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 631п

напряму підготовки (спеціальності):

*121 інженерія програмного забезпечення*

(шифр і назва напряму підготовки / спеціальності)

*Гордеюк Р.О.*

(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](#_Toc185273436)

[Завдання, варіант № 7 4](#_Toc185273437)

[Системна інформація 4](#_Toc185273438)

[Теоретичні відомості 4](#_Toc185273439)

[Результати виконання практичної роботи 6](#_Toc185273440)

[Практична робота 2. Графічні примітиви OpenGL 8](#_Toc185273441)

[Завдання, варіант № 7 8](#_Toc185273442)

[Системна інформація 8](#_Toc185273443)

[Теоретичні відомості 8](#_Toc185273444)

[Результати виконання практичної роботи 10](#_Toc185273445)

[Практична робота 3. Графік функції однієї змінної 13](#_Toc185273446)

[Завдання, варіант № 7 13](#_Toc185273447)

[Теоретичні відомості 13](#_Toc185273448)

[Результати виконання практичної роботи 14](#_Toc185273449)

[Практична робота 4. Криві другого порядку 17](#_Toc185273450)

[Завдання, варіант № 7 17](#_Toc185273451)

[Теоретичні відомості 17](#_Toc185273452)

[Результати виконання практичної роботи 17](#_Toc185273453)

[Практична робота 5. Афінні перетворення у просторі 21](#_Toc185273454)

[Завдання, варіант № 7 21](#_Toc185273455)

[Теоретичні відомості 21](#_Toc185273456)

[Результати виконання практичної роботи 22](#_Toc185273457)

[Практична робота 6. Візуалізація прямої задачі кінематики 25](#_Toc185273458)

[Завдання, варіант № 7 25](#_Toc185273459)

[Теоретичні відомості 25](#_Toc185273460)

[Програмна реалізація кінематичної схеми 25](#_Toc185273461)

[Результати виконання практичної роботи 30](#_Toc185273462)

[Практична робота 7. Екранна заставка з анімацією 33](#_Toc185273463)

[Завдання 33](#_Toc185273464)

[Теоретичні відомості 33](#_Toc185273465)

[Результати виконання практичної роботи 33](#_Toc185273466)

[Загальний перелік посилань 35](#_Toc185273467)

[Додаток А. Лістинг програми до практичної роботи №1 36](#_Toc185273468)

[Додаток Б. Лістинг програми до практичної роботи №2 38](#_Toc185273469)

[Додаток В. Лістинг програми до практичної роботи №3 42](#_Toc185273470)

[Додаток Г. Лістинг програми до практичної роботи №4 47](#_Toc185273471)

[Додаток Г. Лістинг програми до практичної роботи №5 55](#_Toc185273472)

[Додаток Д. Лістинг програми до практичної роботи №6 65](#_Toc185273473)

[Додаток Е. Лістинг програми до практичної роботи №7 72](#_Toc185273474)

# Практична робота 1. Основні принципи роботи з OpenGL

## Завдання, варіант № 7

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.

|  |  |  |
| --- | --- | --- |
| 7 | Примітиви:  GL\_POINTS, GL\_LINES  x1 = -8; x2 = 10  y1 = -2; y2 = 6 |  |

## Системна інформація

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

Processor AMD Ryzen 5 5600 6-Core Processor 3.50 GHz

RAM 32.0 GB (31.9 GB usable)

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

Edition Windows 11 Home Version 23H2

IDE Microsoft Visual Studio Enterprise 2022 (64-bit) version 17.11.4

## Теоретичні відомості

### Вершинні масиви

Використання вершинних масивів зменшує кількість викликів функцій та надмірне використання спільних вершин. Таким чином, ви можете збільшити продуктивність рендерингу[[1]](#footnote-1).

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, glColor3ub() рядок 19 у файлі FirstTasks.cs;

тип, glLineStipple(), glEnable()/glDisable(), рядок 17 у файлі FirstTasks.cs;

товщина glLineWidth(), рядок 45 у файлі FirstTasks.cs

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

Розроблення підпрограм для виключення дублювання коду наведено у рядках 34 – 59 файлу FirstTasks.cs

Застосування циклів для створення зображень наведено у рядках 21 – 30 файлу FirstTasks.cs.

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

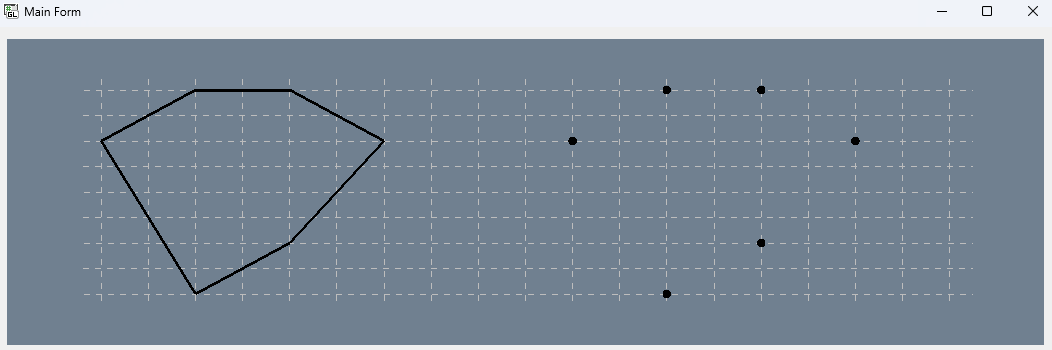


Рисунок 1.1 – Тестування програми при зміні ширини вікна

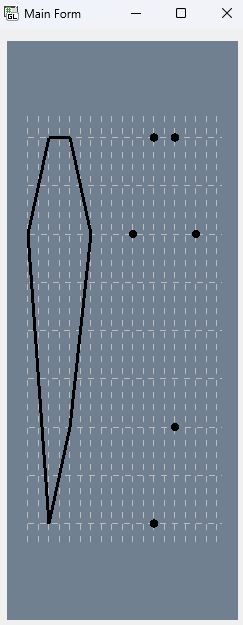


Рисунок 1.2 – Тестування програми при зміні висоти вікна

### Контроль виконання вимог та елементів завдання

В результаті виконання практичної роботи були повністю виконані елементи базового рівня та частково 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 № 7

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. 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. Крім цього, користувач повинен мати можливість змінювати режим відображення графічних примітивів OpenGL: точкове (тільки вершини фігури), контурне і з заповненням кольором. 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.

|  |  |  |
| --- | --- | --- |
| 7 | Сторона фігури a = 50  Примітиви:   GL\_TRIANGLE\_STRIP,  GL\_POLYGON |  |

## Системна інформація

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

Processor AMD Ryzen 5 5600 6-Core Processor 3.50 GHz

RAM 32.0 GB (31.9 GB usable)

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

Edition Windows 11 Home Version 23H2

IDE Microsoft Visual Studio Enterprise 2022 (64-bit) version 17.11.4

## Теоретичні відомості

Графічні примітиви OpenGL — це основні елементи для побудови 2D та 3D графіки. OpenGL надає кілька видів примітивів, таких як точки, лінії, трикутники та багатокутники, за допомогою яких можна створювати складні геометричні об'єкти.

### Основні типи графічних примітивів:

* GL\_POINTS (Точки):

Використовуються для відображення окремих точок на екрані.

Кожна вершина, визначена у вершинному масиві, є точкою.

* GL\_LINES (Лінії):

Дві вершини створюють лінію.

Можна використовувати для побудови лінійних сегментів між точками.

* GL\_LINE\_STRIP і GL\_LINE\_LOOP (Лінійні смуги та петлі):

Використовують послідовність вершин для створення кількох підряд з'єднаних ліній.

Відрізняються тим, що у GL\_LINE\_LOOP остання вершина з'єднується з першою.

* GL\_TRIANGLES (Трикутники):

Найбільш використовуваний примітив для відображення поверхонь.

Кожен трикутник визначається трьома вершинами.

Також є варіанти: GL\_TRIANGLE\_STRIP і GL\_TRIANGLE\_FAN, які дозволяють оптимізувати кількість вершин.

* GL\_POLYGON (Багатокутники):

Дозволяє будувати багатокутники з будь-якою кількістю вершин, починаючи з трьох.

Використовується для заповнених поверхонь.

### Команди для роботи з графічними примітивами в OpenGL:

* glBegin() і glEnd():

Команди, які огортають блоки коду, що описують примітиви.

Наприклад, glBegin(GL\_TRIANGLES) оголошує, що починається блок трикутників, а кожна трійка вершин буде малювати один трикутник.

* glVertex():

Визначає координати вершин для кожного примітиву.

Наприклад, glVertex2f(x, y) задає двовимірні координати вершини.

* glColor():

Додає колір до вершин або примітивів.

### Способи малювання примітивів:

* glDrawArrays():

Команда, яка дозволяє малювати примітиви, використовуючи масиви вершин.

Приймає три параметри: тип примітиву, індекс першої вершини і кількість вершин.

Це один з найефективніших способів малювання, оскільки скорочує кількість викликів функцій.

* glDrawElements():

Дозволяє малювати примітиви, використовуючи індексовані масиви вершин.

Дозволяє уникати дублювання спільних вершин для складних фігур.

### Управління масштабом та координатною системою:

* glOrtho() і gluOrtho2D():

Використовуються для встановлення проекції, яка контролює, як об'єкти відображаються на екрані.

* gluOrtho2D(left, right, bottom, top) встановлює двовимірну ортогональну проекцію, де left, right, bottom, і top — це межі координатної площини.
* glViewport():

Визначає частину вікна, на якій буде відображатися зображення.

Це дозволяє масштабувати або змінювати розмір області, де рендеряться примітиви.

## Результати виконання практичної роботи

### Розв'язання завдання

У даній практичній роботі було розроблено застосунок з використанням бібліотеки OpenGL для відображення правильного багатокутника та можливості замощення області екрану користувачем. Програма реалізована з використанням команд OpenGL для керування примітивами, налаштування координатної системи, відображення фігур та взаємодії з користувачем.

Налаштування координатної системи:

Використовуються функції glOrtho() та glViewport() для встановлення ізотропної системи координат, що дозволяє відображати багатокутники в центрі вікна незалежно від розмірів області рендерингу. Параметри масштабування визначаються розмірами фігури та кількістю плиток, які буде відображено на екрані.

Коректне відображення завдання під час змінення розмірів/положення вікна наведено у рис. 2.1 та 2.2

Відображення багатокутника:

Для відображення правильного багатокутника використано примітиви GL\_TRIANGLE\_STRIP та GL\_POLYGON. Після старту програми у робочій області відображається одна плитка. Розмір плитки визначено згідно з варіантом, де сторона фігури дорівнює 50.

Реалізовано три режими відображення фігур:

* Точковий режим (відображення лише вершин фігури) за допомогою примітиву GL\_POINTS.
* Контурний режим (відображення лише контуру фігури) за допомогою примітиву GL\_LINE\_LOOP.
* Режим із заливкою (заповнення кольором) за допомогою примітиву GL\_TRIANGLE\_STRIP та GL\_POLYGON.

Колірна схема:

Для зафарбування фігур використано шість кольорів: білий, сірий (35%), червоний, зелений, синій та жовтий. Фарба накладається відповідно до положення багатокутників на екрані.

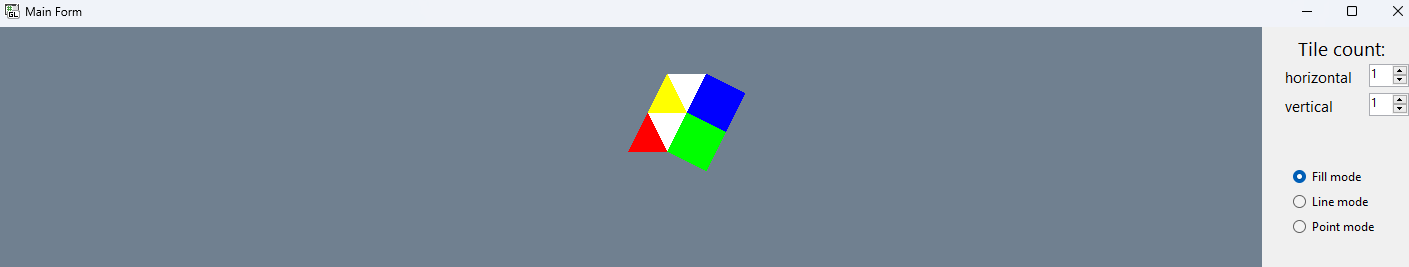


Рисунок 2.1 – Тестування програми при зміні ширини вікна

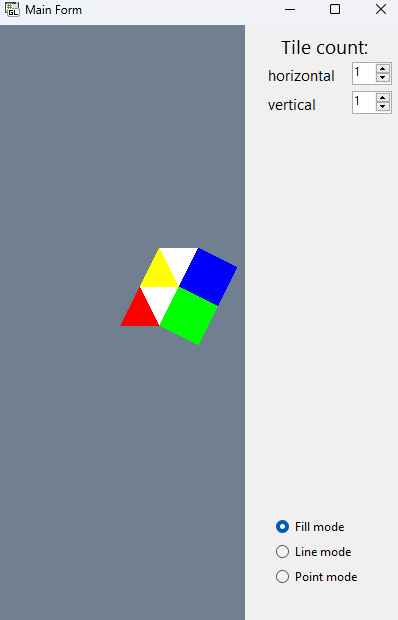


Рисунок 2.2 – Тестування програми при зміні висоти вікна

### Контроль виконання вимог та елементів завдання

В результаті виконання практичної роботи були повністю виконані елементи базового рівня та 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. Графік функції однієї змінної

## Завдання, варіант № 7

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:

* 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 f(x)=0, якщо вони є на завданому інтервалі від Xmin до Xmax.

Для підвищеного рівня складності необхідно додатково реалізувати коректне виведення функції f2x з урахуванням області визначення функції і відобразити лінії розриву.

|  |  |
| --- | --- |
| 7 | Функція |
|  |
|  |

## Теоретичні відомості

У комп'ютерній графіці графіки функцій будуються за допомогою бібліотек, таких як OpenGL. Основні кроки для побудови графіка функції:

Визначення інтервалу та кількості точок. Для побудови графіка функції необхідно обрати інтервал *Xmin* до *Xmax*, на якому буде будуватись графік, і визначити кількість точок для обчислення значень.

Масштабування. Автоматичне чи ручне визначення меж за віссю Y є важливим для коректного відображення графіка. Для цього можна використовувати функції OpenGL, такі як gluOrtho2D().

Відображення осей координат. Використовується для розміщення координатної сітки та маркерів значень.

Робота з точками перетину. Необхідно знайти точки, де функція перетинає вісь абсцис, і відобразити їх на графіку.

Лінії розриву. При наявності розривів в області визначення функції, програма має коректно відображати ці точки.

## Результати виконання практичної роботи

### Розв'язання завдання

Для реалізації завдання було розроблено програму, що будує графік функцій та на заданому інтервалі з можливістю масштабування і ручного налаштування осей. Зокрема, програма:

* Дозволяє користувачу задавати інтервал для осі 𝑋 і автоматично масштабує вісь 𝑌.
* Відображає осі координат, точки перетину функції з віссю абсцис та координатну сітку.
* Коректно відображає функцію з урахуванням ліній розриву.

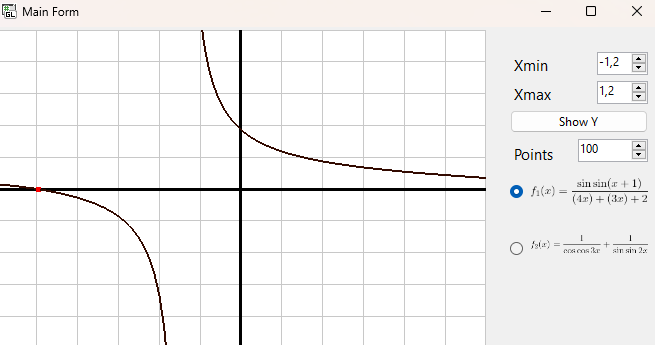


Рисунок 3.1 – Тестування роботи програми

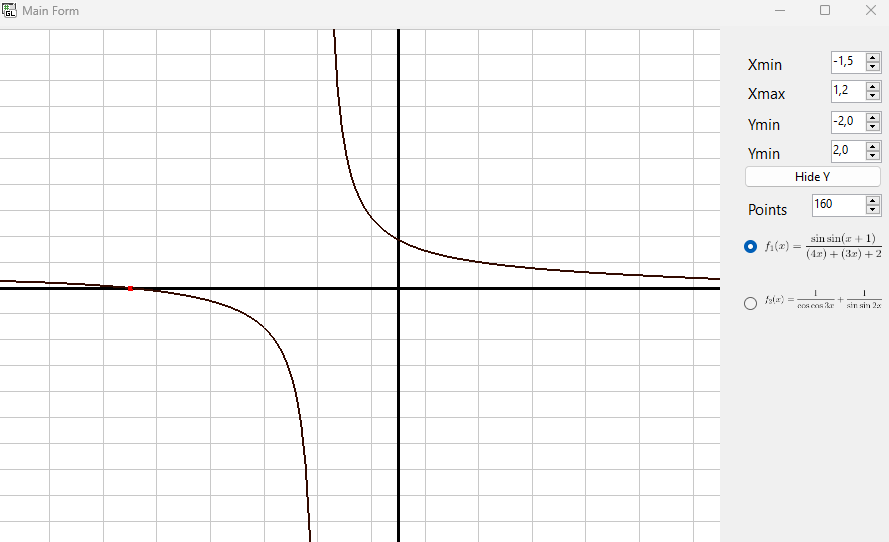


Рисунок 3.2 – Тестування роботи програми при зміні параметрів і розміру вікна

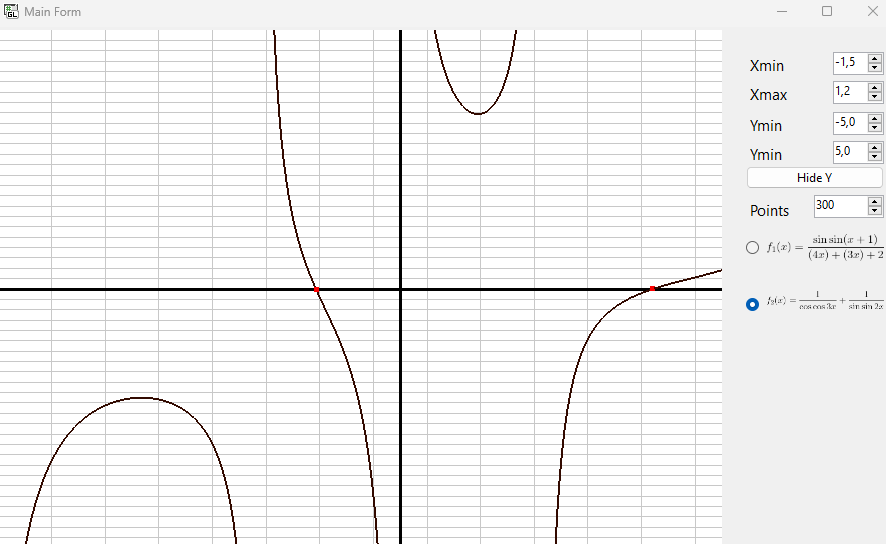


Рисунок 3.3 – Тестування роботи програми при зміні параметрів, функції і розміру вікна

### 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 3.1.

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

# Практична робота 4. Криві другого порядку

## Завдання, варіант № 7

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.

|  |  |
| --- | --- |
| 7 | Еліпс «+» – явне подання; |
| Парабола «++» – явне подання. |

## Теоретичні відомості

Кожну криву другого порядку можна подати як послідовність відрізків. У цьому випадку перетин кривої другого порядку та довільного відрізка можна розглядати як пошук спільної точки двох відрізків , та , , заданих у параметричному вигляді (один з яких є фрагментом кривої). Цю задачу можна подати у вигляді системи, що складається з двох лінійних рівнянь з невідомими параметрами першого і другого відрізків:

І за умови що результат рішення буде задовольняти наступні дві умови: і , інакше відрізки або паралельні, або перетинаються тільки прямі, на яких вони лежать .

