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  *335а*

напряму підготовки (спеціальності):

*122 “Комп’ютерні науки”*

(шифр і назва напряму підготовки / спеціальності)

*Мінаков А.А.*

(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](#_Toc177946558)

[Завдання, варіант № 14 3](#_Toc177946559)

[Системна інформація 3](#_Toc177946560)

[Теоретичні відомості 3](#_Toc177946561)

[Результати виконання практичної роботи](#_Toc177946562) 4

[Практична робота 2. Графічні примітиви OpenGL 8](#_Практична_робота_2.)

[Завдання, варіант № 14 8](#_Завдання,_варіант_№)

[Теоретичні відомості 10](#_Теоретичні_відомості)

[Результати виконання практичної роботи 13](#_Результати_виконання_практичної)

[Загальний перелік посилань 19](#_Toc177946571)

[Додаток А. Лістинг програми до практичної роботи №1 20](#_Toc177946572)

[Додаток Б. Лістинг програми до практичної роботи №2 24](#_Додаток_Б._Лістинг)

# Практична робота 1. Основні принципи роботи з OpenGL

## Завдання, варіант № 14

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.

|  |  |  |
| --- | --- | --- |
| 14 | Примітиви:  GL\_POINTS, GL\_LINE\_STRIP  x1 = -2;  x2 = 7 y1 = -4;  y2 = 0 |  |

## Системна інформація

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

Processor AMD Ryzen 5 3600 6-Core Processor 3.60 GHz

RAM 16.0 GB

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

Edition Windows 10 Pro Version 22H2

IDE Microsoft Visual Studio Community 2022 (64-bit) version 17.8.6

## Теоретичні відомості

### Робота з графічними примітивами

Для відображення графічних елементів у OpenGL використовуються декілька команд, серед яких:

* glColor3d() — для задання кольору примітивів;
* glLineWidth() — для задання товщини ліній;
* glEnable() / glDisable() — для активації та деактивації різних функцій, наприклад, пунктирних ліній;
* glLineStipple() — для налаштування пунктирних ліній.

У нашому коді для малювання координатної сітки, осей та фігури використовувалися ці команди.

### Побудова координатної сітки

Для відображення сітки використовувалися пунктирні лінії, які створюються за допомогою функції glLineStipple(). Сітка дозволяє точно визначати позиції точок на площині та задавати контури фігур. Основні кроки для побудови сітки:

1. Встановлення кольору сітки командою glColor3d().
2. Використання функції glLineWidth() для задання товщини ліній.
3. Активація режиму пунктирних ліній командою glEnable(GL\_LINE\_STIPPLE).
4. Використання вектора для зберігання координат вершин сітки, який передається до OpenGL через функцію glVertexPointer().

### Малювання осей координат

Для побудови осей використовуються аналогічні методи з деякими змінами. Осі відображаються за допомогою товстих ліній, для яких задається інший колір і товщина. У нашому випадку осі зображаються жовтим кольором.

### Малювання полігонів та точок

В програмі передбачено малювання полігонів та окремих точок. Для цього були створені спеціальні класи, такі як Point та MyPolygon. Клас Point відповідає за відображення окремих точок на екрані, тоді як MyPolygon — за малювання полігонів, що складаються з кількох точок.

Для кожного примітиву використовується метод draw(), що викликає необхідні OpenGL-функції для відображення об’єктів.

## Результати виконання практичної роботи

### Розв'язання завдання

1. **Використання команд управління параметрами графічних примітивів:**

Для управління параметрами графічних примітивів були використані наступні команди (Додаток А):

* Колір: glColor3d(), рядки: 21, 49, 62, 112 та 127 у файлі Render.cpp.
* Тип: glLineStipple(), glEnable(), glDisable(), рядки: 23, 24 та 45 у файлі Render.cpp.
* Товщина: glLineWidth(), рядки: 22, 50, 63, 74, 87 та 128 у файлі Render.cpp.

1. **Коректне відображення завдання під час змінення розмірів/положення вікна:**

Коректне відображення завдання під час зміни розмірів вікна продемонстровано на рис. 1.1 та 1.2.

1. **Розроблення підпрограм для виключення дублювання коду:**

Підпрограми для уникнення дублювання коду були розроблені у рядках 20–28 файлу Render.cpp.

1. **Застосування циклів для створення зображень:**

Цикли для автоматичного малювання графічних елементів, таких як сітка, осі і відмітки, використовуються у рядках 35–46 та 86-93 файлу Render.cpp.

1. **Формування зображення векторними командами OpenGL:**

Векторні команди OpenGL, такі як glDrawArrays(), були використані для оптимізації рендерингу. У прикладі малювання сітки, осей та додаткових ліній використовується glDrawArrays() замість багатьох викликів окремих вершин. Це можна знайти у рядку 26 файлу Render.cpp.

1. **Використання ООП (розроблення власних класів):**

Використання об’єктно-орієнтованого підходу було реалізовано за допомогою класів Point та MyPolygon, які наведені у файлі Render.cpp (рядки 102 – 132). Клас Point відповідає за відображення точок, а клас MyPolygon — за побудову фігури.

