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*

освітньої програми

*121 інженерія програмного забезпечення*

(шифр і назва ОП)

*Петренко Є.О.*

(прізвище й ініціали студента)

Прийняв: доцент каф. 603, *к.т.н*

*Лучшев П.О.*

(посада, науковий ступінь, прізвище й ініціали)

Кількість балів:

Харків – 2024

ЗМІСТ

[ПРАКТИЧНА РОБОТА 1. ОСНОВНІ ПРИНЦИПИ РОБОТИ З OPENGL 3](#_Toc185018805)

[Завдання, варіант № 16 3](#_Toc185018806)

[Системна інформація 3](#_Toc185018807)

[Теоретичні відомості 4](#_Toc185018808)

[Результати виконання практичної роботи 6](#_Toc185018809)

[ПРАКТИЧНА РОБОТА 2. ГРАФІЧНІ ПРИМІТИВИ OPENGL 8](#_Toc185018810)

[Завдання, варіант № 16 8](#_Toc185018811)

[Результати виконання практичної роботи 9](#_Toc185018812)

[ПРАКТИЧНА РОБОТА 3. ГРАФІК ФУНКЦІЇ ОДНІЄЇ ЗМІННОЇ 14](#_Toc185018813)

[Завдання, варіант № 16 14](#_Toc185018814)

[Результати виконання практичної роботи 15](#_Toc185018815)

[ПРАКТИЧНА РОБОТА 4. КРИВІ ДРУГОГО ПОРЯДКУ 18](#_Toc185018816)

[Завдання, варіант № 16 18](#_Toc185018817)

[Результати виконання практичної роботи 19](#_Toc185018818)

[ПРАКТИЧНА РОБОТА 5. АФІННІ ПЕРЕТВОРЕННЯ У ПРОСТОРІ 23](#_Toc185018819)

[Завдання, варіант № 16 23](#_Toc185018820)

[Результати виконання практичної роботи 24](#_Toc185018821)

[ДОДАТОК А Лістинг програми до практичної роботи №1 29](#_Toc185018822)

[ДОДАТОК Б Лістинг програми до практичної роботи №2 34](#_Toc185018823)

[ДОДАТОК В Лістинг програми до практичної роботи №3 38](#_Toc185018824)

[ДОДАТОК Г Лістинг програми до практичної роботи №4 42](#_Toc185018825)

[ДОДАТОК Ґ Лістинг програми до практичної роботи №5 46](#_Toc185018826)

# ПРАКТИЧНА РОБОТА 1. ОСНОВНІ ПРИНЦИПИ РОБОТИ З OPENGL

# Завдання, варіант № 16

За допомогою інструментальних засобів, зазначених викладачем, створити простий програмний проєкт із підтримкою бібліотеки OpenGL. Враховуючи систему оцінки, 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.

|  |  |  |
| --- | --- | --- |
| Номер варіанта | Параметри | Фігура |
| 16 | Примітиви:  GL\_POINTS, GL\_LINES  x1 = -3; x2 = 6  y1 = 0; y2 = 4 |  |

# Системна інформація

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

Processor AMD Ryzen 5 4600H @ 3.0Ghz

RAM 16.0 GB (15.4 GB usable)

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

Edition Windows 10 Pro Version 22H2

IDE Microsoft Visual Studio Enterprise 2022 (64-bit) version 17.11.3

# Теоретичні відомості

***Вершинні масиви***

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

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

̶ glVertexPointer(): вказати покажчик на масив вершинних координат

̶ glNormalPointer(): вказати покажчик на звичайний масив

̶ glColorPointer(): вказати вказівник на масив кольорів RGB

̶ glIndexPointer(): вказати покажчик на індексований кольоровий масив

̶ glTexCoordPointer(): вказати покажчик на масив текстурних шнурів

̶ glEdgeFlagPointer(): вказати вказівник на масив прапорців edge

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

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

***Команда glDrawArrays()***

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

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

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

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

...

// activate and specify pointer to vertex array

glEnableClientState(GL\_VERTEX\_ARRAY);

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

// draw a cube

glDrawArrays(GL\_TRIANGLES, 0, 36);

// deactivate vertex arrays after drawing

glDisableClientState(GL\_VERTEX\_ARRAY);

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

***Команда glDrawElements()***

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

# Результати виконання практичної роботи

***Розв’язання завдання***

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

* колір, glColor3d() у рядку 44 у файлі RenderControl.cs; у рядках 33, 60, 79, 100 у файлі Layout.cs; у рядку 50 у файлі Figure.cs;
* тип glLineStipple у рядку 35 файлу Layout.cs;
* товщина ліній glLineWidth у рядках 34, 59, 78, 99 у файлі Layout.cs, у рядку 48 файлу Figure.cs;
* товщина точок glPointSize у рядку 49 файлу Figure.cs.

Коректне відображення під час змінення розмірів/положення вікна наведену на рисунках 1.1 та 1.2.

Розроблення підпрограми для виключення дублювання коду наведено у рядках 46–71 файлу Figure.cs. Ця підпрограма робить можливим малювання фігури, заданої в варіанті, без модифікації масиву координат для типу GL\_LINES (відрізки).

Застосування циклів для створення зображень наведено у рядках 40–50 файлу Layout.cs, та у рядках 53–58, 72–75 файлу Grid.cs.

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

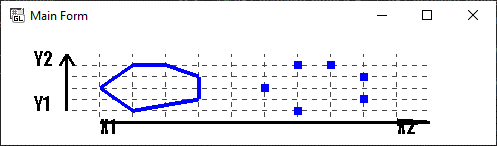


Рисунок 1.1 – Тестування програми при зміні висоти вікна

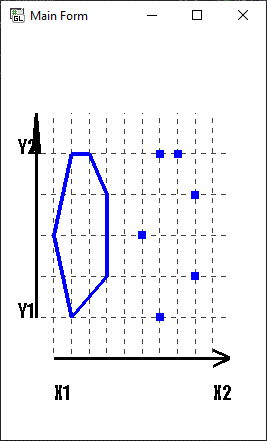


Рисунок 1.2 – Тестування програми при зміні товщини вікна

***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 – Результат виконання практичної роботи №1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Складність | Вимоги до роботи | Бали | Оцінка |
| 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 тощо) | 1 | - |
| 6. | Використання ООП (розроблення власних класів) | 2 | + |

# ПРАКТИЧНА РОБОТА 2. ГРАФІЧНІ ПРИМІТИВИ OPENGL

# Завдання, варіант № 16

Використовуючи інструментальні засоби, що вказані викладачем, і беручи до уваги вимог, створити програмний проєкт з підтримкою OpenGL. За допомогою команд glOrtho / gluOrtho2D і glViewport встановити для робочої області ізотропну систему координат з урахуванням розміру фігури, яку задано у варіанті. Після старту застосунок повинен відображати у робочій області одну плитку (tile).

|  |  |  |
| --- | --- | --- |
| Номер варіанта | Параметри | Фігура |
| 16 | Сторона фігури a = 5  Примітив(и):  GL\_TRIANGLE\_FAN,  GL\_POLYGON |  |

# Результати виконання практичної роботи

***Розв’язання завдання***

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

* колір, glColor3d() у рядках 19, 37, 57, 67, 69, 72, 74 файлу Figure.cs;
* тип GL\_POINTS у рядку 20, GL\_TRIANGLE\_FAN у рядках 36, 66, GL\_POLYGON у рядках 46, 56 файлу Figure.cs;
* товщина точок glPointSize у рядках 18, 31 файлу Figure.cs.

Коректне відображення під час змінення розмірів/положення вікна наведену на рисунках 2.1 та 2.2.

Код реалізації багаторазового замощення плиткою знаходиться у файлі RenderControl.cs рядки 38–50. Забезпечено коректне відображення завдання під час зміни розмірів, положення вікна та параметрів замощення. Взаємодія з користувачем відбувається за допомогою клавіатури (для введення кількості тайлів) та миші (для вибору режиму відображення примітивів). Застосовано мінімальну кількість графічних примітивів. (див. рисунки 2.3–2.6).

Розроблення підпрограми для виключення дублювання коду наведено у рядках 27–79 файлу Figure.cs. Ця підпрограма робить можливим малювання фігури, заданої в варіанті.

