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  *326ст*

напряму підготовки (спеціальності):

*122 Комп’ютерні науки*

(шифр і назва напряму підготовки / спеціальності)

*Цвірко М.С.*

(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](#_Toc15469)

[Завдання, варіант № 24 4](#_Toc27287)

[Системна інформація 4](#_Toc11158)

[Теоретичні відомості 4](#_Toc149)

[Результати виконання практичної роботи 6](#_Toc2643)

[Практична робота 2. Графічні примітиви OpenGL 8](#_Toc8410)

[Завдання, варіант № 24 8](#_Toc6196)

[Системна інформація 10](#_Toc16067)

[Теоретичні відомості 10](#_Toc11500)

[Результати виконання практичної роботи 11](#_Toc26083)

[Практична робота 3. Графік функції однієї змінної 17](#_Toc30175)

[Завдання, варіант № 24 17](#_Toc1463)

[Системна інформація 17](#_Toc1402)

[Теоретичні відомості 18](#_Toc27709)

[Результати виконання практичної роботи 19](#_Toc27137)

[Практична робота 4. Криві другого порядку 23](#_Toc28156)

[Завдання, варіант № 24 23](#_Toc28648)

[Системна інформація 23](#_Toc28535)

[Теоретичні відомості 23](#_Toc5123)

[Результати виконання практичної роботи 24](#_Toc25386)

[Практична робота 5. Квадратичні примітиви. Афінні перетворення у просторі 28](#_Toc3074)

[Завдання, варіант № 24 28](#_Toc10370)

[Системна інформація 28](#_Toc16413)

[Теоретичні відомості 29](#_Toc23400)

[Результати виконання практичної роботи 30](#_Toc9954)

[Практична робота 6. Візуалізація прямої задачі кінематики 34](#_Toc24922)

[Завдання, варіант № 24 34](#_Toc10286)

[Системна інформація 34](#_Toc22698)

[2 Математична модель кінематичної схеми 34](#_Toc27624)

[2.1 Афінні перетворення 34](#_Toc1095)

[2.2 Декомпозиція на сегменти 35](#_Toc17488)

[2.2.1 Математична модель сегмента «2а» 35](#_Toc10873)

[2.2.1 Математична модель сегмента «2а1» 35](#_Toc40)

[2.2.2 Математична модель сегмента «b» 36](#_Toc13773)

[2.3 Фізичні обмеження моделі 37](#_Toc17208)

[2.4 Загальна тривимірна математична модель 37](#_Toc22679)

[3 Програмна реалізація кінематичної схеми 38](#_Toc17799)

[3.1 Початкові дані 38](#_Toc1471)

[3.2 Прив'язка системи координат до видимої області вікна 38](#_Toc17778)

[3.3 Керування об'єктом моделювання та організація інтерфейсу 39](#_Toc1267)

[3.4 Осі координат 41](#_Toc2959)

[3.5 Виведення сегмента 41](#_Toc9620)

[3.6 Порядок виконання афінних перетворень 42](#_Toc5743)

[4 Результати візуалізації кінематичної схеми 44](#_Toc28112)

[Практична робота 7. ЕКРАННА ЗАСТАВКА З АНІМАЦІЄЮ 47](#_Toc30149)

[Завдання, варіант № 24 47](#_Toc20425)

[Теоретичні відомості 47](#_Toc22402)

[Системна інформація 49](#_Toc31931)

[Результати виконання практичної роботи 49](#_Toc7890)

[Загальний перелік посилань 52](#_Toc28281)

[Додаток А. Лістинг програми до практичної роботи №1 53](#_Toc23599)

[Додаток B. Лістинг програми до практичної роботи №2 58](#_Toc16832)

[Додаток C. Лістинг програми до практичної роботи №3 63](#_Toc30434)

[Додаток D. Лістинг програми до практичної роботи №4 74](#_Toc4142)

[Додаток E. Лістинг програми до практичної роботи №5 84](#_Toc7077)

[Додаток F. Лістинг програми до практичної роботи №6 94](#_Toc11997)

[Додаток G. Лістинг програми до практичної роботи №7 100](#_Toc24906)

# Практична робота 1. Основні принципи роботи з OpenGL

## Завдання, варіант № 24

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.

|  |  |  |
| --- | --- | --- |
| 24 | Примітиви:  GL\_POINTS, GL\_LINE\_LOOP  x1 = -8.5; x2 = 0.5  y1 = -3.5; y2 = 0.5 |  |

## Системна інформація

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

Processor AMD Ryzen 5 2600 3.40 GHz

RAM 16.0 GB (15.9 GB usable)

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

Edition Windows 11 Version 23H2

IDE Microsoft Visual Studio Enterprise 2022 (64-bit) version 17.11.5

## Теоретичні відомості

Виведення графічних об'єктів на екран може виконуватися в ізотропному (isotropic) або анізотропному (anisotropic) режимі ідображення, кожен з яких допускає зміну напряму осей X та Y, а також зміну масштабу осей координат. В ізотропному режимі відображення масштаб вздовж осей X і Y завжди однаковий (тобто обох осей використовуються однакові логічні одиниці довжини). Анізотропний режим передбачає використання різних масштабів для різних осей (хоч і однакові масштаби допускаються).

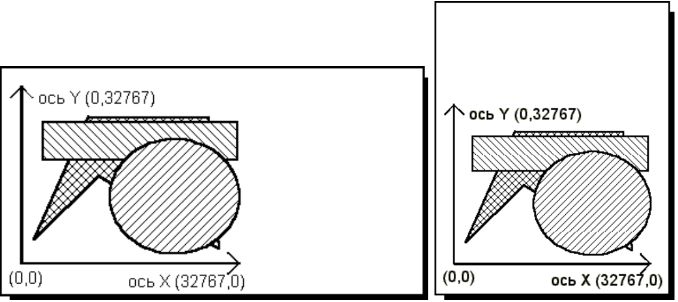


Рис. 1.1. Збереження пропорцій фігур в ізотропному режимі

В ізотропному режимі відображення (рис. 1.1) при зміні розмірів вікна Windows система координат налаштовується таким чином, щоб масштаб по осях X та Y був однаковий. Якщо висота вікна не дорівнює його ширині, то при використанні ізотропного режиму відображення для обчислення масштабу використовується менша зі сторін. При використанні анізотропного режиму налаштування масштабу по кожній осі виконується незалежно, тому область відображення займатиме всю робочу область вікна при будь-якій зміні розмірів цього вікна (рис. 1.2).

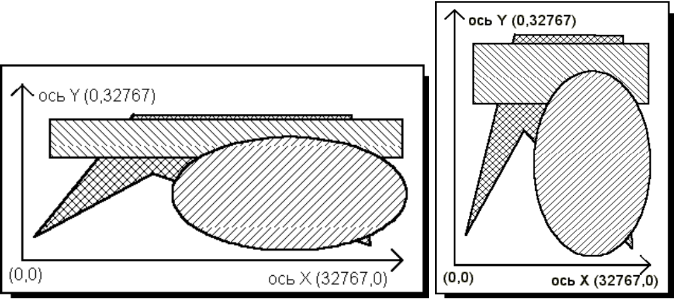


Рис. 1.2. Зміна пропорцій фігур в аніізотропному режимі

Ізотропний режим відображення використовується в тих випадках, коли важливо зберегти пропорції об'єктів (тобто квадрат і окружність повинні залишитися такими, а не трансформуватися в прямокутник і еліпс відповідно) при будь-якій зміні розмірів вікна, в яке виводиться зображення. Анізотропний режим затребуваний у тих випадках, коли зображення повинне займати всю внутрішню поверхню вікна за будь-якої зміни розмірів вікна, коли співвідношення масштабів по осях не має значення.

## Результати виконання практичної роботи

### Розв'язання завдання

Для управління параметрами графічних примітивів було викорастані наступні команди:

* товщина для ліній фігури glLineWidth(), рядок 43, файл DrawFigure.cs;
* тип для курсивних ліній glLineStipple(), glEnable()/glDisable(), рядок 29, файл DrawGrid.cs;

Коректне відображення завдання під час змінення розмірів/положення вікна наведено у рис 1.3 та 1.4.

Застосування циклів для створення зображень наведено у рядках 41 - 67 файлу Draw.cs.

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

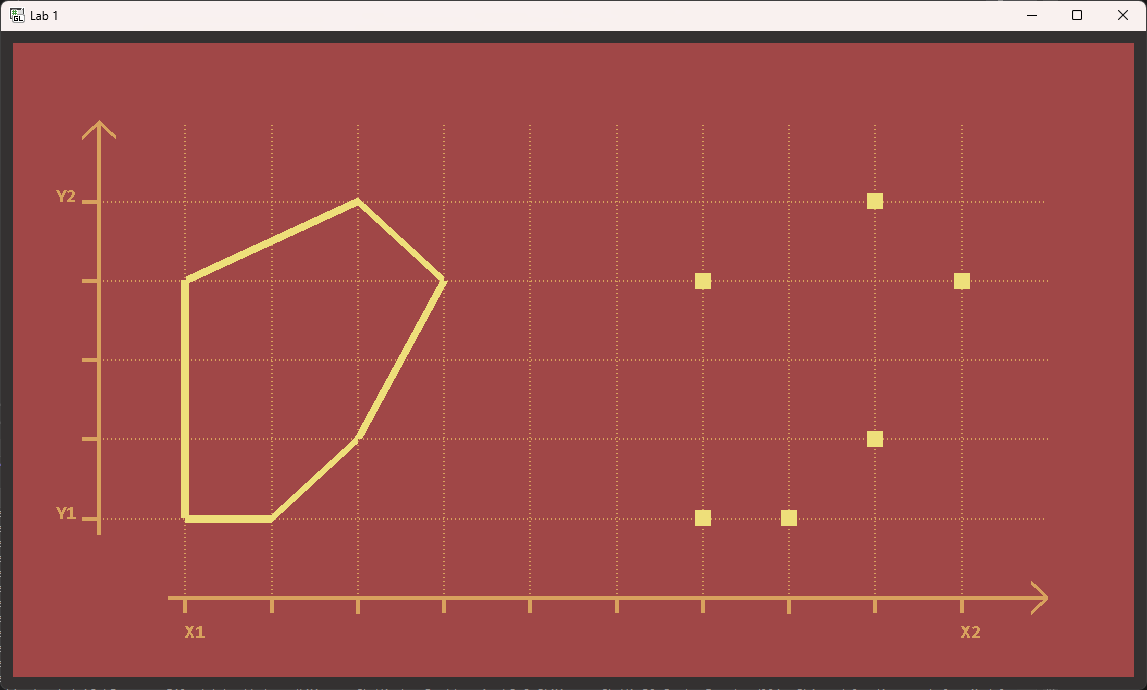


Рисунок 1.3 – Тестування програми при зміні ширини вікна

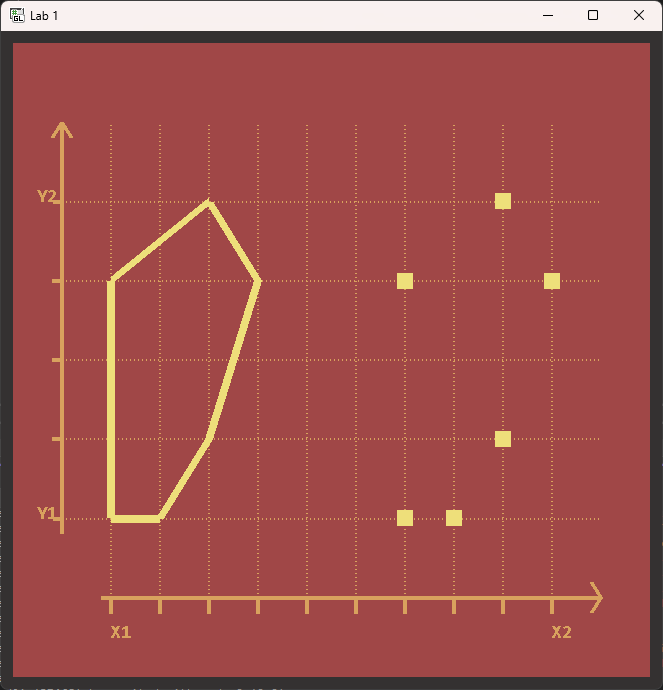


Рисунок 1.4 – Тестування програми при зміні ширини вікна

### 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 № 24

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

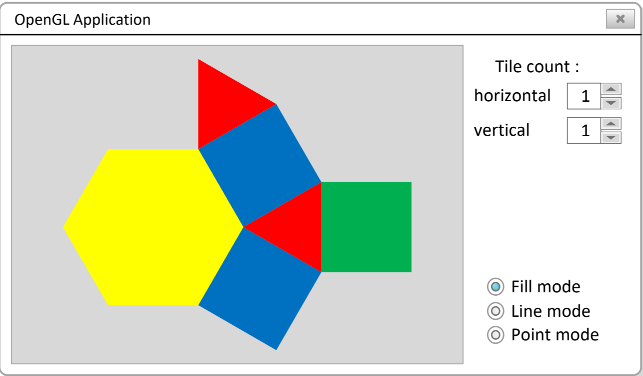
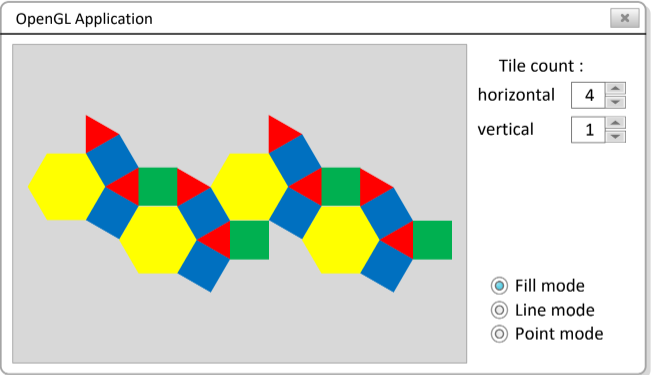


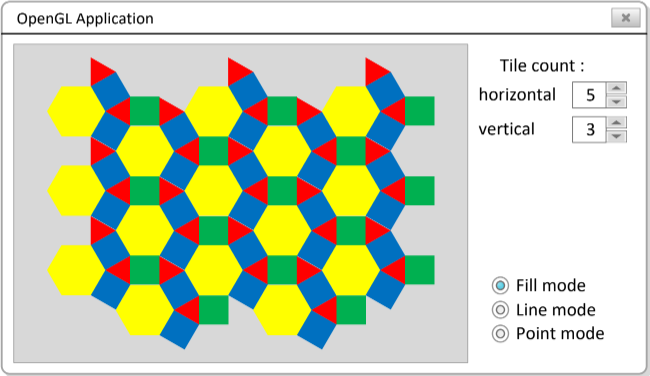
Рисунок 2.1 – Вигляд застосунку після старту

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

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



a



b

Рисунок 2.2 – Вигляд застосунку під час замощення:

а – тільки по горизонталі; б – по горизонталі та вертикалі

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

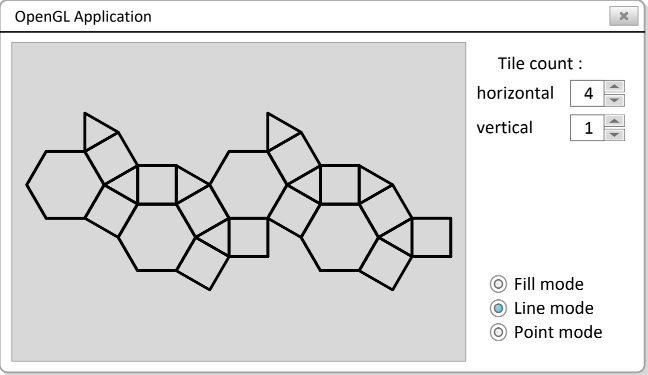


Рисунок 2.3 – Приклад керування режимом виводу графічних примітивів

|  |  |  |
| --- | --- | --- |
| 24 | Сторона фігури a = 10  Примітив(и):  GL\_POLYGON  GL\_TRIANGLE\_STRIP |  |

## Системна інформація

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

Processor AMD Ryzen 5 2600 3.40 GHz

RAM 16.0 GB (15.9 GB usable)

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

Edition Windows 11 Version 23H2

IDE Microsoft Visual Studio Enterprise 2022 (64-bit) version 17.11.5

## Теоретичні відомості

Під час виводу зображення необхідно враховувати, що кожна поверхня графічного примітиву OpenGL має дві сторони і режим виводу для кожної з них може бути налаштований окремо за допомогою команди **glPolygonMode**.

Для зміни режиму (моделі) розфарбування використовують команду **glShadeModel**. Якщо режим зафарбування напівтонами вимкнений, то колір примітиву визначається кольором тільки однієї вершини. Наприклад, для **GL\_TRIANGLE\_STRIP** колір першого трикутника визначається кольором третьої вершини, другого – четвертої вершини і т. д.

Для встановлення шаблону зафарбування необхідно задіяти команди-перемикачі **glEnable** / **glDisable** (як і для шаблону лінії).

## Результати виконання практичної роботи

### Розв'язання завдання

Під час запуску застосунку зображення відповідає варіанту завдання однією плиткою, рисунок **2.4**, рядки **49 - 123**, файл **DrawFigures.cs**, **додаток В**.

Багаторазове заміщення плиткою, рисунок **2.5**, рядки **32 - 38**. Кратність замощення задається користувачем, рядки **51 - 60**. Файл **DrawFigures.cs**, **додаток В**.

Коректне відображення завдання під час зміни як розмірів/положення вікна, так і параметрів замощення, рисунок **2.6 - 2.7**, рядки **30 - 40**, **47 - 65**, файл **RenderControl.cs**, **додаток В**.

Організація взаємодії з користувачем одним зі стандартних засобів, рисунок **2.8**, рядки **10 - 50**, файл **MainForm.cs**, **додаток В**.

Застосування мінімальної кількості графічних примітивів, рядки **68 - 122**, файл **FrawFigures.cs**, **додаток В**.

Керування режимом виводу графічних примітивів, рисунок **2.9 - 3.1**, рядки **62**, файл **DrawFigures.cs**, **додаток В**.

Керування режимом шейдингу графічних примітивів, рисунок **3.2 - 3.3**, рядки **63**, файл **DrawFigures.cs**, **додаток В**.

