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 535б

напряму підготовки (спеціальності):

*121 інженерія програмного забезпечення*

(шифр і назва напряму підготовки / спеціальності)

*Ніколаєв Р.Т.*

(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 2](#_Toc284921075)

[Завдання, варіант № 16 3](#_Toc910167559)

[Системна інформація 3](#_Toc1800046577)

[Теоретичні відомості 3](#_Toc1081951544)

[Результати виконання практичної роботи 4](#_Toc607465962)

[Практична робота 2.ГРАФІЧНІ ПРИМІТИВИ OPENGL 7](#_Toc136520479)

[Завдання, варіант №16 8](#_Toc1348685595)

[Теоретичні відомості 8](#_Toc377581063)

[Результати виконання практичної роботи 9](#_Toc1160935242)

[Практична робота 3.ГРАФІК ФУНКЦІЇ ОДНІЄЇ ЗМІННОЇ 15](#_Toc1046673486)

[Завдання, варіант № 16 16](#_Toc886765777)

[Теоретичні відомості 16](#_Toc951084957)

[Результати виконання практичної роботи 17](#_Toc1354124403)

[Загальний перелік посилань 20](#_Toc1961651297)

[Додаток А.Лістинг програми до практичної роботи №1 21](#_Toc1113004694)

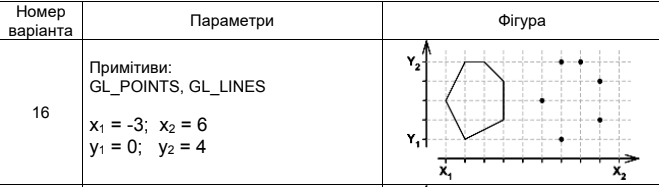
[Додаток Б.Лістинг програми до практичної роботи №2 27](#_Toc2134709750)

[Додаток С.Лістинг програми до практичної роботи №3 31](#_Toc1541866791)

# 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. 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.



## Системна інформація

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

Processor 11th Gen Intel(R) Core(TM) i7-11370H @ 3.30GHz 3.30 GHz

RAM 16,0 ГБ (доступно для використання: 15,7 ГБ)

System type 64-розрядна операційна система, процесор на базі архітектури x64

Edition Windows 11 Home Version

IDE Microsoft Visual Studio Enterprise 2022 (64-bit) version 17.12.3

## Теоретичні відомості

Використання вершинних масивів зменшує кількість викликів функцій та надмірне використання спільних вершин. Це дозволяє підвищити продуктивність рендерингу для завдань, таких як побудова фігури згідно з варіантом 16.

Для побудови фігури в вашому варіанті використовуються примітиви: GL\_POINTS та GL\_LINES.

Фігура побудована на основі масиву координат вершин, які визначаються у коді за допомогою класу Figura. Для відображення осей, сітки та фігури використовуються класи Layout та Figura.

Функції OpenGL для роботи з масивами

OpenGL надає функції, які дозволяють оптимізувати роботу з масивами даних:

glEnableClientState() та glDisableClientState() — для активації та деактивації масивів.

glVertexPointer() — вказує на масив координат вершин.

glDrawArrays() — дозволяє малювати примітиви за масивом вершин.

У вашому коді використовується масив координат вершин для побудови фігури та функція glDrawArrays(). Це зменшує кількість викликів функцій glVertex\*().

Приклад використання glDrawArrays()

Функція glDrawArrays() приймає три аргументи:

Тип примітива (у вашому випадку це GL\_LINES або GL\_POINTS).

Початок читання масиву (індекс у масиві).

Кількість вершин, які потрібно передати до конвеєра рендерингу.

Використання glDrawArrays() дозволяє оптимізувати побудову фігури за допомогою масиву вершин. У вашому варіанті ця функція застосовується для побудови як фігури (лінії), так і окремих точок.

Оптимізація з glDrawElements()

Хоча glDrawArrays() є ефективним методом, він вимагає повторення координат спільних вершин. Для зменшення кількості даних, які передаються, можна використовувати glDrawElements(). Ця функція дозволяє передавати унікальні вершини та використовувати індекси для посилань на них. Це може зменшити обсяг даних у масиві, але у вашому завданні це необов'язково.

## Результати виконання практичної роботи

### Розв'язання завдання

Результати виконання практичної роботи

Розв'язання завдання

Для управління параметрами графічних примітивів у варіанті 16 було використано наступні команди (Додаток А):

Колір: glColor3d(), рядки 68–69 у файлі Figura.cs.

