МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського

«Харківський авіаційний інститут»

факультет програмної інженерії та бізнесу

кафедра інженерії програмного забезпечення

Практичні роботи

Minor «*Розробник ігрових додатків*»

дисципліна « *Комп’ютерна графіка з OpenGL* »

*назва дисципліни*

Виконав: студент 3 курсу групи № *632п*

освітньої програми

*121 інженерія програмного забезпечення*

(шифр і назва ОП)

*Петренко Є.О.*

(прізвище й ініціали студента)

Прийняв: доцент каф. 603, *к.т.н*

*Лучшев П.О.*

(посада, науковий ступінь, прізвище й ініціали)

Кількість балів:

Харків – 2024

ЗМІСТ

[ПРАКТИЧНА РОБОТА 1. ОСНОВНІ ПРИНЦИПИ РОБОТИ З OPENGL 3](#_Toc180350320)

[Завдання, варіант № 16 3](#_Toc180350321)

[Системна інформація 3](#_Toc180350322)

[Теоретичні відомості 4](#_Toc180350323)

[Результати виконання практичної роботи 6](#_Toc180350324)

[ПРАКТИЧНА РОБОТА 2. ГРАФІЧНІ ПРИМІТИВИ OPENGL 8](#_Toc180350325)

[Завдання, варіант № 16 8](#_Toc180350326)

[Результати виконання практичної роботи 9](#_Toc180350327)

[ДОДАТОК А Лістинг програми до практичної роботи №1 14](#_Toc180350328)

[ДОДАТОК Б Лістинг програми до практичної роботи №2 19](#_Toc180350329)

# ПРАКТИЧНА РОБОТА 1. ОСНОВНІ ПРИНЦИПИ РОБОТИ З OPENGL

# Завдання, варіант № 16

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

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

# Системна інформація

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

Processor AMD Ryzen 5 4600H @ 3.0Ghz

RAM 16.0 GB (15.4 GB usable)

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

Edition Windows 10 Pro Version 22H2

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

# Теоретичні відомості

***Вершинні масиви***

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

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

̶ glVertexPointer(): вказати покажчик на масив вершинних координат

̶ glNormalPointer(): вказати покажчик на звичайний масив

̶ glColorPointer(): вказати вказівник на масив кольорів RGB

̶ glIndexPointer(): вказати покажчик на індексований кольоровий масив

̶ glTexCoordPointer(): вказати покажчик на масив текстурних шнурів

̶ glEdgeFlagPointer(): вказати вказівник на масив прапорців edge

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

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

***Команда glDrawArrays()***

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

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

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

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

...

// activate and specify pointer to vertex array

glEnableClientState(GL\_VERTEX\_ARRAY);

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

// draw a cube

glDrawArrays(GL\_TRIANGLES, 0, 36);

// deactivate vertex arrays after drawing

glDisableClientState(GL\_VERTEX\_ARRAY);