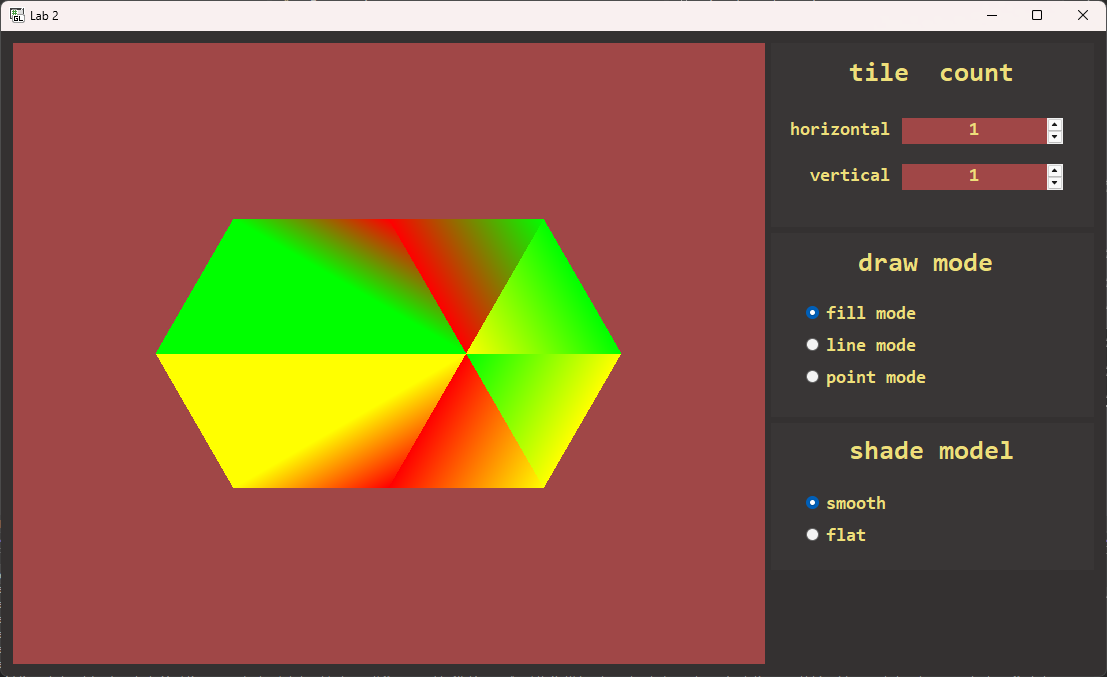


Рисунок 2.4 – Вигляд застосунку після старту

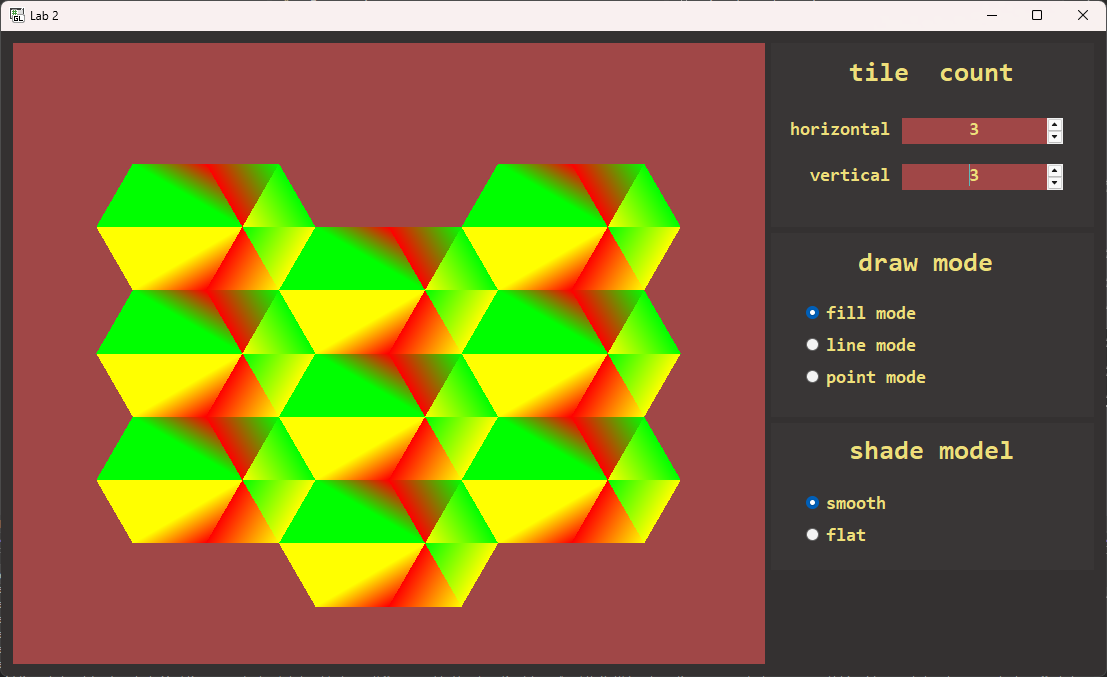


Рисунок 2.5 – Вигляд застосунку під час замощення

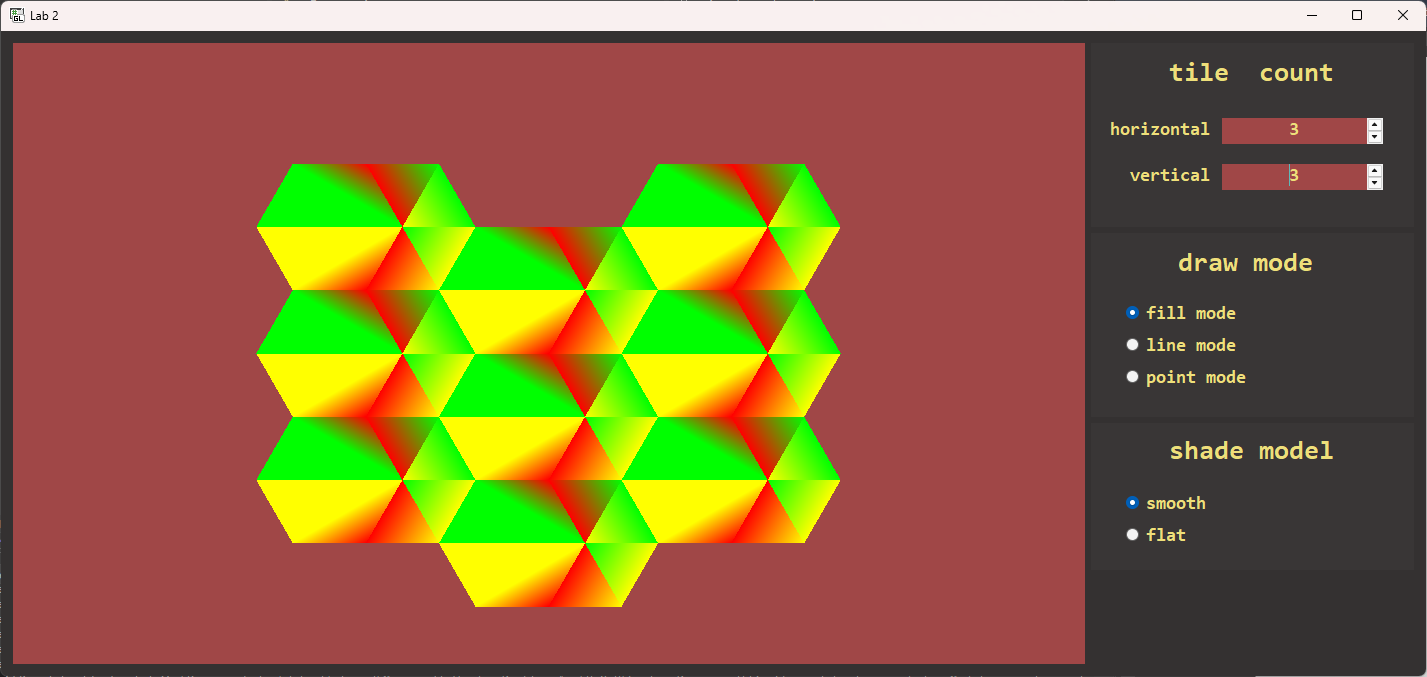


Рисунок 2.6 – Тестування програми при зміні ширини вікна

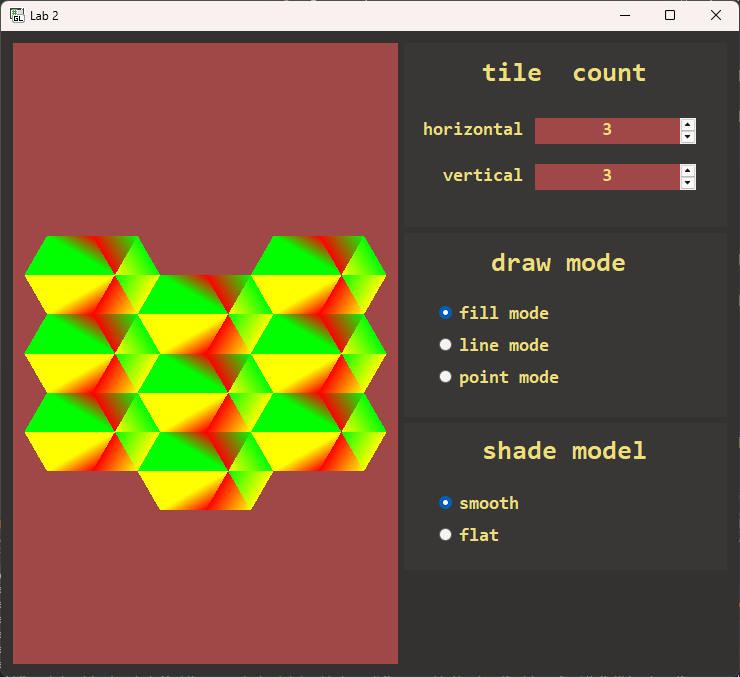


Рисунок 2.7 – Тестування програми при зміні ширини вікна

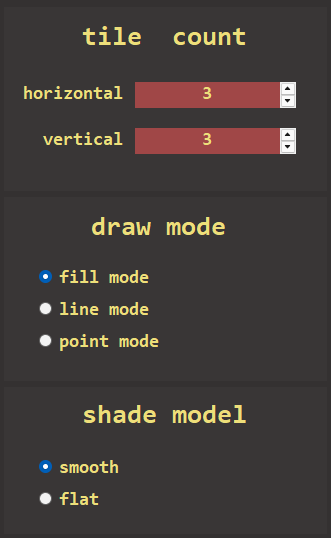


Рисунок 2.8 – Керування параметрами застосунку

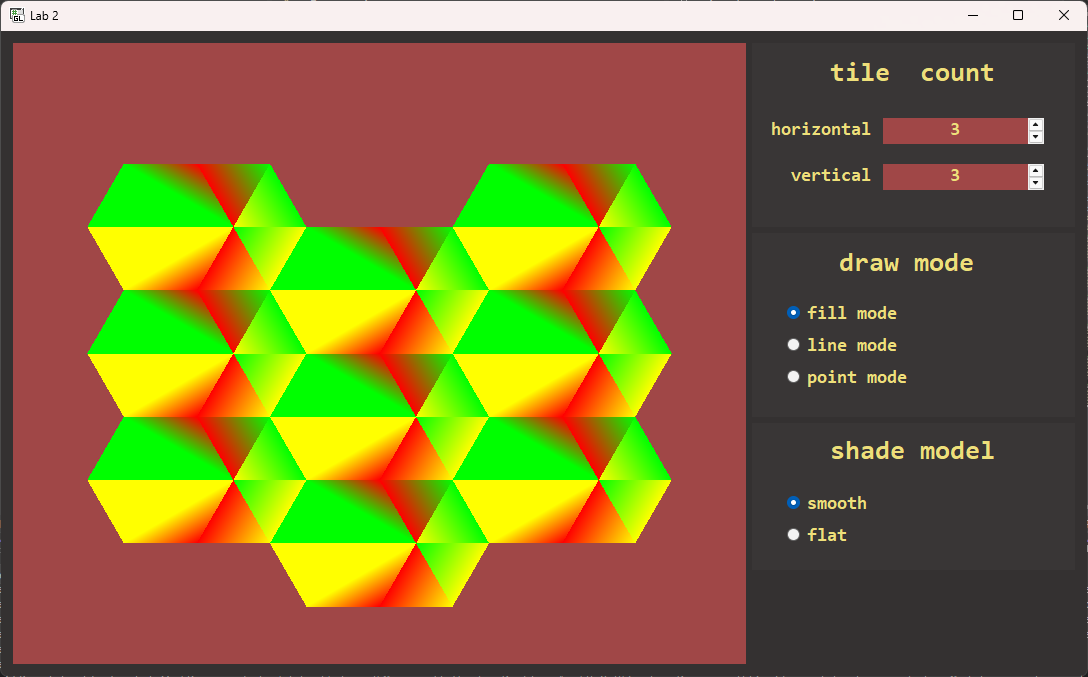


Рисунок 2.9 – Відображення заповнених фігур

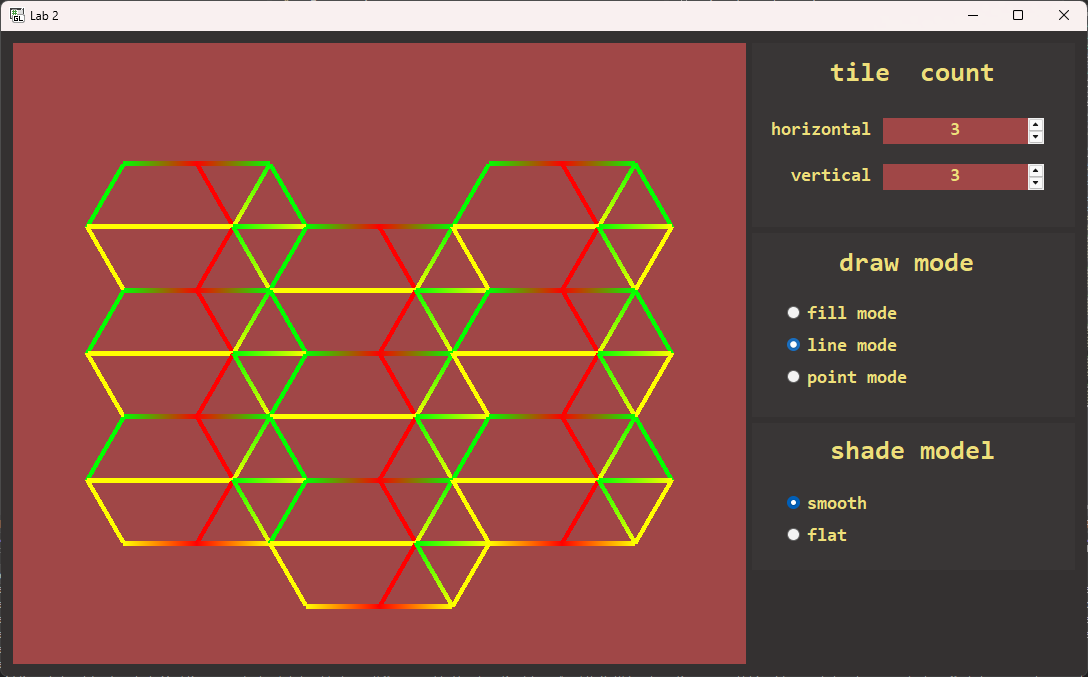


Рисунок 3.0 – Відображення лініями фігур

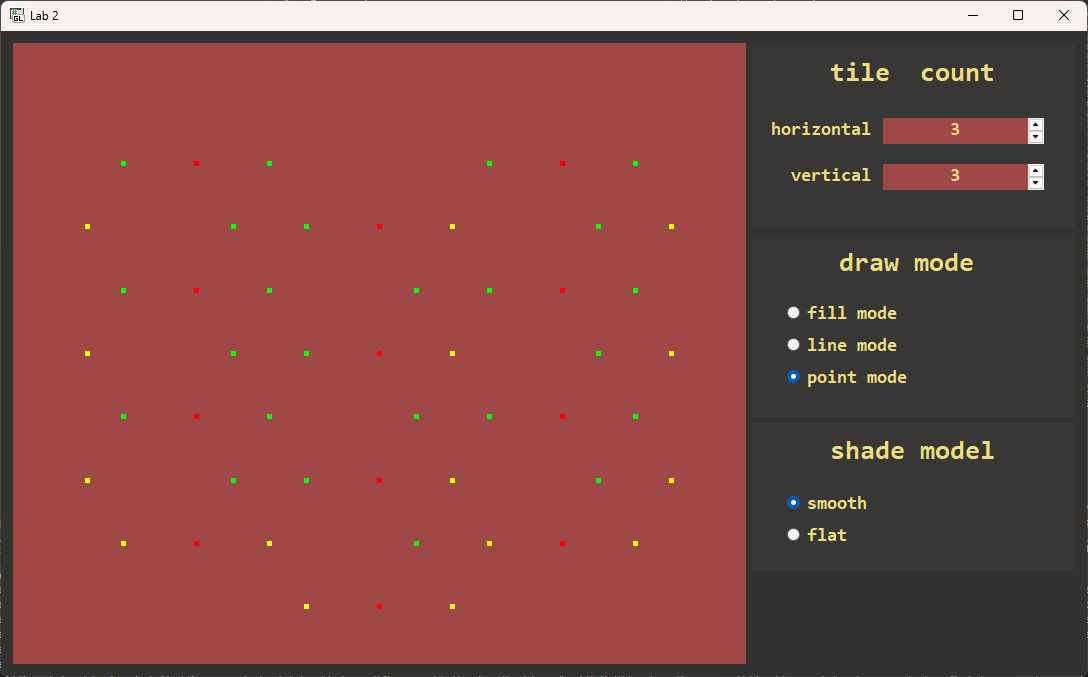


Рисунок 3.1 – Відображення точками фігур

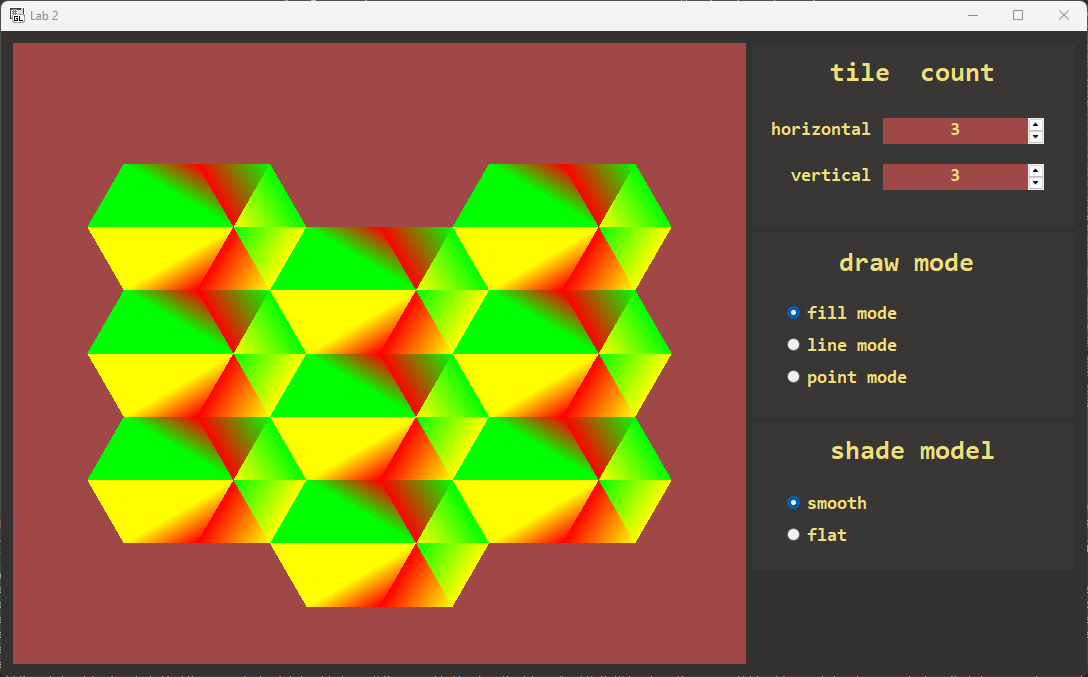


Рисунок 3.2 – Shade model **smooth**

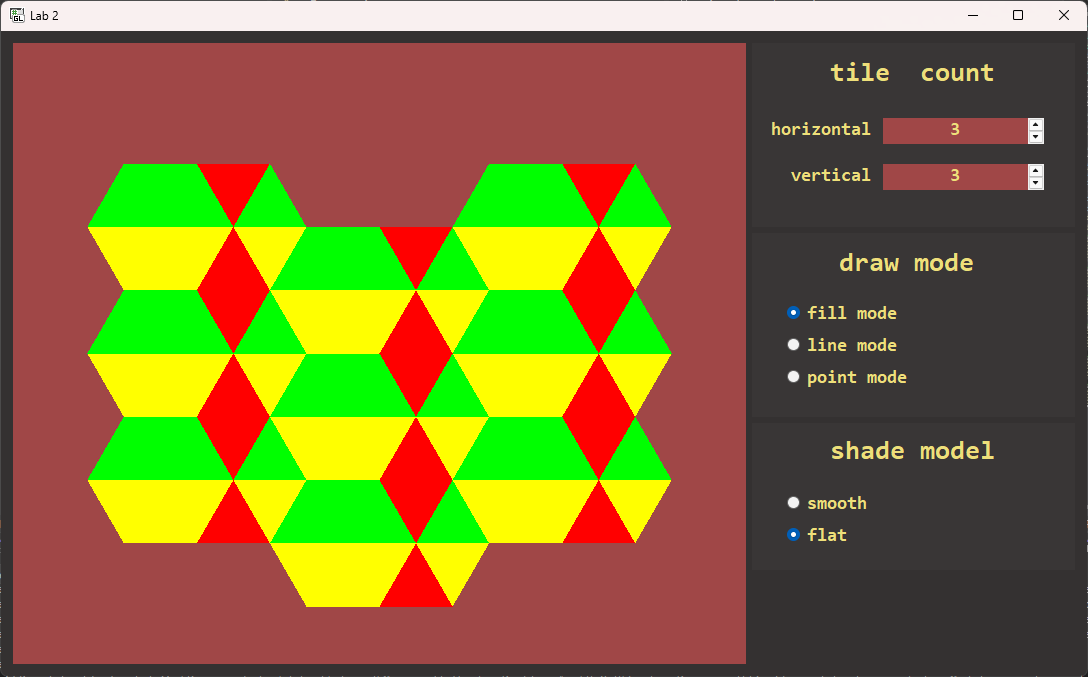


Рисунок 3.3 – Shade model **flat**

### Контроль виконання вимог та елементів завдання

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

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

# Практична робота 3. Графік функції однієї змінної

## Завдання, варіант № 24

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

* дозволяти користувачу задавати інтервал від **Xmin** до **Xmax** з перевіркою **Xmin < Xmax**;
* виконувати для завданого користувачем інтервалу від **Xmin** до **Xmax** автоматичне масштабування за віссю **Y** (додатково допускається наявність ручного режиму встановлення **Ymin** і **Ymax**);
* відображати осі координат (та/або координатну сітку) з виводом значень меж видимої області **Xmin**, **Xmax**, **Ymin** і **Ymax**, при цьому система координат повинна бути анізотропною;
* відображати усі точки, де **f(x) = 0**, якщо вони є на завданому інтервалі від **Xmin** до **Xmax**.

Приклад інтерфейсу користувача наведено на рис. 3.1. Варіанти функції **f1(x)** базового рівня складності наведені в табл. 3.2

Для підвищеного рівня складності необхідно додатково реалізувати коректне виведення функції **f2(x)** з урахуванням області визначення функції (варіанти наведені в табл. 3.3) і відобразити лінії розриву (рис. 3.2).

|  |  |
| --- | --- |
| 24 |  |
|  |

## Системна інформація

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

Processor AMD Ryzen 5 2600 3.40 GHz

RAM 16.0 GB (15.9 GB usable)

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

Edition Windows 11 Version 23H2

IDE Microsoft Visual Studio Enterprise 2022 (64-bit) version 17.11.5

## Теоретичні відомості

Після того, як користувач у режимі діалогу визначив інтервал по осі X, слід задати кількість точок N, необхідно для побудови графіка функції. Ця кількість може бути задана користувачем явно чи отримана програмно, наприклад, відповідати ширині (кількість пікселів) робочої області. На основі цієї інформації вираховується крок аргументу функції:

і розраховуються координати точок функції в межах завданого користувачем інтервалу:

xi = Xmin + i h; yi = f(xi) ; i ∈ 0. . N − 1.

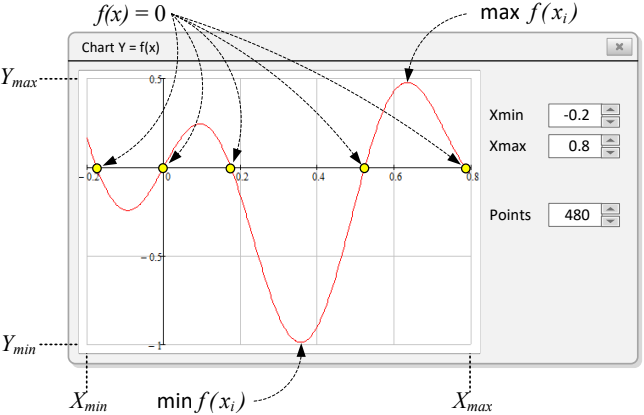


Рис. 3.1. Приклад екранної форми з урахуванням базових вимог до роботи

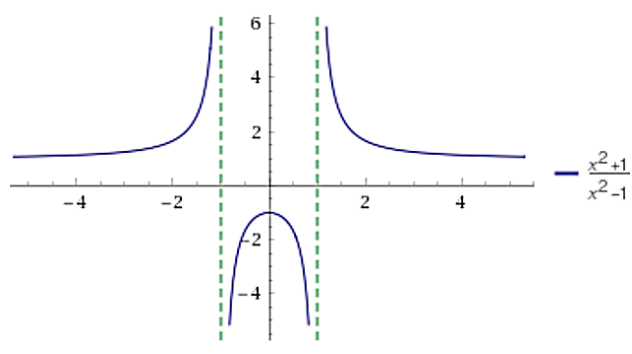


Рис. 3.2. Приклад відображення функції і її ліній розриву (відображені пунктиром)

Використовуючи ці розрахунки, можна знайти межі робочої області по вертикалі:

Ymin = min(yi);

Ymax = max(yi);

i ∈ 0. . N − 1.

Таким чином, знаючи значення меж інтервалу по осі X і обчисливши значення меж по осі Y, можна встановити систему координат (за допомогою команди glOrtho(...)) для виведення графіка y = f(x) на екран.

Знайти корні функції x0 (точки, де f(x0) = 0) можна на основі наступної властивості: якщо на інтервалі від xi до xi+1 є перетин з віссю абсцис, то в результаті примноження відповідних ординат буде виконуватися умова f(xi)f(xi+1) ≤ 0 (рис. 3.3). В цьому випадку координати точок перетину функції з віссю X вираховують спрощеним методом половинного ділення: x0 = (xi + xi+1)⁄2, y0 = f(x0). Якщо брати до уваги дискретність екрану і використовувати для побудови графіка функції кількість точок, що близьке до значення ширини робочої області (в пікселях) чи більше за неї, то спрощений метод половинного ділення дозволить отримати рішення, що візуально не відрізняється від точного.

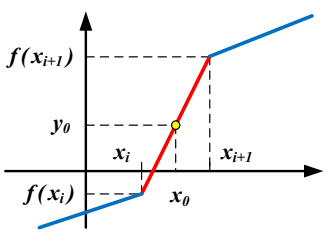


Рис. 3.3. Спрощений варіант методу половинного ділення

Алгоритм коректного відображення на екрані функції f2(x), що має розриви в області визначення, студенти розробляють самостійно. Додатковий аналіз особливостей свого варіанта функції f2(x) можна отримати за допомогою наступних електронних ресурсів [1,2].

## Результати виконання практичної роботи

### Розв'язання завдання

Осі координат і графік функції f1(x) виводяться на заданому користувачем інтервалі від Xmin до Xmax і від Ymin до Ymax. Рисунок **3.4**, рядки **19 - 61** файл **MainForm.cs**, **додаток С**.

Автоматичні обчислення Ymin і Ymax на завданому інтервалі від Xmin до Xmax функції f1(x). Рисунок **3.5**, рядки **76 - 83**, **91 - 101** файл **FuncOne.cs**, **додаток С**.

Обчислення і виведення на екран точок f1(x) = 0. Рисунок **3.6**, рядки **70 - 73**, **104 - 122** файл **FuncOne.cs**, **додаток С**.

Коректне виведення графіка f2(x) (без хибного виводу точок розриву як точок перетину з віссю абсцис) і з відображенням ліній розриву функції. Рисунок **3.7**, рядки **28 - 112**, **114 - 136** файл **FuncTwo.cs**, **додаток С**.

Використання ООП (наслідування). Рядок 1 файл **FuncOne.cs**, Рядок 1 файл **FuncTwo.cs**, **додаток С**.

