МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського

«Харківський авіаційний інститут»

Факультет програмної інженерії та бізнесу

Кафедра інженерії програмного забезпечення

Практичні роботи

Minor *«Розробник ігрових додатків»*

дисципліна *«Комп’ютерна графіка з OpenGL»*

(назва дисципліни)

Виконав: студент 3 курсу групи  *632п*

напряму підготовки (спеціальності):

*121 інженерія програмного забезпечення*

(шифр і назва напряму підготовки / спеціальності)

*Ільченко О.О.*

(прізвище й ініціали студента)

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

(посада, науковий ступінь, прізвище й ініціали)

Національна шкала:

Кількість балів:

Оцінка ECTS:

Зміст

[Практична робота 1. Основні принципи роботи з OpenGL 3](#_Toc186139889)

[Завдання, варіант № 12 3](#_Toc186139890)

[Системна інформація 3](#_Toc186139891)

[Теоретичні відомості 3](#_Toc186139892)

[Результати виконання практичної роботи 5](#_Toc186139893)

[Загальний перелік посилань 8](#_Toc186139894)

[Додаток А. Лістинг програми до практичної роботи №1 9](#_Toc186139895)

# Практична робота 1. Основні принципи роботи з OpenGL

## Завдання, варіант № 12

За допомогою інструментальних засобів, зазначених викладачем, створити простий програмний проєкт із підтримкою бібліотеки OpenGL. Розробити програму із застосуванням команд OpenGL, яка встановлює анізотропну систему координат, створює та виводить варіант зображення на екран/у вікно з урахуванням заданих примітивів та координат x1, y1 та x2, y2 . Для рисування координатної сітки необхідно використовувати пунктирні лінії. Контур фігури, осі та координатну сітку зобразити лініями різної товщини. Для парних варіантів точки повинні мати квадратну форму, а для непарних – круглу.

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

## Системна інформація

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

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

RAM 16.0 GB

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

Edition Windows 10 Pro Version 22H2

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

## Теоретичні відомості

### Вершинні масиви

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

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

glVertexPointer(): вказати покажчик на масив вершинних координат

glNormalPointer(): вказати покажчик на звичайний масив

glColorPointer(): вказати вказівник на масив кольорів RGB

glIndexPointer(): вказати покажчик на індексований кольоровий масив

glTexCoordPointer(): вказати покажчик на масив текстурних шнурів

glEdgeFlagPointer(): вказати вказівник на масив прапорців edge

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

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

### Команда glDrawArrays()

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

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

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

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

...

// activate and specify pointer to vertex array

glEnableClientState(GL\_VERTEX\_ARRAY);

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

// draw a cube

glDrawArrays(GL\_TRIANGLES, 0, 36);

// deactivate vertex arrays after drawing

glDisableClientState(GL\_VERTEX\_ARRAY);