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

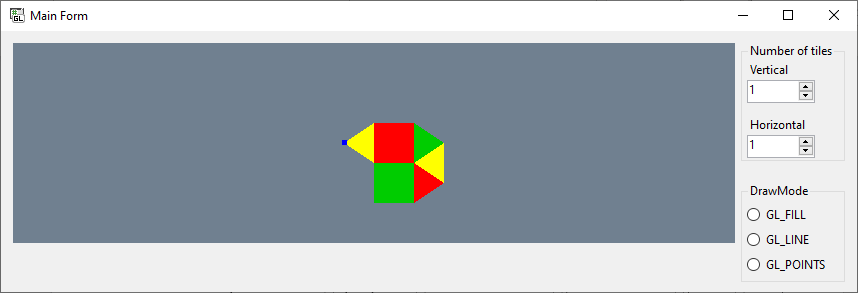


Рисунок 2.1 – Тестування програми при зміні висоти вікна

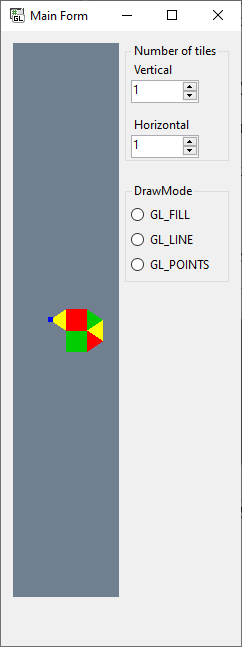


Рисунок 2.2 – Тестування програми при зміні ширини вікна

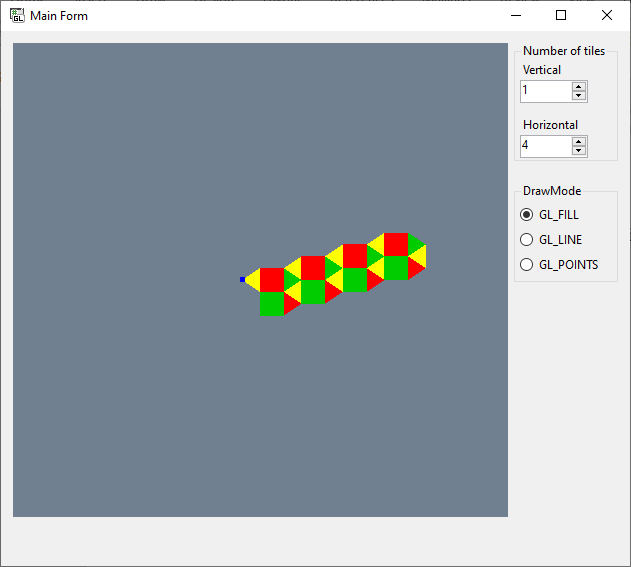


Рисунок 2.3 – Горизонтальне замощення

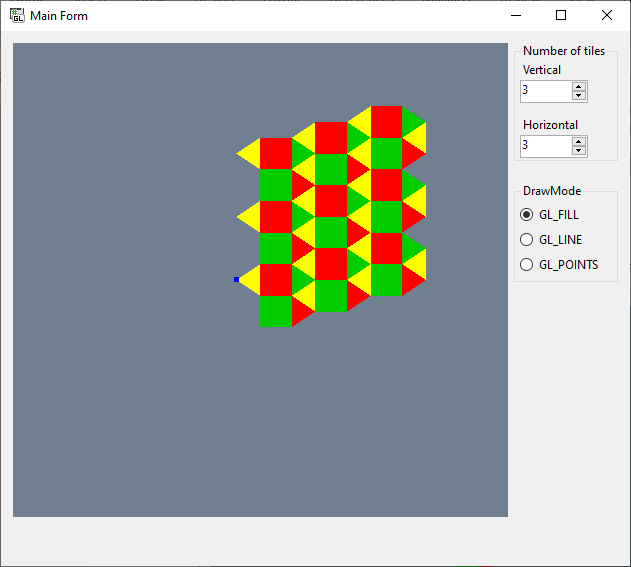


Рисунок 2.4 – Вертикальне та горизонтальне замощення

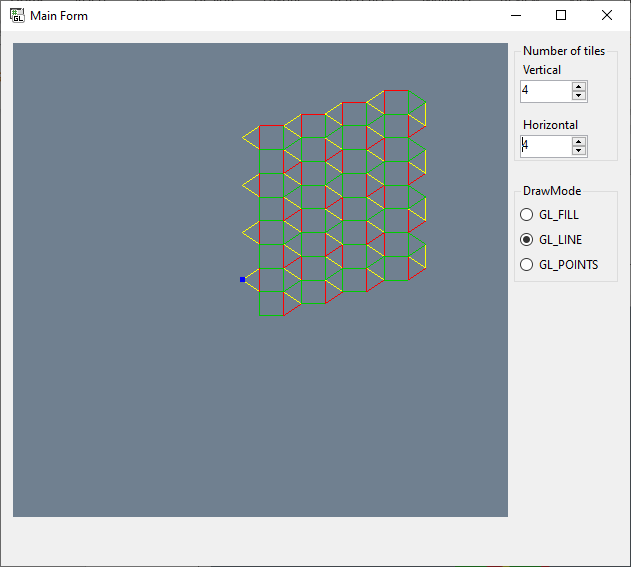


Рисунок 2.5 – Замощення в режимі GL\_LINE

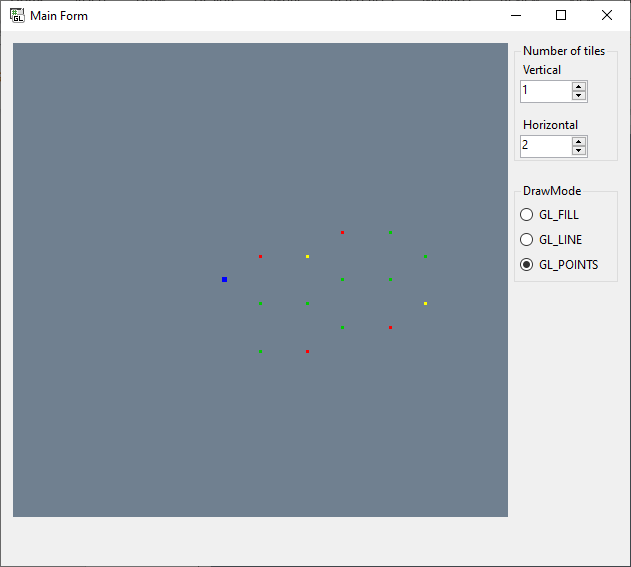


Рисунок 2.6 – Замощення в режимі GL\_POINT

***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 – Результат виконання практичної роботи №2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Складність | Вимоги до роботи | Бали | Оцінка |
| 1. | Базовий рівень | Під час запуску застосунку зображення відповідає варіанту завдання з однією плиткою | 1 | + |
| 2. | Багаторазова замощення плиткою. 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 (розроблення власних класів) | 1 | + |

# ПРАКТИЧНА РОБОТА 3. ГРАФІК ФУНКЦІЇ ОДНІЄЇ ЗМІННОЇ

# Завдання, варіант № 16

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

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

Для підвищеного рівня складності необхідно додатково реалізувати коректне виведення функції 𝑓2 (𝑥) з урахуванням області визначення функції і відобразити лінії розриву.

|  |  |
| --- | --- |
| Номер варіанта | Функція |
| 16 |  |
|  |

# Результати виконання практичної роботи

***Розв’язання завдання***

Код реалізації виведення осей координат і графіка функції f1(x) на заданому користувачем інтервалі від Xmin до Xmax і від Ymin до Ymax знаходиться в файлах Layout.cs рядки 13–41 (осі координат та координатна сітка), DrawFunction.cs рядки 13–47.

Код автоматичного обчислення Ymin і Ymax на завданому інтервалі від Xmin до Xmax функції f1(x) знаходиться в файлі DrawFunction.cs рядки 49–70.

Робота програми з автоматичним обчисленням продемонстрована на рисунках 3.1–3.2.

Код обчислення і виведення на екран точок f1(x)=0 знаходиться в файлі DrawFunction.cs рядки 87–94.

Коректне виведення графіка f2(x) (без хибного виводу точок розриву як точок перетину з віссю абсцис) і з відображенням ліній розриву функцій реалізовано в методі DrawFunc класу DrawFunction у рядках 13–47.

Робота програми зі значеннями, заданими користувачем, продемонстрована на рисунках 3.3–3.4.