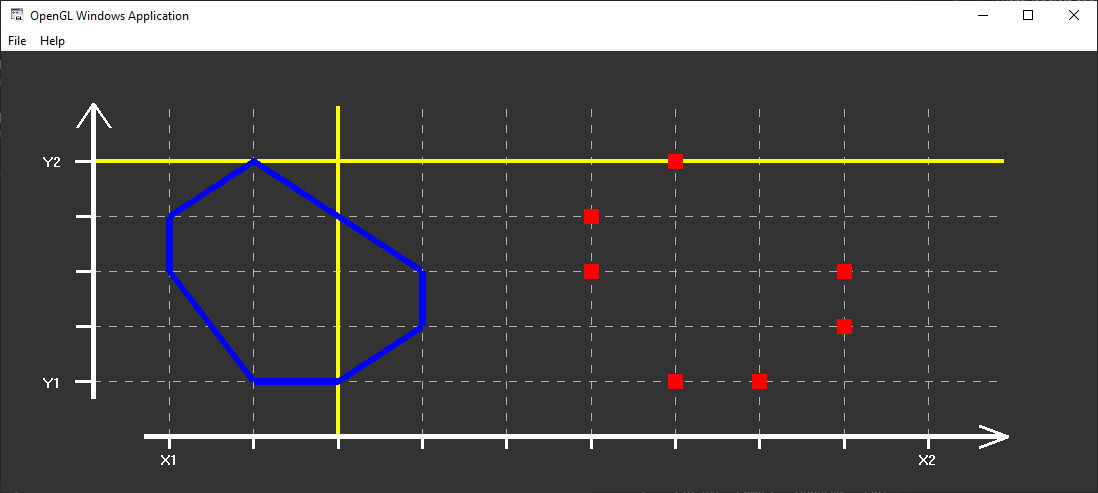


Рисунок 1.1 – Тестування програми при зміні ширини вікна

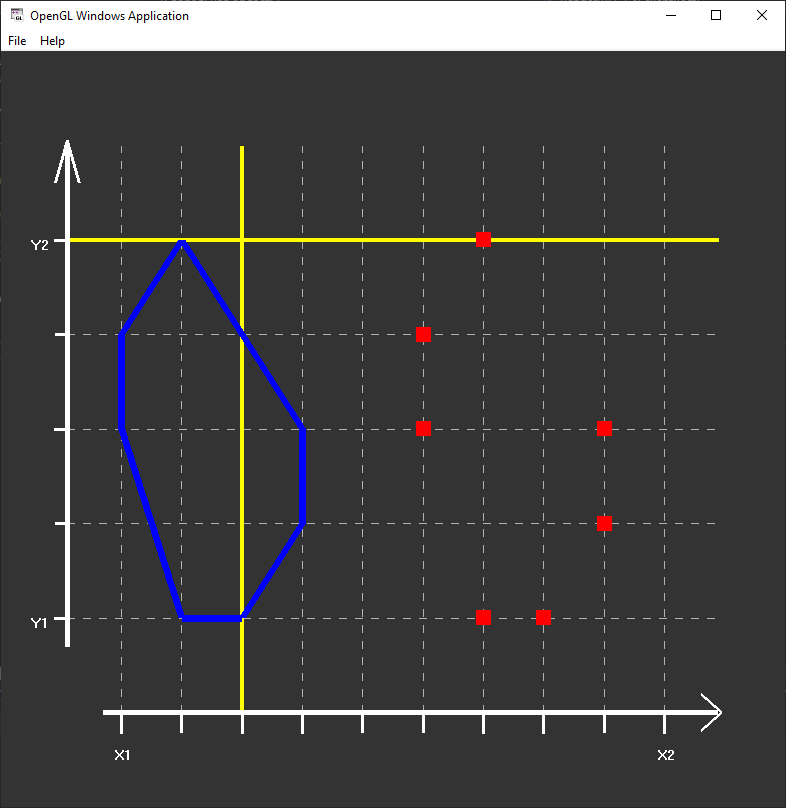


Рисунок 1.2 – Тестування програми при зміні висоти вікна

### Контроль виконання вимог та елементів завдання

В результаті виконання практичної роботи були повністю виконані елементи базового рівня та повністю 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 | **+** |

### Посилання на GitHub

<https://github.com/Mausipupsi/OpenGL>

# Практична робота 2. Графічні примітиви OpenGL

## Завдання, варіант № 14

|  |  |  |
| --- | --- | --- |
|  | Сторона фігури a = 1000  Примітив(и):   GL\_TRIANGLES, GL\_QUAD\_STRIP | 04-02.PNG |

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



Рисунок 2.1 – Вигляд застосунку після старту

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

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



*а*



*б*

Рисунок 2.2 – Вигляд застосунку під час замощення:   
*а* – тільки по горизонталі; *б* – по горизонталі та вертикалі

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



Рисунок 2.3 – Приклад керування режимом виводу графічних примітивів

## Теоретичні відомості

### Windows Forms (WinForms)

Windows Forms — це графічний інтерфейс користувача (GUI) для створення додатків у середовищі .NET. Він надає набір елементів керування (контролів), таких як кнопки, текстові поля, радіокнопки тощо, які можна використовувати для побудови інтерфейсу користувача.

* InitializeComponent(): Метод, який автоматично генерується дизайнером WinForms. Він ініціалізує всі елементи керування на формі, встановлюючи їхні властивості та розташування.

### Події та Обробники Подій

У програмуванні подійна модель дозволяє реагувати на дії користувача, такі як натискання кнопок, зміна значень контролів тощо.

* ValueChanged: Подія, яка виникає при зміні значення елемента керування, наприклад, NumericUpDown. Обробник цієї події використовується для виконання певних дій у відповідь на зміну значення.
* CheckedChanged: Подія, яка виникає при зміні стану радіокнопки (RadioButton). Використовується для визначення, яка радіокнопка вибрана, та відповідного оновлення режиму рендерингу.
* RenderModeChanged: Користувацька подія, яка сповіщає про зміну режиму рендерингу в контролі рендерингу. Дозволяє синхронізувати стан інтерфейсу з поточним режимом рендерингу.

### OpenGL

OpenGL (Open Graphics Library) — це стандартний API для рендерингу 2D та 3D графіки. Він надає функції для малювання геометричних фігур, управління камерою, освітленням та іншими аспектами графічного відображення.

* glClear: Очищує буфери кольору та глибини, підготовлюючи кадр для нової сцени.
* glViewport: Встановлює розмір в'юпорту — області екрану, куди буде рендеритися сцена.
* glMatrixMode: Встановлює поточну матрицю (наприклад, проекційна або модельно-видова).
* glLoadIdentity: Завантажує одиничну матрицю, скидаючи попередні трансформації.
* glOrtho: Встановлює ортографічну проекцію, яка використовується для 2D відображення без перспективних спотворень.
* glBegin / glEnd: Обгортки для визначення початку та кінця малювання примітивів (трикутників, квадрацій тощо).
* glColor3f: Встановлює колір для наступних вершин.
* glVertex2f: Визначає вершину примітива у 2D просторі.
* glFlush: Забезпечує негайне виконання команд рендерингу.

### Режими Рендерингу

Режими рендерингу визначають спосіб, у який OpenGL відображає геометричні примітиви.

* Fill (Заповнення): Примітиви заповнюються кольором, визначеним за допомогою glColor3f. Використовується для відображення суцільних фігур.
* Line (Лінії): Примітиви відображаються лише контурами (гранями), без заповнення. Застосовується для контурних малюнків.
* Point (Точки): Примітиви відображаються як окремі точки. Використовується для створення точкових зображень або ефектів.

### В'юпорт та Проекція

В'юпорт — це область екрану, куди рендериться сцена. Проекція визначає, як тривимірні координати перетворюються у двовимірні координати екрану.

* Ортографічна Проекція (glOrtho): Тип проекції, де паралельні лінії залишаються паралельними після проекції. Підходить для 2D графіки та технічних креслень.
* Співвідношення Сторін (Aspect Ratio): Відношення ширини до висоти в'юпорту. Важливе для збереження пропорцій об'єктів при зміні розміру вікна.

### Малювання Геометричних Фігур

У коді використовується клас ComplexFigure для малювання складних фігур, які складаються з трикутників та квадрейтрипів.

* Трикутники (GL\_TRIANGLES): Основні примітиви для створення складних форм. Три вершини визначають один трикутник.
* Квадрейтрипи (GL\_QUAD\_STRIP): Використовуються для малювання послідовних чотирикутників, спрощуючи процес створення смуг фігур.

### Сітка Фігур

Сітка фігур дозволяє організувати розташування геометричних об'єктів у певній конфігурації, наприклад, гексагональній сітці.