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

### Команда glDrawElements()

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

…

## Результати виконання практичної роботи

### Розв'язання завдання

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

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

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

товщина glLineWidth(), рядок ?? у файлі ?????.cs

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

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

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

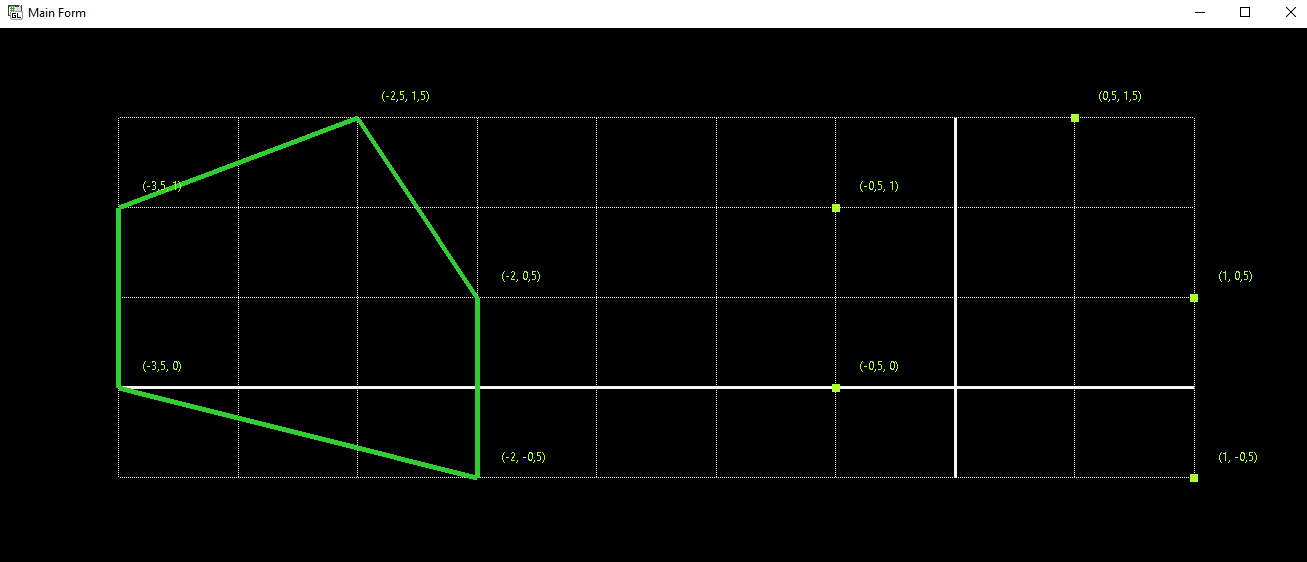


Рисунок 1.1 – Тестування програми при зміні ширини вікна



Рисунок 1.2 – Тестування програми при зміні висоти вікна

### Контроль виконання вимог та елементів завдання

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

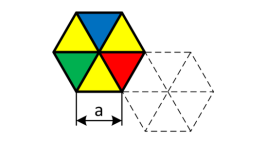
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблиця 1.1 | | | | |
| № з/п | Складність | Вимоги | Бали | Зроблено |
| 1 | Базовий рівень | Використання команд управління параметрами графічних примітивів (колір, тип, товщина) | 2 | **+** |
| 2 | Коректне відображення завдання під час змінення розмірів/положення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

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

**Завдання, варіант № 12**

**Сторона фігури** a = 100

**Примітив:**GL\_TRIANGLES

****

За допомогою інструментальних засобів, зазначених викладачем, створити простий програмний проєкт із підтримкою бібліотеки OpenGL. Враховуючи систему оцінки (табл. 1.1), розробити програму із застосуванням команд OpenGL, яка встановлює анізотропну систему координат, створює та виводить варіант зображення (табл. 1.2) на екран/у вікно з урахуванням заданих примітивів та координат x1, y1 та x2, y2 . Для рисування координатної сітки необхідно використовувати пунктирні лінії. Контур фігури, осі та координатну сітку зобразити лініями різної товщини. Для парних варіантів точки повинні мати квадратну форму, а для непарних – круглу.

**Теоретичні відомості**

**1. OpenGL та його Контекст**

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

**2. Буфери OpenGL**

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

**3. Шейдери**

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

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

**4. Графічні Примітиви**

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

**5. Текстури**

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

**Розбір Коду**

**1. Клас MainForm**

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

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

**2. Клас RenderControl**

Цей клас відповідає за рендеринг графіки з використанням OpenGL.