## Результати виконання практичної роботи

### Розв'язання завдання

Для реалізації завдання було розроблено програму, що малює еліпс або параболу на заданому інтервалі з можливістю вказання точок, що формують відрізок, в області графічного виведення програми за допомогою маніпулятора «миш».

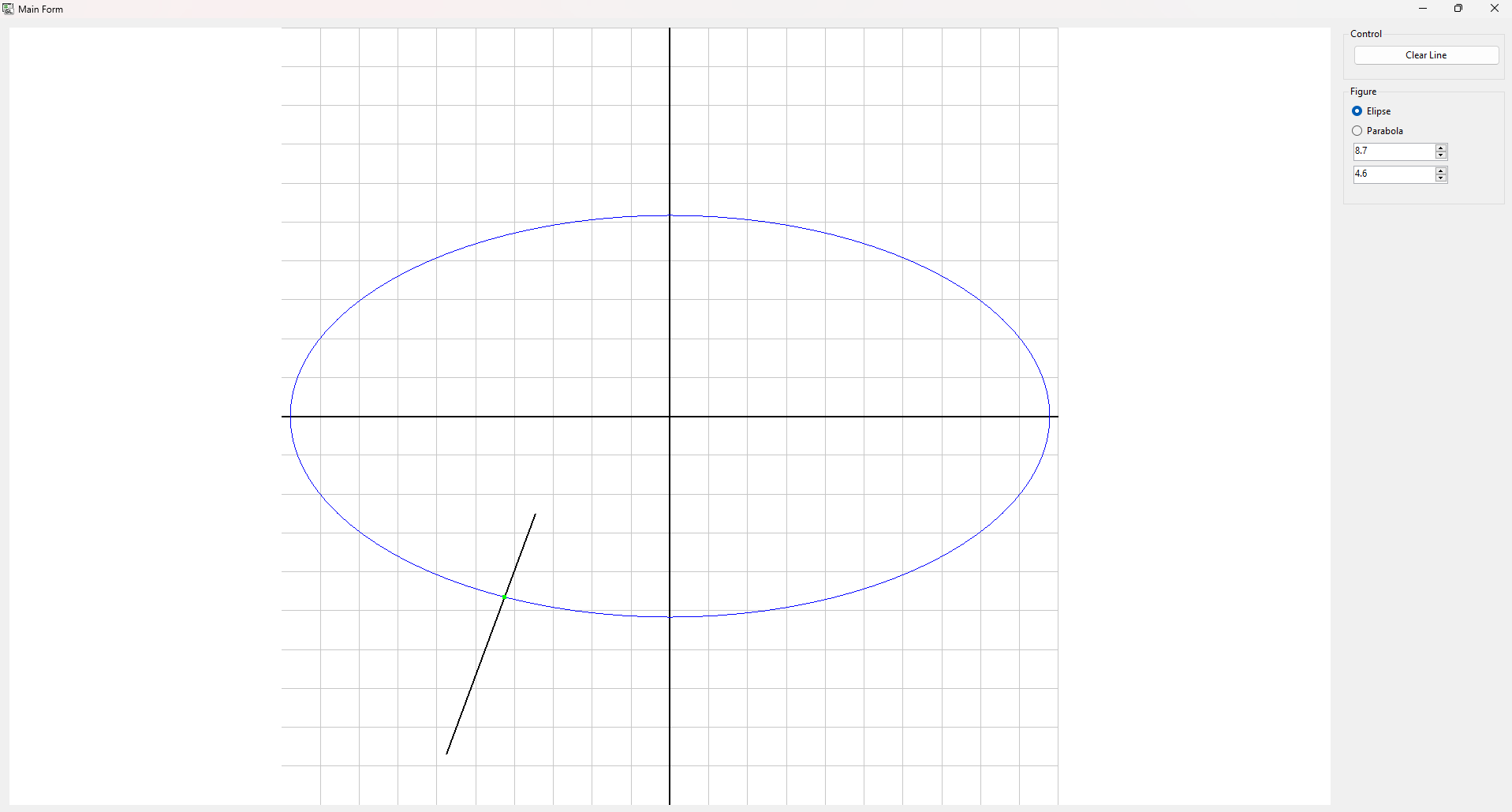


Рисунок 4.1 – Тестування роботи програми коли в нас еліпс та ширина більше за висоту

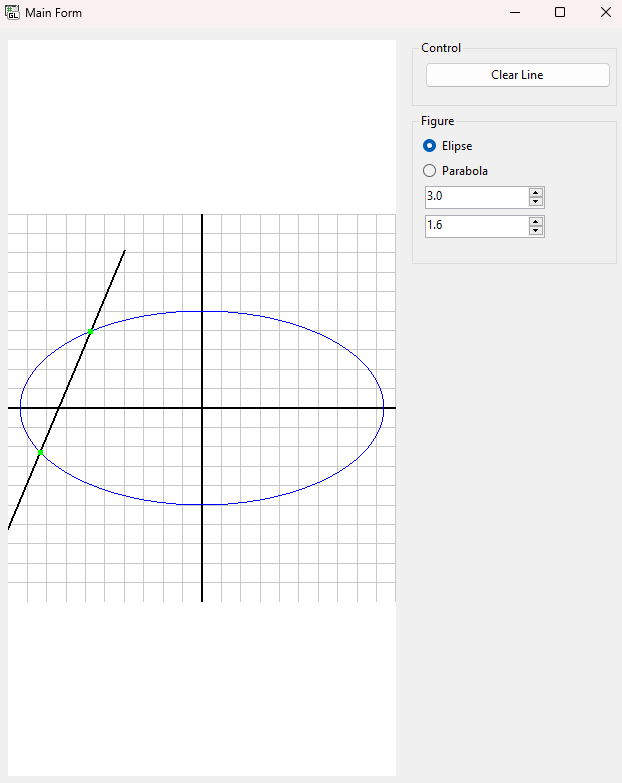


Рисунок 4.2 – Тестування роботи програми коли в нас еліпс та висота більше за ширину

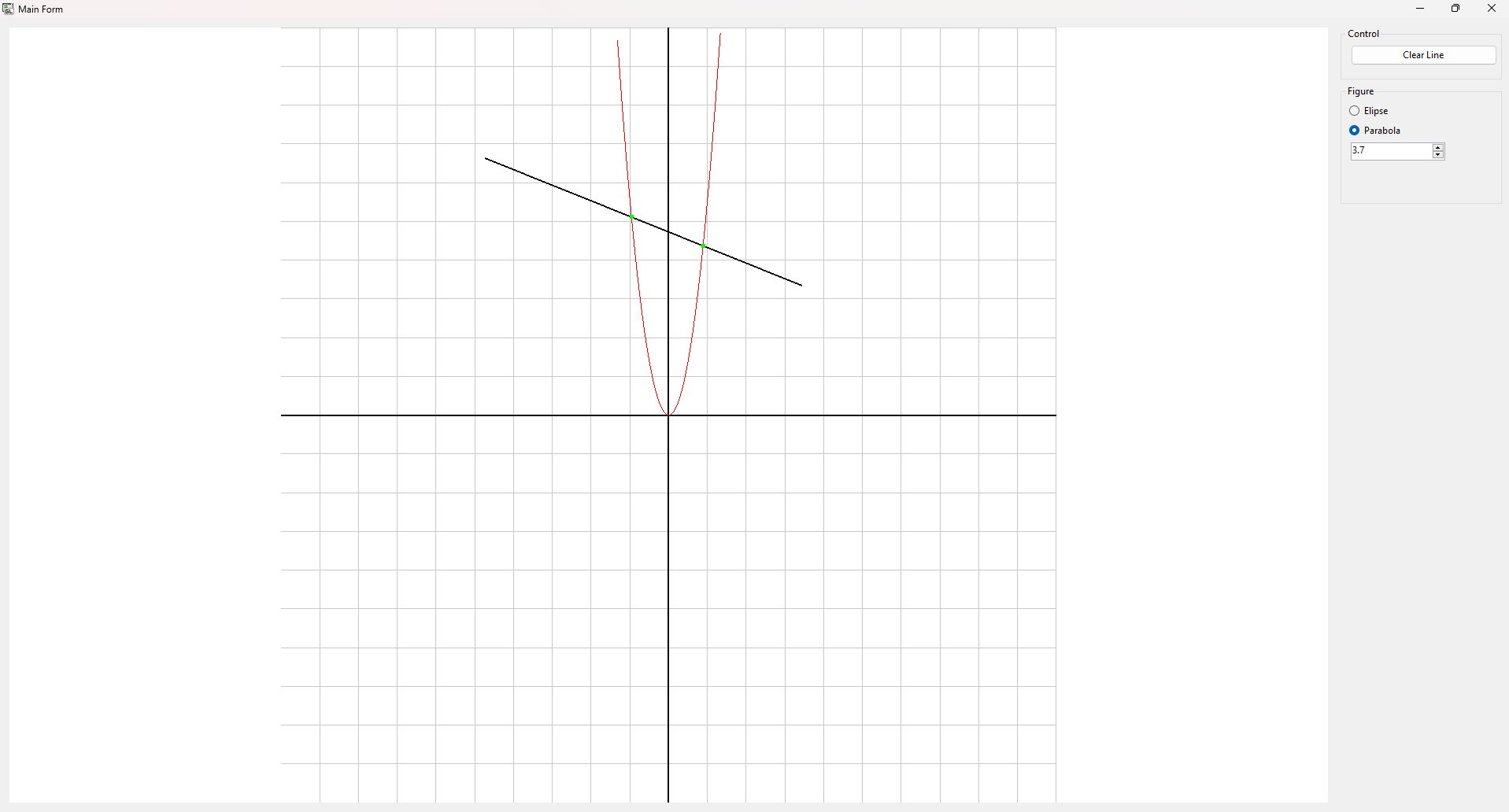


Рисунок 4.3 – Тестування роботи програми коли в нас парабола та ширина більше за висоту

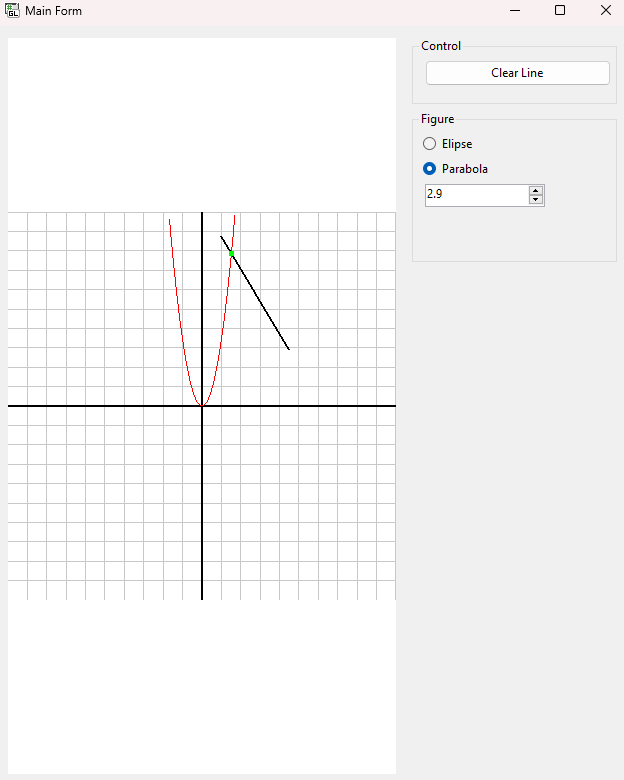


Рисунок 4.4 – Тестування роботи програми коли в нас парабола та висота більше за ширину

### 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 4.1.

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

# Практична робота 5. Афінні перетворення у просторі

## Завдання, варіант № 7

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 [1, 2]. Параметри деталізації об’єктів (slices, stacks), кольору, товщини і типу ліній обирають самостійно.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | X0Z | Сфера | ∥0Z | +2.5 | -0.5 | -2.5 | 1.0 | - | - |
| Циліндр | ⇅0Y | -3.5 | -2.5 | +2.5 | 0.0 | 2.5 | 2.0 |
| Диск | ∥0X | +4.5 | +3.5 | +4.5 | 4.0 | 1.5 | - |

## Теоретичні відомості

Ізотропну систему координат можна встановлювати двома способами.

В першому випадку за допомогою команди glViewport задають робочу область відповідного меншого значення ширини/висоти вікна, а у другому – вводять множник (або дільник), що коректує і рівний відношенню ширини і висоти вікна під час встановлення системи координат (наприклад, командою glOrtho). Глибину потрібно задавати таким чином, щоб за будь-якого розміщення заданої сцени усі об’єкти знаходилися всередині видимої області.

Для відображення квадратичних примітивів використовують точку прив’язки об’єкта. В загальному випадку за відсутності афінних перетворень ця точка прив’язки об’єкта збігається з початком системи координат. Точкою прив’язки для сфери і диска (повного і часткового) виступає їхній центр, а для циліндра (конуса) – центр однієї з основ. Для розміщення кожного з трьох квадратичних об’єктів відповідно варіанту завдання необхідно скористатися одним з афінних перетворень або їхніх комбінацій: поворот, перенос, масштабування. Після виконання перетворень точка прив’язки повинна знаходитися на координатах x0, y0, z0, а вісь об’єкта паралельна (∥) для сфери/диска або колінеарна (⇈, ⇅) для циліндра / конуса з урахуванням варіанта.

## Результати виконання практичної роботи

### Розв'язання завдання

Для реалізації завдання було розроблено програму, має дві системи відображення трьох вимірного простору, де є сітка та лінії для показу осей. Також в просторі знаходяться три фігури по завданню, з можливістю вмикання/вимикання освітлення.

