МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського

«Харківський авіаційний інститут»

Факультет програмної інженерії та бізнесу

Кафедра інженерії програмного забезпечення

Практичні роботи

Minor *«Розробник ігрових додатків»*

дисципліна *«Комп’ютерна графіка з OpenGL»*

(назва дисципліни)

Виконав: студент 3 курсу групи  *631п*

напряму підготовки (спеціальності):

*121 інженерія програмного забезпечення*

(шифр і назва напряму підготовки / спеціальності)

*Бахшалієв А.Е.*

(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](#_Toc186021012)

[Завдання, варіант № 2 3](#_Toc186021013)

[Системна інформація 3](#_Toc186021014)

[Теоретичні відомості 3](#_Toc186021015)

[Вершинні масиви 3](#_Toc186021016)

[Команда glDrawArrays() 4](#_Toc186021017)

[Команда glDrawElements() 5](#_Toc186021018)

[Результати виконання практичної роботи 5](#_Toc186021019)

[Розв'язання завдання 5](#_Toc186021020)

[Контроль виконання вимог та елементів завдання 6](#_Toc186021021)

[Практична робота 2. Назва роботи 7](#_Toc186021022)

[Завдання, варіант № 2 7](#_Toc186021023)

[Системна інформація 7](#_Toc186021024)

[Теоретичні відомості 7](#_Toc186021025)

[Основні типи графічних примітивів: 8](#_Toc186021026)

[Команди для роботи з графічними примітивами в OpenGL: 8](#_Toc186021027)

[Способи малювання примітивів: 8](#_Toc186021028)

[Управління масштабом та координатною системою: 9](#_Toc186021029)

[Результати виконання практичної роботи 9](#_Toc186021030)

[Розв'язання завдання 9](#_Toc186021031)

[Контроль виконання вимог та елементів завдання 11](#_Toc186021032)

[Практична робота 3. Графік функції однієї змінної 13](#_Toc186021033)

[Завдання, варіант № 2 13](#_Toc186021034)

[Теоретичні відомості 13](#_Toc186021035)

[Результати виконання практичної роботи 14](#_Toc186021036)

[Розв'язання завдання 14](#_Toc186021037)

[Контроль виконання вимог та елементів завдання 15](#_Toc186021038)

[Практична робота 4. Криві другого порядку 17](#_Toc186021039)

[Завдання, варіант № 2 17](#_Toc186021040)

[Теоретичні відомості 17](#_Toc186021041)

[Результати виконання практичної роботи 17](#_Toc186021042)

[Контроль виконання вимог та елементів завдання 18](#_Toc186021043)

[Загальний перелік посилань 19](#_Toc186021044)

[Додаток А. Лістинг програми до практичної роботи №1 20](#_Toc186021045)

[Код файлу (MainForm.cs) 20](#_Toc186021046)

[Код файлу (Figures.cs) 20](#_Toc186021047)

[Додаток Б. Лістинг програми до практичної роботи №2 22](#_Toc186021048)

[Код файлу (MainForm.cs) 22](#_Toc186021049)

[Код файлу (RenderControl.cs) 23](#_Toc186021050)

[Код файлу (Draw.cs) 24](#_Toc186021051)

[Код файлу (FigureMode.cs) 27](#_Toc186021052)

[Додаток С. Лістинг програми до практичної роботи №3 28](#_Toc186021053)

[Код файлу (MainForm.cs) 28](#_Toc186021054)

[Код файлу (RenderControl.cs) 29](#_Toc186021055)

[Код файлу (Draw.cs) 30](#_Toc186021056)

[Додаток Г. Лістинг програми до практичної роботи №4 33](#_Toc186021057)

[Код файлу (MainForm.cs) 33](#_Toc186021058)

[Код файлу (RenderControl.cs) 33](#_Toc186021059)

[Код файлу (Draw.cs) 33](#_Toc186021060)

# Практична робота 1. Основні принципи роботи з OpenGL

## Завдання, варіант № 2

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.

|  |  |  |
| --- | --- | --- |
| 2 | Примітиви:  GL\_POINTS, GL\_LINE\_LOOP  x1 = -8; x2 = 1  y1 = -3; y2 = 1 | W001-03 |

## Системна інформація

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

Processor AMD Ryzen 5 5600 6-Core Processor @ 3.50GHz 3.50GHz

RAM 32.0 GB (31.9 GB usable)

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

Edition Windows 11 Version 23H2

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

## Теоретичні відомості

### Вершинні масиви

Використання вершинних масивів зменшує кількість викликів функцій та надмірне використання спільних вершин. Таким чином, ви можете збільшити продуктивність рендерингу[[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 27 у файлі MyForm.cs;
* товщина glLineWidth(), рядок 41 у файлі Figures.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 28 – 55 файлу Figures.cs

Застосування циклів для створення зображень наведено у рядках 15 – 24 файлу Figures.cs.

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

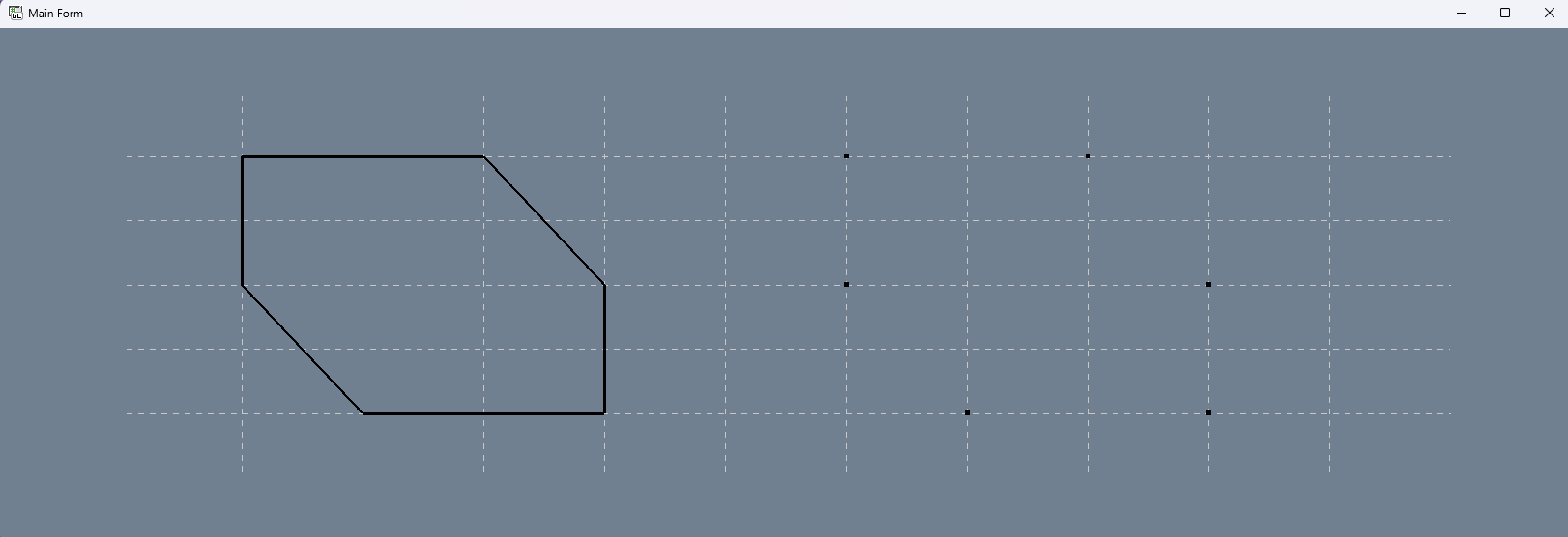


Рисунок 1.1 – Тестування програми при зміні ширини вікна

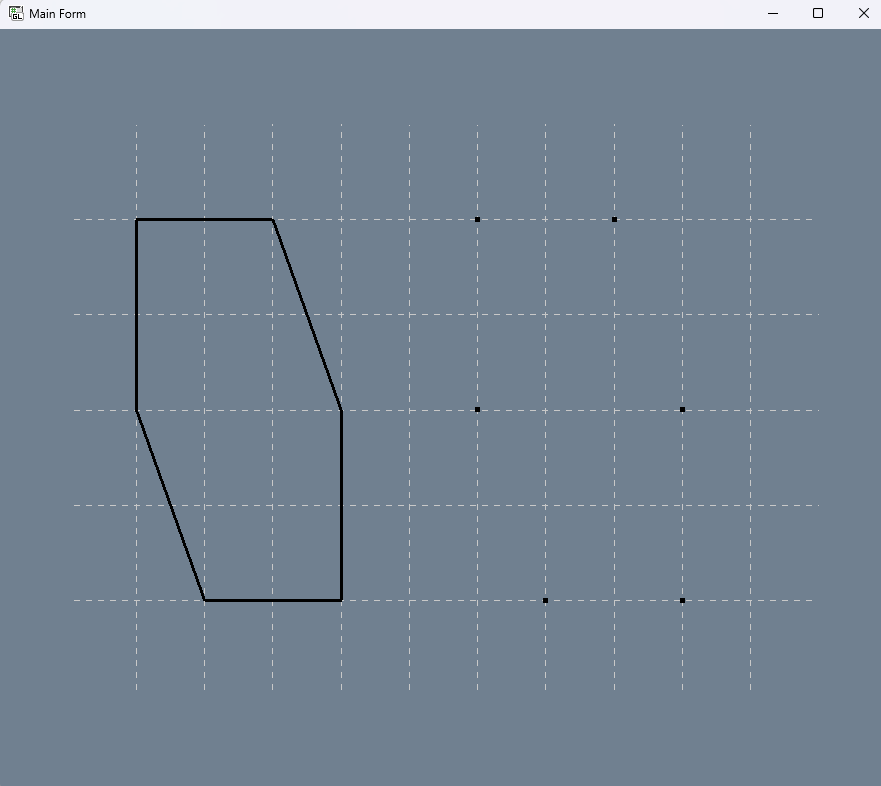


Рисунок 1.2 – Тестування програми при зміні висоти вікна

### Контроль виконання вимог та елементів завдання

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

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

# Практична робота 2. Назва роботи

## Завдання, варіант № 2

Використовуючи інструментальні засоби, що вказані викладачем, і беручи до уваги вимоги, створити програмний проєкт з підтримкою OpenGL. За допомогою команд glOrtho / gluOrtho2D і glViewport встановити для робочої області ізотропну систему координат з урахуванням розміру фігури, яку задано у варіанті. Після старту застосунок повинен відображати у робочій області одну плитку. Усі варіанти заданій основані на правильних багатокутниках, розмір яких визначається величиною одного ребра. Для зафарбування пропонується використовувати шість кольорів: білий, сірий (35 %), червоний, зелений, синій и жовтий. За допомогою клавіатури або маніпулятора «миша» користувач повинен мати можливість виконати замощення (tessellation, tilling) робочої області по горизонталі і вертикалі. При цьому систему координат необхідно скорегувати таким чином, щоб замощена поверхня розташовувалася у центрі робочої області. Крім цього, користувач повинен мати можливість змінювати режим відображення графічних примітивів OpenGL: точкове (тільки вершини фігури), контурне і з заповненням кольором. Передбачається, що перемикання між режимами виконують за подією від клавіатури і/або маніпулятора «миша». При цьому можна використовувати як стандартні елементи керування, так і власні, що реалізовані та відображені засобами OpenGL.

|  |  |  |
| --- | --- | --- |
| 2 | Сторона фігури a = 4.25  Примітиви:   GL\_TRIANGLES,  GL\_QUADS |  |

## Системна інформація

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

Processor AMD Ryzen 5 5600 6-Core Processor @ 3.50GHz 3.50GHz

RAM 32.0 GB (31.9 GB usable)

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

Edition Windows 11 Version 23H2

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

## Теоретичні відомості

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

### Основні типи графічних примітивів:

* GL\_POINTS (Точки):

Використовуються для відображення окремих точок на екрані.

Кожна вершина, визначена у вершинному масиві, є точкою.

* GL\_LINES (Лінії):

Дві вершини створюють лінію.

Можна використовувати для побудови лінійних сегментів між точками.

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

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

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

* GL\_TRIANGLES (Трикутники):

Найбільш використовуваний примітив для відображення поверхонь.

Кожен трикутник визначається трьома вершинами.

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

* GL\_QUADS (Чотирикутники):

Використовується для створення заповнених чотирикутних поверхонь. Чотирикутник визначається точно чотирма вершинами, які задаються у правильному порядку (або за годинниковою, або проти годинникової стрілки).

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

* glBegin() і glEnd():

Команди, які огортають блоки коду, що описують примітиви.

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

* glVertex():

Визначає координати вершин для кожного примітиву.

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

* glColor():

Додає колір до вершин або примітивів.

### Способи малювання примітивів:

* glDrawArrays():

Команда, яка дозволяє малювати примітиви, використовуючи масиви вершин.

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

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

* glDrawElements():

Дозволяє малювати примітиви, використовуючи індексовані масиви вершин.

Дозволяє уникати дублювання спільних вершин для складних фігур.