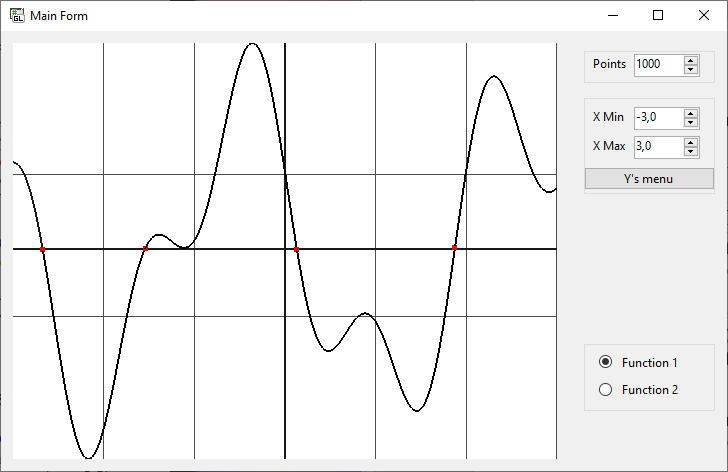
****

Рисунок 3.1 – Функція 1, автоматично обчислені значення Y

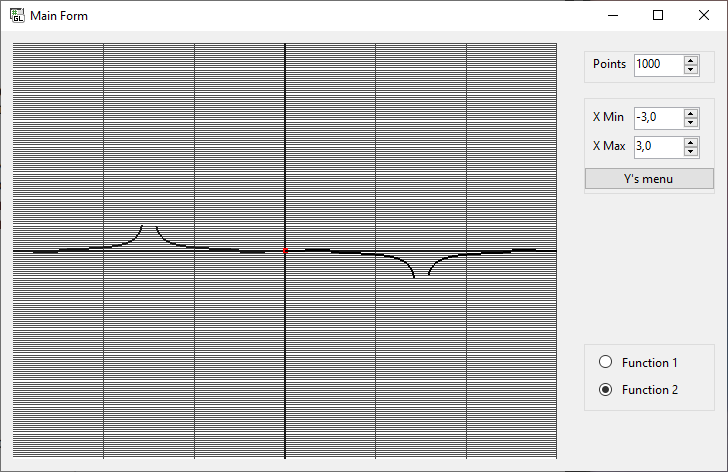


Рисунок 3.2 – Функція 2, автоматично обчислені значення Y

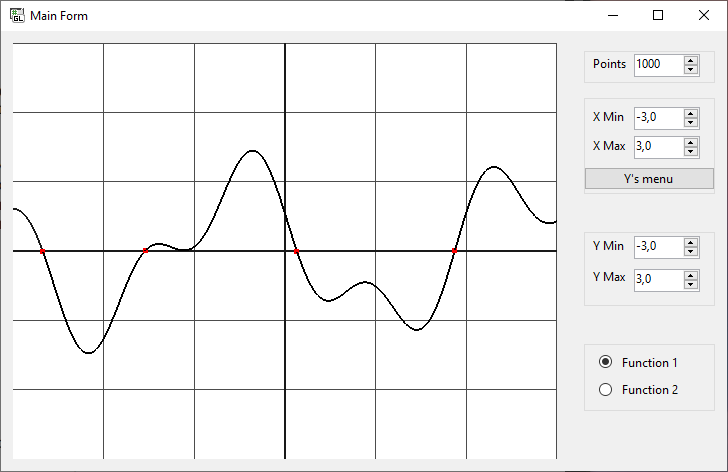


Рисунок 3.3 – Функція 1, Y задані користувачем

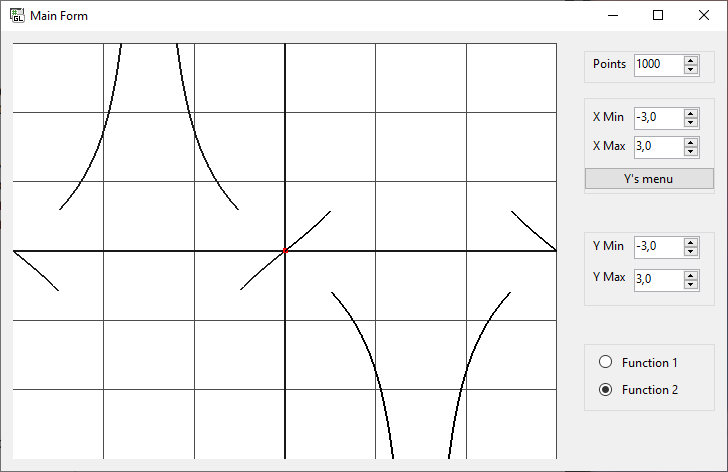


Рисунок 3.4 – Функція 1, Y задані користувачем

***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 – Результат виконання практичної роботи №3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Складність | Вимоги до роботи | Бали | Оцінка |
| 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 функцій | 2 | + |
| 5. | Використання ООП (наслідування, використання віртуальних і абстрактних методів) | 1 | - |

# ПРАКТИЧНА РОБОТА 4. КРИВІ ДРУГОГО ПОРЯДКУ

# Завдання, варіант № 16

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

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:

𝑥0 = (𝑥2 − 𝑥1 )𝑡1 + 𝑥1 = (𝑥4 − 𝑥3 )𝑡2 + 𝑥3

𝑦0 = (𝑦2 − 𝑦1 )𝑡2 + 𝑦1 = (𝑦4 − 𝑦3 )𝑡2 + 𝑦3

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].

|  |  |
| --- | --- |
| Номер варіанта | Функція |
| 16 | Гіпербола (явне подання) ++ |
| Парабола (параметричне подання) + |

# Результати виконання практичної роботи

***Розв’язання завдання***

Код установлення ізотропної системи координат знаходиться в файлі RenderControl.cs у рядках 39–43.

Робота ізотропної системи координат продемонстрована на рисунках 4.1–4.2.

Код виведення кривих другого порядку знаходиться у файлі DrawFigure у рядках 15–47.

Виведення параболи зображено на рисунках 4.1–4.2, гіперболи на рисунку 4.3.

Код виведення відрізка знаходиться у файлі RenderControl.cs у рядках 71–108.

Точки відрізка задаються натисканням на область виведення лівою та правою кнопками миші. Виведення відрізку продемонстровано на рисунку 4.4.

Код виведення точок перетину прямою з кривою другого порядку знаходиться у файлі RenderControl.cs у рядках 110–142.

Точки перетину з’являються при перетинанні відрізка та гіперболи. Їхнє виведення продемонстровано на рисунках 4.5–4.6.

ООП використовується в файлі RenderControl.cs у рядках 16–25 для використання полів, які отримують значення з елементів форми та передають їх в методи.

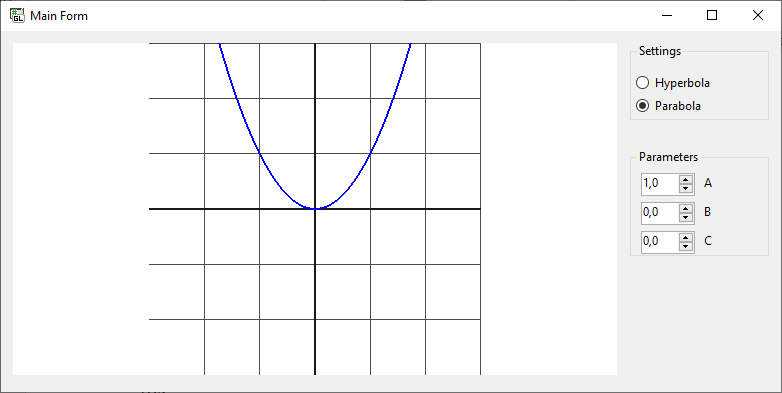
****

Рисунок 4.1 – Тестування програми при зміні ширини вікна

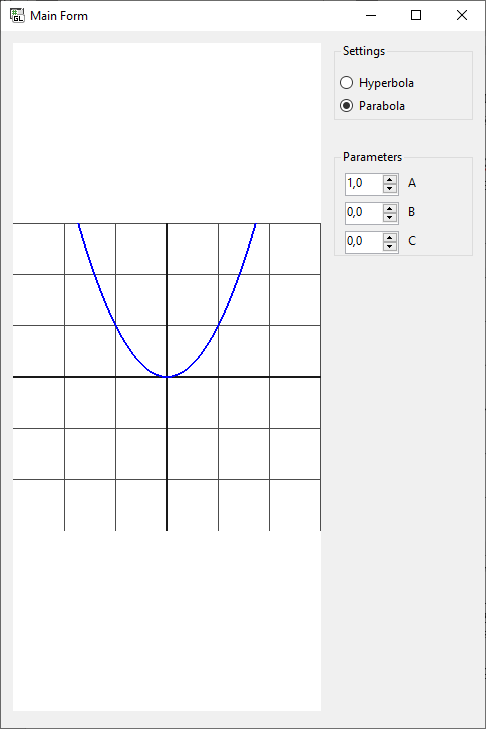


Рисунок 4.2 – Тестування програми при зміні висоти вікна

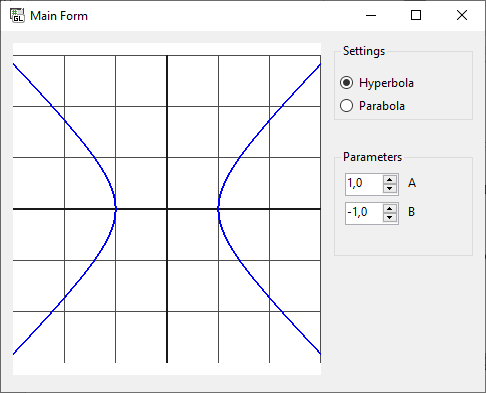


Рисунок 4.3 – Зображення гіперболи

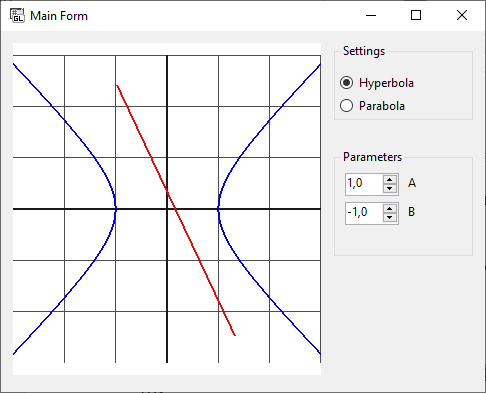


Рисунок 4.4 – Виведення відрізку

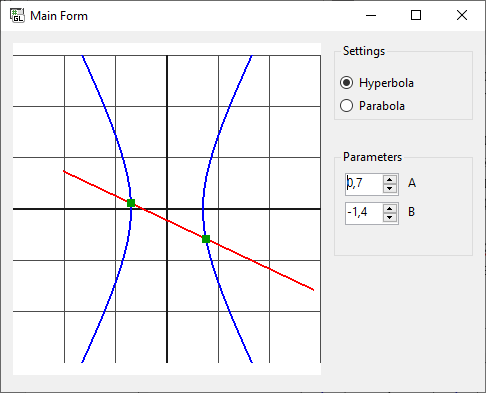


Рисунок 4.5 – Виведення точок перетину. Приклад 1

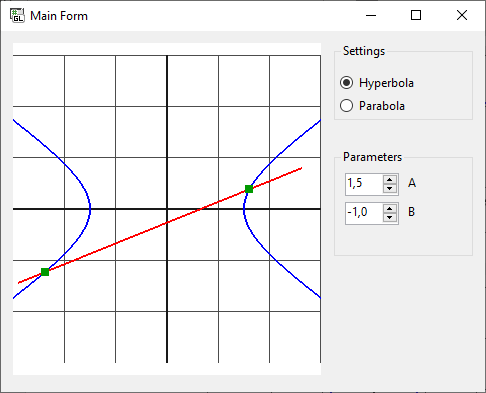


Рисунок 4.6 – Виведення точок перетину. Приклад 2

***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 – Результат виконання практичної роботи №4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Складність | Вимоги до роботи | Бали | Оцінка |
| 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. АФІННІ ПЕРЕТВОРЕННЯ У ПРОСТОРІ

# Завдання, варіант № 16

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.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Номер варіанта | Grid | Параметри квадратичних фігур | | | | | | | | | |
| фігура | ось | x0 | y0 | z0 | R | r | h | ∠start | ∠seep |
| 16 | X0Z | сфера | ||OZ | +4.0 | -1.5 | +2.0 | 2.0 | - | - | - | - |
| конус | ↑↑OX | -3.5 | +0.5 | +2.0 | 0.0 | 1.0 | 2.5 | - | - |
| частинний диск | ||OY | -2.5 | -1.5 | -3.5 | 3.5 | 1.0 | - | 0° | 45° |

# Результати виконання практичної роботи

***Розв’язання завдання***

Код установлення ізотропної системи координат знаходиться в файлі RenderControl.cs у рядках 47–49.