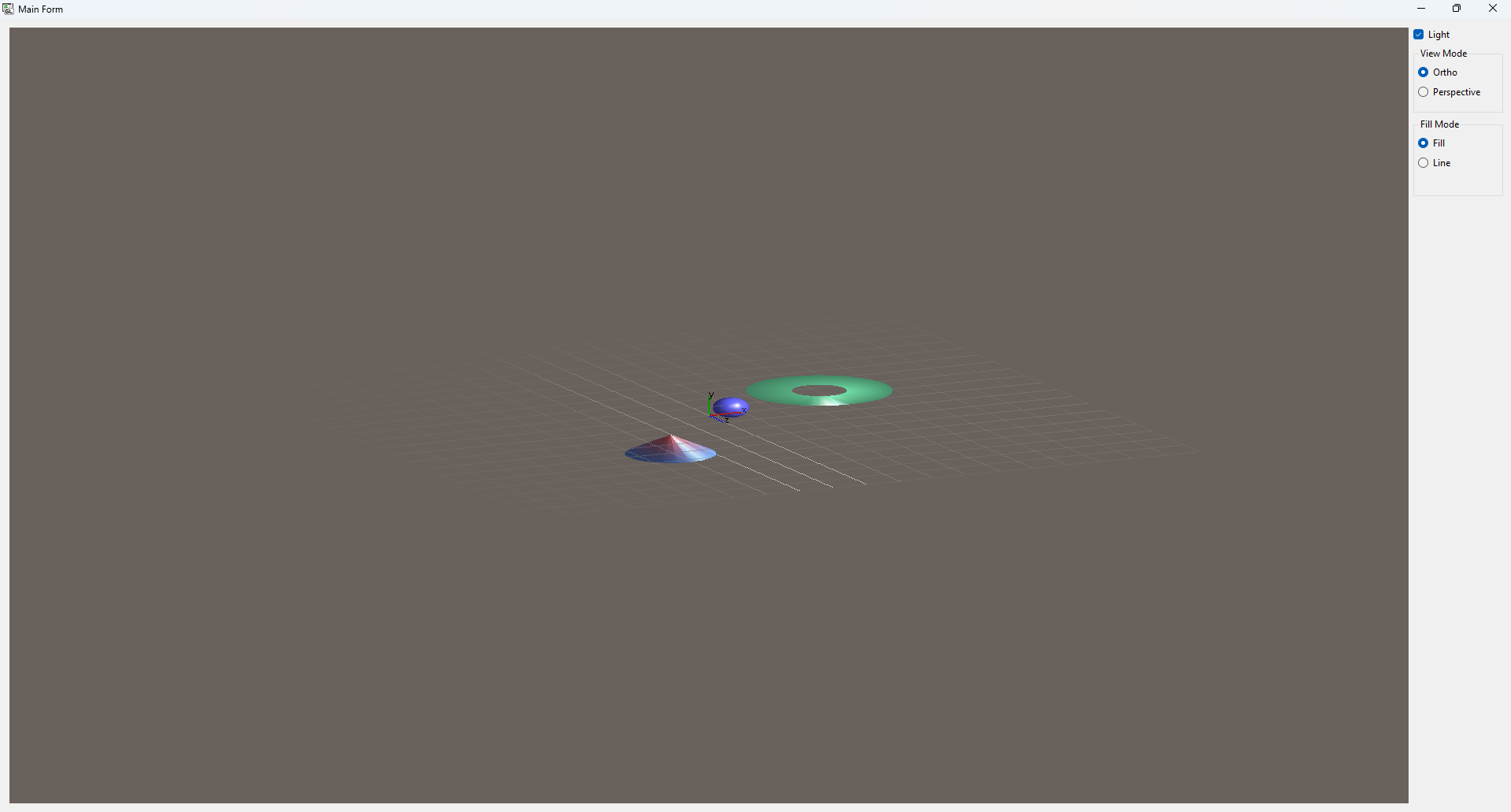


Рисунок 5.1 – Тестування роботи програми в ортогональному відображенні

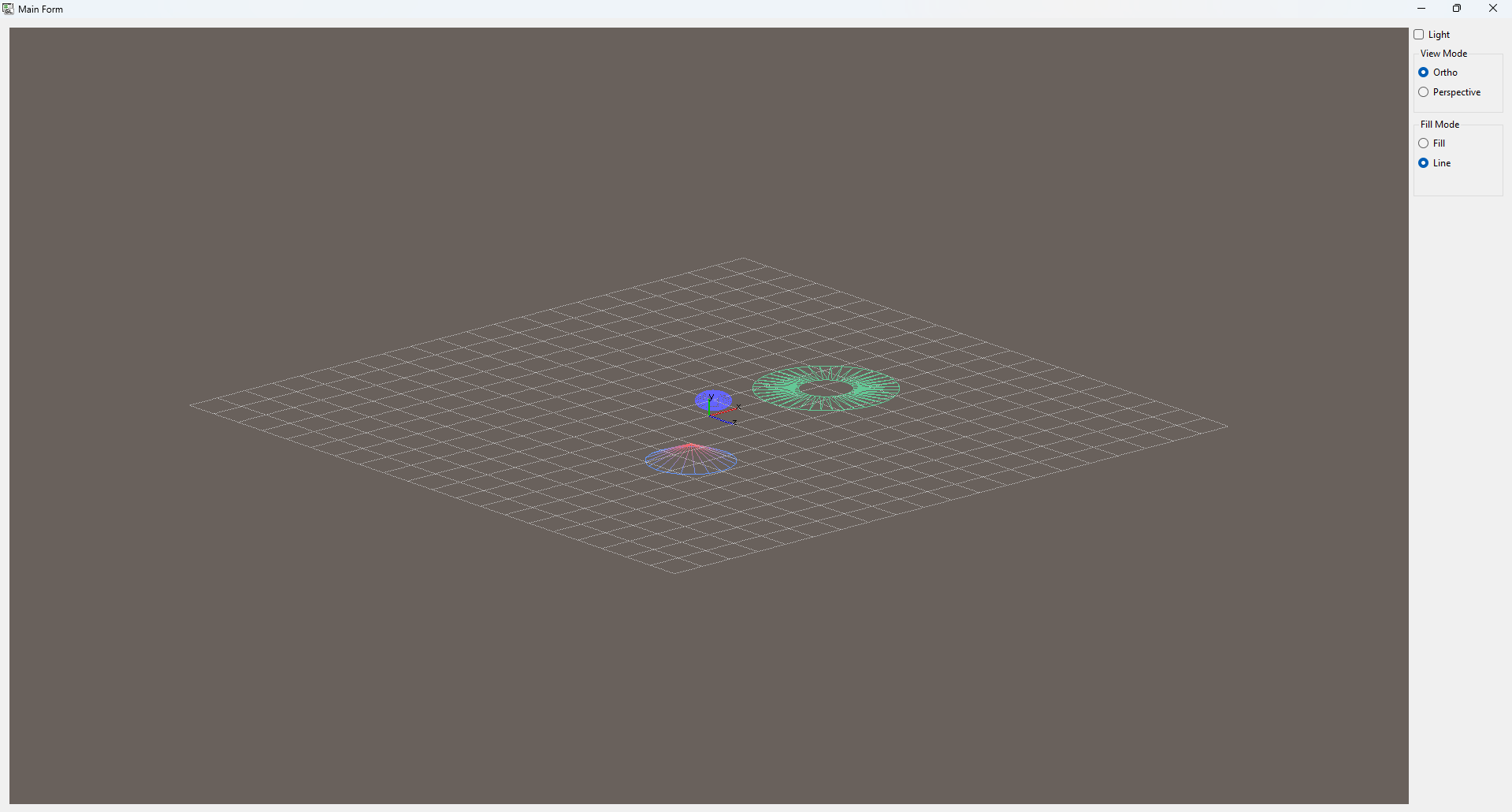


Рисунок 5.2 – Тестування роботи програми в ортогональному відображенні

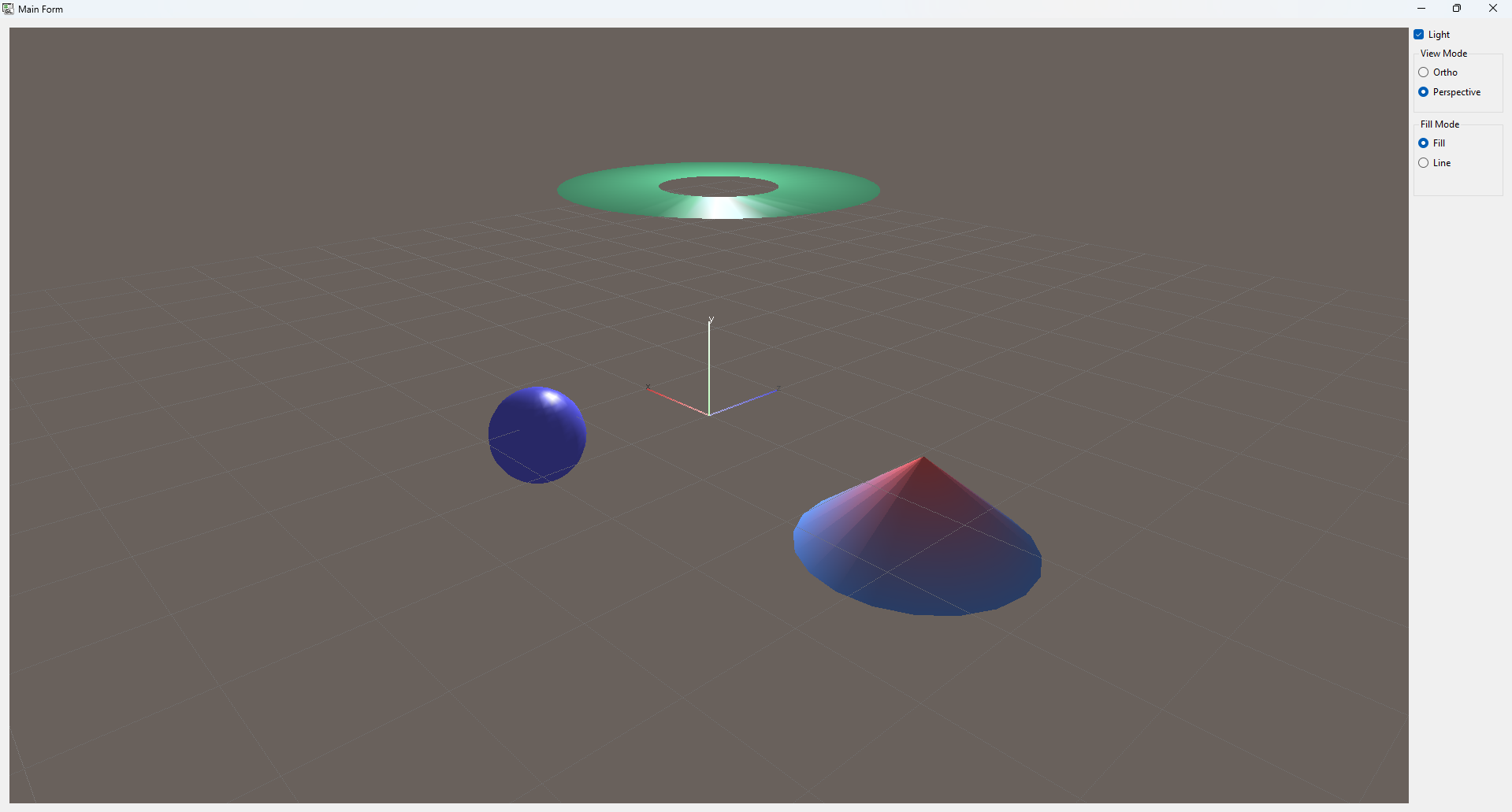


Рисунок 5.3 – Тестування роботи програми в перспективному відображенні

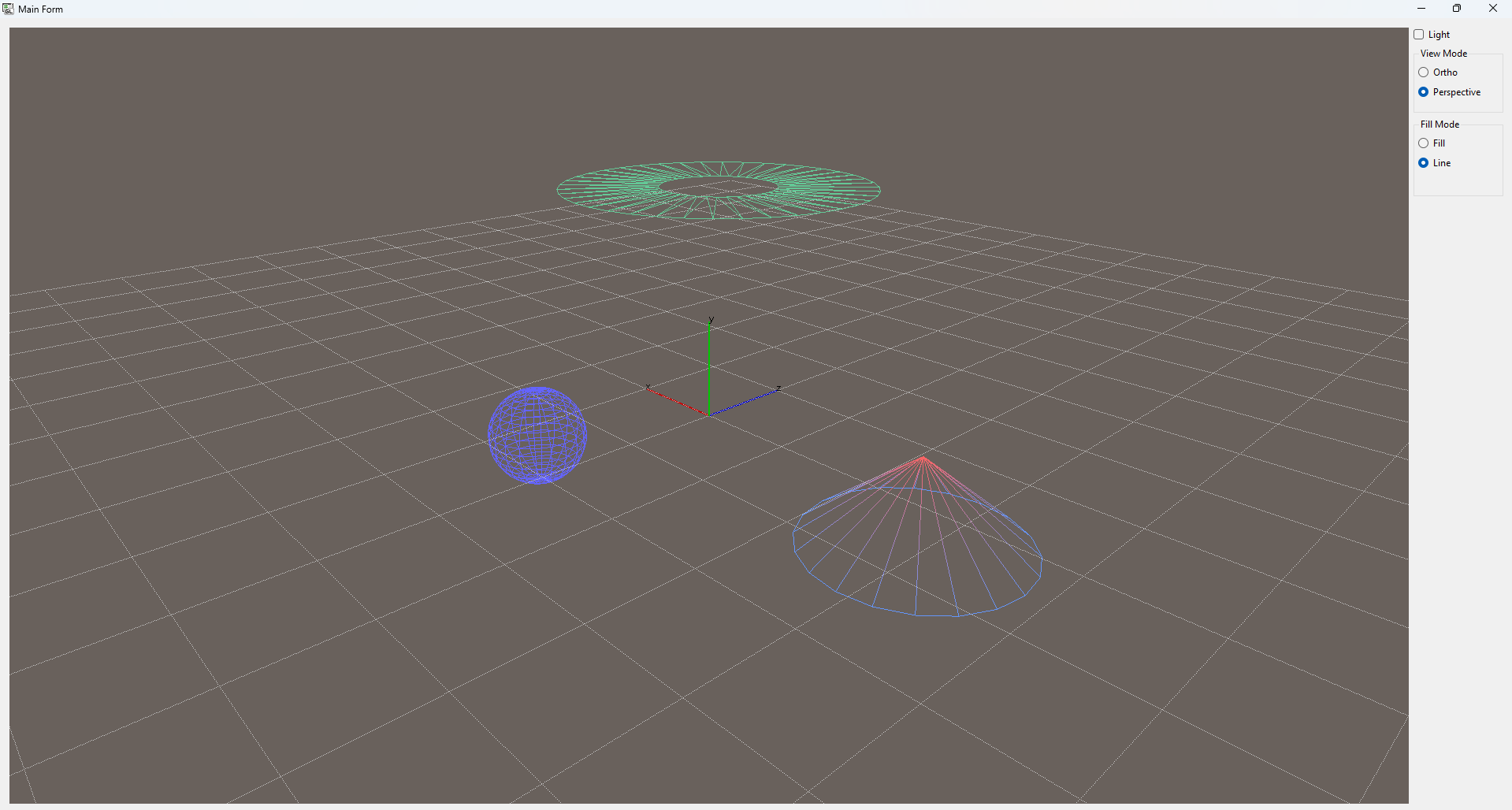


Рисунок 5.4 – Тестування роботи програми в перспективному відображенні

### 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 5.1.

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

# Практична робота 6. Візуалізація прямої задачі кінематики

## Завдання, варіант № 7

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.

|  |  |  |
| --- | --- | --- |
| 7 | a = 0.9  b = 0.24  c = 0.64 |  |

## Теоретичні відомості

Для спрощення обчислень завдання спочатку розглядають на площині XOY. Потім, якщо додати до двомірного рішення обертання навколо осі OY і масштабування, можна одержати повноцінну тривимірну модель.

Для вирішення задачі на площині необхідна комбінація з двох афінних перетворень – обертання та перенесення. Таким чином, слід поетапно підібрати свій набір афінних перетворень для кожного сегмента маніпулятора.

Крім цього, деякі точки з’єднання та точка ковзання формують трикутник, кути якого залежать від значення параметра S. Ці кути обчислюють на основі теореми косинусів.

## Програмна реалізація кінематичної схеми

### Вихідні дані

Вихідні дані кінематичної схеми задано основними значеннями a, b, c, які можуть бути представлені звичайними змінними безпосередню операцію присвоєння, що реалізується за допомогою оголошення цих значень як властивостей:

public double sideA { get; set; } = 0.9;

public double sideB { get; set; } = 0.24;

public double sideC { get; set; } = 0.64;

public double PointBY { get; set; } = 0.4;

public double RotationAngleY { get; set; } = 0.0;

### Створення і використання перспективної проекції

glEnable(GL\_DEPTH\_TEST);

glViewport(0, 0, Width, Height);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

double aspect = (double)Width / Height;

gluPerspective(45.0, aspect, 0.1, 100.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glTranslated(PanOffsetX, PanOffsetY, 0.0);

double cameraX = CameraRadius \* Math.Sin(CameraPhi) \* Math.Cos(CameraTheta);

double cameraY = CameraRadius \* Math.Cos(CameraPhi);

double cameraZ = CameraRadius \* Math.Sin(CameraPhi) \* Math.Sin(CameraTheta);

gluLookAt(

Multi \* cameraX, Multi \* cameraY, Multi \* cameraZ,

0.0, 0.0, 0.0,

0.0, 1.0, 0.0

);

### Використання маніпулятора для управління моделлю

Для організації інтерфейсу з користувачем задіємо клавіатуру і маніпулятор «миша». Керування системою координат і масштабом зв'яжемо з подіями маніпулятора «миша».

private void OnMoudeDown(object sender, MouseEventArgs e)

{

if (e.Button == MouseButtons.Right && (Control.ModifierKeys & Keys.Control) == Keys.Control)

{

renderControl1.FigureRotating = true;

renderControl1.FigureChanging = false;

renderControl1.LastRightMouseX = e.X;

renderControl1.NotRotating = true;

}

if (e.Button == MouseButtons.Left && (Control.ModifierKeys & Keys.Control) == Keys.Control)

{

renderControl1.FigureChanging = true;

renderControl1.LastLeftMouseY = e.Y;

renderControl1.NotRotating = true;

}

else

{

if (renderControl1.NotRotating && e.Button == MouseButtons.Left)

{

renderControl1.NotRotating = false;

renderControl1.FigureChanging = false;

renderControl1.LastLeftMouseX = e.X;

renderControl1.LastLeftMouseY = e.Y;

}

}

renderControl1.Invalidate();

}

private void OnMouseUp(object sender, MouseEventArgs e)

{

if (e.Button == MouseButtons.Right && (Control.ModifierKeys & Keys.Control) == Keys.Control)

{

renderControl1.FigureRotating = false;

renderControl1.FigureChanging = false;

renderControl1.NotRotating = true;

}

if (e.Button == MouseButtons.Left && (Control.ModifierKeys & Keys.Control) == Keys.Control)

{

renderControl1.FigureChanging = false;

renderControl1.FigureRotating = false;

renderControl1.NotRotating = true;

}

else

{

if (!renderControl1.NotRotating && e.Button == MouseButtons.Left)

{

renderControl1.NotRotating = true;

}

}

renderControl1.Invalidate();

}

private void OnMouseMove(object sender, MouseEventArgs e)

{

if (e.Button == MouseButtons.Right && (Control.ModifierKeys & Keys.Control) == Keys.Control)

{

if (renderControl1.FigureRotating)

{

double deltaX = (e.X - renderControl1.LastRightMouseX) \* 0.5;

renderControl1.RotationAngleY = renderControl1.RotationAngleY - deltaX;

renderControl1.LastRightMouseX = e.X;

renderControl1.Invalidate();

}

}

if (e.Button == MouseButtons.Left && (Control.ModifierKeys & Keys.Control) == Keys.Control)

{

if (renderControl1.FigureChanging)

{

double deltaY = (e.Y - renderControl1.LastLeftMouseY) \* 0.01;

renderControl1.PointBY = Math.Clamp(renderControl1.PointBY - deltaY, 0.05, 0.48);

renderControl1.LastLeftMouseY = e.Y;

renderControl1.Invalidate();

}

}

if (!renderControl1.NotRotating)

{

double deltaX = e.X - renderControl1.LastLeftMouseX;

double deltaY = e.Y - renderControl1.LastLeftMouseY;

renderControl1.CameraTheta += deltaX \* 0.01;

renderControl1.CameraPhi -= deltaY \* 0.01;

renderControl1.CameraPhi = Math.Clamp(renderControl1.CameraPhi, 0.1, Math.PI - 0.1);

renderControl1.LastLeftMouseX = e.X;

renderControl1.LastLeftMouseY = e.Y;

renderControl1.Invalidate();

}

}

Для управління масштабуванням скористаємося колесом прокрутки, використовуючи значення e.Delta як прирощення масштабу:

private void OnWheel(object sender, MouseEventArgs e)

{

if (e.Delta < 0)

{

renderControl1.Multi += 0.1;

}

else

{

if (renderControl1.Multi > 0.2)

{

renderControl1.Multi -= 0.1;

}

}

renderControl1.Invalidate();

}

### Осі координат

Для підвищення загальної наочності та спрощення орієнтації елементів кінематичної схеми можна використовувати зображення осей координат у такий спосіб:

public void DrawCoordinatesLines(double x, double y, double z)

{

double size = 0.4;

glDisable(GL\_LIGHTING);

glDisable(GL\_LIGHT0);

glDisable(GL\_COLOR\_MATERIAL);

glLineWidth(2.5f);

glBegin(GL\_LINES);

glColor3ub(200, 0, 0);

glVertex3d(x, y, z);

glVertex3d(x + size, y, z);

glColor3ub(0, 200, 0);

glVertex3d(x, y, z);

glVertex3d(x, y + size, z);

glColor3ub(0, 0, 200);

glVertex3d(x, y, z);

glVertex3d(x, y, z + size);

glEnd();

glColor3ub(0, 0, 0);

Print("x", x + size, y, z);

Print("y", x, y + size, z);

Print("z", x, y, z + size);

}

### Відображення сегмента

private void DrawThickCube(double x1, double y1, double x2, double y2)

{

double thickness = 0.02;

double dxLine = x2 - x1;

double dyLine = y2 - y1;

double length = Math.Sqrt(dxLine \* dxLine + dyLine \* dyLine);

double offsetX = -(dyLine / length) \* thickness;

double offsetY = (dxLine / length) \* thickness;

glBegin(GL\_QUADS);

glVertex3d(x1 - offsetX, y1 - offsetY, -thickness);

glVertex3d(x1 + offsetX, y1 + offsetY, -thickness);

glVertex3d(x2 + offsetX, y2 + offsetY, -thickness);

glVertex3d(x2 - offsetX, y2 - offsetY, -thickness);

glVertex3d(x1 - offsetX, y1 - offsetY, thickness);

glVertex3d(x1 + offsetX, y1 + offsetY, thickness);

glVertex3d(x2 + offsetX, y2 + offsetY, thickness);

glVertex3d(x2 - offsetX, y2 - offsetY, thickness);

glEnd();

glBegin(GL\_QUADS);

glVertex3d(x1 - offsetX, y1 - offsetY, -thickness);

glVertex3d(x1 - offsetX, y1 - offsetY, thickness);

glVertex3d(x2 - offsetX, y2 - offsetY, thickness);

glVertex3d(x2 - offsetX, y2 - offsetY, -thickness);

glVertex3d(x1 + offsetX, y1 + offsetY, -thickness);

glVertex3d(x1 + offsetX, y1 + offsetY, thickness);

glVertex3d(x2 + offsetX, y2 + offsetY, thickness);

glVertex3d(x2 + offsetX, y2 + offsetY, -thickness);

glVertex3d(x1 - offsetX, y1 - offsetY, thickness);

glVertex3d(x1 + offsetX, y1 + offsetY, thickness);

glVertex3d(x2 + offsetX, y2 + offsetY, thickness);

glVertex3d(x2 - offsetX, y2 - offsetY, thickness);

glVertex3d(x1 - offsetX, y1 - offsetY, -thickness);

glVertex3d(x1 + offsetX, y1 + offsetY, -thickness);

glVertex3d(x2 + offsetX, y2 + offsetY, -thickness);

glVertex3d(x2 - offsetX, y2 - offsetY, -thickness);

glEnd();

}

### Використання світла

private void SetupLighting()

{

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

float[] lightPosition = { 5.0f, 5.0f, 5.0f, 1.0f };

glLightfv(GL\_LIGHT0, GL\_POSITION, lightPosition);

float[] lightAmbient = { 0.2f, 0.2f, 0.2f, 1.0f };

float[] lightDiffuse = { 0.8f, 0.8f, 0.8f, 1.0f };

float[] lightSpecular = { 1.0f, 1.0f, 1.0f, 1.0f };

glLightfv(GL\_LIGHT0, GL\_AMBIENT, lightAmbient);

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, lightDiffuse);

glLightfv(GL\_LIGHT0, GL\_SPECULAR, lightSpecular);

glEnable(GL\_COLOR\_MATERIAL);

glColorMaterial(GL\_FRONT, GL\_AMBIENT\_AND\_DIFFUSE);

float[] materialSpecular = { 1.0f, 1.0f, 1.0f, 1.0f };

float materialShininess = 50.0f;

glMaterialfv(GL\_FRONT, GL\_SPECULAR, materialSpecular);

glMaterialf(GL\_FRONT, GL\_SHININESS, materialShininess);

}

## Результати виконання практичної роботи

### Розв'язання завдання

У результаті виконання розробленого додатка на екран виведено маніпулятор, що відповідає заданій кінематичній схемі.