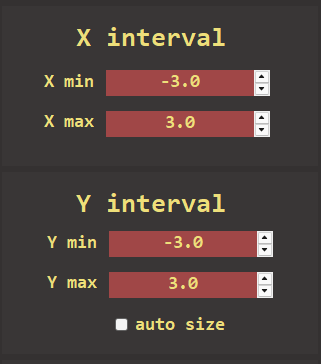


Рисунок 3.4 – Введення інтервалів для X та Y через інтерфейс

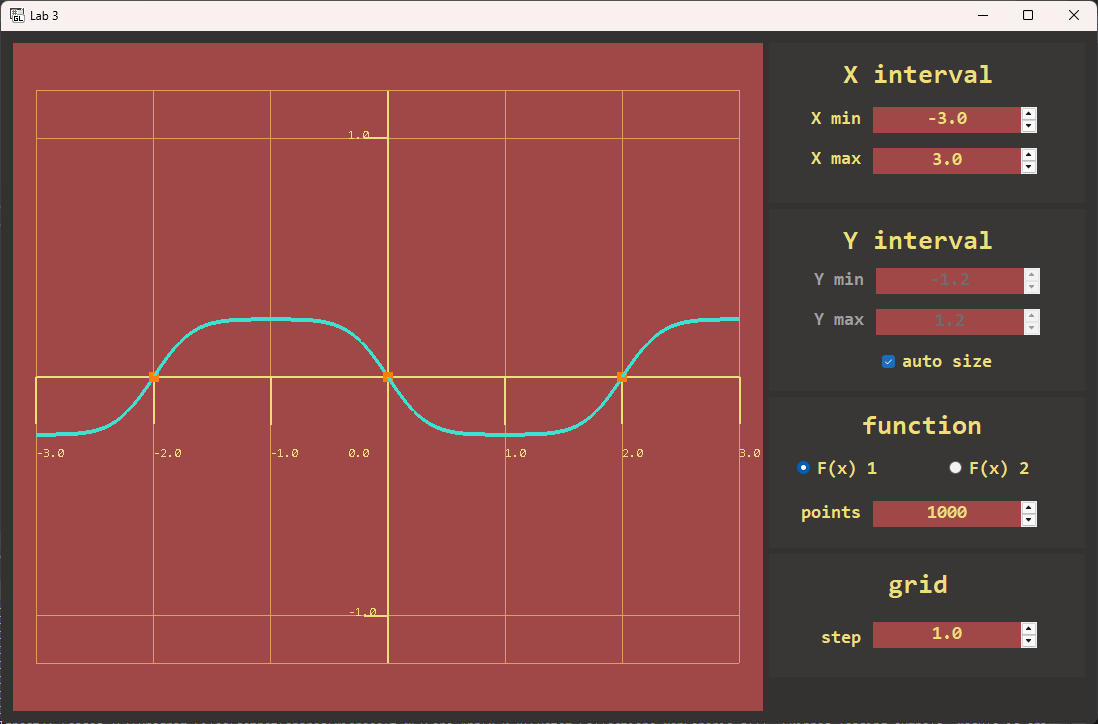


Рисунок 3.5 – Автоматичний розрахунок інтервалів Y

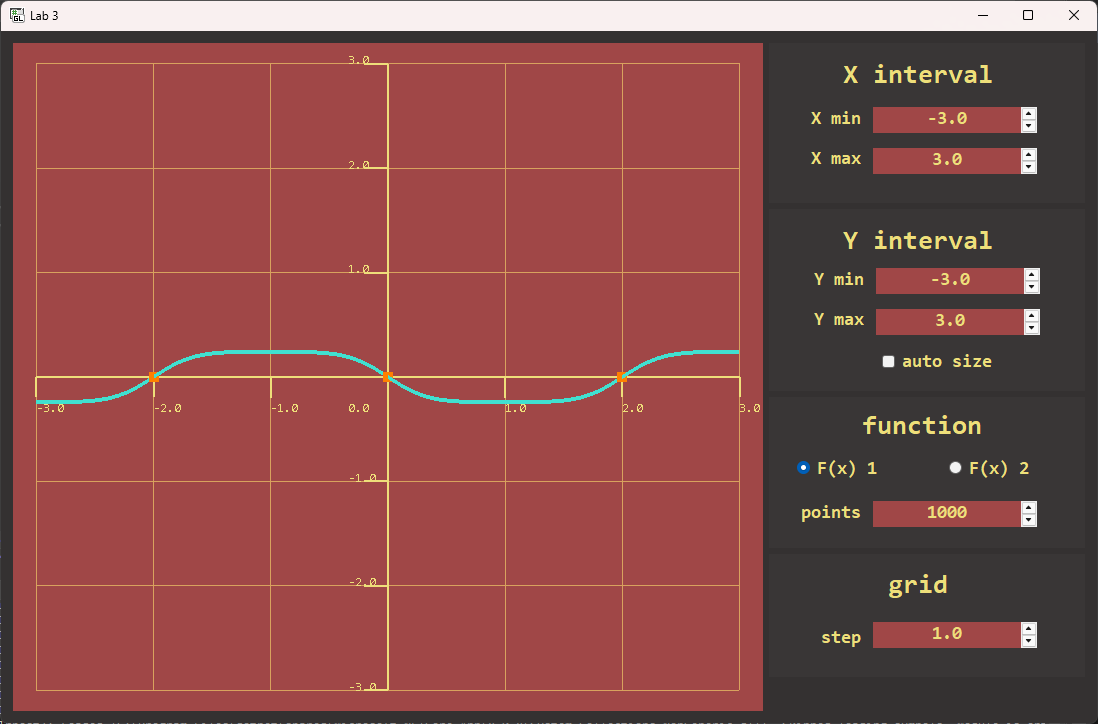


Рисунок 3.6 – Відображення точок (помаранчеві), коли у = 0

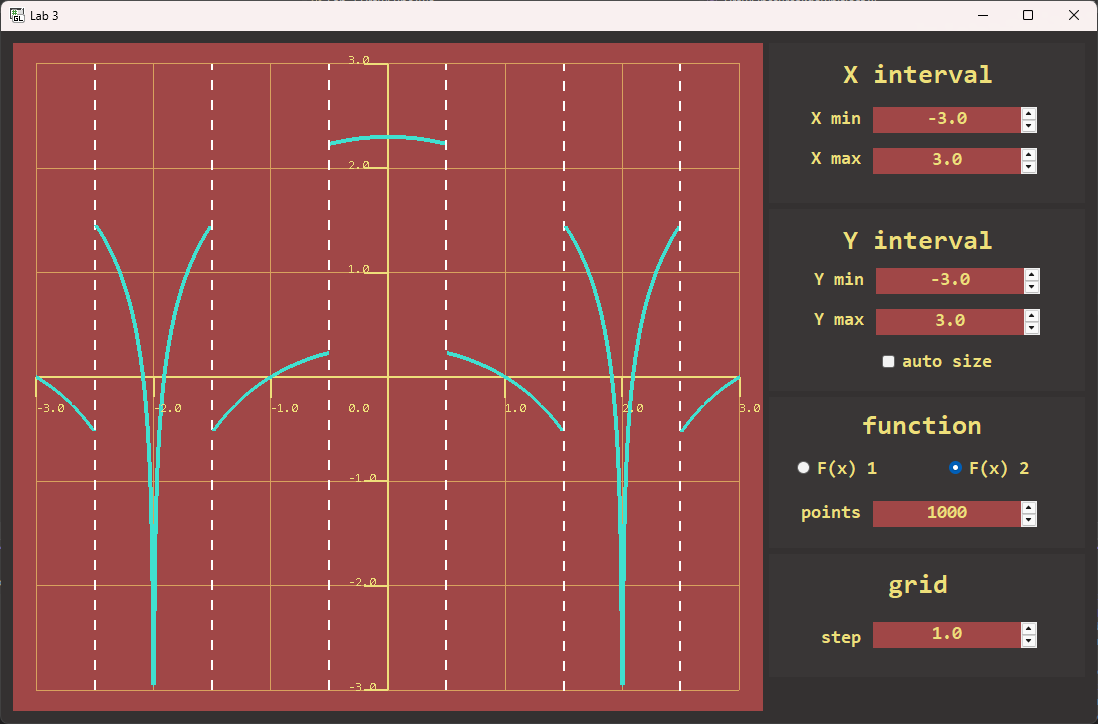


Рисунок 3.7 – Відображення другої функції з лініями разриву (білим пунктиром)

### Контроль виконання вимог та елементів завдання

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

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

# Практична робота 4. Криві другого порядку

## Завдання, варіант № 24

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Окружність | | Гіпербола | |
| Попадання | | Попадання | |
| явне | параметричне | явне | параметричне |
| 24 | **+** |  | **++** |  |

## Системна інформація

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

Processor AMD Ryzen 5 2600 3.40 GHz

RAM 16.0 GB (15.9 GB usable)

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

Edition Windows 11 Version 23H2

IDE Microsoft Visual Studio Enterprise 2022 (64-bit) version 17.11.5

## 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 [x0, y0] двох відрізків[x1, y1], [x2, y2] и [x3, y3], [x4, y4], 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 t1першого і t2 другого відрізків:

І за умови, що результат рішення буде задовольняти наступні дві умови: 0 ≤ t1 ≤ 1 і 0 ≤ t2 ≤ 1, інакше відрізки або паралельні, або перетинаються тільки прямі, на яких вони лежать [1, 2].

## Результати виконання практичної роботи

### Розв'язання завдання

Установлення ізотропної системи координат для вікна з змінюваними розмірами. Рисунок **4.1 - 4.2**, рядки **7 - 13**, **59** файл **RenderControl.cs**, **додаток D**.

Виведення кривих другого порядку відповідно до варіанту завдання. Файл **RenderControl.cs**, **додаток D**.

* Окружість, рисунок **4.3**, рядки **144 - 179**
* Гіпербола, рисунок **4.4**, рядки **246 - 288**

Виведення відрізка та обчислення його точок перетину з кривою другого порядку відповідно до варіанту. Рисунок **4.5**, файл **RenderControl.cs**, **додаток D**.

* Лінія, рядки **181 - 192**
* Точка, рядки **194 - 244**
* Малювання лінії в гіперболі, рядки **179 - 180**

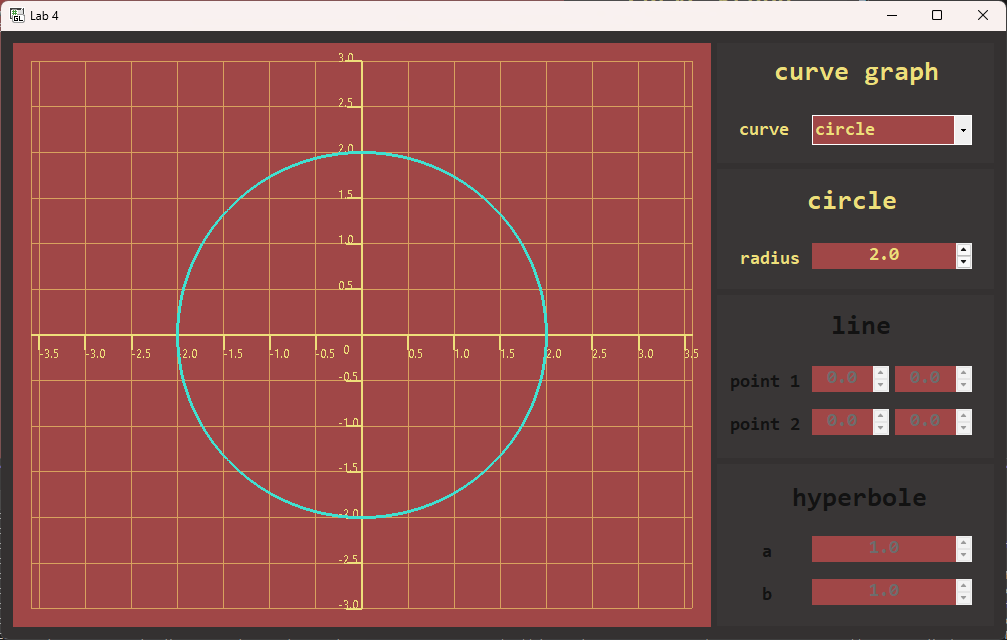


Рисунок 4.1 – Тестування програми при зміні ширини вікна

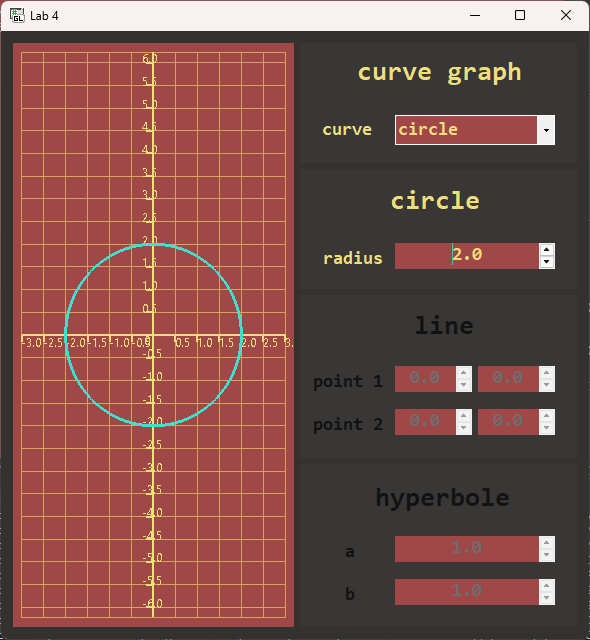


Рисунок 4.2 – Тестування програми при зміні ширини вікна

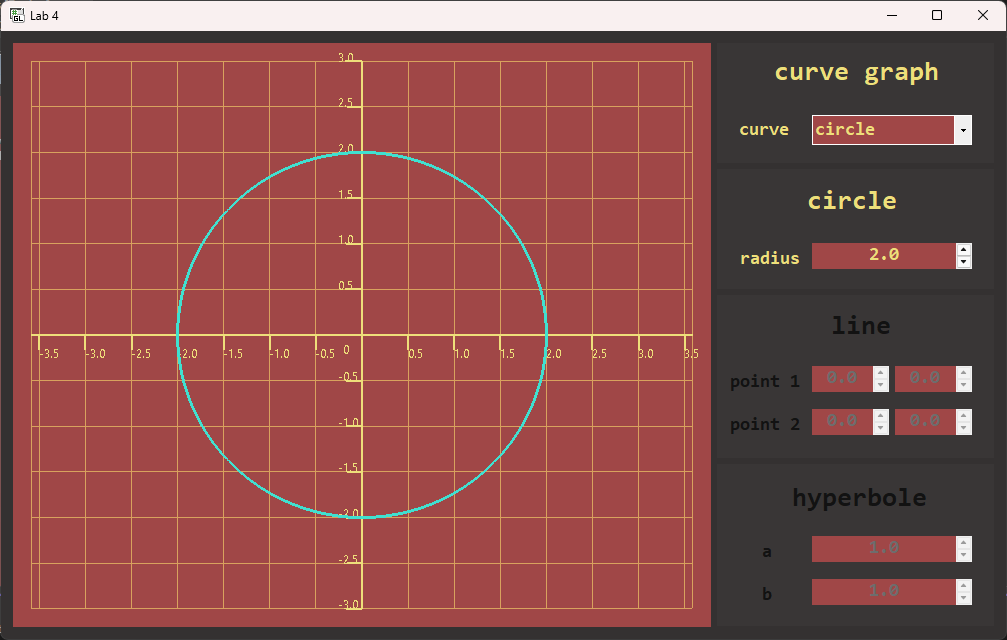


Рисунок 4.3 – Крива другого порядку, окружність

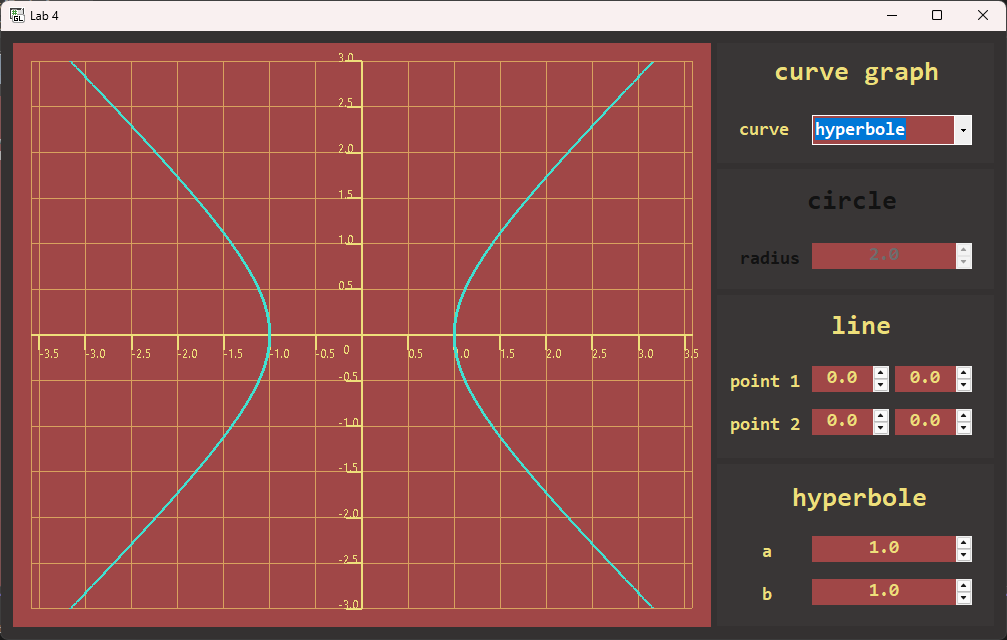


Рисунок 4.3 – Крива другого порядку, гіпербола

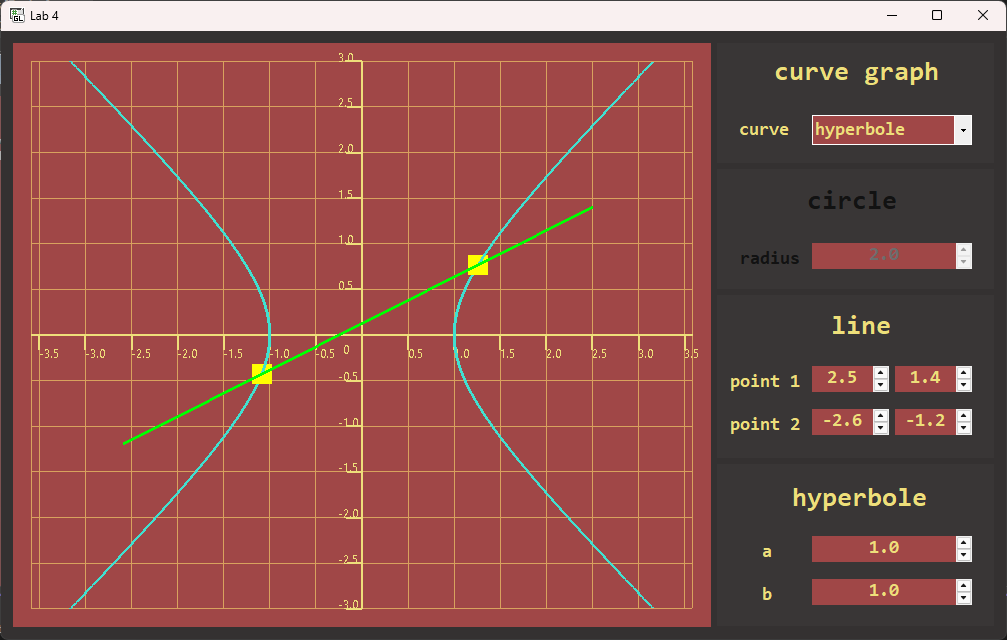


Рисунок 4.4 – Відображення відрізка та точок перетину з окружністю

### 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 4.1.

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

# Практична робота 5. Квадратичні примітиви. Афінні перетворення у просторі

## Завдання, варіант № 24

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

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

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

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **№** | **grid** | **фігура** | **Параметри квадратичних фігур** | | | | | | | | |
| **ось** | **X0** | **Y0** | **Z0** | **R** | **r** | **h** | **∠start** | **∠sweep** |
| 24 | YOZ | сфера | || OZ | +2.0 | +2.0 | +3.5 | 2.0 | - | - | - | - |
| Усічений конус | ⇅OY | -3.5 | +2.5 | -2.0 | 2.0 | 0.5 | 2.0 | - | - |
| диск | || OX | +4.5 | +2.5 | +2.5 | 4.0 | 1.0 | - | 0° | 45° |

## Системна інформація

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

Processor AMD Ryzen 5 2600 3.40 GHz

RAM 16.0 GB (15.9 GB usable)

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

Edition Windows 11 Version 23H2

IDE Microsoft Visual Studio Enterprise 2022 (64-bit) version 17.11.5

## 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.

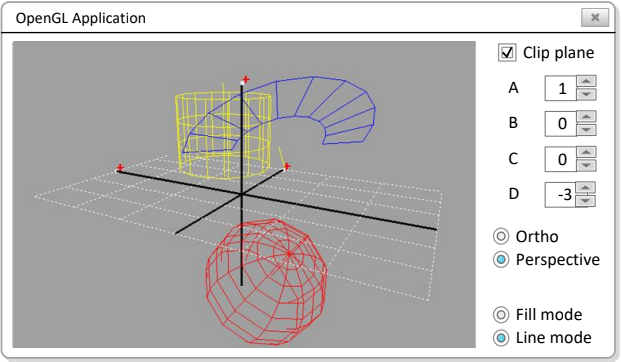


Рис. 5.1. Приклад сцени з квадратичними об’єктами і плоскістю відтину для сфери

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. Під час формування зображення квадратичних примітивів використовують такі параметри:

* **x0, y0, z0** – координати точки прив’язки фігури;
* **R** – радіус сфери або радіус основи циліндра/конуса з центром у точці прив’язки або зовнішній радіус диска;
* **r** – радіус другої основи циліндра / усіченого конуса; внутрішній радіус диска;
* **h** – висота циліндра / конуса / усіченого конуса;
* **∠start** – початковий кут частинного диска;
* **∠sweep** – кут розгортки для частинного диска;
* ось **||**,**⇈**,**⇅** – паралельність / колінеарність осей координат і фігури.

## Результати виконання практичної роботи

### Розв'язання завдання

Коректне (ізотропне) відображення завдання (під час зміни розмірів вікна) у ортографічній проекції. Рядки **9 - 17**, **298**, файл **RenderControl.cs**, **додаток E**.

Під час запуску застосунку відображаються осі 0X, 0Y, 0Z, координатна сітка і каркас квадратичних об’єктів.

* Сітка. Рядки **160 - 279**, файл **RenderControl.cs**, **додаток E**.
* Квадратичні об'єкти. Рядки **306 - 353**, файл **RenderControl.cs**, **додаток E**.

Інтерфейс керування параметрами площини відтину. Файл **RenderControl.cs**, **додаток E**.

* Перспективний та ортографічний вигляд. Рядки **106 - 110**, **290 - 104.**
* Керувння поворотом мишкою. Рядки **104 - 106**, **134 - 158**.

Використання джерел світла для освітлення об’єктів сцени сумісно з командою glColorMaterial. Рядки **116 - 126**, файл **RenderControl.cs**, **додаток E**.

Використання списків відображення (Display Lists). Рядки **77 - 86**, **114**, файл **RenderControl.cs**, **додаток E**.