Робота ізотропної системи координат продемонстрована на рисунках 5.1–5.2.

Код відображення осей тривимірного простору та координатної сітки знаходиться у файлі Layout у рядках 16–59.

Відображення осей тривимірного простору та координатної сітки продемонстровано на рисунках 5.1–5.9.

Інтерфейс керування параметрами площини відтину зображено на рисунку 5.1.

Демонстрація роботи відтину зображено на рисунку 5.3.

Код використання джерел світла для освітлення об’єктів сцени сумісно з командою glColorMaterial знаходиться у файлі RenderControl.cs у рядках 91–115.

Використання джерел світла для освітлення об’єктів сцени продемонстровано на рисунках 5.4–5.5.

Код використання списків відображення (Display Lists) знаходиться у файлі RenderControl.cs у рядках 125–145.

Перемикання на режим відображення фігур за допомогою Display List реалізовано елементом інтерфейсу checkbox. Його роботу продемонстровано на рисунку 5.6.

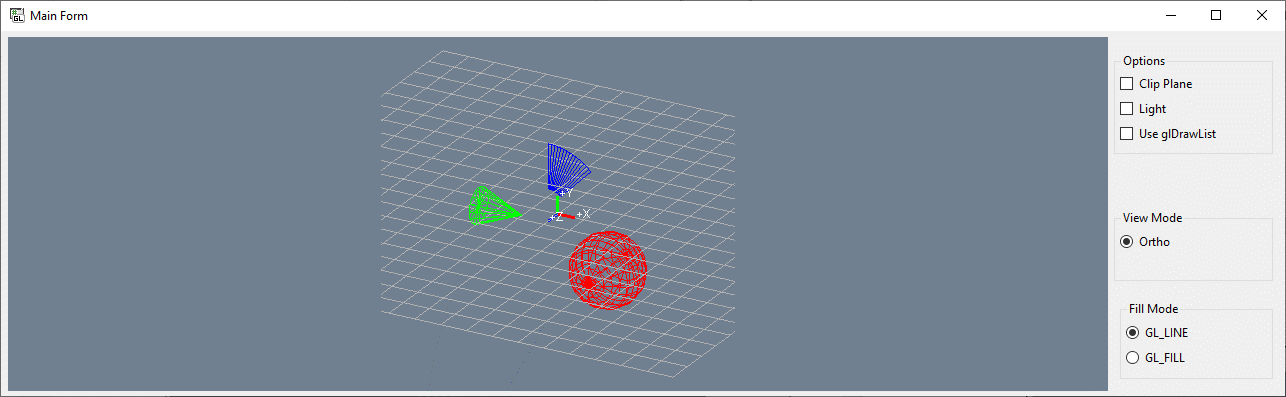
****

Рисунок 5.1 – Тестування програми при зміні ширини вікна

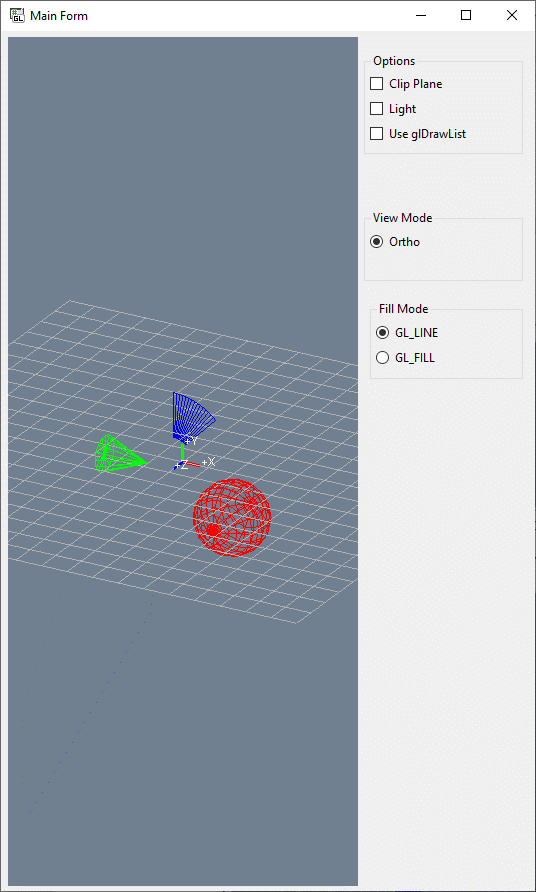


Рисунок 5.2 – Тестування програми при зміні висоти вікна

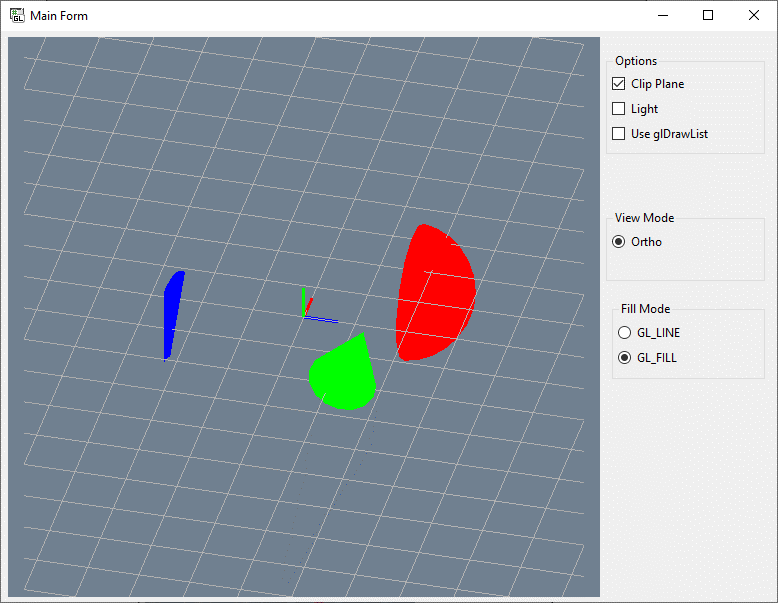


Рисунок 5.3 – Демонстрація роботи відтину

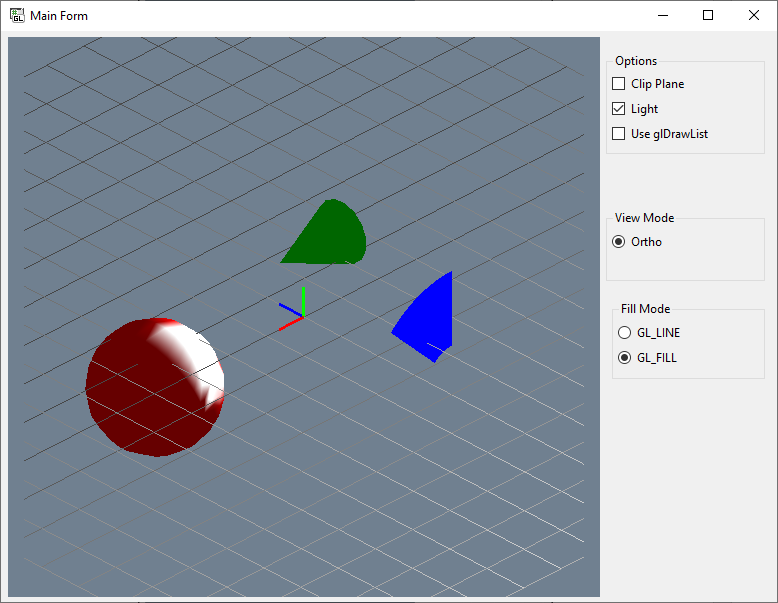


Рисунок 5.4 – Демонстрація роботи світла

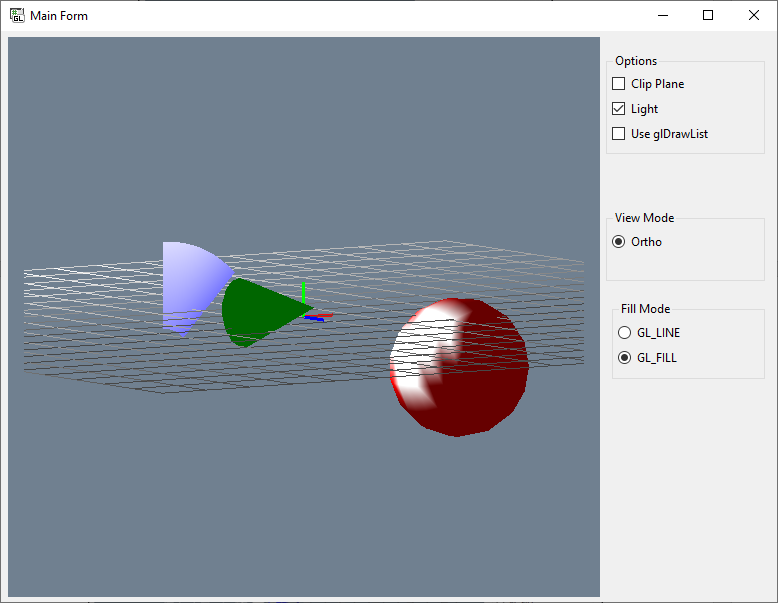


Рисунок 5.5 – Демонстрація роботи світла

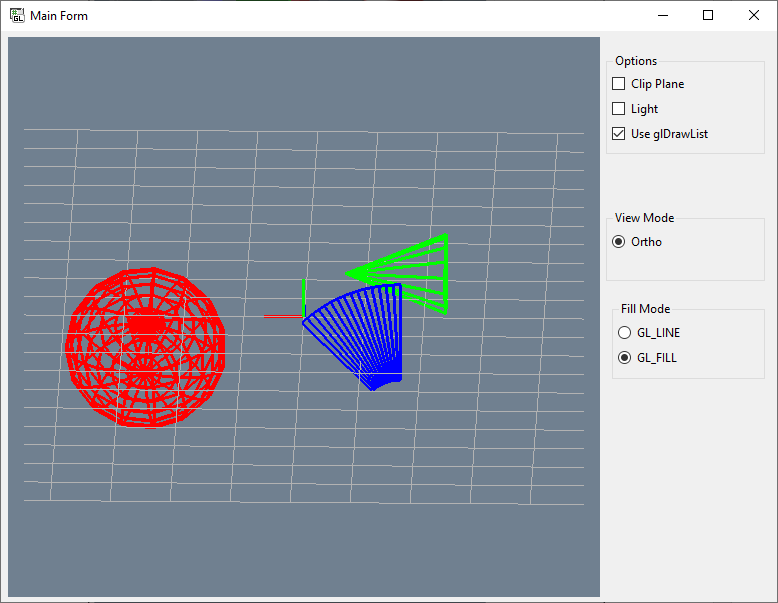


Рисунок 5.6 – Робота glDrawList

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

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

Таблиця 5.1 – Результат виконання практичної роботи №5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Складність | Вимоги до роботи | Бали | Оцінка |
| 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 | – |

# ДОДАТОК А Лістинг програми до практичної роботи №1

**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;

namespace PR1\_Petrenko\_program

{

public partial class RenderControl : OpenGL // Варіант 16

{

private Layout \_layout;

private Figure \_figure;

readonly int X1 = -3;

readonly int Y1 = 0;

readonly int X2 = 6;

readonly int Y2 = 4;

readonly int padding = 3;

public RenderControl()

{

InitializeComponent();

\_layout = new Layout(X1, Y1, X2, Y2, 1);

\_figure = new Figure(new[] { -3, -2, -1, 0, 0, -2 }, new[] { 2, 4, 4, 3, 1, 0 } );

}

private void OnRender(object sender, EventArgs e)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glLoadIdentity();

glViewport(0, 0, Width, Height);

//glOrtho(X1, X2, Y1, Y2, -1, +1);

glOrtho(X1 - padding, X2 + padding, Y1 - padding, Y2 + padding, -1, 1);

\_layout.LoadLayout();

\_figure.DrawComplex();

glColor3d(0, 0, 0);

DrawText("X1", X1, -2);

DrawText("X2", X2, -2);

DrawText("Y1", X1-2, Y1);

DrawText("Y2", X1-2, Y2);

glEnd();

}

}

}

**Код файлу Layout.cs:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Numerics;

using System.Security.Cryptography.X509Certificates;

using System.Text;

using System.Threading.Tasks;

using static PR1\_Petrenko\_program.OpenGL;

namespace PR1\_Petrenko\_program

{

public partial class RenderControl

{

internal class Layout

{

private readonly int \_x1;

private readonly int \_y1;

private readonly int \_x2;

private readonly int \_y2;

private readonly float \_step;

public Layout(int x1, int y1, int x2, int y2, float step)

{

\_x1 = x1;

\_y1 = y1;

\_x2 = x2;

\_y2 = y2;

\_step = step;

}

public void DrawGrid()

{

glColor3d(0.3, 0.3, 0.3);

glLineWidth(1);

glEnable(GL\_LINE\_STIPPLE);

glLineStipple(5, 0xAAAA);

glBegin(GL\_LINES);

for (float x = \_x1; x <= \_x2; x += \_step)

{

glVertex2f(x, \_y1-1);

glVertex2f(x, \_y2+1);

}

for (float y = \_y1; y <= \_y2; y += \_step)

{

glVertex2f(\_x1-1, y);

glVertex2f(\_x2+1, y);

}

glEnd();

glDisable(GL\_LINE\_STIPPLE);

}

public void DrawAxis()

{

glLineWidth(3);

glColor3d(0, 0, 0);

glBegin(GL\_LINES);

//OX

glVertex2d(\_x1, \_y1-1);

glVertex2d(\_x2, \_y1-1);

//OY

glVertex2d(\_x1-1, \_y1);

glVertex2d(\_x1-1, \_y2);

DrawArrowY();

DrawArrowX();

glEnd();

}

public void DrawArrowY()

{

glLineWidth(3);

glColor3d(0, 0, 0);

