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  *633п*

напряму підготовки (спеціальності):

*121 інженерія програмного забезпечення*

(шифр і назва напряму підготовки / спеціальності)

*Силенок Ю. 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 4](#_Toc186028033)

[Завдання, варіант № 19 4](#_Toc186028034)

[Системна інформація 4](#_Toc186028035)

[Теоретичні відомості 4](#_Toc186028036)

[Результати виконання практичної роботи 6](#_Toc186028040)

[Практична робота 2. Графічні примітиви OpenGL 9](#_Toc186028043)

[Завдання, варіант № 19 9](#_Toc186028044)

[Результати виконання практичної роботи 9](#_Toc186028045)

[Практична робота 3. Графічні функції однієї змінної 15](#_Toc186028048)

[Завдання, варіант № 19 15](#_Toc186028049)

[Теоретичні відомості 15](#_Toc186028050)

[Результати виконання практичної роботи 16](#_Toc186028052)

[Практична робота 4. Криві другого порядку 19](#_Toc186028055)

[Завдання, варіант № 19 19](#_Toc186028056)

[Результати виконання практичної роботи 19](#_Toc186028057)

[Практична робота 5. Квадратичні примітиви. Афінні перетворення у просторі 24](#_Toc186028060)

[Завдання, варіант № 19 24](#_Toc186028061)

[Результати виконання практичної роботи 24](#_Toc186028062)

[Практична робота 6. Візуалізація прямої задачі кінематики 29](#_Toc186028065)

[Завдання, варіант № 19 29](#_Toc186028066)

[Результати виконання практичної роботи 29](#_Toc186028067)

[Практична робота 7. Екранна заставка з анімацією 37](#_Toc186028070)

[Завдання, варіант № 19 37](#_Toc186028071)

[Результати виконання практичної роботи 37](#_Toc186028072)

[Загальний перелік посилань 43](#_Toc186028075)

[Додаток А. Лістинг програми до практичної роботи №1 44](#_Toc186028076)

[Додаток Б. Лістинг програми до практичної роботи №2 47](#_Toc186028079)

[Додаток В. Лістинг програми до практичної роботи №3 52](#_Toc186028085)

[Додаток Г. Лістинг програми до практичної роботи №4 59](#_Toc186028091)

[Додаток Ґ. Лістинг програми до практичної роботи №5 65](#_Toc186028098)

[Додаток Д. Лістинг програми до практичної роботи №6 72](#_Toc186028100)

[Додаток Е Лістинг програми до практичної роботи №7 77](#_Toc186028103)

# Практична робота 1. Основні принципи роботи з OpenGL

## Завдання, варіант № 19

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.

|  |  |  |
| --- | --- | --- |
| 19 | Примітиви:  GL\_POINTS, GL\_LINES  x1 = -1.5; x2 = 7.5  y1 = -0.5; y2 = 3.5 |  |

## Системна інформація

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

Processor AMD Ryzen 5 5600H with Radeon Graphics 3.30 GHz

RAM 8.0 GB (5.86 GB usable)

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

Edition Windows 10 Home Version 22H2

IDE Microsoft Visual Studio Community 2022 (64-bit) version 17.11.2

## Теоретичні відомості

### Вершинні масиви

Використання вершинних масивів зменшує кількість викликів функцій та надмірне використання спільних вершин. Таким чином, ви можете збільшити продуктивність рендерингу.

OpenGL надає функції glEnableClientState() та glDisableClientState() для активації та деактивації 6 різних типів масивів. Крім того, є 6 функцій для визначення точних позицій (адрес) масивів, отже, OpenGL може отримати доступ до масивів у вашому додатку.

glVertexPointer(): вказати покажчик на масив вершинних координат

glNormalPointer(): вказати покажчик на звичайний масив

glColorPointer(): вказати вказівник на масив кольорів RGB

glIndexPointer(): вказати покажчик на індексований кольоровий масив

glTexCoordPointer(): вказати покажчик на масив текстурних шнурів

glEdgeFlagPointer(): вказати вказівник на масив прапорців edge

Для кожної заданої функції потрібні різні параметри. Прапорці ребер використовуються для позначення того, чи знаходиться вершина на граничному ребрі чи ні. Отже, єдині ребра, де ввімкнено прапорці країв, будуть видимими, якщо для glPolygonMode() встановлено значення GL\_LINE.

Для кожної заданої функції потрібні різні параметри. Будь ласка, перегляньте інструкції до API OpenGL. Прапорці ребер використовуються для позначення того, чи знаходиться вершина на граничному ребрі чи ні. Отже, єдині ребра, де ввімкнено прапорці країв, будуть видимими, якщо для glPolygonMode() встановлено значення GL\_LINE.

### Команда glDrawArrays()

glDrawArrays() зчитує дані вершин з увімкнених масивів, проходячи прямо по масиву без пропусків або стрибків. Оскільки glDrawArrays() не дозволяє стрибати навколо масивів вершин, вам все одно доведеться повторювати спільні вершини один раз на кожну грань.

glDrawArrays() приймає 3 аргументи. По-перше, це примітивний тип. Другий параметр – це початковий виліт масиву. Останній параметр – це кількість вершин, які потрібно передати конвеєру рендерингу OpenGL.

Для наведеного вище прикладу для малювання куба першим параметром є GL\_TRIANGLES, другим - 0, що означає початок роботи масиву. І останній параметр - 36: у куба 6 сторін і кожній стороні потрібно 6 вершин, щоб намалювати 2 трикутника, 6 × 6 = 36.

GLfloat vertices[] = {...}; // 36 of vertex coords

...

// activate and specify pointer to vertex array

glEnableClientState(GL\_VERTEX\_ARRAY);

glVertexPointer(3, GL\_FLOAT, 0, vertices);

// draw a cube

glDrawArrays(GL\_TRIANGLES, 0, 36);

// deactivate vertex arrays after drawing

glDisableClientState(GL\_VERTEX\_ARRAY);

В результаті використання glDrawArrays() ви можете замінити 36 викликів glVertex\*() одним викликом glDrawArrays(). Однак нам все ще потрібно дублювати спільні вершини, тому кількість вершин, визначених у масиві, все ще становить 36 замість 8. glDrawElements() — це рішення для зменшення кількості вершин у масиві, тому воно дозволяє передавати менше даних до OpenGL.

### Команда glDrawElements()

glDrawElements() малює послідовність примітивів, перескакуючи навколо вершинних масивів з пов'язаними індексами масивів. При цьому зменшується як кількість викликів функцій, так і кількість вершин для передачі. Крім того, OpenGL може кешувати нещодавно оброблені вершини та повторно використовувати їх без повторного надсилання тих самих вершин у конвеєр перетворення вершин кілька разів.

## Результати виконання практичної роботи

### Розв'язання завдання

Для управління параметрами графічних примітивів було використано наступні команди (Додаток А):

колір, glColor3d() рядок 15 у файлі Figure.cs;

тип, glLineStipple(), glEnable()/glDisable(), рядок 73, 74, 93 у файлі Figure.cs;

товщина glLineWidth(), рядках 7, 47, 76 у файлі Figure.cs.

Коректне відображення завдання під час змінення розмірів/положення вікна наведено у рис. 1.1 та 1.2

Застосування циклів для створення зображень наведено у рядках 79 – 90 файлу Figure.cs.

Використання ООП реалізовано за допомогою розроблення власних класів, які наведено у файлах Figure.cs Додатку А.

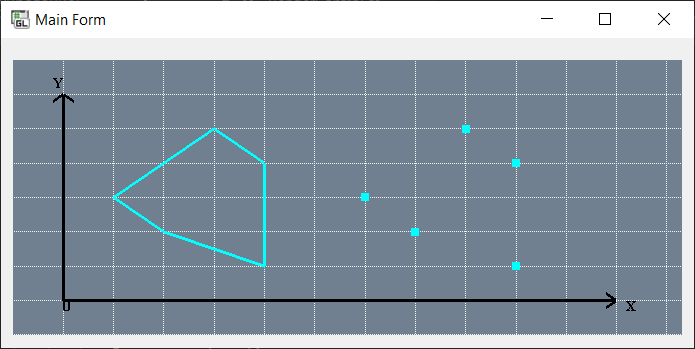


Рисунок 1.1 – Тестування програми при зміні ширини вікна

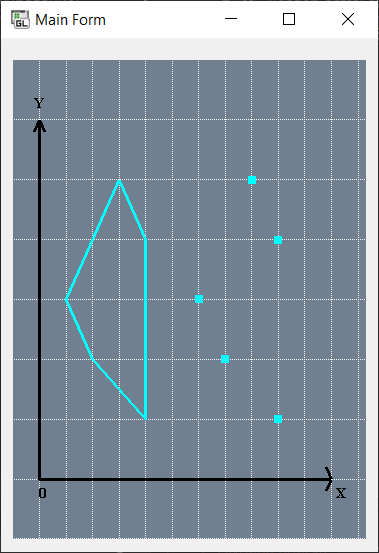


Рисунок 1.2 – Тестування програми при зміні висоти вікна

### Контроль виконання вимог та елементів завдання

В результаті виконання практичної роботи були частково виконані елементи базового рівня та 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 | **+** |

# Практична робота 2. ГРАФІЧНІ ПРИМІТИВИ oPENGL

## Завдання, варіант № 19

Використовуючи інструментальні засоби, що вказані викладачем, і беручи до уваги вимоги 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. Після старту застосунок повинен відображати у робочій області одну плитку (*tile*).

Таблиця 2.1 – Варіаційна частина варіанту 19

|  |  |  |
| --- | --- | --- |
| № | Параметри | Фігура |
| 19 | Сторона фігури a = 0.15  Примітив(и):   GL\_TRIANGLE\_FAN, GL\_POLYGON |  |

## Результати виконання практичної роботи

### Розв'язання завдання

Під час запуску застосунку програми зображення відповідає варіанту завдання з однією плиткою це показано рисунку 2.1.

Багаторазове замощення плиткою та кратність замощення задається користувачем під час роботи застосунку показано на рисунку 2.2

Коректне відображення завдання під час зміни як розмірів/положення вікна, так і параметрів замощення показано на рисунках 2.3-2.6

Організовано взаємодію з користувачем за допомогою маніпулятора миші/клавіатури – взаємодія з інтерфейсом, а саме з елементами керування numericUpDown та radioButton.

Застосування мінімальної (у рамках варіанту) кількості графічних примітивів для виконання завдання – Додаток Б рядки 33 у файлі HexagonRenderer.cs та рядок 19 у файлі TriangleRenderer.cs.

Використання ООП (розробка власних класів):

* RenderSettings – клас посередник між MainForm та RenderControl. MainForm передає налаштування елементів керування в об’єкт RenderSettings, звідки, через метод SetSettings класу посередника, RenderControl отримує відповідні налаштування.
* HexagonRenderer – клас, що виконує малювання шестикутників на основі налаштувань з RenderSettings, має виклик класу TriangleRenderer для малювання внутрішніх трикутників.
* TriangleRenderer – клас, що малює трикутники відповідно до координат, переданих з HexagonRenderer.

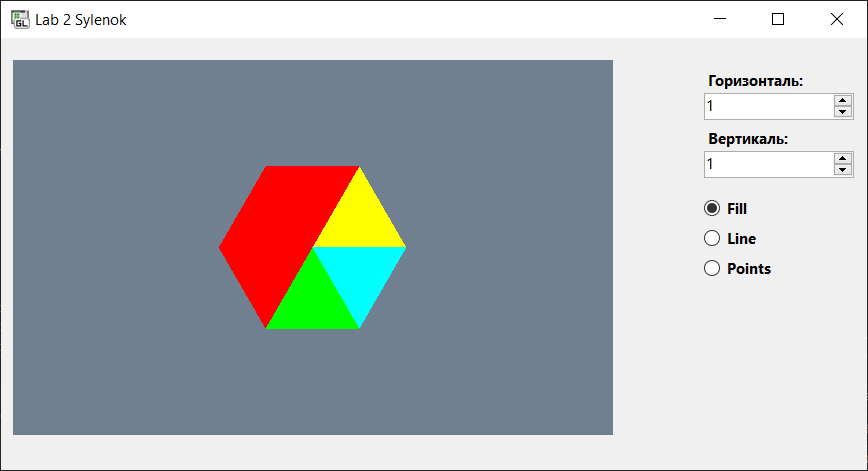


Рисунок 2.1 – Відображення при запуску програми

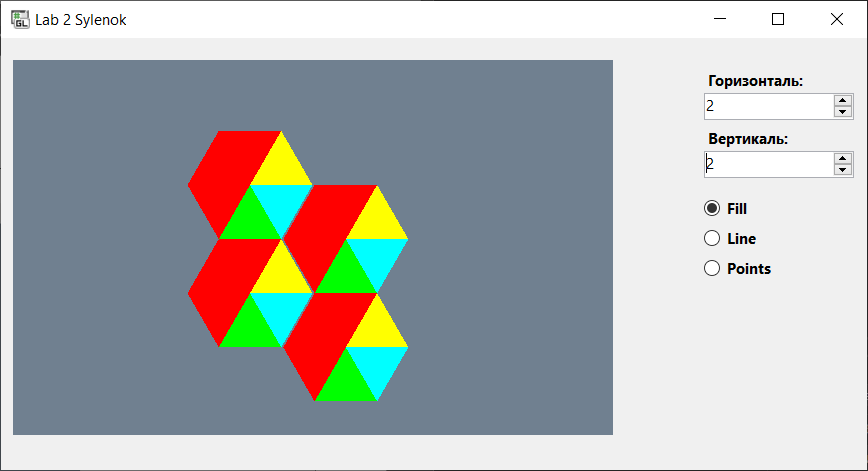


Рисунок 2.2 – Багаторазове замощення плитки та тестування інтерфейсу користувача

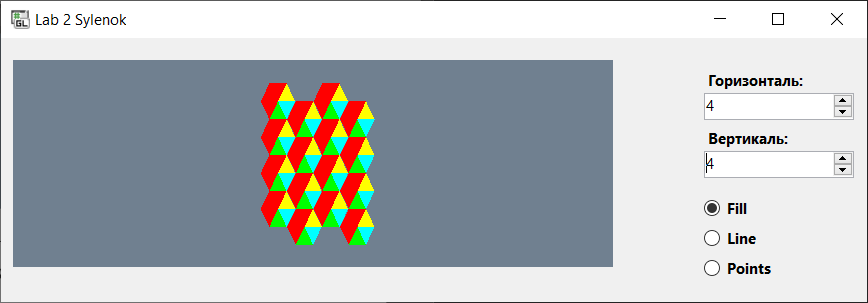


Рисунок 2.3 – Тестування багаторазового замощення вікна при зміні розмірів вікна по горизонталі

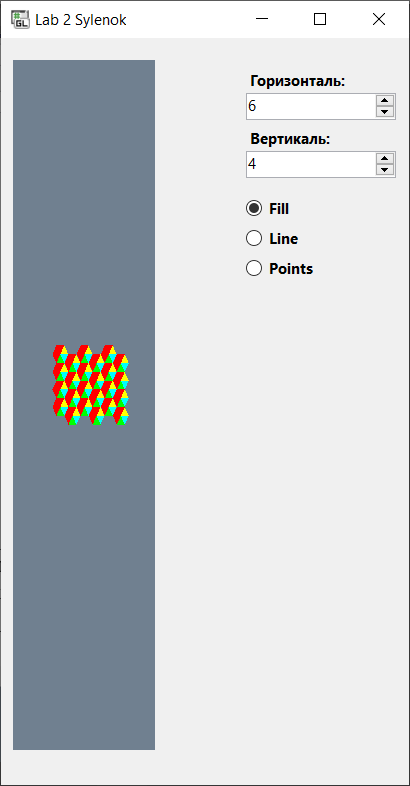


Рисунок 2.4 – Тестування багаторазового замощення вікна при зміні розмірів вікна по вертикалі

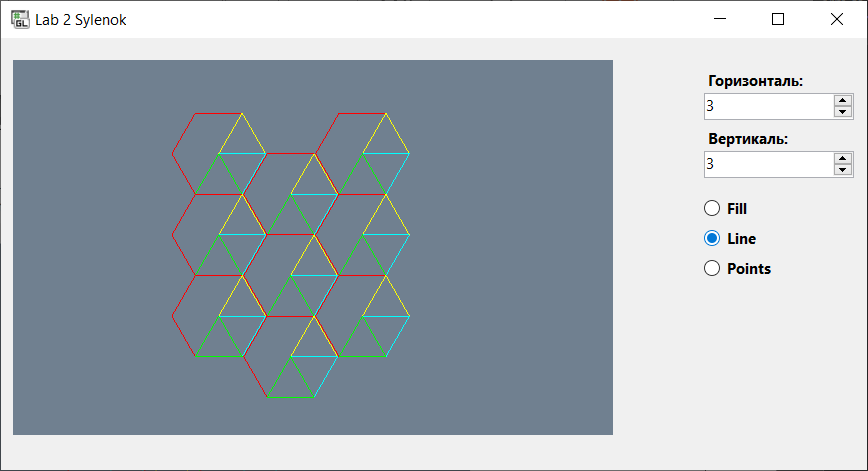


Рисунок 2.5 – Тестування зміни типу відображення на Line

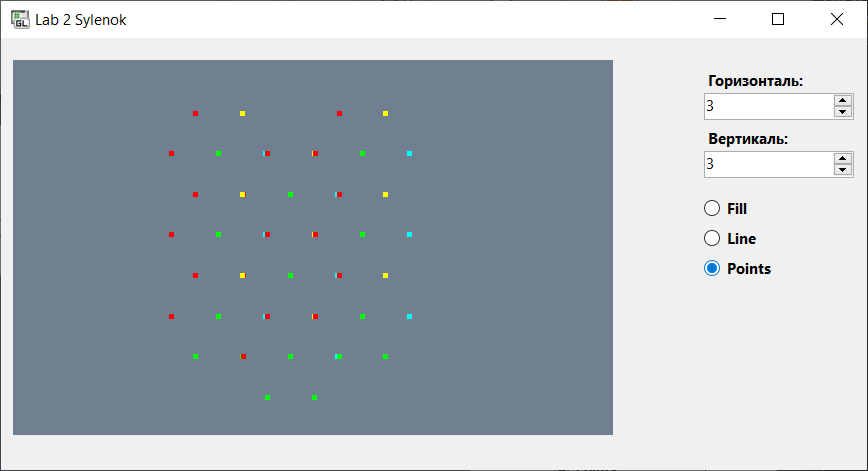


Рисунок 2.6 – Тестування зміни типу відображення на Points

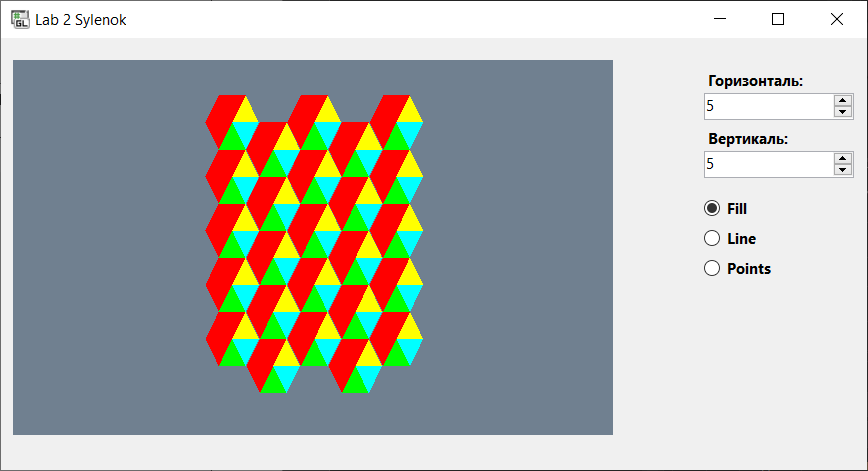


Рисунок 2.7 – Тестування зміни типу відображення на Fill

### Контроль виконання вимог та елементів завдання

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

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

# Практична робота 3. ГРАФІЧНІ ФУНКЦІЇ ОДНІЄЇ ЗМІННОЇ

## Завдання, варіант № 19

Використовуючи інструментальні засоби, що вказані викладачем, розробити програму для побудови графіка функції виду y=f(x) 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 fx=0, якщо вони є на завданому інтервалі від ***Xmin*** до ***Xmax***.