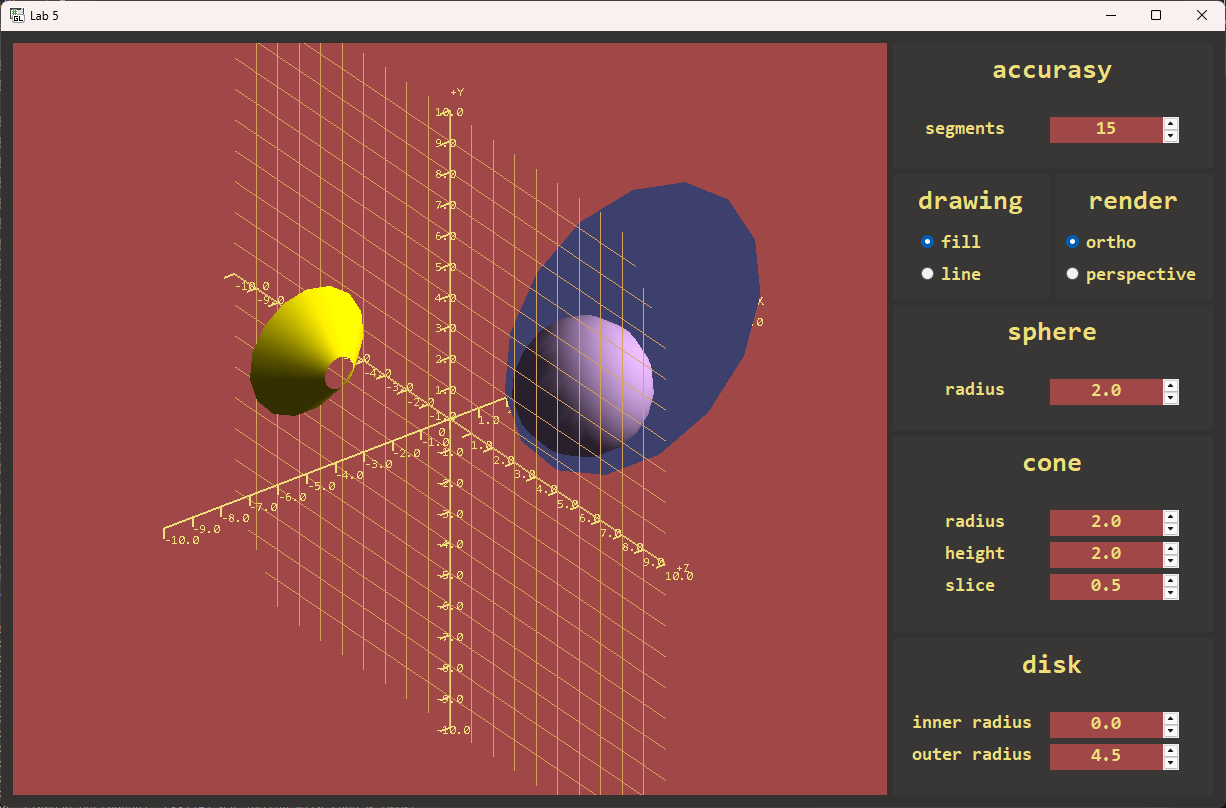


Рисунок 5.1 – Тестування програми при зміні ширини вікна

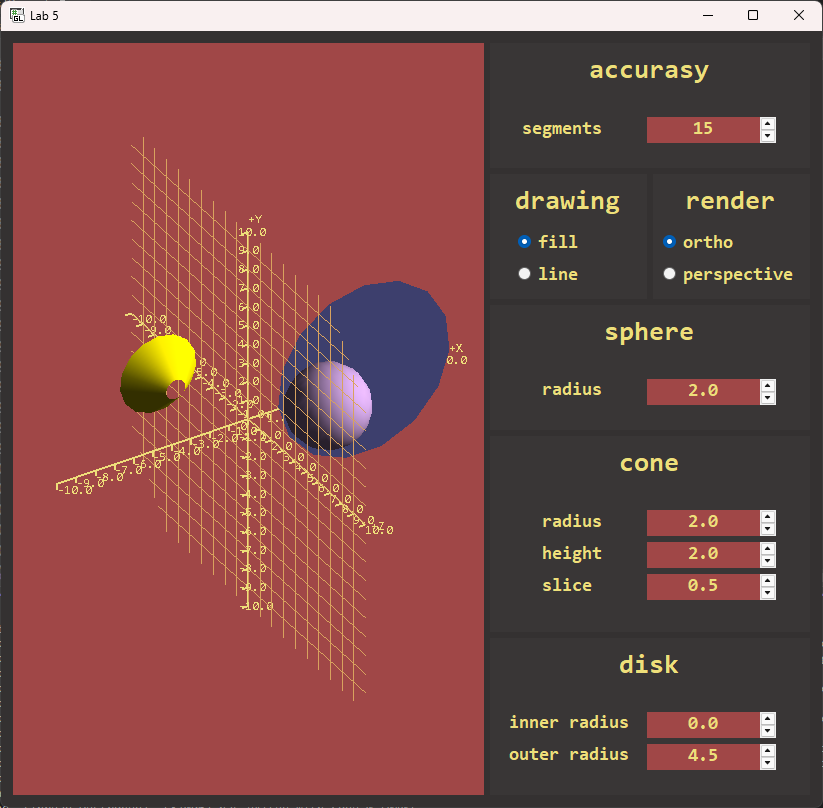


Рисунок 5.2 – Тестування програми при зміні ширини вікна

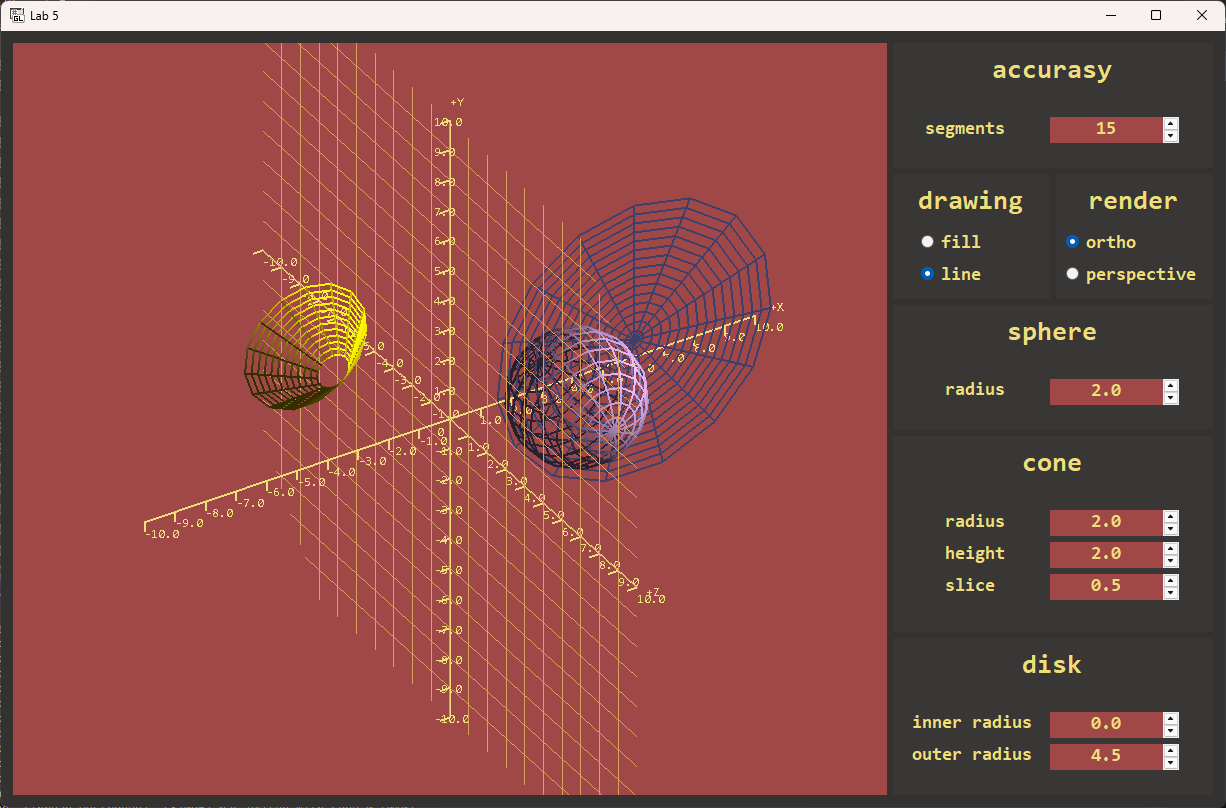


Рисунок 5.3 – Відображення координатної сітки Y0Z та каркасу квадратичних об’єктів

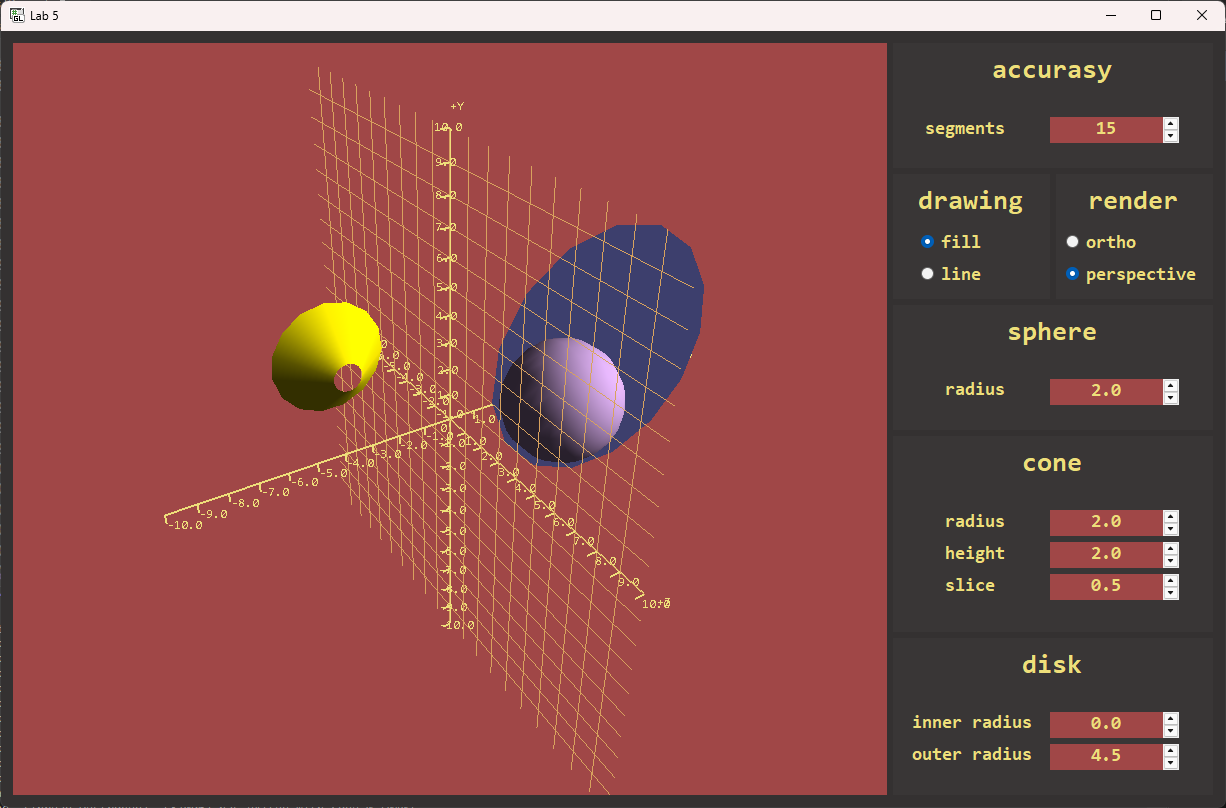


Рисунок 5.4 – Вигляд у перспективі

### 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 | Коректне (ізотропне) відображення завдання (під час зміни розмірів вікна) у ортографічній проекції | 1 | **+** |
| 2 | Під час запуску застосунку відображаються осі 0X, 0Y, 0Z, координатна сітка і каркас квадратичних об’єктів | 1 | **+** |
| 3 | Інтерфейс керування параметрами площини відтину | 1 | **+** |
| 4 | Використання джерел світла для освітлення об’єктів сцени сумісно з командою glColorMaterial | 1 | **+** |
| 5 | Використання списків відображення (Display Lists) | 1 | **+** |
| 6 | Підвищений рівень | Створення зображення сцени в перспективній проекції | 1 | **+** |
| 7 | Накладення текстури на поверхню завданих у варіанті фігур | 1 | **-** |
| 8 | Застосування команди glMaterial для налаштування параметрів відбиття поверхонь об’єктів сцени | 1 | **-** |

# Практична робота 6. Візуалізація прямої задачі кінематики

## Завдання, варіант № 24

Розробити застосунок для виведення на екран моделі маніпулятора, заданої кінематичною схемою (рис. 1.1) і відносними розмірами (a = 0.3, b = 0.5). Для керування маніпулятором та/або зміни точки спостереження необхідно використовувати обробники подій клавіатури та/або маніпулятора «миша», у яких змінюють величину відповідних параметрів:

– кут γ для обертання навколо осі OZ;

– відстань S для зміни кутів θ, ψ;

– кут ay для обертання навколо осі OY.

|  |  |  |
| --- | --- | --- |
| 24 | a = 0.3  b = 0.5 |  |

## Системна інформація

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

Processor AMD Ryzen 5 2600 3.40 GHz

RAM 16.0 GB (15.9 GB usable)

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

Edition Windows 11 Version 23H2

IDE Microsoft Visual Studio Enterprise 2022 (64-bit) version 17.11.5

## 2 Математична модель кінематичної схеми

Розв'язання цієї задачі може бути виконане поетапно. На першому етапі задача розглядається як двомірна тільки в одній площині **Z = const = 0**, а вже на другому етапі розв'язання задачі розглядається у просторі.

## 2.1 Афінні перетворення

Для реалізації поставленої задачі на площині достатньо використовувати комбінацію з двох афінних перетворень: обертання та перенесення. Для заданого варіанту необхідно використовувати обертання навколо осі Z:



та перенесення вздовж осі Y:



Для реалізації завдання у просторі додатково знадобиться обертання навколо осі Y:



обертання навколо осі X:



та масштабування по всіх трьох осях:



## **2.2 Декомпозиція на сегменти**

Фактично всі кінематична схема може бути представлена у вигляді трьох сегментів, які для спрощення можна подати у вигляді відрізків Segment заданого розміру size, кожен з яких у початковому стані займає вертикальне положення (вздовж осі OY):



Таким чином, для вирішення завдання потрібно визначити набір афінних перетворень, які необхідні для перетворення з початкового стану на положення, що задане кінематичною схемою.

## 2.2.1 Математична модель сегмента «2а»

Розташування сегмента a всіх сегментів і для відображення у складі кінематичної схеми позначимо його як S2a, що залежить від повороту відносно осі Z на кут phi:

|  |  |
| --- | --- |
| S2a(phi)= Rz(phi)×Segment(2a) | (2.7) |

## 2.2.1 Математична модель сегмента «2а1»

Положення сегмента 2а1 залежить від двох кутів: theta та phi. При цьому кут theta, у свою чергу, залежить від величини S. Тому для обчислення кута theta розглянемо трикутник, який утворено сторонами a, S і b, в якому один з внутрішніх кутів є суміжним з кутом theta. Тоді можна скористатися теоремою косинусів, задаючи кут суміжний з theta зв'язавши його наступним виразом зі сторонами a, S і b:

|  |  |
| --- | --- |
| b2 = a2 + S2 - 2 \* a \* S \* cos(theta) | (2.8) |

з якого можна вивести залежність кута phi від значення S:

|  |  |
| --- | --- |
|  | (2.9) |

Таким чином, для представлення сегмента у складі кінематичної схеми введемо для сегмента позначення S2a1, положення якого залежить від повороту відносно осі Z на кут theta і величини S:

|  |  |
| --- | --- |
| S2a1(phi, S)= Rz(phi)×Ty(a)×Rz((S))×Segment(2a1) | (2.10) |

## 2.2.2 Математична модель сегмента «b»

Положення сегмента c залежить від трьох кутів: phi та psi. При цьому кут psi, у свою чергу, залежить від величини S. Тому, для обчислення кута розглянемо трикутник, який утворено сторонами a, S і b і містить кут psi. Використовуючи теорему косинусів, наступним виразом можна зв'язати кут psi зі сторонами a, S і b:

|  |  |
| --- | --- |
| S2 = a2 + b2 - 2 \* a \* b \* cos(psi) | (2.11) |

з якого можна вивести залежність кута ag від значення S:

|  |  |
| --- | --- |
|  | (2.12) |

Таким чином, для представлення сегмента у складі кінематичної схеми введемо для сегмента позначення Sb, положення якого залежить від повороту відносно осі Z на кут theta, phi і величини S:

|  |  |
| --- | --- |
| Sb(phi, S)= Rz(phi)×Ty(2a)×Rz(psi(S))×Segment(-b)×Segment(1.5b) | (2.13) |

## 2.3 Фізичні обмеження моделі

Для розрахунку положення двох сегментів використовується теорема косинусів, яка зв'язує сторони трикутника a1, S і c, а так як зміна значення S використовується для управління кінематичної схеми, то необхідно враховувати обмеження на існування трикутника, що будь-яка сторона трикутника менша за суму двох інших сторін. Це обмеження може бути виражене двома предикатами:

|  |  |
| --- | --- |
| CheckMax(S) = S < b - a  CheckMin(S) = S > a | (2.14) |

які обидва набувають значення true у тому випадку, коли значення S знаходиться в допустимих межах.

## 2.4 Загальна тривимірна математична модель

До цих пір розв'язання задачі розглядалося в межах однієї площини. Для отримання загальної тривимірної моделі до повороту відносно осі Z додамо масштабування та повороти щодо осей X та Y. Всі перетворення, які виконуються для всіх сегментів, позначимо як Q:



тоді повна тривимірна модель набуде наступного вигляду:

|  |  |
| --- | --- |
| S2a(phi)= Rz(phi)×Segment(2a)  S2a1(phi, S)= Rz(phi)×Ty(a)×Rz((S))×Segment(2a1)  Sb(phi, S)= Rz(phi)×Ty(2a)×Rz(psi(S))×Segment(-b)×Segment(1.5b) | (2.16) |

а перетворення, які визначені в Q для ефективності можна зберегти, наприклад, у стеку матриць з подальшим відновленням у потрібний момент.

## **3 Програмна реалізація кінематичної схеми**

## 3.1 Початкові дані

Вихідні дані кінематичної схеми задані основними значеннями a, b які можуть бути представлені звичайними змінними:

public partial class RenderControl : OpenGL

{

    private float a = 0.3f;

    private float b = 0.5f;}

...

}

Аналогічним чином можуть бути реалізовані обмеження (2.14) для контролю за допустимими значеннями параметру S:

private float s = 0.2f;

private float S

{

get { return s; }

set { if ((value > (b - a)) && (value < (2 \* a))) s = value; }

}

крім цього, кути phi і psi у свою чергу теж є функціональними залежностями від значення S, що відображено у (2.9) и (2.12) і, відповідно, повинні мати заборону на операцію запису (і водночас переводимо радіани в градуси):

private float theta

{

get { return 180f / MathF.PI \* MathF.Acos((a \* a + S \* S - b \* b) / (2 \* a \* S)); }

}

private float psi

{

get { return 180f / MathF.PI \* MathF.Acos((a\*a + b\*b - S\*S) / (2 \* a \* b)); }

}

## 3.2 Прив'язка системи координат до видимої області вікна

Для коректного відображення пропорцій елементів кінематичної схеми необхідно використовувати ізотропну систему координат, що може бути реалізовано, наприклад, в обробнику події, пов'язаної зі зміною розмірів вікна таким чином:

private float Xmin { get => (AspectRatio > 1) ? -size \* AspectRatio : -size; }

private float Xmax { get => (AspectRatio > 1) ? +size \* AspectRatio : +size; }

private float Ymin { get => (AspectRatio < 1) ? -size / AspectRatio : -size; }

private float Ymax { get => (AspectRatio < 1) ? +size / AspectRatio : +size; }

private float Zmin { get => -size; }

private float Zmax { get => +size; }

. . .

glOrtho(Xmin, Xmax, Ymin, Ymax, Zmin, Zmax);

## 3.3 Керування об'єктом моделювання та організація інтерфейсу

Для організації інтерфейсу з користувачем використовуємо клавіатуру та маніпулятор «миша». Керування системою координат та масштабом зв'яжемо з подіями маніпулятора «миша». Зафіксуємо натискання на ліву кнопку в логічній змінній mouseFlag та збережемо координати, в яких знаходився маніпулятор:

private void OnMouseDown(object sender, System.Windows.Forms.MouseEventArgs e)

{

mouseFlag = e.Button == MouseButtons.Left;

mouseStart = e.Location;

}

потім, якщо ліва кнопка не відпущена і маніпулятор виконує рух, зв'яжемо переміщення по горизонталі з кутом повороту навколо осі Y, а переміщення по вертикалі з кутом повороту відносно осі X:

private void OnMouseMove(object sender, System.Windows.Forms.MouseEventArgs e)

{

if (mouseFlag)

{

Point current = e.Location;

angleX += (current.Y - mouseStart.Y) / 2.0f;

angleY += (current.X - mouseStart.X) / 2.0f;

mouseStart = current;

Invalidate();

}

}

після чого знову зберігаємо поточні координати маніпулятора та повідомляємо вікно про необхідність виконати перемалювання робочої області за допомогою методу Invalidate(). Виконуємо відстеження переміщення маніпулятора доти, доки ліва кнопка не буде відпущена:

private void OnMouseUp(object sender, System.Windows.Forms.MouseEventArgs e)

{

if (mouseFlag)

mouseFlag = !(e.Button == MouseButtons.Left);

}

Для управління масштабуванням скористаємося колесом прокручування, використовуючи значення e.Delta як збільшення масштабу:

private void OnMouseWheel(object sender, MouseEventArgs e)

{

m += e.Delta / 2000.0f;

Invalidate();

}

Так само відстежуємо необхідні події (натискання на клавіатурі), зв'язавши зміну кута обертання theta на 1 градус з клавішами «S» і «W», а значення S зміняться на величину 0.01f клавішами «E» «Q»:

private void OnKeyDown(object sender, KeyEventArgs e)

{

switch (e.KeyCode)

{

case Keys.W:

phi--;

break;

case Keys.S:

phi++;

break;

case Keys.E:

S += 0.01f;

break;

case Keys.Q:

S -= 0.01f;

break;

case Keys.A:

aw++;

break;

case Keys.D:

aw--;

break;

}

Invalidate();

}

## 3.4 Осі координат

Для підвищення загальної наочності та спрощення орієнтації елементів кінематичної схеми можна використовувати зображення осей координат наступним чином:

private void DrawAxis()

{

glColor3d(1f, 1f, 1f);

glLineWidth(1);

glBegin(GL\_LINES);

glVertex3d(-padding, 0f, 0f);

glVertex3d(1f, 0f, 0f);

glVertex3d(0f, -padding, 0f);

glVertex3d(0f, 1f, 0f);

glVertex3d(0f, 0f, -padding);

glVertex3d(0f, 0f, 1f);

glEnd();

DrawText("X", 1f, 0f, 0f);

DrawText("Y", 0f, 1f, 0f);

DrawText("Z", 0f, 0f, 1f);

}

## **3.5 Виведення сегмента**

Для виведення сегмента в початковому стані відповідно до (2.6) використовуємо наступний програмний код, додавши значення r, g, b для управління кольором фігури, що виводиться:

private void Segment(float height, float width, float r, float g, float b)

{

glColor3d(r, g, b);

glLineWidth(5);

glBegin(GL\_LINES);

glVertex3d(0f, 0f, -width);

glVertex3d(0f, height, -width);

glVertex3d(0f, 0f, width);

glVertex3d(0f, height, width);

glVertex3d(0f, 0f, -width);

glVertex3d(0f, 0f, width);

glVertex3d(0f, height, -width);

glVertex3d(0f, height, width);

glEnd();

glLineWidth(1);

}

## 3.6 Порядок виконання афінних перетворень

В результаті формування зображення в остаточному вигляді відбувається в методі, який відповідає за перерисовку області зображення:

private void OnRender(object sender, EventArgs e)

{

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

glLoadIdentity();

glViewport(0, 0, Width, Height);

glOrtho(Xmin, Xmax, Ymin, Ymax, Zmin, Zmax);

glRotated(angleX, 1, 0, 0);

glRotated(angleY, 0, 1, 0);

glScaled(m, m, m);

// Enable depth

glEnable(GL\_DEPTH\_TEST);

DrawAxis();

DrawGridXY();

DrawGridYZ();

DrawGridZX();

glRotated(aw, 0f, -1f, 0f);

// first line

glRotated(phi, 0f, 0f, -1f);

Segment(2 \* a, 0.05f, 1f, 0f, 0f);

// second line

glPushMatrix();

glTranslated(0f, a, 0f);

glRotated(-theta, 0f, 0f, -1f);

Segment(2 \* a, 0.04f, 0f, 1f, 0f);

glPopMatrix();

// third line

glTranslated(0f, 2\*a, 0f);

glRotated(psi, 0f, 0f, -1f);

Segment(-b, 0.05f, 0f, 0f, 1f);

Segment(1.5f \* b, 0.05f, 0f, 0f, 1f);

}

## 4 Результати візуалізації кінематичної схеми

У результаті виконання розробленого додатка на екран виведено маніпулятор (рис. 6.2 – рис. 6.4), що відповідає заданій кінематичній схемі.