glBegin(GL\_LINES);

// Продовження на 1 клітинку

glVertex2d(\_x1-1, \_y2);

glVertex2d(\_x1-1, \_y2+1);

// Ліва половина

glVertex2d(\_x1-1.2, \_y2);

glVertex2d(\_x1-1, \_y2+1);

// Права половина

glVertex2d(\_x1-0.8, \_y2);

glVertex2d(\_x1-1, \_y2+1);

glEnd();

}

public void DrawArrowX()

{

glLineWidth(3);

glColor3d(0, 0, 0);

glBegin(GL\_LINES);

// Продовження на 1 клітинку

glVertex2d(\_x2, \_y1-1);

glVertex2d(\_x2+1, \_y1-1);

// Ліва половина

glVertex2d(\_x2, \_y1-0.8);

glVertex2d(\_x2+1, \_y1-1);

// Права половина

glVertex2d(\_x2, \_y1-1.2);

glVertex2d(\_x2+1, \_y1-1);

glEnd();

}

public void LoadLayout()

{

DrawGrid();

DrawAxis();

}

}

}

}

**Код файлу Figure.cs:**

using System;

using System.Buffers;

using System.Collections.Generic;

using System.Diagnostics;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace PR1\_Petrenko\_program

{

public partial class RenderControl

{

internal class Figure

{

//public Figure() { Debug.WriteLine("Figure created"); }

//int[] var16cordsX = {-3, -2, -1, 0, 0, -2};

//int[] var16cordsY = {2, 4, 4, 3, 1, 0};

private readonly int[] \_coordinatesX;

private readonly int[] \_coordinatesY;

public Figure(int[] coordinatesX, int[] coordinatesY)

{

\_coordinatesX = coordinatesX;

\_coordinatesY = coordinatesY;

}

//public void Draw(uint type, int difference = 0)

//{

// glLineWidth(4);

// glPointSize(8);

// glColor3d(0, 0, 1);

// glBegin(type);

// for (int i = 0; i < \_coordinatesX.Length; i++)

// {

// glVertex2d(\_coordinatesX[i]+difference, \_coordinatesY[i]);

// }

// glEnd();

// glDisable(type);

//}

public void Draw(uint type, int difference = 0, bool complex = false)

{

glLineWidth(4);

glPointSize(8);

glColor3d(0, 0, 1);

glBegin(type);

if (type == GL\_LINES)

{

for (int i = 0; i < \_coordinatesX.Length; i++)

{

glVertex2d(\_coordinatesX[i]+difference, \_coordinatesY[i]);

// % щоб не вийти за межі масива

glVertex2d(\_coordinatesX[(i + 1) % \_coordinatesX.Length]+difference, \_coordinatesY[(i + 1) % \_coordinatesY.Length]);

}

}

else

{

for (int i = 0; i < \_coordinatesX.Length; i++)

{

glVertex2d(\_coordinatesX[i]+difference, \_coordinatesY[i]);

}

}

glEnd();

glDisable(type);

}

public void DrawComplex()

{

Draw(GL\_LINES);

Draw(GL\_POINTS, 5);

}

}

}

}

# ДОДАТОК Б Лістинг програми до практичної роботи №2

**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 static PR2\_Petrenko\_program.OpenGL;

namespace PR2\_Petrenko\_program

{

public partial class RenderControl : OpenGL

{

public uint DrawMode { get; set; } = GL\_FILL;

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

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

Figure \_f = new Figure();

public RenderControl()

{

InitializeComponent();

}

private void OnRender(object sender, EventArgs e)

{

int ortho = 10;

glClear(GL\_COLOR\_BUFFER\_BIT);

glLoadIdentity();

if (Width > Height)

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

else

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

double max = Math.Max(TilesHorizontal, TilesVertical);

glOrtho(-ortho \* max, +ortho \* max, -ortho \* max, +ortho \* max, -1, +1);

for (int row = 0; row < TilesVertical; row++)

{

for (int col = 0; col < TilesHorizontal; col++)

{

double offsetX = col \* 5 \* 1.4;

double offsetY = row \* 5 \* 1.6;

if (col > 0)

offsetY += col \* 2;

\_f.DrawTile(DrawMode, offsetX, offsetY);

}

}

//\_f.DrawYellowTriangle(5, DrawMode);

//\_f.DrawTile(DrawMode, 0, 0);

\_f.DrawCursor();

glDisable(GL\_LINES);

}

}

}

**Код файлу Figure.cs:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace PR2\_Petrenko\_program

{

public partial class RenderControl

{

public class Figure

{

readonly double cursorCenterX = -3;

readonly double cursorCenterY = 0;

public void DrawCursor()

{

glPointSize(5);

glColor3d(0, 0, 1);

glBegin(GL\_POINTS);

glVertex2d(cursorCenterX, cursorCenterY);

glEnd();

glDisable(GL\_POINTS);

}

public void DrawComplexFigure(int sideSize = 5, uint DrawMode = GL\_FILL, double offsetX = 0, double offsetY = 0)

{

//double height = (Math.Sqrt(3) / 2) \* sideSize;

glPointSize(3);