**Функція f1(x) для варіанту 19:**

## Теоретичні відомості

### Метод половинного ділення

Метод половинного ділення – це простий і ефективний числовий метод для знаходження коренів рівнянь з одним невідомим. У цій роботі метод був використаний для обчислення точок перетину графіка функції f(x) із віссю OX.

Основний принцип методу базується на теоремі проміжного значення для неперервних функцій, яка стверджує, що якщо функція f(x) змінює знак на інтервалі [,], тобто f()⋅f()<0, то на цьому інтервалі існує принаймні одна точка x0, для якої f(x0)=0.

Алгоритм методу половинного ділення:

1. Знаходимо два початкові значення і , такі що f()⋅f()<0.
2. Визначаємо середню точку .
3. Якщо f()=0, то є коренем. Інакше:
   * Якщо f(t)⋅f()<0, то корінь знаходиться між і , і замінюється на .
   * Якщо f(t)⋅f()<0, то корінь знаходиться між і , і замінюється на
4. Процес повторюється до тих пір, поки довжина інтервалу не стане меншою за задану точність.

## Результати виконання практичної роботи

### Розв'язання завдання

Осі координат і графік функції ***f1(x)*** виводяться на заданому користувачем інтервалі від *Xmin* до *Xmax*, де *Ymin* до *Ymax* автоматично обчислюється – рисуноки 3.1 та 3.2, Додаток В рядок 5 – функція CalculateYBounds у файлі FunctionRenderer.cs.

Обчислення і виведення на екран точок *f1(x)* = 0 – рисунок 3.1 та 3.2, рядок 88 у файлі RenderControl.cs.

Використання ООП, а саме:

* наслідування, клас Function1Renderer наслідує клас FunctionRenderer у Додатку В;
* використання віртуальних методів, рядок 5 у файлі FunctionRenderer.cs у Додатку В;
* абстрактних методів, рядок 3 у файлі FunctionRenderer.cs у Додатку В.

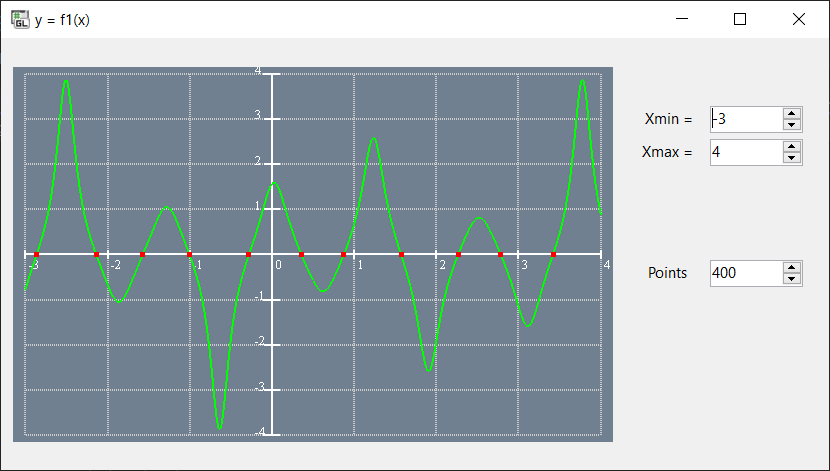


Рисунок 3.1 – Тестування програми на автоматичну обчислення Ymin та Ymax та виведення точок *f1(x)* = 0 № 1

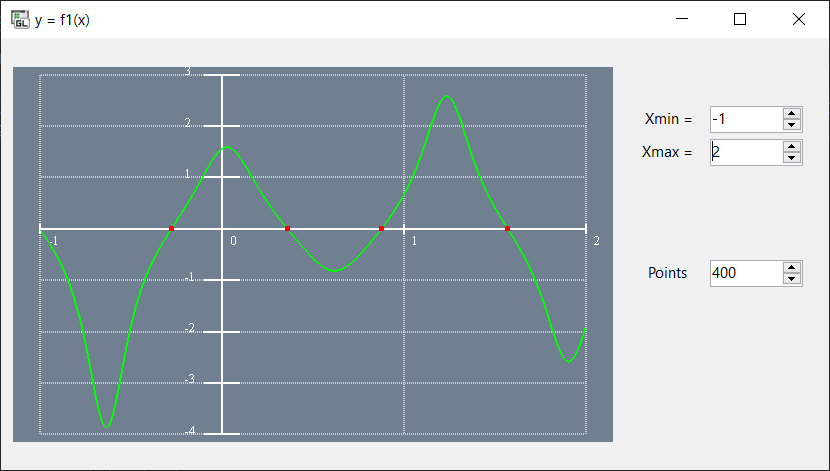


Рисунок 3.1 – Тестування програми на автоматичну обчислення Ymin та Ymax та виведення точок *f1(x)* = 0 № 2

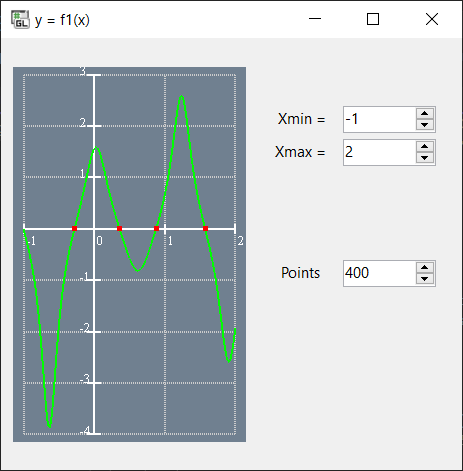


Рисунок 3.3 – Тестування анізотропної системи координат при зміні висоти

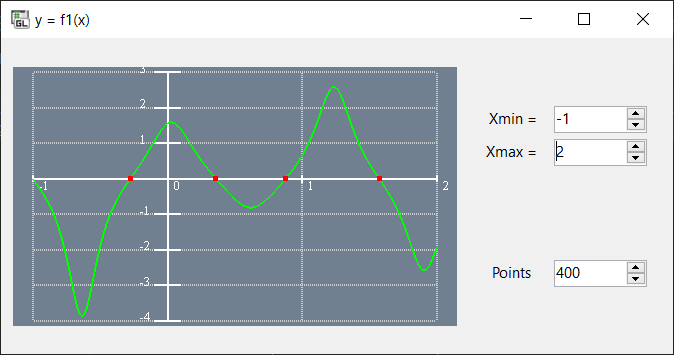


Рисунок 3.4 – Тестування анізотропної системи координат при зміні ширини

### Контроль виконання вимог та елементів завдання

В результаті виконання практичної роботи були повністю виконані елементи базового рівня та частково 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 | **+** |

# Практична робота 4. КРИВІ ДРУГОГО ПОРЯДКУ

## Завдання, варіант № 19

Використовуючи інструментальні засоби, вказані викладачем, розробити програму для виведення кривих другого порядку на екран (у вікно Windows) за допомогою відрізків. Для кривих, які у варіанті відмічені «++», знайти та вивести на екран точки перетину, якщо такі є, з довільним відрізком, координати якого задає користувач.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Таблиця 4.1 – Варіант 19 | | | | | | | | |
| № | Окружність | | Еліпс | | Гіпербола | | Парабола | |
| Подання | | Подання | | Подання | | Подання | |
| явне | парамет­ричне | явне | парамет­ричне | явне | парамет­ричне | явне | парамет­ричне |
| 19 |  |  |  | ++ |  |  | + |  |

## Результати виконання практичної роботи

### Розв'язання завдання

Установлення ізотропної системи координат для вікна з змінюваними розмірами показано на рисунках 4.1 та 4.2.

Виведення кривих другого порядку відповідно до варіанту 19 показано на рисунках 4.3 та 4.4, де парабола представлена явно (клас Parabola у файлі Parabola.cs Додатка Г), а еліпс – параметрично (клас Ellips у файлі Ellips.cs Додатка Г).

Виведення відрізка та обчислення його точок перетину з кривою другого порядку відповідно до варіанту 19 показано на рисунках 4.5 та 4.6 (виведення відрізка – рядок 11 у файлі LineSegment.cs; обчислення точок перетину – рядок 20 у файлі LineSegment.cs Додатка Г).

Вказання положення точок, що формують відрізок, в області графічного виведення програми за допомогою маніпулятора «миш» – функція OnMouseClick (рядок 116) у файлі RenderControl.cs Додатка Г.

Використання ООП. Було розроблено класи Ellips, LineSegment та Parabola, що малюють відповідні фігури.

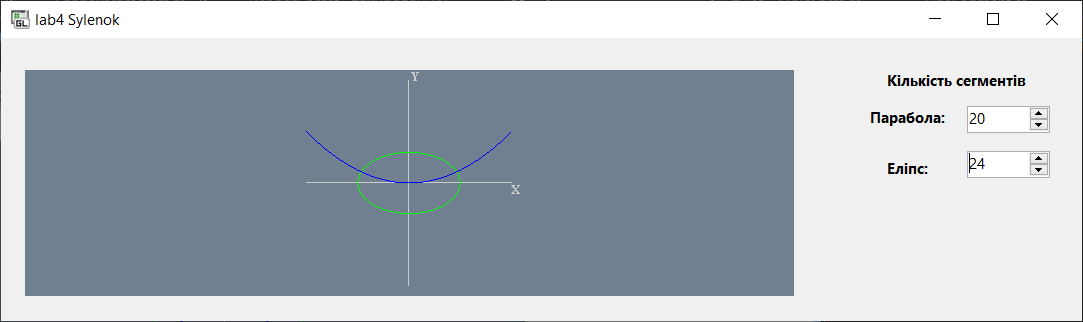


Рисунок 4.1 – Тестування системи координат, зміна по ширині

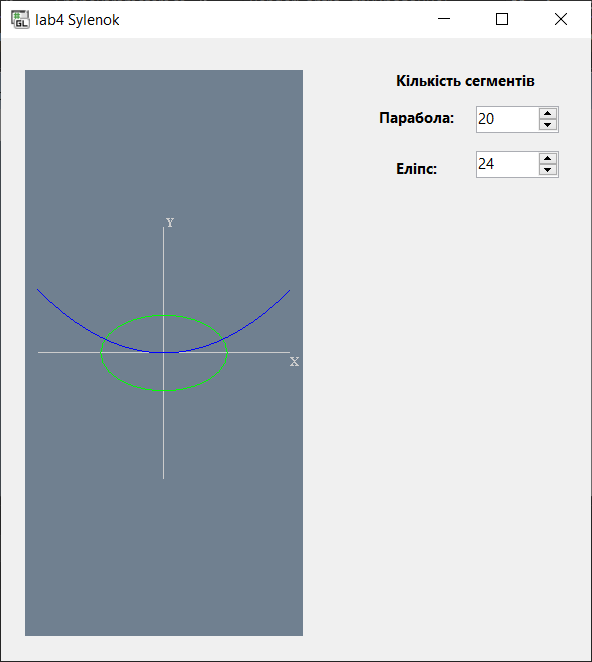


Рисунок 4.2 – Тестування системи координат, зміна по вертикалі

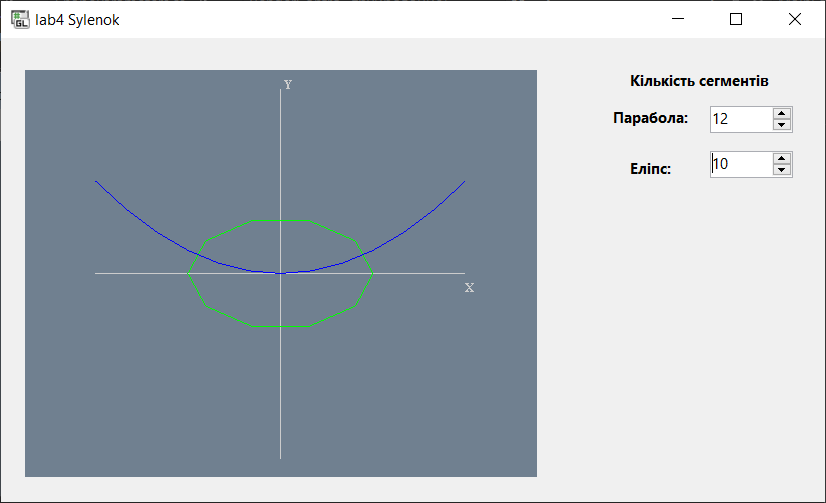


Рисунок 4.3 – Тестування програми при зменшенні кількості сегментів, що утворюють фігури

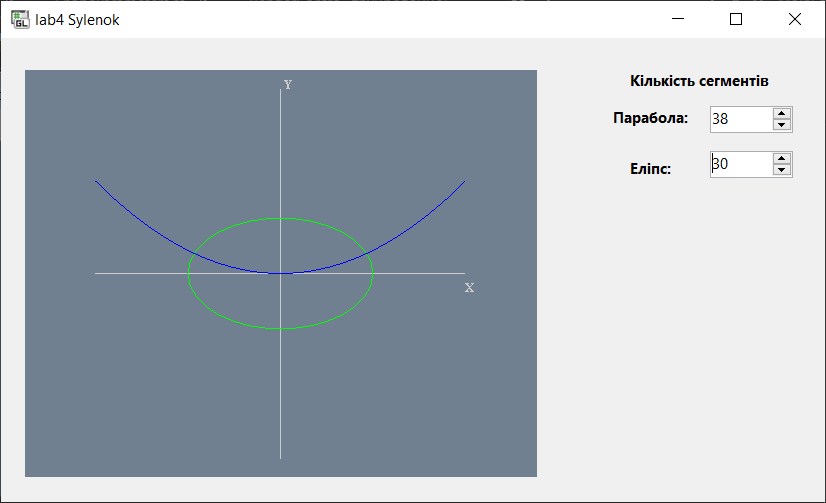


Рисунок 4.4 – Тестування програми при збільшенні кількості сегментів, що утворюють фігури



Рисунок 4.5 – Тестування виведення відрізка, положення якого вказав користувач за допомогою маніпулятора миші (немає перетину з параметричною фігурою еліпса)

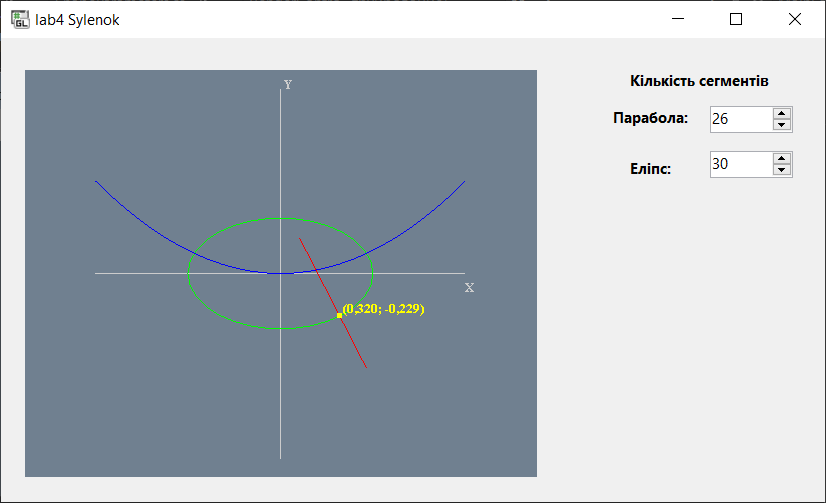


Рисунок 4.6 – Тестування виведення відрізка, положення якого вказав користувач за допомогою маніпулятора миші (має перетин з параметричною фігурою еліпса у точці (0,320; 0,229))

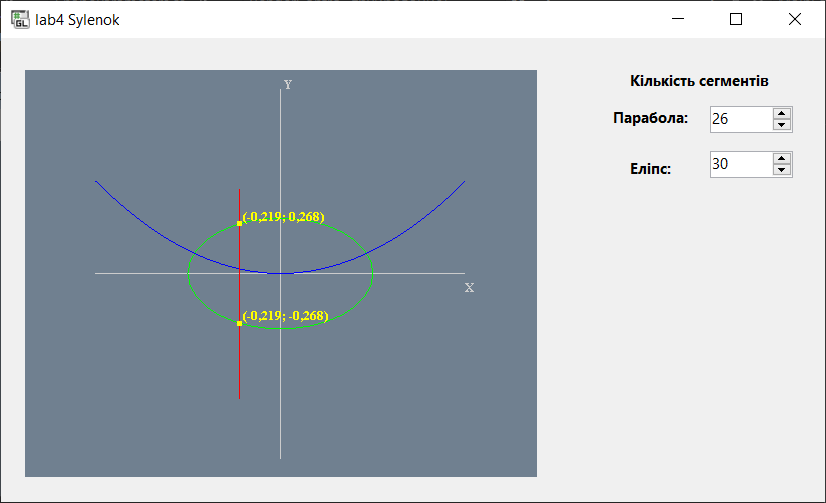


Рисунок 4.7 – Тестування виведення відрізка, положення якого вказав користувач за допомогою маніпулятора миші (має перетин з параметричною фігурою еліпса у точках (-0,219; 0,268) та (-0,219; -0,268))

### Контроль виконання вимог та елементів завдання

В результаті виконання практичної роботи були повністю виконані елементи базового рівня та підвищеного рівня складності, що відображено в таблиці 4.2.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблиця 4.2 | | | | |
| № з/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 | **+** |

# Практична робота 5. КВАДРАТИЧНІ ПРИМІТИВИ. АФІННІ ПЕРЕТВОРЕННЯ У ПРОСТОРІ

## Завдання, варіант № 19

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:

− 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.