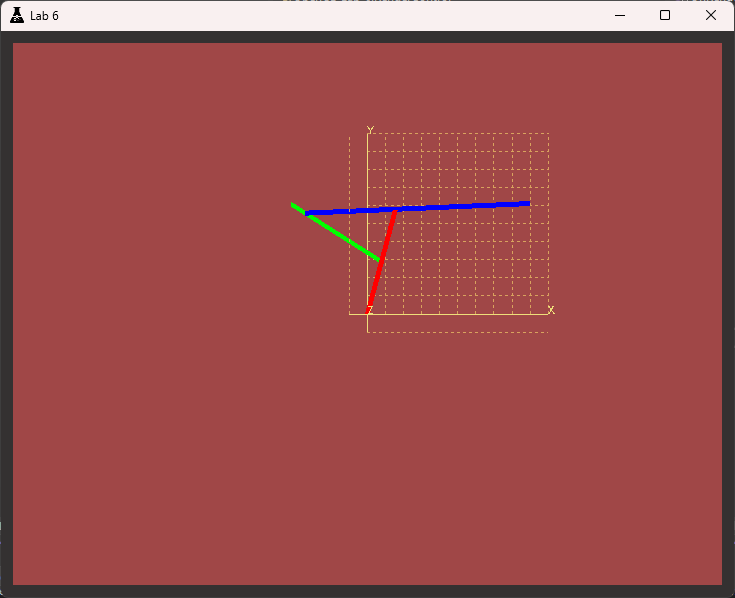


Рисунок 6.2 – Початковий стан кінематичної схеми після запуску програми

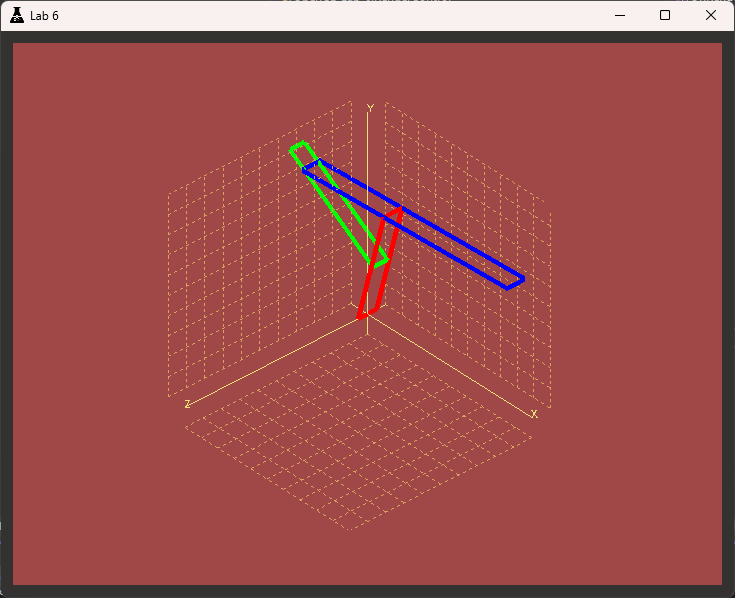


Рисунок 6.3 – Доопрацювання програми до тривимірної моделі

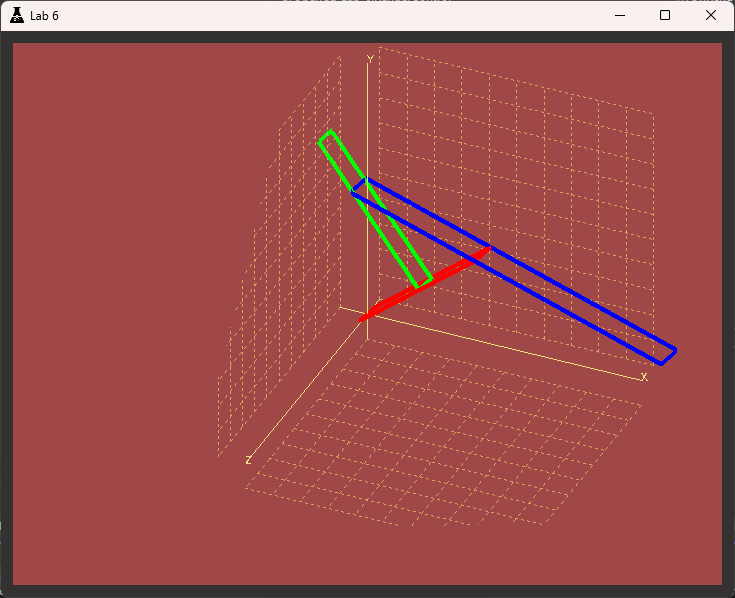


Рисунок 6.4 – Відображення координатної сітки X0Z та каркасу квадратичних об’єктів

### 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 6.1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблиця 1.1 | | | | |
| № з/п | Складність | Вимоги | Бали | Зроблено |
| 1 | Базовий рівень | Реалізація програми двовимірної моделі маніпулятора відповідно до варіанта | 5 | **+** |
| 2 | Доопрацювання програми до тривимірної моделі (обертання точки спостереження, масштаб) | 1 | **+** |
| 3 | Управління моделлю та точкою спостереження маніпулятором «миша» та/або клавіатурою | 1 | **+** |
| 4 | Використання квадратичних примітивів для відображення кінематичної схеми | 3 | **-** |
| 5 | Використання освітлення та визначення матеріалів командою glColorMaterial (...) | 4 | **-** |
| 6 | Вміст звіту відповідає прикладу оформлення | 6 | **+** |
| 7 | Підвищений рівень | Використання ООП (розробка власних класів) | 1 | **-** |
| 8 | Використання текстур для елементівкінематичної схеми | 2 | **-** |
| 9 | Визначення матеріалів командою glMaterial (...), використання прозорості | 2 | **-** |
| 10 | Використання перспективної проекції для відображення моделі маніпулятора | 1 | **-** |
| 11 | Реалізація освітлення з тінню від моделі маніпулятора | 4 | **-** |

# Практична робота 7. ЕКРАННА ЗАСТАВКА З АНІМАЦІЄЮ

## Завдання, варіант № 24

Використовуючи засоби, указані викладачем, створити програму екранної заставки (ScreenSaver) з анімацією (табл. 7.1). Загальний алгоритм такої програми в ОС Windows показано на рис. 7.1. Сюжет анімації обрати самостійно і погодити з викладачем.

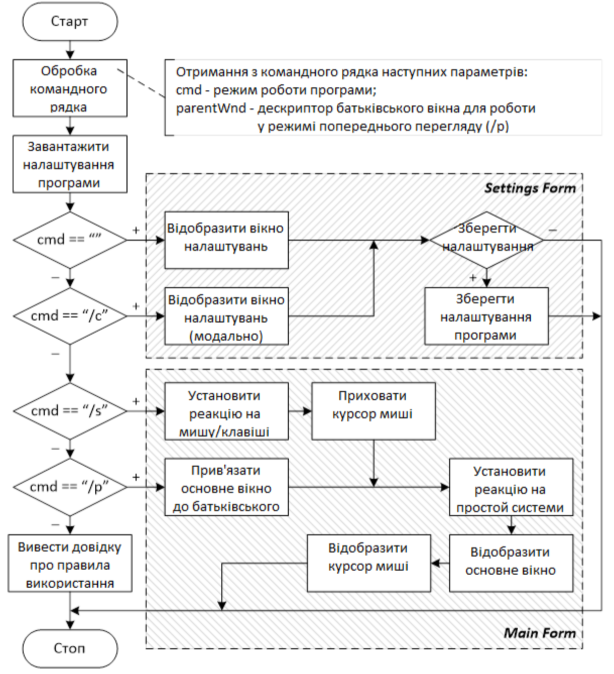


Рис. 7.1 Загальний алгоритм роботи екранної заставки в ОС Windows

## 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) (див. рис. 7.1), 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 (рис. 7.2).

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.

Якщо проект екранної заставки створено з урахуванням шаблону, то етапи розробки наведено у списку задач (TaskList, рис.7.3) середовища розробки.

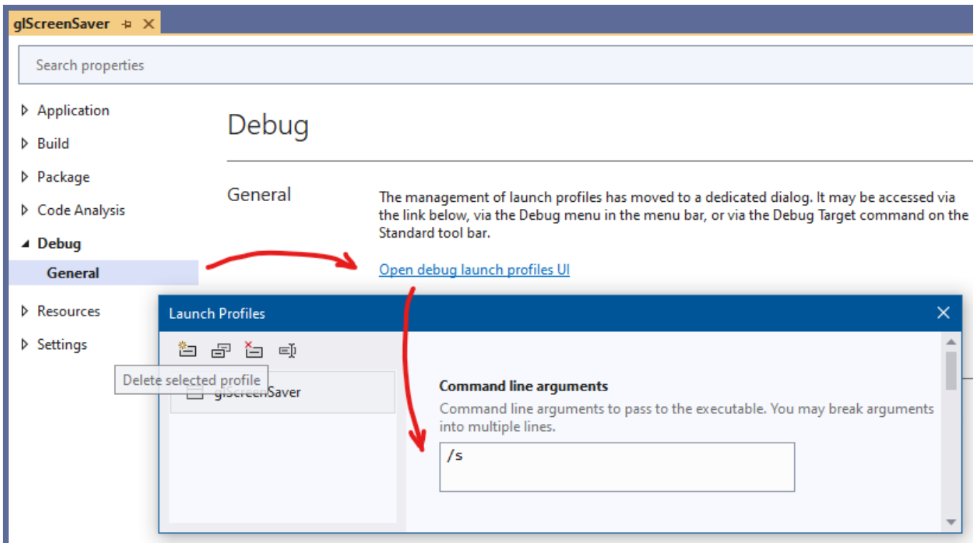


Рис. 7.2. Використання командного рядка у властивостях проєкту

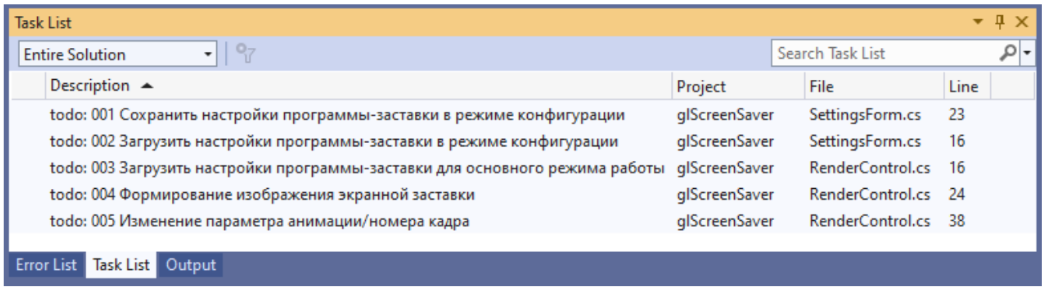


Рис. 7.3. Список задач проекта с этапами разработки экранной заставки

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 (рис. 7.4). Послідовність отримання exe-файлу екранної заставки Visual Studio 2022 наведена в додатку.

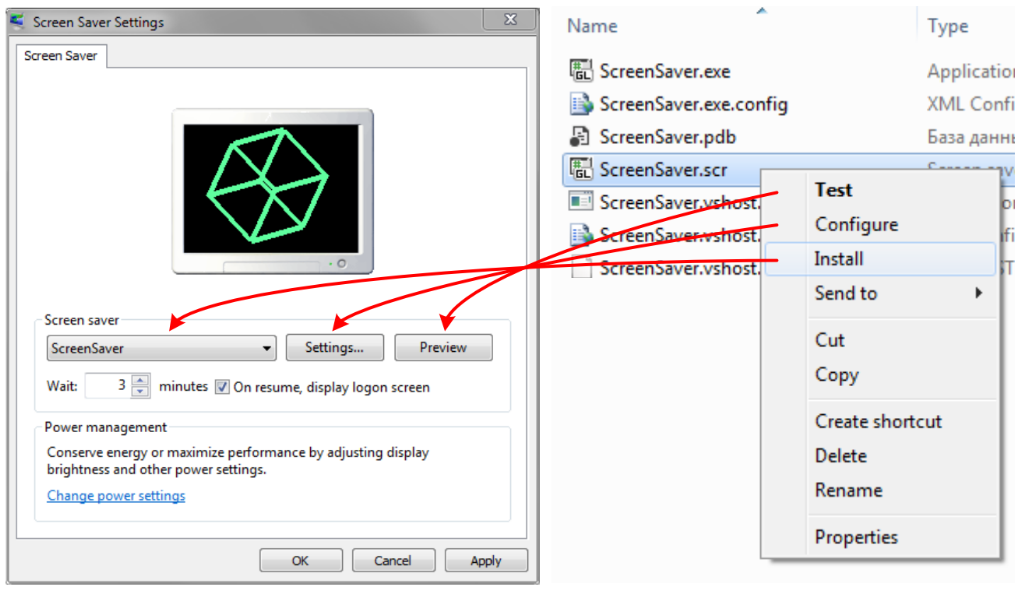


Рис. 7.4. Встановлення, запуск і налаштування екранної заставки Windows

## Системна інформація

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

Processor AMD Ryzen 5 2600 3.40 GHz

RAM 16.0 GB (15.9 GB usable)

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

Edition Windows 11 Version 23H2

IDE Microsoft Visual Studio Enterprise 2022 (64-bit) version 17.11.5

## Результати виконання практичної роботи

### Розв'язання завдання

Реалізація повноекранного режиму (команда «/s»).

Реалізація налаштування програми Screen Saver (команда «/c»).

Збереження конфігурації і налаштувань програми ScreenSaver у реєстрі ОС. Рядки 40 - 56, файл SettingsForm.cs, додаток G.

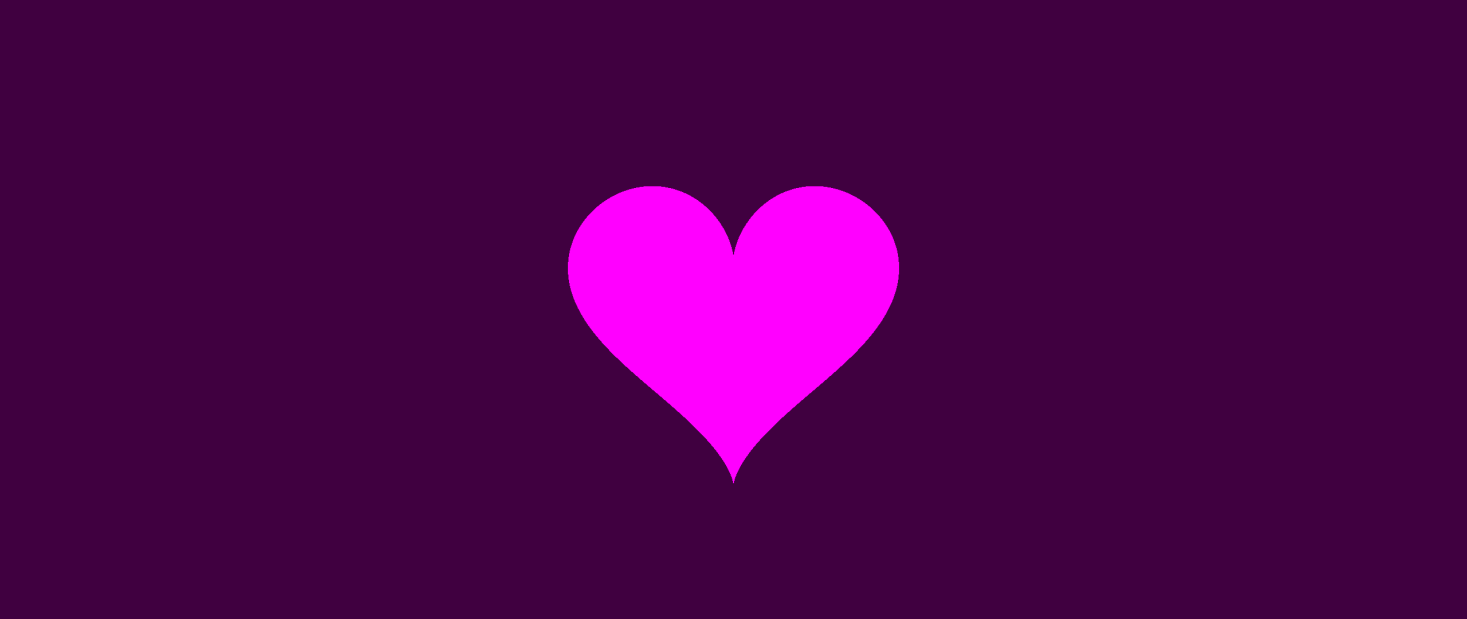


Рисунок 7.1 – Реалізація повноекранного режиму

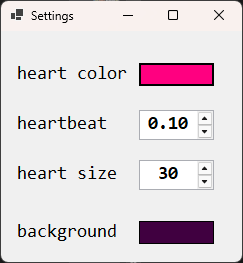


Рисунок 7.2 – Реалізація налаштування програми Screen Saver

### Контроль виконання вимог та елементів завдання

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблиця 7.1 | | | | |
| № з/п | Складність | Вимоги | Бали | Зроблено |
| 1 | Базовий рівень | Прив’язка анімації до механізму простою операційної системи | 2 | **+** |
| 2 | Реалізація повноекранного режиму (команда «/s») | 1 | **+** |
| 3 | Реалізація налаштування програми Screen Saver (команда «/c») | 1 | **+** |
| 4 | Реалізація попереднього перегляду (команда «/p») | 1 | **-** |
| 5 | Підвищений рівень | Збереження конфігурації і налаштувань програми ScreenSaver у реєстрі ОС | 1 | **+** |
| 6 | 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>.
2. Вивчення розривних функцій за допомогою Wolfram | Alpha [Електронний ресурс] // Математика, статистика, анализ данных в WolframAlpha. – Режим доступу : <http://www.wolframalpha-ru.com/2012/06/wolframalpha_16.html.>
3. Discontinuities [Електронний ресурс] // WolframAlpha Computational intelligence. – Режим доступу : <https://www.wolframalpha.com/input/?i=discontinuities+1%2F(x-1).>
4. Роджерс, Д. Математические основы машинной графики / Д. Роджерс, Дж. Адамс ; пер. с англ. – М. : Мир, 2001. – 604 с.
5. Препарата, Ф. Вычислительная геометрия : Введение / Ф. Препарата, М. Шеймос. – М. : Мир, 1989. – 478 с.
6. Аммерал, Л. Интерактивная трехмерная машинная графика / Л. Аммерал. – М. : «Сол Систем», 1992. – 315 с.
7. Шикин, Е. В. Компьютерная графика. Динамика, реалистические изображения / Е. В. Шикин, А. В. Боресков. – М. : Диалог-МИФИ, 1996. – 288 с.

# Додаток А. Лістинг програми до практичної роботи №1

### Код файлу (MainForm.cs)

1. public partial class MainForm : Form
2. {
3. public MainForm()
4. {
5. InitializeComponent();
6. }

### Код файлу (RenderControl.cs)

1. using System;
2. namespace Lab\_1
3. {
4. public partial class RenderControl : OpenGL
5. {
6. private float Xmin, Xmax;
7. private float Ymin, Ymax;
8. private DrawGrid \_grid;
9. private DrawFigure \_figure;
11. public RenderControl()
12. {
13. InitializeComponent();
14. }
15. private void Start(object sender, EventArgs e)
16. {
17. Xmin = -8.5f;
18. Xmax = 0.5f;
19. Ymin = -3.5f;
20. Ymax = 0.5f;
21. \_grid = new DrawGrid(Xmin, Xmax, Ymin, Ymax);
22. \_figure = new DrawFigure(Xmin, Xmax, Ymin, Ymax);
23. }
24. private void OnRender(object sender, EventArgs e)
25. {
26. glClear(GL\_COLOR\_BUFFER\_BIT);
27. glLoadIdentity();
28. glViewport(0, 0, Width, Height);
29. glOrtho(Xmin - 2, Xmax + 2, Ymin - 2, Ymax + 2, -1, 1);
30. \_grid.Draw();
31. glColor3d(216f / 255f, 162f / 255f, 94f / 255f);
32. DrawText("X1", Xmin, Ymin - 1.5);
33. DrawText("X2", Xmax, Ymin - 1.5);
34. DrawText("Y1", Xmin - 1.5, Ymin);
35. DrawText("Y2", Xmin - 1.5, Ymax);
36. glColor3d(0, 0, 0);
37. \_figure.Draw();
38. }
39. }
40. }

### Код файлу (DrawGrid.cs)

1. internal class DrawGrid
2. {
3. private float Xmin, Xmax, Ymin, Ymax;
4. public DrawGrid(float Xmin, float Xmax, float Ymin, float Ymax)
5. {
6. this.Xmin = Xmin;
7. this.Xmax = Xmax;
8. this.Ymin = Ymin;
9. this.Ymax = Ymax;
10. }
11. public void Draw()
12. {
13. glColor3d(0f, 0f, 0f);
14. DGrid();
15. DAxis();
16. }
17. private void DGrid()
18. {
19. glLineStipple(1, 4369);
20. glEnable(GL\_LINE\_STIPPLE);
21. glLineWidth(2);
22. glBegin(GL\_LINES);
23. // draw horizontal lines
24. for (float i = Xmin; i <= Xmax; i += 1)
25. {
26. glVertex2d(i, Ymin - 1);
27. glVertex2d(i, Ymax + 1);
28. }
29. // draw vertical lines
30. for (float i = Ymin; i <= Ymax; i += 1)
31. {
32. glVertex2d(Xmin - 1, i);
33. glVertex2d(Xmax + 1, i);
34. }
35. glEnd();
36. glDisable(GL\_LINE\_STIPPLE);
37. glLineWidth(1);
38. }
39. private void DAxis()
40. {
41. glLineWidth(4);
42. glBegin(GL\_LINES);
43. // Y
44. glVertex2d(Xmin - 1, Ymin - 0.2);
45. glVertex2d(Xmin - 1, Ymax + 1);
46. // arrow
47. glVertex2d(Xmin - 1, Ymax + 1);
48. glVertex2d(Xmin - 1 - 0.2, Ymax + 1 - 0.2);
49. glVertex2d(Xmin - 1, Ymax + 1);
50. glVertex2d(Xmin - 1 + 0.2, Ymax + 1 - 0.2);
51. // strokes
52. for (float i = Ymin; i <= Ymax; i++)
53. {
54. glVertex2d(Xmin - 1, i);
55. glVertex2d(Xmin - 1 - 0.2, i);
56. }
57. // X
58. glVertex2d(Xmin - 0.2, Ymin - 1);
59. glVertex2d(Xmax + 1, Ymin - 1);
60. // arrow
61. glVertex2d(Xmax + 1, Ymin - 1);
62. glVertex2d(Xmax + 1 - 0.2, Ymin - 1 - 0.2);
63. glVertex2d(Xmax + 1, Ymin - 1);
64. glVertex2d(Xmax + 1 - 0.2, Ymin - 1 + 0.2);
65. // strokes
66. for (float i = Xmin; i <= Xmax; i++)
67. {
68. glVertex2d(i, Ymin - 1);
69. glVertex2d(i, Ymin - 1 - 0.2);
70. }
71. glEnd();
72. glLineWidth(1);
73. }
74. }

### Код файлу (DrawFigure.cs)

1. using static Lab\_1.OpenGL;
2. namespace Lab\_1
3. {
4. internal class DrawFigure
5. {
6. private float Xmin, Xmax, Ymin, Ymax;
7. private Vertex[] VertexesFigure;
8. private float merge;
9. public DrawFigure(float Xmin, float Xmax, float Ymin, float Ymax)
10. {
11. this.Xmin = Xmin;
12. this.Xmax = Xmax;
13. this.Ymin = Ymin;
14. this.Ymax = Ymax;
15. VertexesFigure = new Vertex[] {
16. new Vertex(Xmin, Ymin),
17. new Vertex(Xmin + 1f, Ymin),
18. new Vertex(Xmin + 2f, Ymin + 1f),
19. new Vertex(Xmin + 3f, Ymin + 3f),
20. new Vertex(Xmin + 2f, Ymin + 4f),
21. new Vertex(Xmin, Ymin + 3f),
22. };
23. merge = 6f;
24. }
25. public void Draw()
26. {
27. glColor3d(238f / 255f, 223f / 255f, 122f / 255f);
28. DFigure();
29. DPorintsFigure();
30. glColor3d(0, 0, 0);
31. }
32. private void DFigure()
33. {
34. glLineWidth(8);
35. glBegin(GL\_LINE\_LOOP);
36. glVertex2d(VertexesFigure[0].x, VertexesFigure[0].y);
37. glVertex2d(VertexesFigure[1].x, VertexesFigure[1].y);
38. glVertex2d(VertexesFigure[2].x, VertexesFigure[2].y);
39. glVertex2d(VertexesFigure[3].x, VertexesFigure[3].y);
40. glVertex2d(VertexesFigure[4].x, VertexesFigure[4].y);
41. glVertex2d(VertexesFigure[5].x, VertexesFigure[5].y);
42. glEnd();
43. glLineWidth(1);
44. }
45. private void DPorintsFigure()
46. {
47. glPointSize(16);
48. glBegin(GL\_POINTS);
49. glVertex2d(VertexesFigure[0].x + merge, VertexesFigure[0].y);
50. glVertex2d(VertexesFigure[1].x + merge, VertexesFigure[1].y);
51. glVertex2d(VertexesFigure[2].x + merge, VertexesFigure[2].y);
52. glVertex2d(VertexesFigure[3].x + merge, VertexesFigure[3].y);
53. glVertex2d(VertexesFigure[4].x + merge, VertexesFigure[4].y);
54. glVertex2d(VertexesFigure[5].x + merge, VertexesFigure[5].y);
55. glEnd();
56. glPointSize(1);
57. }
58. }
59. }