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

glShadeModel(GL\_FLAT);

// YELLOW TRIANGLE

glBegin(GL\_TRIANGLE\_FAN);

glColor3d(1, 1, 0);

glVertex2d(cursorCenterX + offsetX, cursorCenterY + offsetY);

glVertex2d((cursorCenterX + sideSize / 2 + 1) + offsetX, (cursorCenterY + sideSize / 2.5) + offsetY);

glVertex2d((cursorCenterX + sideSize / 2 + 1) + offsetX, (cursorCenterY - sideSize / 2.5) + offsetY);

glEnd();

// RED SQUARE

glBegin(GL\_POLYGON);

glColor3d(1, 0, 0);

glVertex2d(cursorCenterX + sideSize / 2 + 1 + offsetX, cursorCenterY + sideSize / 2.5 + offsetY); // TOP LEFT

glVertex2d(cursorCenterX + sideSize + 2 + offsetX, cursorCenterY + sideSize / 2.5 + offsetY); // TOP RIGHT

glVertex2d(cursorCenterX + sideSize + 2 + offsetX, cursorCenterY - sideSize / 2.5 + offsetY); // BOTTOM RIGHT

glVertex2d(cursorCenterX + sideSize / 2 + 1 + offsetX, cursorCenterY - sideSize / 2.5 + offsetY); // BOTTOM LEFT

glEnd();

// GREEN SQUARE

glBegin(GL\_POLYGON);

glColor3d(0, 0.8, 0);

glVertex2d(cursorCenterX + sideSize / 2 + 1 + offsetX, cursorCenterY - sideSize / 2.5 + offsetY); // TOP LEFT

glVertex2d(cursorCenterX + sideSize + 2 + offsetX, cursorCenterY - sideSize / 2.5 + offsetY); // TOP RIGHT

glVertex2d(cursorCenterX + sideSize + 2 + offsetX, cursorCenterY - sideSize - 1 + offsetY); // BOTTOM RIGHT

glVertex2d(cursorCenterX + sideSize / 2 + 1 + offsetX, cursorCenterY - sideSize - 1 + offsetY); // BOTTOM LEFT

glEnd();

// GREEN, YELLOW, RED TRIANGLES

glBegin(GL\_TRIANGLE\_FAN);

glColor3d(1, 1, 0);

glVertex2d(cursorCenterX + sideSize + 2 + offsetX, cursorCenterY - sideSize / 2.5 + offsetY); // CENTER

glColor3d(1, 0, 0);

glVertex2d(cursorCenterX + sideSize + 2 + offsetX, cursorCenterY - sideSize - 1 + offsetY); // TOP DOWN

glVertex2d(cursorCenterX + 2 \* sideSize + offsetX, cursorCenterY - sideSize + 1 + offsetY); // DOWN

glColor3d(1, 1, 0);

glVertex2d(cursorCenterX + 2 \* sideSize + offsetX, cursorCenterY + offsetY); // UP

glColor3d(0, 0.8, 0);

glVertex2d(cursorCenterX + sideSize + 2 + offsetX, cursorCenterY + sideSize / 2.5 + offsetY); // TOP

glEnd();

}

public void DrawTile(uint DrawMode, double offsetX, double offsetY)

{

DrawComplexFigure(5, DrawMode, offsetX, offsetY);

}

}

}

}

**Код файлу MainForm.cs:**

using System;

using System.Drawing;

using System.Windows.Forms;

using static PR2\_Petrenko\_program.OpenGL;

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

namespace PR2\_Petrenko\_program

{

public partial class MainForm : Form

{

public MainForm()

{

InitializeComponent();

this.MinimumSize = new Size(256, 300);

renderControl1.DrawMode = GL\_FILL;

}

private void rbFillMode\_CheckedChanged(object sender, System.EventArgs e)

{

renderControl1.DrawMode = GL\_FILL;

renderControl1.Invalidate();

}

private void rbLineMode\_CheckedChanged(object sender, System.EventArgs e)

{

renderControl1.DrawMode = GL\_LINE;

renderControl1.Invalidate();

}

private void rbPointMode\_CheckedChanged(object sender, System.EventArgs e)

{

renderControl1.DrawMode = GL\_POINT;

renderControl1.Invalidate();

}

private void nUDVertical\_ValueChanged(object sender, System.EventArgs e)

{

if (nUDVertical.Value <= 0)

{

nUDVertical.Value = 1;

}

else if (nUDVertical.Value >= 1)

{

renderControl1.TilesVertical = Convert.ToInt32(nUDVertical.Value);

renderControl1.Invalidate();

}

}

private void nUDHorizontal\_ValueChanged(object sender, System.EventArgs e)

{

if (nUDHorizontal.Value <= 0)

{

nUDHorizontal.Value = 1;

}

else if (nUDHorizontal.Value >= 1)

{

renderControl1.TilesHorizontal = Convert.ToInt32(nUDHorizontal.Value);

renderControl1.Invalidate();

}

}

}

}

# ДОДАТОК В Лістинг програми до практичної роботи №3

**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 static PR3\_Petrenko\_program.RenderControl;

namespace PR3\_Petrenko\_program

{

public partial class RenderControl : OpenGL

{

Layout \_l = new Layout();

DrawFunction \_dF = new DrawFunction();

public double points = 1000;

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

public double xMin { get; set; } = -3.0;

public double xMax { get; set; } = 3.0;

public double yMin { get; set; } = -3.0;

public double yMax { get; set; } = 3.0;

public int func { get; set; } = 0;

public RenderControl()

{

InitializeComponent();

}

private void OnRender(object sender, EventArgs e)

{

glClearColor(1, 1, 1, 1);

glClear(GL\_COLOR\_BUFFER\_BIT);

glLoadIdentity();

glViewport(0, 0, Width, Height);

if (!autoY)

(yMin, yMax) = \_dF.AutoY(xMin, xMax, yMin, yMax, points, func);

yMax = Math.Min(yMax, 100);

yMin = Math.Max(yMin, -100);

gluOrtho2D(xMin, xMax, yMin, yMax);

\_l.DrawGrid(xMin, xMax, yMin, yMax);

\_l.DrawAxis(xMin, xMax, yMin, yMax);

\_dF.DrawFunc(xMin, xMax, yMin, yMax, points, func);

}

}

}

**Код файлу Layout.cs:**

namespace PR3\_Petrenko\_program

{

public partial class RenderControl

{

public class Layout

{

public void DrawGrid(double xMin, double xMax, double yMin, double yMax)

{

glLineWidth(0.5f);

glColor3d(0.3, 0.3, 0.3);

glBegin(GL\_LINES);

for (double i = xMin; i <= xMax; i++)

{

glVertex2d(i, yMin);

glVertex2d(i, yMax);

}

for (double i = yMin; i <= yMax; i++)

{

glVertex2d(xMin, i);

glVertex2d(xMax, i);

}

glEnd();

}

public void DrawAxis(double xMin, double xMax, double yMin, double yMax)

{

glLineWidth(2);

glColor3d(0.1, 0.1, 0.1);

glBegin(GL\_LINES);

glVertex2d(xMin, 0);

glVertex2d(xMax, 0);

glVertex2d(0, yMin);

glVertex2d(0, yMax);

glEnd();

}

}

}

}

**Код файлу DrawFunction.cs:**

namespace PR3\_Petrenko\_program

{

public partial class RenderControl

{

public class DrawFunction

{

public void DrawFunc(double xMin, double xMax, double yMin, double yMax, double points, int func)

{

glLineWidth(2);

glBegin(GL\_LINE\_STRIP);

double previousY = double.NaN;

double x = xMin;

double step = (xMax - xMin) / (points - 1);

double y = CalculateFunction(x, func);

glVertex2d(x, y);

for (int i = 1; i < points; i++)

{

previousY = y;

x = xMin + i \* step;

y = CalculateFunction(x, func);

if (!double.IsNaN(previousY) && Math.Abs(y - previousY) > 1.0)

{

glEnd();

glBegin(GL\_LINE\_STRIP);

continue;

}

glVertex2d(x, y);

if ((previousY \* y) <= 0 && previousY != 0)

{

glEnd();

DrawPointsOnX(previousY, x, step, y);

glBegin(GL\_LINE\_STRIP);

glColor3d(0, 0, 0);

glVertex2d(x, y);

}

}

glEnd();

}

public (double, double) AutoY(double xMin, double xMax, double yMin, double yMax, double points, int func)

{

double previousY = double.NaN;

double x = xMin;

double step = (xMax - xMin) / (points - 1);

double y = CalculateFunction(x, func);

double min;

double max;

min = max = y;

for (int i = 1; i < points; i++)

{

previousY = y;

x = xMin + i \* step;

y = CalculateFunction(x, func);

if (y > max)

max = y;

if (y < min)

min = y;

}

return (min, max);

}

public double CalculateFunction(double x, int func)

{

switch (func)

{

case 0:

return Math.Cos(2 \* x + 1) - 0.5 \* Math.Sin(5 \* x);

case 1:

double left = Math.Tan((Math.PI \* x) / 3);

double right = Math.Cos(Math.PI \* x) / Math.Abs(Math.Cos(Math.PI \* x));

return left \* right;

default:

return 0;

}

}

private void DrawPointsOnX(double previousY, double x, double step, double y)

{

glPointSize(5);

glColor3d(1, 0, 0);

glBegin(GL\_POINTS);

glVertex2d(x - step / 2, (previousY + y) / 2);

glEnd();

}

}

}

}

**Код файлу MainForm.cs:**

namespace PR3\_Petrenko\_program

{

public partial class MainForm : Form

{

public MainForm()