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

***Команда glDrawElements()***

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

# Результати виконання практичної роботи

***Розв’язання завдання***

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

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

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

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

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

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

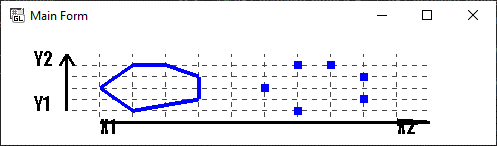


Рисунок 1.1 – Тестування програми при зміні висоти вікна

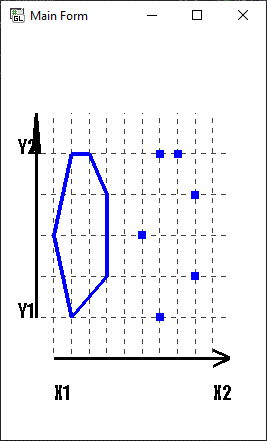


Рисунок 1.2 – Тестування програми при зміні товщини вікна

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

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

Таблиця 1.1 – Результат виконання практичної роботи №1

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

# ПРАКТИЧНА РОБОТА 2. ГРАФІЧНІ ПРИМІТИВИ OPENGL

# Завдання, варіант № 16

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

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

# Результати виконання практичної роботи

***Розв’язання завдання***

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

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

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

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

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

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

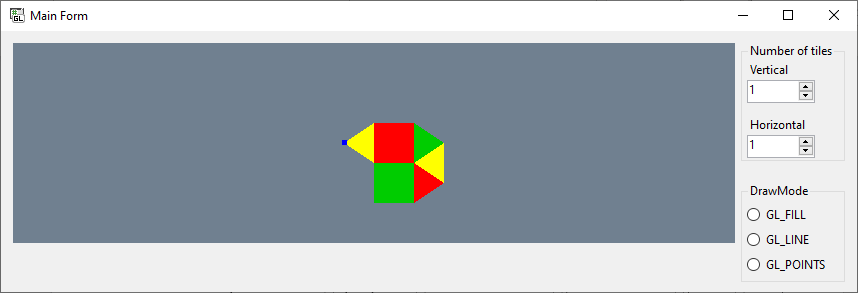


Рисунок 2.1 – Тестування програми при зміні висоти вікна

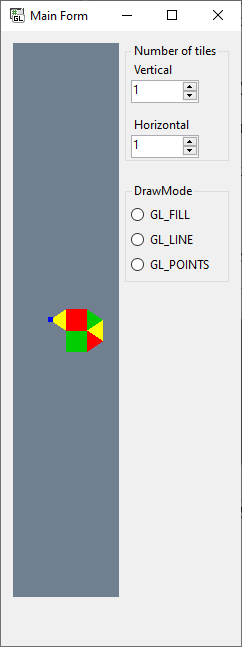


Рисунок 2.2 – Тестування програми при зміні ширини вікна

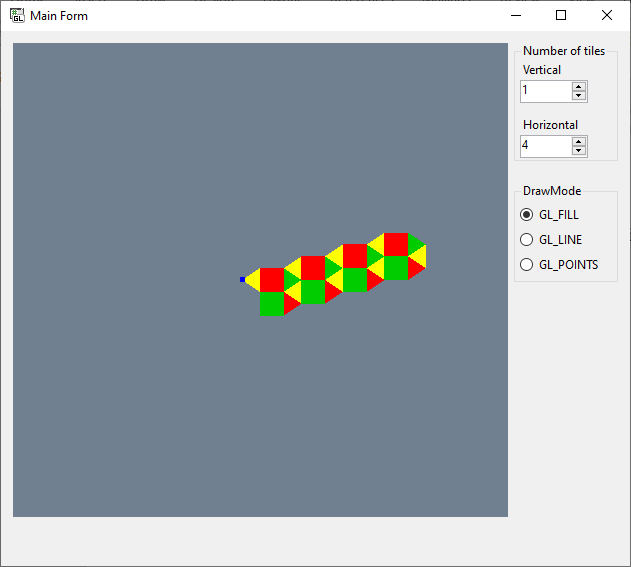


Рисунок 2.3 – Горизонтальне замощення

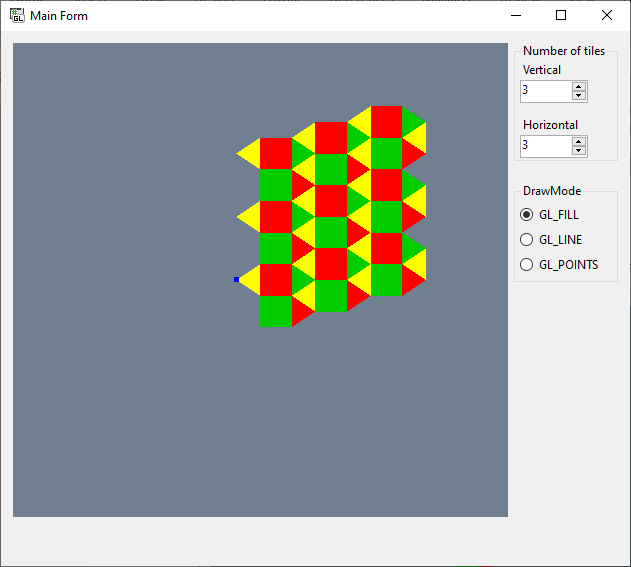


Рисунок 2.4 – Вертикальне та горизонтальне замощення

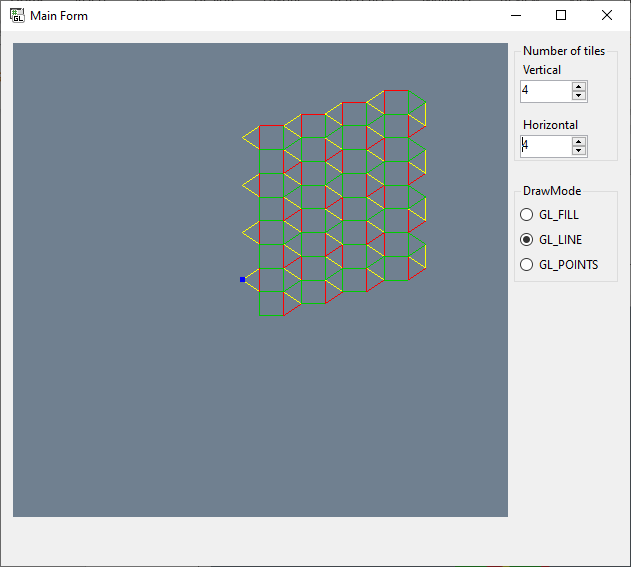


Рисунок 2.5 – Замощення в режимі GL\_LINE

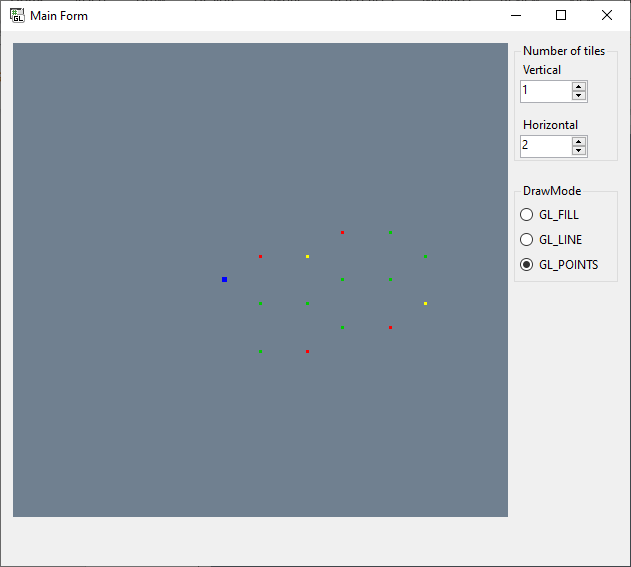


Рисунок 2.6 – Замощення в режимі GL\_POINT

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

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

Таблиця 2.1 – Результат виконання практичної роботи №2

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

# ДОДАТОК А Лістинг програми до практичної роботи №1

**Код файлу RenderControl.cs:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Diagnostics;

using System.Drawing;

using System.Drawing.Design;

using System.Linq;

namespace PR1\_Petrenko\_program

{

public partial class RenderControl : OpenGL // Варіант 16

{

private Layout \_layout;

private Figure \_figure;

readonly int X1 = -3;

readonly int Y1 = 0;

readonly int X2 = 6;

readonly int Y2 = 4;

