownC.Value = 0;
19. numericUpDownTmin.Value = -10;
20. numericUpDownTmax.Value = 10;
21. numericUpDownA\_X.Value = -50;
22. numericUpDownA\_Y.Value = -50;
23. numericUpDownB\_X.Value = 50;
24. numericUpDownB\_Y.Value = 50;
25. // Встановлюємо значення в RenderControl
26. renderControl.IsEllipseSelected = radioButtonEllipse.Checked;
27. renderControl.Ellipse.CenterX = 0;
28. renderControl.Ellipse.CenterY = 0;
29. renderControl.Ellipse.SemiMajorAxis = (double)numericUpDownHalfA.Value;
30. renderControl.Ellipse.SemiMinorAxis = (double)numericUpDownHalfB.Value;
31. renderControl.Parabola.A = (double)numericUpDownA.Value;
32. renderControl.Parabola.B = (double)numericUpDownB.Value;
33. renderControl.Parabola.C = (double)numericUpDownC.Value;
34. renderControl.Parabola.TMin = (double)numericUpDownTmin.Value;
35. renderControl.Parabola.TMax = (double)numericUpDownTmax.Value;
36. renderControl.Segment.AX = (double)numericUpDownA\_X.Value;
37. renderControl.Segment.AY = (double)numericUpDownA\_Y.Value;
38. renderControl.Segment.BX = (double)numericUpDownB\_X.Value;
39. renderControl.Segment.BY = (double)numericUpDownB\_Y.Value;
40. renderControl.Epsilon = 1e-6;
41. // Підписуємося на подію зміни точок відрізка
42. renderControl.SegmentPointsChanged += RenderControl\_SegmentPointsChanged;
43. }
44. // Обробник події зміни точок відрізка
45. private void RenderControl\_SegmentPointsChanged(object sender, EventArgs e)
46. {
47. isUpdatingSegmentPoints = true;
48. numericUpDownA\_X.Value = (decimal)renderControl.Segment.AX;
49. numericUpDownA\_Y.Value = (decimal)renderControl.Segment.AY;
50. numericUpDownB\_X.Value = (decimal)renderControl.Segment.BX;
51. numericUpDownB\_Y.Value = (decimal)renderControl.Segment.BY;
52. isUpdatingSegmentPoints = false;
53. }
54. // Обробник зміни стану радіокнопки еліпса
55. private void radioButtonEllipse\_CheckedChanged(object sender, EventArgs e)
56. {
57. renderControl.IsEllipseSelected = radioButtonEllipse.Checked;
58. renderControl.Invalidate();
59. }
60. // Обробник зміни стану радіокнопки параболи
61. private void radioButtonParabola\_CheckedChanged(object sender, EventArgs e)
62. {
63. renderControl.IsEllipseSelected = !radioButtonParabola.Checked;
64. renderControl.Invalidate();
65. }
66. // Обробник зміни значення піввісі A еліпса
67. private void numericUpDownHalfA\_ValueChanged(object sender, EventArgs e)
68. {
69. renderControl.Ellipse.SemiMajorAxis = (double)numericUpDownHalfA.Value;
70. renderControl.Invalidate();
71. }
72. // Обробник зміни значення піввісі B еліпса
73. private void numericUpDownHalfB\_ValueChanged(object sender, EventArgs e)
74. {
75. renderControl.Ellipse.SemiMinorAxis = (double)numericUpDownHalfB.Value;
76. renderControl.Invalidate();
77. }
78. // Обробник зміни коефіцієнта A параболи
79. private void numericUpDownA\_ValueChanged(object sender, EventArgs e)
80. {
81. renderControl.Parabola.A = (double)numericUpDownA.Value;
82. renderControl.Invalidate();
83. }
84. // Обробник зміни коефіцієнта B параболи
85. private void numericUpDownB\_ValueChanged(object sender, EventArgs e)
86. {
87. renderControl.Parabola.B = (double)numericUpDownB.Value;
88. renderControl.Invalidate();
89. }
90. // Обробник зміни коефіцієнта C параболи
91. private void numericUpDownC\_ValueChanged(object sender, EventArgs e)
92. {
93. renderControl.Parabola.C = (double)numericUpDownC.Value;
94. renderControl.Invalidate();
95. }
96. // Обробник зміни мінімального параметра t параболи
97. private void numericUpDownTmin\_ValueChanged(object sender, EventArgs e)
98. {
99. renderControl.Parabola.TMin = (double)numericUpDownTmin.Value;
100. renderControl.Invalidate();
101. }
102. // Обробник зміни максимального параметра t параболи
103. private void numericUpDownTmax\_ValueChanged(object sender, EventArgs e)
104. {
105. renderControl.Parabola.TMax = (double)numericUpDownTmax.Value;
106. renderControl.Invalidate();
107. }
108. // Обробник зміни координати X точки A відрізка
109. private void numericUpDownA\_X\_ValueChanged(object sender, EventArgs e)
110. {
111. if (!isUpdatingSegmentPoints)
112. {
113. renderControl.Segment.AX = (double)numericUpDownA\_X.Value;
114. renderControl.Invalidate();
115. }
116. }
117. // Обробник зміни координати Y точки A відрізка
118. private void numericUpDownA\_Y\_ValueChanged(object sender, EventArgs e)
119. {
120. if (!isUpdatingSegmentPoints)
121. {
122. renderControl.Segment.AY = (double)numericUpDownA\_Y.Value;
123. renderControl.Invalidate();
124. }
125. }
126. // Обробник зміни координати X точки B відрізка
127. private void numericUpDownB\_X\_ValueChanged(object sender, EventArgs e)
128. {
129. if (!isUpdatingSegmentPoints)
130. {
131. renderControl.Segment.BX = (double)numericUpDownB\_X.Value;
132. renderControl.Invalidate();
133. }
134. }
135. // Обробник зміни координати Y точки B відрізка
136. private void numericUpDownB\_Y\_ValueChanged(object sender, EventArgs e)
137. {
138. if (!isUpdatingSegmentPoints)
139. {
140. renderControl.Segment.BY = (double)numericUpDownB\_Y.Value;
141. renderControl.Invalidate();
142. }
143. }
144. }
145. }

### Код файлу (RenderControl.cs)

1. using System;
2. using System.Collections.Generic;
3. using System.Drawing;
4. using System.Windows.Forms;
5. namespace Minakov\_Task04
6. {
7. public partial class RenderControl : OpenGL
8. {
9. // Властивості для зберігання об'єктів
10. public bool IsEllipseSelected { get; set; } = true;
11. public Ellipse Ellipse { get; set; } = new Ellipse();
12. public Parabola Parabola { get; set; } = new Parabola();
13. public Segment Segment { get; set; } = new Segment();
14. // Epsilon для обчислень
15. public double Epsilon { get; set; } = 1e-6;
16. // Флаг для визначення, яку точку встановлюємо
17. private bool isSettingPointA = true;
18. // Подія для повідомлення про зміну точок відрізка
19. public event EventHandler SegmentPointsChanged;
20. public RenderControl()
21. {