* Гексагональна Сітка: Тип сітки, де кожна фігура (наприклад, шестиугольник) оточена шістьма іншими фігурами. Забезпечує рівномірне розподілення простору та відсутність пропусків.
* Зміщення Позицій (Offsetting Positions): Використовується для створення ефекту гексагонального розміщення, де кожен ряд змінює вертикальну позицію фігур для досягнення щільного розташування.

### Подвійна Буферизація

Подвійна буферизація — техніка, яка використовує два буфери для рендерингу кадрів: один для відображення поточного кадру, а інший для підготовки наступного. Це дозволяє уникнути мерехтіння та забезпечити плавний візуальний перехід між кадрами.

* DoubleBuffered: Властивість контролю WinForms, яка визначає, чи використовувати подвійне буферизування для зменшення мерехтіння при малюванні.

### Класи та Об'єктно-орієнтоване Програмування (ООП)

Код використовує принципи ООП для організації функціональності.

* Клас Button: Представляє кнопку вибору режиму рендерингу. Містить властивості для позиції та режиму, а також методи для малювання та перевірки натискання.
* Клас ComplexFigure: Відповідає за створення та малювання складних геометричних фігур. Використовує методи OpenGL для визначення вигляду фігур залежно від вибраного режиму рендерингу.
* Класи та Інкапсуляція: Використання класів дозволяє інкапсулювати дані та методи, забезпечуючи модульність та повторне використання коду.

### Обробка Подій Миші

Обробка подій миші дозволяє взаємодіяти з користувачем через натискання, переміщення або інші дії миші.

* MouseDown: Подія, яка виникає при натисканні кнопки миші. Використовується для визначення, чи було натиснуто одну з кнопок вибору режиму рендерингу.
* IsPointInside(float x, float y): Метод, який перевіряє, чи знаходиться певна точка (координати миші) всередині області кнопки. Дозволяє визначити, яка кнопка була натиснута.

## Результати виконання практичної роботи

### Розв'язання завдання

1. **Під час запуску застосунку зображення відповідає варіанту завдання з однією плиткою:**

При запуску застосунку за замовчуванням встановлюється кількість плиток по горизонталі та вертикалі рівною 1. Це забезпечує відображення єдиної плитки, яка відповідає початковому варіанту завдання. Файл MainForm.cs, рядки 10-20. Продемонстровано на рис. 1.1 та 1.2.

1. **Багаторазове замощення плиткою. Кратність замощення задається користувачем під час роботи застосунку:**

Користувач може змінювати кількість плиток по горизонталі та вертикалі за допомогою елементів NumericUpDown. Зміни значень викликають оновлення кількості плиток у контролі рендерингу. Багаторазове замощення плиткою продемонстровано на рис. 1.5 та 1.6. Файл MainForm.cs, рядки 22-40 та файл RenderControl.cs, рядки 182-213.

1. **Коректне відображення завдання під час зміни як розмірів/положення вікна, так і параметрів замощення:**

Контроль рендерингу реагує на події зміни розміру вікна та змін параметрів замощення, перерисовуючи зображення відповідно до нових параметрів. Продемонстровано на рис. 1.1 - 1.6. Файл RenderControl.cs, рядки 124-132 та 154-174.

1. **Організація взаємодії з користувачем одним зі стандартних засобів (клавіатура, «миша» та ін.):**

Взаємодія з користувачем здійснюється через мишу. Користувач може натискати на кнопки режиму рендерингу, використовуючи події миші для визначення натискання на відповідні області. Файл RenderControl.cs, рядки 63-82.

1. **Застосування мінімальної (у рамках варіанту) кількості графічних примітивів для виконання завдання:**

Для створення необхідних фігур використовується мінімальна кількість графічних примітивів OpenGL, таких як трикутники та квадрати. Файл RenderControl.cs, рядки 332-357.

1. **Створення власних елементів інтерфейсу за допомогою OpenGL:**

Власні елементи інтерфейсу, такі як кнопки вибору режиму рендерингу, створені та намальовані вручну за допомогою команд OpenGL. Файл RenderControl.cs, рядки 228-301.

1. **Використання ООП (розробка власних класів):**

Об'єктно-орієнтований підхід застосовується через створення власних класів Button та ComplexFigure, які інкапсулюють властивості та методи, необхідні для їхньої функціональності. Файл RenderControl.cs, рядки 228-401.