readonly int padding = 3;

public RenderControl()

{

InitializeComponent();

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

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

}

private void OnRender(object sender, EventArgs e)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glLoadIdentity();

glViewport(0, 0, Width, Height);

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

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

\_layout.LoadLayout();

\_figure.DrawComplex();

glColor3d(0, 0, 0);

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

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

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

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

glEnd();

}

}

}

**Код файлу Layout.cs:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Numerics;

using System.Security.Cryptography.X509Certificates;

using System.Text;

using System.Threading.Tasks;

using static PR1\_Petrenko\_program.OpenGL;

namespace PR1\_Petrenko\_program

{

public partial class RenderControl

{

internal class Layout

{

private readonly int \_x1;

private readonly int \_y1;

private readonly int \_x2;

private readonly int \_y2;

private readonly float \_step;

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

{

\_x1 = x1;

\_y1 = y1;

\_x2 = x2;

\_y2 = y2;

\_step = step;

}

public void DrawGrid()

{

glColor3d(0.3, 0.3, 0.3);

glLineWidth(1);

glEnable(GL\_LINE\_STIPPLE);

glLineStipple(5, 0xAAAA);

glBegin(GL\_LINES);

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

{

glVertex2f(x, \_y1-1);

glVertex2f(x, \_y2+1);

}

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

{

glVertex2f(\_x1-1, y);

glVertex2f(\_x2+1, y);

}

glEnd();

glDisable(GL\_LINE\_STIPPLE);

}

public void DrawAxis()

{

glLineWidth(3);

glColor3d(0, 0, 0);

glBegin(GL\_LINES);

//OX

glVertex2d(\_x1, \_y1-1);

glVertex2d(\_x2, \_y1-1);

//OY

glVertex2d(\_x1-1, \_y1);

glVertex2d(\_x1-1, \_y2);

DrawArrowY();

DrawArrowX();

glEnd();

}

public void DrawArrowY()

{

glLineWidth(3);

glColor3d(0, 0, 0);

glBegin(GL\_LINES);

// Продовження на 1 клітинку

glVertex2d(\_x1-1, \_y2);

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

// Ліва половина

glVertex2d(\_x1-1.2, \_y2);

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

// Права половина

glVertex2d(\_x1-0.8, \_y2);

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

glEnd();

}

public void DrawArrowX()

{

glLineWidth(3);

glColor3d(0, 0, 0);

glBegin(GL\_LINES);

// Продовження на 1 клітинку

glVertex2d(\_x2, \_y1-1);

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

// Ліва половина

glVertex2d(\_x2, \_y1-0.8);

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

// Права половина

glVertex2d(\_x2, \_y1-1.2);