**Варіант 19**

Таблиця 5.1 – Варіаційна частина для варіанту 19

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № | *Grid* | Фігура | Параметри квадратичних фігур | | | | | | | | |
| ось | *x0* | *y0* | *z0* | *R* | *r* | *H* | ∠start | ∠sweep |
| 2 | X0Z | сфера | ∥  0Y | +2.5 | +1.0 | -3.5 | 1.5 | - | - | - | - |
| циліндр | ⇅ 0X | -2.5 | -3.5 | -4.5 | 2.5 | - | 1.5 | - | - |
| частинний диск | ∥  0Z | -3.5 | +3.0 | +3.0 | 3.5 | 1.0 | - | 180° | 135° |

## Результати виконання практичної роботи

### Розв'язання завдання

Установлення ізотропної системи координат для вікна з змінюваними розмірами показано на рисунках 5.1 та 5.2.

Під час запуску застосунку відображаються осі ОХ, ОY, OZ, координатна сітка і каркас квадратичних об’єктів (рисунок 5.3).

Інтерфейс керування параметрами площини відтину (рисунки 5.1-5.6).

Використання джерел світла для освітлення об’єктів сцени сумісно з командою glColorMaterial – рядок 117 (функція EnableLighting) в файлі RenderControl.cs Додатка Ґ.

Створення зображення сцени в перспективній проекції – рядок 66 у файлі RenderControl.cs Додатка Ґ.

Застосування команди glMaterial для налаштування параметрів відбиття поверхонь об’єктів сцени – рядок 94 (функція SetMaterial) у файлі RenderControl.cs Додатка Ґ.

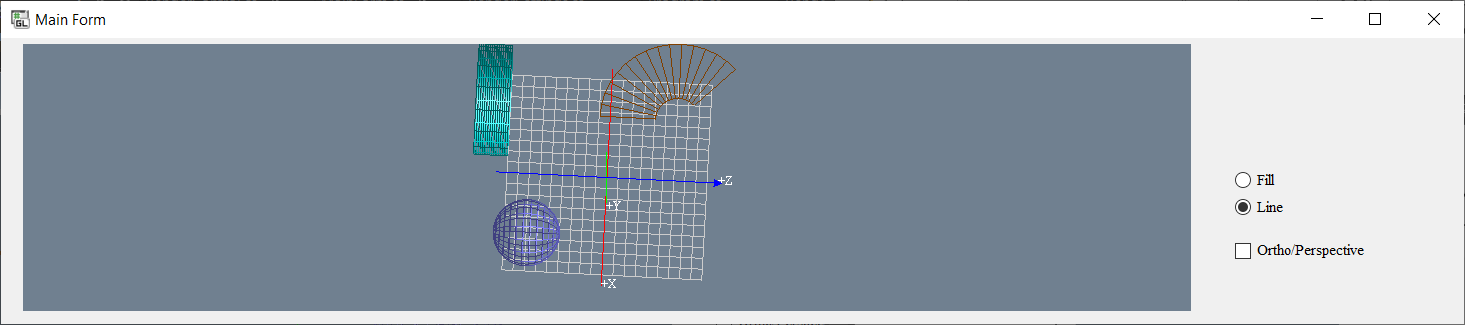


Рисунок 5.1 – Тестування системи координат, зміна по ширині

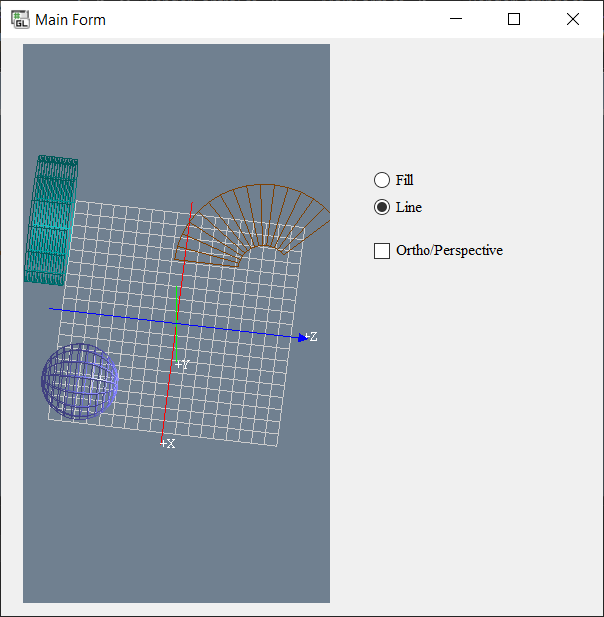


Рисунок 5.2 – Тестування системи координат, зміна по вертикалі

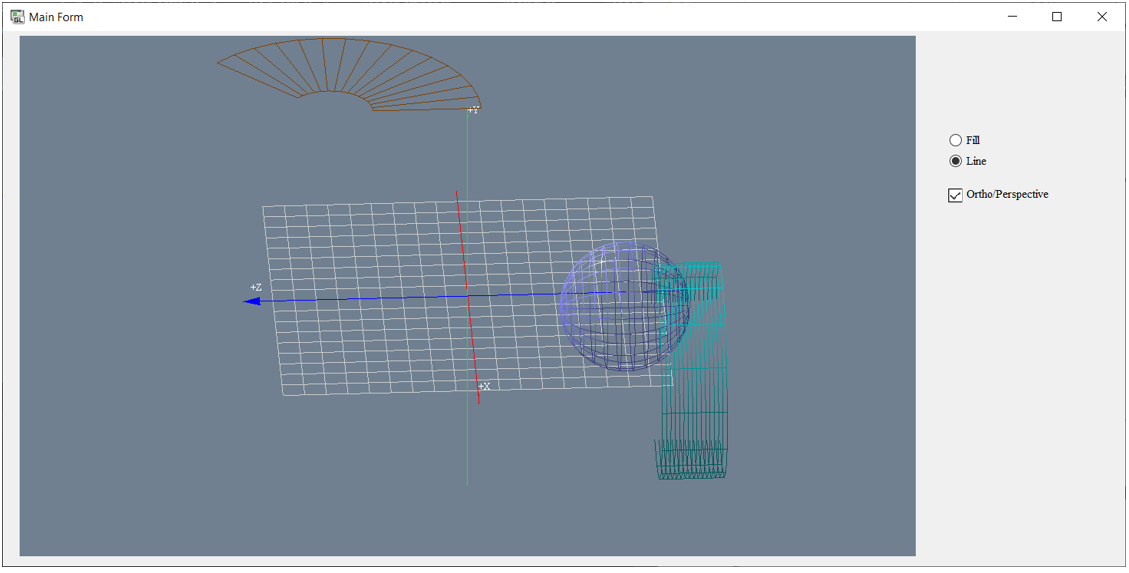


Рисунок 5.3 – Тестування відображення у режимі Ortho та Line

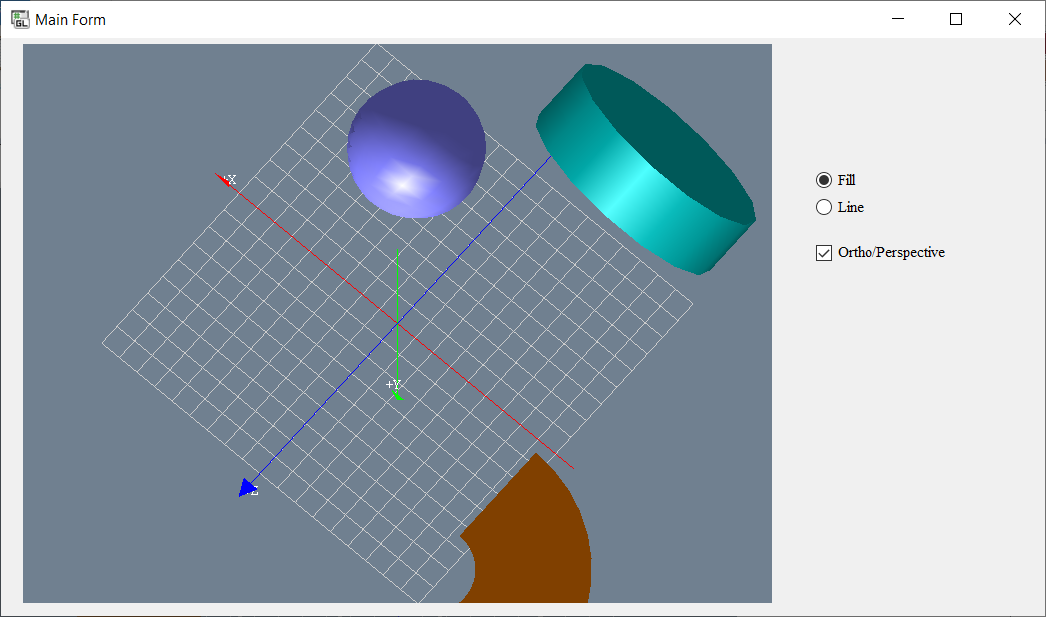


Рисунок 5.4 – Тестування відображення у режимі Ortho та Fill

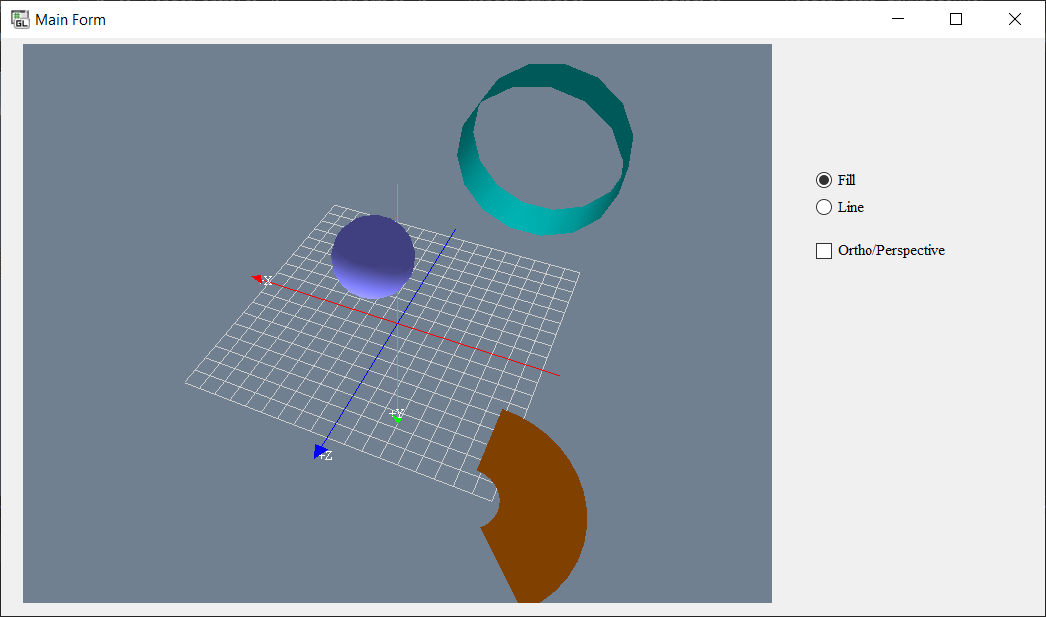


Рисунок 5.5 – Тестування відображення у режимі Perspective та Fill

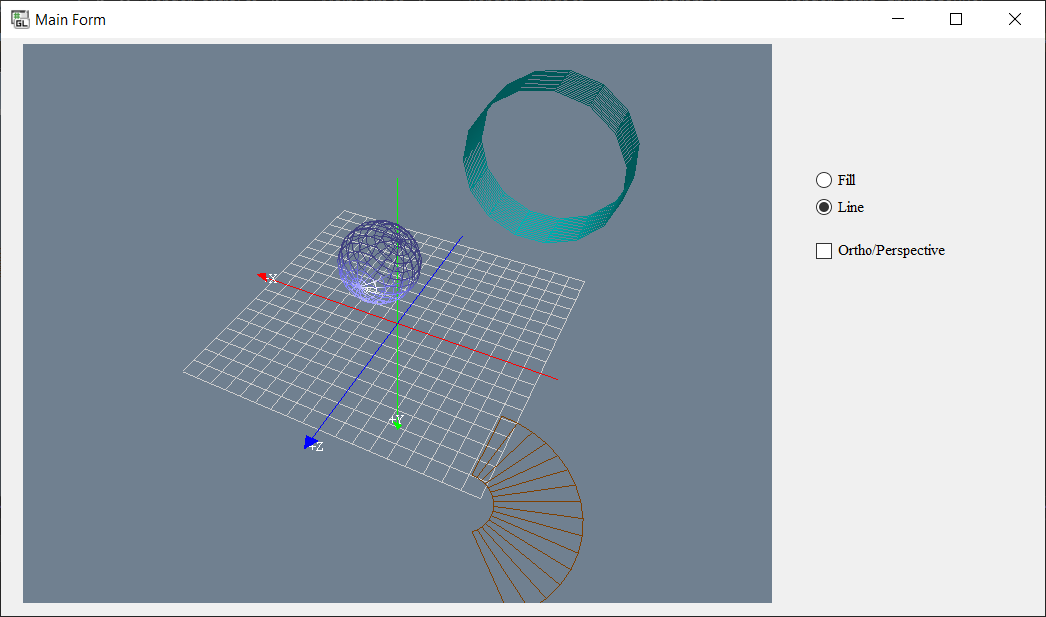


Рисунок 5.6 – Тестування відображення у режимі Perspective та Line

### Контроль виконання вимог та елементів завдання

В результаті виконання практичної роботи були частково виконані елементи базового рівня та підвищеного рівня складності, що відображено в таблиці 5.2.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблиця 5.2 | | | | |
| № з/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 | **+** |

# Практична робота 6. Візуалізація ПРЯМОЇ ЗАДАЧІ КІНЕМАТИКИ

## Завдання, варіант № 19

Використовуючи інструментальні засоби, запропоновані викладачем, створити додаток для виведення на екран моделі маніпулятора за даною кінематичною схемою. Для управління моделлю та точкою спостереження необхідно використовувати клавіатуру та/або маніпулятор «миша», за допомогою яких змінюють значення параметрів, які відповідають руху, наприклад кути *ϕ* , *θ* , *ψ* , відстань *S*).

Таблиця 6.1 – Варіаційна частина для варіанту 19

|  |  |  |
| --- | --- | --- |
| Номер варіанта | Параметри | Кінематична модель |
| 19 | а = 0.34  b = 0.9 | v19-00%20copy |

## Результати виконання практичної роботи

### Розв'язання завдання

Реалізовано програму двовимірної моделі маніпулятора відповідно до варіанту 19, яку було доопрацьовано до тривимірної моделі – можна переглянути на рисунках 6.1-6.11.

Управління моделлю та точкою спостереження реалізовано за допомоги маніпулятора «миша» та клавіатурою – рядки 92, 102, 108, 120, 126 файлу RenderControl.cs Додатка Д.

В програмі реалізовано освітлення моделі маніпулятора – рядок 166 файлу RenderControl.cs Додатка Д.

Розроблено власний клас Segment (файл Segment.cs Додатка Д), де реалізується малювання сегментів маніпулятора.

Використовується перспективна проекція для відображення моделі маніпулятора – рядок 150 файлу RenderControl.cs Додатка Д.