Тип примітивів: glBegin(), glEnd(), glEnable()/glDisable() — реалізовано у файлі Figura.cs для GL\_POINTS та GL\_LINES.

Товщина ліній та розмір точок: glLineWidth() та glPointSize(), рядки 66 та 67 у файлі Figura.cs.

Коректність відображення

Коректне відображення завдання під час змінення розмірів/положення вікна продемонстровано на рисунках 1.1 та 1.2 (див. Додаток Б).

Оптимізація коду

Розроблення підпрограм для виключення дублювання коду наведено у рядках 78–92 файлу Figura.cs.

Застосування циклів для створення зображень реалізовано у рядках 84–92 файлу Layout.cs.

Використання ООП

Об’єктно-орієнтований підхід реалізовано шляхом створення власних класів:

Layout (для побудови осей координат та сітки);

Figura (для малювання фігури).

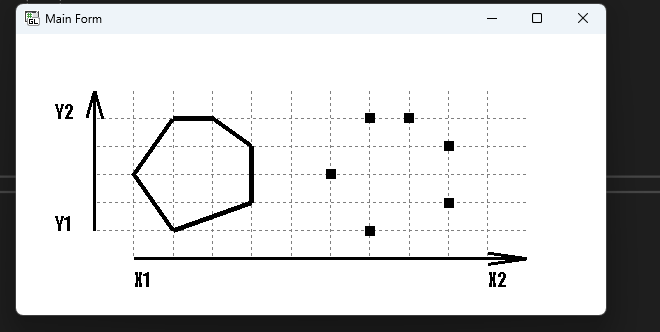


Рисунок 1.1 – Тестування програми при зміні ширини вікна

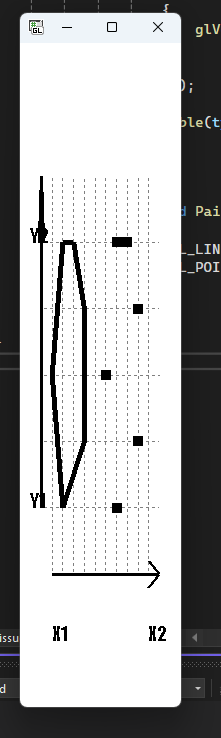


Рисунок 1.2 – Тестування програми при зміні висоти вікна

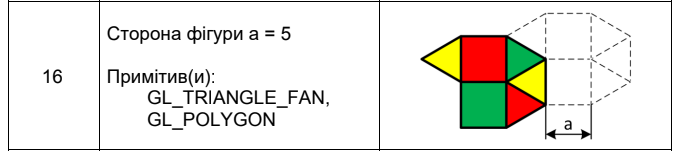
### Контроль виконання вимог та елементів завдання

В результаті виконання практичної роботи були повністю виконані елементи базового рівня та 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. ГРАФІЧНІ ПРИМІТИВИ OPENGL

## Завдання, варіант №16

Використовуючи інструментальні засоби, що вказані викладачем, іберучи до уваги вимого, що наведено в табл. 2.1, створити програмнийпроєкт з підтримкою OpenGL. За допомогою команд glOrtho / gluOrtho2D іglViewport встановити для робочої області ізотропну систему координат зурахуванням розміру фігури, яку задано у варіанті (табл. 2.2). Після стартузастосунок повинен відображати у робочій області одну плитку (tile).Приклад початкового стану застосунку показано на рис. 2.1.  


## Теоретичні відомості

Теоретичні відомості

У ході виконання завдання для варіанту 16 були самостійно вивчені наступні засоби, методи, алгоритми та елементи комп'ютерної графіки на основі OpenGL:

Примітиви OpenGL:

GL\_TRIANGLE\_FAN — використовується для малювання фігури зі спільною центральною точкою та вершинами, які утворюють трикутники.

GL\_POLYGON — застосовується для створення багатокутників, що складаються з вершин, заданих у певному порядку.

Джерело: Офіційна документація OpenGL

Робота з кольорами:

glColor3d() — задає колір для вершин. Для кожного сегмента фігури використовуються різні кольори (жовтий, червоний, зелений), що дозволяє виділити частини фігури.

Джерело: Опис функцій OpenGL

Моделі затінення:

glShadeModel(GL\_FLAT) — дозволяє використовувати плоске затінення, де колір визначається тільки однією вершиною для кожного примітиву.