### Управління масштабом та координатною системою:

* glOrtho() і gluOrtho2D():

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

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

Визначає частину вікна, на якій буде відображатися зображення.

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

## Результати виконання практичної роботи

### Розв'язання завдання

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

Налаштування координатної системи:

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

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

Відображення багатокутника:

Для відображення правильного багатокутника використано примітиви GL\_TRIANGLE\_STRIP та GL\_QUADS. Після старту програми у робочій області відображається одна плитка. Розмір плитки визначено згідно з варіантом, де сторона фігури дорівнює 4.25.

Реалізовано три режими відображення фігур:

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

Колірна схема:

Для зафарбування фігур використано 4 кольори: сірий, червоний, синій та жовтий. Фарба накладається відповідно до положення багатокутників на екрані.

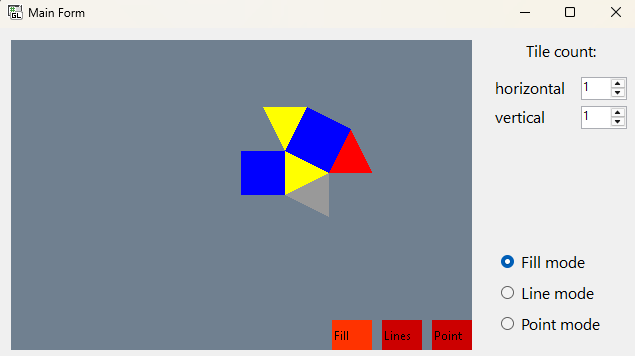


Рисунок 2.1 – Тестування програми при запуску

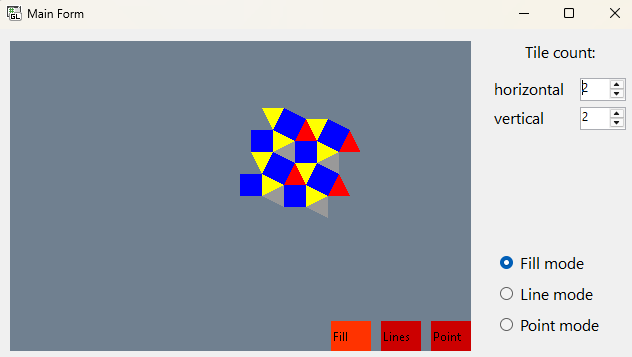


Рисунок 2.2 – Тестування програми при додаванні плиток

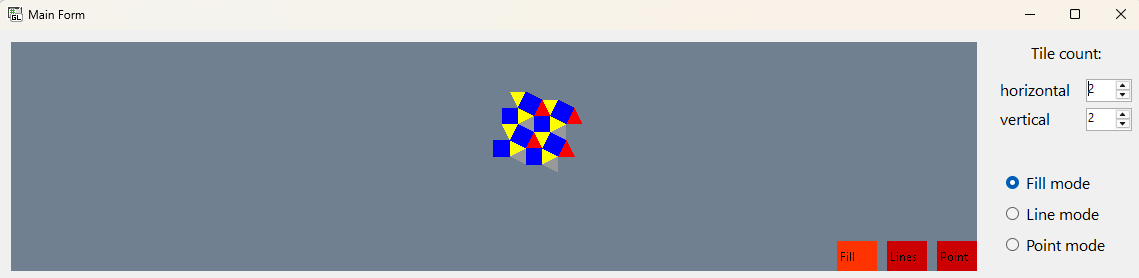


Рисунок 2.3 – Тестування програми при зміні ширини вікна

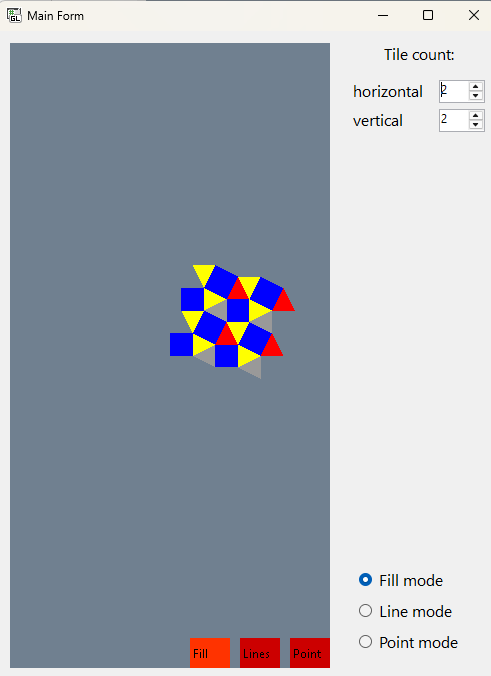


Рисунок 2.4 – Тестування програми при зміні висоти вікна

### Контроль виконання вимог та елементів завдання

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

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

# Практична робота 3. Графік функції однієї змінної

## Завдання, варіант № 2

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

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

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

|  |  |
| --- | --- |
| 2 | Функція |
|  |
|  |

## Теоретичні відомості

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

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

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

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

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

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

## Результати виконання практичної роботи

### Розв'язання завдання

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

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



Рисунок – Тестування роботи програми

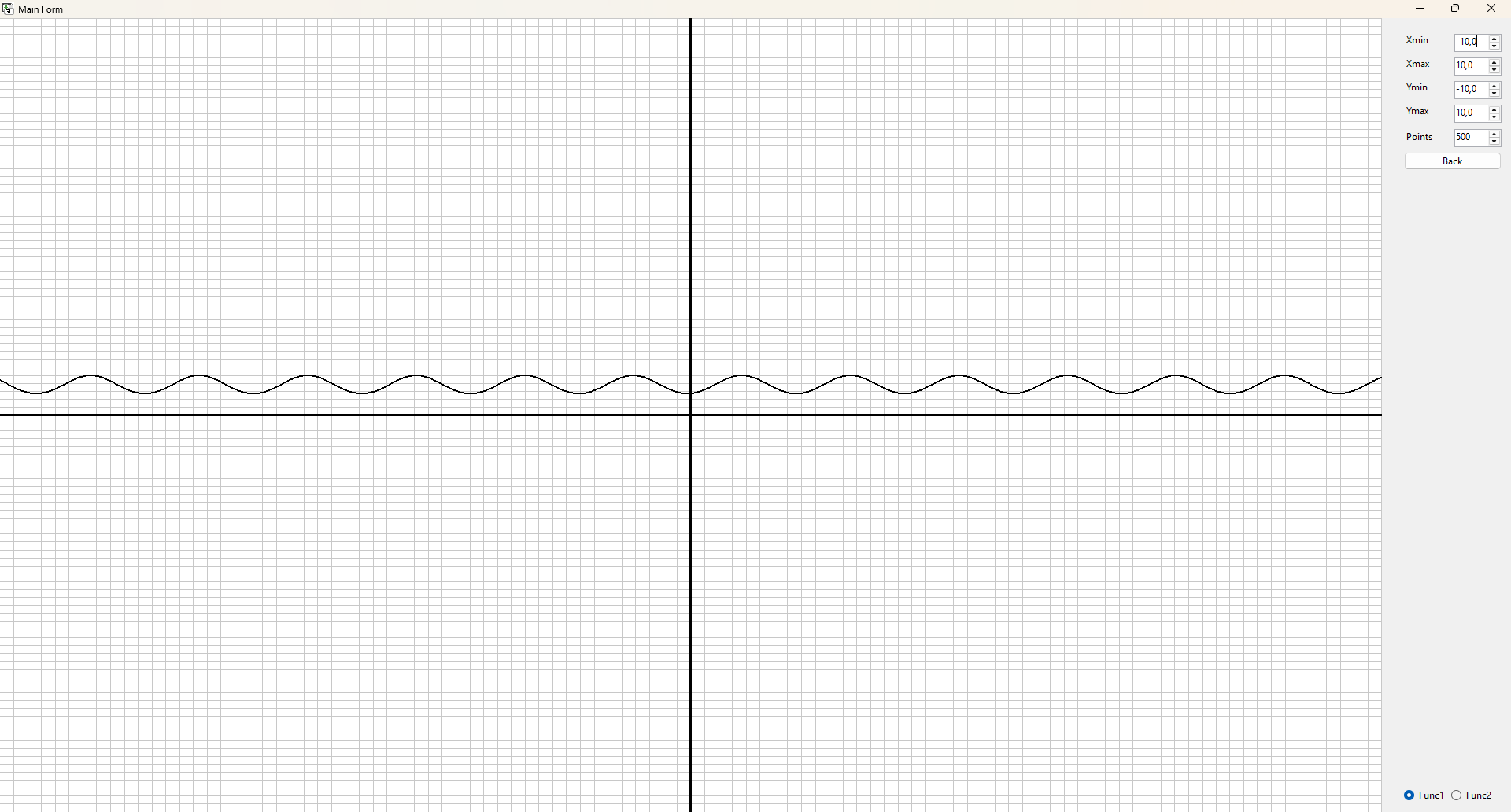


Рисунок 2 – Тестування роботи програми, зміна параметрів і розміру вікна

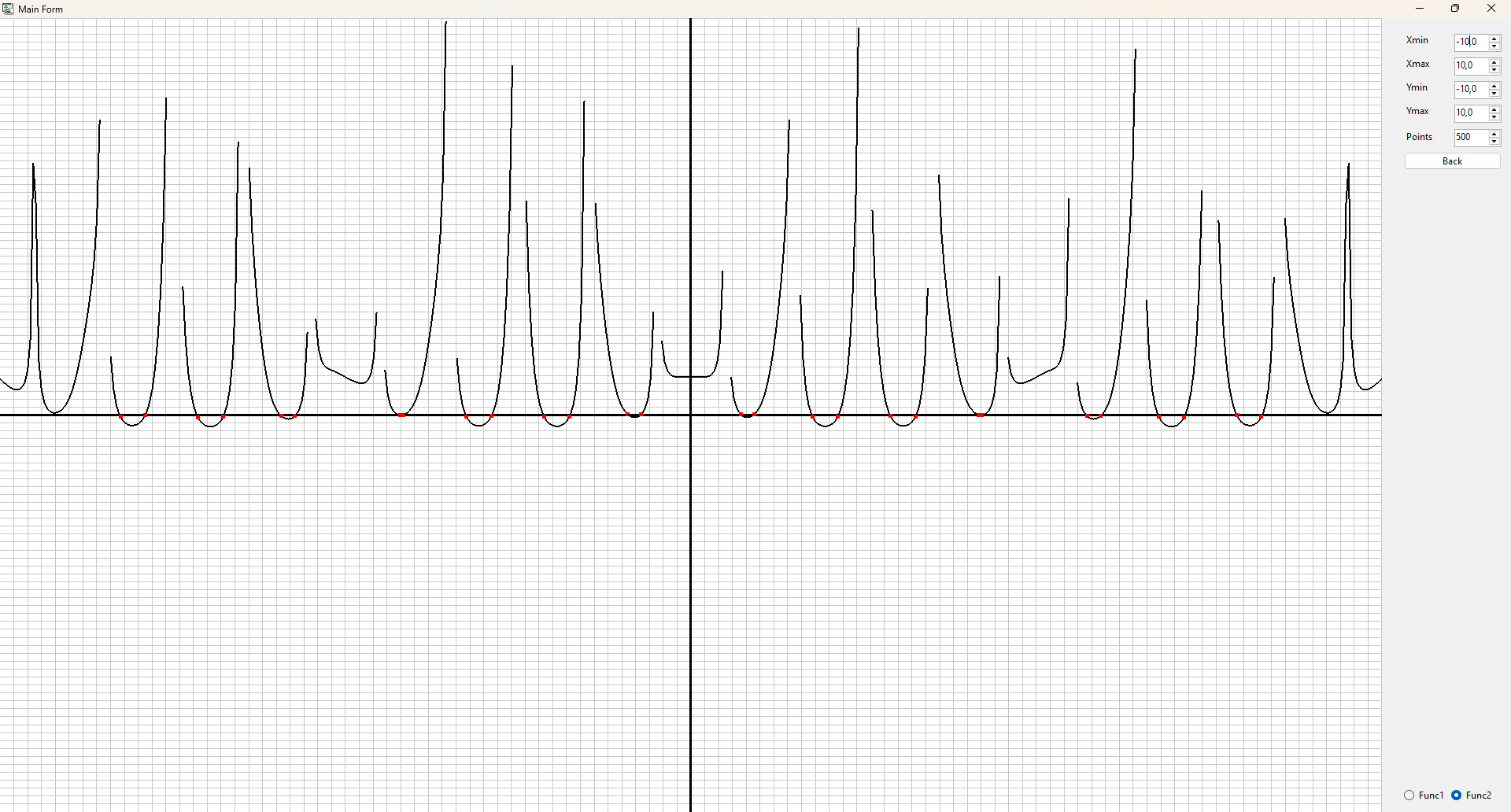


Рисунок 3 – Тестування роботи програми, зміна функції і розміру вікна

### 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 3.1.

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

# Практична робота 4. Криві другого порядку

## Завдання, варіант № 2

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.

|  |  |
| --- | --- |
| 2 | Еліпс «++» – параметричне подання; |
| Парабола «+» – параметричне подання. |