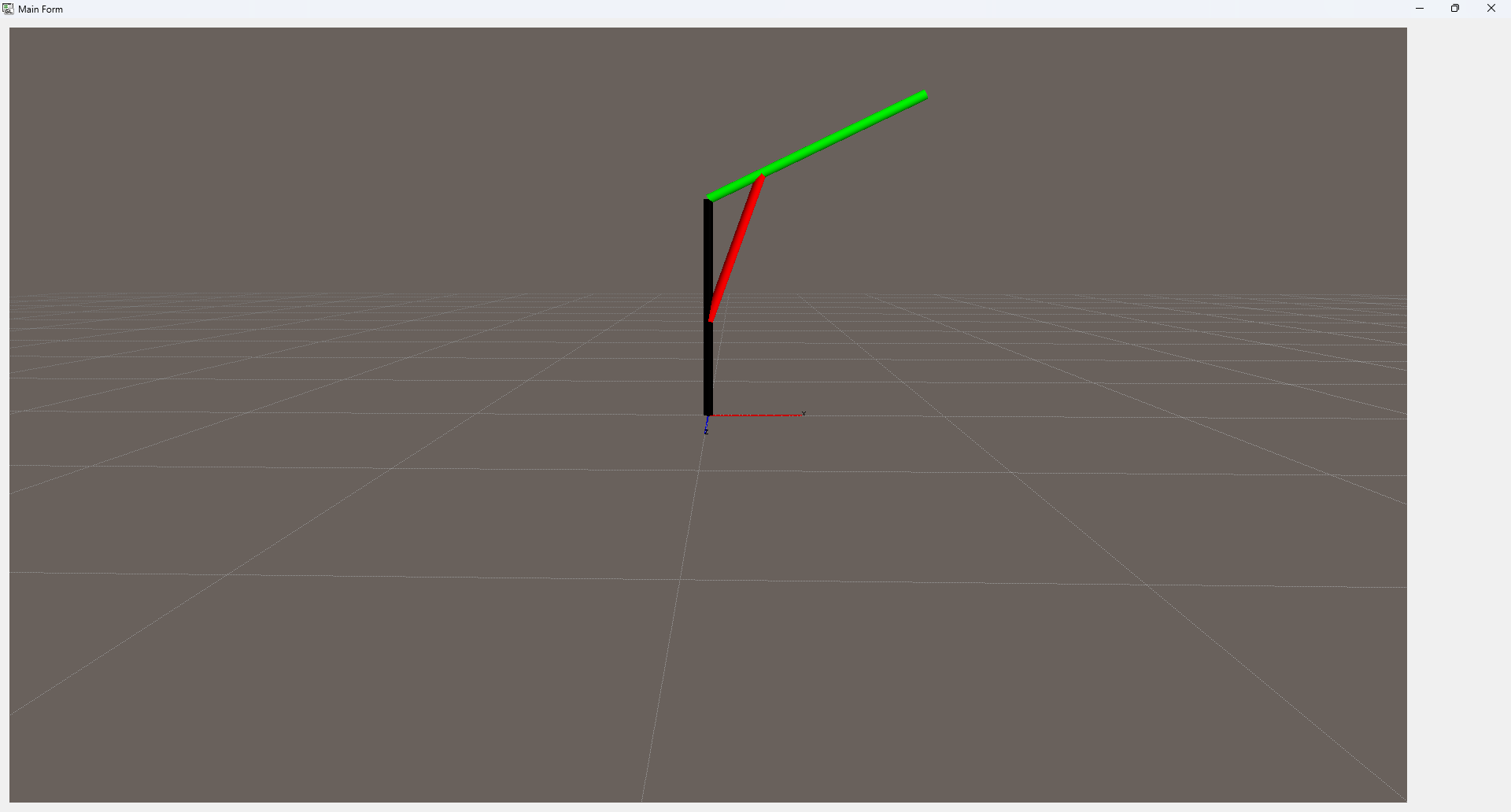


Рисунок 6.1 – Тестування роботи програми, фігура без змін

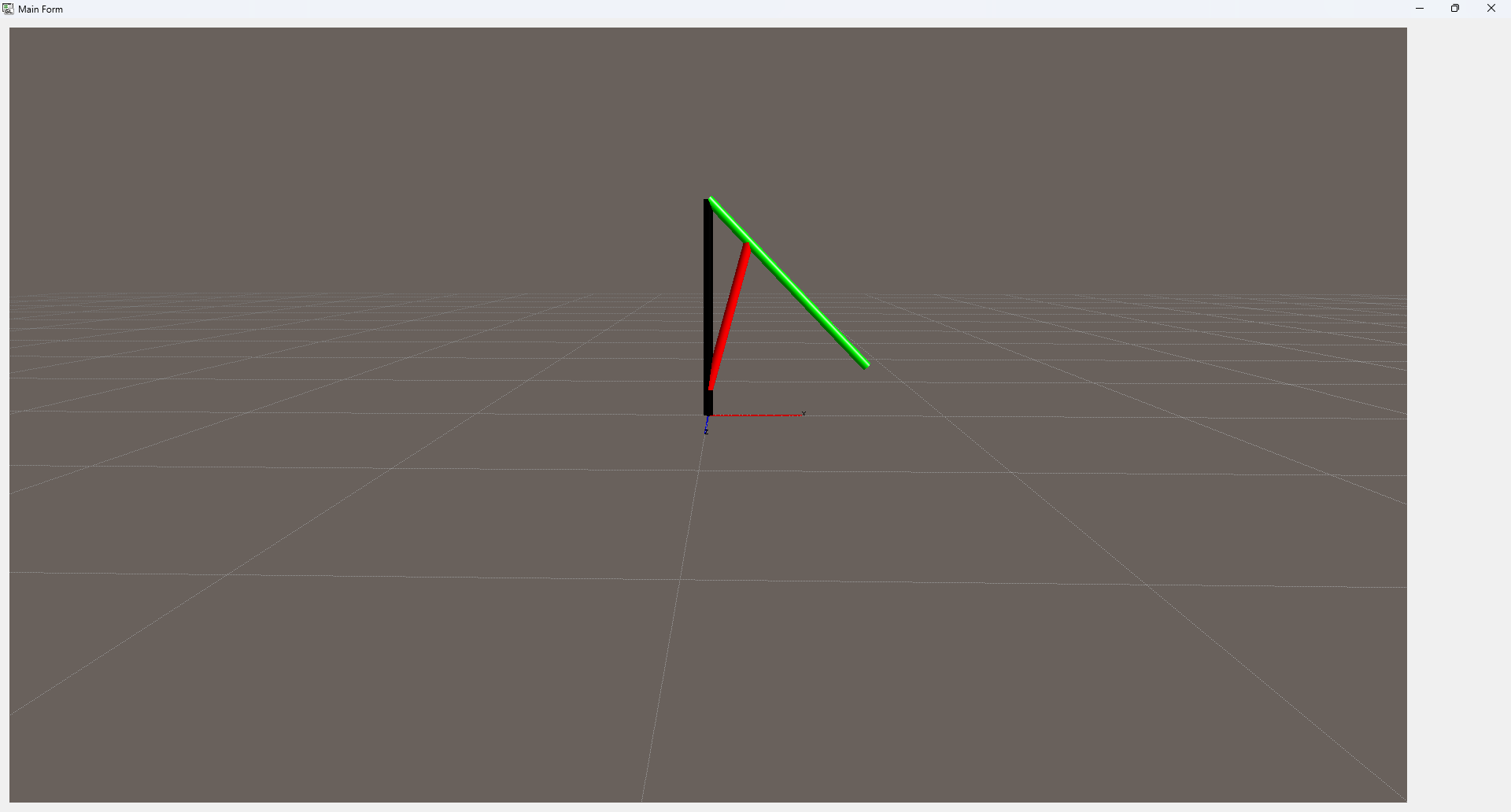


Рисунок 6.2 – Тестування роботи програми, точка переміщена нижче

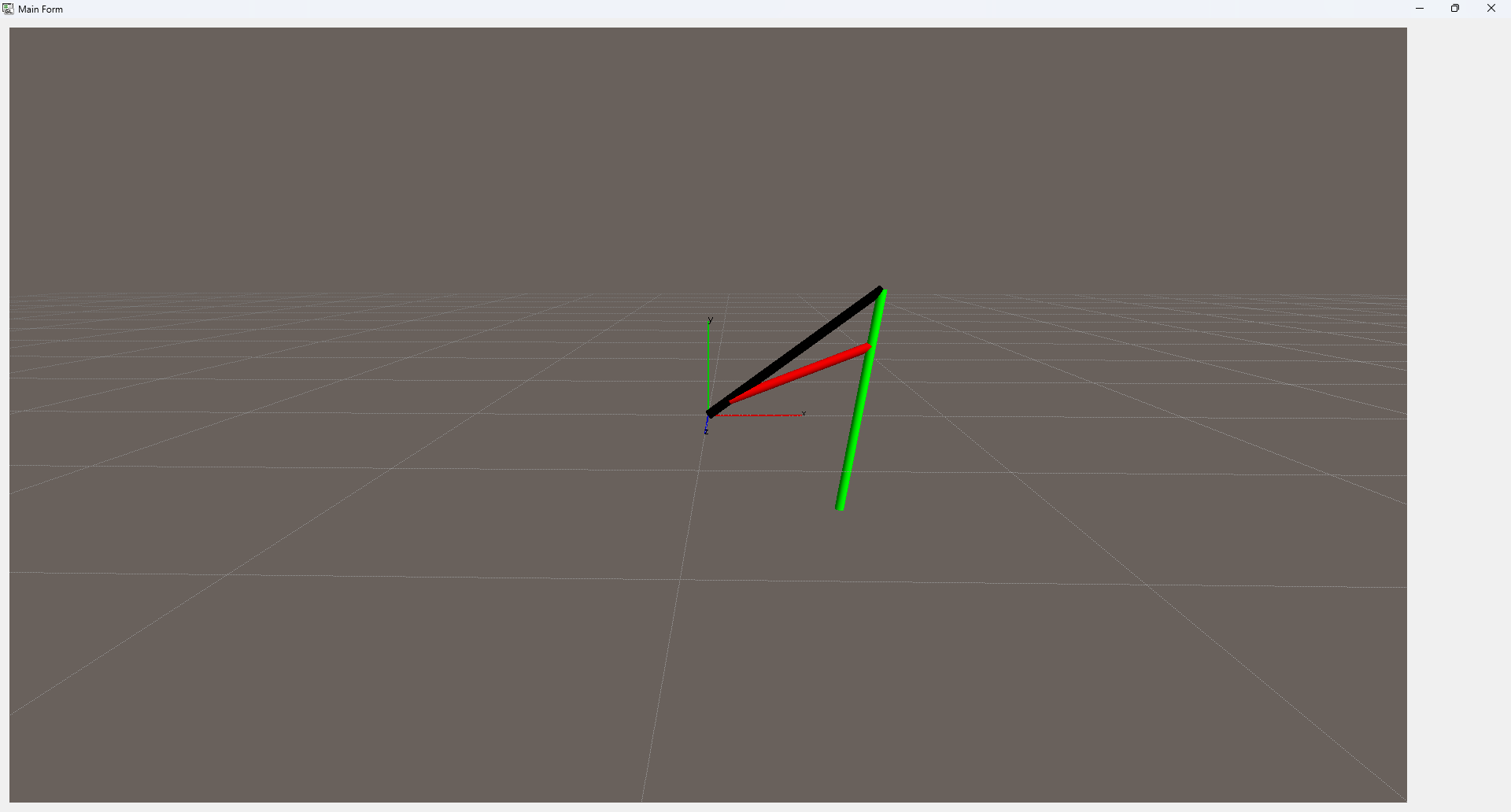


Рисунок 6.3 – Тестування роботи програми, точка переміщена нижче та фігура трохи нахилена

### 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 6.1.

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

# Практична робота 7. Екранна заставка з анімацією

## Завдання

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

## Теоретичні відомості

Фактично ScreenSaver – звичайний виконуваний файл (тільки з розширенням .SCR замість .EXE), яким керують через параметри командного рядка («/c» – configure, «/p» – preview, «/s» – show):

ScreenSaver.scr – показати вікно налаштувань;

ScreenSaver.scr /c – показати вікно налаштувань модально;

ScreenSaver.scr /s – основний повноекранний режим роботи;

ScreenSaver.scr /p hWnd – попередній перегляд основного режиму в батьківському вікні з дескриптором hWnd;

ScreenSaver.scr /a – установлення пароля у Windows 95 (застарілий режим, зараз не використовується).

Формально ці режими можуть бути реалізовані за допомогою вікна налаштувань (Setting Form) і основного вікна (Main Form), проте під час запуску працює тільки одне з них. Для вибору режиму роботи в процесі розробки та налагодження (під керуванням Visual Studio) параметр командного рядка задається у властивостях проєкту.

Процес розробки рекомендується розділити на декілька етапів. На першому етапі реалізують аналіз командного рядка, на другому – діалогове вікно з налаштуваннями і механізм їхнього зчитування/збереження, на третьому – вікно з виводом графіки та прив’язкою анімації до механізму бездіяльності програми [2, 3], а після механізм завершення роботи основного вікна за подією від маніпулятора «миш» чи клавіатури і перевіряють роботу основного вікна у батьківському для режиму попереднього перегляду.

За замовчуванням усі екранні заставки знаходяться у каталозі «c:\windows\system32\\*.scr», де можна розмістити і свою програму [1]. Окрім, установити і просто протестувати заставку можна через контекстне меню операційної системи..