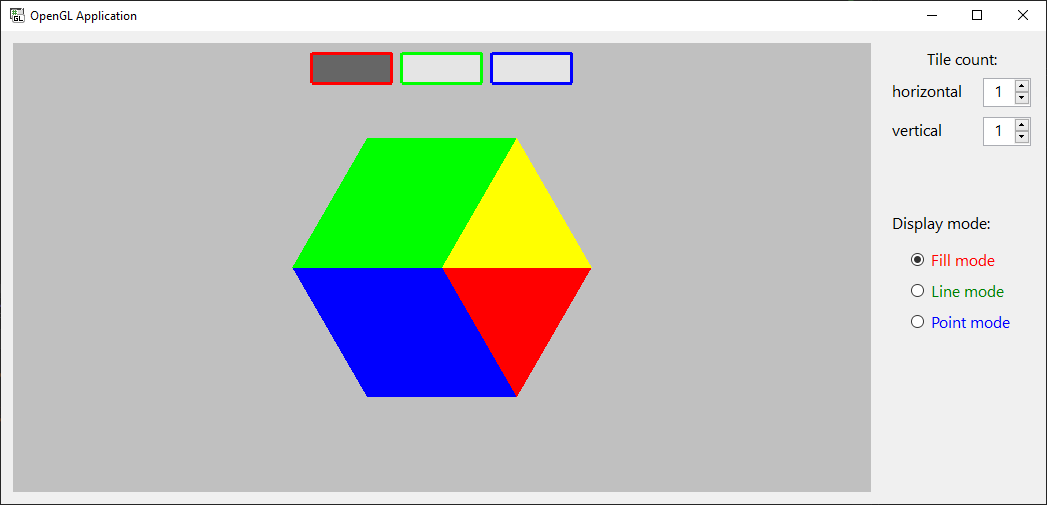


Рисунок 1.1 – Тестування програми при зміні ширини вікна та режимі Fill

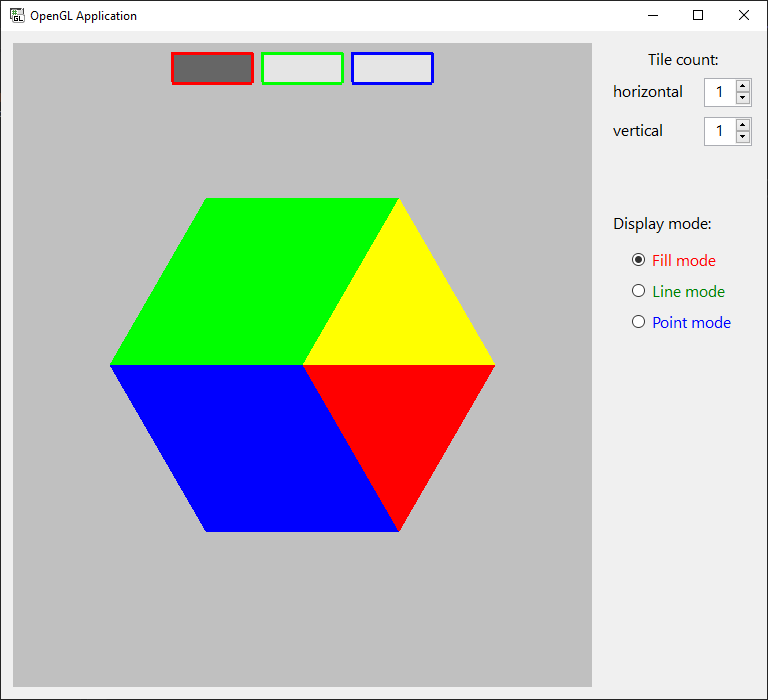


Рисунок 1.2 – Тестування програми при зміні висоти вікна та режимі Fill

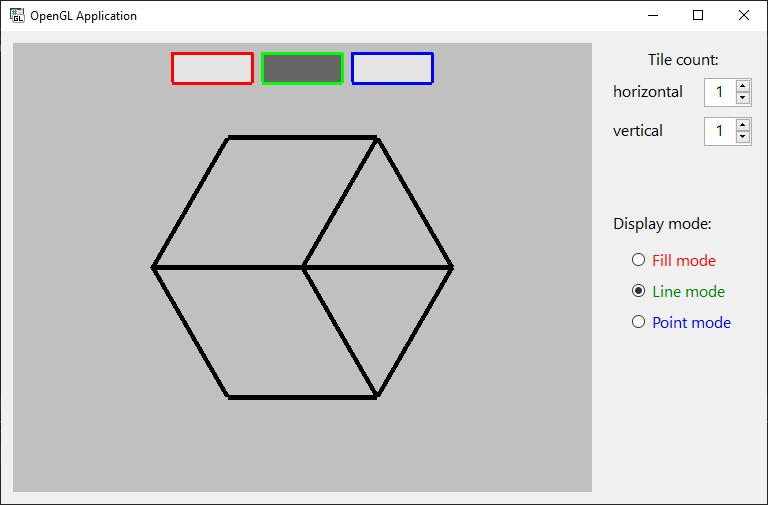


Рисунок 1.3 – Тестування програми в режимі Line

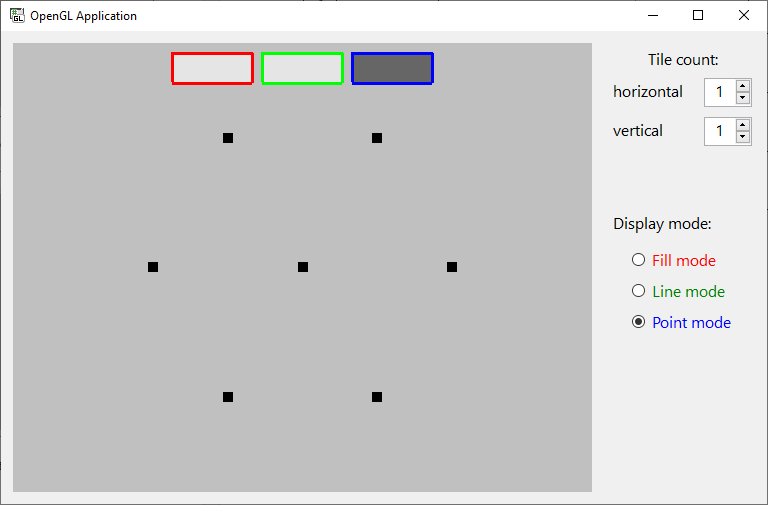


Рисунок 1.4 – Тестування програми в режимі Point

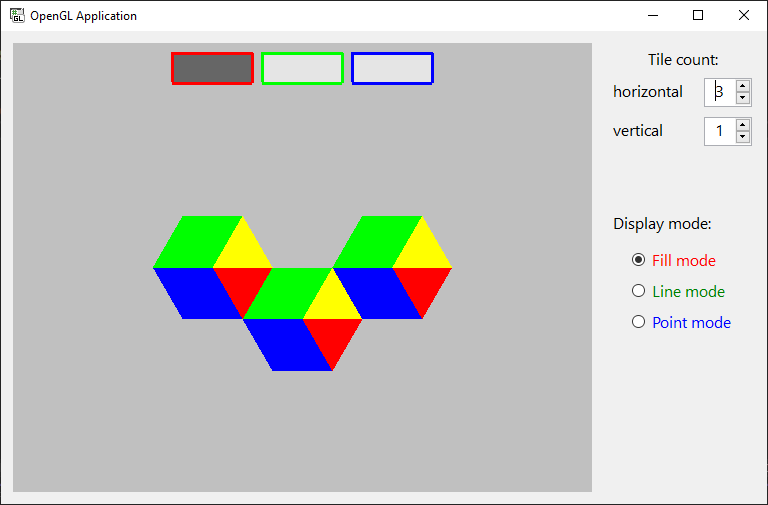


Рисунок 1.5 – Тестування програми при додаванні фігур по горизонталі

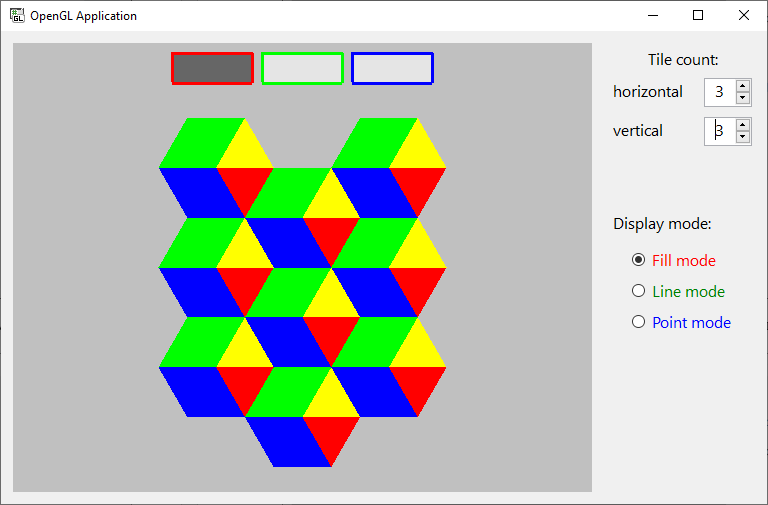


Рисунок 1.6 – Тестування програми при додаванні фігур по вертикалі

### Контроль виконання вимог та елементів завдання

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

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

### Посилання на GitHub

<https://github.com/Mausipupsi/OpenGL>

# Загальний перелік посилань

1. Microsoft. glDrawArrays function [Електронний ресурс] / Microsoft – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/windows/win32/opengl/gldrawarrays>.
2. Microsoft. glLineStipple function [Електронний ресурс] / Microsoft – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/windows/win32/opengl/gllinestipple>
3. Microsoft. glEnable function [Електронний ресурс] / Microsoft – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/windows/win32/opengl/glenable>
4. Microsoft. glDisable function [Електронний ресурс] / Microsoft – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/windows/win32/opengl/gldisable>
5. Microsoft. glColor3d function [Електронний ресурс] / Microsoft – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/windows/win32/opengl/glcolor3d>
6. Microsoft. glLineWidth function [Електронний ресурс] / Microsoft – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/windows/win32/opengl/gllinewidth>
7. Microsoft. glVertexPointer function [Електронний ресурс] / Microsoft – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/windows/win32/opengl/glvertexpointer>
8. Microsoft. glVertex2d function [Електронний ресурс] / Microsoft – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/windows/win32/opengl/glvertex2d>
9. Microsoft. glMatrixMode function [Електронний ресурс] / Microsoft – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/windows/win32/opengl/glmatrixmode>
10. Microsoft. glPolygonMode function [Електронний ресурс] / Microsoft – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/windows/win32/opengl/glpolygonmode>

# Додаток А. Лістинг програми до практичної роботи №1

### Код файлу (Render.cpp)

1. #include "pch.h"
2. #include "glWinApp.h"
3. #include <vector>
4. using namespace std;
5. extern LPCSTR s1, s2, s3;
6. int LoadWindowDefaultFont() {
7. GLuint id = glGenLists(256);
8. wglUseFontBitmaps(wglGetCurrentDC(), 0, 256, id);
9. return id;
10. }
11. void OutText(LPCSTR str, double x, double y, double z = 0) {
12. glRasterPos3d(x, y, z);
13. glListBase(idFont);
14. glCallLists(static\_cast<GLsizei>(strlen(str)), GL\_UNSIGNED\_BYTE, str);
15. }
16. void SetupAndDrawLines(const vector<double>& vertices, float lineWidth, const double\* color) {
17. glColor3dv(color); // Встановлення кольору ліній
18. glLineWidth(lineWidth); // Встановлення ширини ліній
19. glEnableClientState(GL\_VERTEX\_ARRAY); // Включення масиву вершин
20. glVertexPointer(2, GL\_DOUBLE, 0, vertices.data()); // Встановлення вказівника на масив вершин
21. glDrawArrays(GL\_LINES, 0, static\_cast<GLsizei>(vertices.size() / 2)); // Малювання ліній
22. glDisableClientState(GL\_VERTEX\_ARRAY); // Вимкнення масиву вершин
23. }
24. void DrawGrid(double xMin, double xMax, double yMin, double yMax, double step) {
25. glEnable(GL\_LINE\_STIPPLE); // Включення режиму пунктирних ліній
26. glLineStipple(1, 0x00FF); // Встановлення шаблону пунктирних ліній
27. vector<double> vertices; // Створення вектора для зберігання вершин сітки
28. for (double x = xMin; x <= xMax; x += step) {
29. vertices.push\_back(x); // Додавання координат вершин сітки по осі X
30. vertices.push\_back(yMin - 1); // Додавання координат вершин сітки по осі Y
31. vertices.push\_back(x); // Додавання координат вершин сітки по осі X
32. vertices.push\_back(yMax + 1); // Додавання координат вершин сітки по осі Y
33. }
34. for (double y = yMin; y <= yMax; y += step) {
35. vertices.push\_back(xMin - 0.9); // Додавання координат вершин сітки по осі X
36. vertices.push\_back(y); // Додавання координат вершин сітки по осі Y
37. vertices.push\_back(xMax + 0.9); // Додавання координат вершин сітки по осі X
38. vertices.push\_back(y); // Додавання координат вершин сітки по осі Y
39. }
40. double gridColor[] = { 0.7, 0.7, 0.7 }; // Колір сітки
41. SetupAndDrawLines(vertices, 1, gridColor); // Налаштування та малювання ліній сітки
42. glDisable(GL\_LINE\_STIPPLE); // Вимкнення режиму пунктирних ліній