22. InitializeComponent();
23. }
24. // Метод рендерингу графіки
25. private void RenderControl\_Render(object sender, EventArgs e)
26. {
27. // Налаштування системи координат OpenGL
28. glViewport(0, 0, Width, Height);
29. glMatrixMode(GL\_PROJECTION);
30. glLoadIdentity();
31. double worldWidth = 200;
32. double worldHeight = 200;
33. double aspectRatio = (double)Width / Height;
34. if (aspectRatio >= 1.0)
35. {
36. worldWidth = 200 \* aspectRatio;
37. }
38. else
39. {
40. worldHeight = 200 / aspectRatio;
41. }
42. glOrtho(-worldWidth / 2, worldWidth / 2, -worldHeight / 2, worldHeight / 2, -1, 1);
43. glMatrixMode(GL\_MODELVIEW);
44. glLoadIdentity();
45. glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);
46. glLineWidth(1.0f);
47. // Малюємо сітку
48. DrawGrid(worldWidth, worldHeight, 10);
49. // Малюємо осі координат
50. glLineWidth(3.0f);
51. glColor3f(0.5f, 0.5f, 0.5f);
52. glBegin(GL\_LINES);
53. // Ось X
54. glVertex2d(-worldWidth / 2, 0);
55. glVertex2d(worldWidth / 2, 0);
56. // Ось Y
57. glVertex2d(0, -worldHeight / 2);
58. glVertex2d(0, worldHeight / 2);
59. glEnd();
60. // Малюємо відрізок
61. glColor3f(0.0f, 1.0f, 0.0f); // Зелений колір
62. glBegin(GL\_LINES);
63. glVertex2d(Segment.AX, Segment.AY);
64. glVertex2d(Segment.BX, Segment.BY);
65. glEnd();
66. // Малюємо точки A і B
67. glPointSize(8.5f);
68. glBegin(GL\_POINTS);
69. // Точка A
70. glColor3f(1.0f, 0.5f, 0.0f); // Помаранчевий
71. glVertex2d(Segment.AX, Segment.AY);
72. // Точка B
73. glColor3f(0.0f, 0.5f, 1.0f); // Блакитний
74. glVertex2d(Segment.BX, Segment.BY);
75. glEnd();
76. glLineWidth(1.5f);
77. if (IsEllipseSelected)
78. {
79. // Малюємо еліпс
80. glColor3f(0.0f, 0.0f, 1.0f); // Синій колір
81. int numSegments = 100;
82. double theta = 0.0;
83. double deltaTheta = 2 \* Math.PI / numSegments;
84. glBegin(GL\_LINE\_LOOP);
85. for (int i = 0; i < numSegments; i++)
86. {
87. double x = Ellipse.CenterX + Ellipse.SemiMajorAxis \* Math.Cos(theta);
88. double y = Ellipse.CenterY + Ellipse.SemiMinorAxis \* Math.Sin(theta);
89. glVertex2d(x, y);
90. theta += deltaTheta;
91. }
92. glEnd();
93. // Обчислення і малювання точок перетину
94. List<PointF> intersectionPoints = Ellipse.GetIntersectionPoints(Segment, Epsilon);
95. // Малюємо точки перетину
96. glPointSize(8.5f);
97. glColor3f(1.0f, 0.0f, 0.0f); // Червоний колір
98. glBegin(GL\_POINTS);
99. foreach (PointF p in intersectionPoints)
100. {
101. glVertex2d(p.X, p.Y);
102. }
103. glEnd();
104. }
105. else
106. {
107. // Малюємо параболу
108. glColor3f(0.0f, 0.0f, 1.0f); // Синій колір
109. int numSegments = 100;
110. double tMin = Parabola.TMin;
111. double tMax = Parabola.TMax;
112. double deltaT = (tMax - tMin) / numSegments;
113. glBegin(GL\_LINE\_STRIP);
114. for (int i = 0; i <= numSegments; i++)
115. {
116. double t = tMin + i \* deltaT;
117. double x = t;
118. double y = Parabola.A \* t \* t + Parabola.B \* t + Parabola.C;
119. glVertex2d(x, y);
120. }
121. glEnd();
122. // Обчислення і малювання точок перетину
123. List<PointF> intersectionPoints = Parabola.GetIntersectionPoints(Segment, numSegments, Epsilon);
124. // Малюємо точки перетину
125. glPointSize(8.5f);
126. glColor3f(1.0f, 0.0f, 0.0f); // Червоний колір
127. glBegin(GL\_POINTS);
128. foreach (PointF p in intersectionPoints)
129. {
130. glVertex2d(p.X, p.Y);
131. }
132. glEnd();
133. }
134. }
135. // Обробник події кліку миші
136. protected override void OnMouseDown(MouseEventArgs e)
137. {
138. base.OnMouseDown(e);
139. // Перетворюємо координати курсора в світові координати
140. double worldX, worldY;
141. ScreenToWorld(e.X, e.Y, out worldX, out worldY);
142. if (isSettingPointA)
143. {
144. // Встановлюємо точку A
145. Segment.AX = worldX;
146. Segment.AY = worldY;
147. isSettingPointA = false; // Наступний клік встановить точку B
148. }
149. else
150. {
151. // Встановлюємо точку B
152. Segment.BX = worldX;
153. Segment.BY = worldY;
154. isSettingPointA = true; // Скидання флагу
155. }
156. // Повідомляємо підписників про зміну точок відрізка
157. SegmentPointsChanged?.Invoke(this, EventArgs.Empty);
158. // Перемальовуємо контроль
159. Invalidate();
160. }
161. // Метод для перетворення екранних координат у світові
162. private void ScreenToWorld(int xScreen, int yScreen, out double xWorld, out double yWorld)
163. {
164. int viewportWidth = Width;
165. int viewportHeight = Height;
166. double worldWidth = 200;
167. double worldHeight = 200;
168. double aspectRatio = (double)viewportWidth / viewportHeight;
169. if (aspectRatio >= 1.0)
170. {
171. worldWidth = 200 \* aspectRatio;
172. }
173. else
174. {
175. worldHeight = 200 / aspectRatio;
176. }
177. xWorld = (double)xScreen / viewportWidth \* worldWidth - worldWidth / 2;
178. yWorld = -(double)yScreen / viewportHeight \* worldHeight + worldHeight / 2;
179. }
180. // Метод для малювання сітки
181. private static void DrawGrid(double worldWidth, double worldHeight, double gridSpacing)
182. {
183. glColor3f(0.85f, 0.85f, 0.85f); // Світло-сірий колір
184. glBegin(GL\_LINES);
185. // Вертикальні лінії
186. for (double x = 0; x <= worldWidth / 2; x += gridSpacing)
187. {
188. // Позитивні X
189. glVertex2d(x, -worldHeight / 2);
190. glVertex2d(x, worldHeight / 2);
191. // Негативні X
192. if (x != 0)
193. {
194. glVertex2d(-x, -worldHeight / 2);
195. glVertex2d(-x, worldHeight / 2);
196. }
197. }
198. // Горизонтальні лінії
199. for (double y = 0; y <= worldHeight / 2; y += gridSpacing)
200. {
201. // Позитивні Y
202. glVertex2d(-worldWidth / 2, y);
203. glVertex2d(worldWidth / 2, y);
204. // Негативні Y
205. if (y != 0)
206. {
207. glVertex2d(-worldWidth / 2, -y);
208. glVertex2d(worldWidth / 2, -y);
209. }
210. }
211. glEnd();
212. }
213. }
214. }

# Додаток Ґ. Лістинг програми до практичної роботи № 5

### Код файлу (MainForm.cs)

1. using System;
2. using System.Drawing;
3. using System.Windows.Forms;
4. using static Minakov\_Task05.OpenGL;