## Результати виконання практичної роботи

### Розв'язання завдання

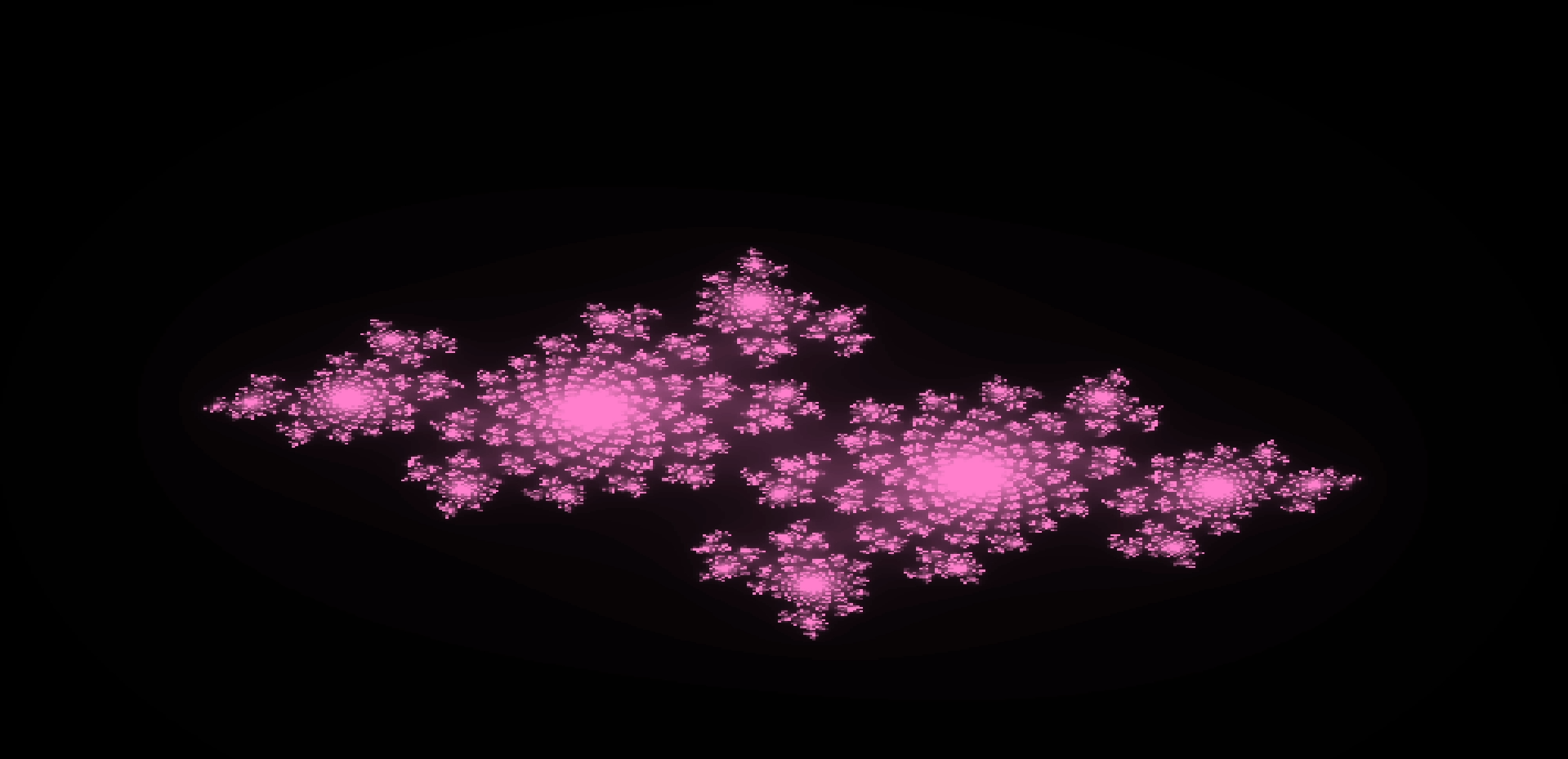


Рисунок 7.1 – Працююча анімація

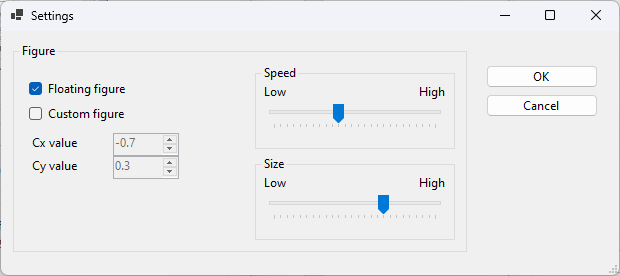


Рисунок 7.2 – Вікно з налаштуваннями

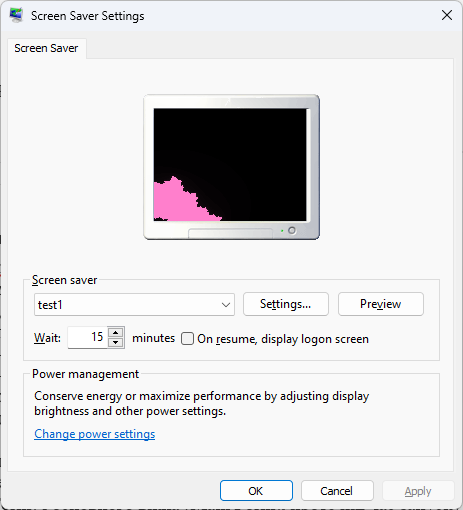


Рисунок 7.3 – Встановлена екранна заставка

### 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 7.1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблиця 7.1 | | | | |
| № з/п | Складність | Вимоги | Бали | Зроблено |
|  | Базовий рівень | Прив’язка анімації до механізму простою операційної системи | 2 | **+** |
|  | Реалізація повноекранного режиму (команда «/s») | 1 | **+** |
|  | Реалізація налаштування програми Screen Saver (команда «/c») | 1 | **+** |
|  | Реалізація попереднього перегляду (команда «/p») | 1 | **+** |
|  | Підвищений рівень | Збереження конфігурації і налаштувань програми ScreenSaver у реєстрі ОС | 1 | **+** |
|  | 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. using System;
2. using System.Collections.Generic;
3. using System.ComponentModel;
4. using System.Diagnostics;
5. using System.Drawing;
6. using System.Drawing.Design;
7. using System.Linq;
8. namespace glWinForm1
9. {
10. public partial class RenderControl : OpenGL
11. {
12. public RenderControl()
13. {
14. InitializeComponent();
15. }
16. private void OnRender(object sender, EventArgs e)
17. {
18. glClear(GL\_COLOR\_BUFFER\_BIT);
19. glLoadIdentity();
20. glViewport(0,0, Width, Height);
21. gluOrtho2D(-10,+12,-4,+8);
23. FirstTask firstTask = new FirstTask();
24. firstTask.DrawGrid();
25. firstTask.DrawShape();
26. firstTask.DrawPoints();
27. }
28. }
29. }

### Код файлу (FirstTasks.cs)

1. using System;
2. using System.Collections.Generic;
3. using System.Drawing;
4. using System.Linq;
5. using System.Text;
6. using System.Threading.Tasks;
7. namespace glWinForm1
8. {
9. public partial class RenderControl
10. {
11. public class FirstTask
12. {
13. public void DrawGrid()
14. {
15. glLineWidth(1);
16. glEnable(GL\_LINE\_STIPPLE);
17. glLineStipple(6, 0xAAAA);
18. glColor3ub(189, 189, 189);
19. glBegin(GL\_LINES);
20. for (int i = -8; i <= 10; i++)
21. {
22. glVertex2d(i, -2.5);
23. glVertex2d(i, 6.5);
24. }
25. for (int j = -2; j <= 6; j++)
26. {
27. glVertex2d(-8.5, j);
28. glVertex2d(10.5, j);
29. }
30. glEnd();
31. glDisable(GL\_LINE\_STIPPLE);
32. }
33. public void Figure(double plusX = 0)
34. {
35. glVertex2d(-6 + plusX, -2);
36. glVertex2d(-8 + plusX, 4);
37. glVertex2d(-6 + plusX, 6);
38. glVertex2d(-4 + plusX, 6);
39. glVertex2d(-2 + plusX, 4);
40. glVertex2d(-4 + plusX, 0);
41. }
42. public void DrawShape()
43. {
44. glLineWidth(3);
45. glColor3ub(0, 0, 0);
46. glBegin(GL\_LINE\_LOOP);
47. Figure(0);
48. glEnd();
49. }
50. public void DrawPoints()
51. {
52. glEnable(GL\_POINT\_SMOOTH);
53. glPointSize(8);
54. glBegin(GL\_POINTS);
55. Figure(10);
56. glEnd();
57. glDisable(GL\_POINT\_SMOOTH);
58. }
59. }
60. }
61. }

# Додаток Б. Лістинг програми до практичної роботи №2

### Код файлу (RenderControl.cs)

1. using System;
2. using System;
3. using System.Collections.Generic;
4. using System.ComponentModel;
5. using System.Diagnostics;
6. using System.Drawing;
7. using System.Drawing.Design;
8. using System.Linq;
9. namespace Task02
10. {
11. public partial class RenderControl : OpenGL
12. {
13. public int Verticales { get; set; } = 1;
14. public int Horizontales { get; set; } = 1;
15. public DrawingMode DrawingMode { get; set; } = 0;
16. public RenderControl()
17. {
18. InitializeComponent();
19. }
20. private void OnRender(object sender, EventArgs e)
21. {
22. glClear(GL\_COLOR\_BUFFER\_BIT);
23. glLoadIdentity();
24. Draw draw = new Draw();
25. double size = 160.0;
26. if (Width > Height)
27. glViewport((Width - Height) / 2, 0, Height, Height);
28. Else
29. glViewport(0, (Height - Width) / 2, Width, Width);
30. double maxSize = Math.Max(Horizontales, Verticales);
31. gluOrtho2D(-size \* maxSize, size \* maxSize, -size \* maxSize, size \* maxSize);
32. draw.DrawFigure(Verticales, Horizontales, 50, DrawingMode);
33. glLoadIdentity();
34. glViewport(0, 0, Width, Height);
35. gluOrtho2D(-Width / 2, Width / 2, -Height / 2, Height / 2);
36. draw.DrawButtons(DrawingMode, -Width / 2, Width / 2, -Height / 2, Height / 2);
37. }
38. }
39. }

### Код файлу (Draw.cs)

1. using System;
2. using System.Collections.Generic;
3. using System.IO;
4. using System.Linq;
5. using System.Text;
6. using System.Threading.Tasks;
7. namespace Task02
8. {
9. public partial class RenderControl
10. {
11. public class Draw
12. {
13. public delegate void outText(string s, double x, double y, double z = 0);
14. public outText Print;
15. public void DrawFigure(int verticales, int horizontales, double size, DrawingMode drawingMode)
16. {
17. double offsetX, offsetY;
18. if (verticales != 0 || horizontales != 0)
19. {
20. for (int i = 0; i < horizontales; i += 1)
21. {
22. for (int j = 0; j < verticales; j += 1)
23. {
24. offsetX = i \* size \* 2;
25. offsetY = j \* size \* 2;
26. if (j > 0)
27. {
28. offsetX += size \* j;
29. }
30. if (i > 0)
31. {
32. offsetY -= (size / 2) \* i;
33. }
34. switch (drawingMode)
35. {
36. case DrawingMode.Fill:
37. glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_FILL);
38. break;
39. case DrawingMode.Line:
40. glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE);
41. break;
42. case DrawingMode.Point:
43. glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_POINT);
44. break;
45. }
46. glColor3f(1, 0, 0);
47. glBegin(GL\_TRIANGLES);
48. CoordinatesTriangleUP(offsetX, offsetY, size);
49. glColor3f(1, 1, 0);
50. CoordinatesTriangleUP(offsetX + size / 2, offsetY + size, size);
51. glColor3f(1, 1, 1);
52. CoordinatesTriangleDown(offsetX, offsetY, size);
53. CoordinatesTriangleDown(offsetX + size / 2, offsetY + size, size);
54. glEnd();
55. glColor3f(0, 1, 0);
56. glBegin(GL\_POLYGON);
57. CoordinatesPolygon(offsetX + size, offsetY, size);
58. glEnd();
59. glColor3f(0, 0, 1);
60. glBegin(GL\_POLYGON);
61. CoordinatesPolygon(offsetX + size \* 1.5, offsetY + size, size);
62. glEnd();
63. }
64. }
65. }
66. }
67. public void DrawButtons(DrawingMode drawingMode, double start\_x, double end\_x, double start\_y, double end\_y)
68. {
69. glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_FILL);
70. double buttonWidth = 50.0;
71. double buttonHeight = 25.0;
72. double spacing = 10.0;
73. double xStart = end\_x - (buttonWidth + spacing);
74. double yStart = end\_y - (buttonHeight + spacing);
75. string[] buttonLabels = { "Fill", "Lines", "Point" };
76. for (int i = 0; i < 3; i++)
77. {
78. double x = xStart - i \* (buttonWidth + spacing);
79. double y = yStart;
80. if ((int)drawingMode == 2 - i)
81. {
82. glColor3f(0, 150, 255);
83. }
84. else
85. {
86. glColor3f(0.0f, 0.6f, 1.0f);
87. }
88. glBegin(GL\_POLYGON);
89. glVertex2d(x, y);
90. glVertex2d(x, y + buttonHeight);
91. glVertex2d(x + buttonWidth, y + buttonHeight);
92. glVertex2d(x + buttonWidth, y);
93. glEnd();
94. glColor3f(0, 0, 0);
95. Print(buttonLabels[2 - i], x + 3, y + buttonHeight / 3);
96. }
97. }
98. public void DrawGrid(double start\_x, double end\_x, double start\_y, double end\_y)
99. {
100. glLineWidth(1);
101. glEnable(GL\_LINE\_STIPPLE);
102. glLineStipple(6, 0xAAAA);
103. glColor3ub(189, 189, 189);
104. glBegin(GL\_LINES);
105. for (int i = (int)start\_x; i <= end\_x; i++)
106. {
107. glVertex2d(i, start\_y);
108. glVertex2d(i, end\_y);
109. }
110. for (int j = (int)start\_y; j <= end\_y; j++)
111. {
112. glVertex2d(start\_x, j);
113. glVertex2d(end\_x, j);
114. }
115. glEnd();
116. glDisable(GL\_LINE\_STIPPLE);
117. }
118. public void CoordinatesPolygon(double start\_x, double start\_y, double size)
119. {
120. glVertex2d(start\_x, start\_y);
121. glVertex2d(start\_x + size / 2, start\_y + size);
122. glVertex2d(start\_x + size \* 1.5, start\_y + size / 2);
123. glVertex2d(start\_x + size, start\_y - size / 2);
124. }
125. public void CoordinatesTriangleUP(double start\_x, double start\_y, double size)
126. {
127. glVertex2d(start\_x, start\_y);
128. glVertex2d(start\_x + size / 2, start\_y + size);
129. glVertex2d(start\_x + size, start\_y);
130. }
131. public void CoordinatesTriangleDown(double start\_x, double start\_y, double size)
132. {
133. glVertex2d(start\_x + size / 2, start\_y + size);
134. glVertex2d(start\_x + size, start\_y);
135. glVertex2d(start\_x + size \* 1.5, start\_y + size);
136. }
137. }
138. }
139. }

# Додаток В. Лістинг програми до практичної роботи №3

### Код файлу (RenderControl.cs)

1. using System;
2. using System.Collections.Generic;
3. using System.ComponentModel;
4. using System.Diagnostics;
5. using System.Drawing;
6. using System.Drawing.Design;
7. using System.Linq;
8. namespace Task03
9. {
10. public partial class RenderControl : OpenGL
11. {
12. DrawFunction drawFunction;
13. public double npoint = 100;
14. public bool CustomChange { get; set; } = false;
15. public double XMin { get; set; } = -1.2;
16. public double XMax { get; set; } = +1.2;
17. public double YMin { get; set; } = -1.2;
18. public double YMax { get; set; } = +1.2;
19. public int ChosedFunc { get; set; } = 0;
20. Func<double, double>[] functionArray;
21. public RenderControl()
22. {
23. InitializeComponent();
24. }
25. private void OnRender(object sender, EventArgs e)
26. {
27. glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
28. glClear(GL\_COLOR\_BUFFER\_BIT);
29. glLoadIdentity();
30. glViewport(0, 0, Width, Height);
31. gluOrtho2D(XMin, XMax, YMin, YMax);
32. drawFunction.DrawGrid(XMin, XMax, YMin, YMax);
33. drawFunction.DrawCoordinateGrid(XMin, XMax, YMin, YMax);
34. drawFunction.DrawFunctionLinesPoints(XMin, XMax, YMin, YMax, npoint, functionArray[ChosedFunc]);
35. }
36. private void RenderControl\_ContextCreated(object sender, EventArgs e)
37. {
38. drawFunction = new DrawFunction();
39. double epsilon = 1e-10;
40. functionArray = new Func<double, double>[]
41. {
42. x => (Math.Sin(Math.Sin(x + 1))) / ((4 \* x) + (3 \* x) + 2),
43. x => {
44. double cosValue = Math.Cos(3 \* x) + epsilon;
45. double sinValue = Math.Sin(2 \* x) + epsilon;
46. return (1 / cosValue) + (1 / sinValue);
47. },
48. };
49. }
50. }
51. }

### Код файлу (DrawFunction.cs)

1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Text;
5. using System.Threading.Tasks;
6. namespace Task03
7. {
8. public partial class RenderControl
9. {
10. public class DrawFunction
11. {
12. public void DrawGrid(double XMin, double XMax, double YMin, double YMax)
13. {
14. glLineWidth(1.0f);
15. glColor3ub(200, 200, 200);
16. glBegin(GL\_LINES);
17. for (double i = XMin; i <= XMax; i += 0.2)
18. {
19. glVertex2d(i, YMin);
20. glVertex2d(i, YMax);
21. }
22. for (double j = YMin; j <= YMax; j += 0.2)
23. {
24. glVertex2d(XMin, j);
25. glVertex2d(XMax, j);
26. }
27. glEnd();
28. }
29. public void DrawCoordinateGrid(double XMin, double XMax, double YMin, double YMax)
30. {
31. glLineWidth(3.0f);
32. glBegin(GL\_LINES);
33. glColor3ub(0, 0, 0);
34. glVertex2d(XMin, 0);
35. glVertex2d(XMax, 0);
36. glVertex2d(0, YMin);
37. glVertex2d(0, YMax);
38. glEnd();
39. }
40. public void DrawFunctionLinesPoints(double XMin, double XMax, double YMin, double YMax, double npoint, Func<double, double> func)
41. {
42. glLineWidth(2.0f);
43. glColor3ub(50, 10, 0);
44. glBegin(GL\_LINE\_STRIP);
45. double previousY = double.NaN;
46. double x = XMin;
47. double h = (XMax - XMin) / (npoint - 1);
48. double y = func(x);
49. glVertex2d(x, y);
50. for (int i = 1; i < npoint; i++)
51. {
52. previousY = y;
53. x = XMin + i \* h;
54. y = func(x);
55. if (!double.IsNaN(previousY) && Math.Abs(y - previousY) > 5.0)
56. {
57. glEnd();
58. glBegin(GL\_LINE\_STRIP);
59. continue;
60. }
61. glVertex2d(x, y);
62. if ((previousY \* y) <= 0 && previousY != 0)
63. {
64. glEnd();
65. DrawPoint(previousY, x, h, y);
66. glBegin(GL\_LINE\_STRIP);
67. glColor3ub(50, 10, 0);
68. glVertex2d(x, y);
69. }
70. }
71. glEnd();
72. }
73. private void DrawPoint(double previousY, double x, double h, double y)
74. {
75. glPointSize(5.0f);
76. glColor3ub(250, 0, 0);
77. glBegin(GL\_POINTS);
78. glVertex2d(x - h / 2, (previousY + y) / 2);
79. glEnd();
80. }
81. }
82. }
83. }

### Код файлу (MainForm.cs)

1. using System.Drawing;
2. using System.Windows.Forms;
3. using static Task03.OpenGL;
4. namespace Task03
5. {
6. public partial class MainForm : Form
7. {
8. public MainForm()
9. {
10. InitializeComponent();
11. }
12. private void OnChanged(object sender, System.EventArgs e)
13. {
14. renderControl1.npoint = (int)nuppoints.Value;
15. renderControl1.Invalidate();
16. }
17. private void OnChangedXmin(object sender, System.EventArgs e)
18. {
19. renderControl1.XMin = (double)nupxmin.Value;
20. renderControl1.Invalidate();
21. }
22. private void OnChangedXmax(object sender, System.EventArgs e)
23. {
24. renderControl1.XMax = (double)nupxmax.Value;
25. renderControl1.Invalidate();
26. }
27. private void showY\_Click(object sender, System.EventArgs e)
28. {
29. if (!renderControl1.CustomChange)
30. {
31. lpoints.Location = new Point(lpoints.Location.X, lpoints.Location.Y + 56);
32. nuppoints.Location = new Point(nuppoints.Location.X, nuppoints.Location.Y + 56);
33. showY.Location = new Point(showY.Location.X, showY.Location.Y + 56);
34. func1.Location = new Point(func1.Location.X, func1.Location.Y + 56);
35. func2.Location = new Point(func2.Location.X, func2.Location.Y + 56);
36. pfunc1.Location = new Point(pfunc1.Location.X, pfunc1.Location.Y + 56);
37. pfunc2.Location = new Point(pfunc2.Location.X, pfunc2.Location.Y + 56);
38. showY.Text = "Hide Y";
39. nupymin.Visible = true;
40. nupymax.Visible = true;
41. lymax.Visible = true;
42. lymin.Visible = true;
43. }
44. else
45. {
46. lpoints.Location = new Point(lpoints.Location.X, lpoints.Location.Y - 56);
47. nuppoints.Location = new Point(nuppoints.Location.X, nuppoints.Location.Y - 56);
48. showY.Location = new Point(showY.Location.X, showY.Location.Y - 56);
49. func1.Location = new Point(func1.Location.X, func1.Location.Y - 56);
50. func2.Location = new Point(func2.Location.X, func2.Location.Y - 56);
51. pfunc1.Location = new Point(pfunc1.Location.X, pfunc1.Location.Y - 56);
52. pfunc2.Location = new Point(pfunc2.Location.X, pfunc2.Location.Y - 56);
53. showY.Text = "Show Y";
54. nupymin.Visible = false;
55. nupymax.Visible = false;
56. lymax.Visible = false;
57. lymin.Visible = false;
58. }
59. renderControl1.CustomChange = !renderControl1.CustomChange;
60. }
61. private void OnChangedYmin(object sender, System.EventArgs e)
62. {
63. renderControl1.YMin = (double)nupymin.Value;
64. renderControl1.Invalidate();
65. }
66. private void OnChangedYmax(object sender, System.EventArgs e)
67. {
68. renderControl1.YMax = (double)nupymax.Value;
69. renderControl1.Invalidate();
70. }
71. private void SwitchFunk1(object sender, System.EventArgs e)
72. {
73. if (func1.Checked)
74. {
75. renderControl1.ChosedFunc = 0;
76. }
77. renderControl1.Invalidate();
78. }
79. private void SwitchFunk2(object sender, System.EventArgs e)
80. {
81. if (func2.Checked)
82. {
83. renderControl1.ChosedFunc = 1;
84. }
85. renderControl1.Invalidate();
86. }
87. private void CheckedFunk1(object sender, System.EventArgs e)
88. {
89. if (!func1.Checked)
90. {
91. func1.Checked = true;
92. }
93. }
94. private void CheckedFunk2(object sender, System.EventArgs e)
95. {
96. if (!func2.Checked)
97. {
98. func2.Checked = true;
99. }
100. }
101. }
102. }