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

**Результати виконання практичної роботи**

***Розв'язання завдання***

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

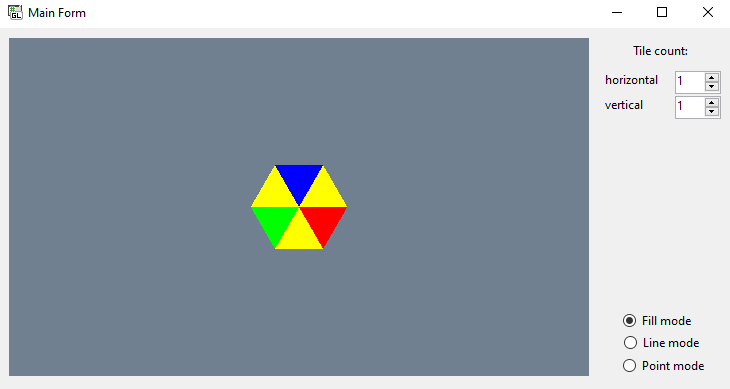


Рисунок 2.1 – Запуск застосунка

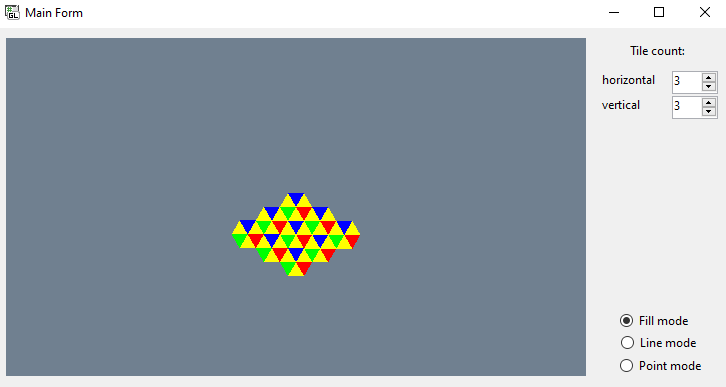


Рисунок 2.2 – Багаторазове замощення плиткою

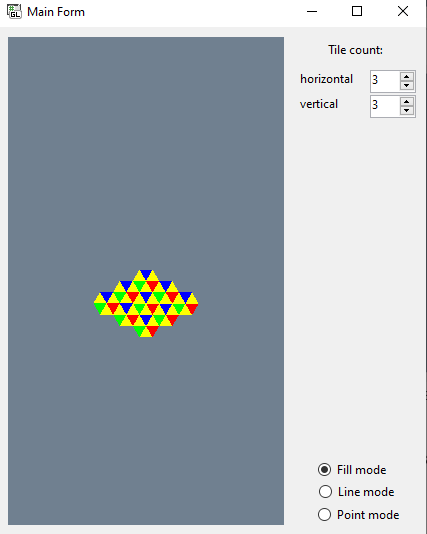


Рисунок 2.3 – Коректне відображення завдання під час зміни розмірів вікна

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

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

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

**Практична робота ?.  
Назва роботи**

**Завдання, варіант № ??**

**Теоретичні відомості**

**Результати виконання практичної роботи**

***Розв'язання завдання***

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

# Загальний перелік посилань

1. Microsoft. glDrawArrays function [Електронний ресурс] / Microsoft – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/windows/win32/opengl/gldrawarrays>.
2. Офіційний вебсайт OpenGL: https://www.opengl.org
3. Документація OpenGL https://www.khronos.org/registry/OpenGL/
4. "OpenGL Programming Guide" (Red Book) — класичний підручник з OpenGL.
5. "OpenGL Shading Language" (Orange Book) — підручник, присвячений шейдерам та програмуванню графіки на рівні шейдерів.

# Додаток А. Лістинг програми до практичної роботи №1

### Код файлу (MainForm.cs)

using System.Windows.Forms;

using static Task\_1.OpenGL;

namespace Task\_1

{

public partial class MainForm : Form

{

public MainForm()

{

InitializeComponent();

}

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

{

}

}

}

### Код файлу (Program.cs)

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Task\_1

{

internal static class Program

{

static Program() => DesignMode = true;

public static bool DesignMode { get; set; }

/// <summary>

/// The main entry point for the application.

/// </summary>

[STAThread]

static void Main()

{

DesignMode = false;

Application.SetHighDpiMode(HighDpiMode.SystemAware);

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new MainForm());

}

}

}

### Код файлу (RenderControl.cs)

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Diagnostics;

using System.Drawing;

using System.Drawing.Design;

using System.Linq;

using System.Runtime.InteropServices;

using static Task\_1.OpenGL;

namespace Task\_1

{

public partial class RenderControl : OpenGL

{

public RenderControl()

{

InitializeComponent();

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

}

private void OnRender(object sender, EventArgs e)

{

float x1 = -4f;

float x2 = 1.5f;

float y1 = -1f;

float y2 = 2f;

// Очищення екрану

glClear(GL\_COLOR\_BUFFER\_BIT);

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

// Встановлення системи координат

glLoadIdentity();

glViewport(0, 0, Width, Height);

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

// Малювання елементів

DrawAxes();

DrawGrid();

DrawFigure();

DrawPoints();

}

private void DrawFigure()

{

glLineWidth(5); // Товщина ліній

glBegin(GL\_LINE\_LOOP);

glColor(Color.LimeGreen); // Синій контур

// Вершини фігури

/\*

glVertex2d(-3.5, 0);

glVertex2d(-3.5, 1);

glVertex2d(-2.5, 1.5);

glVertex2d(-2, 0.5);

glVertex2d(-2, -0.5);