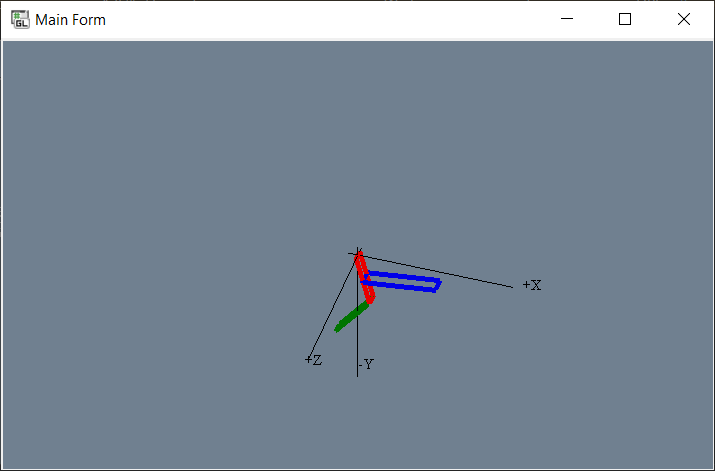


Рисунок 6.1 – Тривимірна модель маніпулятора

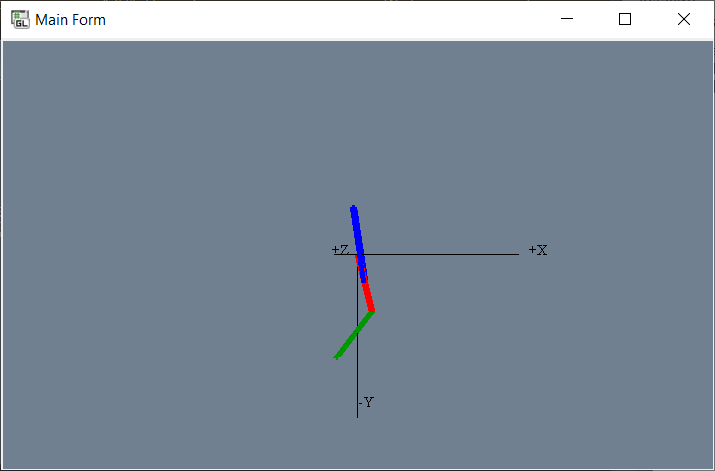


Рисунок 6.2 – Максимальне розгортання синього сигмента

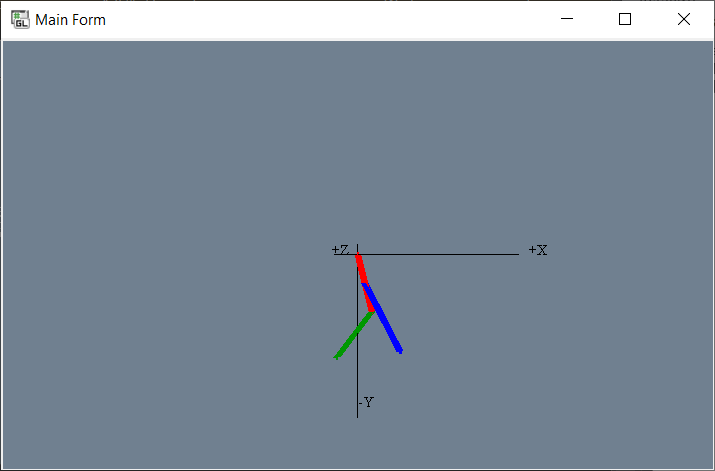


Рисунок 6.3 – Мінімальне згортання синього сегмента

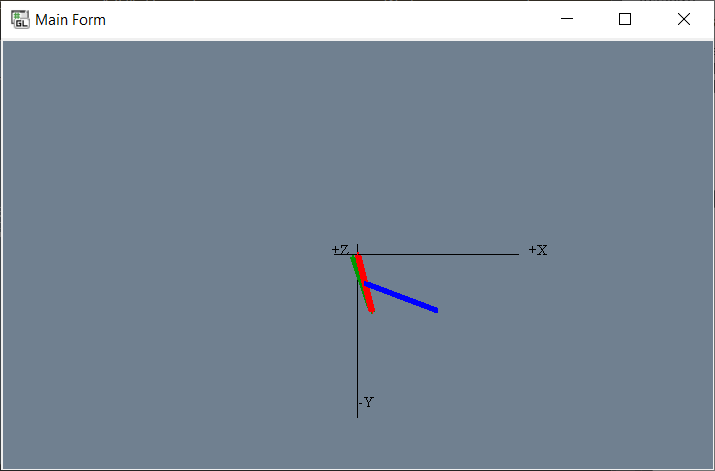


Рисунок 6.4 – Максимальне згортання зеленого сегмента

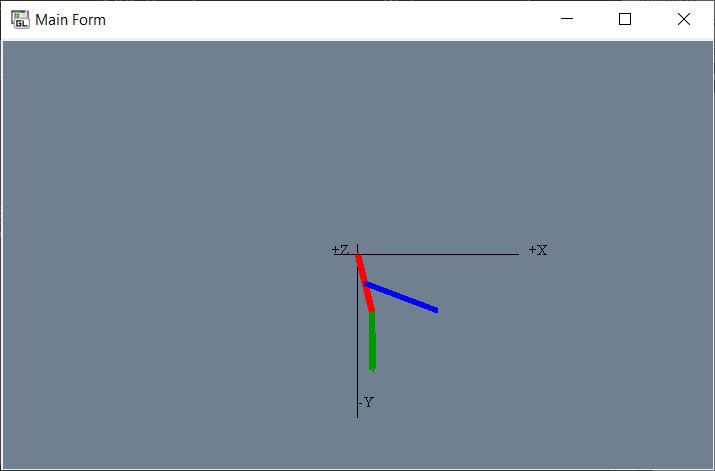


Рисунок 6.5 – Мінімальне розгортання зеленого сегмента

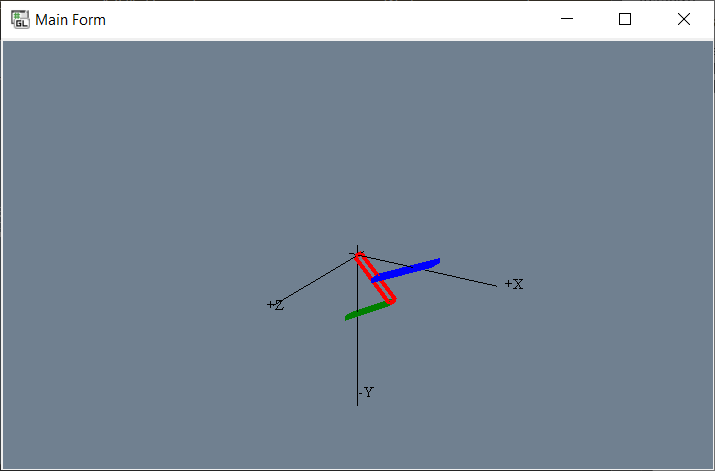


Рисунок 6.6 – Тестування управління моделлю за допомогою маніпулятора «миша» та клавіатури №1

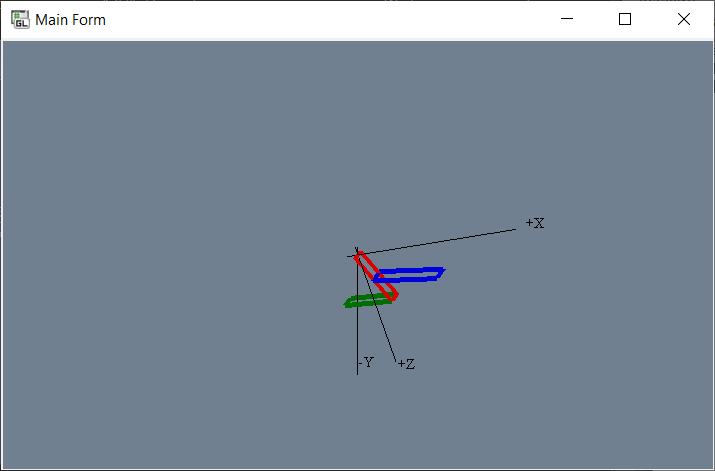


Рисунок 6.7 – Тестування управління моделлю за допомогою маніпулятора «миша» та клавіатури №2

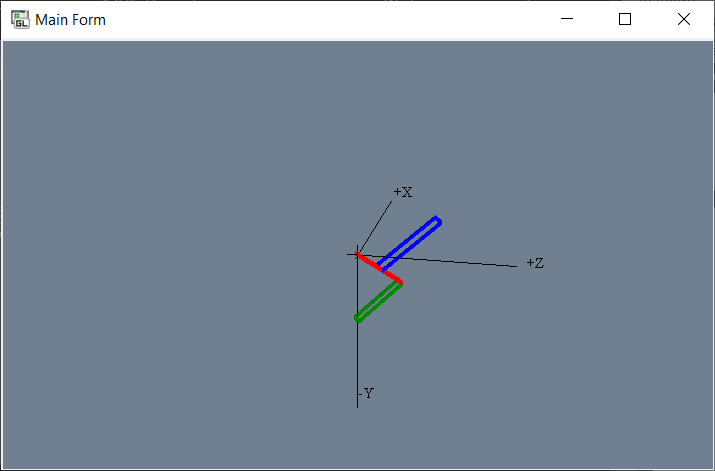


Рисунок 6.8 – Тестування управління моделлю за допомогою маніпулятора «миша» та клавіатури №3

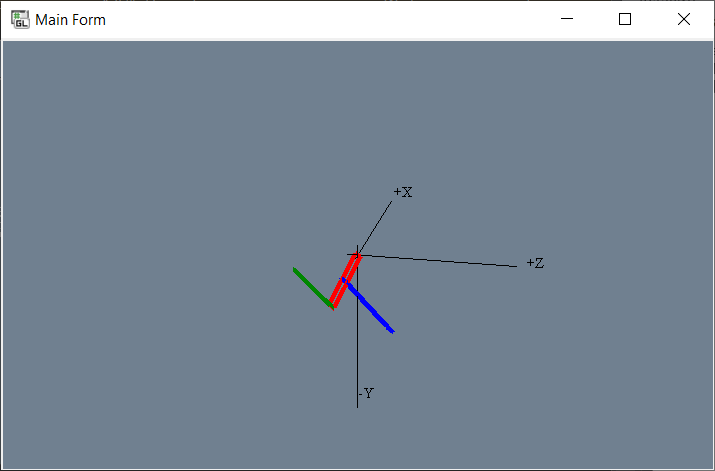


Рисунок 6.9 – Керування тілом маніпулятора (червоним сегментом) є не обмеженим

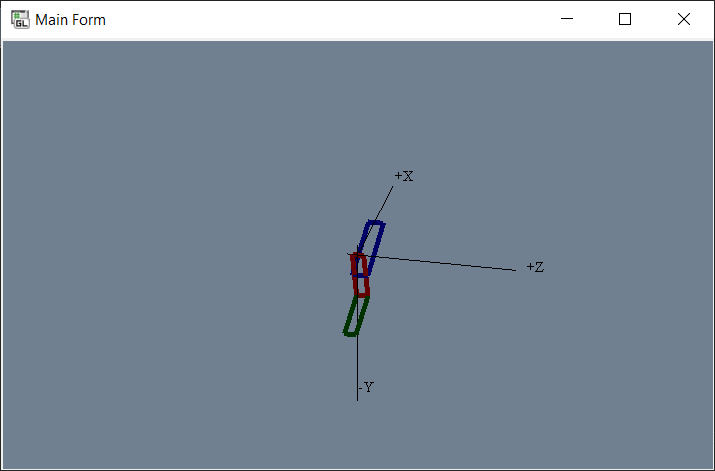


Рисунок 6.10 – Тестування відображення освіщення моделі маніпулятора №1

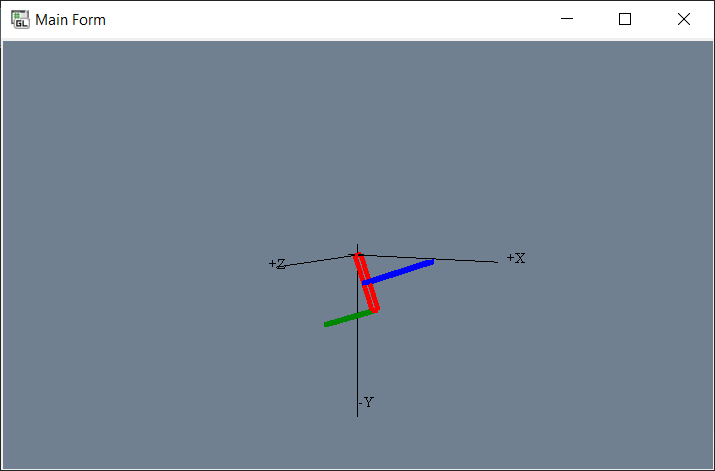


Рисунок 6.11 – Тестування відображення освіщення моделі маніпулятора №2

### Контроль виконання вимог та елементів завдання

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблиця 6.1 | | | | |
| № з/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 | +-  (лише оствілення**)** |
| 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 | **-** |

# Практична робота 7. ЕКРАННА ЗАСТАВКА З АНІМАЦІЄЮ

## Завдання, варіант № 19

Використовуючи засоби, указані викладачем, створити програму екранної заставки (ScreenSaver) з анімацією. Сюжет анімації обрати самостійно і погодити з викладачем.

Обраний сюжет анімації:

В процесі малювання система Лоренца масштабується (поступово збільшується до заданого максимального значення, при досяганні якого система поступово зменшується до мінімального) та паралельно обертається за X та Y координатами. Користувач може налаштувати такі параметри як максимальний масштаб анімації та швидкість обертання по ості Х.

## Результати виконання практичної роботи

### Розв'язання завдання

Прив’язка анімації до механізму простою операційної системи показано на рисунку 7.1.

Реалізовано повноекранний режим – команда «/s».

Реалізація налаштування програми Screen Saver (команда «/c») – файл SettingsForm.cs Додатка Е.

Реалізовано попередній перегляд (команда «/p»).

Збереження конфігурації і налаштувань програми ScreenSaver у реєстрі ОС можна переглянути на рисунках 7.6-7.7 та програмну реалізацію у файлі AppSettings.cs Додатка Е.

Код програмної реалізації анімації можна переглянути в файлі RenderControl.cs Додатка Е.

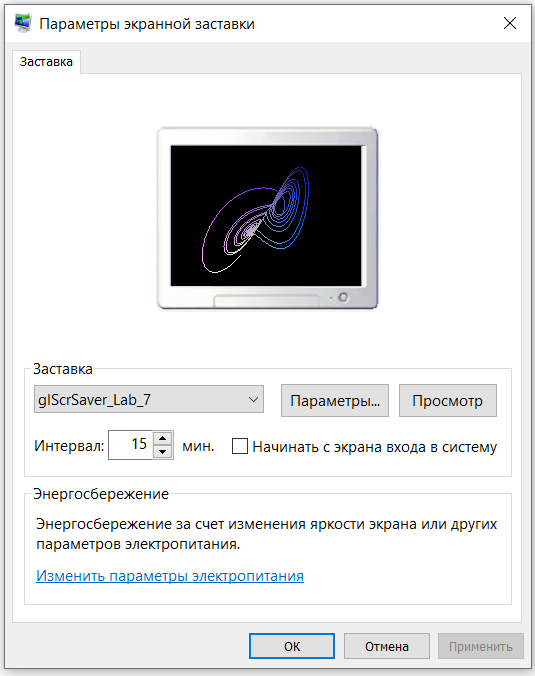


Рисунок 7.1 – Можливість прив’язки створеної анімації до механізму простою операційної системи

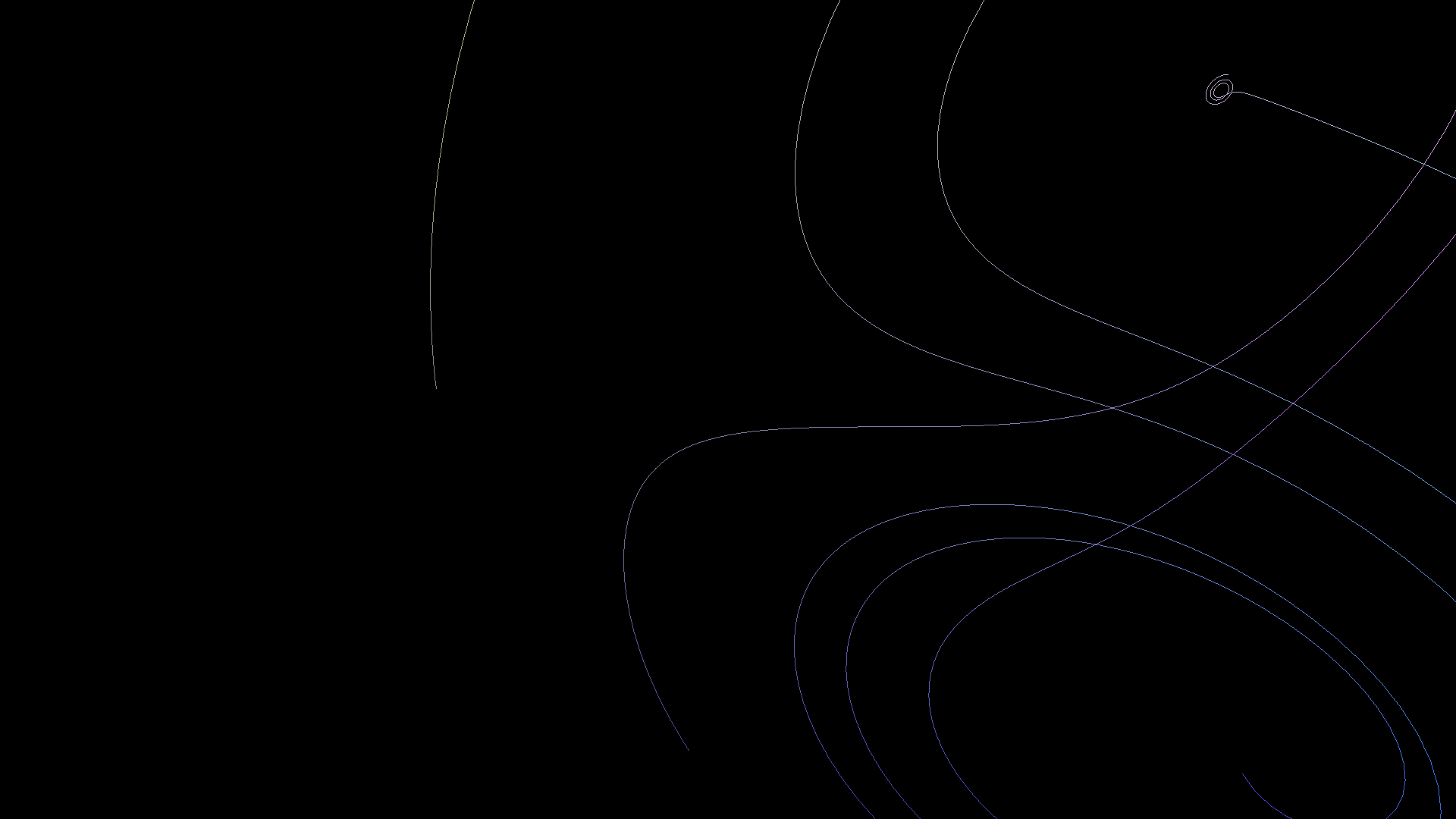


Рисунок 7.2 – Частина анімації, вигляд масштабування системи Лоренца та її обертання за віссю Х №1