# Додаток Г. Лістинг програми до практичної роботи №4

### Код файлу (RenderControl.cs)

1. using System;
2. using System.Collections.Generic;
3. using System.ComponentModel;
4. using System.Diagnostics;
5. using System.Drawing;
6. using System.Drawing.Design;
7. using System.Linq;
8. using System.Security.Policy;
9. namespace Task04
10. {
11. public partial class RenderControl : OpenGL
12. {
13. Draw draw;
14. public double Xls { get; set; }
15. public double Yls { get; set; }
16. public double Xle { get; set; }
17. public double Yle { get; set; }
18. public double Length { get; set; } = 1.5;
19. public bool ChosedFigure { get; set; } = false;
20. public bool IsUp { get; set; } = true;
21. public bool LineExist { get; set; } = false;
22. public double A { get; set; } = 1.0;
23. public double B { get; set; } = 0.7;
24. public RenderControl()
25. {
26. InitializeComponent();
27. }
29. private void OnRender(object sender, EventArgs e)
30. {
31. glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
32. glClear(GL\_COLOR\_BUFFER\_BIT);
33. glLoadIdentity();
35. int size = Math.Min(Width, Height);
36. glViewport((Width - size) / 2, (Height - size) / 2, size, size);
37. Length = draw.FindMax(A, B, ChosedFigure);
38. gluOrtho2D(-Length, Length, -Length, Length);
39. draw.DrawGrid(-Length, Length, -Length, Length);
40. draw.DrawCoordinateGrid(-Length, Length, -Length, Length);
41. if (!ChosedFigure)
42. draw.DrawEllipse(A, B);
43. else
44. draw.DrawParabola(A);
45. if (LineExist)
46. {
47. CreateLine();
48. draw.SearchPoint(A, B, Xls, Yls, Xle, Yle, ChosedFigure);
49. }
50. }
51. private void CreateLine()
52. {
53. if (!IsUp)
54. {
55. draw.DrawLineWait(Xls, Xle, Yls, Yle);
56. }
57. else
58. {
59. draw.DrawLine(Xls, Xle, Yls, Yle);
60. }
61. }
62. private void Created(object sender, EventArgs e)
63. {
64. draw = new Draw();
65. }
66. }
67. }

### Код файлу (Draw.cs)

1. using System;
2. using System.Collections.Generic;
3. using System.Drawing.Printing;
4. using System.Linq;
5. using System.Text;
6. using System.Threading.Tasks;
7. namespace Task04
8. {
9. public partial class RenderControl
10. {
11. public class Draw
12. {
13. public void DrawEllipse(double a, double b)
14. {
15. glColor3f(0.0f, 0.0f, 1.0f);
16. glBegin(GL\_LINE\_LOOP);
17. int segments = 100;
18. for (int i = 0; i <= segments; i++)
19. {
20. double theta = (2.0f \* Math.PI \* i / segments);
21. double x = a \* Math.Cos(theta);
22. double y = b \* Math.Sin(theta);
23. glVertex2d(x, y);
24. }
25. glEnd();
26. }
27. public void DrawParabola(double a)
28. {
29. glColor3f(1.0f, 0.0f, 0.0f);
30. glBegin(GL\_LINE\_STRIP);
32. int segments = 100;
33. double range = 2.0;
34. for (int i = -segments + 1; i <= segments; i++)
35. {
36. double x = i \* range / segments;
37. double y = a \* x \* x;
38. glVertex2d(x, y);
39. }
40. glEnd();
41. }
42. public void SearchPoint(double a, double b, double Xls, double Yls, double Xle, double Yle, bool isChoosed)
43. {
44. glPointSize(5.0f);
45. glColor3f(0.0f, 1.0f, 0.0f);
46. glBegin(GL\_POINTS);
47. double dx = Xle - Xls;
48. double dy = Yle - Yls;
49. if (!isChoosed)
50. {
51. double A = (dx \* dx) / (a \* a) + (dy \* dy) / (b \* b);
52. double B = 2 \* ((Xls \* dx) / (a \* a) + (Yls \* dy) / (b \* b));
53. double C = (Xls \* Xls) / (a \* a) + (Yls \* Yls) / (b \* b) - 1;
54. double discriminant = B \* B - 4 \* A \* C;
55. if (discriminant >= 0)
56. {
57. double sqrtD = Math.Sqrt(discriminant);
58. double t1 = (-B + sqrtD) / (2 \* A);
59. double t2 = (-B - sqrtD) / (2 \* A);
60. if (t1 >= 0 && t1 <= 1)
61. {
62. double x1 = Xls + t1 \* dx;
63. double y1 = Yls + t1 \* dy;
64. glVertex2d(x1, y1);
65. }
66. if (t2 >= 0 && t2 <= 1)
67. {
68. double x2 = Xls + t2 \* dx;
69. double y2 = Yls + t2 \* dy;
70. glVertex2d(x2, y2);
71. }
72. }
73. }
74. else
75. {
76. double A = a \* dx \* dx;
77. double B = 2 \* a \* dx \* Xls - dy;
78. double C = a \* Xls \* Xls - Yls;
79. double discriminant = B \* B - 4 \* A \* C;
80. if (discriminant >= 0)
81. {
82. double sqrtD = Math.Sqrt(discriminant);
83. double t1 = (-B + sqrtD) / (2 \* A);
84. double t2 = (-B - sqrtD) / (2 \* A);
85. if (t1 >= 0 && t1 <= 1)
86. {
87. double x1 = Xls + t1 \* dx;
88. double y1 = Yls + t1 \* dy;
89. glVertex2d(x1, y1);
90. }
91. if (t2 >= 0 && t2 <= 1)
92. {
93. double x2 = Xls + t2 \* dx;
94. double y2 = Yls + t2 \* dy;
95. glVertex2d(x2, y2);
96. }
97. }
98. }
99. glEnd();
100. }
101. public double FindMax(double a, double b, bool isChoosed)
102. {
103. double maxX, maxY;
104. if (!isChoosed)
105. {
106. maxX = Math.Abs(a);
107. maxY = Math.Abs(b);
108. }
109. else
110. {
111. double range = 2.0;
112. maxX = range;
113. maxY = a \* range \* range;
114. maxY = maxY < 0 ? -maxY : maxY;
115. }
116. return Math.Max(maxX, maxY) + 0.2;
117. }
118. public void DrawGrid(double XMin, double XMax, double YMin, double YMax)
119. {
120. glLineWidth(1.0f);
121. glColor3ub(200, 200, 200);
122. glBegin(GL\_LINES);
123. for (double i = XMin; i <= XMax; i += XMax/10)
124. {
125. glVertex2d(i, YMin);
126. glVertex2d(i, YMax);
127. }
128. for (double j = YMin; j <= YMax; j += YMax/10)
129. {
130. glVertex2d(XMin, j);
131. glVertex2d(XMax, j);
132. }
133. glEnd();
134. }
135. public void DrawCoordinateGrid(double XMin, double XMax, double YMin, double YMax)
136. {
137. glLineWidth(2.0f);
138. glBegin(GL\_LINES);
139. glColor3ub(0, 0, 0);
140. glVertex2d(XMin, 0);
141. glVertex2d(XMax, 0);
142. glVertex2d(0, YMin);
143. glVertex2d(0, YMax);
144. glEnd();
145. glLineWidth(1.0f);
146. }
147. public void DrawLineWait(double XMin, double XMax, double YMin, double YMax)
148. {
149. glColor3f(0.5f, 0.5f, 0.5f);
150. glLineWidth(1.4f);
151. glPointSize(3.0f);
152. glBegin(GL\_POINTS);
153. glVertex2d(XMin, YMin);
154. glVertex2d(XMax, YMax);
155. glEnd();
156. glBegin(GL\_LINES);
157. glVertex2d(XMin, YMin);
158. glVertex2d(XMax, YMax);
159. glEnd();
160. }
161. public void DrawLine(double XMin, double XMax, double YMin, double YMax)
162. {
163. glColor3f(0.0f, 0.0f, 0.0f);
164. glLineWidth(1.7f);
165. glBegin(GL\_LINES);
166. glVertex2d(XMin, YMin);
167. glVertex2d(XMax, YMax);
168. glEnd();
169. }
170. }
171. }
172. }

### Код файлу (MainForm.cs)

1. using System;
2. using System.Windows.Forms;
3. using static Task04.OpenGL;
4. namespace Task04
5. {
6. public partial class MainForm : Form
7. {
8. public MainForm()
9. {
10. InitializeComponent();
11. }
12. private void OnDown(object sender, MouseEventArgs e)
13. {
14. if (e.Button == MouseButtons.Left)
15. {
16. double difference = Math.Max(renderControl1.ClientRectangle.Width, renderControl1.ClientRectangle.Height) - Math.Min(renderControl1.ClientRectangle.Width, renderControl1.ClientRectangle.Height);
17. if (difference != 0)
18. {
19. renderControl1.LineExist = renderControl1.ClientRectangle.Width > renderControl1.ClientRectangle.Height ?
20. difference / 2 < e.X && e.X < Math.Min(renderControl1.ClientRectangle.Width, renderControl1.ClientRectangle.Height) + difference / 2 :
21. difference / 2 < renderControl1.ClientRectangle.Height - e.Y && renderControl1.ClientRectangle.Height - e.Y < Math.Min(renderControl1.ClientRectangle.Width, renderControl1.ClientRectangle.Height) + difference / 2;
22. if (renderControl1.LineExist)
23. {
24. renderControl1.IsUp = false;
25. double xLength = renderControl1.ClientRectangle.Width > renderControl1.ClientRectangle.Height ?
26. renderControl1.ClientRectangle.Width / (double)renderControl1.ClientRectangle.Height :
27. 1;
28. double yLength = renderControl1.ClientRectangle.Height > renderControl1.ClientRectangle.Width ?
29. renderControl1.ClientRectangle.Height / (double)renderControl1.ClientRectangle.Width :
30. 1;
31. renderControl1.Xls = renderControl1.Xle = (renderControl1.Length \* xLength) \* ((e.X - (renderControl1.ClientRectangle.Width / 2.0)) / (double)(renderControl1.ClientRectangle.Width / 2.0));
32. renderControl1.Yls = renderControl1.Yle = (renderControl1.Length \* yLength) \* (((renderControl1.ClientRectangle.Height / 2.0) - e.Y) / (double)(renderControl1.ClientRectangle.Height / 2.0));
33. }
34. else
35. {
36. btnClearLine.Enabled = false;
37. }
38. }
39. else
40. {
41. renderControl1.IsUp = false;
42. renderControl1.LineExist = true;
43. double xLength = renderControl1.ClientRectangle.Width > renderControl1.ClientRectangle.Height ?
44. renderControl1.ClientRectangle.Width / (double)renderControl1.ClientRectangle.Height :
45. 1;
46. double yLength = renderControl1.ClientRectangle.Height > renderControl1.ClientRectangle.Width ?
47. renderControl1.ClientRectangle.Height / (double)renderControl1.ClientRectangle.Width :
48. 1;
49. renderControl1.Xls = renderControl1.Xle = (renderControl1.Length \* xLength) \* ((e.X - (renderControl1.ClientRectangle.Width / 2.0)) / (double)(renderControl1.ClientRectangle.Width / 2.0));
50. renderControl1.Yls = renderControl1.Yle = (renderControl1.Length \* yLength) \* (((renderControl1.ClientRectangle.Height / 2.0) - e.Y) / (double)(renderControl1.ClientRectangle.Height / 2.0));
51. }
52. }
53. renderControl1.Invalidate();
54. }
55. private void OnUp(object sender, MouseEventArgs e)
56. {
57. if (e.Button == MouseButtons.Left && renderControl1.LineExist)
58. {
59. double xLength = renderControl1.ClientRectangle.Width > renderControl1.ClientRectangle.Height ?
60. renderControl1.ClientRectangle.Width / (double)renderControl1.ClientRectangle.Height :
61. 1;
62. double yLength = renderControl1.ClientRectangle.Height > renderControl1.ClientRectangle.Width ?
63. renderControl1.ClientRectangle.Height / (double)renderControl1.ClientRectangle.Width :
64. 1;
65. renderControl1.Xle = (renderControl1.Length \* xLength) \* ((e.X - (renderControl1.ClientRectangle.Width / 2.0)) / (double)(renderControl1.ClientRectangle.Width / 2.0));
66. renderControl1.Yle = (renderControl1.Length \* yLength) \* (((renderControl1.ClientRectangle.Height / 2.0) - e.Y) / (double)(renderControl1.ClientRectangle.Height / 2.0));
67. renderControl1.IsUp = true;
68. if (renderControl1.Xle == renderControl1.Xls && renderControl1.Yle == renderControl1.Yls)
69. {
70. renderControl1.LineExist = false;
71. btnClearLine.Enabled = false;
72. }
73. else
74. {
75. btnClearLine.Enabled = true;
76. }
77. }
78. renderControl1.Invalidate();
79. }
80. private void OnMove(object sender, MouseEventArgs e)
81. {
82. if (renderControl1.LineExist)
83. {
84. if (!renderControl1.IsUp)
85. {
86. double xLength = renderControl1.ClientRectangle.Width > renderControl1.ClientRectangle.Height ?
87. renderControl1.ClientRectangle.Width / (double)renderControl1.ClientRectangle.Height :
88. 1;
89. double yLength = renderControl1.ClientRectangle.Height > renderControl1.ClientRectangle.Width ?
90. renderControl1.ClientRectangle.Height / (double)renderControl1.ClientRectangle.Width :
91. 1;
92. renderControl1.Xle = (renderControl1.Length \* xLength) \* ((e.X - (renderControl1.ClientRectangle.Width / 2.0)) / (double)(renderControl1.ClientRectangle.Width / 2.0));
93. renderControl1.Yle = (renderControl1.Length \* yLength) \* (((renderControl1.ClientRectangle.Height / 2.0) - e.Y) / (double)(renderControl1.ClientRectangle.Height / 2.0));
94. renderControl1.Invalidate();
95. }
96. }
97. }
98. private void btnClearLine\_Click(object sender, System.EventArgs e)
99. {
100. renderControl1.LineExist = false;
101. renderControl1.Xls = renderControl1.Xle = renderControl1.Yls = renderControl1.Yle = 0;
102. btnClearLine.Enabled = false;
103. renderControl1.Invalidate();
104. }
105. private void OnCheckedElipse(object sender, System.EventArgs e)
106. {
107. renderControl1.ChosedFigure = false;
108. numB.Visible = true;
109. renderControl1.Invalidate();
110. }
111. private void OnCheckedParabola(object sender, System.EventArgs e)
112. {
113. renderControl1.ChosedFigure = true;
114. numB.Visible = false;
115. renderControl1.Invalidate();
116. }
117. private void OnChangedA(object sender, EventArgs e)
118. {
119. if (numA.Value == numB.Value)
120. {
121. numB.Value = (double)numA.Value > renderControl1.A ? (decimal)((double)numB.Value - 0.1) : (decimal)((double)numB.Value + 0.1);
122. }
123. renderControl1.A = (double)numA.Value;
124. renderControl1.Invalidate();
125. }
126. private void OnChangedB(object sender, EventArgs e)
127. {
128. if (numA.Value == numB.Value)
129. {
130. numA.Value = (double)numB.Value > renderControl1.B ? (decimal)((double)numA.Value - 0.1) : (decimal)((double)numA.Value + 0.1);
131. }
132. renderControl1.B = (double)numB.Value;
133. renderControl1.Invalidate();
134. }
135. }
136. }