\*/

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

new double[] { -3.5, 0 },

new double[] { -3.5, 1 },

new double[] { -2.5, 1.5 },

new double[] { -2, 0.5 },

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

};

foreach (var vertex in points) {

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

}

glEnd(); // Завершуємо малювання

}

private void DrawPoints()

{

glPointSize(8); // Розмір точок

glBegin(GL\_POINTS);

glColor(Color.GreenYellow);

// Координати точок

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

{

new double[] {-3.5, 0},

new double[] {-3.5, 1 },

new double[] { -2.5, 1.5 },

new double[] {-2, 0.5 },

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

};

// Малюємо точки

foreach (var point in points)

{

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

}

glEnd();

// Підписуємо координати

foreach (var point in points)

{

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

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

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

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

}

}

private void DrawAxes()

{

float x1 = -3.5f;

float x2 = 1f;

float y1 = -0.5f;

float y2 = 1.5f;

glLineWidth(3);

glBegin(GL\_LINES);

glColor(Color.WhiteSmoke);

// Вісь X

glVertex2d(x1, 0.0);

glVertex2d(x2, 0.0);

// Вісь Y

glVertex2d(0.0, y1);

glVertex2d(0.0, y2);

glEnd();

}

private void DrawGrid()

{

float x1 = -3.5f;

float x2 = 1f;

float y1 = -0.5f;

float y2 = 1.5f;

glLineWidth(1);

glEnable(GL\_LINE\_STIPPLE);

glLineStipple(1, 0xAAAA);

glBegin(GL\_LINES);

glColor(Color.White);

// Горизонтальні лінії

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

{

glVertex2d(x1, y);

glVertex2d(x2, y);

}

// Вертикальні лінії

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

{

glVertex2d(x, y1);

glVertex2d(x, y2);

}

glEnd();

glDisable(GL\_LINE\_STIPPLE);

}

}

}

# Додаток Б. Лістинг програми до практичної роботи №1

### Код файлу (MainForm.cs)

using System;

using System.Windows.Forms;

namespace Task\_2

{

public partial class MainForm : Form

{

public MainForm()

{

InitializeComponent();

}

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

{

fillButton.Checked = true;

}

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

{

if (fillButton.Checked)

{

// Changed variable names for better clarity

renderControl1.FillMode = true;

renderControl1.LineMode = false;

renderControl1.PointMode = false;

renderControl1.Invalidate();

}

}

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

{

if (lineButton.Checked)

{

renderControl1.FillMode = false;

renderControl1.LineMode = true;

renderControl1.PointMode = false;

renderControl1.Invalidate();

}

}

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

{

if (pointButton.Checked)

{

renderControl1.FillMode = false;

renderControl1.LineMode = false;

renderControl1.PointMode = true;

renderControl1.Invalidate();

}

}

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

{

// Improved variable names for readability

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

renderControl1.Invalidate();

}

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

{

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

renderControl1.Invalidate();

}

}

}

### Код файлу (RenderControl.cs)

using System;

using System.Drawing;

namespace Task\_2

{

public partial class RenderControl : OpenGL

{

// Renamed variables for clarity

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

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

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

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

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

private readonly double hexagonSize = 100;

public RenderControl()

{

InitializeComponent();

}

private void OnRender(object sender, EventArgs e)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glLoadIdentity();

if (Width > Height)

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

else

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

// Adjusted viewport size calculation

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

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

if (FillMode)

DrawPrimitives(GL\_FILL);

if (PointMode)

{

glPointSize(7);

DrawPrimitives(GL\_POINT);

}

if (LineMode)

DrawPrimitives(GL\_LINE);

}

private void DrawPrimitives(uint mode)

{

double hexagonX;

double hexagonY;

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

{

hexagonX = -hexagonSize \* 1.5 \* j;

hexagonY = -hexagonSize \* 0.85 \* j;

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

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

{

glBegin(GL\_TRIANGLES);

// Calculate the vertices for the hexagon

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

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

{

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

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

}

// Define colors in the specified order

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

{

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

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

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

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

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

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

};

// Draw the hexagon using triangles with specified colors

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

{

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

glVertex2d(hexagonX, hexagonY);

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

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

}

glEnd();

// Adjust positions for new hexagons

hexagonX += hexagonSize \* 1.5;

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

}

}

}

}

}

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