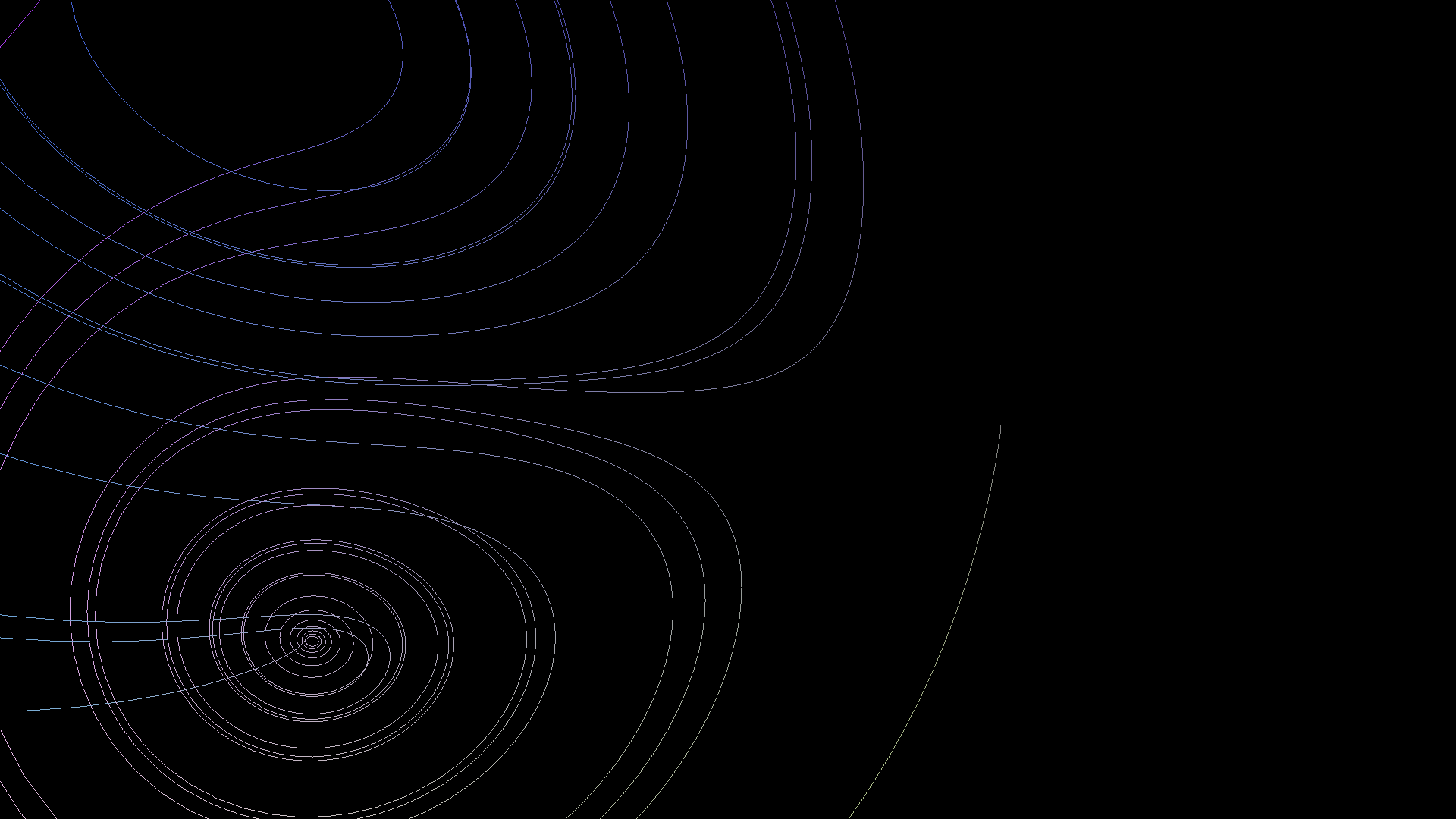


Рисунок 7.3 – Частина анімації, вигляд масштабування системи Лоренца та її обертання за віссю Х №2

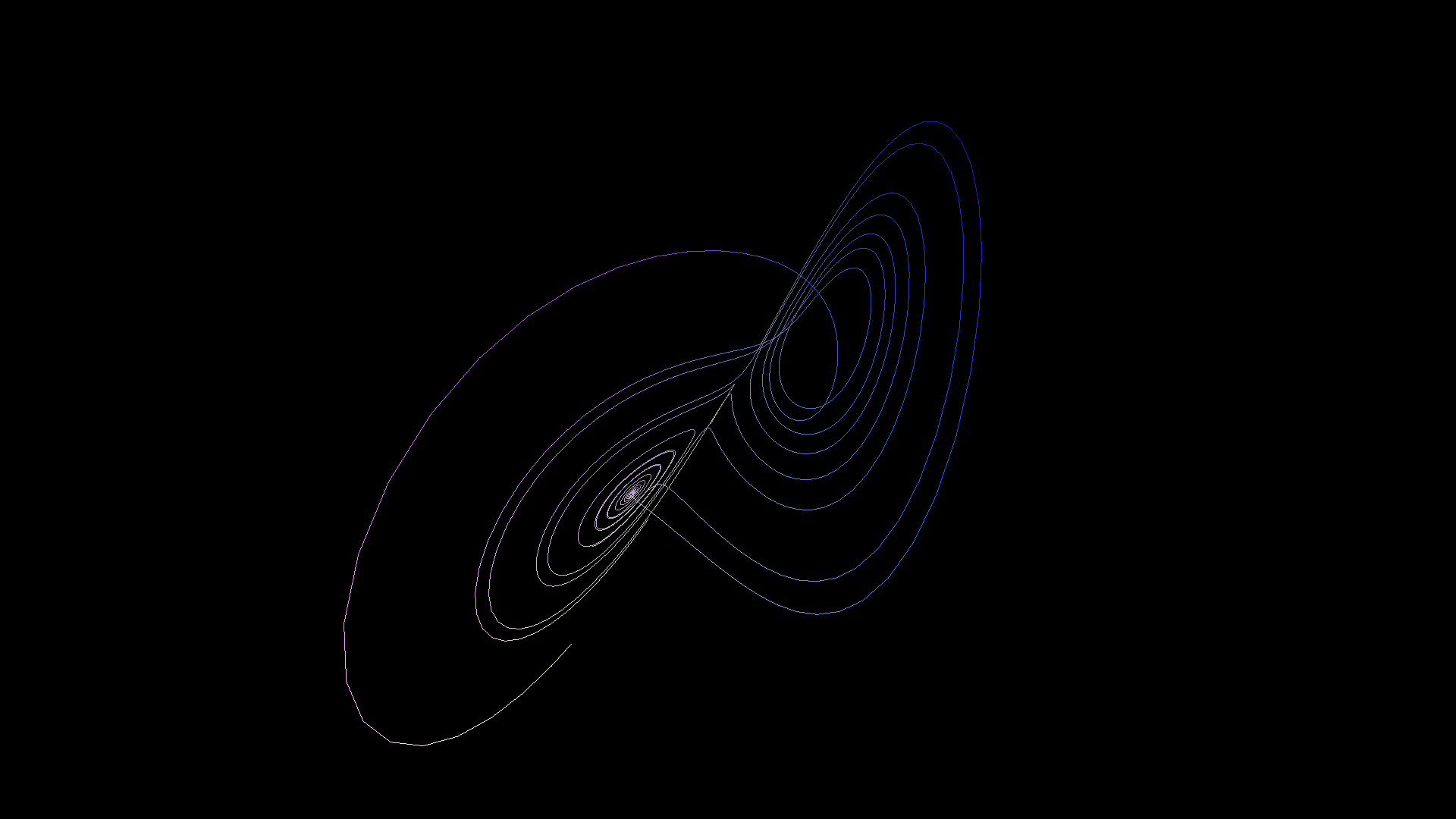


Рисунок 7.4 – Частина анімації, вигляд масштабування системи Лоренца та її обертання за віссю Х №3

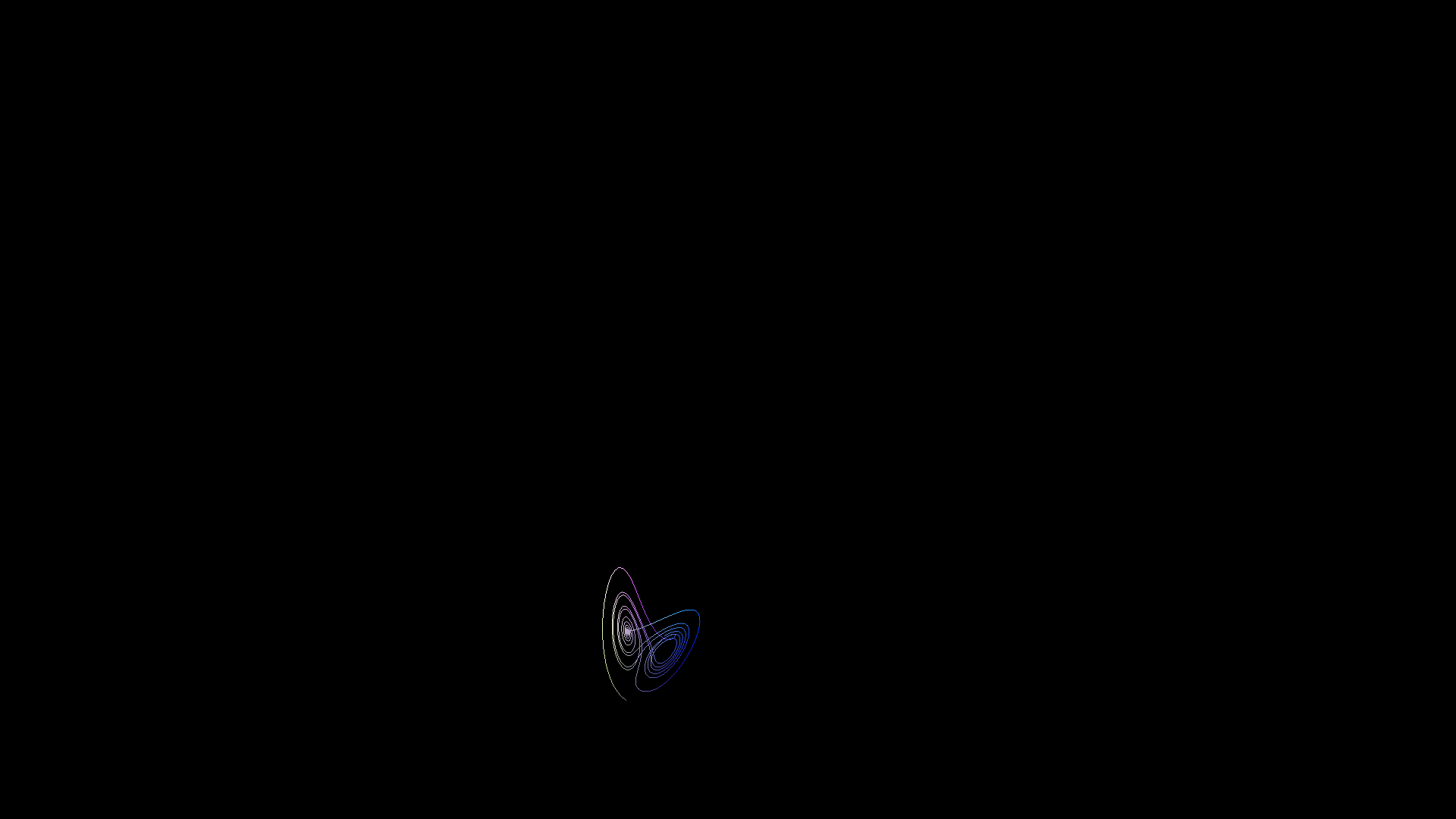


Рисунок 7.5 – Частина анімації, вигляд масштабування системи Лоренца та її обертання за віссю Х №4

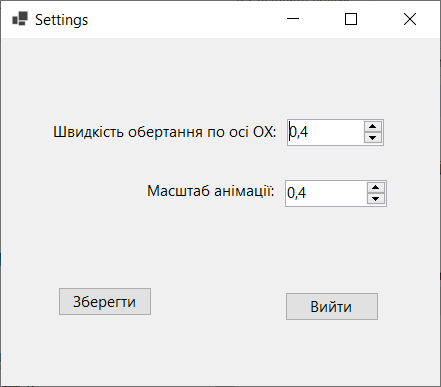


Рисунок 7.6 – Екрана форма налаштування

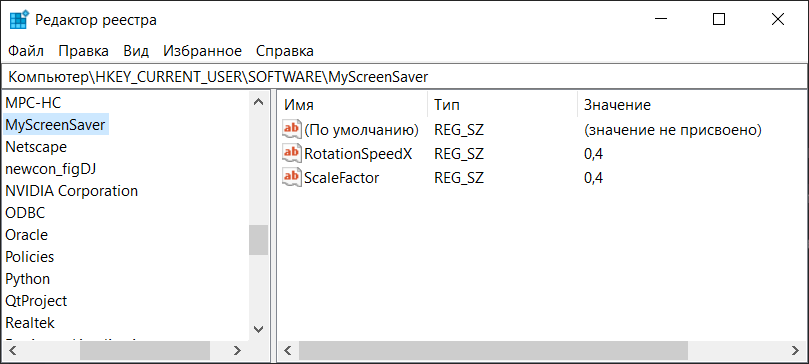


Рисунок 7.7 – Змінні налаштування анімації зберігаються в реєстрі Windows

### Контроль виконання вимог та елементів завдання

В результаті виконання практичної роботи були повністю виконані елементи базового рівня та частково підвищеного рівня складності, що відображено в таблиці 7.1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблиця 7.1 | | | | |
| № з/п | Складність | Вимоги | Бали | Зроблено |
| 1 | Базовий рівень | Прив’язка анімації до механізму простою операційної системи | 2 | **+** |
| 2 | Реалізація повноекранного режиму (команда «/s») | 1 | **+** |
|  | Реалізація налаштування програми Screen Saver (команда «/c») | 1 | **+** |
| 3 | Реалізація попереднього перегляду (команда «/p») | 1 | **+** |
| 4 | Підвищений рівень | Збереження конфігурації і налаштувань програми ScreenSaver у реєстрі ОС | 1 | **+** |
| 5 | Використання складних і видовищних алгоритмів для формування зображення (наприклад, фрактали) | 2 | **-** |

# Загальний перелік посилань

1. Microsoft. glDrawArrays function [Електронний ресурс] / Microsoft – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/windows/win32/opengl/gldrawarrays>.

# Додаток А. Лістинг програми до практичної роботи №1

### Код файлу (RenderControl.cs)

1. public partial class RenderControl : OpenGL
2. {
3. Figure figure = new Figure();
4. public RenderControl()
5. {
6. InitializeComponent();
7. }
8. private void RenderControl\_Render(object sender, EventArgs e)
9. {
10. glClear(GL\_COLOR\_BUFFER\_BIT);
11. glLoadIdentity();
12. glViewport(0, 0, Width, Height);
13. gluOrtho2D(-3.5, 9.8, -2.5, 5.5);
14. figure.DrawGrid();
15. figure.DrawAxes();
16. figure.DrawPentagon();
17. figure.DrawPoints();
19. glColor3f(0, 0, 0);
20. DrawText("X", 8.7, -1.8);
21. DrawText("Y", -2.7, 4.7);
22. DrawText("0", -2.5, -1.8);
24. }
25. }

### Код файлу (Figure.cs)

1. internal class Figure
2. {
3. public void DrawPentagon()
4. {
5. glColor3d(0, 1, 1);
7. glLineWidth(2.5f);
8. glBegin(GL\_LINES);
9. glVertex2d(-1.5, 1.5);
10. glVertex2d(0.5, 3.5);
11. glVertex2d(0.5, 3.5);
12. glVertex2d(1.5, 2.5);
13. glVertex2d(1.5, 2.5);
14. glVertex2d(1.5, -0.5);
15. glVertex2d(1.5, -0.5);
16. glVertex2d(-0.5, 0.5);
17. glVertex2d(-0.5, 0.5);
18. glVertex2d(-1.5, 1.5);
19. glEnd();
20. }
21. public void DrawPoints()
22. {
23. glPointSize(8.0f);
25. glBegin(GL\_POINTS);
26. glVertex2d(3.5, 1.5);
27. glVertex2d(4.5, 0.5);
28. glVertex2d(5.5, 3.5);
29. glVertex2d(6.5, 2.5);
30. glVertex2d(6.5, -0.5);
31. glEnd();
32. }
33. public void DrawAxes()
34. {
35. glColor3f(0.0f, 0.0f, 0.0f);
36. glLineWidth(3.0f);
37. glBegin(GL\_LINES);
38. glVertex2d(-2.5, -1.5);
39. glVertex2d(8.5, -1.5);
40. glVertex2d(8.3, -1.3);
41. glVertex2d(8.5, -1.5);
42. glVertex2d(8.3, -1.7);
43. glVertex2d(8.5, -1.5);
44. glVertex2d(-2.5, -1.5);
45. glVertex2d(-2.5, 4.5);
46. glVertex2d(-2.3, 4.3);
47. glVertex2d(-2.5, 4.5);
48. glVertex2d(-2.7, 4.3);
49. glVertex2d(-2.5, 4.5);
50. glEnd();
51. }
52. public void DrawGrid()
53. {
54. glLineStipple(1, 0xAAAA);
55. glEnable(GL\_LINE\_STIPPLE);
56. glLineWidth(1.0f);
57. glBegin(GL\_LINES);
58. for (double x = -3.5; x <= 9.8; x += 1.0)
59. {
60. glVertex2d(x, -2.5);
61. glVertex2d(x, 5.5);
63. }
64. for (double y = -2.5; y <= 5.5; y += 1.0)
65. {
66. glVertex2d(-3.5, y);
67. glVertex2d(9.8, y);
68. }
69. glEnd();
70. glDisable(GL\_LINE\_STIPPLE);
72. }
73. }