Джерело: Теорія затінення OpenGL

Режими полігонів:

glPolygonMode(GL\_FRONT\_AND\_BACK, DrawMode) — задає, як відображаються полігони: у режимі точок (GL\_POINTS), ліній (GL\_LINE) або заливки (GL\_FILL). У завданні використовується вибір між цими режимами через елементи керування.

Джерело: OpenGL glPolygonMode

Використання масивів вершин:

Для ефективного малювання фігур застосовуються масиви координат вершин із функціями glBegin() та glVertex2d().

Джерело: OpenGL Vertex Specification

Масштабування та розташування фігур:

Функції offsetX та offsetY дозволяють змінювати положення фігур у просторі, створюючи різні елементи конструкції.

Управління інтерфейсом:

Реалізація вибору режиму малювання (GL\_FILL, GL\_LINE, GL\_POINTS) через радіо-кнопки (RadioButton) у формі Windows Forms.

Зміна кількості елементів фігури через компоненти NumericUpDown.

Джерело: Windows Forms Documentation

Ці знання та інструменти дозволили створити фігуру відповідно до заданого варіанту та забезпечити її динамічну зміну в процесі роботи програми.

## Результати виконання практичної роботи

### Розв'язання завдання

Перерахування вимог базового рівня складності

Відображення однієї плитки (рис. 2.1):

Реалізовано за допомогою методу PaintComplexFigure() у файлі Figura.cs, рядки 16–41.

Скріншот виконання див. у Додатку Б, рис. 2.1.

Багаторазове замощення плиткою (рис. 2.2):

Реалізовано через метод PaintTile() у файлі Figura.cs, рядки 43–45.

Задавання кількості плиток здійснюється за допомогою елементів NumericUpDown (nUDVertical та nUDHorizontal) у файлі MainForm.cs, рядки 79–88.

Скріншот виконання див. у Додатку Б, рис. 2.2.

Коректне відображення під час змін розмірів/положення вікна:

Реалізовано у файлі RenderControl, який забезпечує коректне перерахування координат для малювання в нових розмірах вікна.

Скріншоти результатів див. у Додатку Б, рис. 2.3.

Взаємодія з користувачем через стандартні засоби:

Використання миші для вибору режиму малювання (GL\_FILL, GL\_LINE, GL\_POINTS) через елементи RadioButton (файл MainForm.cs, рядки 55–77).

Реалізація відслідковування подій зміни значень у NumericUpDown (файл MainForm.cs, рядки 79–88).

Скріншот демонстрації функціоналу див. у Додатку Б, рис. 2.4.

Застосування мінімальної кількості графічних примітивів:

У файлі Figura.cs використовуються тільки необхідні примітиви: GL\_TRIANGLE\_FAN та GL\_POLYGON (рядки 16–45).

Результат роботи див. у Додатку Б

Перерахування вимог підвищеного рівня складності

Створення власних елементів інтерфейсу за допомогою OpenGL:

Реалізовано осі координат і сітку у класі RenderControl (файл RenderControl.cs, методи DrawAxis() та PaintGrid(), рядки 15–58).

Скріншоти результатів див. у Додатку Б

Використання ООП:

Реалізовано у файлах Figura.cs та RenderControl.cs за допомогою класів:

Figura — відповідає за малювання плиток та фігури;

RenderControl — відповідає за відображення осей та сітки.

Деталі реалізації наведено у рядках коду 10–60 у файлі Figura.cs та 15–58 у файлі RenderControl.cs.

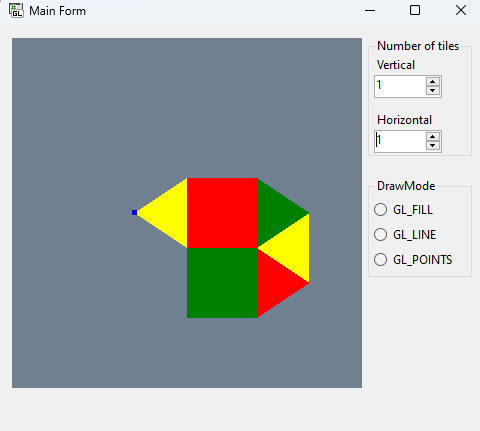
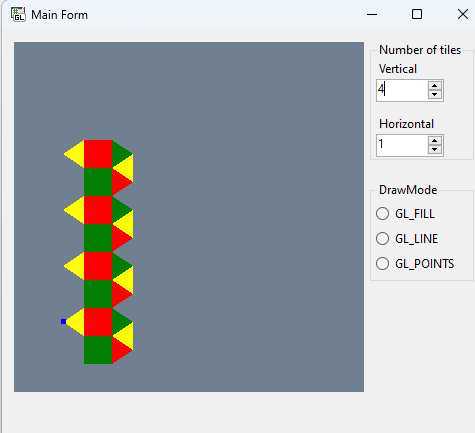


