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  *632п*

напряму підготовки (спеціальності):

*121 інженерія програмного забезпечення*

(шифр і назва напряму підготовки / спеціальності)

*Ільченко О.О.*

(прізвище й ініціали студента)

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

(посада, науковий ступінь, прізвище й ініціали)

Національна шкала:

Кількість балів:

Оцінка ECTS:

Зміст

[Практична робота 1. Основні принципи роботи з OpenGL 3](#_Toc186206956)

[Завдання, варіант № 12 3](#_Toc186206957)

[Системна інформація 3](#_Toc186206958)

[Теоретичні відомості 3](#_Toc186206959)

[Результати виконання практичної роботи 5](#_Toc186206960)

[**Практична робота 2. ОСНОВНІ ПРИНЦИПИ РОБОТИ З OPENGL** 8](#_Toc186206961)

[**Завдання, варіант № 12** 8](#_Toc186206962)

[**Теоретичні відомості** 8](#_Toc186206967)

[**Результати виконання практичної роботи** 9](#_Toc186206968)

[**Практична робота 3. Назва роботи** 13](#_Toc186206969)

[**Завдання, варіант № 12** 13](#_Toc186206970)

[**Теоретичні відомості** 13](#_Toc186206971)

[**Результати виконання практичної роботи** 14](#_Toc186206972)

[Загальний перелік посилань 17](#_Toc186206973)

[Додаток А. Лістинг програми до практичної роботи №1 18](#_Toc186206974)

[Додаток Б. Лістинг програми до практичної роботи №2 22](#_Toc186206975)

[Додаток Б. Лістинг програми до практичної роботи №3 22](#_Toc186206975)

# Практична робота 1. Основні принципи роботи з OpenGL

## Завдання, варіант № 12

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.

|  |  |  |
| --- | --- | --- |
|  | Примітиви:  GL\_POINTS, GL\_LINE\_LOOP  x1 = -3.5; x2 = 1  y1 = -0.5; y2 = 1.5 |  |

## Системна інформація

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

Processor Intel(R) Core(TM) i7-3770K CPU @ 3.50GHz 3.50 GHz

RAM 16.0 GB

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

Edition Windows 10 Pro Version 22H2

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

## Теоретичні відомості

### Вершинні масиви

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

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

glVertexPointer(): вказати покажчик на масив вершинних координат

glNormalPointer(): вказати покажчик на звичайний масив

glColorPointer(): вказати вказівник на масив кольорів RGB

glIndexPointer(): вказати покажчик на індексований кольоровий масив

glTexCoordPointer(): вказати покажчик на масив текстурних шнурів

glEdgeFlagPointer(): вказати вказівник на масив прапорців edge

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

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

### Команда glDrawArrays()

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

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

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

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

...

// activate and specify pointer to vertex array

glEnableClientState(GL\_VERTEX\_ARRAY);

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

// draw a cube

glDrawArrays(GL\_TRIANGLES, 0, 36);

// deactivate vertex arrays after drawing

glDisableClientState(GL\_VERTEX\_ARRAY);

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

### Команда glDrawElements()

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

…

## Результати виконання практичної роботи

### Розв'язання завдання

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

колір, glColor3d() рядок 12 у файлі Figure.cs;

тип, glLineStipple(), glEnable()/glDisable(), рядок 85 у файлі RenderControl.cs;

товщина glLineWidth(), рядок 45 у файлі RenderControl.cs;

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

Розроблення підпрограм для виключення дублювання коду наведено у рядках 102 – 110 файлу RenderControl.cs

Застосування циклів для створення зображень наведено у рядках 65 – 74 файлу RenderControl.cs

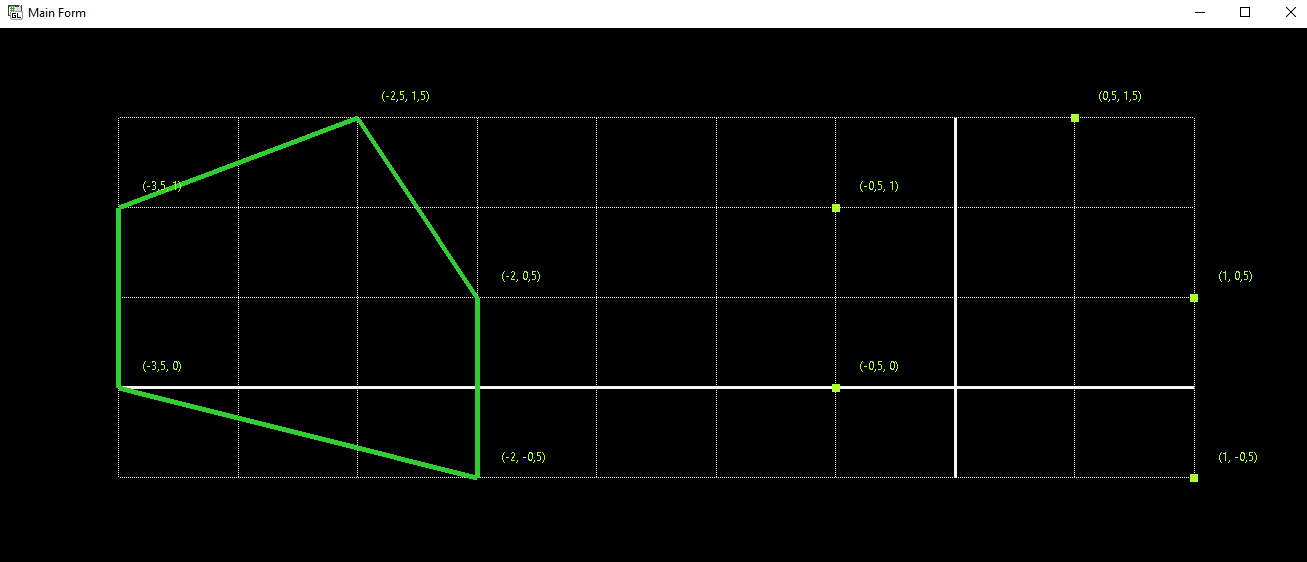


Рисунок 1.1 – Тестування програми при зміні ширини вікна



Рисунок 1.2 – Тестування програми при зміні висоти вікна

### Контроль виконання вимог та елементів завдання

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

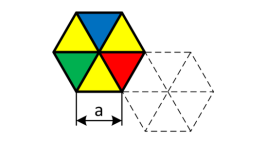
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблиця 1.1 | | | | |
| № з/п | Складність | Вимоги | Бали | Зроблено |
| 1 | Базовий рівень | Використання команд управління параметрами графічних примітивів (колір, тип, товщина) | 2 | **+** |
| 2 | Коректне відображення завдання під час змінення розмірів/положення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

**Практична робота 2.  
ОСНОВНІ ПРИНЦИПИ РОБОТИ З OPENGL**

**Завдання, варіант № 12**

**Сторона фігури** a = 100

**Примітив:**GL\_TRIANGLES

****

За допомогою інструментальних засобів, зазначених викладачем, створити простий програмний проєкт із підтримкою бібліотеки OpenGL. Враховуючи систему оцінки (табл. 1.1), розробити програму із застосуванням команд OpenGL, яка встановлює анізотропну систему координат, створює та виводить варіант зображення (табл. 1.2) 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.