{

InitializeComponent();

this.MinimumSize = new Size(500, 450);

renderControl1.xMin = (double)nUDXmin.Value;

renderControl1.xMax = (double)nUDXmax.Value;

renderControl1.yMin = (double)nUDYmin.Value;

renderControl1.yMax = (double)nUDYmax.Value;

renderControl1.autoY = false;

}

private void nUDXmin\_ValueChanged(object sender, System.EventArgs e)

{

renderControl1.xMin = (double)nUDXmin.Value;

renderControl1.Invalidate();

}

private void nUDXmax\_ValueChanged(object sender, System.EventArgs e)

{

renderControl1.xMax = (double)nUDXmax.Value;

renderControl1.Invalidate();

}

private void nUDYmin\_ValueChanged(object sender, System.EventArgs e)

{

renderControl1.yMin = (double)nUDYmin.Value;

renderControl1.Invalidate();

}

private void nUDYmax\_ValueChanged(object sender, System.EventArgs e)

{

renderControl1.yMax = (double)nUDYmax.Value;

renderControl1.Invalidate();

}

private void rbF1X\_CheckedChanged(object sender, System.EventArgs e)

{

renderControl1.func = 0;

renderControl1.Invalidate();

}

private void rbF2X\_CheckedChanged(object sender, System.EventArgs e)

{

renderControl1.func = 1;

renderControl1.Invalidate();

}

private void nUDPoints\_ValueChanged(object sender, System.EventArgs e)

{

renderControl1.points = (double)nUDPoints.Value;

renderControl1.Invalidate();

}

private void btnShowY\_Click(object sender, System.EventArgs e)

{

if (!renderControl1.autoY)

{

gbY.Visible = true;

renderControl1.yMin = (double)nUDYmin.Value;

renderControl1.yMax = (double)nUDYmax.Value;

}

else

{

gbY.Visible = false;

renderControl1.yMin = -3.0;

renderControl1.yMax = 3.0;

}

renderControl1.autoY = !renderControl1.autoY;

renderControl1.Invalidate();

}

}

}

# ДОДАТОК Г Лістинг програми до практичної роботи №4

**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.Windows.Forms;

namespace PR4\_Petrenko\_program

{

public partial class RenderControl : OpenGL

{

Layout \_l = new Layout();

DrawFunction \_dF = new DrawFunction();

public double margin { get; set; } = 5.0;

public int func { get; set; } = 0;

public double A { get; set; } = 1;

public double B { get; set; } = 0;

public double C { get; set; } = 0;

public double XStart { get; set; } = 0;

public double YStart { get; set; } = 0;

public double XEnd { get; set; } = 0;

public double YEnd { get; set; } = 0;

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

public RenderControl()

{

InitializeComponent();

this.MouseClick += OnMouseClick;

}

private void OnRender(object sender, EventArgs e)

{

glClearColor(1, 1, 1, 1);

glClear(GL\_COLOR\_BUFFER\_BIT);

glLoadIdentity();

if (Width > Height)

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

else

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

gluOrtho2D(-margin, margin, -margin, margin);

\_l.DrawGrid(-margin, margin, -margin, margin);

\_l.DrawAxis(-margin, margin, -margin, margin);

if (func == 0)

\_dF.DrawHyperbola(A, B);

if (func == 1)

{

\_dF.DrawParabola(A, B, C);

DrawLine = false;

XStart = 0; YStart = 0; XEnd = 0; YEnd = 0;

}

if (DrawLine)

{

// Малюємо пряму

glColor3d(1, 0, 0);

glBegin(GL\_LINES);

glVertex2d(XStart, YStart);

glVertex2d(XEnd, YEnd);

glEnd();

// Малюємо точки перетину

DrawIntersectionPoints(A, B, XStart, YStart, XEnd, YEnd);

}

}

private void OnMouseClick(object sender, MouseEventArgs e)

{

double viewportWidth, viewportHeight;

double xOffset = 0, yOffset = 0;

if (Width > Height)

{

viewportWidth = (double)Height;

viewportHeight = (double)Height;

xOffset = (Width - Height) / 2.0;

}

else

{

viewportWidth = (double)Width;

viewportHeight = (double)Width;

yOffset = (Height - Width) / 2.0;

}

double normalizedX = (e.X - xOffset) / viewportWidth \* 2.0 - 1.0;

double normalizedY = -((e.Y - yOffset) / viewportHeight \* 2.0 - 1.0);

double x = normalizedX \* margin;

double y = normalizedY \* margin;

if (e.Button == MouseButtons.Left)

{

XStart = x;

YStart = y;

}

else if (e.Button == MouseButtons.Right)

{

XEnd = x;

YEnd = y;

DrawLine = true;

}

Invalidate();

}

private void DrawIntersectionPoints(double A, double B, double x1, double y1, double x2, double y2)

{

// Параметричні рівняння

double dx = x2 - x1;

double dy = y2 - y1;

// Рівняння прямої у рівнянні гіперболи

double a = Math.Pow(dx / A, 2) - Math.Pow(dy / B, 2);

double b = 2 \* ((x1 \* dx) / (A \* A) - (y1 \* dy) / (B \* B));

double c = Math.Pow(x1 / A, 2) - Math.Pow(y1 / B, 2) - 1;

double discriminant = b \* b - 4 \* a \* c;

if (discriminant < 0)

return;

double t1 = (-b - Math.Sqrt(discriminant)) / (2 \* a);

double t2 = (-b + Math.Sqrt(discriminant)) / (2 \* a);

List<(double x, double y)> intersectionPoints = new List<(double x, double y)>();

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

intersectionPoints.Add((x1 + t1 \* dx, y1 + t1 \* dy));

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

intersectionPoints.Add((x1 + t2 \* dx, y1 + t2 \* dy));

glColor3d(0, 0.6, 0);

foreach (var point in intersectionPoints)

{

glPointSize(8);

glBegin(GL\_POINTS);

glVertex2d(point.x, point.y);

glEnd();

}

}

}

}

**Код файлу DrawFigure.cs:**

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace PR4\_Petrenko\_program

{

public partial class RenderControl

{

public class DrawFunction

{

const int margins = 100;

public void DrawParabola(double a, double b, double c)

{

glColor3d(0, 0, 1);

glBegin(GL\_LINE\_STRIP);

for (double x = -margins; x <= margins; x += 0.1)

{

double y = a \* x \* x + b \* x + c;

glVertex2d(x, y);

}

glEnd();

}

public void DrawHyperbola(double a, double b)

{

glColor3d(0, 0, 1);

glBegin(GL\_LINE\_STRIP);

for (double t = -margins; t <= margins; t += 0.1)

{

double x = a \* Math.Cosh(t);

double y = b \* Math.Sinh(t);

glVertex2d(x, y);

}

glEnd();

glBegin(GL\_LINE\_STRIP);

for (double t = -margins; t <= margins; t += 0.1)

{

double x = -a \* Math.Cosh(t);

double y = -b \* Math.Sinh(t);

glVertex2d(x, y);

}

glEnd();

}

}

}

}

**Код файлу MainForm.cs:**

using System.Drawing;

using System.Windows.Forms;

using static PR4\_Petrenko\_program.OpenGL;

namespace PR4\_Petrenko\_program

{

public partial class MainForm : Form

{

public MainForm()

{

InitializeComponent();

this.MinimumSize = new Size(500, 400);

}

private void rbHyp\_CheckedChanged(object sender, System.EventArgs e)

{

renderControl1.func = 0;

nudC.Visible = false;

lblC.Visible = false;

renderControl1.Invalidate();

}

private void rbPar\_CheckedChanged(object sender, System.EventArgs e)

{

renderControl1.func = 1;

nudC.Visible = true;

lblC.Visible = true;

renderControl1.Invalidate();

}

private void nudA\_ValueChanged(object sender, System.EventArgs e)

{

renderControl1.A = (double)nudA.Value;

renderControl1.Invalidate();

}

private void nudB\_ValueChanged(object sender, System.EventArgs e)

{

renderControl1.B = (double)nudB.Value;

renderControl1.Invalidate();

}

private void nudC\_ValueChanged(object sender, System.EventArgs e)

{

renderControl1.C = (double)nudC.Value;

renderControl1.Invalidate();

}

}

}

# ДОДАТОК Д Лістинг програми до практичної роботи №5

**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.Drawing.Printing;

using System.Linq;

using System.Windows.Forms;

namespace PR5\_Petrenko\_program

{

public partial class RenderControl : OpenGL

{

double size = 10.0;

double modifier = 5.0;

double AspectRatio { get => ((double)Width / Height); }

public int ViewMode { get; set; } = 0;

public uint fillMode { get; set; } = GL\_LINE;

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

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

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

double xMin { get => (AspectRatio > 1) ? -size \* AspectRatio : -size; }

double xMax { get => (AspectRatio > 1) ? +size \* AspectRatio : +size; }

double yMin { get => (AspectRatio < 1) ? -size / AspectRatio : -size; }

double yMax { get => (AspectRatio < 1) ? +size / AspectRatio : +size; }

double zMin { get => -size; }

double zMax { get => +size; }

double ax = 10, ay = 20;

Layout \_l = new Layout();

DrawFigure \_dF = new DrawFigure();

public RenderControl()

{

InitializeComponent();

\_l.Text = DrawText;

}

private void OnRender(object sender, EventArgs e)

{

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

glEnable(GL\_DEPTH\_TEST);

glLoadIdentity();

glMatrixMode(GL\_PROJECTION); //

if (Width > Height)

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

else

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

//glOrtho(xMin \* modifier, xMax \* modifier, yMin \* modifier, yMax \* modifier, zMin \* modifier, zMax \* modifier);

if (ViewMode == 0)

{

glLoadIdentity();

glOrtho(-size \* modifier, size \* modifier, -size \* modifier, size \* modifier, -size \* modifier, size \* modifier);