Рисунок-2.1



А

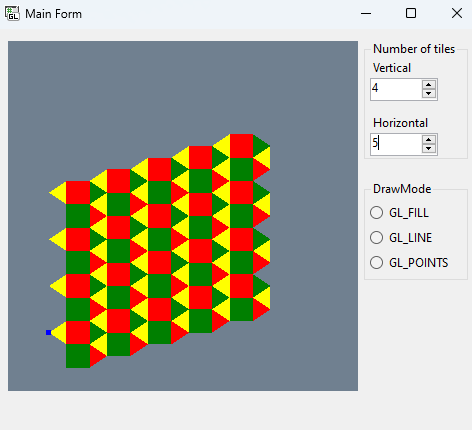


Рисунок 2.2 -а тільки по вертикалі б- гор+вертикаль

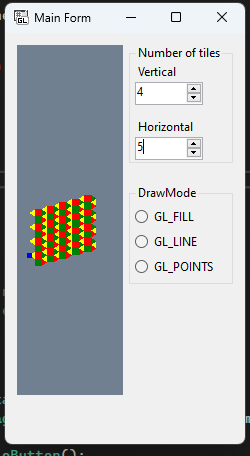


Рисунок 2.3 Коректне відображення під час змін розмірів/положення вікна

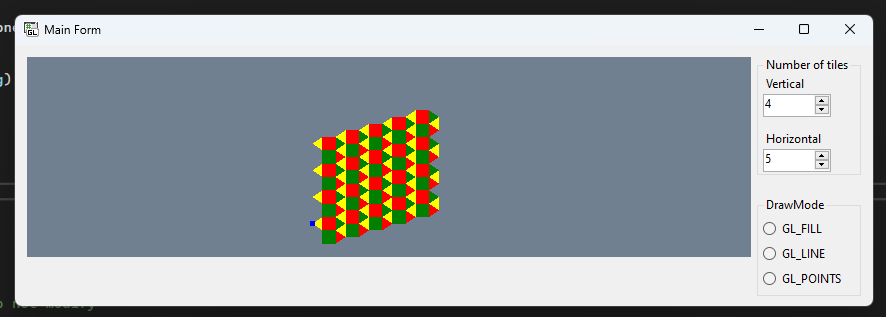


Рисунок 2.3.2 Коректне відображення під час змін розмірів/положення вікна

Рисунок 2.4.1

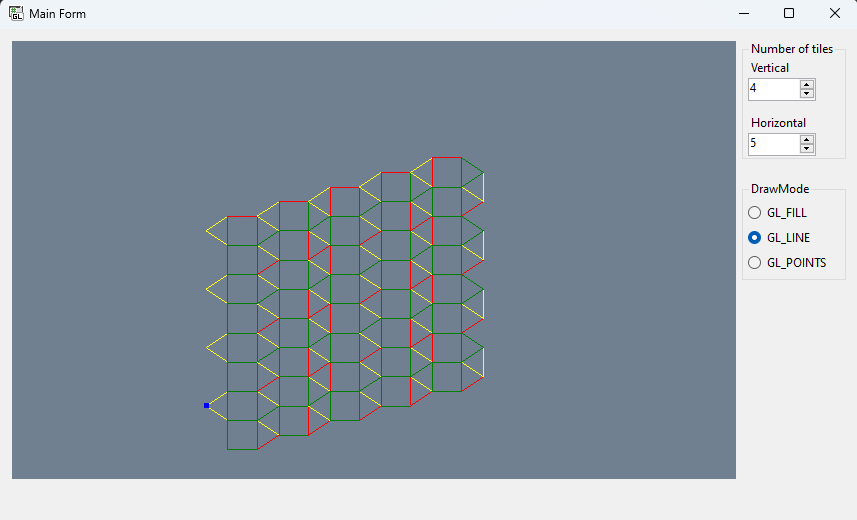
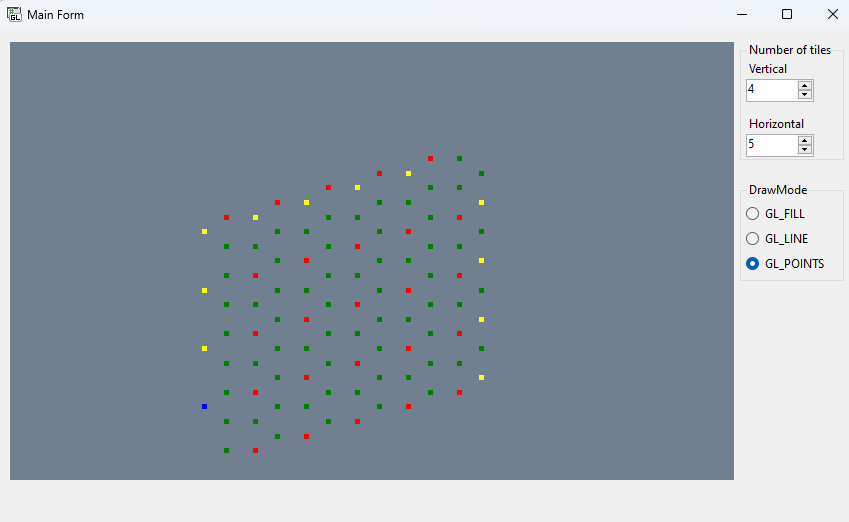