5. namespace Minakov\_Task05
6. {
7. public partial class MainForm : Form
8. {
9. // Зберігає останню позицію миші для обробки руху
10. private Point lastMousePos;
11. // Кути обертання по осях X та Y
12. private float angleX = 0;
13. private float angleY = 0;
14. public MainForm()
15. {
16. InitializeComponent();
17. // Додаємо обробники подій для управління обертанням сцени мишею
18. renderControl1.MouseDown += RenderControl1\_MouseDown;
19. renderControl1.MouseMove += RenderControl1\_MouseMove;
20. // Додаємо обробники подій для радіокнопок режиму рендерингу
21. radioButtonFill.CheckedChanged += RadioButtonFill\_CheckedChanged;
22. radioButtonLine.CheckedChanged += RadioButtonLine\_CheckedChanged;
23. radioButtonPerspective.CheckedChanged += radioButtonPerspective\_CheckedChanged;
24. radioButtonOrtho.CheckedChanged += radioButtonOrtho\_CheckedChanged;
25. // Встановлюємо початковий режим рендерингу на каркасний
26. radioButtonLine.Checked = true;
27. // Встановлюємо початкову проекцію на ортографічну
28. radioButtonOrtho.Checked = true;
29. // Ініціалізація стану управління плоскостями відсічення
30. checkBox\_Clip.Checked = false;
31. numericUpDownA.Visible = false;
32. numericUpDownB.Visible = false;
33. numericUpDownC.Visible = false;
34. numericUpDownD.Visible = false;
35. label1.Visible = false;
36. label2.Visible = false;
37. label3.Visible = false;
38. label4.Visible = false;
39. // Встановлення значень за замовчуванням для коефіцієнтів плоскості відсічення
40. numericUpDownA.Value = 1;
41. numericUpDownB.Value = 0;
42. numericUpDownC.Value = 0;
43. numericUpDownD.Value = 0;
44. // Встановлення плоскості відсічення за замовчуванням
45. double[] defaultPlane = { (double)numericUpDownA.Value, (double)numericUpDownB.Value, (double)numericUpDownC.Value, (double)numericUpDownD.Value };
46. renderControl1.SetClippingPlane(false, defaultPlane);
47. // Додаємо обробник події для чекбоксу плоскості відсічення
48. checkBox\_Clip.CheckedChanged += checkBox\_Clip\_CheckedChanged;
49. // Додаємо обробники подій для зміни значень коефіцієнтів плоскості відсічення
50. numericUpDownA.ValueChanged += numericUpDownA\_ValueChanged;
51. numericUpDownB.ValueChanged += numericUpDownB\_ValueChanged;
52. numericUpDownC.ValueChanged += numericUpDownC\_ValueChanged;
53. numericUpDownD.ValueChanged += numericUpDownD\_ValueChanged;
54. }
55. // Обробник події натискання кнопки миші
56. private void RenderControl1\_MouseDown(object sender, MouseEventArgs e)
57. {
58. lastMousePos = e.Location;
59. }
60. // Обробник події переміщення миші
61. private void RenderControl1\_MouseMove(object sender, MouseEventArgs e)
62. {
63. if (e.Button == MouseButtons.Left)
64. {
65. // Зміна кутів обертання на основі переміщення миші
66. angleX += (e.Y - lastMousePos.Y);
67. angleY += (e.X - lastMousePos.X);
68. lastMousePos = e.Location;
69. // Передача нових кутів в RenderControl і перерисовка
70. renderControl1.SetRotation(angleX, angleY);
71. renderControl1.Invalidate();
72. }
73. }
74. // Обробник зміни стану радіокнопки "Заливка"
75. private void RadioButtonFill\_CheckedChanged(object sender, EventArgs e)
76. {
77. if (radioButtonFill.Checked)
78. {
79. renderControl1.SetRenderingMode(true); // Встановлення режиму заливки
80. }
81. }
82. // Обробник зміни стану радіокнопки "Каркас"
83. private void RadioButtonLine\_CheckedChanged(object sender, EventArgs e)
84. {
85. if (radioButtonLine.Checked)
86. {
87. renderControl1.SetRenderingMode(false); // Встановлення каркасного режиму
88. }
89. }
90. // Обробник зміни стану чекбоксу плоскості відсічення
91. private void checkBox\_Clip\_CheckedChanged(object sender, EventArgs e)
92. {
93. bool isChecked = checkBox\_Clip.Checked;
94. // Показуємо або ховаємо елементи керування плоскостями відсічення
95. numericUpDownA.Visible = isChecked;
96. numericUpDownB.Visible = isChecked;
97. numericUpDownC.Visible = isChecked;
98. numericUpDownD.Visible = isChecked;
99. label1.Visible = isChecked;
100. label2.Visible = isChecked;
101. label3.Visible = isChecked;
102. label4.Visible = isChecked;
103. if (isChecked)
104. {
105. UpdateClippingPlane(); // Оновлюємо плоскості відсічення
106. }
107. else
108. {
109. renderControl1.SetClippingPlane(false, null); // Вимикаємо плоскості відсічення
110. }
111. }
112. // Метод для оновлення плоскості відсічення на основі введених значень
113. private void UpdateClippingPlane()
114. {
115. double A = (double)numericUpDownA.Value;
116. double B = (double)numericUpDownB.Value;
117. double C = (double)numericUpDownC.Value;
118. double D = (double)numericUpDownD.Value;
119. double[] planeEquation = { A, B, C, D };
120. renderControl1.SetClippingPlane(true, planeEquation);
121. }
122. // Обробники змін значень числових полів для плоскості відсічення
123. private void numericUpDownA\_ValueChanged(object sender, EventArgs e)
124. {
125. if (checkBox\_Clip.Checked)
126. {
127. UpdateClippingPlane();
128. }
129. }
130. private void numericUpDownB\_ValueChanged(object sender, EventArgs e)
131. {
132. if (checkBox\_Clip.Checked)
133. {
134. UpdateClippingPlane();
135. }
136. }
137. private void numericUpDownC\_ValueChanged(object sender, EventArgs e)
138. {
139. if (checkBox\_Clip.Checked)
140. {
141. UpdateClippingPlane();
142. }
143. }
144. private void numericUpDownD\_ValueChanged(object sender, EventArgs e)
145. {
146. if (checkBox\_Clip.Checked)
147. {
148. UpdateClippingPlane();
149. }
150. }
151. // Обробник зміни стану радіокнопки "Перспективна проекція"
152. private void radioButtonPerspective\_CheckedChanged(object sender, EventArgs e)
153. {
154. if (radioButtonPerspective.Checked)
155. {
156. renderControl1.SetProjectionMode(false); // Встановлення перспективної проекції
157. }
158. }
159. // Обробник зміни стану радіокнопки "Ортографічна проекція"
160. private void radioButtonOrtho\_CheckedChanged(object sender, EventArgs e)
161. {
162. if (radioButtonOrtho.Checked)