## Теоретичні відомості

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

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

## Результати виконання практичної роботи

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

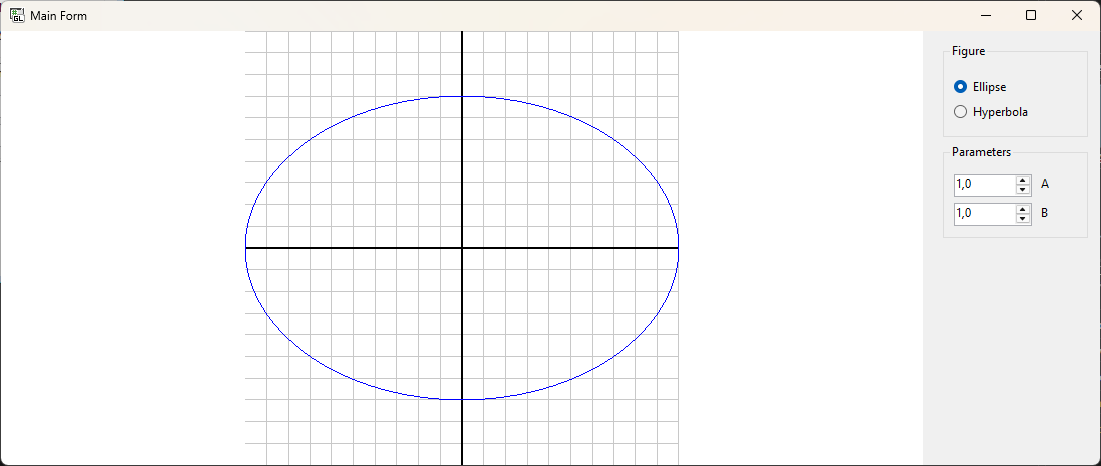


Рисунок 4.1 – Тестування роботи програми з еліпсом та шириною більше за висоту

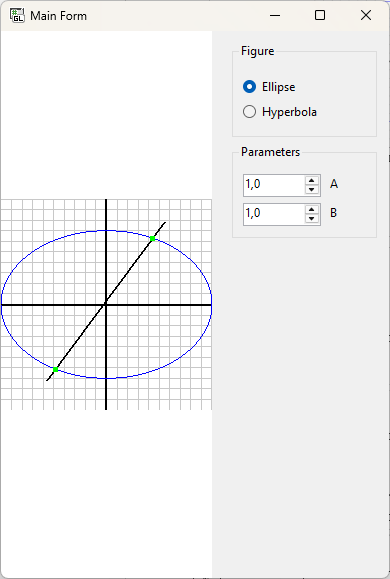


Рисунок 4.2 – Тестування роботи програми з еліпсом та шириною менше за висоту

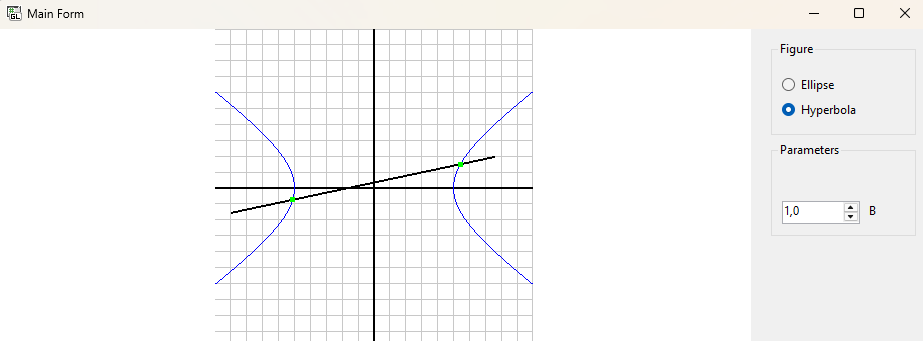


Рисунок 4.3 – Тестування роботи програми з гіперболою та шириною більше за висоту

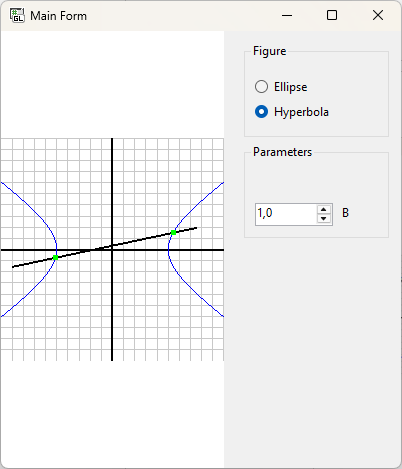


Рисунок 4.4 – Тестування роботи програми з гіперболою та шириною менше за висоту

### Контроль виконання вимог та елементів завдання

В результаті виконання практичної роботи були повністю виконані елементи базового рівня та підвищеного рівня складності, що відображено в таблиці 4.1.

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

# Практична робота 5. Афінні перетворення у просторі

## Завдання, варіант № 2

Використовуючи інструментальні засоби, вказані викладачем, розробити програму з використання засобів OpenGL, яка встановлює ізотропну систему координат, створює і виводить зображення тривимірної сцени з такими елементами:

- осі координат з нулем у центрі екрана та вказанням осі та додатного напрямку;

- координатна сітка (grid) в одній з площин (X0Y, X0Z чи Y0Z);

- три квадратичні фігури – gluDisk / gluPartialDisk, gluSphere,gluCylinder в режимі відображення каркаса і з спрощеною моделлю освітлення glEnable(GL\_COLOR\_MATERIAL) для базового рівня складності;

- площина відтину для однієї з фігур (сфера, циліндр чи конус);

- повноцінна модель освітлення та/або текстурами для реалізації завдання з підвищеною складністю.

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

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | X0Z | Сфера | ∥0Y | -4.0 | -2.5 | -4.5 | 3.0 | - | - |
| Усічений конус | ⇅0X | -4.5 | +2.5 | +2.5 | 2.5 | 0.5 | 2.0 |
| Диск | ∥0Z | +3.5 | -1.0 | +3.0 | 2.0 | 0.0 | - |

## Теоретичні відомості

Ізотропну систему координат можна встановлювати двома способами.

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

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

## Результати виконання практичної роботи

### Розв'язання завдання

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

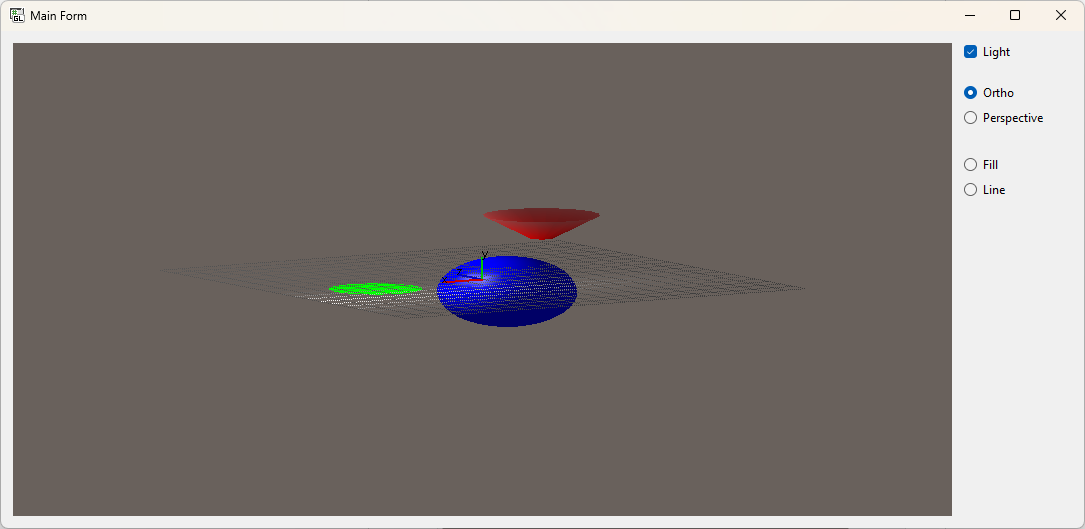


Рисунок 5.1 – Тестування роботи програми в ортогональному відображенні з заповненням фігур

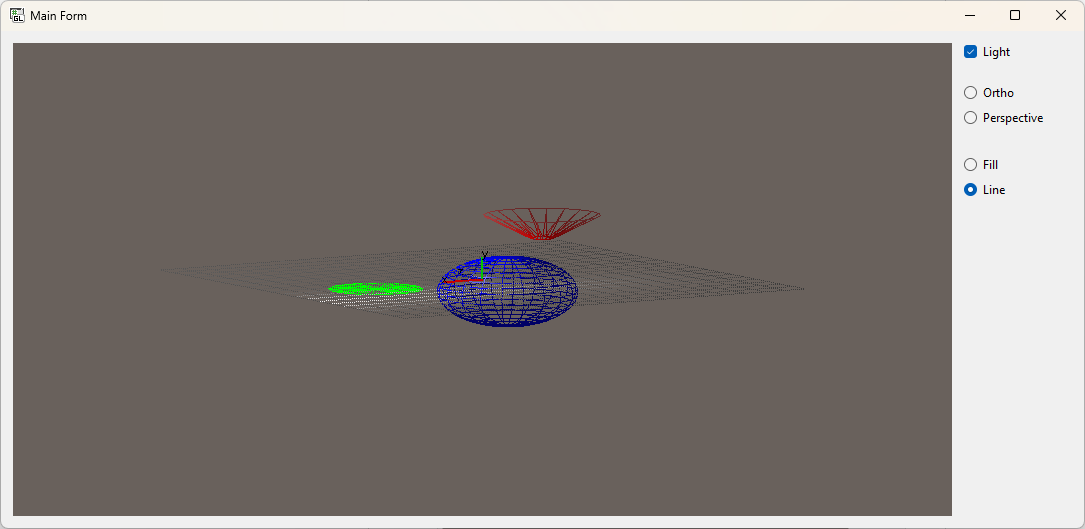


Рисунок 5.2 – Тестування роботи програми в ортогональному відображенні без заповнення фігур

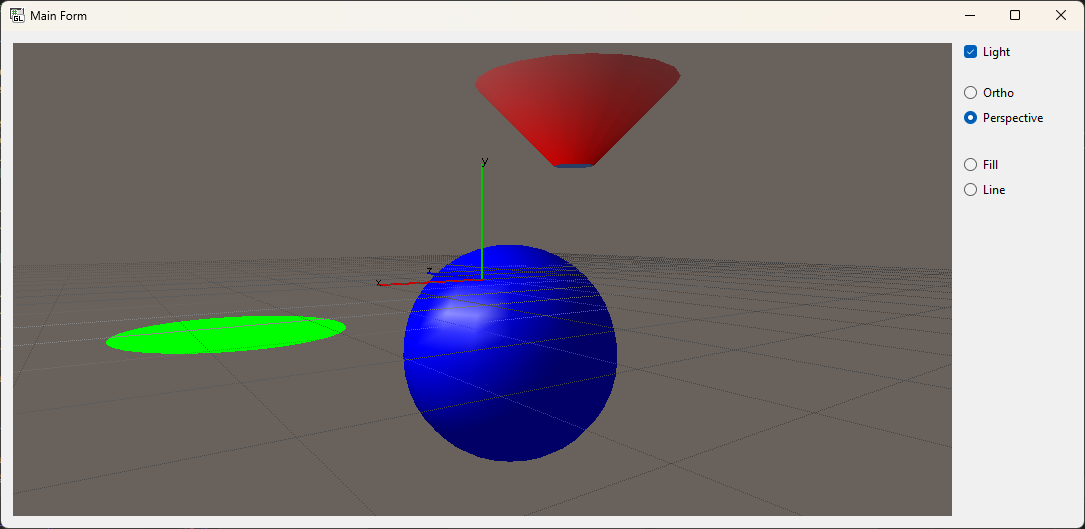


Рисунок 5.3 – Тестування роботи програми в перспективному відображенні з заповненням фігур

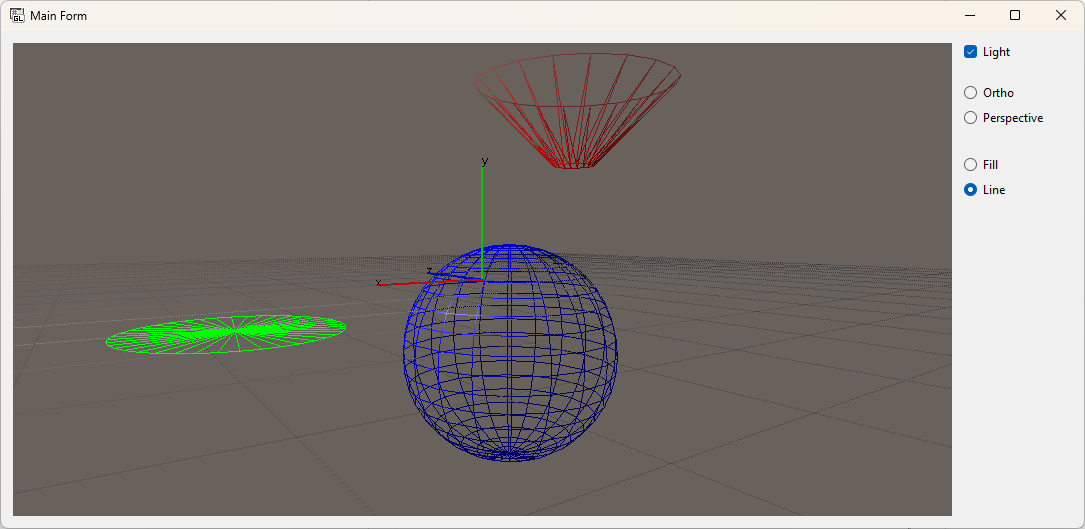


Рисунок 5.4 – Тестування роботи програми в перспективному відображенні без заповнення фігур