Рисунок 2.4.1

Рисунок 2.4.2

### 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.

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Складність | Вимоги | Бали |
| 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+ |

Таблиця 2.1

# Загальний перелік посилань

**Загальний перелік посилань**

1 Репозіторій GitHub <https://github.com/RostyslavNikolaiev/Task01/tree/main>

2 Офіційна документація OpenGL:

3 Методичні матеріали з основ OpenGL:

# Додаток А. Лістинг програми до практичної роботи №1

### Код файлу (MainForm.cs)

1. namespace Nikolaiev\_task01

* {
* partial class MainForm
* {
* private System.ComponentModel.IContainer components = null;
* /// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>
* protected override void Dispose(bool disposing)
* {
* if (disposing && (components != null))
* {
* components.Dispose();
* }
* base.Dispose(disposing);
* }
* #region Windows Form Designer generated code
* /// <summary>
* /// Required method for Designer support - do not modify
* /// the contents of this method with the code editor.
* /// </summary>
* private void InitializeComponent()
* {
* System.ComponentModel.ComponentResourceManager resources = new System.ComponentModel.ComponentResourceManager(typeof(MainForm));
* renderControl1 = new RenderControl();
* SuspendLayout();
* //
* // renderControl1
* //
* renderControl1.BackColor = System.Drawing.Color.Transparent;
* renderControl1.Dock = System.Windows.Forms.DockStyle.Fill;
* renderControl1.Font = new System.Drawing.Font("Impact", 14.25F, System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point);
* renderControl1.ForeColor = System.Drawing.Color.White;
* renderControl1.Location = new System.Drawing.Point(0, 0);
* renderControl1.Name = "renderControl1";
* renderControl1.Size = new System.Drawing.Size(543, 281);
* renderControl1.TabIndex = 0;
* renderControl1.TextCodePage = 65001;
* //
* // MainForm
* //
* AutoScaleDimensions = new System.Drawing.SizeF(10F, 19F);
* AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
* ClientSize = new System.Drawing.Size(543, 281);
* Controls.Add(renderControl1);
* Font = new System.Drawing.Font("Play", 12F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point);
* Icon = (System.Drawing.Icon)resources.GetObject("$this.Icon");
* Margin = new System.Windows.Forms.Padding(4);
* Name = "MainForm";
* Text = "Main Form";
* ResumeLayout(false);
* }
* #endregion
* private RenderControl renderControl1;
* }
* }