163. {
164. renderControl1.SetProjectionMode(true); // Встановлення ортографічної проекції
165. }
166. }
167. }
168. }

### Код файлу (RenderControl.cs)

1. using System;
2. using System.Windows.Forms;
3. using static Minakov\_Task05.OpenGL;
4. namespace Minakov\_Task05
5. {
6. public partial class RenderControl : OpenGL
7. {
8. // Кути обертання по осях X та Y
9. private float angleX = 0;
10. private float angleY = 0;
11. // Режим рендерингу: заповнений (true) або каркасний (false)
12. private bool isFilledMode = false;
13. // Параметри плоскості відсічення
14. private bool clippingEnabled = false;
15. private double[] clipPlane = { 1.0, 0.0, 0.0, 0.0 };
16. // Режим проекції: ортографічна (true) або перспективна (false)
17. private bool isOrthographic = true;
18. // Ідентифікатори списків відображення для фігур
19. private uint sphereListFilled;
20. private uint coneListFilled;
21. private uint diskListFilled;
22. private uint sphereListWireframe;
23. private uint coneListWireframe;
24. private uint diskListWireframe;
25. // Флаг ініціалізації списків відображення
26. private bool displayListsInitialized = false;
27. public RenderControl()
28. {
29. InitializeComponent();
30. // Ініціалізація списків відображення перенесена в окремий метод
31. }
32. // Встановлює кути обертання
33. public void SetRotation(float x, float y)
34. {
35. angleX = x;
36. angleY = y;
37. }
38. // Встановлює режим рендерингу
39. public void SetRenderingMode(bool filled)
40. {
41. isFilledMode = filled;
42. Invalidate(); // Перерисовуємо контрол
43. }
44. // Встановлює параметри плоскості відсічення
45. public void SetClippingPlane(bool enabled, double[] planeEquation)
46. {
47. clippingEnabled = enabled;
48. if (planeEquation != null && planeEquation.Length == 4)
49. {
50. clipPlane = planeEquation;
51. }
52. Invalidate(); // Перерисовуємо контрол
53. }
54. // Встановлює режим проекції
55. public void SetProjectionMode(bool orthographic)
56. {
57. isOrthographic = orthographic;
58. Invalidate(); // Перерисовуємо контрол
59. }
60. // Метод рендерингу сцени
61. private void RenderControl\_Render(object sender, EventArgs e)
62. {
63. // Перевіряємо, чи ініціалізовані списки відображення
64. if (!displayListsInitialized)
65. {
66. InitializeDisplayLists();
67. displayListsInitialized = true;
68. }
69. // Очищення буферів кольору та глибини
70. glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);
71. // Увімкнення тесту глибини
72. glEnable(GL\_DEPTH\_TEST);
73. // Встановлення області перегляду (viewport) для ізотропної системи координат
74. int side = Math.Min(Width, Height);
75. int viewportX = (Width - side) / 2;
76. int viewportY = (Height - side) / 2;
77. glViewport(viewportX, viewportY, side, side);
78. // Встановлення матриці проекції
79. glMatrixMode(GL\_PROJECTION);
80. glLoadIdentity();
81. if (isOrthographic)
82. {
83. // Встановлення ортографічної проекції
84. glOrtho(-10, 10, -10, 10, -10, 10);
85. }
86. else
87. {
88. // Встановлення перспективної проекції
89. gluPerspective(45.0, 1.0, 1.0, 100.0);
90. }
91. // Встановлення матриці моделі-вид
92. glMatrixMode(GL\_MODELVIEW);
93. glLoadIdentity();
94. if (!isOrthographic)
95. {
96. // Переміщення камери назад для перспективної проекції
97. glTranslatef(0.0f, 0.0f, -20.0f);
98. }
99. // Застосування обертання на основі кутів
100. glRotatef(angleX, 1.0f, 0.0f, 0.0f);
101. glRotatef(angleY, 0.0f, 1.0f, 0.0f);
102. // Встановлення режиму полігонів для осей та сітки
103. glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE);
104. // Малюємо координатні осі
105. DrawCoordinateAxes();
106. // Малюємо координатну сітку в площині X0Z
107. DrawGrid();
108. // Встановлюємо режим полігонів для фігур
109. if (isFilledMode)
110. {
111. glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_FILL);
112. }
113. else
114. {
115. glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE);
116. }
117. // Увімкнення освітлення
118. glEnable(GL\_LIGHTING);
119. glEnable(GL\_LIGHT0);
120. glDisable(GL\_COLOR\_MATERIAL);
121. // Налаштування параметрів освітлення
122. SetupLighting();
123. // Малюємо три квадратичні фігури
124. DrawFigures();
125. // Вимкнення освітлення
126. glDisable(GL\_LIGHT0);
127. glDisable(GL\_LIGHTING);
128. // Вимкнення тесту глибини (за потребою)
129. //glDisable(GL\_DEPTH\_TEST);
130. }
131. // Ініціалізація списків відображення для фігур
132. private void InitializeDisplayLists()
133. {
134. // Генерація ідентифікаторів списків відображення
135. sphereListFilled = glGenLists(1);
136. coneListFilled = glGenLists(1);
137. diskListFilled = glGenLists(1);
138. sphereListWireframe = glGenLists(1);
139. coneListWireframe = glGenLists(1);
140. diskListWireframe = glGenLists(1);
141. // Створення квадрика для малювання фігур
142. IntPtr quadric = gluNewQuadric();
143. // Створення списків відображення для заповнених фігур
144. gluQuadricDrawStyle(quadric, GLU\_FILL);
145. // Сфера (заповнена)
146. glNewList(sphereListFilled, GL\_COMPILE);
147. gluSphere(quadric, 1.5, 16, 16);
148. glEndList();
149. // Конус (заповнений)
150. glNewList(coneListFilled, GL\_COMPILE);
151. gluCylinder(quadric, 3.0, 1.0, 2.5, 16, 16);
152. glEndList();
153. // Диск (заповнений)
154. glNewList(diskListFilled, GL\_COMPILE);
155. gluPartialDisk(quadric, 0.0, 2.0, 16, 1, 225.0, 180.0);
156. glEndList();
157. // Створення списків відображення для каркасних фігур
158. gluQuadricDrawStyle(quadric, GLU\_LINE);
159. // Сфера (каркасна)
160. glNewList(sphereListWireframe, GL\_COMPILE);
161. gluSphere(quadric, 1.5, 16, 16);
162. glEndList();
163. // Конус (каркасний)
164. glNewList(coneListWireframe, GL\_COMPILE);
165. gluCylinder(quadric, 3.0, 1.0, 2.5, 16, 16);
166. glEndList();
167. // Диск (каркасний)
168. glNewList(diskListWireframe, GL\_COMPILE);
169. gluPartialDisk(quadric, 0.0, 2.0, 16, 1, 225.0, 180.0);
170. glEndList();
171. // Видалення квадрика після створення списків