### Код файлу (Vertex.cs)

1. public struct Vertex
2. {
3. public float x;
4. public float y;
5. public Vertex(float x, float y)
6. {
7. this.x = x;
8. this.y = y;
9. }
10. }

# Додаток B. Лістинг програми до практичної роботи №2

### Код файлу (MainForm.cs)

1. public partial class MainForm : Form
2. {
3. public MainForm()
4. {
5. InitializeComponent();
6. }
7. private void hor\_numeric\_ValueChanged(object sender, System.EventArgs e)
8. {
9. renderControl1.figures.horCount = (int)hor\_numeric.Value;
10. renderControl1.UpdateSideWindow();
11. }
12. private void ver\_numeric\_ValueChanged(object sender, System.EventArgs e)
13. {
14. renderControl1.figures.verCount = (int)ver\_numeric.Value;
15. renderControl1.UpdateSideWindow();
16. }
17. private void fillMode\_rb\_CheckedChanged(object sender, System.EventArgs e)
18. {
19. renderControl1.figures.polygonMode = GL\_FILL;
20. renderControl1.Invalidate();
21. }
22. private void lineMode\_rb\_CheckedChanged(object sender, System.EventArgs e)
23. {
24. renderControl1.figures.polygonMode = GL\_LINE;
25. renderControl1.Invalidate();
26. }
27. private void pointMode\_rb\_CheckedChanged(object sender, System.EventArgs e)
28. {
29. renderControl1.figures.polygonMode = GL\_POINT;
30. renderControl1.Invalidate();
31. }
32. private void smoothModel\_rb\_CheckedChanged(object sender, System.EventArgs e)
33. {
34. renderControl1.figures.shadeModel = GL\_SMOOTH;
35. renderControl1.Invalidate();
36. }
37. private void flatModel\_rb\_CheckedChanged(object sender, System.EventArgs e)
38. {
39. renderControl1.figures.shadeModel = GL\_FLAT;
40. renderControl1.Invalidate();
41. }
42. }

### Код файлу (RenderControl.cs)

1. using System;
2. namespace Lab\_2
3. {
4. public partial class RenderControl : OpenGL
5. {
6. private WindowSize windowSize;
7. private float sideWindow;
8. public DrawFigures figures { get; set; }
10. public RenderControl()
11. {
12. InitializeComponent();
13. }
14. private void OnStart(object sender, EventArgs e)
15. {
16. sideWindow = 20f;
17. windowSize = new WindowSize(-sideWindow, sideWindow, -sideWindow, sideWindow);
18. figures = new DrawFigures(windowSize);
19. }
20. private void OnRender(object sender, EventArgs e)
21. {
22. glClear(GL\_COLOR\_BUFFER\_BIT);
23. glLoadIdentity();
24. // Устанавливаем квадратную область просмотра, сохраняя пропорции
25. if (Width > Height)
26. {
27. // Если ширина больше высоты, центрируем по горизонтали
28. glViewport((Width - Height) / 2, 0, Height, Height);
29. }
30. else
31. {
32. // Если высота больше ширины, центрируем по вертикали
33. glViewport(0, (Height - Width) / 2, Width, Width);
34. }
35. glOrtho(windowSize.Xmin, windowSize.Xmax, windowSize.Ymin, windowSize.Ymax, -1, 1);
36. figures.Draw();
37. }
38. public void UpdateSideWindow()
39. {
40. float \_hor = figures.horCount \* figures.sideFigure \* 2.5f;
41. float \_ver = figures.verCount \* (figures.sideFigure \* MathF.Sqrt(3));
42. if (\_hor > \_ver)
43. {
44. sideWindow = \_hor / 2 + 5;
45. }
46. else
47. {
48. sideWindow = \_ver / 2 + 10 + (figures.sideFigure \* MathF.Sqrt(3)) / 2;
49. }
50. windowSize = new WindowSize(-sideWindow, sideWindow, -sideWindow, sideWindow);
51. figures.SetWindowSize(windowSize);
52. Invalidate();
53. }
54. }
55. }

### Код файлу (DrawFigures.cs)

1. using System;
2. using static Lab\_2.OpenGL;
3. namespace Lab\_2
4. {
5. public class DrawFigures
6. {
7. private WindowSize windowSize;
8. private Random random = new Random();
9. public float sideFigure { get; }
10. public int horCount { set; get; }
11. public int verCount { set; get; }
12. public uint polygonMode { set; get; }
13. public uint shadeModel { set; get; }
14. public DrawFigures(WindowSize windowSize)
15. {
16. this.windowSize = windowSize;
17. sideFigure = 10f;
18. horCount = 1;
19. verCount = 1;
20. polygonMode = GL\_FILL;
21. shadeModel = GL\_SMOOTH;
22. }
23. public void Draw()
24. {
25. for(int ver = 0; ver < verCount; ver++)
26. {
27. for (int hor = 0; hor < horCount; hor++)
28. {
29. DFigures(hor, ver);
30. }
31. }
32. }
33. public void SetWindowSize(WindowSize windowSize)
34. {
35. this.windowSize = windowSize;
36. }
37. private double RandomColor() =>
38. random.NextDouble();
39. private void DFigures(int hor, int ver)
40. {
41. float zero\_x = (1.5f - (1.25f \* (horCount - 1))) \* sideFigure;
42. float zero\_y = (verCount - 1) \* (sideFigure \* MathF.Sqrt(3)) / 2;
43. zero\_y = hor % 2 != 0 ? zero\_y - (sideFigure \* MathF.Sqrt(3)) / 2 : zero\_y;
44. float coef\_x = hor \* (sideFigure \* 2.5f);
45. float coef\_y = ver \* (sideFigure \* MathF.Sqrt(3));
46. float pos\_x = zero\_x + coef\_x;
47. float pos\_y = zero\_y - coef\_y;
48. glPolygonMode(GL\_FRONT\_AND\_BACK, polygonMode);
49. glShadeModel(shadeModel);
50. glPointSize(5);
51. glLineWidth(5);
52. // figure 1
53. glBegin(GL\_POLYGON);
54. glColor3d(0, 1, 0);
55. glVertex2d(pos\_x - sideFigure, pos\_y);
56. glVertex2d(pos\_x - (3 \* sideFigure), pos\_y);
57. glVertex2d(pos\_x - (2.5 \* sideFigure), pos\_y + ((sideFigure \* Math.Sqrt(3)) / 2));
58. glColor3d(1, 0, 0);
59. glVertex2d(pos\_x - (1.5 \* sideFigure), pos\_y + ((sideFigure \* Math.Sqrt(3)) / 2));
60. glEnd();
61. // figure 2
62. glBegin(GL\_POLYGON);
63. glColor3d(1, 0, 0);
64. glVertex2d(pos\_x - sideFigure, pos\_y);
65. glVertex2d(pos\_x - (1.5 \* sideFigure), pos\_y + ((sideFigure \* Math.Sqrt(3)) / 2));
66. glColor3d(0, 1, 0);
67. glVertex2d(pos\_x - (0.5 \* sideFigure), pos\_y + ((sideFigure \* Math.Sqrt(3)) / 2));
68. glEnd();
69. // figure 3
70. glBegin(GL\_POLYGON);
71. glColor3d(0, 1, 0);
72. glVertex2d(pos\_x, pos\_y);
73. glVertex2d(pos\_x - (0.5 \* sideFigure), pos\_y + ((sideFigure \* Math.Sqrt(3)) / 2));
74. glColor3d(1, 1, 0);
75. glVertex2d(pos\_x - sideFigure, pos\_y);
76. glEnd();
77. // figure 4
78. glBegin(GL\_POLYGON);
79. glColor3d(1, 1, 0);
80. glVertex2d(pos\_x - sideFigure, pos\_y);
81. glVertex2d(pos\_x - (3 \* sideFigure), pos\_y);
82. glVertex2d(pos\_x - (2.5 \* sideFigure), pos\_y - ((sideFigure \* Math.Sqrt(3)) / 2));
83. glColor3d(1, 0, 0);
84. glVertex2d(pos\_x - (1.5 \* sideFigure), pos\_y - ((sideFigure \* Math.Sqrt(3)) / 2));
85. glEnd();
86. // figure 5
87. glBegin(GL\_POLYGON);
88. glColor3d(1, 0, 0);
89. glVertex2d(pos\_x - sideFigure, pos\_y);
90. glVertex2d(pos\_x - (1.5 \* sideFigure), pos\_y - ((sideFigure \* Math.Sqrt(3)) / 2));
91. glColor3d(1, 1, 0);
92. glVertex2d(pos\_x - (0.5 \* sideFigure), pos\_y - ((sideFigure \* Math.Sqrt(3)) / 2));
93. glEnd();
94. // figure 6
95. glBegin(GL\_POLYGON);
96. glColor3d(1, 1, 0);
97. glVertex2d(pos\_x, pos\_y);
98. glVertex2d(pos\_x - (0.5 \* sideFigure), pos\_y - ((sideFigure \* Math.Sqrt(3)) / 2));
99. glColor3d(0, 1, 0);
100. glVertex2d(pos\_x - sideFigure, pos\_y);
101. glEnd();
102. }
103. }
104. }

### Код файлу (WindowSize.cs)

1. public struct WindowSize
2. {
3. public float Xmin, Xmax;
4. public float Ymin, Ymax;
5. public WindowSize(float Xmin, float Xmax, float Ymin, float Ymax)
6. {
7. this.Xmin = Xmin;
8. this.Xmax = Xmax;
9. this.Ymin = Ymin;
10. this.Ymax = Ymax;
11. }
12. }

# Додаток C. Лістинг програми до практичної роботи №3

### Код файлу (MainForm.cs)

1. public partial class MainForm : Form
2. {
3. private decimal step = (decimal)0.1f;
5. public MainForm()
6. {
7. InitializeComponent();
8. Xmax\_numeric.Minimum = Xmin\_numeric.Value + step;
9. Xmin\_numeric.Maximum = Xmax\_numeric.Value - step;
10. Ymax\_numeric.Minimum = Ymin\_numeric.Value + step;
11. Ymin\_numeric.Maximum = Ymax\_numeric.Value - step;
12. renderControl1.Ymax\_numeric = Ymax\_numeric;
13. renderControl1.Ymin\_numeric = Ymin\_numeric;
14. }
15. private void Xmin\_numeric\_ValueChanged(object sender, System.EventArgs e)
16. {
17. renderControl1.UpdateWindowSize(new WindowSize(
18. (float)Xmin\_numeric.Value,
19. (float)Xmax\_numeric.Value,
20. (float)Ymin\_numeric.Value,
21. (float)Ymax\_numeric.Value));
22. Xmax\_numeric.Minimum = Xmin\_numeric.Value + step;
23. }
24. private void Xmax\_numeric\_ValueChanged(object sender, System.EventArgs e)
25. {
26. renderControl1.UpdateWindowSize(new WindowSize(
27. (float)Xmin\_numeric.Value,
28. (float)Xmax\_numeric.Value,
29. (float)Ymin\_numeric.Value,
30. (float)Ymax\_numeric.Value));
31. Xmin\_numeric.Maximum = Xmax\_numeric.Value - step;
32. }
33. private void Ymin\_numeric\_ValueChanged(object sender, System.EventArgs e)
34. {
35. renderControl1.UpdateWindowSize(new WindowSize(
36. (float)Xmin\_numeric.Value,
37. (float)Xmax\_numeric.Value,
38. (float)Ymin\_numeric.Value,
39. (float)Ymax\_numeric.Value));
40. Ymax\_numeric.Minimum = Ymin\_numeric.Value + step;
41. }
42. private void Ymax\_numeric\_ValueChanged(object sender, System.EventArgs e)
43. {
44. renderControl1.UpdateWindowSize(new WindowSize(
45. (float)Xmin\_numeric.Value,
46. (float)Xmax\_numeric.Value,
47. (float)Ymin\_numeric.Value,
48. (float)Ymax\_numeric.Value));
49. Ymin\_numeric.Maximum = Ymax\_numeric.Value - step;
50. }
51. private void Yauto\_cb\_CheckedChanged(object sender, System.EventArgs e)
52. {
53. renderControl1.UpdateFunctions((int)points\_numeric.Value,
54. Yauto\_cb.Checked);
55. if(Yauto\_cb.Checked)
56. {
57. Ymanual\_panel.Enabled = false;
58. }
59. else
60. {
61. Ymanual\_panel.Enabled = true;
62. }
63. }
64. private void funcOne\_rb\_CheckedChanged(object sender, System.EventArgs e)
65. {
66. renderControl1.funcIndex = 1;
67. renderControl1.Invalidate();
68. }
69. private void funcTwo\_rb\_CheckedChanged(object sender, System.EventArgs e)
70. {
71. renderControl1.funcIndex = 2;
72. renderControl1.Invalidate();
73. }
74. private void points\_numeric\_ValueChanged(object sender, System.EventArgs e)
75. {
76. renderControl1.UpdateFunctions((int)points\_numeric.Value,
77. Yauto\_cb.Checked);
78. }
79. private void step\_numeric\_ValueChanged(object sender, System.EventArgs e)
80. {
81. renderControl1.UpdateConfig((float)step\_numeric.Value);
82. }
83. }

### Код файлу (RenderControl.cs)

1. using System;
2. using System.Windows.Forms;
3. namespace Lab\_3
4. {
5. public partial class RenderControl : OpenGL
6. {
7. public WindowSize windowSize;
8. public DrawConfig drawConfig;
9. private float \_width;
10. private float \_height;
11. public int points { set; get; }
12. public int funcIndex { set; get; }
13. public bool autoSizeY { set; get; }
14. public NumericUpDown Ymin\_numeric { set; get; }
15. public NumericUpDown Ymax\_numeric { set; get; }
16. private DrawFuncOne funcOne;
17. private DrawFuncTwo funcTwo;
18. public RenderControl()
19. {
20. InitializeComponent();
21. }
22. private void OnStart(object sender, EventArgs e)
23. {
24. windowSize = new WindowSize(-3f, 3f, -3f, 3f);
25. \_width = -windowSize.Xmin + windowSize.Xmax;
26. \_height = -windowSize.Ymin + windowSize.Ymax;
27. drawConfig = new DrawConfig(\_width, \_height, 0.2f, 1f);
28. funcIndex = 1;
29. autoSizeY = false;
30. points = 1000;
31. funcOne = new DrawFuncOne(windowSize, drawConfig, points, autoSizeY, Ymin\_numeric, Ymax\_numeric);
32. funcTwo = new DrawFuncTwo(windowSize, drawConfig, points, autoSizeY, Ymin\_numeric, Ymax\_numeric);
33. }
34. private void OnRender(object sender, EventArgs e)
35. {
36. glClear(GL\_COLOR\_BUFFER\_BIT);
37. glLoadIdentity();
38. glViewport(0, 0, Width, Height);
39. glOrtho(windowSize.Xmin - drawConfig.margin, windowSize.Xmax + drawConfig.margin,
40. windowSize.Ymin - drawConfig.margin, windowSize.Ymax + drawConfig.margin,
41. -1, 1);
42. switch(funcIndex)
43. {
44. case 1:
45. funcOne.DrawCoordinateSystem();
46. funcOne.DrawFunction();
47. break;
48. case 2:
49. funcTwo.DrawCoordinateSystem();
50. funcTwo.DrawFunction();
51. break;
52. }
53. DrawTextAxis();
54. }
55. private void DrawTextAxis()
56. {
57. glColor3d(238f / 255f, 223f / 255f, 122f / 255f);
58. // X axis
59. float \_start = drawConfig.step \* (((-windowSize.Xmin) / drawConfig.step) % 1);
60. for (float colum = \_start; colum < \_width + drawConfig.step; colum += drawConfig.step)
61. {
62. if ((windowSize.Xmin + colum != 0) && (windowSize.Xmin + colum >= windowSize.Xmin && windowSize.Xmin + colum <= windowSize.Xmax))
63. {
64. DrawText((windowSize.Xmin + colum).ToString("F1"), windowSize.Xmin + colum, -drawConfig.step / 3);
65. }
66. }
67. // Y axis
68. \_start = drawConfig.step \* ((windowSize.Ymax / drawConfig.step) % 1);
69. for (float row = \_start; row < drawConfig.height + drawConfig.step; row += drawConfig.step)
70. {
71. if ((windowSize.Ymax - row != 0) && (windowSize.Ymax - row <= windowSize.Ymax && windowSize.Ymax - row >= windowSize.Ymin))
72. {
73. DrawText((windowSize.Ymax - row).ToString("F1"), -drawConfig.step / 3, windowSize.Ymax - row);
74. }
75. }
76. // 0
77. DrawText((0).ToString("F1"), -drawConfig.step / 3, -drawConfig.step / 3);
78. }
79. public void UpdateWindowSize(WindowSize windowSize)
80. {
81. this.windowSize = windowSize;
82. \_width = -windowSize.Xmin + windowSize.Xmax;
83. \_height = -windowSize.Ymin + windowSize.Ymax;
84. UpdateConfig(drawConfig.step);
85. }
86. public void UpdateConfig(float step)
87. {
88. drawConfig = new DrawConfig(\_width, \_height, 0.2f, step);
89. UpdateFunctions(points, autoSizeY);
90. }
91. public void UpdateFunctions(int points, bool autoSizeY)
92. {
93. this.points = points;
94. this.autoSizeY = autoSizeY;
96. funcOne = new DrawFuncOne(windowSize, drawConfig, points, autoSizeY, Ymin\_numeric, Ymax\_numeric);
97. funcTwo = new DrawFuncTwo(windowSize, drawConfig, points, autoSizeY, Ymin\_numeric, Ymax\_numeric);
98. Invalidate();
99. }
100. }
101. }

### Код файлу (DrawConfig.cs)

1. public struct DrawConfig
2. {
3. public float width;
4. public float height;
5. public float margin;
6. public float step;
7. public DrawConfig(float width, float height, float margin, float step)
8. {
9. this.width = width;
10. this.height = height;
11. this.margin = margin;
12. this.step = step;
13. }
14. }

### Код файлу (Draw.cs)

1. using static Lab\_3.OpenGL;
2. namespace Lab\_3
3. {
4. internal class Draw
5. {
6. protected WindowSize windowSize;
7. protected DrawConfig config;
8. public Draw(WindowSize windowSize,DrawConfig config)
9. {
10. this.windowSize = windowSize;
11. this.config = config;
12. }
13. public void DrawCoordinateSystem()
14. {
15. DrawGrid();
16. DrawAxis();
17. }
18. private void DrawGrid()
19. {
20. glLineWidth(1);
21. glBegin(GL\_LINES);
22. glColor3d(216f / 255f, 162f / 255f, 94f / 255f);
23. float \_start = config.step \* ((windowSize.Ymax / config.step) % 1);
24. // draw horizontal lines
25. for (float row = \_start; row < config.height; row += config.step)
26. {
27. glVertex2d(windowSize.Xmin, windowSize.Ymax - row);
28. glVertex2d(windowSize.Xmax, windowSize.Ymax - row);
29. }
30. \_start = config.step \* (((-windowSize.Xmin) / config.step) % 1);
31. // draw vertical lines
32. for (float colum = \_start; colum < config.width; colum += config.step)
33. {
34. glVertex2d(windowSize.Xmin + colum, windowSize.Ymax);
35. glVertex2d(windowSize.Xmin + colum, windowSize.Ymin);
36. }
37. // draw borders
38. glVertex2d(windowSize.Xmin, windowSize.Ymax);
39. glVertex2d(windowSize.Xmax, windowSize.Ymax);
40. glVertex2d(windowSize.Xmin, windowSize.Ymin);
41. glVertex2d(windowSize.Xmax, windowSize.Ymin);
42. glVertex2d(windowSize.Xmax, windowSize.Ymax);
43. glVertex2d(windowSize.Xmax, windowSize.Ymin);
44. glVertex2d(windowSize.Xmin, windowSize.Ymax);
45. glVertex2d(windowSize.Xmin, windowSize.Ymin);
46. glEnd();
47. }
48. private void DrawAxis()
49. {
50. glLineWidth(2);
51. glBegin(GL\_LINES);
52. glColor3d(238f / 255f, 223f / 255f, 122f / 255f);
53. // X axis
54. glVertex2d(windowSize.Xmax, 0f);
55. glVertex2d(windowSize.Xmin, 0f);
56. float \_start = config.step \* (((-windowSize.Xmin) / config.step) % 1);
57. // draw vertical lines
58. for (float colum = \_start; colum < config.width + config.step; colum += config.step)
59. {
60. if (windowSize.Xmin + colum >= windowSize.Xmin && windowSize.Xmin + colum <= windowSize.Xmax)
61. {
62. glVertex2d(windowSize.Xmin + colum, 0);
63. glVertex2d(windowSize.Xmin + colum, -config.step / 5);
64. }
65. }
66. // Y axis
67. glVertex2d(0f, windowSize.Ymax);
68. glVertex2d(0f, windowSize.Ymin);
69. \_start = config.step \* ((windowSize.Ymax / config.step) % 1);
70. // draw horizontal lines
71. for (float row = \_start; row < config.height + config.step; row += config.step)
72. {
73. if (windowSize.Ymax - row <= windowSize.Ymax && windowSize.Ymax - row >= windowSize.Ymin)
74. {
75. glVertex2d(-config.step / 5, windowSize.Ymax - row);
76. glVertex2d(0, windowSize.Ymax - row);
77. }
78. }
79. glLineWidth(1);
80. glEnd();
81. }
82. }
83. }

### Код файлу (FuncOne.cs)

1. using System;
2. using System.Windows.Forms;
3. using static Lab\_3.OpenGL;
4. namespace Lab\_3
5. {
6. internal class DrawFuncOne : Draw
7. {
8. private int points;
9. private bool isAutoY;
10. private NumericUpDown Ymin\_numeric;
11. private NumericUpDown Ymax\_numeric;
12. public DrawFuncOne(WindowSize windowSize, DrawConfig drawConfig, int points, bool isAutoY, NumericUpDown Ymin\_numeric, NumericUpDown Ymax\_numeric)
13. : base(windowSize, drawConfig)
14. {
15. this.points = points;
16. this.isAutoY = isAutoY;
17. this.Ymin\_numeric = Ymin\_numeric;
18. this.Ymax\_numeric = Ymax\_numeric;
19. }
20. private float Func(float x) =>
21. MathF.Sin(MathF.PI \* x / 2) / (MathF.Cos(MathF.PI \* x) - MathF.PI);
22. public void DrawFunction()
23. {
24. float h = config.width / (points - 1);
25. float x = 0;
26. float y = 0;
27. float \_x = 0;
28. float \_y = 0;
29. float \_Ymin = float.MaxValue;
30. float \_Ymax = float.MinValue;
31. glLineWidth(4);
32. glBegin(GL\_LINES);
33. glColor3d(64f / 255f, 224f / 255f, 208f / 255f);
34. for (int i = 0; i < points; i++)
35. {
36. x = windowSize.Xmin + i \* h;
37. y = Func(x);
38. if (!isAutoY)
39. {
40. if (y > windowSize.Ymax)
41. {
42. y = windowSize.Ymax;
43. continue;
44. }
45. else if (y < windowSize.Ymin)
46. {
47. y = windowSize.Ymin;
48. continue;
49. }
50. }
51. if (i > 0)
52. {
53. glVertex2d(\_x, \_y);
54. glVertex2d(x, y);
55. if (\_y \* y <= 0)
56. {
57. DrawPoint((\_x + x) / 2, (\_y + y) / 2);
58. }
59. }
60. if (y < \_Ymin && y < 0)
61. {
62. \_Ymin = y;
63. }
64. if (y > \_Ymax && y > 0)
65. {
66. \_Ymax = y;
67. }
68. \_x = x;
69. \_y = y;
70. }
71. glEnd();
72. \_Ymin = MathF.Round(\_Ymin, 1);
73. \_Ymax = MathF.Round(\_Ymax, 1);
74. if (\_Ymin < -100) \_Ymin = -99.0f;
75. if (\_Ymax > 100) \_Ymax = 99.0f;
76. if (isAutoY)
77. {
78. Ymin\_numeric.Value = (decimal)(\_Ymin - config.step);
79. Ymax\_numeric.Value = (decimal)(\_Ymax + config.step);
80. }
81. }
82. private void DrawPoint(float x, float y)
83. {
84. // end draw function
85. glEnd();
86. // draw point
87. glPointSize(10);
88. glBegin(GL\_POINTS);
89. glColor3d(255f / 255f, 128f / 255f, 0f / 255f);
90. glVertex2d(x, y);
91. glEnd();
92. // continue draw function
93. glBegin(GL\_LINES);
94. glColor3d(64f / 255f, 224f / 255f, 208f / 255f);
95. }
96. }
97. }