# Додаток Б. Лістинг програми до практичної роботи №2

### Код файлу (RenderControl.cs)

1. public partial class RenderControl : OpenGL
2. {
3. private RenderSettings settings;
4. public RenderControl()
5. {
6. InitializeComponent();
7. }
8. public void SetSettings(RenderSettings renderSettings)
9. {
10. settings = renderSettings;
11. }
13. private void OnRender(object sender, EventArgs e)
14. {
15. glClear(GL\_COLOR\_BUFFER\_BIT);
16. glLoadIdentity();
17. float aspectRatio = (float)Width / Height;
18. int horizont = settings.Horizont;
19. int vertical = settings.Vertical;
20. if (Width > Height)
21. glViewport((Width - Height) / 2, 0, Height, Height);
22. else glViewport(0, (Height - Width) / 2, Width, Width);
23. if (vertical > horizont)
24. gluOrtho2D(-0.3 \* (vertical), +0.3 \* (vertical), -0.3 \* (vertical / 1), 0);
25. else if (vertical == 1 && horizont == 1)
26. gluOrtho2D(-0.3, +0.3, -0.3, +0.3);
27. else gluOrtho2D(-0.3 \* (horizont / 2), +0.3 \* (horizont), -0.3 \* (horizont / 1), +0.3);
28. for (int v = 0; v < vertical; v++)
29. {
30. for (int h = 0; h < horizont; h++)
31. {
32. double offsetX = 0.23 \* h;
33. double offsetY = -0.1299 \* (h % 2) - 0.2598 \* v;
34. DrawHexagonAndTriangles(offsetX, offsetY);
35. }
36. }
37. }
38. private void DrawHexagonAndTriangles(double offsetX, double offsetY)
39. {
40. HexagonRenderer hexagonRenderer = new HexagonRenderer(0.15, settings.Fill, settings.Line, settings.Points);
41. hexagonRenderer.DrawHexagon(offsetX, offsetY);
42. }
43. }

### Код файлу (HexagonRenderer.cs)

1. public class HexagonRenderer
2. {
3. private double radius;
4. private bool fill;
5. private bool line;
6. private bool points;
7. public HexagonRenderer(double radius, bool fill, bool line, bool points)
8. {
9. this.radius = radius;
10. this.fill = fill;
11. this.line = line;
12. this.points = points;
13. }
14. public void DrawHexagon(double offsetX, double offsetY)
15. {
16. int numVertices = 6;
17. double[,] vertices = new double[numVertices, 2];
18. for (int i = 0; i < numVertices; i++)
19. {
20. double angle = 2.0 \* Math.PI \* i / numVertices;
21. double x = radius \* Math.Cos(angle) + offsetX;
22. double y = radius \* Math.Sin(angle) + offsetY;
23. vertices[i, 0] = x;
24. vertices[i, 1] = y;
25. }
26. if (fill)
27. {
28. glColor3f(1.0f, 0.0f, 0.0f);
29. glBegin(GL\_POLYGON);
30. for (int i = 0; i < numVertices; i++)
31. {
32. glVertex2d(vertices[i, 0], vertices[i, 1]);
33. }
34. glEnd();
35. }
36. if (line)
37. {
38. glColor3f(1.0f, 0.0f, 0.0f);
39. glBegin(GL\_LINE\_LOOP);
40. for (int i = 0; i < numVertices; i++)
41. {
42. glVertex2d(vertices[i, 0], vertices[i, 1]);
43. }
44. glEnd();
45. }
46. if (points)
47. {
48. glPointSize(5.0f);
49. glColor3f(1.0f, 0.0f, 0.0f);
50. glBegin(GL\_POINTS);
51. for (int i = 0; i < numVertices; i++)
52. {
53. glVertex2d(vertices[i, 0], vertices[i, 1]);
54. }
55. glEnd();
56. }
57. TriangleRenderer triangleRenderer = new TriangleRenderer(fill, line, points);
58. triangleRenderer.DrawTriangle(vertices, 0, 1, offsetX, offsetY, 0, 1, 1);
59. triangleRenderer.DrawTriangle(vertices, 0, 5, offsetX, offsetY, 0, 1, 0);
60. triangleRenderer.DrawTriangle(vertices, 4, 5, offsetX, offsetY, 1, 1, 0);
61. }
62. }

### Код файлу (RenderSettings.cs)

1. public class RenderSettings
2. {
3. public int Horizont { get; set; } = 1;
4. public int Vertical { get; set; } = 1;
5. public bool Fill { get; set; } = true;
6. public bool Line { get; set; } = false;
7. public bool Points { get; set; } = false;
8. }

### Код файлу (TriangleRenderer.cs)

1. public class TriangleRenderer
2. {
3. private bool fill;
4. private bool line;
5. private bool points;
6. public TriangleRenderer(bool fill, bool line, bool points)
7. {
8. this.fill = fill;
9. this.line = line;
10. this.points = points;
11. }
12. public void DrawTriangle(double[,] vertices, int v1, int v2, double offsetX, double offsetY, float r, float g, float b)
13. {
14. if (fill)
15. {
16. glColor3f(r, g, b);
17. glBegin(GL\_TRIANGLE\_FAN);
18. glVertex2d(offsetX, offsetY);
19. glVertex2d(vertices[v1, 0], vertices[v1, 1]);
20. glVertex2d(vertices[v2, 0], vertices[v2, 1]);
21. glEnd();
22. }
23. if (line)
24. {
25. glColor3f(r, g, b);
26. glBegin(GL\_LINE\_LOOP);
27. glVertex2d(offsetX, offsetY);
28. glVertex2d(vertices[v1, 0], vertices[v1, 1]);
29. glVertex2d(vertices[v2, 0], vertices[v2, 1]);
30. glEnd();
31. }
32. if (points)
33. {
34. glPointSize(5.0f);
35. glColor3f(r, g, b);
36. glBegin(GL\_POINTS);
37. glVertex2d(offsetX, offsetY);
38. glVertex2d(vertices[v1, 0], vertices[v1, 1]);
39. glVertex2d(vertices[v2, 0], vertices[v2, 1]);
40. glEnd();
41. }
42. }
43. }

### Код файлу (TriangleRenderer.cs)

1. public partial class MainForm : Form
2. {
3. private RenderSettings renderSettings;
4. public MainForm()
5. {
6. InitializeComponent();
7. renderSettings = new RenderSettings
8. {
9. Horizont = 1,
10. Vertical = 1,
11. Fill = true,
12. Line = false,
13. Points = false
14. };
16. numericUpDown\_Horizont.Value = renderSettings.Horizont;
17. numericUpDown\_Vertical.Value = renderSettings.Vertical;
18. radioButton\_Fill.Checked = renderSettings.Fill;
19. renderControl1.SetSettings(renderSettings);
20. }
21. private void numericUpDown\_Horizont\_ValueChanged(object sender, EventArgs e)
22. {
23. renderSettings.Horizont = (int)numericUpDown\_Horizont.Value;
24. renderControl1.Invalidate();
25. }
26. private void numericUpDown\_Vertical\_ValueChanged(object sender, EventArgs e)
27. {
28. renderSettings.Vertical = (int)numericUpDown\_Vertical.Value;
29. renderControl1.Invalidate();
30. }
31. private void radioButton\_Fill\_CheckedChanged(object sender, EventArgs e)
32. {
33. renderSettings.Fill = radioButton\_Fill.Checked;
34. renderControl1.Invalidate();
35. }
36. private void radioButton\_Line\_CheckedChanged(object sender, EventArgs e)
37. {
38. renderSettings.Line = radioButton\_Line.Checked;
39. renderControl1.Invalidate();
40. }
41. private void radioButton\_Points\_CheckedChanged(object sender, EventArgs e)
42. {
43. renderSettings.Points = radioButton\_Points.Checked;
44. renderControl1.Invalidate();
45. }
46. }

# Додаток В. Лістинг програми до практичної роботи №3

### Код файлу (RenderControl.cs)

1. public partial class RenderControl : OpenGL
2. {
3. public RenderParameters Parameters { get; set; } = new RenderParameters();
4. public RenderControl()
5. {
6. InitializeComponent();
7. }
8. private void OnRender(object sender, EventArgs e)
9. {
10. double calculatedYmin;
11. double calculatedYmax;
12. Parameters.CurrentRenderer.CalculateYBounds(
13. Parameters.Xmin,
14. Parameters.Xmax,
15. Parameters.N,
16. out calculatedYmin,
17. out calculatedYmax
18. );
19. Parameters.Ymin = calculatedYmin;
20. Parameters.Ymax = calculatedYmax;
21. glClear(GL\_COLOR\_BUFFER\_BIT);
22. glLoadIdentity();
23. glViewport(0, 0, Width, Height);
24. gluOrtho2D(Parameters.Xmin-0.15, Parameters.Xmax+0.15, Parameters.Ymin-0.15, Parameters.Ymax+0.15);
25. DrawAxesAndGrid();
26. DrawAxisLabels();
27. DrawTicks();
28. DrawGraph();
29. DrawIntersections();
31. }
32. private void DrawGraph()
33. {
34. double step = (Parameters.Xmax - Parameters.Xmin) / Parameters.N;
35. bool isContinuous = true;
36. for (double x = Parameters.Xmin; x < Parameters.Xmax; x += step)
37. {
38. double y1 = Parameters.CurrentRenderer.CalculateFunction(x);
39. double y2 = Parameters.CurrentRenderer.CalculateFunction(x + step);
40. if (double.IsInfinity(y1) || double.IsInfinity(y2) || Math.Abs(y2 - y1) > 10 || !IsDefined(x) || !IsDefined(x + step))
41. {
42. isContinuous = false;
43. }
44. else
45. {
46. if (isContinuous)
47. {
48. glColor3f(0.0f, 1.0f, 0.0f);
49. glBegin(GL\_LINES);
50. glVertex2d(x, y1);
51. glVertex2d(x + step, y2);
52. glEnd();
53. }
54. else
55. {
56. glLineStipple(1, 0xAAAA);
57. glEnable(GL\_LINE\_STIPPLE);
58. glColor3f(1.0f, 0.0f, 0.0f);
59. glBegin(GL\_LINES);
60. glVertex2d(x, y1);
61. glVertex2d(x + step, y2);
62. glEnd();
63. glDisable(GL\_LINE\_STIPPLE);
64. }
65. isContinuous = true;
66. }
67. }
68. }
69. private bool IsDefined(double x)
70. {
71. double cosValue = Math.Cos(Math.PI \* x);
72. return Math.Abs(cosValue) > 1e-10;
73. }
74. private void DrawIntersections()
75. {
76. glPointSize(5);
77. glColor3f(1.0f, 0.0f, 0.0f);
78. double step = (Parameters.Xmax - Parameters.Xmin) / Parameters.N;
79. for (double x = Parameters.Xmin; x <= Parameters.Xmax - step; x += step)
80. {
81. double y1 = Parameters.CurrentRenderer.CalculateFunction(x);
82. double y2 = Parameters.CurrentRenderer.CalculateFunction(x + step);
83. if (y1 \* y2 <= 0 && IsDefined(x) && IsDefined(x + step))
84. {
85. double x0 = BisectionMethod(x, x + step);
86. double y0 = Parameters.CurrentRenderer.CalculateFunction(x0);
87. glBegin(GL\_POINTS);
88. glVertex2d(x0, 0);
89. glEnd();
90. }
91. }
92. }
93. private double BisectionMethod(double xLeft, double xRight)
94. {
95. double tolerance = 1e-5;
96. double midPoint;
97. while ((xRight - xLeft) > tolerance)
98. {
99. midPoint = (xLeft + xRight) / 2.0;
100. double yMid = Parameters.CurrentRenderer.CalculateFunction(midPoint);
101. double yLeft = Parameters.CurrentRenderer.CalculateFunction(xLeft);
102. if (yMid == 0)
103. {
104. return midPoint;
105. }
106. else if (yLeft \* yMid < 0)
107. {
108. xRight = midPoint;
109. }
110. else
111. {
112. xLeft = midPoint;
113. }
114. }
115. return (xLeft + xRight) / 2.0;
116. }
117. private void DrawAxesAndGrid()
118. {
119. glLineStipple(1, 0xAAAA);
120. glEnable(GL\_LINE\_STIPPLE);
121. glBegin(GL\_LINES);
122. glColor3f(0.8f, 0.8f, 0.8f);
123. for (double x = Parameters.Xmin; x <= Parameters.Xmax; x += 1.0)
124. {
125. glVertex2d(x, Parameters.Ymin);
126. glVertex2d(x, Parameters.Ymax);
127. }
128. for (double y = Parameters.Ymin; y <= Parameters.Ymax; y += 1.0)
129. {
130. glVertex2d(Parameters.Xmin, y);
131. glVertex2d(Parameters.Xmax, y);
132. }
133. glEnd();
134. glDisable(GL\_LINE\_STIPPLE);
135. glLineWidth(2.0f);
136. glColor3f(1.0f, 1.0f, 1.0f);
137. glBegin(GL\_LINES);
138. glVertex2d(Parameters.Xmin, 0);
139. glVertex2d(Parameters.Xmax, 0);
140. glVertex2d(0, Parameters.Ymin);
141. glVertex2d(0, Parameters.Ymax);
142. glEnd();
143. }
144. private void DrawAxisLabels()
145. {
146. glColor3f(1.0f, 1.0f, 1.0f);
147. for (double x = Math.Ceiling(Parameters.Xmin); x <= Parameters.Xmax; x += 1.0)
148. {
149. DrawText($"{x}", x + 0.05, -0.3);
150. }
151. for (double y = Math.Ceiling(Parameters.Ymin); y <= Parameters.Ymax; y += 1.0)
152. {
153. if (Math.Abs(y) > 0.1)
154. {
155. DrawText($"{y}", -0.2, y);
156. }
157. }
158. }
159. private void DrawTicks()
160. {
161. glColor3f(1.0f, 1.0f, 1.0f);
162. glBegin(GL\_LINES);
163. for (double x = Math.Ceiling(Parameters.Xmin); x <= Parameters.Xmax; x += 1.0)
164. {
165. glLineWidth(2.0f);
166. glVertex2d(x, -0.1);
167. glVertex2d(x, 0.1);
168. }
169. for (double y = Math.Ceiling(Parameters.Ymin); y <= Parameters.Ymax; y += 1.0)
170. {
171. if (Math.Abs(y) > 0.1)
172. {
173. glLineWidth(2.0f);
174. glVertex2d(-0.1, y);
175. glVertex2d(0.1, y);
176. }
177. }
178. glEnd();
179. }
181. }

### Код файлу (RenderParameters.cs)

1. public class RenderParameters
2. {
3. public double Xmin { get; set; } = -2;
4. public double Xmax { get; set; } = 4;
5. public int N { get; set; } = 2000;
6. public double Ymin { get; set; } = -2;
7. public double Ymax { get; set; } = 4;
8. public FunctionRenderer CurrentRenderer { get; set; } = new Function1Renderer();
9. public void UseFunction1()
10. {
11. CurrentRenderer = new Function1Renderer();
12. }
13. public void UseFunction2(Func<double> getYmin, Func<double> getYmax)
14. {
15. CurrentRenderer = new Function2Renderer(getYmin, getYmax);
16. }
17. }

### Код файлу (FunctionRenderer.cs)

1. public abstract class FunctionRenderer
2. {
3. public abstract double CalculateFunction(double x);
4. public virtual void CalculateYBounds(double Xmin, double Xmax, int N, out double Ymin, out double Ymax)
5. {
6. double step = (Xmax - Xmin) / N;
7. double ymin = double.MaxValue;
8. double ymax = double.MinValue;
9. for (double x = Xmin; x <= Xmax; x += step)
10. {
11. double y = CalculateFunction(x);
12. if (!double.IsInfinity(y) && !double.IsNaN(y))
13. {
14. if (y < ymin) ymin = y;
15. if (y > ymax) ymax = y;
16. }
17. }
18. Ymin = Math.Floor(ymin);
19. Ymax = Math.Ceiling(ymax);
20. }
21. }

### Код файлу (MainForm.cs)

1. public partial class MainForm : Form
2. {
3. private RenderParameters parameters;
4. public MainForm()
5. {
6. InitializeComponent();
7. parameters = new RenderParameters();
8. renderControl1.Parameters = parameters;
9. }
10. private void numericUpDown\_Xmin\_ValueChanged(object sender, EventArgs e)
11. {
12. parameters.Xmin = (double)numericUpDown\_Xmin.Value;
13. renderControl1.Invalidate();
14. }
15. private void numericUpDown\_Xmax\_ValueChanged(object sender, EventArgs e)
16. {
17. parameters.Xmax = (double)numericUpDown\_Xmax.Value;
18. renderControl1.Invalidate();
19. }
20. private void numericUpDown\_Ymin\_ValueChanged(object sender, EventArgs e)
21. {
22. parameters.Ymin = (double)numericUpDown\_Ymin.Value;
23. renderControl1.Invalidate();
24. }
25. private void numericUpDown\_Ymax\_ValueChanged(object sender, EventArgs e)
26. {
27. parameters.Ymax = (double)numericUpDown\_Ymax.Value;
28. renderControl1.Invalidate();
29. }
30. private void numericUpDown\_N\_ValueChanged(object sender, EventArgs e)
31. {
32. parameters.N = (int)numericUpDown\_N.Value;
33. renderControl1.Invalidate();
34. }
35. private void checkBox\_F2\_CheckedChanged(object sender, EventArgs e)
36. {
37. if (checkBox\_F2.Checked)
38. {
39. numericUpDown\_Ymin.Value = (decimal)parameters.Ymin;
40. numericUpDown\_Ymax.Value = (decimal)parameters.Ymax;
41. label\_Ymax.Visible = true;
42. label\_Ymin.Visible = true;
43. numericUpDown\_Ymax.Visible = true;
44. numericUpDown\_Ymin.Visible = true;
45. parameters.UseFunction2(
46. () => (double)numericUpDown\_Ymin.Value,
47. () => (double)numericUpDown\_Ymax.Value
48. );
49. }
50. else
51. {
52. parameters.UseFunction1();
53. label\_Ymax.Visible = false;
54. label\_Ymin.Visible = false;
55. numericUpDown\_Ymax.Visible = false;
56. numericUpDown\_Ymin.Visible = false;
57. }
58. renderControl1.Invalidate();
59. }
60. }