**Теоретичні відомості**

**1. OpenGL та його Контекст**

OpenGL (Open Graphics Library) — це кросплатформенний API для рендерингу 2D та 3D графіки. Контекст OpenGL — це середовище, в якому виконуються всі команди OpenGL, і воно включає всі змінні стану OpenGL.

**2. Буфери OpenGL**

* **Color Buffer**: Зберігає інформацію про кольори пікселів на екрані.
* **Depth Buffer**: Зберігає інформацію про глибину кожного пікселя, що допомагає визначати, який об'єкт розташований ближче до камери.
* **Stencil Buffer**: Використовується для маскування окремих пікселів під час рендерингу.

**3. Шейдери**

Шейдери — це маленькі програми, що виконуються на GPU. Основні типи шейдерів:

* **Vertex Shader**: Обробляє вершини графічних примітивів, визначаючи їх позиції у просторі.
* **Fragment Shader**: Обробляє фрагменти (пікселі) графічних примітивів, визначаючи їх колір та інші властивості.

**4. Графічні Примітиви**

Основні графічні примітиви в OpenGL включають точки, лінії та трикутники. Ваш код використовує трикутники для малювання шестикутників.

**5. Текстури**

Текстури — це зображення, які застосовуються до поверхонь графічних об'єктів для додавання деталей.

**Розбір Коду**

**1. Клас MainForm**

Цей клас відповідає за роботу з інтерфейсом користувача. Він визначає поведінку кнопок і перемикачів, які впливають на рендеринг.

* **fillButton\_CheckedChanged**: Встановлює режим заповнення для рендерингу.
* **lineButton\_CheckedChanged**: Встановлює режим лінійного рендерингу.
* **pointButton\_CheckedChanged**: Встановлює режим точкового рендерингу.
* **numericUpDown1\_ValueChanged та numericUpDown2\_ValueChanged**: Оновлюють кількість шестикутників та рядків відповідно.

**2. Клас RenderControl**

Цей клас відповідає за рендеринг графіки з використанням OpenGL.

* **OnRender**: Метод, що викликається для оновлення зображення. Він очищає буфер, встановлює координати та викликає методи для малювання примітивів.
* **DrawPrimitives**: Метод, який малює шестикутники з використанням трикутників. Він обчислює вершини шестикутника, встановлює кольори для кожного трикутника та малює їх.

**Результати виконання практичної роботи**

***Розв'язання завдання***

Перерахування вимог базового та підвищеного рівня складності з таблиці 2.1 з копіями екрана, які демонструють виконання поставленого завдання та/або з посиланнями на додатки та рядки коду, в яких ці вимоги реалізовані.

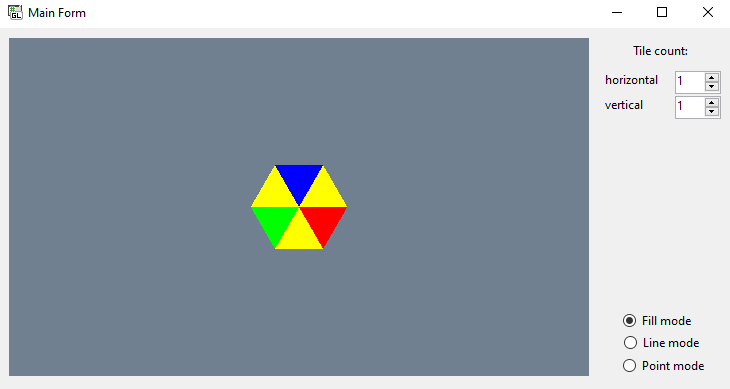


Рисунок 2.1 – Запуск застосунка

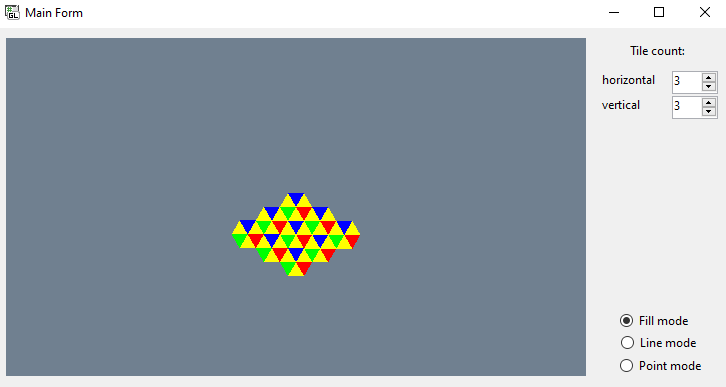


Рисунок 2.2 – Багаторазове замощення плиткою

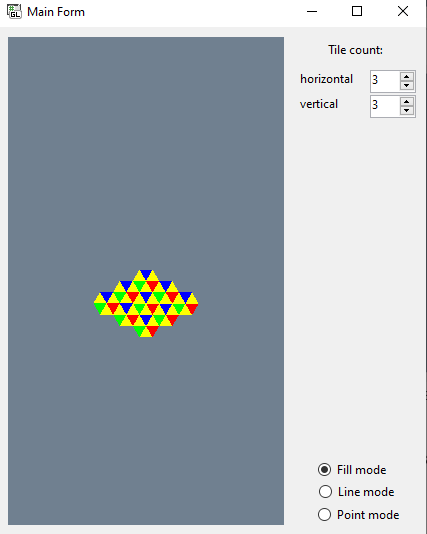


Рисунок 2.3 – Коректне відображення завдання під час зміни розмірів вікна

***Контроль виконання вимог та елементів завдання***

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

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

**Практична робота 3.  
Назва роботи**

**Завдання, варіант № 12**





**Теоретичні відомості**

**Вступ**

Графік функції однієї змінної є важливим інструментом для аналізу математичних залежностей між змінними. У цій лабораторній роботі ми розглянемо основні поняття та принципи перетворення координат для побудови двомірного графіка, використовуючи мову програмування C# та бібліотеку OpenGL. Цей підхід дозволяє створювати візуально привабливі та точні графіки, які допомагають краще зрозуміти властивості функцій.