172. gluDeleteQuadric(quadric);
173. }
174. // Метод для малювання координатних осей
175. private void DrawCoordinateAxes()
176. {
177. glLineWidth(2.0f);
178. glBegin(GL\_LINES);
179. // Вісь X (червона)
180. glColor3d(1, 0, 0);
181. glVertex3d(0, 0, 0);
182. glVertex3d(5, 0, 0);
183. // Вісь Y (зелена)
184. glColor3d(0, 1, 0);
185. glVertex3d(0, 0, 0);
186. glVertex3d(0, 5, 0);
187. // Вісь Z (синя)
188. glColor3d(0, 0, 1);
189. glVertex3d(0, 0, 0);
190. glVertex3d(0, 0, 5);
191. glEnd();
192. }
193. // Метод для малювання координатної сітки в площині X0Z
194. private void DrawGrid()
195. {
196. glColor3d(0.7, 0.7, 0.7); // Світло-сірий колір
197. glLineWidth(1.0f);
198. int gridSize = 10; // Розмір сітки
199. glBegin(GL\_LINES);
200. for (int i = -gridSize; i <= gridSize; i++)
201. {
202. // Лінії паралельні осі X
203. glVertex3d(-gridSize, 0, i);
204. glVertex3d(gridSize, 0, i);
205. // Лінії паралельні осі Z
206. glVertex3d(i, 0, -gridSize);
207. glVertex3d(i, 0, gridSize);
208. }
209. glEnd();
210. }
211. // Метод для налаштування параметрів освітлення
212. private void SetupLighting()
213. {
214. // Параметри позиції світла
215. float[] lightPosition = { 5.0f, 10.0f, 5.0f, 1.0f };
216. // Параметри навколишнього освітлення
217. float[] lightAmbient = { 0.3f, 0.3f, 0.3f, 1.0f };
218. // Параметри дифузного освітлення
219. float[] lightDiffuse = { 0.8f, 0.8f, 0.8f, 1.0f };
220. // Параметри спекулярного освітлення
221. float[] lightSpecular = { 1.0f, 1.0f, 1.0f, 1.0f };
222. // Встановлення параметрів світла
223. glLightfv(GL\_LIGHT0, GL\_POSITION, lightPosition);
224. glLightfv(GL\_LIGHT0, GL\_AMBIENT, lightAmbient);
225. glLightfv(GL\_LIGHT0, GL\_DIFFUSE, lightDiffuse);
226. glLightfv(GL\_LIGHT0, GL\_SPECULAR, lightSpecular);
227. // Нормалізація нормалей
228. glEnable(GL\_NORMALIZE);
229. }
230. // Метод для малювання фігур
231. private void DrawFigures()
232. {
233. // Увімкнення плоскості відсічення, якщо необхідно
234. if (clippingEnabled)
235. {
236. glEnable(GL\_CLIP\_PLANE0);
237. glClipPlane(GL\_CLIP\_PLANE0, clipPlane);
238. }
239. // Сфера (розташована паралельно осі Z)
240. glPushMatrix();
241. glTranslated(2.5, 1.0, -2.5);
242. // Встановлення матеріалу для сфери
243. float[] sphereAmbient = { 0.25f, 0.0f, 0.0f, 1.0f };
244. float[] sphereDiffuse = { 0.8f, 0.0f, 0.0f, 1.0f }; // Червоний колір
245. float[] sphereSpecular = { 0.9f, 0.9f, 0.9f, 1.0f };
246. float sphereShininess = 50.0f;
247. glMaterialfv(GL\_FRONT, GL\_AMBIENT, sphereAmbient);
248. glMaterialfv(GL\_FRONT, GL\_DIFFUSE, sphereDiffuse);
249. glMaterialfv(GL\_FRONT, GL\_SPECULAR, sphereSpecular);
250. glMaterialf(GL\_FRONT, GL\_SHININESS, sphereShininess);
251. // Виклик відповідного списку відображення
252. if (isFilledMode)
253. {
254. glCallList(sphereListFilled);
255. }
256. else
257. {
258. glCallList(sphereListWireframe);
259. }
260. glPopMatrix();
261. // Вимкнення плоскості відсічення
262. if (clippingEnabled)
263. {
264. glDisable(GL\_CLIP\_PLANE0);
265. }
266. // Усічений конус (розташований паралельно осі Y)
267. glPushMatrix();
268. glTranslated(-3.0, -2.5, -4.5);
269. glRotated(-90, 1, 0, 0);
270. // Встановлення матеріалу для конуса
271. float[] coneAmbient = { 0.0f, 0.25f, 0.0f, 1.0f };
272. float[] coneDiffuse = { 0.0f, 0.8f, 0.0f, 1.0f }; // Зелений колір
273. float[] coneSpecular = { 0.9f, 0.9f, 0.9f, 1.0f };
274. float coneShininess = 30.0f;
275. glMaterialfv(GL\_FRONT, GL\_AMBIENT, coneAmbient);
276. glMaterialfv(GL\_FRONT, GL\_DIFFUSE, coneDiffuse);
277. glMaterialfv(GL\_FRONT, GL\_SPECULAR, coneSpecular);
278. glMaterialf(GL\_FRONT, GL\_SHININESS, coneShininess);
279. if (isFilledMode)
280. {
281. glCallList(coneListFilled);
282. }
283. else
284. {
285. glCallList(coneListWireframe);
286. }
287. glPopMatrix();
288. // Частковий диск (розташований паралельно осі X)
289. glPushMatrix();
290. glTranslated(-3.5, 0.5, 2.0);
291. glRotated(-90, 0, 1, 0);
292. // Встановлення матеріалу для диска
293. float[] diskAmbient = { 0.0f, 0.0f, 0.25f, 1.0f };
294. float[] diskDiffuse = { 0.0f, 0.0f, 0.8f, 1.0f }; // Синій колір
295. float[] diskSpecular = { 0.9f, 0.9f, 0.9f, 1.0f };
296. float diskShininess = 80.0f;
297. glMaterialfv(GL\_FRONT, GL\_AMBIENT, diskAmbient);
298. glMaterialfv(GL\_FRONT, GL\_DIFFUSE, diskDiffuse);
299. glMaterialfv(GL\_FRONT, GL\_SPECULAR, diskSpecular);
300. glMaterialf(GL\_FRONT, GL\_SHININESS, diskShininess);
301. if (isFilledMode)
302. {
303. glCallList(diskListFilled);
304. }
305. else
306. {
307. glCallList(diskListWireframe);
308. }
309. glPopMatrix();
310. }
311. }
312. }

# Додаток Д. Лістинг програми до практичної роботи № 6

### Код файлу (MainForm.cs)

1. using System;
2. using System.Windows.Forms;
3. namespace Minakov\_Task06
4. {
5. public partial class MainForm : Form
6. {
7. public MainForm()
8. {
9. InitializeComponent();
10. }
11. }
12. }

### Код файлу (RenderControl.cs)

1. using System;
2. using System.Windows.Forms;
3. using System.Drawing;
4. using static Minakov\_Task06.OpenGL;
5. using System.Runtime.InteropServices;
6. namespace Minakov\_Task06
7. {
8. // Клас для представлення точки в двовимірному просторі
9. public class Point2D
10. {
11. // Властивості координат X та Y
12. public double X { get; set; }
13. public double Y { get; set; }
14. // Конструктор для ініціалізації точки
15. public Point2D(double x, double y)
16. {
17. X = x;
18. Y = y;
19. }
20. // Метод для обчислення відстані до іншої точки
21. public double DistanceTo(Point2D other)
22. {
23. double dx = other.X - X;
24. double dy = other.Y - Y;
25. return Math.Sqrt(dx \* dx + dy \* dy);
26. }
27. // Статичний метод для обчислення довжини вектора
28. public static double Length(double x, double y)
29. {
30. return Math.Sqrt(x \* x + y \* y);
31. }
32. }
33. // Клас, що представляє модель робочої руки маніпулятора
34. public class RobotArm
35. {
36. // Довжини ланок маніпулятора
37. private double a;