### Контроль виконання вимог та елементів завдання

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

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

# Практична робота 6. Візуалізація прямої задачі кінематики

## Завдання, варіант № 2

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

|  |  |  |
| --- | --- | --- |
| 2 | a = 0.4  b = 0.7  c = 0.24 |  |

## Теоретичні відомості

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

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

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

## Математична модель кінематичної схеми

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

## Афінні перетворення

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



та перенесення вздовж осі Y:



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



обертання навколо осі X:



та масштабування по всіх трьох осях:



## Декомпозиція на сегменти

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



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

## 2.2.1 Математична модель сегмента «a+b»

Розташування сегмента a+b залежить від кута: phi. При цьому кут phi, у свою чергу, залежить від величини S. Тому, для обчислення кута розглянемо трикутник, який утворено сторонами 2a, a+S і c і містить кут psi. Використовуючи теорему косинусів, наступним виразом можна зв'язати кут psi зі сторонами 2a, a+S і c:

|  |  |
| --- | --- |
| c2 = (a+S)2 + (2a)2 - 2 \* (a+S) \* (2a) \* cos(phi) | (2.7) |

з якого можна вивести залежність кута ag від значення S:

|  |  |
| --- | --- |
|  | (2.8) |

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

|  |  |
| --- | --- |
| Sab(phi, S)= Rz(phi(S))×Segment(a+b) | (2.9) |

## 2.2.1 Математична модель сегмента «b»

Розташування сегмента b залежить від двох кутів: phi та psi. Для відображення у складі кінематичної схеми позначимо його як Sb:

|  |  |
| --- | --- |
| Sb(psi, S)= Rz(phi(S))×Ty(a)×Rz(psi)×Segment(b) | (2.10) |

## 2.2.2 Математична модель сегмента «c»

Положення сегмента c залежить від кута theta. При цьому кут theta, у свою чергу, залежить від величини S. Тому, для обчислення кута розглянемо трикутник, який утворено сторонами 2a, a+S і c і містить кут theta. Використовуючи теорему косинусів, наступним виразом можна зв'язати кут theta зі сторонами 2a, a+S і c:

|  |  |
| --- | --- |
| (a+S)2 = c2 + (2a)2 - 2 \* c \* 2a \* cos(theta) | (2.11) |

з якого можна вивести залежність кута ag від значення S:

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

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

|  |  |
| --- | --- |
| Sc(theta, S)= Ty(2a)×Rz(psi(S))×Segment(c) | (2.13) |

## Фізичні обмеження моделі

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

|  |  |
| --- | --- |
| CheckMax(S) = S ≤ a + c  CheckMin(S) = S ≥ a - c | (2.14) |

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

## Загальна тривимірна математична модель

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



тоді повна тривимірна модель набуде наступного вигляду:

|  |  |
| --- | --- |
| Sab(phi, S)= Rz(phi(S))×Segment(a+b)  Sb(psi, S)= Rz(phi(S))×Ty(a)×Rz(psi)×Segment(b)  Sc(theta, S)= Ty(2a)×Rz(psi(S))×Segment(c) | (2.16) |

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

## Програмна реалізація кінематичної схеми

## Початкові дані

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

public partial class RenderControl : OpenGL

{

    private float a = 0.4f;

    private float b = 0.7f;

    private float c = 0.24f;

    ...

}

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

private float s = 0.5f;

private float S

{

get { return s; }

set { if((value >= a - c) && (value <= c + a)) s = value; }

}

крім цього, кути phi і thetha у свою чергу теж є функціональними залежностями від значення S, що відображено у (2.8) и (2.12) і, відповідно, повинні мати заборону на операцію запису (і водночас переводимо радіани в градуси):

private float theta

{

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

}

private float phi

{

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

}

## Прив'язка системи координат до видимої області вікна

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

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

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

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

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

private float Zmin { get => -size; }

private float Zmax { get => +size; }

. . .

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

## Керування об'єктом моделювання та організація інтерфейсу

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

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

{

mouseFlag = e.Button == MouseButtons.Left;

mouseStart = e.Location;

}

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

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

{

if (mouseFlag)

{

Point current = e.Location;

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

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

mouseStart = current;

Invalidate();

}

}

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

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

{

if (mouseFlag)

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

}

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

private void OnMouseWheel(object sender, MouseEventArgs e)

{

m += e.Delta / 2000.0f;

Invalidate();

}

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

private void OnKeyDown(object sender, KeyEventArgs e)

{

switch (e.KeyCode)

{

case Keys.W:

psi--;

break;

case Keys.S:

psi++;

break;

case Keys.E:

S += 0.01f;

break;

case Keys.Q:

S -= 0.01f;

break;

case Keys.A:

aw++;

break;

case Keys.D:

aw--;

break;

}

Invalidate();

}

## Осі координат

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

private void DrawAxis()

{

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

glLineWidth(1);

glBegin(GL\_LINES);

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

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

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

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

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

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

glEnd();

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

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

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

}

## Виведення сегмента

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

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

{

glColor3d(r, g, b);

glLineWidth(5);

glBegin(GL\_LINES);

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

glVertex3d(0f, height, -width);

glVertex3d(0f, 0f, width);

glVertex3d(0f, height, width);

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

glVertex3d(0f, 0f, width);

glVertex3d(0f, height, -width);

glVertex3d(0f, height, width);

glEnd();

glLineWidth(1);

}

## Порядок виконання афінних перетворень

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

private void OnRender(object sender, EventArgs e)

{

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

glLoadIdentity();

glViewport(0, 0, Width, Height);

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

glRotated(angleX, 1, 0, 0);

glRotated(angleY, 0, 1, 0);

glScaled(m, m, m);

// Enable depth

glEnable(GL\_DEPTH\_TEST);

DrawAxis();

DrawGridXY();

DrawGridYZ();

DrawGridZX();

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

glPushMatrix();

// first line

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

Segment(a + b, 0.05f, 1f, 0f, 0f);

// second line

glTranslated(0f, a, 0f);

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

Segment(b, 0.04f, 0f, 1f, 0f);

glPopMatrix();

// third line

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

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

Segment(c, 0.04f, 0f, 0f, 1f);

}

## Результати виконання практичної роботи

### Розв'язання завдання

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

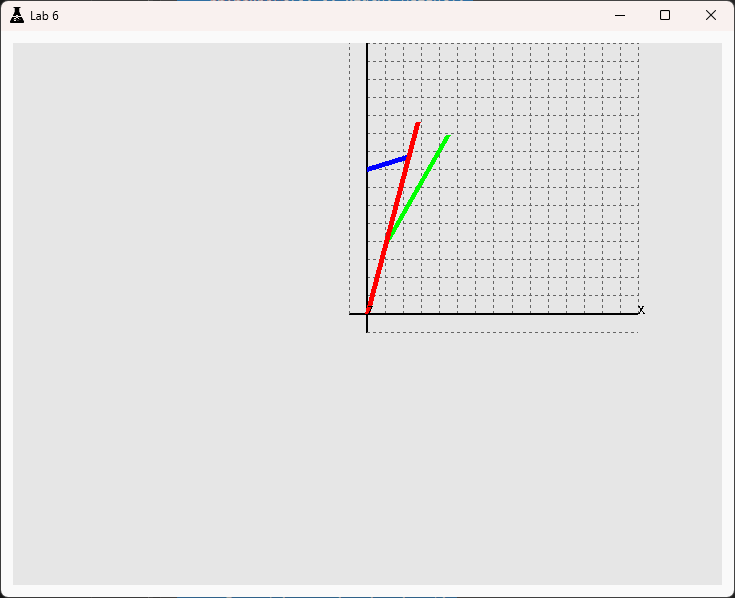


Рисунок 6.1 – Початковий стан кінематичної схеми після запуску програми

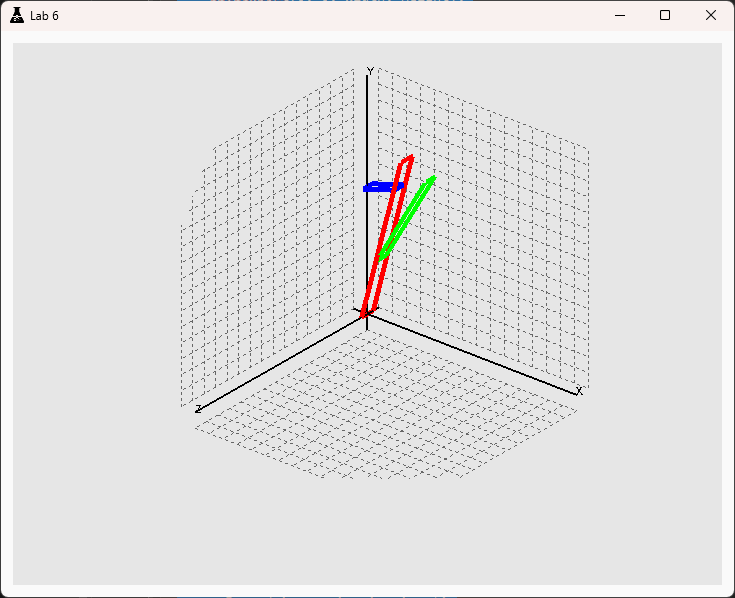


Рисунок 6.2 – Доопрацювання програми до тривимірної моделі

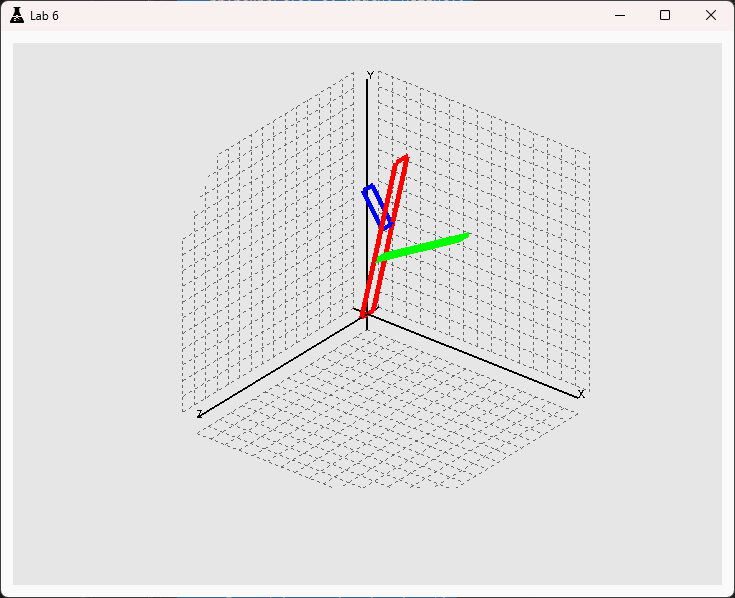


Рисунок 6.3 – Відображення координатної сітки X0Z та каркасу квадратичних об’єктів

### Контроль виконання вимог та елементів завдання

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

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

# Практична робота 7. Екранна заставка з анімацією

## Завдання

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

## Теоретичні відомості

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

ScreenSaver.scr – показати вікно налаштувань;

ScreenSaver.scr /c – показати вікно налаштувань модально;

ScreenSaver.scr /s – основний повноекранний режим роботи;