### Код файлу (Function1Renderer.cs)

1. public class Function1Renderer : FunctionRenderer
2. {
3. public override double CalculateFunction(double x)
4. {
5. return Math.Tan((0.5 \* Math.Sin(2 \* x)) / 1.5 + Math.Cos(5 \* x));
6. }
7. }

# Додаток Г. Лістинг програми до практичної роботи №4

### Код файлу (RenderControl.cs)

1. public partial class RenderControl : OpenGL
2. {
3. double size = 1.1;
4. double xMin => -size;
5. double xMax => size;
6. double yMin => -size;
7. double yMax => size;
8. public int segmentsParabola = 5;
9. public int segmentsEllipse = 6;
10. private List<PointF> segmentPoints = new List<PointF>();
11. private List<Shape> shapes = new List<Shape>();
12. private LineSegment lineSegment;
13. public int SegmentsParabola
14. {
15. get { return segmentsParabola; }
16. set
17. {
18. segmentsParabola = value;
19. UpdateShapes();
20. Invalidate();
21. }
22. }
23. public int SegmentsEllipse
24. {
25. get { return segmentsEllipse; }
26. set
27. {
28. segmentsEllipse = value;
29. UpdateShapes();
30. Invalidate();
31. }
32. }
33. private void UpdateShapes()
34. {
35. shapes.Clear();
36. shapes.Add(new Ellipse(0.5, 0.3, segmentsEllipse));
37. shapes.Add(new Parabola(0.5, segmentsParabola));
38. }
39. public RenderControl()
40. {
41. InitializeComponent();
42. shapes.Add(new Ellipse(0.5, 0.3, segmentsEllipse));
43. shapes.Add(new Parabola(0.5, segmentsParabola));
44. }
45. private void OnRender(object sender, EventArgs e)
46. {
47. glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);
48. glLoadIdentity();
49. if (Width > Height)
50. glViewport((Width - Height) / 2, 0, Height, Height);
51. else glViewport(0, (Height - Width) / 2, Width, Width);
52. glOrtho(xMin, xMax, yMin, yMax, -1.0, 1.0);
53. DrawAxes();
54. foreach (var shape in shapes)
55. {
56. shape.Draw();
57. }
58. lineSegment?.Draw();
59. if (lineSegment != null)
60. {
61. foreach (var shape in shapes)
62. {
63. List<PointF> intersections = shape.GetIntersectionPoints(lineSegment);
64. DrawIntersectionPoints(intersections);
65. }
66. }
67. }
68. private void DrawAxes()
69. {
70. glColor3d(0.8, 0.8, 0.8);
71. glBegin(GL\_LINES);
72. glVertex2d(-1.0, 0.0);
73. glVertex2d(1.0, 0.0);
74. glVertex2d(0.0, -1.0);
75. glVertex2d(0.0, 1.0);
76. glEnd();
77. DrawText("X", +1, -0.1);
78. DrawText("Y", +0.02, +1);
80. }
81. private void DrawIntersectionPoints(List<PointF> intersections)
82. {
83. glColor3d(1.0, 1.0, 0.0);
84. glPointSize(5.0f);
85. glBegin(GL\_POINTS);
86. foreach (var point in intersections)
87. {
88. glVertex2d(point.X, point.Y);
89. }
90. glEnd();
91. foreach (var point in intersections)
92. {
93. string text = $"({point.X:F3}; {point.Y:F3})";
94. DrawText(text, point.X + 0.02, point.Y + 0.02);
95. }
96. }
97. private void OnMouseClick(object sender, System.Windows.Forms.MouseEventArgs e)
98. {
99. float glX = (float)((e.X / (double)Width) \* 2.2 - 1.1);
100. float glY = (float)(((Height - e.Y) / (double)Height) \* 2.2 - 1.1);
101. if (segmentPoints.Count < 2)
102. {
103. segmentPoints.Add(new PointF(glX, glY));
104. if (segmentPoints.Count == 2)
105. {
106. lineSegment = new LineSegment(segmentPoints[0], segmentPoints[1]);
107. }
108. }
109. else
110. {
111. segmentPoints.Clear();
112. lineSegment = null;
113. }
114. Invalidate();
115. }
116. }

### Код файлу (MainForm.cs)

1. public partial class MainForm : Form
2. {
4. public MainForm()
5. {
6. InitializeComponent();
7. }
8. private void numeric\_Parabola\_ValueChanged(object sender, System.EventArgs e)
9. {
10. renderControl1.SegmentsParabola = (int)numeric\_Parabola.Value;
11. renderControl1.Invalidate();
12. }
13. private void numeric\_Elips\_ValueChanged(object sender, System.EventArgs e)
14. {
15. renderControl1.SegmentsEllipse = (int)numeric\_Elips.Value;
16. renderControl1.Invalidate();
17. }
18. }

### Код файлу (LineSegment.cs)

1. public class LineSegment
2. {
3. public PointF Start { get; set; }
4. public PointF End { get; set; }
5. public LineSegment(PointF start, PointF end)
6. {
7. Start = start;
8. End = end;
9. }
10. public void Draw()
11. {
12. glColor3d(1.0, 0.0, 0.0);
13. glBegin(GL\_LINES);
14. glVertex2d(Start.X, Start.Y);
15. glVertex2d(End.X, End.Y);
16. glEnd();
17. }
18. public bool CheckIntersection(PointF p1, PointF p2, out PointF intersection)
19. {
20. intersection = default;
21. double a11 = End.X - Start.X;
22. double a12 = p1.X - p2.X;
23. double a21 = End.Y - Start.Y;
24. double a22 = p1.Y - p2.Y;
25. double b1 = p1.X - Start.X;
26. double b2 = p1.Y - Start.Y;
27. double det = a11 \* a22 - a12 \* a21;
28. if (Math.Abs(det) < 1e-6)
29. return false;
30. double det1 = b1 \* a22 - a12 \* b2;
31. double det2 = a11 \* b2 - b1 \* a21;
32. double t1 = det1 / det;
33. double t2 = det2 / det;
34. if (t1 < 0 || t1 > 1 || t2 < 0 || t2 > 1)
35. return false;
36. intersection = new PointF(
37. (float)(Start.X + t1 \* (End.X - Start.X)),
38. (float)(Start.Y + t1 \* (End.Y - Start.Y))
39. );
40. return true;
41. }
42. }

### Код файлу (Shape.cs)

1. public abstract class Shape
2. {
3. public abstract void Draw();
4. public abstract List<PointF> GetIntersectionPoints(LineSegment line);
5. }

### Код файлу (Ellipse.cs)

1. public class Ellipse : Shape
2. {
3. public double A { get; set; }
4. public double B { get; set; }
5. private int Segments { get; set; }
6. public Ellipse(double a, double b, int segments)
7. {
8. A = a;
9. B = b;
10. Segments = segments;
11. }
12. public override void Draw()
13. {
14. glColor3d(0.0, 1.0, 0.0);
15. glBegin(GL\_LINE\_LOOP);
16. double angleStep = 2 \* Math.PI / Segments;
17. for (int i = 0; i <= Segments; i++)
18. {
19. double angle = i \* angleStep;
20. double x = A \* Math.Cos(angle);
21. double y = B \* Math.Sin(angle);
22. glVertex2d(x, y);
23. }
24. glEnd();
25. }
26. public override List<PointF> GetIntersectionPoints(LineSegment line)
27. {
28. List<PointF> intersections = new List<PointF>();
29. double angleStep = 2 \* Math.PI / Segments;
30. PointF previousPoint = new PointF((float)(A \* Math.Cos(0)), (float)(B \* Math.Sin(0)));
31. for (int i = 1; i <= Segments; i++)
32. {
33. double angle = i \* angleStep;
34. PointF currentPoint = new PointF(
35. (float)(A \* Math.Cos(angle)),
36. (float)(B \* Math.Sin(angle))
37. );
38. if (line.CheckIntersection(previousPoint, currentPoint, out PointF intersection))
39. {
40. intersections.Add(intersection);
41. }
42. previousPoint = currentPoint;
43. }
44. return intersections;
45. }
46. }

### Код файлу (Parabola.cs)

1. public class Parabola : Shape
2. {
3. public double A { get; set; }
4. private int Points { get; set; }
5. public Parabola(double a, int points)
6. {
7. A = a;
8. Points = points;
9. }
10. public override void Draw()
11. {
12. glColor3d(0.0, 0.0, 1.0);
13. glBegin(GL\_LINE\_STRIP);
14. double step = 2.0 / Points;
15. for (double x = -1.0; x <= 1.0; x += step)
16. {
17. double y = A \* x \* x;
18. glVertex2d(x, y);
19. }
20. glEnd();
21. }
22. public override List<PointF> GetIntersectionPoints(LineSegment line)
23. {
24. return new List<PointF>();
25. }
26. }

# Додаток Ґ. Лістинг програми до практичної роботи №5

### Код файлу (RenderControl.cs)

1. public partial class RenderControl : OpenGL
2. {
3. private bool isWireframe = true;
4. public bool IsWireframe
5. {
6. get => isWireframe;
7. set
8. {
9. isWireframe = value;
10. Invalidate();
11. }
12. }
13. private bool orthoPerstective = true;
14. public bool OrthoPerspective
15. {
16. get => orthoPerstective;
17. set
18. {
19. orthoPerstective = value;
20. Invalidate();
21. }
22. }
23. public int segments = 15;
25. public double sphereRadius = 1.5;
27. public double cylinderRadius = 2.5;
28. public double cylinderHeight = 1.5;
30. public double diskInnerRadius = 1.0;
31. public double diskOuterRadius = 3.5;
32. public int diskStartAngle = 180;
33. public int diskSweepAngle = 135;
34. double size = 6.0;
35. double AspectRatio { get => (double)Width / Height; }
36. double xMin { get => (AspectRatio > 1) ? -size \* AspectRatio : -size; }
37. double xMax { get => (AspectRatio > 1) ? +size \* AspectRatio : +size; }
38. double yMin { get => (AspectRatio < 1) ? -size / AspectRatio : -size; }
39. double yMax { get => (AspectRatio < 1) ? +size / AspectRatio : +size; }
40. double zMin { get => -size; }
41. double zMax { get => +size; }
42. double ax = 10, ay = -20;
43. double sphereX = 2.5, sphereY = 1.0, sphereZ = -3.5;
44. double cylinderX = -2.5, cylinderY = -3.5, cylinderZ = -4.5;
45. public RenderControl()
46. {
47. InitializeComponent();
48. }
49. private void OnRender(object sender, EventArgs e)
50. {
51. glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);
52. glLoadIdentity();
53. glViewport(0, 0, Width, Height);
54. if (orthoPerstective)
55. glOrtho(xMin, xMax, yMin, yMax, zMin, zMax);
56. else
57. {
58. glMatrixMode(GL\_PROJECTION);
59. glLoadIdentity();
60. gluPerspective(45.0, AspectRatio, 1.0, 50.0);
61. glMatrixMode(GL\_MODELVIEW);
62. glLoadIdentity();
63. gluLookAt(0.0, 4.0, 20.0,
64. 0.0, 0.0, 0.0,
65. 0.0, 1.0, 0.0);
66. }
67. glRotated(ax--, 1, 0, 0);
68. glRotated(ay++, 0, 1, 0);
69. //DrawGridLabels();
70. DrawGridXZWithLabels();
71. DrawAxes();
72. glEnable(GL\_LIGHTING);
73. EnableLighting();
74. DrawSphere();
75. DrawPartialDisk();
76. DrawCylinder();
77. glDisable(GL\_LIGHTING);
78. }
79. private void SetMaterial(float[] ambient, float[] diffuse, float[] specular, float shininess)
80. {
81. glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_AMBIENT, ambient);
82. glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, diffuse);
83. glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_SPECULAR, specular);
84. glMaterialf(GL\_FRONT\_AND\_BACK, GL\_SHININESS, shininess);
85. }
86. private void EnableLighting()
87. {
88. glEnable(GL\_LIGHTING);
89. glEnable(GL\_LIGHT0);
90. float[] lightPosition = { 0.0f, 15.0f, 15.0f, 1.0f };
91. float[] lightAmbient = { 0.3f, 0.3f, 0.3f, 1.0f };
92. float[] lightDiffuse = { 0.8f, 0.8f, 0.8f, 1.0f };
93. float[] lightSpecular = { 1.0f, 1.0f, 1.0f, 1.0f };
94. glLightfv(GL\_LIGHT0, GL\_POSITION, lightPosition);
95. glLightfv(GL\_LIGHT0, GL\_AMBIENT, lightAmbient);
96. glLightfv(GL\_LIGHT0, GL\_DIFFUSE, lightDiffuse);
97. glLightfv(GL\_LIGHT0, GL\_SPECULAR, lightSpecular);
98. glEnable(GL\_COLOR\_MATERIAL);
99. glColorMaterial(GL\_FRONT\_AND\_BACK, GL\_AMBIENT\_AND\_DIFFUSE);
100. glEnable(GL\_NORMALIZE);
101. }
102. private void DrawSphere()
103. {
104. float[] ambient = { 0.2f, 0.2f, 0.6f, 1.0f };
105. float[] diffuse = { 0.5f, 0.5f, 1.0f, 1.0f };
106. float[] specular = { 1.0f, 1.0f, 1.0f, 1.0f };
107. float shininess = 50.0f;
108. SetMaterial(ambient, diffuse, specular, shininess);
109. glPushMatrix();
110. glTranslated(sphereX, sphereY, sphereZ);
111. glColor3d(0.5, 0.5, 1.0);
112. IntPtr quadric = gluNewQuadric();
113. if (quadric == IntPtr.Zero)
114. {
115. throw new Exception("Не вдалося створити Quadric об'єкт.");
116. }
117. try
118. {
119. if (isWireframe)
120. {
121. gluQuadricDrawStyle(quadric, GLU\_LINE);
122. }
123. else
124. {
125. gluQuadricDrawStyle(quadric, GLU\_FILL);
126. }
127. gluQuadricNormals(quadric, GLU\_SMOOTH);
128. gluSphere(quadric, sphereRadius, segments, segments);
129. }
130. finally
131. {
132. gluDeleteQuadric(quadric);
133. }
134. glPopMatrix();
135. }
136. private void DrawPartialDisk()
137. {
138. float[] ambient = { 0.6f, 0.2f, 0.2f, 1.0f };
139. float[] diffuse = { 0.8f, 0.4f, 0.4f, 1.0f };
140. float[] specular = { 1.0f, 1.0f, 1.0f, 1.0f };
141. float shininess = 10.0f;
142. SetMaterial(ambient, diffuse, specular, shininess);
143. glPushMatrix();
144. glTranslated(-3.5, 3.0, 3.0);
145. glRotated(90, 1, 0, 0);
146. glColor3d(1.0, 0.5, 0.0);
147. IntPtr quadric = gluNewQuadric();
148. if (quadric == IntPtr.Zero)
149. {
150. throw new Exception("Не вдалося створити Quadric об'єкт.");
151. }
152. try
153. {
154. if (isWireframe)
155. {
156. gluQuadricDrawStyle(quadric, GLU\_LINE);
157. }
158. else
159. {
160. gluQuadricDrawStyle(quadric, GLU\_FILL);
161. }
162. gluQuadricNormals(quadric, GLU\_SMOOTH);
163. gluPartialDisk(quadric, diskInnerRadius, diskOuterRadius, segments, 1, diskStartAngle, diskSweepAngle);
164. }
165. finally
166. {
167. gluDeleteQuadric(quadric);
168. }
169. glPopMatrix();
170. }
171. private void DrawCylinder()
172. {
173. float[] ambient = { 0.2f, 0.6f, 0.6f, 1.0f };
174. float[] diffuse = { 0.4f, 0.8f, 0.8f, 1.0f };
175. float[] specular = { 1.0f, 1.0f, 1.0f, 1.0f };
176. float shininess = 25.0f;
177. SetMaterial(ambient, diffuse, specular, shininess);
178. glPushMatrix();
179. glTranslated(cylinderX, cylinderY, cylinderZ);
180. glRotated(180, 1, 0, 0);
181. glColor3d(0.0, 0.7, 0.7);
182. IntPtr quadric = gluNewQuadric();
183. if (quadric == IntPtr.Zero)
184. {
185. throw new Exception("Не вдалося створити Quadric об'єкт.");
186. }
187. try
188. {
189. if (isWireframe)
190. {
191. gluQuadricDrawStyle(quadric, GLU\_LINE);
192. }
193. else
194. {
195. gluQuadricDrawStyle(quadric, GLU\_FILL);
196. }
197. gluQuadricNormals(quadric, GLU\_SMOOTH);
198. gluCylinder(quadric, cylinderRadius, cylinderRadius, cylinderHeight, segments, segments);
199. }
200. finally
201. {
202. gluDeleteQuadric(quadric);
203. }
204. glPopMatrix();
205. }
206. private void DrawAxes()
207. {
208. glBegin(GL\_LINES);
209. glColor3d(1.0, 0.0, 0.0);
210. glVertex3d(-5.0, 0.0, 0.0);
211. glVertex3d(5.0, 0.0, 0.0);
212. glColor3d(0.0, 1.0, 0.0);
213. glVertex3d(0.0, -5.0, 0.0);
214. glVertex3d(0.0, 5.0, 0.0);
215. glColor3d(0.0, 0.0, 1.0);
216. glVertex3d(0.0, 0.0, -5.0);
217. glVertex3d(0.0, 0.0, 5.0);
219. glEnd();
220. glColor3d(1, 1, 1);
221. DrawText("+X", 5, 0.3, 0);
222. DrawText("+Y", 0.3, 5, 0);
223. DrawText("+Z", 0, 0.3, 5);
224. DrawAxisArrows();
225. }
227. private void DrawGridXZWithLabels()
228. {
229. glColor3d(0.8, 0.8, 0.8);
230. glBegin(GL\_LINES);
231. for (double x = -4.5; x <= 4.5; x += 0.5)
232. {
233. glVertex3d(x, 0.0, -4.5);
234. glVertex3d(x, 0.0, 4.5);
235. }
236. for (double z = -4.5; z <= 4.5; z += 0.5)
237. {
238. glVertex3d(-4.5, 0.0, z);
239. glVertex3d(4.5, 0.0, z);
240. }
241. glEnd();
242. // Додавання міток сітки
243. //for (double x = -4.5; x <= 4.5; x += 0.5)
244. //{
245. // for (double z = -4.5; z <= 4.5; z += 0.5)
246. // {
247. // if ((x == sphereX && z == sphereZ) || (x == -2.5 && z == -4.5) || (x == -3.5 && z == 3.0))
248. // {
249. // glColor3d(1, 1, 1);
250. // DrawText($"\*({x:F1}, {z:F1})", x, 0.0, z);
251. // }
252. // }
253. //}
254. }
255. private void DrawGridLabels()
256. {
257. glColor3d(1.0, 1.0, 1.0);
258. DrawText($"\*", sphereX, sphereY, sphereZ);
259. DrawText($"\*", -2.5, -3.5, -4.5);
260. DrawText($"\*", -3.5, 3.0, 3.0);
262. for (double i = -4.5; i <= 4.5; i += 0.5)
263. {
264. if (i != 0)
265. {
266. DrawText($"{i:F1}", i, -0.3, 0.0);
267. DrawText($"{i:F1}", -0.2, i, 0.0);
268. DrawText($"{i:F1}", 0.0, -0.3, i);
269. }
270. }
271. DrawText("0", 0.0, -0.2, 0.0);
272. }
273. private void DrawAxisArrows()
274. {
275. glBegin(GL\_TRIANGLES);
276. glColor3d(1.0, 0.0, 0.0);
277. glVertex3d(5.2, 0.0, 0.0);
278. glVertex3d(4.8, 0.2, 0.0);
279. glVertex3d(4.8, -0.2, 0.0);
280. glEnd();
281. glBegin(GL\_TRIANGLES);
282. glColor3d(0.0, 1.0, 0.0);
283. glVertex3d(0.0, 5.2, 0.0);
284. glVertex3d(0.2, 4.8, 0.0);
285. glVertex3d(-0.2, 4.8, 0.0);
286. glEnd();
287. glBegin(GL\_TRIANGLES);
288. glColor3d(0.0, 0.0, 1.0);
289. glVertex3d(0.0, 0.0, 5.2);
290. glVertex3d(0.2, 0.0, 4.8);
291. glVertex3d(-0.2, 0.0, 4.8);
292. glEnd();
293. }
294. bool flag = false;
295. Point start;
296. private void RenderControl\_MouseDown(object sender, System.Windows.Forms.MouseEventArgs e)
297. {
298. flag = e.Button == System.Windows.Forms.MouseButtons.Left;
299. start = e.Location;
300. }
301. private void RenderControl\_MouseUp(object sender, System.Windows.Forms.MouseEventArgs e)
302. {
303. if (flag)
304. flag = !(e.Button == System.Windows.Forms.MouseButtons.Left);
305. }
306. private void RenderControl\_MouseMove(object sender, System.Windows.Forms.MouseEventArgs e)
307. {
308. if (flag)
309. {
310. Point current = e.Location;
311. ax += (current.Y - start.Y) / 2.0;
312. ay += (current.X - start.X) / 2.0;
313. start = current;
314. Invalidate();
315. }
316. }
317. }