38. private double b;
39. private double c;
40. // Параметри руху
41. private double S;
42. private double oldS;
43. private double psi;
44. private double oldPsi;
45. // Точки кінематичної схеми
46. private Point2D O;
47. private Point2D B;
48. private Point2D C;
49. private Point2D A;
50. // Конструктор для ініціалізації робочої руки
51. public RobotArm(double a, double b, double c)
52. {
53. this.a = a;
54. this.b = b;
55. this.c = c;
56. S = 0.0;
57. oldS = 0.0;
58. psi = 0.0;
59. oldPsi = 0.0;
60. O = new Point2D(0.0, 0.0);
61. B = new Point2D(c, S);
62. C = new Point2D(0.0, 0.0);
63. A = new Point2D(0.0, 0.0);
64. }
65. // Методи для встановлення параметрів S та psi
66. public void SetS(double newS)
67. {
68. S = newS;
69. }
70. public void SetPsi(double newPsi)
71. {
72. psi = newPsi;
73. }
74. // Гетери для параметрів S та psi
75. public double GetS() { return S; }
76. public double GetPsi() { return psi; }
77. // Гетери для точок O, B, C, A
78. public Point2D GetO() { return O; }
79. public Point2D GetB() { return B; }
80. public Point2D GetC() { return C; }
81. public Point2D GetA() { return A; }
82. // Метод для оновлення положення маніпулятора
83. public void UpdateArm()
84. {
85. // Оновлюємо координати точки B
86. B.X = c;
87. B.Y = S;
88. // Обчислюємо відстань між точками O та B
89. double D = O.DistanceTo(B);
90. // Перевіряємо, чи не перевищує відстань максимально допустимий
91. if (D > 2 \* a)
92. {
93. S = oldS;
94. B.X = c; B.Y = S;
95. D = O.DistanceTo(B);
96. }
97. // Обчислюємо точку C на основі поточних координат O та B
98. Point2D computedC = ComputeC(O, B, a);
99. if (computedC == null)
100. {
101. // Якщо обчислення не вдалося, відновлюємо попередні значення
102. S = oldS;
103. psi = oldPsi;
104. B.X = c; B.Y = S;
105. D = O.DistanceTo(B);
106. computedC = ComputeC(O, B, a);
107. }
108. C = computedC;
109. // Обчислюємо координати точки A з урахуванням кута psi
110. double psiRad = Math.PI \* psi / 180.0;
111. A.X = C.X + b \* Math.Cos(psiRad);
112. A.Y = C.Y + b \* Math.Sin(psiRad);
113. // Перевіряємо перетин сегментів, щоб уникнути некоректних положень
114. if (SegmentsIntersect(O.X, O.Y, B.X, B.Y, C.X, C.Y, A.X, A.Y))
115. {
116. // Якщо відбувається перетин, відновлюємо попередні значення
117. S = oldS;
118. psi = oldPsi;
119. B.X = c; B.Y = S;
120. computedC = ComputeC(O, B, a);
121. C = computedC;
122. psiRad = Math.PI \* psi / 180.0;
123. A.X = C.X + b \* Math.Cos(psiRad);
124. A.Y = C.Y + b \* Math.Sin(psiRad);
125. }
126. // Оновлюємо попередні значення, якщо все коректно
127. if (O.DistanceTo(B) <= 2 \* a && !SegmentsIntersect(O.X, O.Y, B.X, B.Y, C.X, C.Y, A.X, A.Y))
128. {
129. oldS = S;
130. oldPsi = psi;
131. }
132. }
133. // Метод для обчислення точки C
134. private Point2D ComputeC(Point2D O, Point2D B, double a)
135. {
136. double D = O.DistanceTo(B);
137. if (D > 2 \* a) return null;
138. double Mx = (O.X + B.X) / 2.0;
139. double My = (O.Y + B.Y) / 2.0;
140. double h = Math.Sqrt(a \* a - (D \* D / 4.0));
141. double By = B.Y - O.Y;
142. double Bx = B.X - O.X;
143. double len = Point2D.Length(Bx, By);
144. if (len < 1e-12) return null;
145. double ux = -By / len;
146. double uy = Bx / len;
147. double C1x = Mx + h \* ux;
148. double C1y = My + h \* uy;
149. double C2x = Mx - h \* ux;
150. double C2y = My - h \* uy;
151. double Cx, Cy;
152. // Вибір правильного положення точки C
153. if (C1y < 0 && C2y < 0)
154. {
155. Cx = C2x; Cy = C2y;
156. }
157. else if (C1y < 0)
158. {
159. Cx = C1x; Cy = C1y;
160. }
161. else
162. {
163. Cx = C2x; Cy = C2y;
164. }
165. return new Point2D(Cx, Cy);
166. }
167. // Метод для перевірки перетину двох сегментів
168. private bool SegmentsIntersect(double x1, double y1, double x2, double y2,
169. double x3, double y3, double x4, double y4)
170. {
171. double denom = (y4 - y3) \* (x2 - x1) - (x4 - x3) \* (y2 - y1);
172. if (Math.Abs(denom) < 1e-12) return false;
173. double ua = ((x4 - x3) \* (y1 - y3) - (y4 - y3) \* (x1 - x3)) / denom;
174. double ub = ((x2 - x1) \* (y1 - y3) - (y2 - y1) \* (x1 - x3)) / denom;
175. return (ua >= 0 && ua <= 1 && ub >= 0 && ub <= 1);
176. }
177. }
178. // Частковий клас для відображення та рендерингу моделі маніпулятора
179. public partial class RenderControl : OpenGL
180. {
181. // Об'єкт робочої руки
182. private RobotArm robotArm;
183. // Параметри для обертання та масштабування сцени
184. private double ay = 0;
185. private double ax = 0;
186. private double m = 1.0;
187. private bool MoveAxes = false;
188. private int dx;
189. private int dy;
190. // Квадричний об'єкт для малювання сфер
191. private IntPtr quadric;
192. // Параметри освітлення
193. float[] light\_pos = { 1.0f, 1.0f, 1.0f, 0.0f };
194. float[] floorPlane = { 0.0f, 1.0f, 0.0f, 1.0f };
195. // Конструктор контролу рендерингу
196. public RenderControl()
197. {
198. InitializeComponent();
199. // Ініціалізуємо робочу руку з заданими параметрами
200. robotArm = new RobotArm(0.46, 0.82, 0.6);
201. // Підписуємося на події миші та клавіатури
202. MouseDown += Mouse\_Down;
203. MouseUp += Mouse\_Up;
204. MouseMove += Mouse\_Move;
205. PreviewKeyDown += Key\_Down;
206. // Створюємо таймер для регулярного оновлення сцени
207. var timer = new Timer();
208. timer.Interval = 16; // Приблизно 60 FPS
209. timer.Tick += OnRender;
210. timer.Start();
211. }
212. // Обробник події створення контролу
213. protected override void OnHandleCreated(EventArgs e)
214. {
215. base.OnHandleCreated(e);
216. quadric = gluNewQuadric(); // Створюємо квадричний об'єкт
217. }
218. // Обробник події знищення контролу
219. protected override void OnHandleDestroyed(EventArgs e)
220. {
221. gluDeleteQuadric(quadric); // Видаляємо квадричний об'єкт
222. quadric = IntPtr.Zero;
223. base.OnHandleDestroyed(e);
224. }
225. // Метод для рендерингу сцени
226. private void OnRender(object sender, EventArgs e)
227. {
228. // Оновлюємо положення маніпулятора