43. }
44. void DrawAxes(double xMin, double xMax, double yMin, double yMax) {
45. vector<double> vertices = {
46. xMin - 0.9, 0, xMax + 0.9, 0, // Координати вершин осі X
47. 0, yMin - 1, 0, yMax + 1 // Координати вершин осі Y
48. };
49. double axesColor[] = { 1, 1, 0 }; // Колір осей
50. SetupAndDrawLines(vertices, 4, axesColor); // Налаштування та малювання осей
51. vector<double> customLines = {
52. xMin - 0.9, yMin - 0.3, xMin - 0.9, yMax + 1, // Координати додаткових ліній
53. xMin - 0.3, yMin - 1, xMax + 0.9, yMin - 1 // Координати додаткових ліній
54. };
55. double customLinesColor[] = { 1, 1, 1 }; // Колір додаткових ліній
56. SetupAndDrawLines(customLines, 5, customLinesColor); // Налаштування та малювання додаткових ліній
57. glLineWidth(3); // Встановлення ширини ліній
58. glBegin(GL\_LINES); // Початок малювання ліній
59. glVertex2d(xMin - 0.9, yMax + 1.05); // Координати вершин стрілок осі Y
60. glVertex2d(xMin - 1.1, yMax + 0.6); // Координати вершин стрілок осі Y
61. glVertex2d(xMin - 0.9, yMax + 1.05); // Координати вершин стрілок осі Y
62. glVertex2d(xMin - 0.7, yMax + 0.6); // Координати вершин стрілок осі Y
63. glVertex2d(xMax + 0.95, yMin - 1); // Координати вершин стрілок осі X
64. glVertex2d(xMax + 0.6, yMin - 1.2); // Координати вершин стрілок осі X
65. glVertex2d(xMax + 0.95, yMin - 1); // Координати вершин стрілок осі X
66. glVertex2d(xMax + 0.6, yMin - 0.8); // Координати вершин стрілок осі X
67. glEnd(); // Кінець малювання ліній
68. glLineWidth(3); // Встановлення ширини ліній
69. glBegin(GL\_LINES); // Початок малювання ліній
70. for (double y = yMin; y <= yMax; y++) {
71. glVertex2d(xMin - 0.9, y); // Координати вершин поділок осі Y
72. glVertex2d(xMin - 1.12, y); // Координати вершин поділок осі Y
73. }
74. for (double x = xMin; x <= xMax; x++) {
75. glVertex2d(x, yMin - 1.22); // Координати вершин поділок осі X
76. glVertex2d(x, yMin - 1); // Координати вершин поділок осі X
77. }
78. glEnd(); // Кінець малювання ліній
79. OutText("Y1", xMin - 1.5, yMin - 0.1); // Відображення тексту "Y1" на осі Y
80. OutText("Y2", xMin - 1.5, yMax - 0.1); // Відображення тексту "Y2" на осі Y
81. OutText("X1", xMin - 0.1, yMin - 1.5); // Відображення тексту "X1" на осі X
82. OutText("X2", xMax - 0.1, yMin - 1.5); // Відображення тексту "X2" на осі X
83. }
84. class Point {
85. public:
86. double x, y; // Координати точки
87. Point(double x = 0, double y = 0) : x(x), y(y) {} // Конструктор з параметрами за замовчуванням
88. void draw() const {
89. glColor3d(1, 0, 0); // Встановлення червоного кольору для точки
90. glPointSize(15); // Встановлення розміру точки
91. glBegin(GL\_POINTS); // Початок малювання точки
92. glVertex2d(x, y); // Встановлення координат точки
93. glEnd(); // Кінець малювання точки
94. }
95. };
96. class MyPolygon {
97. public:
98. vector<Point> points; // Вектор точок, що складають полігон
99. MyPolygon(const vector<Point>& points) : points(points) {} // Конструктор з ініціалізацією вектора точок
100. void draw() const {
101. glColor3d(0, 0, 1); // Встановлення синього кольору для полігону
102. glLineWidth(7); // Встановлення ширини ліній полігону
103. glBegin(GL\_LINE\_STRIP); // Початок малювання полігону
104. for (const auto& point : points) {
105. glVertex2d(point.x, point.y); // Встановлення координат точок полігону
106. }
107. glEnd(); // Кінець малювання полігону
108. }
109. };
110. void Render(RECT& clientRect) {
111. glClearColor(0.2f, 0.2f, 0.2f, 1.0f); // Встановлення кольору фону
112. glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT | GL\_STENCIL\_BUFFER\_BIT); // Очищення буферів кольору, глибини та трафарету
113. glLoadIdentity(); // Скидання поточної матриці
114. int Height = clientRect.bottom - clientRect.top; // Визначення висоти вікна
115. int Width = clientRect.right - clientRect.left; // Визначення ширини вікна
116. double xMin = -2, xMax = 7, yMin = -4, yMax = 0; // Встановлення меж координатної системи
117. glViewport(0, 0, Width, Height); // Встановлення області перегляду
118. gluOrtho2D(xMin - 2.0, xMax + 2.0, yMin - 2.0, yMax + 2.0); // Встановлення ортогональної проекції
119. DrawGrid(xMin, xMax, yMin, yMax, 1.0); // Малювання сітки
120. DrawAxes(xMin, xMax, yMin, yMax); // Малювання осей
121. vector<Point> polygonPoints = {
122. Point(-1, -4), Point(-2, -2), Point(-2, -1), Point(-1, 0), Point(1, -2), Point(1, -3), Point(0, -4), Point(-1, -4)
123. }; // Визначення точок полігону
124. vector<Point> points = {
125. Point(4, -4), Point(3, -2), Point(3, -1), Point(4, 0), Point(6, -2), Point(6, -3), Point(5, -4), Point(4, -4)
126. }; // Визначення окремих точок
127. MyPolygon polygon(polygonPoints); // Створення об'єкта полігону
128. polygon.draw(); // Малювання полігону
129. for (const auto& point : points) {
130. point.draw(); // Малювання окремих точок
131. }
132. }