# Додаток Д. Лістинг програми до практичної роботи №6

### Код файлу (RenderControl.cs)

1. public partial class RenderControl : OpenGL
2. {
3. public event Action UpdateInfo;
4. private double ax = +20;
5. private double ay = -30;
6. private double m = 1;
7. private double aw = 0;
8. private bool fDown = false;
9. private double x0, y0;
10. public double a { get { return 1.3; } }
11. public double a1 { get { return 0.31 \* a; } }
12. public double c { get { return 0.55; } }
13. private bool MoveAxes = true;
14. private int dx;
15. private int dy;
16. private double az;
17. private double s = 0.2;
18. private double sb = 0.2;
19. public double S
20. {
21. get { return s; }
22. set
23. {
24. if (value <= (a1 + c) && value >= (c - a1))
25. s = value;
26. }
27. }
28. public double Sb
29. {
30. get { return sb; }
31. set
32. {
33. if (value <= (a1 + c) && value >= (c - a1))
34. sb = value;
35. }
36. }
37. private double ag
38. {
39. get
40. {
41. return 180.0 / Math.PI \* (Math.PI - Math.Acos((a1 \* a1 + S \* S - c \* c) / (2 \* a1 \* S)));
42. }
43. }
44. private double agb
45. {
46. get
47. {
48. return 180.0 / Math.PI \* (Math.PI - Math.Acos((a1 \* a1 + Sb \* Sb - c \* c) / (2 \* a1 \* Sb)));
49. }
50. }
51. private double az\_m;
52. public double a\_m { get { return -0.68; } }
53. public double a1\_m { get { return -0.34; } }
54. public double b\_m { get { return -0.9; } }
55. public double c\_m { get { return -0.68; } }
56. public double dist { get; set; }
57. public bool gridX { get; set; }
58. public bool gridY { get; set; }
59. public bool gridZ { get; set; }
60. public bool Perspectiv;
61. private double size = 2.5;
62. private double AspectRatio => (double)Width / Height;
63. private double xMin => (AspectRatio > 1) ? -size \* AspectRatio : -size;
64. private double xMax => (AspectRatio > 1) ? +size \* AspectRatio : +size;
65. private double yMin => (AspectRatio < 1) ? -size / AspectRatio : -size;
66. private double yMax => (AspectRatio < 1) ? +size / AspectRatio : +size;
67. private double zMin => -size \* 2.0;
68. private double zMax => +size \* 2.0;
69. private Segment redSegment;
70. private Segment greenSegment;
71. private Segment blueSegment;
72. public RenderControl()
73. {
74. InitializeComponent();
75. MouseWheel += OnMouseWheel;
76. redSegment = new Segment(a\_m, 5);
77. greenSegment = new Segment(c\_m, 5);
78. blueSegment = new Segment(b\_m, 5);
79. }
80. private void OnMouseDown(object sender, MouseEventArgs e)
81. {
82. if (e.Button == MouseButtons.Left)
83. {
84. fDown = true;
85. x0 = e.Location.X;
86. y0 = e.Location.Y;
87. }
88. }
89. private void OnMouseUp(object sender, MouseEventArgs e)
90. {
91. if (fDown && e.Button == MouseButtons.Left)
92. fDown = false;
93. }
94. private void OnMouseMove(object sender, MouseEventArgs e)
95. {
96. if (fDown)
97. {
98. ay -= (x0 - e.X) / 2.0;
99. ax -= (y0 - e.Y) / 2.0;
100. x0 = e.X;
101. y0 = e.Y;
102. Invalidate();
103. }
104. }
105. private void OnMouseWheel(object sender, MouseEventArgs e)
106. {
107. m += e.Delta / 1000.0;
108. Invalidate();
109. }
110. private void RenderControl\_PreviewKeyDown(object sender, PreviewKeyDownEventArgs e)
111. {
112. switch (e.KeyCode)
113. {
114. case Keys.Left: az -= 1; break;
115. case Keys.Right: az += 1; break;
116. case Keys.Up: S += 0.01; break;
117. case Keys.Down: S -= 0.01; break;
118. case Keys.W: Sb += 0.01; break;
119. case Keys.S: Sb -= 0.01; break;
120. case Keys.Q: aw += 1; break;
121. case Keys.E: aw -= 1; break;
122. }
123. Invalidate();
124. }
125. private void OnRender(object sender, EventArgs e)
126. {
127. glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);
128. glMatrixMode(GL\_PROJECTION);
129. glLoadIdentity();
130. if (Perspectiv)
131. gluPerspective(45, AspectRatio, 0.1, 1000.0);
132. else
133. glOrtho(xMin, xMax, yMin, yMax, zMin, zMax);
134. glMatrixMode(GL\_MODELVIEW);
135. glLoadIdentity();
136. glViewport(0, 0, Width, Height);
137. if (Perspectiv)
138. glTranslated(0, 0, -size);
139. glRotated(ax, 1, 0, 0);
140. glRotated(ay, 0, 1, 0);
141. glScaled(m, m, m);
142. glEnable(GL\_LIGHTING);
143. glEnable(GL\_COLOR\_MATERIAL);
144. glEnable(GL\_DEPTH\_TEST);
145. glEnable(GL\_LIGHT0);
146. glColor(Color.Black);
147. Axes(1.9);
148. dist = S;
149. glRotated(aw, 0, -1, 0);
150. glRotated(az, 0, 0, -1);
151. glPushMatrix();
152. glColor(Color.Red);
153. redSegment.Draw();
154. glTranslated(0, a\_m, 0);
155. glRotated(agb, 0, 0, -1);
156. glColor(Color.Green);
157. greenSegment.Draw();
158. glPopMatrix();
159. glPushMatrix();
160. glTranslated(0, a1\_m, 0);
161. glRotated(ag, 0, 0, 1);
162. glColor(Color.Blue);
163. blueSegment.Draw();
164. glPopMatrix();
165. glDisable(GL\_LIGHTING);
166. glDisable(GL\_COLOR\_MATERIAL);
167. glDisable(GL\_DEPTH\_TEST);
168. UpdateInfo?.Invoke();
169. }
170. private void Axes(double size)
171. {
172. double a = size / 15.0;
173. glBegin(GL\_LINES);
174. glVertex3d(-a, 0, 0); glVertex3d(+size, 0, 0);
175. glVertex3d(0, a, 0); glVertex3d(0, -size, 0);
176. glVertex3d(0, 0, -a); glVertex3d(0, 0, +size);
177. glEnd();
178. DrawText("+X", size + a, 0, 0);
179. DrawText("-Y", 0, -size + a, 0);
180. DrawText("+Z", 0, 0, size + a);
181. Grid(gridX, gridY, gridZ, size, -size \* 0.1, 5);
182. }
183. private void Grid(bool x, bool y, bool z, double s, double shift, int count = 10)
184. {
185. double h = s / count;
186. glColor(Color.AliceBlue);
187. glEnable(GL\_LINE\_STIPPLE);
188. glLineStipple(1, 0xCCCC);
189. glBegin(GL\_LINES);
190. for (int i = 0; i <= count; i++)
191. {
192. if (z) { glVertex3d(i \* h, 0, shift); glVertex3d(i \* h, s, shift); }
193. if (y) { glVertex3d(i, shift, i \* h); glVertex3d(s, shift, i \* h); }
194. if (x) { glVertex3d(shift, 0, i \* h); glVertex3d(shift, s, i \* h); }
195. }
196. glEnd();
197. glDisable(GL\_LINE\_STIPPLE);
198. }
199. }

### Код файлу (Segment.cs)

1. public class Segment
2. {
3. public double Size { get; set; }
4. public int LineWidth { get; set; }
5. public Segment(double size, int lineWidth)
6. {
7. Size = size;
8. LineWidth = lineWidth;
9. }
10. public void Draw()
11. {
13. glLineWidth(LineWidth);
14. glBegin(GL\_LINE\_STRIP);
15. glVertex3d(0.0, 0.0, Size \* 0.1);
16. glVertex3d(0.0, Size, Size \* 0.1);
17. glVertex3d(0.0, Size, -Size \* 0.1);
18. glVertex3d(0.0, 0.0, -Size \* 0.1);
19. glVertex3d(0.0, 0.0, Size \* 0.1);
20. glEnd();
21. glLineWidth(1);
22. }
23. }

# Додаток Е Лістинг програми до практичної роботи №7

### Код файлу (AppSettings.cs)

1. public static class AppSettings
2. {
3. private const string RegistryPath = "Software\\MyScreenSaver";
4. public static double RotationSpeedX
5. {
6. get => GetRegistryValue("RotationSpeedX", 0.2);
7. set => SetRegistryValue("RotationSpeedX", value);
8. }
9. public static double ScaleFactor
10. {
11. get => GetRegistryValue("ScaleFactor", 0.5);
12. set => SetRegistryValue("ScaleFactor", value);
13. }
14. private static double GetRegistryValue(string key, double defaultValue)
15. {
16. using (var regKey = Registry.CurrentUser.OpenSubKey(RegistryPath))
17. {
18. if (regKey?.GetValue(key) is string valueStr && double.TryParse(valueStr, out double value))
19. {
20. return value;
21. }
22. return defaultValue;
23. }
24. }
25. private static void SetRegistryValue(string key, double value)
26. {
27. using (var regKey = Registry.CurrentUser.CreateSubKey(RegistryPath))
28. {
29. regKey?.SetValue(key, value.ToString());
30. }
31. }
32. }

### Код файлу (SettingsForm.cs)

1. public partial class SettingsForm : Form
2. {
3. public SettingsForm()
4. {
5. InitializeComponent();
6. numericUpDownRotationSpeedX.Value = (decimal)AppSettings.RotationSpeedX;
7. numericUpDownScaleFactor.Value = (decimal)AppSettings.ScaleFactor;
8. }
9. private void SettingsFormLoad(object sender, EventArgs e)
10. {
11. try
12. {
13. numericUpDownRotationSpeedX.Value = (decimal)AppSettings.RotationSpeedX;
14. numericUpDownScaleFactor.Value = (decimal)AppSettings.ScaleFactor;
15. Debug.WriteLine("Settings loaded successfully.");
16. }
17. catch (Exception ex)
18. {
19. MessageBox.Show($"Error loading settings: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
20. }
21. }
22. private void SettingsFormClosed(object sender, FormClosedEventArgs e)
23. {
24. try
25. {
26. AppSettings.RotationSpeedX = (double)numericUpDownRotationSpeedX.Value;
27. AppSettings.ScaleFactor = (double)numericUpDownScaleFactor.Value;
28. Debug.WriteLine("Settings saved successfully.");
29. }
30. catch (Exception ex)
31. {
32. MessageBox.Show($"Error saving settings: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
33. }
34. }
35. private void buttonSave\_Click(object sender, EventArgs e)
36. {
37. Close();
38. }
39. private void buttonCancel\_Click(object sender, EventArgs e)
40. {
41. Close();
42. }
43. }

### Код файлу (RenderControl.cs)

1. public partial class RenderControl : OpenGL
2. {
3. private double x = 0.1, y = 0.0, z = 0.0;
4. private double dt = 0.01;
5. private const double sigma = 10.0;
6. private const double rho = 28.0;
7. private const double beta = 8.0 / 3.0;
8. private double rotationX = 0.0;
9. private double rotationY = 0.0;
10. private double rotationSpeedX; // Швидкість повороту по X
11. private double rotationSpeedY = 0.3; // Фіксована швидкість повороту по Y
12. private double offsetX = 0.0;
13. private double offsetY = 0.0;
14. private double offsetZ = -30.0; // Зсув по Z (переміщення назад)
15. private double scaleFactor; // Масштабування системи
16. private double scaleSpeed = 0.01; // Швидкість зміни масштабу
17. private bool scalingUp = true; // Напрямок зміни масштабу
18. private List<(double x, double y, double z)> trajectory = new List<(double x, double y, double z)>();
19. public RenderControl()
20. {
21. InitializeComponent();
22. }
23. private void OnContextCreated(object sender, EventArgs e)
24. {
25. glClearColor(Color.Black);
26. try
27. {
28. rotationSpeedX = AppSettings.RotationSpeedX;
29. scaleFactor = AppSettings.ScaleFactor;
30. Debug.WriteLine("Settings applied to RenderControl.");
31. }
32. catch (Exception ex)
33. {
34. Debug.WriteLine($"Error loading settings in RenderControl: {ex.Message}");
35. }
36. }
37. private void OnRender(object sender, EventArgs e)
38. {
39. glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);
40. glLoadIdentity();
41. glViewport(0, 0, Width, Height);
42. glOrtho(-10, 10, -10, 10, -1, 1);
43. glMatrixMode(GL\_PROJECTION);
44. glLoadIdentity();
45. gluPerspective(45.0, (double)Width / Height, 1.0, 120.0);
46. glMatrixMode(GL\_MODELVIEW);
47. glLoadIdentity();
48. gluLookAt(0.0, 0.0, 100.0, offsetX, offsetY, offsetZ, 0.0, 1.0, 0.0);
49. glRotated(rotationX, 1.0, 0.0, 0.0);
50. glRotated(rotationY, 0.0, 1.0, 0.0);
51. DrawLorenz();
52. UpdateLorenz();
53. rotationX += rotationSpeedX;
54. glFlush();
55. }
56. private void DrawLorenz()
57. {
58. glBegin(GL\_LINE\_STRIP);
59. foreach (var (px, py, pz) in trajectory)
60. {
61. glColor3d((px + 20) / 40.0, (py + 30) / 60.0, (pz + 50) / 100.0);
62. glVertex3d((px \* scaleFactor) + offsetX, (py \* scaleFactor) + offsetY, (pz \* scaleFactor) + offsetZ);
63. }
64. glEnd();
65. }
66. private void UpdateLorenz()
67. {
68. double dx = sigma \* (y - x) \* dt;
69. double dy = (x \* (rho - z) - y) \* dt;
70. double dz = (x \* y - beta \* z) \* dt;
71. x += dx;
72. y += dy;
73. z += dz;
74. trajectory.Add((x, y, z));
75. if (trajectory.Count > 3000)
76. {
77. trajectory.RemoveAt(0);
78. }
79. }
80. private void timer\_Tick(object sender, EventArgs e)
81. {
82. rotationX += rotationSpeedX;
83. rotationY += rotationSpeedY;
84. if (rotationX >= 360.0) rotationX -= 360.0;
85. if (rotationY >= 360.0) rotationY -= 360.0;
86. if (scalingUp)
87. {
88. scaleFactor += scaleSpeed;
89. if (scaleFactor >= 3.0)
90. {
91. scalingUp = false;
92. }
93. }
94. else
95. {
96. scaleFactor -= scaleSpeed;
97. if (scaleFactor <= 0.2)
98. {
99. scalingUp = true;
100. }
101. }
102. Invalidate();
103. }
104. }