**Основні Поняття**

Функція однієї змінної – це математична залежність між двома змінними, виражена у формі y=f(x)y = f(x), де yy представляє значення функції при заданому значенні xx. Для візуалізації функцій використовують декартову систему координат (x, y), де точки на площині представляються парами чисел.

Перетворення координат – це процес зміни координат точки для її візуалізації на екрані. Це включає такі операції, як масштабування, зсув, та поворот. OpenGL – це потужна бібліотека для роботи з графікою, яка надає інтерфейси для візуалізації тривимірних та двомірних об'єктів.

**Етапи Побудови Графіка**

Ініціалізація OpenGL включає створення вікна для відображення графіки та налаштування контексту OpenGL. Підготовка даних починається з визначення функції, яку потрібно побудувати, наприклад, y=sin(x)y = sin(x). Також необхідно визначити діапазон значень для xx і відповідних значень yy.

Перетворення координат включає масштабування, тобто зміну розміру об'єктів, та зсув, що полягає у переміщенні об'єктів. Візуалізація полягає у відображенні точок або ліній для представлення графіка функції та оновленні вікна для візуалізації при зміні даних.

**Приклади Функцій**

До базового рівня належать лінійні функції y=mx+by = mx + b та квадратичні функції y=ax2+bx+cy = ax^2 + bx + c. Розширений рівень включає тригонометричні функції, такі як y=sin(x)y = sin(x), y=cos(x)y = cos(x), та експоненціальні функції, наприклад, y=exy = e^x.

**Методика Реалізації**

Масштабування координат полягає у перетворенні значень координат для їх відображення у вікні. Використання анізотропних систем дозволяє коректно відображати графіки. Обробка розривів включає визначення та обробку точок розриву для коректного відображення неперервних функцій. Об'єктно-орієнтоване програмування (ООП) передбачає використання класів та об'єктів для організації коду, а також реалізацію функцій у вигляді методів класів для зручності та повторного використання.

**Результати виконання практичної роботи**

***Розв'язання завдання***

Для корректного відображення осей координат на заданих інтервалах в файлі RenderControl.cs на лініях 75–83 Метод DrawMainAxes() використовує команди OpenGL для відображення осей x та y, а також DrawFunction(FunctionF1, 0, 0, 255) всередині OnRender: Рядок 142.

Для Автоматичного обчислення в файлі RenderControl.cs на лініях Метод CalculateFunctionInfo(Func<float, float> func): Рядки 49-66.

Автоматично оновлює y\_min та y\_max, якщо не ввімкнено ручне налаштування.

Логіка оновлення y\_min та y\_max у рядках 57-59.

Для Обчислення і виведення на екран точок в файлі RenderControl.cs на лініях Логіка пошуку кореня:

Всередині CalculateFunctionInfo(Func<float, float> func): Рядки 61-64.

Використовує умову prev\_y \* y <= 0 для знаходження нулів (x) і зберігає їх в point\_zeros.

Виведення точок на екран:

Метод DrawZeroPoints(): Рядки 98-106.

Відображає обчислені нулі на графіку за допомогою помаранчевих точок.

Коректне виведення графіка f₂(x) (без хибного виводу точок розриву як точок перетину з віссю абсцис) і з відображенням ліній розриву функції наведено на рис 3.1, а також Метод FunctionF2(float x): Рядки 70-73.

Гарантує, що точки розриву не будуть побудовані, повертаючи float.NaN для неприпустимих значень.

Малювання (x): DrawFunction(FunctionF2, 50, 205, 50) всередині OnRender: Рядок 144.

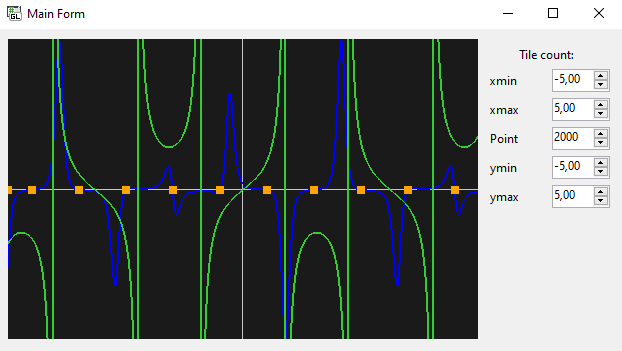


Рисунок 3.1 – Коректне виведення графіків

***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 | | | | |
| № п/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 заданому 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 № 12**

Окружність: параметричне подання

Гіпербола: явне подання

**Теоретичні відомості**

**Методика Реалізаці**

**Результати виконання практичної роботи**

***Розв'язання завдання***

Для Установлення ізотропної системи координат для вікна з змінюваними розмірами було встановлено ізотропну систему координат, яка дозволяє змінювати розміри вікна та зберігати правильні пропорції при зміні розміру вікна всередині RenderControl.cs.

Для Виведення кривих другого порядку відповідно до варіанту завдання було реалізовано виведення кривих другого порядку, таких як парабола або гіпербола, залежно від вибору користувача всередині RenderControl.cs

Для Виведення відрізка та обчислення його точок перетину з кривою другого порядку відповідно до варіанту було додано можливість виведення відрізка та обчислення точок його перетину з кривою другого порядку всередині RenderControl.cs.