### Код файлу (FuncTwo.cs)

1. using System;
2. using System.Windows.Forms;
3. using static Lab\_3.OpenGL;
4. namespace Lab\_3
5. {
6. internal class DrawFuncTwo : Draw
7. {
8. private int points;
9. private bool isAutoY;
10. private NumericUpDown Ymin\_numeric;
11. private NumericUpDown Ymax\_numeric;
12. public DrawFuncTwo(WindowSize windowSize, DrawConfig drawConfig, int points, bool isAutoY, NumericUpDown Ymin\_numeric, NumericUpDown Ymax\_numeric)
13. : base(windowSize, drawConfig)
14. {
15. this.points = points;
16. this.isAutoY = isAutoY;
17. this.Ymin\_numeric = Ymin\_numeric;
18. this.Ymax\_numeric = Ymax\_numeric;
19. }
20. private float Func(float x) =>
21. MathF.Cos(MathF.PI \* x) / MathF.Abs(MathF.Cos(MathF.PI \* x)) + MathF.Log10(MathF.Cos(MathF.PI \* x / 2) + 1) + 1;
22. public void DrawFunction()
23. {
24. float h = config.width / (points - 1);
25. float breakdown = 1f / points \* config.width;
26. float x = 0;
27. float y = 0;
28. float \_x = 0;
29. float \_y = 0;
30. float \_Ymin = float.MaxValue;
31. float \_Ymax = float.MinValue;
32. glLineWidth(4);
33. glBegin(GL\_LINES);
34. glColor3d(64f / 255f, 224f / 255f, 208f / 255f);
35. for (int i = 0; i < points; i++)
36. {
37. if (i > 0)
38. {
39. \_x = x;
40. \_y = y;
41. }
42. x = windowSize.Xmin + i \* h;
43. y = Func(x);
44. if (MathF.Abs(\_x - MathF.Round(\_x)) < 0.5f + breakdown && MathF.Abs(\_x - MathF.Round(\_x)) > 0.5f - breakdown)
45. {
46. continue;
47. }
48. else if (MathF.Abs(x - MathF.Round(x)) < 0.5f + breakdown && MathF.Abs(x - MathF.Round(x)) > 0.5f - breakdown)
49. {
50. DrawLinesBreakdown(x);
51. continue;
52. }
53. if (!isAutoY)
54. {
55. if (y > windowSize.Ymax)
56. {
57. y = windowSize.Ymax;
58. continue;
59. }
60. else if (y < windowSize.Ymin)
61. {
62. y = windowSize.Ymin;
63. continue;
64. }
65. }
66. if (i > 0)
67. {
68. glVertex2d(\_x, \_y);
69. glVertex2d(x, y);
70. }
71. if (y < \_Ymin && y < 0)
72. {
73. \_Ymin = y;
74. }
75. if (y > \_Ymax && y > 0)
76. {
77. \_Ymax = y;
78. }
79. }
80. glEnd();
81. \_Ymin = MathF.Round(\_Ymin, 1);
82. \_Ymax = MathF.Round(\_Ymax, 1);
83. if (\_Ymin < -100) \_Ymin = -99.0f + config.step;
84. if (\_Ymax > 100) \_Ymax = 99.0f - config.step;
85. if (isAutoY)
86. {
87. Ymin\_numeric.Value = (decimal)(\_Ymin);
88. Ymax\_numeric.Value = (decimal)(\_Ymax);
89. }
90. }
91. private void DrawLinesBreakdown(float x)
92. {
93. // end draw function
94. glEnd();
95. glLineStipple(10, 21845);
96. glEnable(GL\_LINE\_STIPPLE);
97. glLineWidth(2);
98. glBegin(GL\_LINES);
99. glColor3d(255f / 255f, 255f / 255f, 255f / 255f);
100. glVertex2d(x, windowSize.Ymin);
101. glVertex2d(x, windowSize.Ymax);
102. glEnd();
103. glDisable(GL\_LINE\_STIPPLE);
104. glLineWidth(4);
105. glBegin(GL\_LINES);
106. glColor3d(64f / 255f, 224f / 255f, 208f / 255f);
107. }
108. }
109. }

# Додаток D. Лістинг програми до практичної роботи №4

### Код файлу (MainForm.cs)

1. using System.Windows.Forms;
2. using static Lab\_4.OpenGL;
3. namespace Lab\_4
4. {
5. public partial class MainForm : Form
6. {
7. public MainForm()
8. {
9. InitializeComponent();
10. }
11. private void curve\_cb\_SelectedIndexChanged(object sender, System.EventArgs e)
12. {
13. switch (curve\_cb.SelectedIndex)
14. {
15. case 0:
16. renderControl1.SetDrawCurve(Curve.circle);
17. hyperbole\_panel.Enabled = false;
18. line\_panel.Enabled = false;
19. circle\_panel.Enabled = true;
20. break;
21. case 1:
22. renderControl1.SetDrawCurve(Curve.hyperbole);
23. hyperbole\_panel.Enabled = true;
24. line\_panel.Enabled = true;
25. circle\_panel.Enabled = false;
26. break;
27. }
28. }
29. private void el\_a\_numeric\_ValueChanged(object sender, System.EventArgs e)
30. {
31. renderControl1.SetCircle((float)circle\_radius\_numeric.Value);
32. }
33. private void point1\_x\_numeric\_ValueChanged(object sender, System.EventArgs e)
34. {
35. renderControl1.SetPointsLine(new PointLine((float)point1\_x\_numeric.Value, (float)point1\_y\_numeric.Value),
36. new PointLine((float)point2\_x\_numeric.Value, (float)point2\_y\_numeric.Value));
37. }
38. private void point1\_y\_numeric\_ValueChanged(object sender, System.EventArgs e)
39. {
40. renderControl1.SetPointsLine(new PointLine((float)point1\_x\_numeric.Value, (float)point1\_y\_numeric.Value),
41. new PointLine((float)point2\_x\_numeric.Value, (float)point2\_y\_numeric.Value));
42. }
43. private void point2\_x\_numeric\_ValueChanged(object sender, System.EventArgs e)
44. {
45. renderControl1.SetPointsLine(new PointLine((float)point1\_x\_numeric.Value, (float)point1\_y\_numeric.Value),
46. new PointLine((float)point2\_x\_numeric.Value, (float)point2\_y\_numeric.Value));
47. }
48. private void point2\_y\_numeric\_ValueChanged(object sender, System.EventArgs e)
49. {
50. renderControl1.SetPointsLine(new PointLine((float)point1\_x\_numeric.Value, (float)point1\_y\_numeric.Value),
51. new PointLine((float)point2\_x\_numeric.Value, (float)point2\_y\_numeric.Value));
52. }
53. private void par\_h\_numeric\_ValueChanged(object sender, System.EventArgs e)
54. {
55. renderControl1.SetHyperbole((float)hyp\_b\_numeric.Value,
56. (float)hyp\_a\_numeric.Value);
57. }
58. private void par\_k\_numeric\_ValueChanged(object sender, System.EventArgs e)
59. {
60. renderControl1.SetHyperbole((float)hyp\_b\_numeric.Value,
61. (float)hyp\_a\_numeric.Value);
62. }
63. private void MainForm\_Load(object sender, System.EventArgs e)
64. {
65. }
66. }
67. }

### Код файлу (RenderControl.cs)

1. using System;
2. namespace Lab\_4
3. {
4. public partial class RenderControl : OpenGL
5. {
6. private float size;
7. private float AspectRatio { get => (float)Width / Height; }
8. private float Xmin { get => (AspectRatio > 1) ? -size \* AspectRatio : -size; }
9. private float Xmax { get => (AspectRatio > 1) ? +size \* AspectRatio : +size; }
10. private float Ymin { get => (AspectRatio < 1) ? -size / AspectRatio : -size; }
11. private float Ymax { get => (AspectRatio < 1) ? +size / AspectRatio : +size; }
12. private float margin;
13. private float step;
14. private float \_heigth;
15. private float \_width;
16. private Curve curve;
17. private Line line;
18. private Circle circle;
19. private Hyperbole hyperbole;
20. private DrawSystemCoordinate systemCoordinate;
21. public RenderControl()
22. {
23. InitializeComponent();
24. }
25. private void OnStart(object sender, EventArgs e)
26. {
27. size = 3.0f;
28. margin = 0.2f;
29. step = 0.5f;
30. curve = Curve.circle;
31. line = new Line(new PointLine(0f, 0f),
32. new PointLine(0f, 0f));
33. circle = new Circle(1f);
34. hyperbole = new Hyperbole(1f, 1f);
35. UpdateHeightWidth();
36. }
37. private void OnRender(object sender, EventArgs e)
38. {
39. glClear(GL\_COLOR\_BUFFER\_BIT);
40. glLoadIdentity();
41. glViewport(0, 0, Width, Height);
42. glOrtho(Xmin - margin, Xmax + margin, Ymin - margin, Ymax + margin, -1, 1);
43. UpdateHeightWidth();
44. systemCoordinate.Draw();
45. DrawTextAxis();
46. switch(curve)
47. {
48. case Curve.circle:
49. DrawCircle();
50. break;
51. case Curve.hyperbole:
52. DrawHyperbole();
53. DrawLine();
54. break;
55. }
56. }
57. public void SetDrawCurve(Curve curve)
58. {
59. this.curve = curve;
60. Invalidate();
61. }
62. private void UpdateHeightWidth()
63. {
64. \_heigth = (-Ymin) + Ymax;
65. \_width = (-Xmin) + Xmax;
66. systemCoordinate = new DrawSystemCoordinate(Xmin, Xmax, Ymin, Ymax, \_heigth, \_width, step);
67. }
68. public void SetPointsLine(PointLine point\_1, PointLine point\_2)
69. {
70. line = new Line(point\_1, point\_2);
71. Invalidate();
72. }
73. public void SetHyperbole(float b, float a)
74. {
75. hyperbole = new Hyperbole(b, a);
76. Invalidate();
77. }
78. public void SetCircle(float r)
79. {
80. circle = new Circle(r);
81. Invalidate();
82. }
83. private void DrawTextAxis()
84. {
85. glColor3d(238f / 255f, 223f / 255f, 122f / 255f);
86. // X axis
87. float \_start = step \* (((-Xmin) / step) % 1);
88. for (float colum = \_start; colum < \_width + step; colum += step)
89. {
90. if (Xmin + colum >= 0.1f || Xmin + colum <= -0.1f)
91. {
92. DrawText((Xmin + colum).ToString("F1"), Xmin + colum, -step / 2);
93. }
94. }
95. // Y axis
96. \_start = step \* ((Ymax / step) % 1);
97. for (float row = \_start; row < \_heigth + step; row += step)
98. {
99. if (Ymax - row >= 0.1f || Ymax - row <= -0.1f)
100. {
101. DrawText((Ymax - row).ToString("F1"), -step / 2, Ymax - row);
102. }
103. }
104. // 0
105. DrawText((0).ToString(), -step / 2.5f, -step / 2.5f);
106. }
107. private void DrawCircle()
108. {
109. float x = 0;
110. float y = 0;
111. float \_x = 0;
112. float \_y = 0;
113. float c = 0.001f;
114. glLineWidth(3);
115. glBegin(GL\_LINES);
116. glColor3d(64f / 255f, 224f / 255f, 208f / 255f);
117. for (x = -circle.r; x < circle.r + c; x += c)
118. {
119. x = MathF.Round(x, 3);
120. y = MathF.Sqrt((circle.r \* circle.r) - (x \* x));
121. if (x > -circle.r)
122. {
123. glVertex2d(\_x, \_y);
124. glVertex2d(x, y);
125. glVertex2d(\_x, -\_y);
126. glVertex2d(x, -y);
127. }
128. \_y = y;
129. \_x = x;
130. }
131. glEnd();
132. }
133. private void DrawLine()
134. {
135. glLineWidth(3);
136. glBegin(GL\_LINES);
137. glColor3d(0f / 255f, 255f / 255f, 0f / 255f);
138. glVertex2d(line.point\_1.x, line.point\_1.y);
139. glVertex2d(line.point\_2.x, line.point\_2.y);
140. glEnd();
141. }
142. private void DrawPoint(PointLine el\_point\_1, PointLine el\_point\_2)
143. {
144. // circle
145. float x1 = el\_point\_1.x;
146. float y1 = el\_point\_1.y;
147. float x2 = el\_point\_2.x;
148. float y2 = el\_point\_2.y;
149. // line
150. float x3 = line.point\_1.x;
151. float y3 = line.point\_1.y;
152. float x4 = line.point\_2.x;
153. float y4 = line.point\_2.y;
154. // point
155. float x0 = 0;
156. float y0 = 0;
157. float D = (x2 - x1) \* (y4 - y3) - (y2 - y1) \* (x4 - x3);
158. float t1 = 0;
159. float t2 = 0;
160. if (D != 0)
161. {
162. t1 = ((x3 - x1) \* (y4 - y3) - (y3 - y1) \* (x4 - x3)) / D;
163. t2 = ((x3 - x1) \* (y2 - y1) - (y3 - y1) \* (x2 - x1)) / D;
164. if ((t1 >= 0 && t1 <= 1) && (t2 >= 0 && t2 <= 1))
165. {
166. x0 = (x2 - x1) \* t1 + x1;
167. y0 = (y2 - y1) \* t1 + y1;
168. // end draw function
169. glEnd();
170. glPointSize(20);
171. glBegin(GL\_POINTS);
172. glColor3d(255f / 255f, 255f / 255f, 0f / 255f);
173. glVertex2d(x0, y0);
174. glEnd();
175. // continue draw function
176. glBegin(GL\_LINES);
177. glColor3d(64f / 255f, 224f / 255f, 208f / 255f);
178. }
179. }
180. }
181. private void DrawHyperbole()
182. {
183. float x = 0, y = 0;
184. float \_x = 0, \_y = 0;
185. float c = 0.001f;
186. glLineWidth(3);
187. glBegin(GL\_LINES);
188. glColor3d(64f / 255f, 224f / 255f, 208f / 255f);
189. for (x = Xmin; x <= Xmax; x += c)
190. {
191. x = MathF.Round(x, 3);
192. y = hyperbole.b \* MathF.Sqrt((((x \* x) / (hyperbole.a \* hyperbole.a)) - 1));
193. if (y > Ymax || y < Ymin)
194. {
195. \_x = x;
196. \_y = y;
197. continue;
198. }
199. if (x > Xmin + c)
200. {
201. glVertex2d(\_x, \_y);
202. glVertex2d(x, y);
203. glVertex2d(\_x, -\_y);
204. glVertex2d(x, -y);
205. DrawPoint(new PointLine(\_x, \_y), new PointLine(x, y));
206. DrawPoint(new PointLine(\_x, -\_y), new PointLine(x, -y));
207. }
208. \_x = x;
209. \_y = y;
210. }
211. glEnd();
212. }
213. }
214. }

### Код файлу (DrawSystemCoordinate.cs)

1. using static Lab\_4.OpenGL;
2. namespace Lab\_4
3. {
4. internal class DrawSystemCoordinate
5. {
6. private float Xmin, Xmax;
7. private float Ymin, Ymax;
8. private float \_heigth;
9. private float \_width;
10. private float step;
11. public DrawSystemCoordinate(float Xmin, float Xmax, float Ymin, float Ymax, float \_heigth, float \_width, float step)
12. {
13. this.Xmin = Xmin;
14. this.Xmax = Xmax;
15. this.Ymin = Ymin;
16. this.Ymax = Ymax;
17. this.\_heigth = \_heigth;
18. this.\_width = \_width;
19. this.step = step;
20. }
21. public void Draw()
22. {
23. DrawGrid();
24. DrawAxis();
25. }
26. private void DrawGrid()
27. {
28. glLineWidth(1);
29. glBegin(GL\_LINES);
30. glColor3d(216f / 255f, 162f / 255f, 94f / 255f);
31. float \_start = step \* ((Ymax / step) % 1);
32. // draw horizontal lines
33. for (float row = \_start; row < \_heigth; row += step)
34. {
35. glVertex2d(Xmin, Ymax - row);
36. glVertex2d(Xmax, Ymax - row);
37. }
38. \_start = step \* (((-Xmin) / step) % 1);
39. // draw vertical lines
40. for (float colum = \_start; colum < \_width; colum += step)
41. {
42. glVertex2d(Xmin + colum, Ymax);
43. glVertex2d(Xmin + colum, Ymin);
44. }
45. // draw borders
46. glVertex2d(Xmin, Ymax);
47. glVertex2d(Xmax, Ymax);
48. glVertex2d(Xmin, Ymin);
49. glVertex2d(Xmax, Ymin);
50. glVertex2d(Xmax, Ymax);
51. glVertex2d(Xmax, Ymin);
52. glVertex2d(Xmin, Ymax);
53. glVertex2d(Xmin, Ymin);
54. glEnd();
55. }
56. private void DrawAxis()
57. {
58. glLineWidth(2);
59. glBegin(GL\_LINES);
60. glColor3d(238f / 255f, 223f / 255f, 122f / 255f);
61. // X axis
62. glVertex2d(Xmax, 0f);
63. glVertex2d(Xmin, 0f);
64. float \_start = step \* (((-Xmin) / step) % 1);
65. // draw vertical lines
66. for (float colum = \_start; colum < \_width + step; colum += step)
67. {
68. glVertex2d(Xmin + colum, 0);
69. glVertex2d(Xmin + colum, -step / 3);
70. }
71. // Y axis
72. glVertex2d(0f, Ymax);
73. glVertex2d(0f, Ymin);
74. \_start = step \* ((Ymax / step) % 1);
75. // draw horizontal lines
76. for (float row = \_start; row < \_heigth + step; row += step)
77. {
78. glVertex2d(-step / 3, Ymax - row);
79. glVertex2d(0, Ymax - row);
80. }
81. glLineWidth(1);
82. glEnd();
83. }
84. }
85. }

### Код файлу (PointLine.cs)

1. public struct PointLine
2. {
3. public float x;
4. public float y;
5. public PointLine(float x, float y)
6. {
7. this.x = x;
8. this.y = y;
9. }
10. }

### Код файлу (Circle.cs)

1. namespace Lab\_4
2. {
3. public struct Circle
4. {
5. public float r;
6. public Circle(float r)
7. {
8. this.r = r;
9. }
10. }
11. }

### Код файлу (Line.cs)

1. public struct Line
2. {
3. public PointLine point\_1;
4. public PointLine point\_2;
5. public Line(PointLine point\_1, PointLine point\_2)
6. {
7. this.point\_1 = point\_1;
8. this.point\_2 = point\_2;
9. }
10. }

### Код файлу (Elipse.cs)

1. namespace Lab\_4
2. {
3. public struct Hyperbole
4. {
5. public float b;
6. public float a;
7. public Hyperbole(float b, float a)
8. {
9. this.b = b;
10. this.a = a;
11. }
12. }
13. }

### Код файлу (Curve.cs)

1. namespace Lab\_4
2. {
3. public enum Curve
4. {
5. circle,
6. hyperbole
7. }
8. }

# Додаток E. Лістинг програми до практичної роботи №5

### Код файлу (MainForm.cs)

1. namespace Lab\_4
2. {
3. public enum Curve
4. {
5. circle,
6. hyperbole
7. }
8. }
9. using System.Windows.Forms;
10. using static Lab\_5.OpenGL;
11. namespace Lab\_5
12. {
13. public partial class MainForm : Form
14. {
15. public MainForm()
16. {
17. InitializeComponent();
18. }
19. private void segments\_numeric\_ValueChanged(object sender, System.EventArgs e)
20. {
21. renderControl1.SetSegments((int)segments\_numeric.Value);
22. }
23. private void fillMode\_rb\_CheckedChanged(object sender, System.EventArgs e)
24. {
25. renderControl1.SetDrawMode(GLU\_FILL);
26. }
27. private void lineMode\_rb\_CheckedChanged(object sender, System.EventArgs e)
28. {
29. renderControl1.SetDrawMode(GLU\_LINE);
30. }
31. private void orthoMode\_rb\_CheckedChanged(object sender, System.EventArgs e)
32. {
33. renderControl1.SetRenderMode(false);
34. }
35. private void perspectiveMode\_rb\_CheckedChanged(object sender, System.EventArgs e)
36. {
37. renderControl1.SetRenderMode(true);
38. }
39. private void sphereRadius\_numeric\_ValueChanged(object sender, System.EventArgs e)
40. {
41. renderControl1.SetSphereRadius((float)sphereRadius\_numeric.Value);
42. }
43. private void coneRadius\_numeric\_ValueChanged(object sender, System.EventArgs e)
44. {
45. renderControl1.SetConeRadius((float)coneRadius\_numeric.Value);
46. }
47. private void coneHeight\_numeric\_ValueChanged(object sender, System.EventArgs e)
48. {
49. renderControl1.SetConeHeight((float)coneHeight\_numeric.Value);
50. coneSlice\_numeric.Maximum = coneHeight\_numeric.Value - coneSlice\_numeric.Increment;
51. }
52. private void diskInnerRadius\_numeric\_ValueChanged(object sender, System.EventArgs e)
53. {
54. renderControl1.SetDiskInnerRadius((float)diskInnerRadius\_numeric.Value);
55. diskOuterRadius\_numeric.Minimum = diskInnerRadius\_numeric.Value + (decimal)0.1f;
56. }
57. private void diskOuterRadius\_numeric\_ValueChanged(object sender, System.EventArgs e)
58. {
59. renderControl1.SetDiskOuterRadius((float)diskOuterRadius\_numeric.Value);
60. diskInnerRadius\_numeric.Maximum = diskOuterRadius\_numeric.Value - (decimal)0.1f;
61. }
62. private void MainForm\_Load(object sender, System.EventArgs e)
63. {
64. }
65. private void coneSlice\_numeric\_ValueChanged(object sender, System.EventArgs e)
66. {
67. renderControl1.SetConeSlice((float)coneSlice\_numeric.Value);
68. }
69. }
70. }