229. robotArm.UpdateArm();
230. glEnable(GL\_DEPTH\_TEST); // Увімкнути тест глибини
231. glClearColor(0.9f, 0.9f, 0.95f, 1f); // Встановлюємо колір фону
232. glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT | GL\_STENCIL\_BUFFER\_BIT); // Очищуємо буфери
233. glViewport(0, 0, ClientSize.Width, ClientSize.Height); // Встановлюємо область перегляду
234. // Налаштування перспективної проекції
235. glMatrixMode(GL\_PROJECTION);
236. glLoadIdentity();
237. double aspect = (double)ClientSize.Width / (double)ClientSize.Height;
238. gluPerspective(45.0, aspect, 0.1, 20.0);
239. // Налаштування моделі перегляду
240. glMatrixMode(GL\_MODELVIEW);
241. glLoadIdentity();
242. gluLookAt(0.0, 0.0, 5.0, // Позиція камери
243. 0.0, 0.0, 0.0, // Точка, на яку дивиться камера
244. 0.0, 1.0, 0.0); // Напрямок "вгору"
245. // Обертання та масштабування сцени
246. glRotated(ax, 1, 0, 0);
247. glRotated(ay, 0, 1, 0);
248. glScaled(m, m, m);
249. // Налаштування освітлення
250. SetupLighting();
251. // Малюємо підлогу
252. DrawFloor();
253. // Малюємо маніпулятор з освітленням
254. DrawManipulator(true);
255. // Малюємо осі координат і текстові позначки
256. glDisable(GL\_LIGHTING);
257. Axes(1.7);
258. glEnable(GL\_LIGHTING);
259. // Малюємо тінь маніпулятора
260. DrawShadow();
261. glFlush(); // Завершуємо рендеринг
262. }
263. // Метод для налаштування освітлення сцени
264. private void SetupLighting()
265. {
266. glEnable(GL\_LIGHTING); // Увімкнути освітлення
267. glEnable(GL\_LIGHT0); // Увімкнути перше джерело світла
268. glLightModeli(GL\_LIGHT\_MODEL\_TWO\_SIDE, 1); // Двостороннє освітлення
269. // Встановлюємо компоненти освітлення
270. float[] lightAmbient = { 0.2f, 0.2f, 0.2f, 1f };
271. float[] lightDiffuse = { 0.8f, 0.8f, 0.8f, 1f };
272. float[] lightSpecular = { 1.0f, 1.0f, 1.0f, 1f };
273. glLightfv(GL\_LIGHT0, GL\_POSITION, light\_pos);
274. glLightfv(GL\_LIGHT0, GL\_AMBIENT, lightAmbient);
275. glLightfv(GL\_LIGHT0, GL\_DIFFUSE, lightDiffuse);
276. glLightfv(GL\_LIGHT0, GL\_SPECULAR, lightSpecular);
277. glEnable(GL\_NORMALIZE); // Нормалізувати нормалі
278. glShadeModel(GL\_SMOOTH); // Гладке затінення
279. }
280. // Метод для малювання підлоги
281. private void DrawFloor()
282. {
283. glDisable(GL\_LIGHTING); // Вимкнути освітлення для підлоги
284. glColor3f(0.7f, 0.7f, 0.7f); // Встановлюємо колір підлоги
285. double size = 5.0; // Розмір підлоги
286. double y = -1.0; // Висота підлоги
287. glBegin(GL\_QUADS);
288. glNormal3d(0, 1, 0); // Нормаль до поверхні
289. glVertex3d(-size, y, -size);
290. glVertex3d(size, y, -size);
291. glVertex3d(size, y, size);
292. glVertex3d(-size, y, size);
293. glEnd();
294. glEnable(GL\_LIGHTING); // Знову увімкнути освітлення
295. }
296. // Метод для малювання маніпулятора
297. private void DrawManipulator(bool withLighting)
298. {
299. // Отримуємо поточні точки маніпулятора
300. Point2D O = robotArm.GetO();
301. Point2D B = robotArm.GetB();
302. Point2D C = robotArm.GetC();
303. Point2D A = robotArm.GetA();
304. // Встановлюємо матеріал для вузлів, якщо потрібне освітлення
305. if (withLighting)
306. SetMaterial(1.0, 0.5, 0.0, 1.0);
307. // Малюємо бази маніпулятора
308. DrawCube(O.X, O.Y, 0, 0.1, 1.0, 0.5, 0.0, 1.0);
309. DrawCube(B.X, B.Y, 0, 0.1, 1.0, 0.5, 0.0, 1.0);
310. // Малюємо ланки маніпулятора
311. DrawCuboidSegment(O.X, O.Y, C.X, C.Y, 1.0, 0.0, 0.0, 1.0);
312. DrawCuboidSegment(C.X, C.Y, B.X, B.Y, 0.0, 1.0, 0.0, 1.0);
313. DrawCuboidSegment(C.X, C.Y, A.X, A.Y, 0.0, 0.0, 1.0, 1.0);
314. // Малюємо сферу в точці C
315. DrawSphere(C.X, C.Y, 0, 0.07, 1.0, 1.0, 0.0, 1.0);
316. }
317. // Метод для малювання тіні маніпулятора
318. private void DrawShadow()
319. {
320. float[] shadowMat = new float[16];
321. ShadowMatrix(shadowMat, floorPlane, light\_pos);
322. glPushMatrix();
323. // Налаштування для малювання тіні
324. glDisable(GL\_LIGHTING); // Вимкнути освітлення
325. glColor3f(0.0f, 0.0f, 0.0f); // Колір тіні
326. glEnable(GL\_POLYGON\_OFFSET\_FILL); // Увімкнути зсув полігонів
327. glPolygonOffset(-1.0f, -1.0f); // Налаштування зсуву
328. glMultMatrixf(shadowMat); // Застосувати матрицю тіні
329. // Малюємо тільки маніпулятор без осей і тексту
330. DrawManipulator(false);
331. // Відновлюємо попередні налаштування
332. glDisable(GL\_POLYGON\_OFFSET\_FILL);
333. glEnable(GL\_LIGHTING);
334. glPopMatrix();
335. }
336. // Метод для створення матриці тіні
337. private void ShadowMatrix(float[] mat, float[] plane, float[] light)
338. {
339. float dot = plane[0] \* light[0] + plane[1] \* light[1] + plane[2] \* light[2] + plane[3] \* light[3];
340. mat[0] = dot - light[0] \* plane[0];
341. mat[4] = -light[0] \* plane[1];
342. mat[8] = -light[0] \* plane[2];
343. mat[12] = -light[0] \* plane[3];
344. mat[1] = -light[1] \* plane[0];
345. mat[5] = dot - light[1] \* plane[1];
346. mat[9] = -light[1] \* plane[2];
347. mat[13] = -light[1] \* plane[3];
348. mat[2] = -light[2] \* plane[0];
349. mat[6] = -light[2] \* plane[1];
350. mat[10] = dot - light[2] \* plane[2];
351. mat[14] = -light[2] \* plane[3];
352. mat[3] = -light[3] \* plane[0];
353. mat[7] = -light[3] \* plane[1];
354. mat[11] = -light[3] \* plane[2];
355. mat[15] = dot - light[3] \* plane[3];
356. }
357. // Метод для встановлення матеріалу об'єктів
358. private void SetMaterial(double r, double g, double b, double a)
359. {
360. float[] mat\_ambient = { (float)(r \* 0.2f), (float)(g \* 0.2f), (float)(b \* 0.2f), (float)a };
361. float[] mat\_diffuse = { (float)r, (float)g, (float)b, (float)a };
362. float[] mat\_specular = { 1.0f, 1.0f, 1.0f, (float)a };
363. float[] mat\_shininess = { 50.0f };
364. glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_AMBIENT, mat\_ambient);
365. glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, mat\_diffuse);
366. glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_SPECULAR, mat\_specular);
367. glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_SHININESS, mat\_shininess);