# Додаток Б. Лістинг програми до практичної роботи №2

### Код файлу (MainForm.cs)

1. using System;
2. using System.Windows.Forms;
3. using static Minakov\_Lab2.OpenGL;
4. namespace Minakov\_Lab2
5. {
6. public partial class MainForm : Form
7. {
8. // Конструктор головної форми
9. public MainForm()
10. {
11. InitializeComponent();
12. // Встановлення початкових значень для числових контролів
13. numericUpDownHorizontal.Value = 1;
14. numericUpDownVertical.Value = 1;
15. // Встановлення радіокнопки "Fill" за замовчуванням
16. radioButtonFill.Checked = true;
17. // Підписка на подію зміни режиму рендерингу
18. renderControl.RenderModeChanged += RenderControl\_RenderModeChanged;
19. }
20. // Обробник зміни значення горизонтальної кількості плиток
21. private void NumericUpDownHorizontal\_ValueChanged(object sender, EventArgs e)
22. {
23. UpdateTileCount();
24. }
25. // Обробник зміни значення вертикальної кількості плиток
26. private void NumericUpDownVertical\_ValueChanged(object sender, EventArgs e)
27. {
28. UpdateTileCount();
29. }
30. // Метод оновлення кількості плиток на основі введених значень
31. private void UpdateTileCount()
32. {
33. int horizontal = (int)numericUpDownHorizontal.Value;
34. int vertical = (int)numericUpDownVertical.Value;
35. renderControl.UpdateTileCount(horizontal, vertical);
36. }
37. // Обробник зміни стану радіокнопки "Fill"
38. private void RadioButtonFill\_CheckedChanged(object sender, EventArgs e)
39. {
40. if (radioButtonFill.Checked)
41. {
42. renderControl.SetRenderMode(RenderControl.RenderMode.Fill);
43. }
44. }
45. // Обробник зміни стану радіокнопки "Line"
46. private void RadioButtonLine\_CheckedChanged(object sender, EventArgs e)
47. {
48. if (radioButtonLine.Checked)
49. {
50. renderControl.SetRenderMode(RenderControl.RenderMode.Line);
51. }
52. }
53. // Обробник зміни стану радіокнопки "Point"
54. private void RadioButtonPoint\_CheckedChanged(object sender, EventArgs e)
55. {
56. if (radioButtonPoint.Checked)
57. {
58. renderControl.SetRenderMode(RenderControl.RenderMode.Point);
59. }
60. }
61. // Обробник події зміни режиму рендерингу в контролі рендерингу
62. private void RenderControl\_RenderModeChanged(object sender, RenderControl.RenderModeChangedEventArgs e)
63. {
64. switch (e.NewRenderMode)
65. {
66. case RenderControl.RenderMode.Fill:
67. radioButtonFill.Checked = true;
68. break;
69. case RenderControl.RenderMode.Line:
70. radioButtonLine.Checked = true;
71. break;
72. case RenderControl.RenderMode.Point:
73. radioButtonPoint.Checked = true;
74. break;
75. }
76. }
77. }
78. }