Для Вказання положення точок, що формуютьвідрізок, в області графічного виведенняпрограми за допомогою маніпулятора «миш» було реалізовано вказування положення точок відрізка за допомогою миші в графічному виведенні всередині MainForm.cs

Для Використання ООП було застосовано об'єктно-орієнтований підхід для створення класів для фігур, обробки введення та виведення графіки. всередині RenderControl.cs, Figures.cs

Корректна роботи програми наведена на рисунках 4.1-4.2

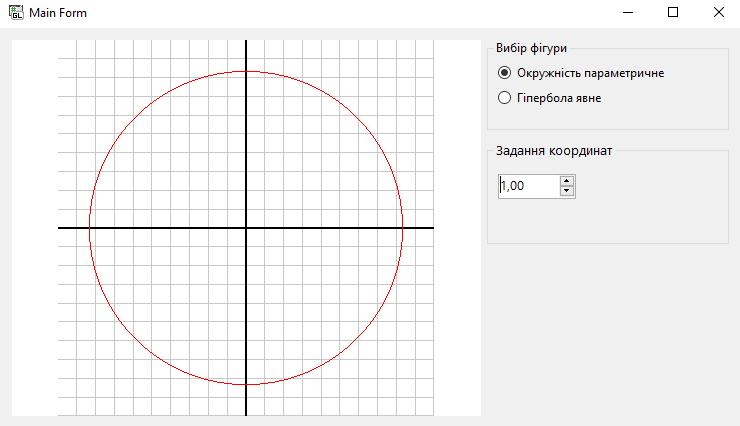


Рисунок 4.1 – Коректне виведення кола

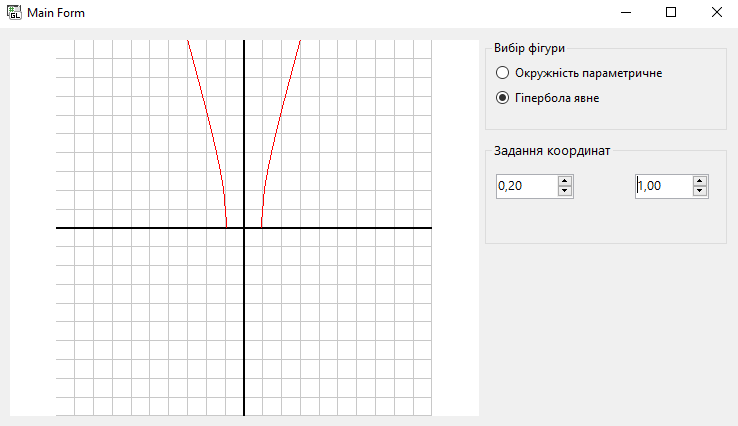


Рисунок 4.2 – Коректне виведення гіперболи

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

# 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. Офіційний вебсайт OpenGL: https://www.opengl.org
3. Документація OpenGL https://www.khronos.org/registry/OpenGL/
4. "OpenGL Programming Guide" (Red Book) — класичний підручник з OpenGL.
5. "OpenGL Shading Language" (Orange Book) — підручник, присвячений шейдерам та програмуванню графіки на рівні шейдерів.

# Додаток А. Лістинг програми до практичної роботи №1

### Код файлу (MainForm.cs)

using System.Windows.Forms;

using static Task\_1.OpenGL;

namespace Task\_1

{

public partial class MainForm : Form

{

public MainForm()

{

InitializeComponent();

}

private void MainForm\_Load(object sender, System.EventArgs e)

{

}

}

}

### Код файлу (Program.cs)

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Task\_1

{

internal static class Program

{

static Program() => DesignMode = true;

public static bool DesignMode { get; set; }

/// <summary>

/// The main entry point for the application.

/// </summary>

[STAThread]

static void Main()

{

DesignMode = false;

Application.SetHighDpiMode(HighDpiMode.SystemAware);

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new MainForm());

}

}

}

### Код файлу (RenderControl.cs)

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Diagnostics;

using System.Drawing;

using System.Drawing.Design;

using System.Linq;

using System.Runtime.InteropServices;

using static Task\_1.OpenGL;

namespace Task\_1

{

public partial class RenderControl : OpenGL

{

public RenderControl()

{

InitializeComponent();

Render += OnRender; // Підключаємо метод рендерингу до події Render

}

private void OnRender(object sender, EventArgs e)

{

float x1 = -4f;

float x2 = 1.5f;

float y1 = -1f;

float y2 = 2f;

// Очищення екрану

glClear(GL\_COLOR\_BUFFER\_BIT);

glClearColor(0f, 0f, 0f, 1.0f); // Midnight Blue background

// Встановлення системи координат

glLoadIdentity();

glViewport(0, 0, Width, Height);

glOrtho(x1, x2, y1, y2, -10, 10); // Межі системи координат згідно із завданням (+0.5 щоб було видно координати)

// Малювання елементів

DrawAxes();

DrawGrid();

DrawFigure();

DrawPoints();

}

private void DrawFigure()

{

glLineWidth(5); // Товщина ліній

glBegin(GL\_LINE\_LOOP);

glColor(Color.LimeGreen); // Синій контур

// Вершини фігури

/\*

glVertex2d(-3.5, 0);

glVertex2d(-3.5, 1);

glVertex2d(-2.5, 1.5);

glVertex2d(-2, 0.5);

glVertex2d(-2, -0.5);

\*/

double[][] points = new double[][] {

new double[] { -3.5, 0 },

new double[] { -3.5, 1 },

new double[] { -2.5, 1.5 },

new double[] { -2, 0.5 },

new double[] { -2, -0.5 },

};

foreach (var vertex in points) {

glVertex2d(vertex[0], vertex[1]);

}

glEnd(); // Завершуємо малювання

}

private void DrawPoints()

{

glPointSize(8); // Розмір точок

glBegin(GL\_POINTS);

glColor(Color.GreenYellow);

// Координати точок

double[][] points = new double[][]

{

new double[] {-3.5, 0},

new double[] {-3.5, 1 },

new double[] { -2.5, 1.5 },

new double[] {-2, 0.5 },

new double[] { -2, -0.5 },

};

// Малюємо точки

foreach (var point in points)

{

glVertex2d(point[0] + 3, point[1]);

}

glEnd();

// Підписуємо координати

foreach (var point in points)

{

string label = $"({point[0]}, {point[1]})";

DrawText(label, point[0] + 0.1, point[1] + 0.1);

string label2 = $"({point[0] + 3}, {point[1]})";

DrawText(label2, point[0] + 3 + 0.1, point[1] + 0.1);

}

}

private void DrawAxes()

{

float x1 = -3.5f;

float x2 = 1f;

float y1 = -0.5f;

float y2 = 1.5f;

glLineWidth(3);

glBegin(GL\_LINES);

glColor(Color.WhiteSmoke);