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

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

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

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

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

## Результати виконання практичної роботи

### Розв'язання завдання

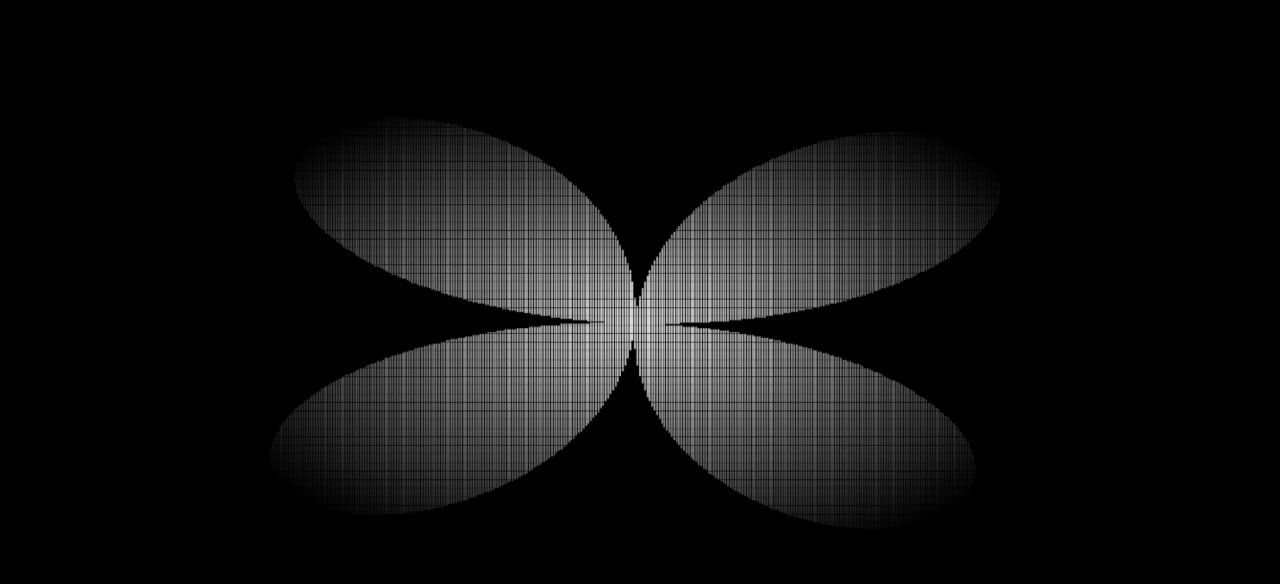


Рисунок 7.1 – Працююча анімація

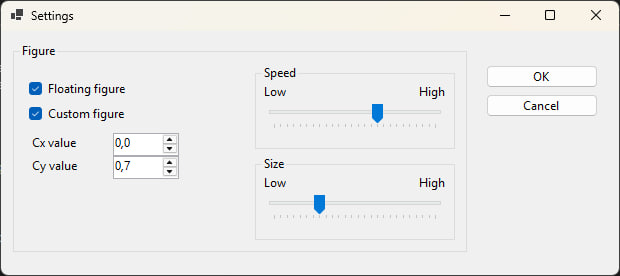


Рисунок 7.2 – Вікно з налаштуваннями

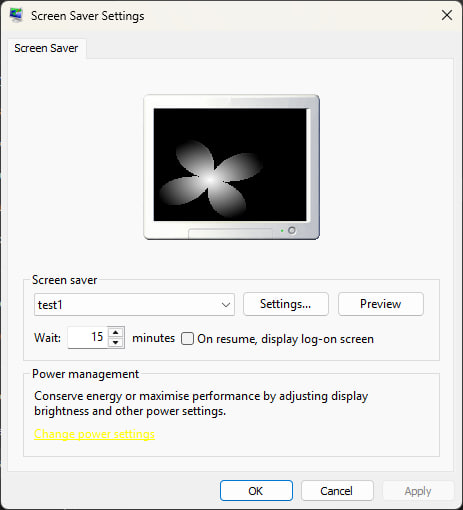


Рисунок 7.3 – Встановлена екранна заставка

### Контроль виконання вимог та елементів завдання

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

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

# Загальний перелік посилань

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

# Додаток А. Лістинг програми до практичної роботи №1

### Код файлу (MainForm.cs)

1. using System;
2. using System.Collections.Generic;
3. using System.ComponentModel;
4. using System.Diagnostics;
5. using System.Drawing;
6. using System.Drawing.Design;
7. using System.Linq;
8. namespace Lab1
9. {
10. public partial class RenderControl : OpenGL
11. {
12. public RenderControl()
13. {
14. InitializeComponent();
15. }
16. private void RenderControl\_Render(object sender, EventArgs e)

19. {

1. Figures figures = new Figures();

21. glClear(GL\_COLOR\_BUFFER\_BIT);

1. glLoadIdentity();
2. glViewport(0, 0, Width, Height);
3. gluOrtho2D(-10, +3, -5, +3);

25.

26. glLineWidth(1);

27. glEnable(GL\_LINE\_STIPPLE);

28. glLineStipple(6, 0xAAAA);

29. glColor3ub(200, 200, 200);

1. glBegin(GL\_LINES);
2. figures.Grid();
3. figures.DrawFigure();
4. figures.DrawPoints();
5. }
6. }
7. }

### Код файлу (Figures.cs)

1 using System;

2 using System.Collections.Generic;

3 using System.Linq;

4 using System.Text;

5 using System.Threading.Tasks;

6

7 namespace e Lab1

8 {

9 public partial class RenderControl

10 {

11 public class Figures

12 {

13 public void Grid()

14 {

15 for (int i = -8; i <= 1; i++)

16 {

17 glVertex2d(i, -4);

18 glVertex2d(i, 2);

19 }

20 for (int j = -3; j <= 1; j++)

21 {

22 glVertex2d(-9, j);

23 glVertex2d(2, j);

24 }

25 glEnd();

26 glDisable(GL\_LINE\_STIPPLE);

27 }

28 public void Form(int OffsetX = 0)

29 {

30 glBegin(GL\_LINE\_LOOP);

31 glVertex2d(-8 + OffsetX, 1);

32 glVertex2d(-6 + OffsetX, 1);

33 glVertex2d(-5 + OffsetX, -1);

34 glVertex2d(-5 + OffsetX, -3);

35 glVertex2d(-7 + OffsetX, -3);

36 glVertex2d(-8 + OffsetX, -1);

37 glEnd();

38 }

39 public void DrawFigure()

40 {

41 glLineWidth(3);

42 glColor3ub(0, 0, 0);

43 glBegin(GL\_LINE\_LOOP);

44 Form(0);

45 glEnd();

46 }

47 public void DrawPoints()

48 {

49 glEnable(GL\_POINT);

50 glPointSize(5);

51 glBegin(GL\_POINTS);

52 Form(5);

53 glEnd();

54 glDisable(GL\_POINT);

55 }

56 }

57 }

58 }

# Додаток Б. Лістинг програми до практичної роботи №2

### Код файлу (MainForm.cs)

using System;

using System.Drawing.Drawing2D;

using System.Windows.Forms;

using OpenGL\_Lab\_2.Enum;

using static OpenGL\_Lab\_2.OpenGL;

namespace OpenGL\_Lab\_2

{

public partial class MainForm : Form

{

public MainForm()

{

InitializeComponent();

}

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

{

renderControl1.Horizontales = (int)horizontalNumericUpDown.Value;

renderControl1.Verticales = (int)verticalNumericUpDown.Value;

renderControl1.Invalidate();

}

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

{

if (fillViewRadioButton.Checked)

{

renderControl1.figureView = FigureMode.Fill;

}

else if (lineViewRadioButton.Checked)

{

renderControl1.figureView = FigureMode.Line;

}

else if (pointViewRadioButton.Checked)

{

renderControl1.figureView = FigureMode.Point;

}

renderControl1.Invalidate();

}

private void OnClick(object sender, MouseEventArgs e)

{

double sizeX = renderControl1.ClientRectangle.Width;

double sizeY = renderControl1.ClientRectangle.Height;

double adjustedY = sizeY - e.Y;

double buttonWidth = 40;

double buttonHeight = 30;

double buttonSpace = 10;

double xStart = sizeX - (buttonWidth);

double yStart = buttonSpace;

for (int i = 0; i < 3; i++)

{

int count = 2;

double x = xStart - i \* (buttonWidth + buttonSpace);

double y = yStart;

if (e.X >= x && e.X <= x + buttonWidth && adjustedY >= y && adjustedY <= y + buttonHeight)

{

renderControl1.figureView = (FigureMode)count - i;

switch (count - i)

{

case 0:

fillViewRadioButton.Checked = true; break;

case 1:

lineViewRadioButton.Checked = true; break;

case 2:

pointViewRadioButton.Checked = true; break;

}

renderControl1.Invalidate();

break;

}

}

}

}

}

### 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 OpenGL\_Lab\_2.Enum;

using static OpenGL\_Lab\_2.RenderControl;

namespace OpenGL\_Lab\_2

{

public partial class RenderControl : OpenGL

{

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

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

public FigureMode figureView { get; set; } = 0;

Draw draw;

double size = 15.0;

double sizeSide = 4.25;

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);

}

double maxSize = Math.Max(Horizontales, Verticales);

gluOrtho2D(-size \* maxSize, size \* maxSize, -size \* maxSize, size \* maxSize);

draw.DrawFigure(Verticales, Horizontales, sizeSide, figureView);

glLoadIdentity();

glViewport(0, 0, Width, Height);

gluOrtho2D(-Width / 2, Width / 2, -Height / 2, Height / 2);

draw.DrawButtons(figureView, -Width / 2, Width / 2, -Height / 2, Height / 2);

}

private void RenderControl\_ContextCreated(object sender, EventArgs e)

{

draw = new Draw();

draw.Print = DrawText;

}

}

}

### Код файлу (Draw.cs)

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using OpenGL\_Lab\_2.Enum;

namespace OpenGL\_Lab\_2

{

public partial class RenderControl

{

public class Draw

{

public delegate void outText(string s, double x, double y, double z = 0);

public outText Print;

public void DrawFigure(int verticales, int horizontales, double size, FigureMode figureView)

{

double displacementX, displacementY;

if (verticales != 0 || horizontales != 0)

{

for (int i = 0; i < horizontales; i += 1)

{

for (int j = 0; j < verticales; j += 1)

{

displacementX = i \* size \* 2;

displacementY = j \* size \* 2;

if (j > 0)

{

displacementX += (size \* j) / 2;

}

if (i > 0)

{

displacementY -= (size / 2) \* i;

}

switch (figureView)

{

case FigureMode.Fill:

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

break;

case FigureMode.Line:

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

break;

case FigureMode.Point:

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

break;

}

glBegin(GL\_QUADS);

glColor3f(0, 0, 1);

QuadsCoordinates(displacementX, displacementY, size);

glBegin(GL\_QUADS);

glColor3f(0, 0, 1);

QuadsTurnedCoordinates(displacementX + size, displacementY + size, size);

glEnd();

glBegin(GL\_TRIANGLES);

glColor3f(1, 1, 0);

TriangleBottomCoordinates(displacementX, displacementY + size, size);

glColor3f(1, 1, 0);

TriangleRightCoordinates(displacementX + size, displacementY, size);

glColor3f(1, 0, 0);

TriangleTopCoordinates(displacementX + size \* 2, displacementY + size / 2, size);

glColor3f(0.6f, 0.6f, 0.6f);

TriangleLeftCoordinates(displacementX + size, displacementY - size / 2, size);

glEnd();

}

}

}

}

public void QuadsCoordinates(double start\_x, double start\_y, double size)

{

glVertex2d(start\_x, start\_y);

glVertex2d(start\_x + size, start\_y);

glVertex2d(start\_x + size, start\_y + size);

glVertex2d(start\_x, start\_y + size);

}

public void QuadsTurnedCoordinates(double start\_x, double start\_y, double size)

{

glVertex2d(start\_x, start\_y);

glVertex2d(start\_x + size / 2, start\_y + size);

glVertex2d(start\_x + size \* 1.5, start\_y + size / 2);

glVertex2d(start\_x + size, start\_y - size / 2);

}

public void TriangleTopCoordinates(double start\_x, double start\_y, double size)

{

glVertex2d(start\_x, start\_y);

glVertex2d(start\_x + size / 2, start\_y + size);

glVertex2d(start\_x + size, start\_y);

}

public void TriangleBottomCoordinates(double start\_x, double start\_y, double size)

{

glVertex2d(start\_x + size / 2, start\_y + size);

glVertex2d(start\_x + size, start\_y);

glVertex2d(start\_x + size \* 1.5, start\_y + size);

}

public void TriangleRightCoordinates(double start\_x, double start\_y, double size)

{

glVertex2d(start\_x, start\_y);

glVertex2d(start\_x, start\_y + size);

glVertex2d(start\_x + size, start\_y + size / 2);

}

public void TriangleLeftCoordinates(double start\_x, double start\_y, double size)

{

glVertex2d(start\_x, start\_y + size / 2);

glVertex2d(start\_x + size, start\_y + size);

glVertex2d(start\_x + size, start\_y);

}

public void DrawButtons(FigureMode figureMode, double start\_x, double end\_x, double start\_y, double end\_y)

{

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

string[] buttonLabels = { "Fill", "Lines", "Point" };

double buttonWidth = 40;

double buttonHeight = 30;

double space = 10;

double xStart = end\_x - buttonWidth;

double yStart = start\_y;

for (int i = 0; i < 3; i++)

{

double x = xStart - i \* (buttonWidth + space);

double y = yStart;

if ((int)figureMode == 2 - i)

{

glColor3f(1, 0.2f, 0);

}

else

{

glColor3f(0.8f, 0.0f, 0.0f);

}

glBegin(GL\_QUADS);

glVertex2d(x, y);

glVertex2d(x, y + buttonHeight);

glVertex2d(x + buttonWidth, y + buttonHeight);

glVertex2d(x + buttonWidth, y);

glEnd();

glColor3f(0, 0, 0);

Print(buttonLabels[2 - i], x + 3, y + buttonHeight / 3);