### Код файлу (RenderControl.cs)

1. using System;
2. using System.Collections.Generic;
3. using System.Windows.Forms;
4. using static Minakov\_Lab2.OpenGL;
5. using static Minakov\_Lab2.RenderControl;
6. namespace Minakov\_Lab2
7. {
8. public partial class RenderControl : OpenGL
9. {
10. // Кількість плиток по горизонталі та вертикалі
11. private int horizontalCount = 1;
12. private int verticalCount = 1;
13. // Поточний режим рендерингу
14. private RenderMode currentMode = RenderMode.Fill;
15. // Перелік можливих режимів рендерингу
16. public enum RenderMode
17. {
18. Fill,
19. Line,
20. Point
21. }
22. // Подія зміни режиму рендерингу
23. public event EventHandler<RenderModeChangedEventArgs> RenderModeChanged;
24. // Клас аргументів для події зміни режиму рендерингу
25. public class RenderModeChangedEventArgs : EventArgs
26. {
27. public RenderMode NewRenderMode { get; private set; }
28. public RenderModeChangedEventArgs(RenderMode newRenderMode)
29. {
30. NewRenderMode = newRenderMode;
31. }
32. }
33. // Список кнопок для вибору режиму рендерингу
34. private readonly List<Button> buttons = new();
35. // Конструктор контролу рендерингу
36. public RenderControl()
37. {
38. InitializeComponent();
39. InitializeButtons();
40. // Підписка на події зміни розміру, натискання миші та малювання
41. this.Resize += new EventHandler(RenderControl\_Resize);
42. this.MouseDown += new MouseEventHandler(RenderControl\_MouseDown);
43. this.Paint += new PaintEventHandler(RenderControl\_Paint);
44. this.DoubleBuffered = false;
45. SetStyle(ControlStyles.Opaque, true);
46. }
47. // Обробник події зміни розміру контролу
48. private void RenderControl\_Resize(object sender, EventArgs e)
49. {
50. InitializeButtons();
51. Invalidate();
52. }
53. // Обробник події натискання миші
54. private void RenderControl\_MouseDown(object sender, MouseEventArgs e)
55. {
56. float mouseX = e.X;
57. float mouseY = Height - e.Y;
58. // Перевірка, чи натиснута кнопка
59. foreach (var button in buttons)
60. {
61. if (button.IsPointInside(mouseX, mouseY))
62. {
63. if (currentMode != button.Mode)
64. {
65. currentMode = button.Mode;
66. Invalidate();
67. OnRenderModeChanged(new RenderModeChangedEventArgs(currentMode));
68. }
69. break;
70. }
71. }
72. }
73. // Метод виклику події зміни режиму рендерингу
74. protected virtual void OnRenderModeChanged(RenderModeChangedEventArgs e)
75. {
76. RenderModeChanged?.Invoke(this, e);
77. }
78. // Обробник події малювання контролу
79. private void RenderControl\_Paint(object sender, PaintEventArgs e)
80. {
81. RenderControl\_Render(sender, e);
82. }
83. // Ініціалізація кнопок вибору режиму рендерингу
84. private void InitializeButtons()
85. {
86. buttons.Clear();
87. float buttonWidth = 80f;
88. float buttonHeight = 30f;
89. float spacing = 10f;
90. float totalWidth = 3 \* buttonWidth + 2 \* spacing;
91. float startX = (Width - totalWidth) / 2f;
92. float startY = Height - buttonHeight - 10f;
93. // Створення трьох кнопок: Fill, Line, Point
94. buttons.Add(new Button(startX, startY, startX + buttonWidth, startY + buttonHeight, "Fill", RenderMode.Fill));
95. buttons.Add(new Button(startX + buttonWidth + spacing, startY, startX + 2 \* buttonWidth + spacing, startY + buttonHeight, "Line", RenderMode.Line));
96. buttons.Add(new Button(startX + 2 \* (buttonWidth + spacing), startY, startX + 3 \* buttonWidth + 2 \* spacing, startY + buttonHeight, "Point", RenderMode.Point));
97. }
98. // Метод рендерингу контролу
99. private void RenderControl\_Render(object sender, EventArgs e)
100. {
101. glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT); // Очищення буферів
102. SetupViewport();
103. DrawButtons(); // Малювання кнопок
104. DrawFigures(); // Малювання фігур
105. glFlush();
106. }
107. // Налаштування в'юпорту та проекції
108. private void SetupViewport()
109. {
110. glViewport(0, 0, Width, Height);
111. glMatrixMode(GL\_PROJECTION);
112. glLoadIdentity();
113. glOrtho(0, Width, 0, Height, -1, 1);
114. glMatrixMode(GL\_MODELVIEW);
115. glLoadIdentity();
116. }
117. // Метод малювання кнопок
118. private void DrawButtons()
119. {
120. glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_FILL);
121. glLineWidth(1f);
122. glPointSize(1f);
123. glDisable(GL\_LINE\_STIPPLE);
124. glDisable(GL\_POINT\_SMOOTH);
125. // Малювання кожної кнопки
126. foreach (var button in buttons)
127. {
128. bool isSelected = button.Mode == currentMode;
129. button.Draw(isSelected);
130. }
131. }
132. // Метод малювання фігур залежно від режиму
133. private void DrawFigures()
134. {
135. glMatrixMode(GL\_PROJECTION);
136. glLoadIdentity();
137. float aspectRatio = (float)Width / Height;
138. float a = 1000;
139. float figureWidth = a \* 2f;
140. float figureHeight = (float)(2f \* Math.Sqrt(a \* a - a \* a / 4f));
141. float horizontalSpacing = a + a / 2f;
142. float totalWidth = figureWidth + (horizontalCount - 1) \* horizontalSpacing;
143. float totalHeight = figureHeight \* verticalCount;
144. float maxDimension = Math.Max(totalWidth, totalHeight);
145. float halfSize = maxDimension \* 0.75f;
146. // Налаштування ортотичної проекції з урахуванням співвідношення сторін
147. if (aspectRatio >= 1.0f)
148. {
149. glOrtho(-halfSize \* aspectRatio, halfSize \* aspectRatio, -halfSize, halfSize, -1, 1);
150. }
151. else
152. {
153. glOrtho(-halfSize, halfSize, -halfSize / aspectRatio, halfSize / aspectRatio, -1, 1);
154. }
155. glMatrixMode(GL\_MODELVIEW);
156. glLoadIdentity();
157. float startX = -totalWidth / 2f;
158. float startY = totalHeight / 2f - figureHeight / 2f;
159. // Малювання сітки фігур
160. for (int row = 0; row < verticalCount; row++)
161. {
162. float xPos = startX;
163. float yPos = startY - row \* figureHeight;
164. for (int col = 0; col < horizontalCount; col++)
165. {
166. var figure = new ComplexFigure(xPos, yPos, a, figureHeight);