// Вісь X

glVertex2d(x1, 0.0);

glVertex2d(x2, 0.0);

// Вісь Y

glVertex2d(0.0, y1);

glVertex2d(0.0, y2);

glEnd();

}

private void DrawGrid()

{

float x1 = -3.5f;

float x2 = 1f;

float y1 = -0.5f;

float y2 = 1.5f;

glLineWidth(1);

glEnable(GL\_LINE\_STIPPLE);

glLineStipple(1, 0xAAAA);

glBegin(GL\_LINES);

glColor(Color.White);

// Горизонтальні лінії

for (double y = y1; y <= y2; y += 0.5)

{

glVertex2d(x1, y);

glVertex2d(x2, y);

}

// Вертикальні лінії

for (double x = x1; x <= x2; x += 0.5)

{

glVertex2d(x, y1);

glVertex2d(x, y2);

}

glEnd();

glDisable(GL\_LINE\_STIPPLE);

}

}

}

# Додаток Б. Лістинг програми до практичної роботи №2

### Код файлу (MainForm.cs)

using System;

using System.Windows.Forms;

namespace Task\_2

{

public partial class MainForm : Form

{

public MainForm()

{

InitializeComponent();

}

private void MainForm\_Load(object sender, EventArgs e)

{

fillButton.Checked = true;

}

private void fillButton\_CheckedChanged(object sender, EventArgs e)

{

if (fillButton.Checked)

{

// Changed variable names for better clarity

renderControl1.FillMode = true;

renderControl1.LineMode = false;

renderControl1.PointMode = false;

renderControl1.Invalidate();

}

}

private void lineButton\_CheckedChanged(object sender, EventArgs e)

{

if (lineButton.Checked)

{

renderControl1.FillMode = false;

renderControl1.LineMode = true;

renderControl1.PointMode = false;

renderControl1.Invalidate();

}

}

private void pointButton\_CheckedChanged(object sender, EventArgs e)

{

if (pointButton.Checked)

{

renderControl1.FillMode = false;

renderControl1.LineMode = false;

renderControl1.PointMode = true;

renderControl1.Invalidate();

}

}

private void numericUpDown1\_ValueChanged(object sender, EventArgs e)

{

// Improved variable names for readability

renderControl1.HexagonCount = Convert.ToInt32(((NumericUpDown)sender).Value);

renderControl1.Invalidate();

}

private void numericUpDown2\_ValueChanged(object sender, EventArgs e)

{

renderControl1.RowCount = Convert.ToInt32(((NumericUpDown)sender).Value);

renderControl1.Invalidate();

}

}

}

### Код файлу (RenderControl.cs)

using System;

using System.Drawing;

namespace Task\_2

{

public partial class RenderControl : OpenGL

{

// Renamed variables for clarity

public bool FillMode { get; set; } = false;

public bool PointMode { get; set; } = false;

public bool LineMode { get; set; } = false;

public int HexagonCount { get; set; } = 1;

public int RowCount { get; set; } = 1;

private readonly double hexagonSize = 100;

public RenderControl()

{

InitializeComponent();

}

private void OnRender(object sender, EventArgs e)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glLoadIdentity();

if (Width > Height)

glViewport((Width - Height) / 2, 0, Height, Height);

else

glViewport(0, (Height - Width) / 2, Width, Width);

// Adjusted viewport size calculation

double viewportSize = Math.Max(HexagonCount, RowCount) \* hexagonSize \* 3.5;

glOrtho(-viewportSize, viewportSize, -viewportSize, viewportSize, -1, +1);

if (FillMode)

DrawPrimitives(GL\_FILL);

if (PointMode)

{

glPointSize(7);

DrawPrimitives(GL\_POINT);

}

if (LineMode)

DrawPrimitives(GL\_LINE);

}

private void DrawPrimitives(uint mode)

{

double hexagonX;

double hexagonY;

for (int j = 0; j < RowCount; j++)

{

hexagonX = -hexagonSize \* 1.5 \* j;

hexagonY = -hexagonSize \* 0.85 \* j;

glPolygonMode(GL\_FRONT\_AND\_BACK, mode);

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

{

glBegin(GL\_TRIANGLES);

// Calculate the vertices for the hexagon

double[][] vertices = new double[6][];

for (int k = 0; k < 6; k++)

{

double angle = 2 \* Math.PI \* k / 6;

vertices[k] = new double[] { hexagonX + hexagonSize \* Math.Cos(angle), hexagonY + hexagonSize \* Math.Sin(angle) };

}

// Define colors in the specified order

float[][] colors = new float[][]

{

new float[] {1.0f, 1.0f, 0.0f}, // Green

new float[] {0.0f, 0.0f, 1.0f}, // Purple

new float[] {1.0f, 1.0f, 0.0f}, // Red

new float[] {0.0f, 1.0f, 0.0f}, // Yellow

new float[] {1.0f, 1.0f, 0.0f}, // Black

new float[] {1.0f, 0.0f, 0.0f} // White

};

// Draw the hexagon using triangles with specified colors

for (int k = 0; k < 6; k++)

{

glColor3fv(colors[k]); // Apply color to each triangle

glVertex2d(hexagonX, hexagonY);

glVertex2d(vertices[k][0], vertices[k][1]);

glVertex2d(vertices[(k + 1) % 6][0], vertices[(k + 1) % 6][1]);

}

glEnd();

// Adjust positions for new hexagons

hexagonX += hexagonSize \* 1.5;

hexagonY -= hexagonSize \* Math.Sqrt(3) \* 0.5;

}

}

}

}

}

# Додаток В. Лістинг програми до практичної роботи №3

### Код файлу (MainForm.cs)

using System;

using System.Windows.Forms;

namespace Task\_3

{

public partial class MainForm : Form

{

public MainForm()

{

InitializeComponent();

}

private void MainForm\_Load(object sender, EventArgs e)

{

}

private void numericUpDown1\_ValueChanged(object sender, EventArgs e) // for xmin

{

NumericUpDown numericUpDown = (NumericUpDown)sender;

renderControl1.SetXMin(Convert.ToSingle(numericUpDown.Value));

renderControl1.Invalidate();

}

private void numericUpDown2\_ValueChanged(object sender, EventArgs e) // for xmax

{

NumericUpDown numericUpDown = (NumericUpDown)sender;

renderControl1.SetXMax(Convert.ToSingle(numericUpDown.Value));

renderControl1.Invalidate();