}

}

}

}

}

### Код файлу (FigureMode.cs)

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace OpenGL\_Lab\_2.Enum

{

public enum FigureMode

{

Fill,

Line,

Point

}

}

# Додаток С. Лістинг програми до практичної роботи №3

### Код файлу (MainForm.cs)

using System.Diagnostics.Eventing.Reader;

using System.Drawing;

using System.Windows.Forms;

using static OpenGL\_Lab\_3.OpenGL;

namespace OpenGL\_Lab\_3

{

public partial class MainForm : Form

{

public MainForm()

{

InitializeComponent();

}

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

{

renderControl1.XMin = (double)numericUpDownXmin.Value;

//numericUpDownXmin.Maximum = (decimal)((double)numericUpDownXmax.Value - 1);

renderControl1.XMax = (double)numericUpDownXmax.Value;

//numericUpDownXmax.Minimum = (decimal)((double)numericUpDownXmin.Value + 1);

renderControl1.YMin = (double)numericUpDownYmin.Value;

//numericUpDownYmin.Maximum = (decimal)((double)numericUpDownYmax.Value - 1);

renderControl1.YMax = (double)numericUpDownYmax.Value;

//numericUpDownYmax.Minimum = (decimal)((double)numericUpDownYmin.Value + 1);

renderControl1.Invalidate();

}

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

{

renderControl1.pointNum = (int)numericUpDownPoints.Value;

renderControl1.Invalidate();

}

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

{

if (radioButtonFunc1.Checked)

{

renderControl1.Func = 0;

}

if (radioButtonFunc2.Checked)

{

renderControl1.Func = 1;

}

renderControl1.Invalidate();

}

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

{

if (!renderControl1.AutoCalcY)

{

buttonAutoCalcY.Text = "Back";

label5.Location = new Point(label5.Location.X, label5.Location.Y + 60);

numericUpDownPoints.Location = new Point(numericUpDownPoints.Location.X, numericUpDownPoints.Location.Y + 60);

buttonAutoCalcY.Location = new Point(buttonAutoCalcY.Location.X, buttonAutoCalcY.Location.Y + 60);

numericUpDownYmin.Visible = true;

numericUpDownYmax.Visible = true;

label3.Visible = true;

label4.Visible = true;

}

else

{

buttonAutoCalcY.Text = "Auto Calc (Y)";

label5.Location = new Point(label5.Location.X, label5.Location.Y - 60);

numericUpDownPoints.Location = new Point(numericUpDownPoints.Location.X, numericUpDownPoints.Location.Y - 60);

buttonAutoCalcY.Location = new Point(buttonAutoCalcY.Location.X, buttonAutoCalcY.Location.Y - 60);

numericUpDownYmin.Visible = false;

numericUpDownYmax.Visible = false;

label3.Visible = false;

label4.Visible = false;

}

renderControl1.AutoCalcY = !renderControl1.AutoCalcY;

renderControl1.Invalidate();

}

}

}

### Код файлу (RenderControl.cs)

using OpenGL\_Lab\_3;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Diagnostics;

using System.Drawing;

using System.Drawing.Design;

using System.Linq;

using static OpenGL\_Lab\_3.RenderControl;

namespace OpenGL\_Lab\_3

{

public partial class RenderControl : OpenGL

{

Draw draw;

public double pointNum = 500;

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

public double XMin { get; set; } = -1;

public double XMax { get; set; } = +1;

public double YMin { get; set; } = -1;

public double YMax { get; set; } = +1;

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

Func<double, double>[] functionArray;

public RenderControl()

{

InitializeComponent();

}

private void RenderControl\_Render(object sender, EventArgs e)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glLoadIdentity();

glViewport(0, 0, Width, Height);

if (!AutoCalcY)

{

(YMin, YMax) = draw.FindMinAndMax(XMin, XMax, pointNum, functionArray[Func]);

}

YMax = Math.Min(YMax, 100);

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

gluOrtho2D(XMin, XMax, YMin, YMax);

draw.DrawGrid(XMin, XMax, YMin, YMax);

draw.CoordinateGrid(XMin, XMax, YMin, YMax);

draw.FunctionLinesAndPoints(XMin, XMax, YMin, YMax, pointNum, functionArray[Func]);

}

private void RenderControl\_ContextCreated(object sender, EventArgs e)

{

draw = new Draw();

functionArray = new Func<double, double>[]

{

// f1(x) = cos(cos(2x + 0.1))

x => {

double innerCos = Math.Cos(2 \* x + 0.1);

return Math.Cos(innerCos);

},

// f2(x) = (cos(cos(pi \* x)) \* e^(cos(5x))) / |cos(cos(pi \* x))| - 0.5

x => {

double innerCosPiX = Math.Cos(Math.PI \* x);

double expValue = Math.Exp(Math.Cos(5 \* x));

double numerator = Math.Cos(innerCosPiX) \* expValue;

double denominator = Math.Abs(innerCosPiX);

return numerator / denominator - 0.5;

},

};

}

}

}

### Код файлу (Draw.cs)

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace OpenGL\_Lab\_3

{

public partial class RenderControl

{

public class Draw

{

public void DrawGrid(double XMin, double XMax, double YMin, double YMax)

{

glLineWidth(1.0f);

glColor3ub(200, 200, 200);

glBegin(GL\_LINES);

for (double i = XMin; i <= XMax; i += 0.2)

{

glVertex2d(i, YMin);

glVertex2d(i, YMax);

}

for (double j = YMin; j <= YMax; j += 0.2)

{

glVertex2d(XMin, j);

glVertex2d(XMax, j);

}

glEnd();

}

public void CoordinateGrid(double XMin, double XMax, double YMin, double YMax)

{

glLineWidth(3.0f);

glBegin(GL\_LINES);

glColor3ub(0, 0, 0);

glVertex2d(XMin, 0);

glVertex2d(XMax, 0);

glVertex2d(0, YMin);

glVertex2d(0, YMax);

glEnd();

}

public void FunctionLinesAndPoints(double XMin, double XMax, double YMin, double YMax, double pointNum, Func<double, double> func)

{

glLineWidth(2.0f);

glColor3ub(0, 0, 0);

glBegin(GL\_LINE\_STRIP);

double previousY = double.NaN;

double stepSize = (XMax - XMin) / (pointNum - 1);

double x = XMin;

double y = func(x);

glVertex2d(x, y);

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

{

previousY = y;

x = XMin + i \* stepSize;

y = func(x);

if (!double.IsNaN(previousY) && Math.Abs(y - previousY) > 5.0)

{

glEnd();

glBegin(GL\_LINE\_STRIP);

continue;

}

glVertex2d(x, y);

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

{

glEnd();

Point(previousY, x, stepSize, y);

glBegin(GL\_LINE\_STRIP);

glColor3ub(0, 0, 0);

glVertex2d(x, y);

}

}

glEnd();

}

public (double, double) FindMinAndMax(double XMin, double XMax, double pointNum, Func<double, double> func)

{

double stepSize = (XMax - XMin) / (pointNum - 1);

double x = XMin;

double y = func(x);

double minY = y, maxY = y;

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

{

x = XMin + i \* stepSize;

y = func(x);

if (y > maxY)

{

maxY = y;

}

if (y < minY)

{

minY = y;

}

}

return (minY, maxY);

}

private void Point(double previousY, double x, double h, double y)

{

glPointSize(5.0f);

glColor3ub(250, 0, 0);

glBegin(GL\_POINTS);

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

glEnd();

}

}

}

}

# Додаток Г. Лістинг програми до практичної роботи №4

### Код файлу (MainForm.cs)

using System;

using System.Drawing;

using System.Reflection.Emit;

using System.Windows.Forms;

using static OpenGL\_Lab\_4.OpenGL;

namespace OpenGL\_Lab\_4

{

public partial class MainForm : Form

{

public MainForm()

{

InitializeComponent();

}

private void OnCheckedChangedRb1(object sender, System.EventArgs e)

{

renderControl1.ChosedFigure = false;

numericUpDownA.Visible = true;

label1.Visible = true;

renderControl1.Invalidate();

}

private void OnCheckedChangedRb2(object sender, System.EventArgs e)

{

renderControl1.ChosedFigure = true;

numericUpDownA.Visible = false;

label1.Visible = false;

renderControl1.CoefficientA = 1.0;

renderControl1.Invalidate();

}

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

{

if (numericUpDownA.Value == numericUpDownB.Value)

{

numericUpDownB.Value = (double)numericUpDownA.Value > renderControl1.CoefficientA ?

(decimal)((double)numericUpDownB.Value - 0.1) :

(decimal)((double)numericUpDownB.Value + 0.1);

}

renderControl1.CoefficientA = (double)numericUpDownA.Value;

renderControl1.Invalidate();

}

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

{

if (numericUpDownA.Value == numericUpDownB.Value)

{

numericUpDownA.Value = (double)numericUpDownB.Value > renderControl1.CoefficientB ?

(decimal)((double)numericUpDownA.Value - 0.1) :

(decimal)((double)numericUpDownA.Value + 0.1);

}

renderControl1.CoefficientB = (double)numericUpDownB.Value;

renderControl1.Invalidate();

}

private void OnMouseDown(object sender, MouseEventArgs e)

{

if (e.Button == MouseButtons.Left)

{

double width = renderControl1.ClientRectangle.Width;

double height = renderControl1.ClientRectangle.Height;

double minSide = Math.Min(width, height);

double maxSide = Math.Max(width, height);

double difference = maxSide - minSide;

bool isWidthGreater = width > height;

double xLength = isWidthGreater ? width / height : 1;

double yLength = !isWidthGreater ? height / width : 1;

bool isInBounds = difference / 2 < (isWidthGreater ? e.X : height - e.Y) &&

(isWidthGreater ? e.X : height - e.Y) < minSide + difference / 2;

renderControl1.LineIsExist = (difference != 0) ? isInBounds : true;

if (renderControl1.LineIsExist)

{

renderControl1.MouseIsUp = false;

renderControl1.StartLineX = renderControl1.FinishLineX = (renderControl1.Length \* xLength) \* ((e.X - width / 2.0) / (width / 2.0));

renderControl1.StartLineY = renderControl1.FinishLineY = (renderControl1.Length \* yLength) \* ((height / 2.0 - e.Y) / (height / 2.0));

}

}

renderControl1.Invalidate();

}

private void OnMouseUp(object sender, MouseEventArgs e)

{

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

{

double width = renderControl1.ClientRectangle.Width;

double height = renderControl1.ClientRectangle.Height;

bool isWidthGreater = width > height;

double xLength = isWidthGreater ? width / height : 1;

double yLength = !isWidthGreater ? height / width : 1;

renderControl1.FinishLineX = (renderControl1.Length \* xLength) \* ((e.X - width / 2.0) / (width / 2.0));

renderControl1.FinishLineY = (renderControl1.Length \* yLength) \* ((height / 2.0 - e.Y) / (height / 2.0));

renderControl1.MouseIsUp = true;

bool isLineSame = renderControl1.FinishLineX == renderControl1.StartLineX && renderControl1.FinishLineY == renderControl1.StartLineY;

renderControl1.LineIsExist = !isLineSame;

}

renderControl1.Invalidate();

}

}

}

### 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 OpenGL\_Lab\_4

{

public partial class RenderControl : OpenGL

{

Draw draw;

public double StartLineX { get; set; }

public double StartLineY { get; set; }

public double FinishLineX { get; set; }

public double FinishLineY { get; set; }

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

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

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

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

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

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

public RenderControl()

{

InitializeComponent();

}

private void OnRender(object sender, EventArgs e)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glLoadIdentity();

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

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

Length = draw.FindMax(CoefficientA, CoefficientB, ChosedFigure);

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

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

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

if (!ChosedFigure)

draw.DrawParametricEllipse(CoefficientA, CoefficientB);

else

draw.DrawHyperbola(CoefficientA, CoefficientB);

if (LineIsExist)

{

CreateLine();

draw.SearchPoint(CoefficientA, CoefficientB, StartLineX, StartLineY, FinishLineX, FinishLineY, ChosedFigure);

}

}

private void CreateLine()

{

if (!MouseIsUp)

{

draw.DrawLineWait(StartLineX, FinishLineX, StartLineY, FinishLineY);

}

else

{

draw.DrawLine(StartLineX, FinishLineX, StartLineY, FinishLineY);

}

}

private void OnCreated(object sender, EventArgs e)

{

draw = new Draw();

}

}

}

### Код файлу (Draw.cs)

using System;

using System.Collections.Generic;

using System.Drawing.Printing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace OpenGL\_Lab\_4

{

public partial class RenderControl

{

public class Draw

{

public void DrawParametricEllipse(double a, double b)

{

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

glBegin(GL\_LINE\_LOOP);

int segments = 100;

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

{

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

double x = 0.0 + a \* Math.Cos(t);

double y = 0.0 + b \* Math.Sin(t);

glVertex2d(x, y);

}

glEnd();

}

public void DrawHyperbola(double a, double b)

{

glColor3d(0, 0, 1);

glBegin(GL\_LINE\_STRIP);

double tMax = 3.0;

for (double t = -tMax; t <= tMax; t += 0.05)

{

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

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

glVertex2d(x, y);