167. figure.Draw(currentMode);
168. // Зміщення позиції
169. if (col % 2 == 0)
170. {
171. xPos += horizontalSpacing;
172. yPos -= figureHeight / 2f;
173. }
174. else
175. {
176. xPos += horizontalSpacing;
177. yPos += figureHeight / 2f;
178. }
179. }
180. }
181. }
182. // Метод оновлення кількості плиток та перемалювання
183. public void UpdateTileCount(int horizontal, int vertical)
184. {
185. horizontalCount = horizontal;
186. verticalCount = vertical;
187. Invalidate();
188. }
189. // Метод встановлення нового режиму рендерингу
190. public void SetRenderMode(RenderMode mode)
191. {
192. if (currentMode != mode)
193. {
194. currentMode = mode;
195. Invalidate();
196. OnRenderModeChanged(new RenderModeChangedEventArgs(currentMode));
197. }
198. }
199. }
200. // Клас, що представляє кнопку вибору режиму рендерингу
201. public class Button
202. {
203. public float X1 { get; set; }
204. public float Y1 { get; set; }
205. public float X2 { get; set; }
206. public float Y2 { get; set; }
207. public string Text { get; set; }
208. public RenderControl.RenderMode Mode { get; set; }
209. // Конструктор кнопки
210. public Button(float x1, float y1, float x2, float y2, string text, RenderControl.RenderMode mode)
211. {
212. X1 = x1;
213. Y1 = y1;
214. X2 = x2;
215. Y2 = y2;
216. Text = text;
217. Mode = mode;
218. }
219. // Метод малювання кнопки
220. public void Draw(bool isSelected)
221. {
222. // Встановлення кольору фону кнопки
223. if (isSelected)
224. {
225. glColor3f(0.4f, 0.4f, 0.4f); // Темніший фон для вибраної кнопки
226. }
227. else
228. {
229. glColor3f(0.9f, 0.9f, 0.9f); // Світліший фон для невибраних кнопок
230. }
231. // Малювання прямокутника кнопки
232. glBegin(GL\_QUADS);
233. glVertex2f(X1, Y1);
234. glVertex2f(X2, Y1);
235. glVertex2f(X2, Y2);
236. glVertex2f(X1, Y2);
237. glEnd();
238. // Встановлення кольору обводки в залежності від режиму
239. switch (Mode)
240. {
241. case RenderMode.Fill:
242. glColor3f(1f, 0f, 0f); // Червоний для Fill
243. break;
244. case RenderMode.Line:
245. glColor3f(0f, 1f, 0f); // Зелений для Line
246. break;
247. case RenderMode.Point:
248. glColor3f(0f, 0f, 1f); // Синій для Point
249. break;
250. default:
251. glColor3f(0f, 0f, 0f); // Чорний за замовчуванням
252. break;
253. }
254. // Малювання обводки кнопки
255. glLineWidth(3f);
256. glBegin(GL\_LINE\_LOOP);
257. glVertex2f(X1, Y1);
258. glVertex2f(X2, Y1);
259. glVertex2f(X2, Y2);
260. glVertex2f(X1, Y2);
261. glEnd();
262. }
263. // Метод перевірки, чи знаходиться точка всередині кнопки
264. public bool IsPointInside(float x, float y)
265. {
266. return x >= X1 && x <= X2 && y >= Y1 && y <= Y2;
267. }
268. }
269. // Клас, що представляє фігуру для малювання
270. public class ComplexFigure
271. {
272. public float X { get; set; }
273. public float Y { get; set; }
274. public float A { get; set; }
275. public float FigureHeight { get; set; }
276. // Конструктор фігури
277. public ComplexFigure(float x, float y, float a, float figureHeight)
278. {
279. X = x;
280. Y = y;
281. A = a;
282. FigureHeight = figureHeight / 2f;
283. }
284. // Метод малювання фігури залежно від режиму
285. public void Draw(RenderControl.RenderMode mode)
286. {
287. SetRenderMode(mode);
288. // Встановлення кольору для ліній або точок
289. if (mode == RenderControl.RenderMode.Line || mode == RenderControl.RenderMode.Point)
290. {
291. glColor3f(0f, 0f, 0f); // Чорний колір
292. }
293. // Малювання трикутників
294. glBegin(GL\_TRIANGLES);
295. if (mode == RenderControl.RenderMode.Fill)
296. {
297. DrawTriangle(X + A + A / 2f, Y + FigureHeight, X + A + A, Y, X + A, Y, 1f, 1f, 0f);
298. DrawTriangle(X + A + A / 2f, Y - FigureHeight, X + A + A, Y, X + A, Y, 1f, 0f, 0f);
299. }
300. else
301. {
302. DrawTriangle(X + A + A / 2f, Y + FigureHeight, X + A + A, Y, X + A, Y, 0f, 0f, 0f);
303. DrawTriangle(X + A + A / 2f, Y - FigureHeight, X + A + A, Y, X + A, Y, 0f, 0f, 0f);
304. }
305. glEnd();
306. // Малювання квадрейтрипів
307. glBegin(GL\_QUAD\_STRIP);
308. if (mode == RenderControl.RenderMode.Fill)
309. {
310. DrawQuadStrip(X, Y, X + A / 2f, Y + FigureHeight, X + A, Y, X + A + A / 2f, Y + FigureHeight, 0f, 1f, 0f);
311. DrawQuadStrip(X, Y, X + A, Y, X + A / 2f, Y - FigureHeight, X + A + A / 2f, Y - FigureHeight, 0f, 0f, 1f);
312. }
313. else
314. {
315. DrawQuadStrip(X, Y, X + A / 2f, Y + FigureHeight, X + A, Y, X + A + A / 2f, Y + FigureHeight, 0f, 0f, 0f);
316. DrawQuadStrip(X, Y, X + A, Y, X + A / 2f, Y - FigureHeight, X + A + A / 2f, Y - FigureHeight, 0f, 0f, 0f);
317. }
318. glEnd();
319. }
320. // Допоміжний метод малювання трикутника з кольором
321. private static void DrawTriangle(float x1, float y1, float x2, float y2, float x3, float y3, float r, float g, float b)
322. {
323. glColor3f(r, g, b);
324. glVertex2f(x1, y1);
325. glVertex2f(x2, y2);
326. glVertex2f(x3, y3);
327. }
328. // Допоміжний метод малювання квадрейтрипа з кольором
329. private static void DrawQuadStrip(float x1, float y1, float x2, float y2, float x3, float y3, float x4, float y4, float r, float g, float b)
330. {
331. glColor3f(r, g, b);
332. glVertex2f(x1, y1);
333. glVertex2f(x2, y2);
334. glVertex2f(x3, y3);
335. glVertex2f(x4, y4);
336. }
337. // Метод встановлення режиму рендерингу
338. private static void SetRenderMode(RenderControl.RenderMode mode)
339. {
340. switch (mode)
341. {
342. case RenderControl.RenderMode.Fill:
343. glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_FILL);
344. glLineWidth(1f);
345. glPointSize(1f);
346. break;
347. case RenderControl.RenderMode.Line:
348. glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE);
349. glLineWidth(5f);
350. glPointSize(1f);
351. break;
352. case RenderControl.RenderMode.Point:
353. glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_POINT);
354. glLineWidth(1f);
355. glPointSize(10f);
356. break;
357. }
358. }
359. }
360. }