368. }
369. // Метод для малювання сегмента кубоїда між двома точками
370. private void DrawCuboidSegment(double x1, double y1, double x2, double y2, double r, double g, double b, double alpha)
371. {
372. double dx = x2 - x1;
373. double dy = y2 - y1;
374. double length = Math.Sqrt(dx \* dx + dy \* dy);
375. double angle = Math.Atan2(dy, dx) \* 180.0 / Math.PI;
376. double thickness = 0.05;
377. glPushMatrix();
378. glTranslated(x1, y1, 0); // Переміщення до початкової точки
379. glRotated(angle, 0, 0, 1); // Обертання сегмента
380. SetMaterial(r, g, b, alpha); // Встановлення матеріалу
381. double half = thickness / 2.0;
382. glBegin(GL\_QUADS);
383. // Верхня грань
384. glNormal3d(0, 1, 0);
385. glVertex3d(0, half, half);
386. glVertex3d(length, half, half);
387. glVertex3d(length, half, -half);
388. glVertex3d(0, half, -half);
389. // Нижня грань
390. glNormal3d(0, -1, 0);
391. glVertex3d(0, -half, -half);
392. glVertex3d(length, -half, -half);
393. glVertex3d(length, -half, half);
394. glVertex3d(0, -half, half);
395. // Передня грань
396. glNormal3d(0, 0, 1);
397. glVertex3d(0, -half, half);
398. glVertex3d(length, -half, half);
399. glVertex3d(length, half, half);
400. glVertex3d(0, half, half);
401. // Задня грань
402. glNormal3d(0, 0, -1);
403. glVertex3d(0, half, -half);
404. glVertex3d(length, half, -half);
405. glVertex3d(length, -half, -half);
406. glVertex3d(0, -half, -half);
407. // Права грань
408. glNormal3d(1, 0, 0);
409. glVertex3d(length, -half, half);
410. glVertex3d(length, -half, -half);
411. glVertex3d(length, half, -half);
412. glVertex3d(length, half, half);
413. // Ліва грань
414. glNormal3d(-1, 0, 0);
415. glVertex3d(0, half, half);
416. glVertex3d(0, half, -half);
417. glVertex3d(0, -half, -half);
418. glVertex3d(0, -half, half);
419. glEnd();
420. glPopMatrix();
421. }
422. // Метод для малювання куба
423. private void DrawCube(double x, double y, double z, double size, double r, double g, double b, double alpha)
424. {
425. double half = size / 2.0;
426. glPushMatrix();
427. glTranslated(x, y, z); // Переміщення до позиції куба
428. SetMaterial(r, g, b, alpha); // Встановлення матеріалу
429. glBegin(GL\_QUADS);
430. // Передня грань
431. glNormal3d(0, 0, 1);
432. glVertex3d(-half, -half, half);
433. glVertex3d(half, -half, half);
434. glVertex3d(half, half, half);
435. glVertex3d(-half, half, half);
436. // Задня грань
437. glNormal3d(0, 0, -1);
438. glVertex3d(-half, half, -half);
439. glVertex3d(half, half, -half);
440. glVertex3d(half, -half, -half);
441. glVertex3d(-half, -half, -half);
442. // Ліва грань
443. glNormal3d(-1, 0, 0);
444. glVertex3d(-half, -half, -half);
445. glVertex3d(-half, -half, half);
446. glVertex3d(-half, half, half);
447. glVertex3d(-half, half, -half);
448. // Права грань
449. glNormal3d(1, 0, 0);
450. glVertex3d(half, -half, half);
451. glVertex3d(half, -half, -half);
452. glVertex3d(half, half, -half);
453. glVertex3d(half, half, half);
454. // Верхня грань
455. glNormal3d(0, 1, 0);
456. glVertex3d(-half, half, half);
457. glVertex3d(half, half, half);
458. glVertex3d(half, half, -half);
459. glVertex3d(-half, half, -half);
460. // Нижня грань
461. glNormal3d(0, -1, 0);
462. glVertex3d(-half, -half, -half);
463. glVertex3d(half, -half, -half);
464. glVertex3d(half, -half, half);
465. glVertex3d(-half, -half, half);
466. glEnd();
467. glPopMatrix();
468. }
469. // Метод для малювання сфери
470. private void DrawSphere(double x, double y, double z, double radius, double r, double g, double b, double alpha)
471. {
472. glPushMatrix();
473. glTranslated(x, y, z); // Переміщення до позиції сфери
474. SetMaterial(r, g, b, alpha); // Встановлення матеріалу
475. gluSphere(quadric, radius, 20, 20); // Малюємо сферу
476. glPopMatrix();
477. }
478. // Метод для малювання осей координат
479. private void Axes(double size)
480. {
481. glColor3d(1, 1, 1); // Встановлюємо колір осей
482. glBegin(GL\_LINES);
483. // Вісь X
484. glVertex3d(-size \* 0.1, 0.0, 0.0); glVertex3d(size, 0.0, 0.0);
485. // Вісь Y
486. glVertex3d(0.0, -size \* 0.1, 0.0); glVertex3d(0.0, size, 0.0);
487. // Вісь Z
488. glVertex3d(0.0, 0.0, -size \* 0.1); glVertex3d(0.0, 0.0, size);
489. glEnd();
490. // Додаємо текстові позначки до осей
491. DrawText("X", size \* 1.05, 0, 0);
492. DrawText("Y", 0, size \* 1.05, 0);
493. DrawText("Z", 0, 0, size \* 1.05);
494. }
495. // Обробник події натискання кнопки миші
496. private void Mouse\_Down(object sender, MouseEventArgs e)
497. {
498. MoveAxes = (e.Button == MouseButtons.Left); // Визначаємо, чи рухаються осі
499. dx = e.X; dy = e.Y; // Зберігаємо початкові координати миші
500. }
501. // Обробник події відпускання кнопки миші
502. private void Mouse\_Up(object sender, MouseEventArgs e)
503. {
504. MoveAxes = MoveAxes && (e.Button != MouseButtons.Left); // Зупиняємо рух осей, якщо кнопка відпущена
505. }
506. // Обробник події руху миші
507. private void Mouse\_Move(object sender, MouseEventArgs e)
508. {
509. if (MoveAxes)
510. {
511. ay += (e.X - dx) / 2.0; // Оновлюємо кут обертання по осі Y
512. ax += (e.Y - dy) / 2.0; // Оновлюємо кут обертання по осі X
513. dx = e.X; dy = e.Y; // Оновлюємо поточні координати миші
514. Invalidate(); // Запитуємо перерисовку сцени
515. }
516. }
517. // Обробник події натискання клавіш
518. private void Key\_Down(object sender, PreviewKeyDownEventArgs e)
519. {
520. switch (e.KeyCode)
521. {
522. case Keys.PageUp:
523. robotArm.SetS(robotArm.GetS() + 0.01); // Збільшуємо параметр S
524. break;
525. case Keys.PageDown:
526. robotArm.SetS(robotArm.GetS() - 0.01); // Зменшуємо параметр S
527. break;
528. case Keys.Left:
529. robotArm.SetPsi(robotArm.GetPsi() - 1); // Зменшуємо кут psi
530. break;
531. case Keys.Right:
532. robotArm.SetPsi(robotArm.GetPsi() + 1); // Збільшуємо кут psi
533. break;
534. }
535. Invalidate(); // Запитуємо перерисовку сцени
536. }
537. }
538. }