}

glEnd();

glBegin(GL\_LINE\_STRIP);

for (double t = -tMax; t <= tMax; t += 0.05)

{

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

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

glVertex2d(x, y);

}

glEnd();

}

public void SearchPoint(double a, double b, double Xls, double Yls, double Xle, double Yle, bool isChoosed)

{

glPointSize(5.0f);

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

glBegin(GL\_POINTS);

double dx = Xle - Xls;

double dy = Yle - Yls;

double A, B, C;

if (isChoosed)

{

A = (dx \* dx) / (a \* a) - (dy \* dy) / (b \* b);

B = 2 \* (Xls \* dx / (a \* a) - Yls \* dy / (b \* b));

C = (Xls \* Xls) / (a \* a) - (Yls \* Yls) / (b \* b) - 1;

}

else

{

A = (dx \* dx) / (a \* a) + (dy \* dy) / (b \* b);

B = 2 \* (Xls \* dx / (a \* a) + Yls \* dy / (b \* b));

C = (Xls \* Xls) / (a \* a) + (Yls \* Yls) / (b \* b) - 1;

}

HandleDiscriminant(A, B, C, dx, dy, Xls, Yls);

glEnd();

}

private void HandleDiscriminant(double A, double B, double C, double dx, double dy, double Xls, double Yls)

{

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

if (discriminant >= 0)

{

double sqrtD = Math.Sqrt(discriminant);

if (A != 0)

{

AddPoint((-B + sqrtD) / (2 \* A), dx, dy, Xls, Yls);

AddPoint((-B - sqrtD) / (2 \* A), dx, dy, Xls, Yls);

}

else if (B != 0)

{

AddPoint(-C / B, dx, dy, Xls, Yls);

}

}

}

private void AddPoint(double t, double dx, double dy, double Xls, double Yls)

{

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

{

double x = Xls + t \* dx;

double y = Yls + t \* dy;

glVertex2d(x, y);

}

}

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;

}

return Math.Max(maxX, maxY);

}

public void DrawGrid(double XMin, double XMax, double YMin, double YMax)

{

glLineWidth(1.0f);

glColor3ub(200, 200, 200);

glBegin(GL\_LINES);

for (double i = XMin; i <= XMax; i += XMax / 10)

{

glVertex2d(i, YMin);

glVertex2d(i, YMax);

}

for (double j = YMin; j <= YMax; j += YMax / 10)

{

glVertex2d(XMin, j);

glVertex2d(XMax, j);

}

glEnd();

}

public void DrawCoordinateGrid(double XMin, double XMax, double YMin, double YMax)

{

glLineWidth(2.0f);

glBegin(GL\_LINES);

glColor3ub(0, 0, 0);

glVertex2d(XMin, 0);

glVertex2d(XMax, 0);

glVertex2d(0, YMin);

glVertex2d(0, YMax);

glEnd();

glLineWidth(1.0f);

}

public void DrawLineWait(double XMin, double XMax, double YMin, double YMax)

{

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

glLineWidth(1.7f);

glBegin(GL\_LINES);

glVertex2d(XMin, YMin);

glVertex2d(XMax, YMax);

glEnd();

}

public void DrawLine(double XMin, double XMax, double YMin, double YMax)

{

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

glLineWidth(1.7f);

glBegin(GL\_LINES);

glVertex2d(XMin, YMin);

glVertex2d(XMax, YMax);

glEnd();

}

}

}

}

# Додаток е. Лістинг програми до практичної роботи №5

### Код файлу (MainForm.cs)

using System;

using System.Windows.Forms;

using static Task05.OpenGL;

namespace Task05

{

public partial class MainForm : Form

{

public MainForm()

{

InitializeComponent();

}

private void OnWheel(object sender, MouseEventArgs e)

{

if (e.Delta < 0)

{

renderControl1.Multi += 0.1;

}

else

{

if (renderControl1.Multi > 0.2)

{

renderControl1.Multi -= 0.1;

}

}

renderControl1.Invalidate();

}

private void OnMouseDown(object sender, MouseEventArgs e)

{

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

{

renderControl1.NotRotating = false;

renderControl1.LastLeftMouseX = e.X;

renderControl1.LastLeftMouseY = e.Y;

}

else if (e.Button == MouseButtons.Right && renderControl1.NotPanning)

{

renderControl1.NotPanning = false;

renderControl1.LastRightMouseX = e.X;

renderControl1.LastRightMouseY = e.Y;

}

}

private void OnMouseMove(object sender, MouseEventArgs e)

{

if (renderControl1.Mode)

{

if (!renderControl1.NotRotating)

{

double deltaX = e.X - renderControl1.LastLeftMouseX;

double deltaY = e.Y - renderControl1.LastLeftMouseY;

renderControl1.CameraTheta += deltaX \* 0.01;

renderControl1.CameraPhi -= deltaY \* 0.01;

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

renderControl1.LastLeftMouseX = e.X;

renderControl1.LastLeftMouseY = e.Y;

renderControl1.Invalidate();

}

}

else

{

if (!renderControl1.NotRotating)

{

double deltaX = e.X - renderControl1.LastLeftMouseX;

double deltaY = e.Y - renderControl1.LastLeftMouseY;

renderControl1.AngleX += deltaX \* 0.2;

renderControl1.AngleY += deltaY \* 0.2;

renderControl1.LastLeftMouseX = e.X;

renderControl1.LastLeftMouseY = e.Y;

renderControl1.Invalidate();

}

}

if (!renderControl1.NotPanning)

{

double deltaX = e.X - renderControl1.LastRightMouseX;

double deltaY = e.Y - renderControl1.LastRightMouseY;

renderControl1.PanOffsetX += deltaX \* 0.01;

renderControl1.PanOffsetY -= deltaY \* 0.01;

renderControl1.LastRightMouseX = e.X;

renderControl1.LastRightMouseY = e.Y;

renderControl1.Invalidate();

}

}

private void OnMouseUp(object sender, MouseEventArgs e)

{

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

{

renderControl1.NotRotating = true;

}

if (!renderControl1.NotPanning && e.Button == MouseButtons.Right)

{

renderControl1.NotPanning = true;

}

}

private void OnChekedLight(object sender, EventArgs e)

{

renderControl1.LightOn = cbLight.Checked;

renderControl1.Invalidate();

}

private void OnCheckedOrtho(object sender, EventArgs e)

{

if (rdbtnOrtho.Checked)

{

renderControl1.Mode = false;

renderControl1.Invalidate();

}

}

private void OnCheckedPerspective(object sender, EventArgs e)

{

if (rdbtnPerspective.Checked)

{

renderControl1.Mode = true;

renderControl1.Invalidate();

}

}

private void OnCheckedFill(object sender, EventArgs e)

{

if (rdbtnFill.Checked)

{

renderControl1.Fill = true;

renderControl1.Invalidate();

}

}

private void OnCheckedLine(object sender, EventArgs e)

{

if (rdbtnLine.Checked)

{

renderControl1.Fill = false;

renderControl1.Invalidate();

}

}

}

}

### 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 Task05

{

public partial class RenderControl : OpenGL

{

Draw draw;

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

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

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

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

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

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

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

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

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

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

public double CameraRadius { get; set; } = 10.0;

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

public double CameraPhi { get; set; } = Math.PI / 2;

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

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

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

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

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

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

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

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

private uint coordinatesDisplayList;

private uint gridDisplayList;

private uint sphereDisplayList;

private uint coneDisplayList;

private uint discDisplayList;

public RenderControl()

{

InitializeComponent();

}

private void OnRender(object sender, EventArgs e)

{

glClearColor(0.41f, 0.38f, 0.36f, 1.0f);

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

glEnable(GL\_DEPTH\_TEST);

glViewport(0, 0, Width, Height);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

double size = 20;

double aspect = (double)Width / Height;

if (Mode)

{

gluPerspective(45.0, aspect, 0.1, 100.0);

}

else

{

glOrtho(-size \* Multi, size \* Multi, -size \* Multi, size \* Multi, -size \* 2.5 \* Multi, size \* 2.5 \* Multi);

}

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glTranslated(PanOffsetX, PanOffsetY, 0.0);

if (Mode)

{

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

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

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

gluLookAt(

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

CenterX, CenterY, CenterZ,

0.0, 1.0, 0.0

);

}

else

{

glRotatef((float)AngleY, 1.0f, 0.0f, 0.0f);

glRotatef((float)AngleX, 0.0f, 1.0f, 0.0f);

}

glCallList(coordinatesDisplayList);

if (LightOn)

SetupLighting();

else

{

glDisable(GL\_LIGHTING);

glDisable(GL\_LIGHT0);

glDisable(GL\_COLOR\_MATERIAL);

}

glCallList(gridDisplayList);

double scale = CameraRadius;

float lineWidth = Math.Clamp((float)(1.0 / scale), 0.1f, 5.0f);

glLineWidth(lineWidth);

draw.DrawSphere(-4.0, -2.5, 4.5, 3.0, Fill);

draw.DrawCone(-4.5, +2.5, +2.5, 2.5, 0.5, 2.0, Fill);

draw.DrawDisc(+3.5, -1.0, +3.0, 2.0, 0.0, Fill);

}

private void SetupLighting()

{

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

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

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

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

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

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

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

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

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

glEnable(GL\_COLOR\_MATERIAL);

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

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

float materialShininess = 50.0f;

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

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

}

private void InitializeDisplayLists()

{

coordinatesDisplayList = glGenLists(1);

glNewList(coordinatesDisplayList, GL\_COMPILE);

draw.DrawCoordinatesLines(CenterX, CenterY, CenterZ);

glEndList();

gridDisplayList = glGenLists(1);

glNewList(gridDisplayList, GL\_COMPILE);

draw.DrawGrid(10);

glEndList();

sphereDisplayList = glGenLists(1);

glNewList(sphereDisplayList, GL\_COMPILE);

draw.DrawSphere(-4.0, -2.5, 4.5, 3.0, Fill);

glEndList();

coneDisplayList = glGenLists(1);

glNewList(coneDisplayList, GL\_COMPILE);

draw.DrawCone(-4.5, +2.5, +2.5, 2.5, 0.5, 2.0, Fill);

glEndList();

discDisplayList = glGenLists(1);

glNewList(discDisplayList, GL\_COMPILE);

draw.DrawDisc(+3.5, -1.0, +3.0, 2.0, 0.0, Fill);

glEndList();

}

private void DeleteDisplayLists()

{

glDeleteLists(coordinatesDisplayList, 1);

glDeleteLists(gridDisplayList, 1);

glDeleteLists(sphereDisplayList, 1);

glDeleteLists(coneDisplayList, 1);

glDeleteLists(discDisplayList, 1);

}

private void OnCreate(object sender, EventArgs e)

{

draw = new Draw();

draw.Print = DrawText;

InitializeDisplayLists();

}

private void OnDelete(object sender, EventArgs e)

{

DeleteDisplayLists();

}

}

}

### Код файлу (Draw.cs)

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Task05

{

public partial class RenderControl

{

public class Draw

{

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

public outText Print;

public void DrawGrid(double size)

{

glEnable(GL\_LINE\_STIPPLE);

glLineStipple(1, 0xAAAA);

glLineWidth(0.5f);

glColor3ub(100, 100, 100);

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();

glDisable(GL\_LINE\_STIPPLE);

}

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

{

glDisable(GL\_LIGHTING);

glDisable(GL\_LIGHT0);

glDisable(GL\_COLOR\_MATERIAL);

glLineWidth(2.5f);

glBegin(GL\_LINES);

glColor3ub(200, 0, 0);

glVertex3d(x, y, z);

glVertex3d(x+2, y, z);

glColor3ub(0, 200, 0);

glVertex3d(x, y, z);

glVertex3d(x, y+2, z);

glColor3ub(0, 0, 200);

glVertex3d(x, y, z);

glVertex3d(x, y, z+2);

glEnd();

glColor3ub(0, 0, 0);

Print("x", x + 2, y, z);

Print("y", x, y + 2, z);

Print("z", x, y, z + 2);

}

public void DrawSphere(double x0, double y0, double z0, double radius, bool Fill, int slices = 20, int stacks = 20)

{

if (Fill)

{

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

}

else

{

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

}

glPushMatrix();

glTranslated(x0, y0, z0);

glClipPlane(GL\_CLIP\_PLANE0, new double[] { x0+1, 0, 0, 0.5 });

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

{

double phi1 = Math.PI \* i / stacks;

double phi2 = Math.PI \* (i + 1) / stacks;

glBegin(GL\_QUAD\_STRIP);

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

{

double theta = 2.0 \* Math.PI \* j / slices;

double x1 = radius \* Math.Sin(phi1) \* Math.Cos(theta);

double y1 = radius \* Math.Cos(phi1);

double z1 = radius \* Math.Sin(phi1) \* Math.Sin(theta);

double x2 = radius \* Math.Sin(phi2) \* Math.Cos(theta);

double y2 = radius \* Math.Cos(phi2);

double z2 = radius \* Math.Sin(phi2) \* Math.Sin(theta);

glColor3ub(0, 0, 255);

glNormal3d(x1 / radius, y1 / radius, z1 / radius);

glVertex3d(x1, y1, z1);

glNormal3d(x2 / radius, y2 / radius, z2 / radius);

glVertex3d(x2, y2, z2);