# Додаток Г. Лістинг програми до практичної роботи №5

### Код файлу (RenderControl.cs)

1. using System;
2. using System.Collections.Generic;
3. using System.ComponentModel;
4. using System.Diagnostics;
5. using System.Drawing;
6. using System.Drawing.Design;
7. using System.Linq;
8. namespace Task05
9. {
10. public partial class RenderControl : OpenGL
11. {
12. Draw draw;
13. // Multiplier for zoom in and out
14. public double Multi { get; set; } = 1.0;
16. // Switching modes view (since I only have 2 mods, i decided to use bool type for check)
17. public bool Mode { get; set; } = true;
18. // Checking if light is on
19. public bool LightOn { get; set; } = true;
20. // Checking left mouse button pressed
21. public bool NotRotating { get; set; } = true;
22. // Checking right mouse button pressed
23. public bool NotPanning { get; set; } = true;
24. // Checking for fill mode
25. public bool Fill { get; set; } = true;
26. // Right and Left mouse buttons coordinates click
27. public double LastRightMouseX { get; set; } = 0.0;
28. public double LastRightMouseY { get; set; } = 0.0;
29. public double LastLeftMouseX { get; set; } = 0.0;
30. public double LastLeftMouseY { get; set; } = 0.0;
31. // Rotating the camera for perspective mode
32. public double CameraRadius { get; set; } = 10.0; // Distance to scene
33. public double CameraTheta { get; set; } = 0.0; // Angle of Y axis
34. public double CameraPhi { get; set; } = Math.PI / 2; // Angle up or down
35. // Rotating the camera for ortho mode
36. public double AngleX { get; set; } = 0.0;
37. public double AngleY { get; set; } = 0.0;
38. // It doesn't do anything so far, but i was planning to change the center of the camera scene earlier
39. public double CenterX { get; set; } = 0.0;
40. public double CenterY { get; set; } = 0.0;
41. public double CenterZ { get; set; } = 0.0;
42. // Change camera position without change the center of the camera scene
43. public double PanOffsetX { get; set; } = 0.0;
44. public double PanOffsetY { get; set; } = 0.0;
45. // This was necessary to change the initial position of the Z-axis, but is now not used
46. public double PanOffsetZ { get; set; } = 0.0;
47. // Elements of glCallList
48. private uint coordinatesDisplayList;
49. private uint gridDisplayList;
50. private uint sphereDisplayList;
51. private uint coneDisplayList;
52. private uint discDisplayList;
53. public RenderControl()
54. {
55. InitializeComponent();
56. }
57. private void OnRender(object sender, EventArgs e)
58. {
59. glClearColor(0.41f, 0.38f, 0.36f, 1.0f);
60. glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);
61. glEnable(GL\_DEPTH\_TEST);
62. glViewport(0, 0, Width, Height);
63. glMatrixMode(GL\_PROJECTION);
64. glLoadIdentity();
65. double size = 20;
66. double aspect = (double)Width / Height;
67. if (Mode)
68. {
69. gluPerspective(45.0, aspect, 0.1, 100.0);
70. }
71. else
72. {
73. glOrtho(-size \* Multi, size \* Multi, -size \* Multi, size \* Multi, -size \* 2.5 \* Multi, size \* 2.5 \* Multi);
74. }
75. glMatrixMode(GL\_MODELVIEW);
76. glLoadIdentity();
77. glTranslated(PanOffsetX, PanOffsetY, 0.0);
78. if (Mode)
79. {
80. double cameraX = CameraRadius \* Math.Sin(CameraPhi) \* Math.Cos(CameraTheta);
81. double cameraY = CameraRadius \* Math.Cos(CameraPhi);
82. double cameraZ = CameraRadius \* Math.Sin(CameraPhi) \* Math.Sin(CameraTheta);
83. gluLookAt(
84. Multi \* cameraX, Multi \* cameraY, Multi \* cameraZ, // Camera position
85. CenterX, CenterY, CenterZ, // Scene center
86. 0.0, 1.0, 0.0
87. );
88. }
89. else
90. {
91. glRotatef((float)AngleY, 1.0f, 0.0f, 0.0f);
92. glRotatef((float)AngleX, 0.0f, 1.0f, 0.0f);
93. }
94. if (LightOn)
95. SetupLighting();
96. else
97. {
98. glDisable(GL\_LIGHTING);
99. glDisable(GL\_LIGHT0);
100. glDisable(GL\_COLOR\_MATERIAL);
101. }
103. glCallList(gridDisplayList);
104. glCallList(coordinatesDisplayList);
105. //glCallList(sphereDisplayList);
106. //glCallList(coneDisplayList);
107. //glCallList(discDisplayList);
108. double scale = CameraRadius;
109. float lineWidth = Math.Clamp((float)(1.0 / scale), 0.1f, 5.0f);
110. glLineWidth(lineWidth);
111. draw.DrawSphere(2.5, -0.5, -2.5, 1.0, Fill);
112. draw.DrawCone(-3.5, -2.5, 2.5, 0.0, 2.5, 2.0, Fill);
113. draw.DrawDisc(4.5, 3.5, 4.5, 1.5, 4.0, Fill);
114. }
115. private void SetupLighting()
116. {
117. glEnable(GL\_LIGHTING);
118. glEnable(GL\_LIGHT0);
119. float[] lightPosition = { 5.0f, 5.0f, 5.0f, 1.0f };
120. glLightfv(GL\_LIGHT0, GL\_POSITION, lightPosition);
121. float[] lightAmbient = { 0.2f, 0.2f, 0.2f, 1.0f };
122. float[] lightDiffuse = { 0.8f, 0.8f, 0.8f, 1.0f };
123. float[] lightSpecular = { 1.0f, 1.0f, 1.0f, 1.0f };
124. glLightfv(GL\_LIGHT0, GL\_AMBIENT, lightAmbient);
125. glLightfv(GL\_LIGHT0, GL\_DIFFUSE, lightDiffuse);
126. glLightfv(GL\_LIGHT0, GL\_SPECULAR, lightSpecular);
127. glEnable(GL\_COLOR\_MATERIAL);
128. glColorMaterial(GL\_FRONT, GL\_AMBIENT\_AND\_DIFFUSE);
129. float[] materialSpecular = { 1.0f, 1.0f, 1.0f, 1.0f };
130. float materialShininess = 50.0f;
131. glMaterialfv(GL\_FRONT, GL\_SPECULAR, materialSpecular);
132. glMaterialf(GL\_FRONT, GL\_SHININESS, materialShininess);
133. }
134. private void InitializeDisplayLists()
135. {
136. coordinatesDisplayList = glGenLists(1);
137. glNewList(coordinatesDisplayList, GL\_COMPILE);
138. draw.DrawCoordinatesLines(CenterX, CenterY, CenterZ);
139. glEndList();
140. gridDisplayList = glGenLists(1);
141. glNewList(gridDisplayList, GL\_COMPILE);
142. draw.DrawGrid(20);
143. glEndList();
144. sphereDisplayList = glGenLists(1);
145. glNewList(sphereDisplayList, GL\_COMPILE);
146. draw.DrawSphere(2.5, -0.5, -2.5, 1.0, Fill);
147. glEndList();
148. coneDisplayList = glGenLists(1);
149. glNewList(coneDisplayList, GL\_COMPILE);
150. draw.DrawCone(-3.5, -2.5, 2.5, 0.0, 2.5, 2.0, Fill);
151. glEndList();
152. discDisplayList = glGenLists(1);
153. glNewList(discDisplayList, GL\_COMPILE);
154. draw.DrawDisc(4.5, 3.5, 4.5, 1.5, 4.0, Fill);
155. glEndList();
156. }
157. private void DeleteDisplayLists()
158. {
159. glDeleteLists(coordinatesDisplayList, 1);
160. glDeleteLists(gridDisplayList, 1);
161. glDeleteLists(sphereDisplayList, 1);
162. glDeleteLists(coneDisplayList, 1);
163. glDeleteLists(discDisplayList, 1);
164. }
165. private void OnCreate(object sender, EventArgs e)
166. {
167. draw = new Draw();
168. draw.Print = DrawText;
169. InitializeDisplayLists();
170. }
171. private void OnDelete(object sender, EventArgs e)
172. {
173. DeleteDisplayLists();
174. }
175. }
176. }

### Код файлу (Draw.cs)

1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Text;
5. using System.Threading.Tasks;
6. namespace Task05
7. {
8. public partial class RenderControl
9. {
11. public class Draw
12. {
13. public delegate void outText(string s, double x, double y, double z);
14. public outText Print;
15. public void DrawGrid(double size)
16. {
17. glEnable(GL\_LINE\_STIPPLE);
18. glLineStipple(1, 0xAAAA);
19. glLineWidth(0.5f);
20. glColor3ub(200, 200, 200);
21. glBegin(GL\_LINES);
22. for (double i = -size; i <= size; i += size / 10)
23. {
24. glVertex3d(i, 0, -size);
25. glVertex3d(i, 0, size);
26. }
27. for (double j = -size; j <= size; j += size / 10)
28. {
29. glVertex3d(-size, 0, j);
30. glVertex3d(size, 0, j);
31. }
32. glEnd();
33. glDisable(GL\_LINE\_STIPPLE);
34. }
35. public void DrawCoordinatesLines(double x, double y, double z)
36. {
37. glLineWidth(2.5f);
38. glBegin(GL\_LINES);
39. glColor3ub(200, 0, 0);
40. glVertex3d(x, y, z);
41. glVertex3d(x+2, y, z);
42. glColor3ub(0, 200, 0);
43. glVertex3d(x, y, z);
44. glVertex3d(x, y+2, z);
45. glColor3ub(0, 0, 200);
46. glVertex3d(x, y, z);
47. glVertex3d(x, y, z+2);
48. glEnd();
49. glColor3ub(0, 0, 0);
50. Print("x", x + 2, y, z);
51. Print("y", x, y + 2, z);
52. Print("z", x, y, z + 2);
53. }
54. public void DrawSphere(double x0, double y0, double z0, double radius, bool Fill, int slices = 20, int stacks = 20)
55. {
56. if (Fill)
57. {
58. glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_FILL);
59. }
60. else
61. {
62. glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE);
63. }
64. glPushMatrix();
65. glTranslated(x0, y0, z0);
66. for (int i = 0; i < stacks; i++)
67. {
68. double phi1 = Math.PI \* i / stacks;
69. double phi2 = Math.PI \* (i + 1) / stacks;
70. glBegin(GL\_QUAD\_STRIP);
71. for (int j = 0; j <= slices; j++)
72. {
73. double theta = 2.0 \* Math.PI \* j / slices;
74. double x1 = radius \* Math.Sin(phi1) \* Math.Cos(theta);
75. double y1 = radius \* Math.Cos(phi1);
76. double z1 = radius \* Math.Sin(phi1) \* Math.Sin(theta);
77. double x2 = radius \* Math.Sin(phi2) \* Math.Cos(theta);
78. double y2 = radius \* Math.Cos(phi2);
79. double z2 = radius \* Math.Sin(phi2) \* Math.Sin(theta);
80. glColor3ub(100, 100, 255);
81. glNormal3d(x1 / radius, y1 / radius, z1 / radius);
82. glVertex3d(x1, y1, z1);
83. glNormal3d(x2 / radius, y2 / radius, z2 / radius);
84. glVertex3d(x2, y2, z2);
85. }
86. glEnd();
87. }
88. glPopMatrix();
89. }
90. public void DrawCone(double x0, double y0, double z0, double radius, double radius\_small, double height, bool Fill, int slices = 20)
91. {
92. if (Fill)
93. {
94. glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_FILL);
95. }
96. else
97. {
98. glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE);
99. }
100. glPushMatrix();
101. glTranslated(x0, y0, z0);
102. double normalFactor = Math.Sqrt(radius\_small \* radius\_small + height \* height);
103. glBegin(GL\_TRIANGLE\_STRIP);
104. for (int i = 0; i <= slices; i++)
105. {
106. double theta = 2.0 \* Math.PI \* i / slices;
107. double x1 = radius\_small \* Math.Cos(theta);
108. double z1 = radius\_small \* Math.Sin(theta);
109. double x2 = radius \* Math.Cos(theta);
110. double z2 = radius \* Math.Sin(theta);
111. double nx = x1 / normalFactor;
112. double ny = height / normalFactor;
113. double nz = z1 / normalFactor;
114. glColor3ub(100, 150, 250);
115. glNormal3d(nx, ny, nz);
116. glVertex3d(x1, 0, z1);
117. glColor3ub(250, 100, 100);
118. glNormal3d(nx, ny, nz);
119. glVertex3d(x2, height, z2);
120. }
121. glEnd();
122. glBegin(GL\_POLYGON);
123. glColor3ub(100, 150, 250);
124. glNormal3d(0, -1, 0);
125. for (int i = 0; i <= slices; i++)
126. {
127. double theta = 2.0 \* Math.PI \* i / slices;
128. double x = radius\_small \* Math.Cos(theta);
129. double z = radius\_small \* Math.Sin(theta);
130. glVertex3d(x, 0, z);
131. }
132. glEnd();
133. glPopMatrix();
134. }
135. public void DrawDisc(double x0, double y0, double z0, double radius\_small, double radius, bool Fill, int slices = 40)
136. {
137. if (Fill)
138. {
139. glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_FILL);
140. }
141. else
142. {
143. glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE);
144. }
145. glPushMatrix();
146. glTranslated(x0, y0, z0);
147. glNormal3d(0, 1, 0);
148. glBegin(GL\_TRIANGLE\_STRIP);
149. for (int i = 0; i <= slices; i++)
150. {
151. double theta = 2.0 \* Math.PI \* i / slices;
152. double x1 = radius\_small \* Math.Cos(theta);
153. double z1 = radius\_small \* Math.Sin(theta);
154. double x2 = radius \* Math.Cos(theta);
155. double z2 = radius \* Math.Sin(theta);
156. glColor3ub(100, 200, 150);
157. glVertex3d(x1, 0, z1);
158. glVertex3d(x2, 0, z2);
159. }
160. glEnd();
161. glPopMatrix();
162. }
163. }
164. }
165. }

### Код файлу (MainForm.cs)

1. using System;
2. using System.Windows.Forms;
3. using static Task05.OpenGL;
4. namespace Task05
5. {
6. public partial class MainForm : Form
7. {
8. public MainForm()
9. {
10. InitializeComponent();
11. }
12. private void OnWheel(object sender, MouseEventArgs e)
13. {
14. if (e.Delta < 0)
15. {
16. renderControl1.Multi += 0.1;
17. }
18. else
19. {
20. if (renderControl1.Multi > 0.2)
21. {
22. renderControl1.Multi -= 0.1;
23. }
24. }
25. //if ((Control.ModifierKeys & Keys.Shift) != 0)
26. //{
27. // double deltaZ = e.Delta > 0 ? 0.5 : -0.5;
28. // renderControl1.CenterZ += deltaZ;
29. //}
30. //else
31. //{
32. // double zoomFactor = e.Delta > 0 ? 1.1 : 0.9;
33. // renderControl1.Multi \*= zoomFactor;
34. //}
35. renderControl1.Invalidate();
36. }
37. private void OnMouseDown(object sender, MouseEventArgs e)
38. {
39. if (renderControl1.NotRotating && e.Button == MouseButtons.Left)
40. {
41. renderControl1.NotRotating = false;
42. renderControl1.LastLeftMouseX = e.X;
43. renderControl1.LastLeftMouseY = e.Y;
44. }
45. if (renderControl1.NotPanning && e.Button == MouseButtons.Right)
46. {
47. renderControl1.NotPanning = false;
48. renderControl1.LastRightMouseX = e.X;
49. renderControl1.LastRightMouseY = e.Y;
50. }
51. }
52. private void OnMouseMove(object sender, MouseEventArgs e)
53. {
54. if (renderControl1.Mode)
55. {
56. if (!renderControl1.NotRotating)
57. {
58. double deltaX = e.X - renderControl1.LastLeftMouseX;
59. double deltaY = e.Y - renderControl1.LastLeftMouseY;
60. renderControl1.CameraTheta += deltaX \* 0.01;
61. renderControl1.CameraPhi -= deltaY \* 0.01;
62. renderControl1.CameraPhi = Math.Clamp(renderControl1.CameraPhi, 0.1, Math.PI - 0.1);
63. renderControl1.LastLeftMouseX = e.X;
64. renderControl1.LastLeftMouseY = e.Y;
65. renderControl1.Invalidate();
66. }
67. }
68. else
69. {
70. if (!renderControl1.NotRotating)
71. {
72. double deltaX = e.X - renderControl1.LastLeftMouseX;
73. double deltaY = e.Y - renderControl1.LastLeftMouseY;
74. // Changing axis angles
75. renderControl1.AngleX += deltaX \* 0.2;
76. renderControl1.AngleY += deltaY \* 0.2;
77. renderControl1.LastLeftMouseX = e.X;
78. renderControl1.LastLeftMouseY = e.Y;
79. renderControl1.Invalidate();
80. }
81. }
82. if (!renderControl1.NotPanning)
83. {
84. double deltaX = e.X - renderControl1.LastRightMouseX;
85. double deltaY = e.Y - renderControl1.LastRightMouseY;
86. renderControl1.PanOffsetX += deltaX \* 0.01;
87. renderControl1.PanOffsetY -= deltaY \* 0.01;
88. //renderControl1.CenterX = renderControl1.PanOffsetX;
89. //renderControl1.CenterY = renderControl1.PanOffsetY;
90. //if (renderControl1.CenterY < 0.0) renderControl1.CenterY = 0.0;
91. renderControl1.LastRightMouseX = e.X;
92. renderControl1.LastRightMouseY = e.Y;
93. renderControl1.Invalidate();
94. }
95. }
96. private void OnMouseUp(object sender, MouseEventArgs e)
97. {
98. if (!renderControl1.NotRotating && e.Button == MouseButtons.Left)
99. {
100. renderControl1.NotRotating = true;
101. }
102. if (!renderControl1.NotPanning && e.Button == MouseButtons.Right)
103. {
104. renderControl1.NotPanning = true;
105. }
106. }
107. private void OnChekedLight(object sender, EventArgs e)
108. {
109. renderControl1.LightOn = cbLight.Checked;
110. renderControl1.Invalidate();
111. }
112. private void OnCheckedOrtho(object sender, EventArgs e)
113. {
114. if (rdbtnOrtho.Checked)
115. {
116. renderControl1.Mode = false;
117. renderControl1.Invalidate();
118. }
119. }
120. private void OnCheckedPerspective(object sender, EventArgs e)
121. {
122. if (rdbtnPerspective.Checked)
123. {
124. renderControl1.Mode = true;
125. renderControl1.Invalidate();
126. }
127. }
128. private void OnCheckedFill(object sender, EventArgs e)
129. {
130. if (rdbtnFill.Checked)
131. {
132. renderControl1.Fill = true;
133. renderControl1.Invalidate();
134. }
135. }
136. private void OnCheckedLine(object sender, EventArgs e)
137. {
138. if (rdbtnLine.Checked)
139. {
140. renderControl1.Fill = false;
141. renderControl1.Invalidate();
142. }
143. }
144. }
145. }