}

private void numericUpDown3\_ValueChanged(object sender, EventArgs e) // for points

{

NumericUpDown numericUpDown = (NumericUpDown)sender;

renderControl1.SetPointCount(Convert.ToInt32(numericUpDown.Value));

renderControl1.Invalidate();

}

private void numericUpDown5\_ValueChanged(object sender, EventArgs e) // for ymin

{

NumericUpDown numericUpDown = (NumericUpDown)sender;

renderControl1.SetYMin(Convert.ToSingle(numericUpDown.Value));

renderControl1.Invalidate();

}

private void numericUpDown4\_ValueChanged(object sender, EventArgs e) // for ymax

{

NumericUpDown numericUpDown = (NumericUpDown)sender;

renderControl1.SetYMax(Convert.ToSingle(numericUpDown.Value));

renderControl1.Invalidate();

}

}

}

### Код файлу (RenderControl.cs)

using System;

using System.Collections.Generic;

using static System.Math;

namespace Task\_3

{

public partial class RenderControl : OpenGL

{

private float x\_min, x\_max;

private float y\_min = -2, y\_max = 2.5f;

private int point;

private List<float> point\_zeros = new List<float>();

private float increment = 0;

private bool manualYRange = false; // New flag for manual Y range

public RenderControl()

{

this.x\_min = -0.2f;

this.x\_max = 0.8f;

this.point = 480; // Number of points

this.increment = (Abs(x\_min) + Abs(x\_max)) / Convert.ToSingle(point);

InitializeComponent();

}

public void SetXMin(float x\_min)

{

this.x\_min = x\_min;

}

public void SetXMax(float x\_max)

{

this.x\_max = x\_max;

}

public void SetPointCount(int point)

{

this.point = point;

this.increment = (Abs(x\_min) + Abs(x\_max)) / Convert.ToSingle(point);

}

public void SetYMin(float y\_min)

{

this.y\_min = y\_min;

manualYRange = true; // Enable manual Y range

}

public void SetYMax(float y\_max)

{

this.y\_max = y\_max;

manualYRange = true; // Enable manual Y range

}

// Calculate Y range and zeros of f(x)

private void CalculateFunctionInfo(Func<float, float> func)

{

if (!manualYRange)

{

y\_min = float.MaxValue;

y\_max = float.MinValue;

}

point\_zeros.Clear();

float prev\_y = float.MaxValue;

for (float x = x\_min; x < x\_max; x += increment)

{

float y = func(x);

// Update Ymin and Ymax (Automatic scaling for Y-axis if not manual)

if (!manualYRange)

{

if (y < y\_min) y\_min = y - 0.1f;

if (y > y\_max) y\_max = y + 0.1f;

}

// Find roots of the function (f(x) = 0)

if (prev\_y \* y <= 0)

{

point\_zeros.Add(x - increment / 2); // Midpoint of interval

}

prev\_y = y;

}

}

// Function f1(x) = (cos(pi\*x)) / ((sin((5\*pi\*x)/3) + 1.5)^3)

private float FunctionF1(float x)

{

return (float)(Cos(PI \* x) / Pow(Sin((5 \* PI \* x) / 3) + 1.5, 3));

}

// Function f2(x) = tg(2\*sin(x))

private float FunctionF2(float x)

{

if (Cos(2 \* Sin(x)) == 0) return float.NaN; // Handle discontinuity

return (float)Tan(2 \* Sin(x));

}

// Draw coordinate axes

private void DrawMainAxes()

{

glLineWidth(1);

glColor3ub(200, 200, 200);

glBegin(GL\_LINES);

glVertex2d(x\_min, 0);

glVertex2d(x\_max, 0);

glVertex2d(0, y\_min);

glVertex2d(0, y\_max);

glEnd();

}

// Draw function graph

private void DrawFunction(Func<float, float> func, byte r, byte g, byte b)

{

glLineWidth(2);

glColor3ub(r, g, b); // Color for the graph

glBegin(GL\_LINE\_STRIP);

for (float x = x\_min; x < x\_max; x += increment)

{

float y = func(x);

if (!float.IsNaN(y) && !float.IsInfinity(y)) // Skip invalid values

glVertex2f(x, y);

}

glEnd();

}

// Highlight zero points

private void DrawZeroPoints()

{

glPointSize(8);

glColor3ub(255, 165, 0); // Orange color for zeros

glBegin(GL\_POINTS);

foreach (var zero in point\_zeros)

{

glVertex2f(zero, 0);

}

glEnd();

}

private void OnRender(object sender, EventArgs e)

{

glClearColor(0.1f, 0.1f, 0.1f, 1.0f); // Dark grey background

glClear(GL\_COLOR\_BUFFER\_BIT);

glLoadIdentity();

glViewport(0, 0, Width, Height);

// Compute graph data for f1(x)

CalculateFunctionInfo(FunctionF1);

glOrtho(x\_min, x\_max, y\_min, y\_max, -1, 1);

DrawMainAxes(); // Draw coordinate axes

DrawFunction(FunctionF1, 0, 0, 255); // Blue for f1(x)

DrawZeroPoints(); // Highlight zeros of f1(x)

// Advanced: Draw f2(x) with proper handling of discontinuities

DrawFunction(FunctionF2, 50, 205, 50); // Lime green for f2(x)

}

}

}

# Додаток В. Лістинг програми до практичної роботи №3

### Код файлу (MainForm.cs)

//MainForm.cs

using System;

using System.Windows.Forms;

using static Task04.OpenGL;

namespace Task04

{

public partial class MainForm : Form

{

public MainForm()

{

InitializeComponent();

}

private void ChosedCircle(object sender, System.EventArgs e)

{

renderControl1.figur = false; // Select parametric circle

Coordinat2.Visible = false; // Hide second parameter input

renderControl1.Invalidate();

}

private void ChoseHyperbola(object sender, EventArgs e)

{

renderControl1.figur = true; // Select explicit hyperbola

Coordinat2.Visible = true; // Show second parameter input

renderControl1.Invalidate();

}

private void Mouse\_Click(object sender, MouseEventArgs e)

{

if (e.Button == MouseButtons.Left)

{

double difference = Math.Max(renderControl1.ClientRectangle.Width, renderControl1.ClientRectangle.Height) - Math.Min(renderControl1.ClientRectangle.Width, renderControl1.ClientRectangle.Height);

if (difference != 0)