### Код файлу (RenderControl.cs)

1. using System;
2. using System.Drawing;
3. using System.Windows.Forms;
4. namespace Lab\_5
5. {
6. public partial class RenderControl : OpenGL
7. {
8. private float size;
9. private float AspectRatio { get => (float)Width / Height; }
10. private float Xmin { get => (AspectRatio > 1) ? -size \* AspectRatio : -size; }
11. private float Xmax { get => (AspectRatio > 1) ? +size \* AspectRatio : +size; }
12. private float Ymin { get => (AspectRatio < 1) ? -size / AspectRatio : -size; }
13. private float Ymax { get => (AspectRatio < 1) ? +size / AspectRatio : +size; }
14. private float Zmin { get => -size; }
15. private float Zmax { get => +size; }
16. private float angleX;
17. private float angleY;
18. private float gridStep;
19. private float gridMerge;
20. private float \_heigth;
21. private float \_width;
22. private float segment;
23. private uint gluMode;
24. private float sphereRadius;
25. private float coneHeight;
26. private float coneRadius;
27. private float coneSlice;
28. private float innerRadius;
29. private float outerRadius;
30. private bool isPerspective;
31. private uint idAxis;
32. private IntPtr qObj;
33. public RenderControl()
34. {
35. InitializeComponent();
36. }
37. private void OnStart(object sender, EventArgs e)
38. {
39. size = 10.5f;
40. angleY = 20.0f;
41. angleX = 10.0f;
42. gridStep = 1f;
43. gridMerge = 0.5f;
44. \_heigth = size \* 2;
45. \_width = size \* 2;
46. segment = 15.0f;
47. gluMode = GLU\_FILL;
48. sphereRadius = 1.0f;
49. coneHeight = 1.5f;
50. coneRadius = 2.5f;
51. coneSlice = 1.0f;
52. innerRadius = 1.0f;
53. outerRadius = 3.0f;
54. isPerspective = false;
55. idAxis = glGenLists(3);
56. glNewList(idAxis, GL\_COMPILE);
57. DrawGrid();
58. DrawAxis();
59. DrawTextAxis();
60. glEndList();
61. qObj = gluNewQuadric();
62. }
63. private void OnRender(object sender, EventArgs e)
64. {
65. glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);
66. glLoadIdentity();
67. glViewport(0, 0, Width, Height);
68. // Устанавливаем проекцию
69. SetProjection();
70. if (isPerspective)
71. {
72. // Для перспективной проекции добавляем отступ камеры назад
73. glTranslatef(0, 0, -size \* 3);
74. }
75. glRotated(angleX, 1, 0, 0);
76. glRotated(angleY, 0, 1, 0);
77. // Enable depth
78. glEnable(GL\_DEPTH\_TEST);
79. gluQuadricDrawStyle(qObj, gluMode);
80. // Draw system coorditane
81. glCallList(idAxis);
82. glEnable(GL\_COLOR\_MATERIAL);
83. glEnable(GL\_LIGHTING);
84. glEnable(GL\_LIGHT0);
85. glLightfv(GL\_LIGHT0, GL\_POSITION, new float[] { size, size, size, 0 });
86. //Draw figures
87. DrawSphere();
88. DrawConus();
89. DrawDisk();
90. glDisable(GL\_LIGHTING);
91. }
92. private void OnDestroy(object sender, EventArgs e)
93. {
94. gluDeleteQuadric(qObj);
95. }
96. private bool mouseFlag = false;
97. private Point mouseStart;
98. private void OnMouseDown(object sender, MouseEventArgs e)
99. {
100. mouseFlag = e.Button == MouseButtons.Left;
101. mouseStart = e.Location;
102. }
103. private void OnMouseMove(object sender, MouseEventArgs e)
104. {
105. if (mouseFlag)
106. {
107. Point current = e.Location;
108. angleX += (current.Y - mouseStart.Y) / 2.0f;
109. angleY += (current.X - mouseStart.X) / 2.0f;
110. mouseStart = current;
111. Invalidate();
112. }
113. }
114. private void OnMouseUp(object sender, MouseEventArgs e)
115. {
116. if (mouseFlag)
117. mouseFlag = !(e.Button == MouseButtons.Left);
118. }
119. private void DrawAxis()
120. {
121. glLineWidth(2);
122. glBegin(GL\_LINES);
123. glColor3d(238f / 255f, 223f / 255f, 122f / 255f);
124. // X axis
125. glVertex3d(-size + gridMerge, 0.0f, 0.0f);
126. glVertex3d(size - gridMerge, 0.0f, 0.0f);
127. float \_start = gridStep \* (((-size + gridMerge) / gridStep) % 1);
128. // draw lines
129. for (float colum = \_start; colum < \_width - gridMerge; colum += gridStep)
130. {
131. glVertex3d(-size + gridMerge + colum, 0f, 0f);
132. glVertex3d(-size + gridMerge + colum, -gridStep / 3, 0f);
133. }
134. // Y axis
135. glVertex3d(0.0f, -size + gridMerge, 0.0f);
136. glVertex3d(0.0f, size - gridMerge, 0.0f);
137. \_start = gridStep \* (((size - gridMerge) / gridStep) % 1);
138. // draw lines
139. for (float row = \_start; row < \_heigth - gridMerge; row += gridStep)
140. {
141. glVertex3d(-gridStep / 3, size - gridMerge - row, 0f);
142. glVertex3d(0f, size - gridMerge - row, 0f);
143. }
144. // Z axis
145. glVertex3d(0.0f, 0.0f, -size + gridMerge);
146. glVertex3d(0.0f, 0.0f, size - gridMerge);
147. \_start = gridStep \* (((size - gridMerge) / gridStep) % 1);
148. // draw lines
149. for (float row = \_start; row < \_heigth - gridMerge; row += gridStep)
150. {
151. glVertex3d(-gridStep / 3, 0f, size - gridMerge - row);
152. glVertex3d(0f, 0f, size - gridMerge - row);
153. }
154. glEnd();
155. }
156. private void DrawTextAxis()
157. {
158. // draw text axis
159. DrawText("+X", size, 0, 0);
160. DrawText("+Y", 0, size, 0);
161. DrawText("+Z", 0, 0, size);
162. glColor3d(238f / 255f, 223f / 255f, 122f / 255f);
163. // Z axis
164. float \_start = gridStep \* (((-size) / gridStep) % 1);
165. for (float colum = \_start + gridStep; colum < \_width; colum += gridStep)
166. {
167. if (-size + colum >= 0.1f || -size + colum <= -0.1f)
168. {
169. DrawText((-size + colum).ToString("F1"), 0f, -gridStep / 2, -size + colum);
170. }
171. }
172. // Y axis
173. \_start = gridStep \* ((size / gridStep) % 1);
174. for (float row = \_start; row < \_heigth; row += gridStep)
175. {
176. if (size - row >= 0.1f || size - row <= -0.1f)
177. {
178. DrawText((size - row).ToString("F1"), -gridStep / 2, size - row, 0f);
179. }
180. }
181. // X axis
182. \_start = gridStep \* ((size / gridStep) % 1);
183. for (float row = \_start; row < \_heigth; row += gridStep)
184. {
185. if (size - row >= 0.1f || size - row <= -0.1f)
186. {
187. DrawText((size - row).ToString("F1"), size - row, -gridStep / 2, 0f);
188. }
189. }
190. // 0
191. DrawText((0).ToString(), -gridStep / 2.5f, -gridStep / 2.5f);
192. }
193. private void DrawGrid()
194. {
195. glLineWidth(1);
196. glBegin(GL\_LINES);
197. glColor3d(216f / 255f, 162f / 255f, 94f / 255f);
198. float \_start = gridStep \* ((size / gridStep) % 1) + gridMerge;
199. // draw Y lines
200. for (float row = \_start; row < \_heigth - gridMerge \* 2; row += gridStep)
201. {
202. glVertex3d(0.0f, size - row - gridMerge, size - gridMerge);
203. glVertex3d(0.0f, size - row - gridMerge, -size + gridMerge);
204. }
205. \_start = gridStep \* ((size / gridStep) % 1) + gridMerge;
206. // draw Z lines
207. for (float colum = \_start; colum < \_width - gridMerge \* 2; colum += gridStep)
208. {
209. glVertex3d(0.0f, -size + gridMerge, size - colum - gridMerge);
210. glVertex3d(0.0f, size - gridMerge, size - colum - gridMerge);
211. }
212. glEnd();
213. }
214. private void SetProjection()
215. {
216. // Переходим на матрицу проекции и сбрасываем её
217. glMatrixMode(GL\_PROJECTION);
218. glLoadIdentity();
219. if (isPerspective)
220. {
221. // Настройка перспективной проекции
222. float fieldOfView = 45.0f;
223. float nearClip = 0.1f;
224. float farClip = 100.0f;
225. gluPerspective(fieldOfView, AspectRatio, nearClip, farClip);
226. }
227. else
228. {
229. // Настройка ортогональной проекции
230. glOrtho(Xmin, Xmax, Ymin, Ymax, Zmin, Zmax);
231. }
232. // Возвращаемся к матрице модели
233. glMatrixMode(GL\_MODELVIEW);
234. glLoadIdentity();
235. }
236. private void DrawDisk()
237. {
238. glPushMatrix();
239. float x0 = 4.5f;
240. float y0 = 2.5f;
241. float z0 = 2.5f;
242. glTranslatef(x0, y0, z0);
243. glColor3d(78f / 255f, 81f / 255f, 140f / 255f);
244. gluDisk(qObj, innerRadius, outerRadius, (int)segment, (int)segment);
245. glPopMatrix();
246. }
247. private void DrawConus()
248. {
249. glPushMatrix();
250. float x0 = -3.5f;
251. float y0 = 2.5f;
252. float z0 = -2.0f;
253. float slice = coneHeight - coneSlice;
254. // slice
255. glClipPlane(GL\_CLIP\_PLANE0, new double[] { 0, 0, -(z0 / z0), -(z0 / z0) \* -(z0 + slice) });
256. glEnable(GL\_CLIP\_PLANE0);
257. glTranslatef(x0, y0, z0);
258. glColor3d(255f / 255f, 237f / 255f, 0f / 255f);
259. gluCylinder(qObj, coneRadius, 0.0f, coneHeight, (int)segment, (int)segment);
260. glDisable(GL\_CLIP\_PLANE0);
261. glPopMatrix();
262. }
263. private void DrawSphere()
264. {
265. glPushMatrix();
266. float x0 = 2.0f;
267. float y0 = 2.0f;
268. float z0 = 3.5f;
269. glTranslatef(x0, y0, z0);
270. glColor3d(201f / 255f, 160f / 255f, 220f / 255f);
271. gluSphere(qObj, sphereRadius, (int)segment, (int)segment);
272. glPopMatrix();
273. }
274. public void SetSegments(int \_segment)
275. {
276. segment = \_segment;
277. Invalidate();
278. }
279. public void SetDrawMode(uint \_gluMode)
280. {
281. gluMode = \_gluMode;
282. Invalidate();
283. }
284. public void SetRenderMode(bool \_isPerspective)
285. {
286. isPerspective = \_isPerspective;
287. Invalidate();
288. }
289. public void SetSphereRadius(float \_radius)
290. {
291. sphereRadius = \_radius;
292. Invalidate();
293. }
294. public void SetConeRadius(float \_radius)
295. {
296. coneRadius = \_radius;
297. Invalidate();
298. }
299. public void SetConeHeight(float \_height)
300. {
301. coneHeight = \_height;
302. Invalidate();
303. }
304. public void SetConeSlice(float \_slice)
305. {
306. coneSlice = \_slice;
307. Invalidate();
308. }
309. public void SetDiskInnerRadius(float \_radius)
310. {
311. innerRadius = \_radius;
312. Invalidate();
313. }
314. public void SetDiskOuterRadius(float \_radius)
315. {
316. outerRadius = \_radius;
317. Invalidate();
318. }
319. }
320. }

# Додаток F. Лістинг програми до практичної роботи №6

### Код файлу (MainForm.cs)

1. public partial class MainForm : Form
2. {
3. public MainForm()
4. {
5. InitializeComponent();
6. }
7. }

### Код файлу (RenderControl.cs)

1. using System;
2. using System.Drawing;
3. using System.Windows.Forms;
4. namespace OpenGL\_Lab\_6
5. {
6. public partial class RenderControl : OpenGL
7. {
8. private float size = 1.5f;
9. private float AspectRatio { get => (float)Width / Height; }
10. private float Xmin { get => (AspectRatio > 1) ? -size \* AspectRatio : -size; }
11. private float Xmax { get => (AspectRatio > 1) ? +size \* AspectRatio : +size; }
12. private float Ymin { get => (AspectRatio < 1) ? -size / AspectRatio : -size; }
13. private float Ymax { get => (AspectRatio < 1) ? +size / AspectRatio : +size; }
14. private float Zmin { get => -size; }
15. private float Zmax { get => +size; }
16. private float angleX;
17. private float angleY;
18. private float m;
19. private float a = 0.3f;
20. private float b = 0.5f;
22. private float aw;
23. private float theta
24. {
25. get { return 180f / MathF.PI \* MathF.Acos((a \* a + S \* S - b \* b) / (2 \* a \* S)); }
26. }
27. private float psi
28. {
29. get { return 180f / MathF.PI \* MathF.Acos((a\*a + b\*b - S\*S) / (2 \* a \* b)); }
30. }
31. private float phi;
32. private float s = 0.2f;
33. private float S
34. {
35. get { return s; }
36. set { if ((value > (b - a)) && (value < (2 \* a))) s = value; }
37. }
38. public RenderControl()
39. {
40. InitializeComponent();
41. MouseWheel += OnMouseWheel;
42. }
43. private void OnStart(object sender, EventArgs e)
44. {
45. angleX = 0f;
46. angleY = 0f;
47. m = 1f;
48. S = b;
49. phi = 15.0f;
50. aw = 0f;
51. }
52. private void OnRender(object sender, EventArgs e)
53. {
54. glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);
55. glLoadIdentity();
56. glViewport(0, 0, Width, Height);
57. glOrtho(Xmin, Xmax, Ymin, Ymax, Zmin, Zmax);
58. glRotated(angleX, 1, 0, 0);
59. glRotated(angleY, 0, 1, 0);
60. glScaled(m, m, m);
61. // Enable depth
62. glEnable(GL\_DEPTH\_TEST);
63. DrawAxis();
64. DrawGridXY();
65. DrawGridYZ();
66. DrawGridZX();
67. glRotated(aw, 0f, -1f, 0f);
68. // first line
69. glRotated(phi, 0f, 0f, -1f);
70. Segment(2 \* a, 0.05f, 1f, 0f, 0f);
71. // second line
72. glPushMatrix();
73. glTranslated(0f, a, 0f);
74. glRotated(-theta, 0f, 0f, -1f);
75. Segment(2 \* a, 0.04f, 0f, 1f, 0f);
76. glPopMatrix();
77. // third line
78. glTranslated(0f, 2\*a, 0f);
79. glRotated(psi, 0f, 0f, -1f);
80. Segment(-b, 0.05f, 0f, 0f, 1f);
81. Segment(1.5f \* b, 0.05f, 0f, 0f, 1f);
82. }
83. private void Segment(float height, float width, float r, float g, float b)
84. {
85. glColor3d(r, g, b);
86. glLineWidth(5);
87. glBegin(GL\_LINES);
88. glVertex3d(0f, 0f, -width);
89. glVertex3d(0f, height, -width);
90. glVertex3d(0f, 0f, width);
91. glVertex3d(0f, height, width);
92. glVertex3d(0f, 0f, -width);
93. glVertex3d(0f, 0f, width);
94. glVertex3d(0f, height, -width);
95. glVertex3d(0f, height, width);
96. glEnd();
97. glLineWidth(1);
98. }
99. private float padding = 0.1f;
100. private void DrawAxis()
101. {
102. glColor3d(238f / 255f, 223f / 255f, 122f / 255f);
103. glLineWidth(1);
104. glBegin(GL\_LINES);
105. glVertex3d(-padding, 0f, 0f);
106. glVertex3d(1f, 0f, 0f);
107. glVertex3d(0f, -padding, 0f);
108. glVertex3d(0f, 1f, 0f);
109. glVertex3d(0f, 0f, -padding);
110. glVertex3d(0f, 0f, 1f);
111. glEnd();
112. DrawText("X", 1f, 0f, 0f);
113. DrawText("Y", 0f, 1f, 0f);
114. DrawText("Z", 0f, 0f, 1f);
115. }
116. private void DrawGridXY()
117. {
118. glColor3d(216f / 255f, 162f / 255f, 94f / 255f);
119. glLineStipple(3, 21845);
120. glEnable(GL\_LINE\_STIPPLE);
121. glLineWidth(1);
122. glBegin(GL\_LINES);
123. for (float value = 0; value <= 1f + padding; value += 0.1f)
124. {
125. glVertex3d(value, 0f, -padding);
126. glVertex3d(value, 1f, -padding);
127. glVertex3d(0f, value, -padding);
128. glVertex3d(1f, value, -padding);
129. }
130. glEnd();
131. glDisable(GL\_LINE\_STIPPLE);
132. }
133. private void DrawGridYZ()
134. {
135. glColor3d(216f / 255f, 162f / 255f, 94f / 255f);
136. glLineStipple(3, 21845);
137. glEnable(GL\_LINE\_STIPPLE);
138. glLineWidth(1);
139. glBegin(GL\_LINES);
140. for (float value = 0; value <= 1f + padding; value += 0.1f)
141. {
142. glVertex3d(-padding, value, 0f);
143. glVertex3d(-padding, value, 1f);
144. glVertex3d(-padding, 0f, value);
145. glVertex3d(-padding, 1f, value);
146. }
147. glEnd();
148. glDisable(GL\_LINE\_STIPPLE);
149. }
150. private void DrawGridZX()
151. {
152. glColor3d(216f / 255f, 162f / 255f, 94f / 255f);
153. glLineStipple(3, 21845);
154. glEnable(GL\_LINE\_STIPPLE);
155. glLineWidth(1);
156. glBegin(GL\_LINES);
157. for (float value = 0; value <= 1f + padding; value += 0.1f)
158. {
159. glVertex3d(value, -padding, 0f);
160. glVertex3d(value, -padding, 1f);
161. glVertex3d(0f, -padding, value);
162. glVertex3d(1f, -padding, value);
163. }
164. glEnd();
165. glDisable(GL\_LINE\_STIPPLE);
166. }
167. private void OnKeyDown(object sender, KeyEventArgs e)
168. {
169. switch (e.KeyCode)
170. {
171. case Keys.W:
172. phi--;
173. break;
174. case Keys.S:
175. phi++;
176. break;
177. case Keys.E:
178. S += 0.01f;
179. break;
180. case Keys.Q:
181. S -= 0.01f;
182. break;
183. case Keys.A:
184. aw++;
185. break;
186. case Keys.D:
187. aw--;
188. break;
189. }
190. Invalidate();
191. }
192. private bool mouseFlag = false;
193. private Point mouseStart;
194. private void OnMouseDown(object sender, System.Windows.Forms.MouseEventArgs e)
195. {
196. mouseFlag = e.Button == MouseButtons.Left;
197. mouseStart = e.Location;
198. }
199. private void OnMouseUp(object sender, System.Windows.Forms.MouseEventArgs e)
200. {
201. if (mouseFlag)
202. mouseFlag = !(e.Button == MouseButtons.Left);
203. }
204. private void OnMouseMove(object sender, System.Windows.Forms.MouseEventArgs e)
205. {
206. if (mouseFlag)
207. {
208. Point current = e.Location;
209. angleX += (current.Y - mouseStart.Y) / 2.0f;
210. angleY += (current.X - mouseStart.X) / 2.0f;
211. mouseStart = current;
212. Invalidate();
213. }
214. }
215. private void OnMouseWheel(object sender, MouseEventArgs e)
216. {
217. m += e.Delta / 2000.0f;
218. Invalidate();
219. }
220. }
221. }

# Додаток G. Лістинг програми до практичної роботи №7

### Код файлу (RenderControl.cs)

1. using Microsoft.Win32;
2. using System;
3. using System.Drawing;
4. namespace OpenGLLab7
5. {
6. public partial class RenderControl : OpenGL
7. {
8. public RenderControl()
9. {
10. InitializeComponent();
11. }
12. float dy = 0.1f;
13. float size = 30f;
14. Color heartColor = Color.Crimson;
15. private void OnContextCreated(object sender, EventArgs e)
16. {
17. // todo: 003 Загрузить настройки программы-заставки для основного режима работы
18. using (RegistryKey key = Registry.CurrentUser.OpenSubKey(@"Software\MyScreenSaver"))
19. {
20. if (key != null)
21. {
22. if (key.GetValue("HeartColor") != null)
23. heartColor = Color.FromArgb((int)key.GetValue("HeartColor"));
24. if (key.GetValue("BackgroundColor") != null)
25. glClearColor(Color.FromArgb((int)key.GetValue("BackgroundColor")));
26. if (key.GetValue("Heartbeat") != null)
27. dy = (float)Convert.ToDecimal(key.GetValue("Heartbeat"));
28. if (key.GetValue("HeartSize") != null)
29. size = (float)Convert.ToDecimal(key.GetValue("HeartSize"));
30. }
31. }
32. }
33. float yleft = -1;
34. private void OnRender(object sender, EventArgs e)
35. {
36. // todo: 004 Формирование изображения экранной заставки
37. // ...
38. glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);
39. glLoadIdentity();
40. // Устанавливаем квадратную область просмотра, сохраняя пропорции
41. if (Width > Height)
42. {
43. // Если ширина больше высоты, центрируем по горизонтали
44. glViewport((Width - Height) / 2, 0, Height, Height);
45. }
46. else
47. {
48. // Если высота больше ширины, центрируем по вертикали
49. glViewport(0, (Height - Width) / 2, Width, Width);
50. }
51. glOrtho(-1, 1, -1, 1, -1, 1);
52. glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_FILL);
53. glColor3d(heartColor.R, heartColor.G, heartColor.B);
54. glBegin(GL\_POLYGON);
56. float step = 0.01f;
57. for (float t = 0; t < 2 \* MathF.PI; t += step)
58. {
59. float x1 = 16 \* MathF.Pow(MathF.Sin(t), 3) / size;
60. float y1 = (13 \* MathF.Cos(t) - 5 \* MathF.Cos(2 \* t) - 2 \* MathF.Cos(3 \* t) - MathF.Cos(4 \* t)) / size;
61. float x2 = 16 \* MathF.Pow(MathF.Sin(t + step), 3) / size;
62. float y2 = (13 \* MathF.Cos(t + step) - 5 \* MathF.Cos(2 \* (t + step)) - 2 \* MathF.Cos(3 \* (t + step)) - MathF.Cos(4 \* (t + step))) / size;
63. glVertex2d(x1, y1);
64. glVertex2d(x2, y2);
65. }
66. glEnd();
67. // todo: 005 Изменение параметра анимации/номера кадра
68. // ...
69. if ((yleft >= 1) || (yleft <= -1)) dy = -dy;
70. yleft -= dy;
71. size += dy;
72. }
73. }
74. }

### Код файлу (SettingsForm.cs)

1. using Microsoft.Win32;
2. using System;
3. using System.Diagnostics;
4. using System.Drawing;
5. using System.Windows.Forms;
6. namespace OpenGLLab7
7. {
8. public partial class SettingsForm : Form
9. {
10. public SettingsForm()
11. {
12. InitializeComponent();
13. }
14. private void SettingsFormLoad(object sender, EventArgs e)
15. {
16. // todo: 002 Загрузить настройки программы-заставки в режиме конфигурации
17. using (RegistryKey key = Registry.CurrentUser.OpenSubKey(@"Software\MyScreenSaver"))
18. {
19. if (key != null)
20. {
21. if (key.GetValue("HeartColor") != null)
22. colorHeart\_button.BackColor = Color.FromArgb((int)key.GetValue("HeartColor"));
23. if (key.GetValue("BackgroundColor") != null)
24. bgColor\_button.BackColor = Color.FromArgb((int)key.GetValue("BackgroundColor"));
25. if (key.GetValue("Heartbeat") != null)
26. heartbeat\_numeric.Value = Convert.ToDecimal(key.GetValue("Heartbeat"));
27. if(key.GetValue("HeartSize") != null)
28. sizeHeart\_numeric.Value = Convert.ToDecimal(key.GetValue("HeartSize"));
29. }
30. }
31. Debug.WriteLine("Load screen saver's settings.");
32. }
33. private void SettingsFormClosed(object sender, FormClosedEventArgs e)
34. {
35. // todo: 001 Сохранить настройки программы-заставки в режиме конфигурации
36. using (RegistryKey key = Registry.CurrentUser.CreateSubKey(@"Software\MyScreenSaver"))
37. {
38. if (key != null)
39. {
40. key.SetValue("HeartColor", colorHeart\_button.BackColor.ToArgb());
41. key.SetValue("Heartbeat", (float)heartbeat\_numeric.Value);
42. key.SetValue("HeartSize", (float)sizeHeart\_numeric.Value);
43. key.SetValue("BackgroundColor", bgColor\_button.BackColor.ToArgb());
44. }
45. }
46. Debug.WriteLine("Save screen saver's settings.");
47. }
48. private void colorHeart\_button\_Click(object sender, EventArgs e)
49. {
50. // Создаем экземпляр диалога выбора цвета
51. using (ColorDialog colorDialog = new ColorDialog())
52. {
53. // Настраиваем диалог, если нужно (например, начальный цвет)
54. colorDialog.FullOpen = true; // Показывает всю палитру
55. colorDialog.Color = this.BackColor; // Начальный цвет
56. // Показываем диалог и проверяем, выбрал ли пользователь цвет
57. if (colorDialog.ShowDialog() == DialogResult.OK)
58. {
59. // Устанавливаем выбранный цвет, например, как цвет фона формы
60. colorHeart\_button.BackColor = colorDialog.Color;
61. }
62. }
63. }
64. private void bgColor\_button\_Click(object sender, EventArgs e)
65. {
66. // Создаем экземпляр диалога выбора цвета
67. using (ColorDialog colorDialog = new ColorDialog())
68. {
69. // Настраиваем диалог, если нужно (например, начальный цвет)
70. colorDialog.FullOpen = true; // Показывает всю палитру
71. colorDialog.Color = this.BackColor; // Начальный цвет
72. // Показываем диалог и проверяем, выбрал ли пользователь цвет
73. if (colorDialog.ShowDialog() == DialogResult.OK)
74. {
75. // Устанавливаем выбранный цвет, например, как цвет фона формы
76. bgColor\_button.BackColor = colorDialog.Color;
77. }
78. }
79. }
80. }
81. }