}

else

{

glLoadIdentity();

gluPerspective(30.0 \* modifier, AspectRatio, 0.1, 100.0);

}

glMatrixMode(GL\_MODELVIEW); //

glRotated(ax, 1, 0, 0);

glRotated(ay, 0, 1, 0);

\_l.DrawGrid();

\_l.DrawAxis();

if (isLight)

SetupLighting();

else

DisableLighting();

if (!useDrawList)

{

\_dF.DrawSphere(4.0, -1.5, 2.0, 2.0, fillMode, clipPlane);

\_dF.DrawCone(-3.5, +0.5, 2.0, 0.0, 1.0, 2.5, fillMode);

\_dF.DrawDisk(-2.5, -1.5, -3.5, 3.5, 1.0, 0.0, 45.0, fillMode);

}

else

{

DrawList();

glCallList(SphereDL);

glCallList(ConeDL);

glCallList(DiskDL);

}

}

private void SetupLighting()

{

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

float[] lightPosition = { 3.0f, 3.0f, 3.0f, 5.0f };

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

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

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

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

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

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

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

glEnable(GL\_COLOR\_MATERIAL);

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

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

float materialShininess = 30.0f;

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

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

}

private void DisableLighting()

{

glDisable(GL\_LIGHTING);

glDisable(GL\_LIGHT0);

glDisable(GL\_COLOR\_MATERIAL);

}

private uint SphereDL;

private uint ConeDL;

private uint DiskDL;

public void DrawList()

{

SphereDL = glGenLists(1);

glNewList(SphereDL, GL\_COMPILE);

\_dF.DrawSphere(4.0, -1.5, 2.0, 2.0, GL\_LINE, clipPlane);

glEndList();

ConeDL = glGenLists(1);

glNewList(ConeDL, GL\_COMPILE);

\_dF.DrawCone(-3.5, +0.5, 2.0, 0.0, 1.0, 2.5, GL\_LINE);

glEndList();

DiskDL = glGenLists(1);

glNewList(DiskDL, GL\_COMPILE);

\_dF.DrawDisk(-2.5, -1.5, -3.5, 3.5, 1.0, 0.0, 45.0, GL\_LINE);

glEndList();

}

bool flag = false;

Point start;

private void OnDown(object sender, MouseEventArgs e)

{

flag = e.Button == MouseButtons.Left;

start = e.Location;

}

private void OnUp(object sender, MouseEventArgs e)

{

if (flag)

{

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

}

}

private void OnMove(object sender, MouseEventArgs e)

{

if (flag)

{

Point current = e.Location;

ax += (current.Y - start.Y) / 2;

ay += (current.X - start.X) / 2;

start = current;

Invalidate();

}

}

private void OnWheel(object sender, MouseEventArgs e)

{

if (e.Delta < 0)

{

modifier += 0.1;

}

else

{

if (modifier > 0.2)

{

modifier -= 0.1;

}

}

Invalidate();

}

}

}

**Код файлу Layout.cs:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace PR5\_Petrenko\_program

{

public partial class RenderControl

{

public class Layout

{

public delegate void outText(string s, double x, double y, double z);

public outText Text;

public void DrawAxis()

{

glDisable(GL\_LIGHTING);

glDisable(GL\_LIGHT0);

glLineWidth(3);

glBegin(GL\_LINES);

glColor3d(1, 0, 0);

glVertex3d(0, 0, 0);

glVertex3d(1.0, 0.0, 0.0);

glColor3d(0, 1, 0);

glVertex3d(0, 0, 0);

glVertex3d(0.0, 1.0, 0.0);

glColor3d(0, 0, 1);

glVertex3d(0, 0, 0);

glVertex3d(0.0, 0.0, 1.0);

glEnd();

glColor3d(1, 1, 1);

Text("+X", 1, 0, 0);

Text("+Y", 0, 1, 0);

Text("+Z", 0, 0, 1);

}

public void DrawGrid()

{

glLineWidth(1);

double size = 15;

double step = 1;

glColor3d(0.7, 0.7, 0.7);

glBegin(GL\_LINES);

for (double i = -size; i <= size; i += size / 10)

{

glVertex3d(i, 0, -size);

glVertex3d(i, 0, size);

}

for (double j = -size; j <= size; j += size / 10)

{

glVertex3d(-size, 0, j);

glVertex3d(size, 0, j);

}

glEnd();

}

}

}

}

**Код файлу DrawFigure.cs:**

using System;

using System.Collections.Generic;

using System.Configuration;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace PR5\_Petrenko\_program

{

public partial class RenderControl

{

public class DrawFigure

{

private int \_slices = 15;

private int \_stacks = 15;

public void DrawSphere(double x0, double y0, double z0, double R, uint mode, bool clipPlane)

{

int slices = \_slices;

int stacks = \_stacks;

glColor3d(1, 0, 0);

glPushMatrix();

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

glTranslated(x0, y0, z0);

if (clipPlane)

{

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

{

double lat0 = Math.PI \* (-0.5 + (double)i / stacks);

z0 = R \* Math.Sin(lat0);

if (z0 < 0) continue;

double zr0 = R \* Math.Cos(lat0);

double lat1 = Math.PI \* (-0.5 + (double)(i + 1) / stacks);

double z1 = R \* Math.Sin(lat1);

if (z1 < 0) continue;

double zr1 = R \* Math.Cos(lat1);

glBegin(GL\_QUAD\_STRIP);

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

{

double lng = 2 \* Math.PI \* (double)(j - 1) / slices;

double x = Math.Cos(lng);

double y = Math.Sin(lng);

glNormal3d(x \* zr0, y \* zr0, z0);

glVertex3d(x \* zr0, y \* zr0, z0);

glNormal3d(x \* zr1, y \* zr1, z1);

glVertex3d(x \* zr1, y \* zr1, z1);

}

glEnd();

}

}

else

{

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

{

double lat0 = Math.PI \* (-0.5 + (double)i / stacks);

z0 = R \* Math.Sin(lat0);

double zr0 = R \* Math.Cos(lat0);

double lat1 = Math.PI \* (-0.5 + (double)(i + 1) / stacks);

double z1 = R \* Math.Sin(lat1);

double zr1 = R \* Math.Cos(lat1);

glBegin(GL\_QUAD\_STRIP);

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

{

double lng = 2 \* Math.PI \* (double)(j - 1) / slices;

double x = Math.Cos(lng);

double y = Math.Sin(lng);

glNormal3d(x \* zr0, y \* zr0, z0);

glVertex3d(x \* zr0, y \* zr0, z0);

glNormal3d(x \* zr1, y \* zr1, z1);

glVertex3d(x \* zr1, y \* zr1, z1);

}

glEnd();

}

}

glPopMatrix();

}

public void DrawCone(double x0, double y0, double z0, double R, double r, double h, uint mode)

{

int slices = \_slices;

glColor3d(0, 1, 0);

glPushMatrix();

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

glTranslated(x0, y0, z0);

glBegin(GL\_TRIANGLE\_FAN);

glVertex3d(h, 0.0, 0.0);

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

{

double angle = 2.0 \* Math.PI \* i / slices;

double y = r \* Math.Sin(angle);

double z = r \* Math.Cos(angle);

glVertex3d(0.0, y, z);

}

glEnd();

glBegin(GL\_TRIANGLE\_FAN);

glVertex3d(0.0, 0.0, 0.0);

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

{

double angle = 2.0 \* Math.PI \* i / slices;

double y = r \* Math.Sin(angle);

double z = r \* Math.Cos(angle);

glVertex3d(0.0, y, z);

}

glEnd();

glPopMatrix();

}

public void DrawDisk(double x0, double y0, double z0, double r, double R, double start, double sweep, uint mode)

{

int slices = \_slices;

glColor3d(0, 0, 1);

IntPtr quadric = gluNewQuadric();

if (mode == GL\_LINE)

{

gluQuadricDrawStyle(quadric, GLU\_LINE);

}

else

{

gluQuadricDrawStyle(quadric, GLU\_FILL);

}

glPushMatrix();

glTranslated(x0, y0, z0);

gluPartialDisk(quadric, R, r, slices, 1, start, sweep);

glPopMatrix();

gluDeleteQuadric(quadric);

}

}

}

}

**Код файлу MainForm.cs:**

using System.Drawing;

using System.Windows.Forms;

using static PR5\_Petrenko\_program.OpenGL;

namespace PR5\_Petrenko\_program

{

public partial class MainForm : Form

{

public MainForm()

{

InitializeComponent();

this.MinimumSize = new Size(550, 400);

}

private void rbOrtho\_CheckedChanged(object sender, System.EventArgs e)

{

renderControl1.ViewMode = 0;

renderControl1.Invalidate();

}

private void rbPerspective\_CheckedChanged(object sender, System.EventArgs e)

{

renderControl1.ViewMode = 1;

renderControl1.Invalidate();

}

private void rbLineMode\_CheckedChanged(object sender, System.EventArgs e)

{

renderControl1.fillMode = GL\_LINE;

renderControl1.Invalidate();

}

private void rbFillMode\_CheckedChanged(object sender, System.EventArgs e)

{

renderControl1.fillMode = GL\_FILL;

renderControl1.Invalidate();

}

private void cbClip\_CheckedChanged(object sender, System.EventArgs e)

{

renderControl1.clipPlane = cbClip.Checked;

renderControl1.Invalidate();

}

private void cbLight\_CheckedChanged(object sender, System.EventArgs e)

{

renderControl1.isLight = cbLight.Checked;

renderControl1.Invalidate();

}

private void cbDrawList\_CheckedChanged(object sender, System.EventArgs e)

{

renderControl1.useDrawList = cbDrawList.Checked;

renderControl1.Invalidate();

}

}

}