{

renderControl1.LineExist = renderControl1.ClientRectangle.Width > renderControl1.ClientRectangle.Height ?

difference / 2 < e.X && e.X < Math.Min(renderControl1.ClientRectangle.Width, renderControl1.ClientRectangle.Height) + difference / 2 :

difference / 2 < renderControl1.ClientRectangle.Height - e.Y && renderControl1.ClientRectangle.Height - e.Y < Math.Min(renderControl1.ClientRectangle.Width, renderControl1.ClientRectangle.Height) + difference / 2;

if (renderControl1.LineExist)

{

renderControl1.IsUp = false;

double xLength = renderControl1.ClientRectangle.Width > renderControl1.ClientRectangle.Height ?

renderControl1.ClientRectangle.Width / (double)renderControl1.ClientRectangle.Height :

1;

double yLength = renderControl1.ClientRectangle.Height > renderControl1.ClientRectangle.Width ?

renderControl1.ClientRectangle.Height / (double)renderControl1.ClientRectangle.Width :

1;

renderControl1.StartX = renderControl1.EndX = (renderControl1.Length \* xLength) \* ((e.X - (renderControl1.ClientRectangle.Width / 2.0)) / (double)(renderControl1.ClientRectangle.Width / 2.0));

renderControl1.StartY = renderControl1.EndY = (renderControl1.Length \* yLength) \* (((renderControl1.ClientRectangle.Height / 2.0) - e.Y) / (double)(renderControl1.ClientRectangle.Height / 2.0));

}

else

{

}

}

else

{

renderControl1.IsUp = false;

renderControl1.LineExist = true;

double xLength = renderControl1.ClientRectangle.Width > renderControl1.ClientRectangle.Height ?

renderControl1.ClientRectangle.Width / (double)renderControl1.ClientRectangle.Height :

1;

double yLength = renderControl1.ClientRectangle.Height > renderControl1.ClientRectangle.Width ?

renderControl1.ClientRectangle.Height / (double)renderControl1.ClientRectangle.Width :

1;

renderControl1.StartX = renderControl1.EndX = (renderControl1.Length \* xLength) \* ((e.X - (renderControl1.ClientRectangle.Width / 2.0)) / (double)(renderControl1.ClientRectangle.Width / 2.0));

renderControl1.StartY = renderControl1.EndY = (renderControl1.Length \* yLength) \* (((renderControl1.ClientRectangle.Height / 2.0) - e.Y) / (double)(renderControl1.ClientRectangle.Height / 2.0));

}

}

renderControl1.Invalidate();

}

private void oCoordinat1(object sender, EventArgs e)

{

renderControl1.A = (double)Coordinat1.Value;

renderControl1.Invalidate();

}

private void oCoordinat2(object sender, EventArgs e)

{

renderControl1.B = (double)Coordinat2.Value;

renderControl1.Invalidate();

}

private void ReleaseTheMouse(object sender, MouseEventArgs e)

{

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

{

double xLength = renderControl1.ClientRectangle.Width > renderControl1.ClientRectangle.Height ?

renderControl1.ClientRectangle.Width / (double)renderControl1.ClientRectangle.Height :

1;

double yLength = renderControl1.ClientRectangle.Height > renderControl1.ClientRectangle.Width ?

renderControl1.ClientRectangle.Height / (double)renderControl1.ClientRectangle.Width :

1;

renderControl1.EndX = (renderControl1.Length \* xLength) \* ((e.X - (renderControl1.ClientRectangle.Width / 2.0)) / (double)(renderControl1.ClientRectangle.Width / 2.0));

renderControl1.EndY = (renderControl1.Length \* yLength) \* (((renderControl1.ClientRectangle.Height / 2.0) - e.Y) / (double)(renderControl1.ClientRectangle.Height / 2.0));

renderControl1.IsUp = true;

if (renderControl1.EndX == renderControl1.StartX && renderControl1.EndY == renderControl1.StartY)

{

renderControl1.LineExist = false;

}

else

{

}

}

renderControl1.Invalidate();

}

private void Mouse\_Movement(object sender, MouseEventArgs e)

{

if (renderControl1.LineExist)

{

if (!renderControl1.IsUp)

{

double xLength = renderControl1.ClientRectangle.Width > renderControl1.ClientRectangle.Height ?

renderControl1.ClientRectangle.Width / (double)renderControl1.ClientRectangle.Height :

1;

double yLength = renderControl1.ClientRectangle.Height > renderControl1.ClientRectangle.Width ?

renderControl1.ClientRectangle.Height / (double)renderControl1.ClientRectangle.Width :

1;

renderControl1.EndX = (renderControl1.Length \* xLength) \* ((e.X - (renderControl1.ClientRectangle.Width / 2.0)) / (double)(renderControl1.ClientRectangle.Width / 2.0));

renderControl1.EndY = (renderControl1.Length \* yLength) \* (((renderControl1.ClientRectangle.Height / 2.0) - e.Y) / (double)(renderControl1.ClientRectangle.Height / 2.0));

renderControl1.Invalidate();

}

}

}

}

}

### Код файлу (RenderControl.cs)

//RenderContol.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Diagnostics;

using System.Drawing;

using System.Drawing.Design;

using System.Linq;

using System.Security.Policy;

namespace Task04

{

public partial class RenderControl : OpenGL

{

Figures draw;

public double StartX { get; set; }

public double EndX { get; set; }

public double StartY { get; set; }

public double EndY { get; set; }

public double Length { get; set; } = 1.5;

public bool figur { get; set; } = false;

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

public bool LineExist { get; set; } = false;

public double A { get; set; } = 2.0;

public double B { get; set; } = 1.0;

public RenderControl()

{

InitializeComponent();

}

private void OnRender(object sender, EventArgs e)

{

glClearColor(1.0f, 1.0f, 1.0f, 1.0f);

glClear(GL\_COLOR\_BUFFER\_BIT);

glLoadIdentity();

int size = Math.Min(Width, Height);

glViewport((Width - size) / 2, (Height - size) / 2, size, size);

Length = draw.FindMax(A, B, figur);

gluOrtho2D(-Length, Length, -Length, Length);

draw.DrawGrid(-Length, Length, -Length, Length);

draw.DrawCoordinateGrid(-Length, Length, -Length, Length);

if (!figur)

draw.ParametricCircle(A);

else

draw.ExplicitHyperbola(A, B);

if (LineExist)