}

glEnd();

}

glPopMatrix();

}

public void DrawCone(double x0, double y0, double z0, double radius, double radius\_small, double height, bool Fill, int slices = 20)

{

if (Fill)

{

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

}

else

{

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

}

glPushMatrix();

glTranslated(x0, y0, z0);

double normalFactor = Math.Sqrt(radius\_small \* radius\_small + height \* height);

glBegin(GL\_TRIANGLE\_STRIP);

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

{

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

double x1 = radius\_small \* Math.Cos(theta);

double z1 = radius\_small \* Math.Sin(theta);

double x2 = radius \* Math.Cos(theta);

double z2 = radius \* Math.Sin(theta);

double nx = x1 / normalFactor;

double ny = height / normalFactor;

double nz = z1 / normalFactor;

glColor3ub(250, 0, 0);

glNormal3d(nx, ny, nz);

glVertex3d(x1, 0, z1);

glColor3ub(250, 100, 100);

glNormal3d(nx, ny, nz);

glVertex3d(x2, height, z2);

}

glEnd();

glBegin(GL\_POLYGON);

glColor3ub(100, 150, 250);

glNormal3d(0, -1, 0);

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

{

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

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

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

glVertex3d(x, 0, z);

}

glEnd();

glPopMatrix();

}

public void DrawDisc(double x0, double y0, double z0, double radius\_small, double radius, bool Fill, int slices = 40)

{

if (Fill)

{

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

}

else

{

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

}

glPushMatrix();

glTranslated(x0, y0, z0);

glNormal3d(0, 1, 0);

glBegin(GL\_TRIANGLE\_STRIP);

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

{

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

double x1 = radius\_small \* Math.Cos(theta);

double z1 = radius\_small \* Math.Sin(theta);

double x2 = radius \* Math.Cos(theta);

double z2 = radius \* Math.Sin(theta);

glColor3ub(0, 250, 0);

glVertex3d(x1, 0, z1);

glVertex3d(x2, 0, z2);

}

glEnd();

glPopMatrix();

}

# Додаток Ж. Лістинг програми до практичної роботи №6

### Код файлу (RenderControl.cs)

using System;

using System.Drawing;

using System.Windows.Forms;

namespace OpenGL\_Lab\_6

{

public partial class RenderControl : OpenGL

{

private float size = 1.5f;

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

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

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

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

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

private float Zmin { get => -size; }

private float Zmax { get => +size; }

private float angleX;

private float angleY;

private float m;

private float a = 0.4f;

private float b = 0.7f;

private float c = 0.24f;

private float aw;

private IntPtr qObj;

private float theta

{

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

}

private float psi;

private float phi

{

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

}

private float s = 0.5f;

private float S

{

get { return s; }

set { if((value >= a - c) && (value <= c + a)) s = value; }

}

public RenderControl()

{

InitializeComponent();

MouseWheel += OnMouseWheel;

}

private void OnStart(object sender, EventArgs e)

{

angleX = 0f;

angleY = 0f;

m = 1f;

psi = 15.0f;

aw = 0f;

qObj = gluNewQuadric();

}

private void OnRender(object sender, EventArgs e)

{

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

glLoadIdentity();

glViewport(0, 0, Width, Height);

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

glRotated(angleX, 1, 0, 0);

glRotated(angleY, 0, 1, 0);

glScaled(m, m, m);

// Enable depth

glEnable(GL\_DEPTH\_TEST);

DrawAxis();

DrawGridXY();

DrawGridYZ();

DrawGridZX();

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

glPushMatrix();

// first line

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

Segment(a + b, 0.05f, 1f, 0f, 0f);

// second line

glTranslated(0f, a, 0f);

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

Segment(b, 0.04f, 0f, 1f, 0f);

glPopMatrix();

// third line

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

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

Segment(c, 0.04f, 0f, 0f, 1f);

}

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

{

glColor3d(r, g, b);

glLineWidth(5);

glBegin(GL\_LINES);

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

glVertex3d(0f, height, -width);

glVertex3d(0f, 0f, width);

glVertex3d(0f, height, width);

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

glVertex3d(0f, 0f, width);

glVertex3d(0f, height, -width);

glVertex3d(0f, height, width);

glEnd();

glLineWidth(1);

}

private float padding = 0.1f;

private void DrawAxis()

{

glColor3d(0f, 0f, 0f);

glLineWidth(2);

glBegin(GL\_LINES);

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

glVertex3d(1.5f, 0f, 0f);

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

glVertex3d(0f, 1.5f, 0f);

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

glVertex3d(0f, 0f, 1.5f);

glEnd();

DrawText("X", 1.5f, 0f, 0f);

DrawText("Y", 0f, 1.5f, 0f);

DrawText("Z", 0f, 0f, 1.5f);

glLineWidth(1);

}

private void DrawGridXY()

{

glColor3d(0.4f, 0.4f, 0.4f);

glLineStipple(3, 21845);

glEnable(GL\_LINE\_STIPPLE);

glLineWidth(1);

glBegin(GL\_LINES);

for (float value = 0; value <= 1.5f + padding; value += 0.1f)

{

glVertex3d(value, 0f, -padding);

glVertex3d(value, 1.5f, -padding);

glVertex3d(0f, value, -padding);

glVertex3d(1.5f, value, -padding);

}

glEnd();

glDisable(GL\_LINE\_STIPPLE);

}

private void DrawGridYZ()

{

glColor3d(0.4f, 0.4f, 0.4f);

glLineStipple(3, 21845);

glEnable(GL\_LINE\_STIPPLE);

glLineWidth(1);

glBegin(GL\_LINES);

for (float value = 0; value <= 1.5f + padding; value += 0.1f)

{

glVertex3d(-padding, value, 0f);

glVertex3d(-padding, value, 1.5f);

glVertex3d(-padding, 0f, value);

glVertex3d(-padding, 1.5f, value);

}

glEnd();

glDisable(GL\_LINE\_STIPPLE);

}

private void DrawGridZX()

{

glColor3d(0.4f, 0.4f, 0.4f);

glLineStipple(3, 21845);

glEnable(GL\_LINE\_STIPPLE);

glLineWidth(1);

glBegin(GL\_LINES);

for (float value = 0; value <= 1.5f + padding; value += 0.1f)

{

glVertex3d(value, -padding, 0f);

glVertex3d(value, -padding, 1.5f);

glVertex3d(0f, -padding, value);

glVertex3d(1.5f, -padding, value);

}

glEnd();

glDisable(GL\_LINE\_STIPPLE);

}

private void OnKeyDown(object sender, KeyEventArgs e)

{

switch (e.KeyCode)

{

case Keys.W:

psi--;

break;

case Keys.S:

psi++;

break;

case Keys.E:

S += 0.01f;

break;

case Keys.Q:

S -= 0.01f;

break;

case Keys.A:

aw++;

break;

case Keys.D:

aw--;

break;

}

Invalidate();

}

private bool mouseFlag = false;

private Point mouseStart;

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

{

mouseFlag = e.Button == MouseButtons.Left;

mouseStart = e.Location;

}

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

{

if (mouseFlag)

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

}

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

{

if (mouseFlag)

{

Point current = e.Location;

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

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

mouseStart = current;

Invalidate();

}

}

private void OnMouseWheel(object sender, MouseEventArgs e)

{

m += e.Delta / 2000.0f;

Invalidate();

}

}

}

# Додаток з. Лістинг програми до практичної роботи №7

### Код файлу (RenderControl.cs)

using System;

using System.Drawing;

namespace test1

{

public partial class RenderControl : OpenGL

{

public RenderControl()

{

InitializeComponent();

}

private double proprller = 0.0;

private double proprllerSpeed = 0.1;

private double offsetX = 0.0, offsetY = 0.0;

private double cx = -0.7, cy = 0.27015;

private double cxStep = 0.002, cyStep = 0.002;

private bool isMovingEnabled = true;

private double speedValue = 0.005;

private double sizeValue = 1.0;

private void OnContextCreated(object sender, EventArgs e)

{

glClearColor(Color.Black);

isMovingEnabled = Properties.Settings.Default.IsFloatingFigure;

cx = (double)Properties.Settings.Default.CxValue;

cy = (double)Properties.Settings.Default.CyValue;

speedValue = 0.001 \* Properties.Settings.Default.SpeedValue;

sizeValue = 0.5 \* Properties.Settings.Default.SizeValue;

}

private void OnRender(object sender, EventArgs e)

{

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

glLoadIdentity();

double viewLeft = -2 + offsetX;

double viewRight = 2 + offsetX;

double viewBottom = -2 + offsetY;

double viewTop = 2 + offsetY;

glOrtho(viewLeft, viewRight, viewBottom, viewTop, -1, 1);

DrawProprller();

if (isMovingEnabled)

{

offsetX += speedValue;

offsetY += speedValue;

if (offsetX > 2 || offsetX < -2) speedValue = -speedValue;

cx += cxStep;

cy += cyStep;

if (cx > 0.7 || cx < -0.7) cxStep = -cxStep;

if (cy > 0.7 || cy < -0.7) cyStep = -cyStep;

}

}

private void DrawProprller()

{

int resolution = 500;

double scale = 4.0 / resolution;

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

{

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

{

double x = (i - resolution / 2.0) \* scale + offsetX;

double y = (j - resolution / 2.0) \* scale + offsetY;

double distance = Math.Sqrt(x \* x + y \* y);

double angle = Math.Atan2(y, x);

double radius = 1.5 \* Math.Abs(Math.Sin(angle \* 2.0 + proprller));

double heartX = radius \* Math.Cos(angle);

double heartY = radius \* Math.Sin(angle);

if (distance <= Math.Sqrt(heartX \* heartX + heartY \* heartY))

{

double color = 1.0 - distance / 1.5;

glColor3d(color, color, color);

glPointSize(2.0f);

glBegin(GL\_POINTS);

glVertex2d(x, y);

glEnd();

}

}

}

proprller += proprllerSpeed;

if (proprller > Math.PI \* 2)

proprller -= Math.PI \* 2;

}

}

}

### Код файлу (SettingsForm.cs)

using System;

using System.Diagnostics;

using System.Windows.Forms;

namespace test1

{

public partial class SettingsForm : Form

{

public SettingsForm()

{

InitializeComponent();

}

private void SettingsFormLoad(object sender, EventArgs e)

{

chkbxCustomFigure.Checked = Properties.Settings.Default.IsCustomFigure;

nudCx.Value = Properties.Settings.Default.CxValue;

nudCy.Value = Properties.Settings.Default.CyValue;

chkbxFloatFigure.Checked = Properties.Settings.Default.IsFloatingFigure;

trbarSpeed.Value = Properties.Settings.Default.SpeedValue;

trbarSize.Value = Properties.Settings.Default.SizeValue;

Debug.WriteLine("Load screen saver's settings.");

}

private void SettingsFormClosed(object sender, FormClosedEventArgs e)

{

Properties.Settings.Default.IsCustomFigure = chkbxCustomFigure.Checked;

Properties.Settings.Default.CxValue = nudCx.Value;

Properties.Settings.Default.CyValue = nudCy.Value;

Properties.Settings.Default.IsFloatingFigure = chkbxFloatFigure.Checked;

Properties.Settings.Default.SpeedValue = trbarSpeed.Value;

Properties.Settings.Default.SizeValue = trbarSize.Value;

Properties.Settings.Default.Save();

Debug.WriteLine("Save screen saver's settings.");

}

private void OnChangedCustom(object sender, EventArgs e)

{

if (chkbxCustomFigure.Checked)

{

nudCx.Enabled = true;

nudCy.Enabled = true;

}

else

{

nudCx.Enabled = false;

nudCy.Enabled = false;

}

}

private void OnChangedFloating(object sender, EventArgs e)

{

if (chkbxFloatFigure.Checked)

{

trbarSpeed.Enabled = true;

}

else

{

trbarSpeed.Enabled = false;

}

}

private void OnClickOkBtn(object sender, EventArgs e)

{

Properties.Settings.Default.IsCustomFigure = chkbxCustomFigure.Checked;

Properties.Settings.Default.CxValue = nudCx.Value;

Properties.Settings.Default.CyValue = nudCy.Value;

Properties.Settings.Default.IsFloatingFigure = chkbxFloatFigure.Checked;

Properties.Settings.Default.SpeedValue = trbarSpeed.Value;

Properties.Settings.Default.SizeValue = trbarSize.Value;

Properties.Settings.Default.Save();

}

private void OnClickCancelBtn(object sender, EventArgs e)

{

chkbxCustomFigure.Checked = Properties.Settings.Default.IsCustomFigure;

nudCx.Value = Properties.Settings.Default.CxValue;

nudCy.Value = Properties.Settings.Default.CyValue;

chkbxFloatFigure.Checked = Properties.Settings.Default.IsFloatingFigure;

trbarSpeed.Value = Properties.Settings.Default.SpeedValue;

trbarSize.Value = Properties.Settings.Default.SizeValue;

}

}

}

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