# Додаток Д. Лістинг програми до практичної роботи №6

### Код файлу (RenderControl.cs)

1. using System;
2. using System.Collections.Generic;
3. using System.ComponentModel;
4. using System.Diagnostics;
5. using System.Drawing;
6. using System.Drawing.Design;
7. using System.Linq;
8. namespace Task06
9. {
10. public partial class RenderControl : OpenGL
11. {
12. Draw draw;
13. public bool NotRotating { get; set; } = true;
14. public bool FigureChanging { get; set; } = false;
15. public bool FigureRotating { get; set; } = false;
16. public double Multi { get; set; } = 1.0;
17. public double LastRightMouseX { get; set; } = 0.0;
18. public double LastLeftMouseX { get; set; } = 0.0;
19. public double LastLeftMouseY { get; set; } = 0.0;
20. public double CameraRadius { get; set; } = 10.0;
21. public double CameraTheta { get; set; } = 0.0;
22. public double CameraPhi { get; set; } = Math.PI / 2;
23. public double PanOffsetX { get; set; } = 0.0;
24. public double PanOffsetY { get; set; } = 0.0;
25. public double sideA { get; set; } = 0.9;
26. public double sideB { get; set; } = 0.24;
27. public double sideC { get; set; } = 0.64;
28. public double PointBY { get; set; } = 0.4;
29. public double RotationAngleY { get; set; } = 0.0;
30. public RenderControl()
31. {
32. InitializeComponent();
33. }
34. private void OnRender(object sender, EventArgs e)
35. {
36. glClearColor(0.41f, 0.38f, 0.36f, 1.0f);
37. glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);
38. glEnable(GL\_DEPTH\_TEST);
39. glViewport(0, 0, Width, Height);
40. glMatrixMode(GL\_PROJECTION);
41. glLoadIdentity();
42. double aspect = (double)Width / Height;
43. gluPerspective(45.0, aspect, 0.1, 100.0);
44. glMatrixMode(GL\_MODELVIEW);
45. glLoadIdentity();
46. glTranslated(PanOffsetX, PanOffsetY, 0.0);
47. double cameraX = CameraRadius \* Math.Sin(CameraPhi) \* Math.Cos(CameraTheta);
48. double cameraY = CameraRadius \* Math.Cos(CameraPhi);
49. double cameraZ = CameraRadius \* Math.Sin(CameraPhi) \* Math.Sin(CameraTheta);
50. gluLookAt(
51. Multi \* cameraX, Multi \* cameraY, Multi \* cameraZ,
52. 0.0, 0.0, 0.0,
53. 0.0, 1.0, 0.0
54. );
55. draw.DrawCoordinatesLines(0, 0, 0);
56. SetupLighting();
57. draw.DrawGrid(10);
58. FigureChanging = draw.DrawFuncrtionalityFigure(PointBY, sideA, sideB, sideC, RotationAngleY, FigureChanging);
59. }
60. private void OnCreated(object sender, EventArgs e)
61. {
62. draw = new Draw();
63. draw.Print = DrawText;
64. }
65. private void SetupLighting()
66. {
67. glEnable(GL\_LIGHTING);
68. glEnable(GL\_LIGHT0);
69. float[] lightPosition = { 5.0f, 5.0f, 5.0f, 1.0f };
70. glLightfv(GL\_LIGHT0, GL\_POSITION, lightPosition);
71. float[] lightAmbient = { 0.2f, 0.2f, 0.2f, 1.0f };
72. float[] lightDiffuse = { 0.8f, 0.8f, 0.8f, 1.0f };
73. float[] lightSpecular = { 1.0f, 1.0f, 1.0f, 1.0f };
74. glLightfv(GL\_LIGHT0, GL\_AMBIENT, lightAmbient);
75. glLightfv(GL\_LIGHT0, GL\_DIFFUSE, lightDiffuse);
76. glLightfv(GL\_LIGHT0, GL\_SPECULAR, lightSpecular);
77. glEnable(GL\_COLOR\_MATERIAL);
78. glColorMaterial(GL\_FRONT, GL\_AMBIENT\_AND\_DIFFUSE);
79. float[] materialSpecular = { 1.0f, 1.0f, 1.0f, 1.0f };
80. float materialShininess = 50.0f;
81. glMaterialfv(GL\_FRONT, GL\_SPECULAR, materialSpecular);
82. glMaterialf(GL\_FRONT, GL\_SHININESS, materialShininess);
83. }
84. }
85. }

### Код файлу (MainForm.cs)

1. using System;
2. using System.Windows.Forms;
3. using static Task06.OpenGL;
4. namespace Task06
5. {
6. public partial class MainForm : Form
7. {
8. public MainForm()
9. {
10. InitializeComponent();
11. }
12. private void OnWheel(object sender, MouseEventArgs e)
13. {
14. if (e.Delta < 0)
15. {
16. renderControl1.Multi += 0.1;
17. }
18. else
19. {
20. if (renderControl1.Multi > 0.2)
21. {
22. renderControl1.Multi -= 0.1;
23. }
24. }
25. renderControl1.Invalidate();
26. }
27. private void OnMoudeDown(object sender, MouseEventArgs e)
28. {
29. if (e.Button == MouseButtons.Right && (Control.ModifierKeys & Keys.Control) == Keys.Control)
30. {
31. renderControl1.FigureRotating = true;
32. renderControl1.FigureChanging = false;
33. renderControl1.LastRightMouseX = e.X;
34. renderControl1.NotRotating = true;
35. }
36. if (e.Button == MouseButtons.Left && (Control.ModifierKeys & Keys.Control) == Keys.Control)
37. {
38. renderControl1.FigureChanging = true;
39. renderControl1.LastLeftMouseY = e.Y;
40. renderControl1.NotRotating = true;
41. }
42. else
43. {
44. if (renderControl1.NotRotating && e.Button == MouseButtons.Left)
45. {
46. renderControl1.NotRotating = false;
47. renderControl1.FigureChanging = false;
48. renderControl1.LastLeftMouseX = e.X;
49. renderControl1.LastLeftMouseY = e.Y;
50. }
51. }
52. renderControl1.Invalidate();
53. }
54. private void OnMouseUp(object sender, MouseEventArgs e)
55. {
56. if (e.Button == MouseButtons.Right && (Control.ModifierKeys & Keys.Control) == Keys.Control)
57. {
58. renderControl1.FigureRotating = false;
59. renderControl1.FigureChanging = false;
60. renderControl1.NotRotating = true;
61. }
62. if (e.Button == MouseButtons.Left && (Control.ModifierKeys & Keys.Control) == Keys.Control)
63. {
64. renderControl1.FigureChanging = false;
65. renderControl1.FigureRotating = false;
66. renderControl1.NotRotating = true;
67. }
68. else
69. {
70. if (!renderControl1.NotRotating && e.Button == MouseButtons.Left)
71. {
72. renderControl1.NotRotating = true;
73. }
74. }
75. renderControl1.Invalidate();
76. }
77. private void OnMouseMove(object sender, MouseEventArgs e)
78. {
79. if (e.Button == MouseButtons.Right && (Control.ModifierKeys & Keys.Control) == Keys.Control)
80. {
81. if (renderControl1.FigureRotating)
82. {
83. double deltaX = (e.X - renderControl1.LastRightMouseX) \* 0.5;
84. renderControl1.RotationAngleY = renderControl1.RotationAngleY - deltaX;
85. renderControl1.LastRightMouseX = e.X;
86. renderControl1.Invalidate();
87. }
88. }
89. if (e.Button == MouseButtons.Left && (Control.ModifierKeys & Keys.Control) == Keys.Control)
90. {
91. if (renderControl1.FigureChanging)
92. {
93. double deltaY = (e.Y - renderControl1.LastLeftMouseY) \* 0.01;
94. renderControl1.PointBY = Math.Clamp(renderControl1.PointBY - deltaY, 0.05, 0.48);
95. renderControl1.LastLeftMouseY = e.Y;
96. renderControl1.Invalidate();
97. }
98. }
99. if (!renderControl1.NotRotating)
100. {
101. double deltaX = e.X - renderControl1.LastLeftMouseX;
102. double deltaY = e.Y - renderControl1.LastLeftMouseY;
103. renderControl1.CameraTheta += deltaX \* 0.01;
104. renderControl1.CameraPhi -= deltaY \* 0.01;
105. renderControl1.CameraPhi = Math.Clamp(renderControl1.CameraPhi, 0.1, Math.PI - 0.1);
106. renderControl1.LastLeftMouseX = e.X;
107. renderControl1.LastLeftMouseY = e.Y;
108. renderControl1.Invalidate();
109. }
110. }
111. }
112. }

### Код файлу (Draw.cs)

1. using System;
2. using System.Collections.Generic;
3. using System.Data;
4. using System.Linq;
5. using System.Text;
6. using System.Threading.Tasks;
7. namespace Task06
8. {
9. public partial class RenderControl
10. {
11. public class Draw
12. {
13. public delegate void outText(string s, double x, double y, double z);
14. public outText Print;
15. public void DrawGrid(double size)
16. {
17. glEnable(GL\_LINE\_STIPPLE);
18. glLineStipple(1, 0xAAAA);
19. glLineWidth(0.5f);
20. glColor3ub(200, 200, 200);
21. glBegin(GL\_LINES);
22. for (double i = -size; i <= size; i += size / 10)
23. {
24. glVertex3d(i, 0, -size);
25. glVertex3d(i, 0, size);
26. }
27. for (double j = -size; j <= size; j += size / 10)
28. {
29. glVertex3d(-size, 0, j);
30. glVertex3d(size, 0, j);
31. }
32. glEnd();
33. glDisable(GL\_LINE\_STIPPLE);
34. }
35. public void DrawCoordinatesLines(double x, double y, double z)
36. {
37. double size = 0.4;
38. glDisable(GL\_LIGHTING);
39. glDisable(GL\_LIGHT0);
40. glDisable(GL\_COLOR\_MATERIAL);
41. glLineWidth(2.5f);
42. glBegin(GL\_LINES);
43. glColor3ub(200, 0, 0);
44. glVertex3d(x, y, z);
45. glVertex3d(x + size, y, z);
46. glColor3ub(0, 200, 0);
47. glVertex3d(x, y, z);
48. glVertex3d(x, y + size, z);
49. glColor3ub(0, 0, 200);
50. glVertex3d(x, y, z);
51. glVertex3d(x, y, z + size);
52. glEnd();
53. glColor3ub(0, 0, 0);
54. Print("x", x + size, y, z);
55. Print("y", x, y + size, z);
56. Print("z", x, y, z + size);
57. }
58. public bool DrawFuncrtionalityFigure(double PointBY, double sideA, double sideB, double sideC, double rotationAngleY, bool figureChanging)
59. {
60. glPushMatrix();
61. glRotated(rotationAngleY, 0.0, 0.0, 1.0);
62. double xB = 0.0, yB = PointBY;
63. double xC = 0.0, yC = sideA;
64. double xD1, yD1, xA, yA;
65. double dx = xB - xC;
66. double dy = yB - yC;
67. double d = Math.Sqrt(dx \* dx + dy \* dy);
68. if (d > sideB + sideC || d < Math.Abs(sideB - sideC))
69. {
70. glPopMatrix();
71. return false;
72. }
73. double a = (sideB \* sideB - sideC \* sideC + d \* d) / (2 \* d);
74. double h = Math.Sqrt(sideB \* sideB - a \* a);
75. double xMid = xC + (a / d) \* dx;
76. double yMid = yC + (a / d) \* dy;
77. xD1 = xMid + (h / d) \* (-dy);
78. yD1 = yMid + (h / d) \* dx;
79. double vectorCDx = xD1 - xC;
80. double vectorCDy = yD1 - yC;
81. double vectorLength = Math.Sqrt(vectorCDx \* vectorCDx + vectorCDy \* vectorCDy);
82. double unitVectorX = vectorCDx / vectorLength;
83. double unitVectorY = vectorCDy / vectorLength;
84. double distanceToA = 3 \* sideB;
85. xA = xD1 + unitVectorX \* distanceToA;
86. yA = yD1 + unitVectorY \* distanceToA;
87. glColor3d(0, 0, 0);
88. // OC
89. DrawThickCube(0.0, 0.0, xC, yC);
90. // BD
91. DrawThickCube(xB, yB, xD1, yD1);
92. // CA
93. DrawThickCube(xC, yC, xA, yA);
94. glPopMatrix();
95. return figureChanging;
96. }
97. private void DrawThickCube(double x1, double y1, double x2, double y2)
98. {
99. double thickness = 0.02;
100. double dxLine = x2 - x1;
101. double dyLine = y2 - y1;
102. double length = Math.Sqrt(dxLine \* dxLine + dyLine \* dyLine);
103. double offsetX = -(dyLine / length) \* thickness;
104. double offsetY = (dxLine / length) \* thickness;
105. glBegin(GL\_QUADS);
106. glVertex3d(x1 - offsetX, y1 - offsetY, -thickness);
107. glVertex3d(x1 + offsetX, y1 + offsetY, -thickness);
108. glVertex3d(x2 + offsetX, y2 + offsetY, -thickness);
109. glVertex3d(x2 - offsetX, y2 - offsetY, -thickness);
110. glVertex3d(x1 - offsetX, y1 - offsetY, thickness);
111. glVertex3d(x1 + offsetX, y1 + offsetY, thickness);
112. glVertex3d(x2 + offsetX, y2 + offsetY, thickness);
113. glVertex3d(x2 - offsetX, y2 - offsetY, thickness);
114. glEnd();
115. glBegin(GL\_QUADS);
116. glVertex3d(x1 - offsetX, y1 - offsetY, -thickness);
117. glVertex3d(x1 - offsetX, y1 - offsetY, thickness);
118. glVertex3d(x2 - offsetX, y2 - offsetY, thickness);
119. glVertex3d(x2 - offsetX, y2 - offsetY, -thickness);
120. glVertex3d(x1 + offsetX, y1 + offsetY, -thickness);
121. glVertex3d(x1 + offsetX, y1 + offsetY, thickness);
122. glVertex3d(x2 + offsetX, y2 + offsetY, thickness);
123. glVertex3d(x2 + offsetX, y2 + offsetY, -thickness);
124. glVertex3d(x1 - offsetX, y1 - offsetY, thickness);
125. glVertex3d(x1 + offsetX, y1 + offsetY, thickness);
126. glVertex3d(x2 + offsetX, y2 + offsetY, thickness);
127. glVertex3d(x2 - offsetX, y2 - offsetY, thickness);
128. glVertex3d(x1 - offsetX, y1 - offsetY, -thickness);
129. glVertex3d(x1 + offsetX, y1 + offsetY, -thickness);
130. glVertex3d(x2 + offsetX, y2 + offsetY, -thickness);
131. glVertex3d(x2 - offsetX, y2 - offsetY, -thickness);
132. glEnd();
133. }
134. }
135. }
136. }

# Додаток Е. Лістинг програми до практичної роботи №7

### Код файлу (RenderControl.cs)

1. using System;
2. using System.Drawing;
3. namespace test1
4. {
5. public partial class RenderControl : OpenGL
6. {
7. public RenderControl()
8. {
9. InitializeComponent();
10. }
11. private double offsetX = 0.0, offsetY = 0.0;
12. private double cx = -0.7, cy = 0.27015;
13. private double cxStep = 0.002, cyStep = 0.002;
14. private bool isMovingEnabled = true;
15. private double speedValue = 0.005;
16. private double sizeValue = 1.0;
17. private void OnContextCreated(object sender, EventArgs e)
18. {
19. glClearColor(Color.Black);
20. isMovingEnabled = Properties.Settings.Default.IsFloatingFigure;
21. cx = (double)Properties.Settings.Default.CxValue;
22. cy = (double)Properties.Settings.Default.CyValue;
23. speedValue = 0.001 \* Properties.Settings.Default.SpeedValue;
24. sizeValue = 0.5 \* Properties.Settings.Default.SizeValue;
25. }
26. private void OnRender(object sender, EventArgs e)
27. {
28. glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);
29. glLoadIdentity();
30. double viewLeft = -2 + offsetX;
31. double viewRight = 2 + offsetX;
32. double viewBottom = -2 + offsetY;
33. double viewTop = 2 + offsetY;
34. glOrtho(viewLeft, viewRight, viewBottom, viewTop, -1, 1);
35. DrawJuliaSet();
36. if (isMovingEnabled)
37. {
38. offsetX += speedValue;
39. offsetY += speedValue;
40. if (offsetX > 2 || offsetX < -2) speedValue = -speedValue;
41. cx += cxStep;
42. cy += cyStep;
43. if (cx > 0.7 || cx < -0.7) cxStep = -cxStep;
44. if (cy > 0.7 || cy < -0.7) cyStep = -cyStep;
45. }
46. }
47. private void DrawJuliaSet()
48. {
49. int resolution = 500;
50. double scale = 4.0 / resolution;
51. for (int i = 0; i < resolution; i++)
52. {
53. for (int j = 0; j < resolution; j++)
54. {
55. double x = (i - resolution / 2.0) \* scale + offsetX;
56. double y = (j - resolution / 2.0) \* scale + offsetY;
57. int iterations = ComputeJulia(x, y, cx, cy);
58. double color = iterations / 100.0;
59. double pointSize = sizeValue;
60. glPointSize((float)pointSize);
61. glColor3d(color, color \* 0.5, color \* 0.8);
62. glBegin(GL\_POINTS);
63. glVertex2d(x, y);
64. glEnd();
65. }
66. }
67. }
68. private int ComputeJulia(double x, double y, double cx, double cy)
69. {
70. int maxIterations = 100;
71. int iteration = 0;
72. while (x \* x + y \* y < 4 && iteration < maxIterations)
73. {
74. double xtemp = x \* x - y \* y + cx;
75. y = 2 \* x \* y + cy;
76. x = xtemp;
77. iteration++;
78. }
79. return iteration;
80. }
81. }
82. }

### Код файлу (SettingForm.cs)

1. using System;
2. using System.Diagnostics;
3. using System.Windows.Forms;
4. namespace test1
5. {
6. public partial class SettingsForm : Form
7. {
8. public SettingsForm()
9. {
10. InitializeComponent();
11. }
12. private void SettingsFormLoad(object sender, EventArgs e)
13. {
14. chkbxCustomFigure.Checked = Properties.Settings.Default.IsCustomFigure;
15. nudCx.Value = Properties.Settings.Default.CxValue;
16. nudCy.Value = Properties.Settings.Default.CyValue;
17. chkbxFloatFigure.Checked = Properties.Settings.Default.IsFloatingFigure;
18. trbarSpeed.Value = Properties.Settings.Default.SpeedValue;
19. trbarSize.Value = Properties.Settings.Default.SizeValue;
20. Debug.WriteLine("Load screen saver's settings.");
21. }
22. private void SettingsFormClosed(object sender, FormClosedEventArgs e)
23. {
24. Properties.Settings.Default.IsCustomFigure = chkbxCustomFigure.Checked;
25. Properties.Settings.Default.CxValue = nudCx.Value;
26. Properties.Settings.Default.CyValue = nudCy.Value;
27. Properties.Settings.Default.IsFloatingFigure = chkbxFloatFigure.Checked;
28. Properties.Settings.Default.SpeedValue = trbarSpeed.Value;
29. Properties.Settings.Default.SizeValue = trbarSize.Value;
30. Properties.Settings.Default.Save();
31. Debug.WriteLine("Save screen saver's settings.");
32. }
33. private void OnChangedCustom(object sender, EventArgs e)
34. {
35. if (chkbxCustomFigure.Checked)
36. {
37. nudCx.Enabled = true;
38. nudCy.Enabled = true;
39. }
40. else
41. {
42. nudCx.Enabled = false;
43. nudCy.Enabled = false;
44. }
45. }
46. private void OnChangedFloating(object sender, EventArgs e)
47. {
48. if (chkbxFloatFigure.Checked)
49. {
50. trbarSpeed.Enabled = true;
51. }
52. else
53. {
54. trbarSpeed.Enabled = false;
55. }
56. }
57. private void OnClickOkBtn(object sender, EventArgs e)
58. {
59. Properties.Settings.Default.IsCustomFigure = chkbxCustomFigure.Checked;
60. Properties.Settings.Default.CxValue = nudCx.Value;
61. Properties.Settings.Default.CyValue = nudCy.Value;
62. Properties.Settings.Default.IsFloatingFigure = chkbxFloatFigure.Checked;
63. Properties.Settings.Default.SpeedValue = trbarSpeed.Value;
64. Properties.Settings.Default.SizeValue = trbarSize.Value;
65. Properties.Settings.Default.Save();
66. }
67. private void OnClickCancelBtn(object sender, EventArgs e)
68. {
69. chkbxCustomFigure.Checked = Properties.Settings.Default.IsCustomFigure;
70. nudCx.Value = Properties.Settings.Default.CxValue;
71. nudCy.Value = Properties.Settings.Default.CyValue;
72. chkbxFloatFigure.Checked = Properties.Settings.Default.IsFloatingFigure;
73. trbarSpeed.Value = Properties.Settings.Default.SpeedValue;
74. trbarSize.Value = Properties.Settings.Default.SizeValue;
75. }
76. }
77. }

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