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

glEnd();

}

public void LoadLayout()

{

DrawGrid();

DrawAxis();

}

}

}

}

**Код файлу Figure.cs:**

using System;

using System.Buffers;

using System.Collections.Generic;

using System.Diagnostics;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace PR1\_Petrenko\_program

{

public partial class RenderControl

{

internal class Figure

{

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

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

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

private readonly int[] \_coordinatesX;

private readonly int[] \_coordinatesY;

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

{

\_coordinatesX = coordinatesX;

\_coordinatesY = coordinatesY;

}

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

//{

// glLineWidth(4);

// glPointSize(8);

// glColor3d(0, 0, 1);

// glBegin(type);

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

// {

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

// }

// glEnd();

// glDisable(type);

//}

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

{

glLineWidth(4);

glPointSize(8);

glColor3d(0, 0, 1);

glBegin(type);

if (type == GL\_LINES)

{

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

{

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

// % щоб не вийти за межі масива

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

}

}

else

{

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

{

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

}

}

glEnd();

glDisable(type);

}

public void DrawComplex()

{

Draw(GL\_LINES);

Draw(GL\_POINTS, 5);

}

}

}

}

# ДОДАТОК Б Лістинг програми до практичної роботи №2

**Код файлу RenderControl.cs:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Diagnostics;

using System.Drawing;

using System.Drawing.Design;

using System.Linq;

using static PR2\_Petrenko\_program.OpenGL;

namespace PR2\_Petrenko\_program

{

public partial class RenderControl : OpenGL

{

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

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

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

Figure \_f = new Figure();

public RenderControl()

{

InitializeComponent();

}

private void OnRender(object sender, EventArgs e)

{

int ortho = 10;

glClear(GL\_COLOR\_BUFFER\_BIT);

glLoadIdentity();

if (Width > Height)

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

else

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

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

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

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

{

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

{

double offsetX = col \* 5 \* 1.4;

double offsetY = row \* 5 \* 1.6;

if (col > 0)

offsetY += col \* 2;

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

}

}

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

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

\_f.DrawCursor();

glDisable(GL\_LINES);

}

}

}

**Код файлу Figure.cs:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace PR2\_Petrenko\_program

{

public partial class RenderControl

{

public class Figure

{

readonly double cursorCenterX = -3;

readonly double cursorCenterY = 0;

public void DrawCursor()

{

glPointSize(5);

glColor3d(0, 0, 1);

glBegin(GL\_POINTS);

glVertex2d(cursorCenterX, cursorCenterY);

glEnd();

glDisable(GL\_POINTS);

}

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

{

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

glPointSize(3);

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

glShadeModel(GL\_FLAT);

// YELLOW TRIANGLE

glBegin(GL\_TRIANGLE\_FAN);

glColor3d(1, 1, 0);

glVertex2d(cursorCenterX + offsetX, cursorCenterY + offsetY);

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

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

glEnd();

// RED SQUARE

glBegin(GL\_POLYGON);

glColor3d(1, 0, 0);

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

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

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

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

glEnd();

// GREEN SQUARE

glBegin(GL\_POLYGON);

glColor3d(0, 0.8, 0);

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

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

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

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

glEnd();

// GREEN, YELLOW, RED TRIANGLES

glBegin(GL\_TRIANGLE\_FAN);

glColor3d(1, 1, 0);

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

glColor3d(1, 0, 0);

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

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

glColor3d(1, 1, 0);

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

glColor3d(0, 0.8, 0);

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

glEnd();

}

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

{

DrawComplexFigure(5, DrawMode, offsetX, offsetY);

}

}

}

}

**Код файлу MainForm.cs:**

using System;

using System.Drawing;

using System.Windows.Forms;

using static PR2\_Petrenko\_program.OpenGL;

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

namespace PR2\_Petrenko\_program

{

public partial class MainForm : Form

{

public MainForm()

{

InitializeComponent();

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

renderControl1.DrawMode = GL\_FILL;

}

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

{

renderControl1.DrawMode = GL\_FILL;

renderControl1.Invalidate();

}

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

{

renderControl1.DrawMode = GL\_LINE;

renderControl1.Invalidate();

}

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

{

renderControl1.DrawMode = GL\_POINT;

renderControl1.Invalidate();

}

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

{

if (nUDVertical.Value <= 0)

{

nUDVertical.Value = 1;

}

else if (nUDVertical.Value >= 1)

{

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

renderControl1.Invalidate();

}

}

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

{

if (nUDHorizontal.Value <= 0)

{

nUDHorizontal.Value = 1;

}

else if (nUDHorizontal.Value >= 1)

{

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

renderControl1.Invalidate();

}

}

}

}