{

CreateLine();

draw.SearchPoint(A, B, StartX, StartY, EndX, EndY, figur);

}

}

private void CreateLine()

{

if (!IsUp)

{

draw.DrawLineWait(StartX, EndX, StartY, EndY);

}

else

{

draw.DrawLine(StartX, EndX, StartY, EndY);

}

}

private void Created(object sender, EventArgs e)

{

draw = new Figures();

}

}

}

### Код файлу (Figures.cs)

using System;

using System.Collections.Generic;

using System.Drawing.Printing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using static System.Windows.Forms.VisualStyles.VisualStyleElement;

namespace Task04

{

public partial class RenderControl

{

public class Figures

{

public void ParametricCircle(double radius)

{

glColor3f(255f, 0, 0);

glBegin(GL\_LINE\_LOOP);

int segments = 100; // Number of segments

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

{

double t = 2.0 \* Math.PI \* i / segments; // Angle in radians

double x = radius \* Math.Cos(t);

double y = radius \* Math.Sin(t);

glVertex2d(x, y);

}

glEnd();

}

// Draw Explicit Hyperbola

public void ExplicitHyperbola(double a, double b)

{

glColor3f(255f, 0, 0);

glBegin(GL\_LINE\_STRIP);

double range = 2.0; // Range for x-values

int segments = 100;

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

{

double x = range \* (i - segments / 2.0) / (segments / 2.0);

double y = b \* Math.Sqrt((x \* x) / (a \* a) - 1);

glVertex2d(x, y);

}

glEnd();

}

public void SearchPoint(double a, double b, double StartX, double EndX, double StartY, double EndY, bool isChoosed)

{

glPointSize(5.0f);

glColor3f(0.0f, 1.0f, 1.0f);

glBegin(GL\_POINTS);

double dx = EndX - StartX;

double dy = EndY - StartY;

if (!isChoosed)

{

double A = dx \* dx + dy \* dy;

double B = 2 \* (StartX \* dx + StartY \* dy);

double C = StartX \* StartX + StartY \* StartY - a \* a;

double discriminant = B \* B - 4 \* A \* C;

if (discriminant >= 0)

{

double sqrtD = Math.Sqrt(discriminant);

double t1 = (-B + sqrtD) / (2 \* A);

double t2 = (-B - sqrtD) / (2 \* A);

if (t1 >= 0 && t1 <= 1)

{

double x1 = StartX + t1 \* dx;

double y1 = StartY + t1 \* dy;

glVertex2d(x1, y1);

}

if (t2 >= 0 && t2 <= 1)

{

double x2 = StartX + t2 \* dx;

double y2 = StartY + t2 \* dy;

glVertex2d(x2, y2);

}

}

}

else

{

double A = a \* dx \* dx;

double B = 2 \* a \* dx \* StartX - dy;

double C = a \* StartX \* StartX - StartY;

double discriminant = B \* B - 4 \* A \* C;

if (discriminant >= 0)

{

double sqrtD = Math.Sqrt(discriminant);

double t1 = (-B + sqrtD) / (2 \* A);

double t2 = (-B - sqrtD) / (2 \* A);

if (t1 >= 0 && t1 <= 1)

{

double x1 = StartX + t1 \* dx;

double y1 = StartY + t1 \* dy;

glVertex2d(x1, y1);

}

if (t2 >= 0 && t2 <= 1)

{

double x2 = StartX + t2 \* dx;

double y2 = StartY + t2 \* dy;

glVertex2d(x2, y2);

}

}

}

glEnd();

}

public void Circle(double radius)

{

glColor3f(1.0f, 0.5f, 1.0f);

glBegin(GL\_LINE\_LOOP);

int segments = 100;

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

{

double theta = 2.0 \* Math.PI \* i / segments; // Кут в радіанах

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

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

glVertex2d(x, y);

}

glEnd();

}

public void ParametricPorabola(double a)

{

glColor3f(1.0f, 0.0f, 0.0f);

glBegin(GL\_LINE\_STRIP);

int segments = 100;

double range = 2.0; // Діапазон значень для параметру t

for (int i = -segments; i <= segments; i++)

{

double t = i \* range / segments; // Параметр t

double x = t;

double y = a \* t \* t;

glVertex2d(x, y);

}

glEnd();

}

public double FindMax(double a, double b, bool isChoosed)

{

double maxX, maxY;

if (!isChoosed)

{

maxX = Math.Abs(a);

maxY = Math.Abs(b);

}

else

{

double range = 2.0;

maxX = range;

maxY = a \* range \* range;

maxY = maxY < 0 ? -maxY : maxY;

}

return Math.Max(maxX, maxY) + 0.2;

}

public void DrawGrid(double minX, double maxX, double minY, double maxY)

{

glLineWidth(1.0f);

glColor3ub(200, 200, 200);

glBegin(GL\_LINES);

for (double i = minX; i <= maxX; i += maxX / 10)

{

glVertex2d(i, minY);

glVertex2d(i, maxY);

}

for (double j = minY; j <= maxY; j += maxY / 10)

{

glVertex2d(minX, j);

glVertex2d(maxX, j);

}

glEnd();

}

public void DrawCoordinateGrid(double minX, double maxX, double YMin, double YMax)

{

glLineWidth(2.0f);

glBegin(GL\_LINES);

glColor3ub(0, 0, 0);

glVertex2d(minX, 0);

glVertex2d(maxX, 0);

glVertex2d(0, YMin);

glVertex2d(0, YMax);

glEnd();

glLineWidth(1.0f);

}

public void DrawLineWait(double minX, double maxX, double YMin, double YMax)

{

glColor3f(0.5f, 0.5f, 0.5f);

glLineWidth(1.4f);

glPointSize(3.0f);

glBegin(GL\_POINTS);

glVertex2d(minX, YMin);

glVertex2d(maxX, YMax);

glEnd();

glBegin(GL\_LINES);

glVertex2d(minX, YMin);

glVertex2d(maxX, YMax);

glEnd();

}

public void DrawLine(double minX, double maxX, double YMin, double YMax)

{

glColor3f(0.0f, 0.0f, 0.0f);

glLineWidth(1.7f);

glBegin(GL\_LINES);

glVertex2d(minX, YMin);

glVertex2d(maxX, YMax);

glEnd();

}

}

}

}

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