### Код файлу (Figura.cs)

1. 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 Nikolaiev\_task01
* {
* public partial class RenderControl
* {
* internal class Figura
* {
* private readonly int[] \_cordX;
* private readonly int[] \_\_cordY;
* public Figura(int[] coordinatesX, int[] coordinatesY)
* {
* \_cordX = coordinatesX;
* \_\_cordY = coordinatesY;
* }
* public void Paint(uint type, int difference = 0, bool complex = false)
* {
* glLineWidth(5);
* glPointSize(10);
* glColor3d(0, 0, 0);
* glBegin(type);
* if (type == GL\_LINES)
* {
* for (int i = 0; i < \_cordX.Length; i++)
* {
* glVertex2d(\_cordX[i]+difference, \_\_cordY[i]);
* glVertex2d(\_cordX[(i + 1) % \_cordX.Length]+difference, \_\_cordY[(i + 1) % \_\_cordY.Length]);
* }
* }
* else
* {
* for (int i = 0; i < \_cordX.Length; i++)
* {
* glVertex2d(\_cordX[i]+difference, \_\_cordY[i]);
* }
* }
* glEnd();
* glDisable(type);
* }

* public void Paintall()
* {
* Paint(GL\_LINES);
* Paint(GL\_POINTS, 5);
* }
* }
* }
* }

**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 Nikolaiev\_task01.OpenGL;

namespace Nikolaiev\_task01

{

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 DrawAxis()

{

glLineWidth(3);

glColor3d(0, 0, 0);

glBegin(GL\_LINES);

glVertex2d(\_x1, \_y1-1);

glVertex2d(\_x2, \_y1-1);

glVertex2d(\_x1-1, \_y1);

glVertex2d(\_x1-1, \_y2);

PaintArrowY();

PaintArrowX();

glEnd();

}

public void PaintArrowY()

{

glLineWidth(3);

glColor3d(0, 0, 0);

glBegin(GL\_LINES);

glVertex2d(\_x1-1, \_y2);

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

glVertex2d(\_x1-1.2, \_y2);

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

glVertex2d(\_x1-0.8, \_y2);

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

glEnd();

}

public void PaintArrowX()

{

glLineWidth(3);

glColor3d(0, 0, 0);

glBegin(GL\_LINES);

glVertex2d(\_x2, \_y1-1);

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

glVertex2d(\_x2, \_y1-0.8);

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

glVertex2d(\_x2, \_y1-1.2);

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

glEnd();

}

public void PaintGrid()

{

glColor3d(0.5, 0.5, 0.5);

glLineWidth(1);

glEnable(GL\_LINE\_STIPPLE);

glLineStipple(3, 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 LoadLayout()

{

PaintGrid();

DrawAxis();

}

}

}

}

# Додаток Б. Лістинг програми до практичної роботи №2

### Код файлу (Program.cs)

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Task\_02

{

internal static class Program

{

static Program() => DesignMode = true;

public static bool DesignMode { get; set; }

[STAThread]

static void Main()

{

DesignMode = false;

Application.SetHighDpiMode(HighDpiMode.SystemAware);

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new MainForm());

}

}

}

### Код файлу (Figura.cs)

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Task\_02

{

public partial class RenderControl

{

public class Figura

{

readonly double cursorCenterX = -3;

readonly double cursorCenterY = 0;

public void PaintCursor()

{

glPointSize(5);

glColor3d(0, 0, 1);

glBegin(GL\_POINTS);

glVertex2d(cursorCenterX, cursorCenterY);

glEnd();

glDisable(GL\_POINTS);

}

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

{

glPointSize(5);

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

glShadeModel(GL\_FLAT);

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();

glBegin(GL\_POLYGON);

glColor3d(1, 0, 0);

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

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

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

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

glEnd();

glBegin(GL\_POLYGON);

glColor3d(0, 0.5, 0);

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

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

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

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

glEnd();

glBegin(GL\_TRIANGLE\_FAN);

glColor3d(1, 1, 0);

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

glColor3d(1, 0, 0);

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

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

glColor3d(1, 1, 0);

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

glColor3d(0, 0.5, 0);

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

glEnd();

}

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

{

PaintComplexFigure(5, DrawMode, offsetX, offsetY);

}

}

}

}

### Код файлу (Render.cs)

using System.ComponentModel;

using System.Drawing;

namespace Task\_02

{

[ToolboxItem(true), ToolboxBitmap(typeof(RenderControl), "RenderControl.bmp"), DefaultEvent("")]

partial class RenderControl

{

private System.ComponentModel.IContainer components = null;

/// <param name="disposing">true </param>

protected override void Dispose(bool disposing)

{

if (disposing && (components != null))

{

components.Dispose();

}

base.Dispose(disposing);

}

#region Component Designer generated code

private void InitializeComponent()

{

SuspendLayout();

AutoScaleMode = System.Windows.Forms.AutoScaleMode.None;

BackColor = Color.SlateGray;

ForeColor = Color.White;

Name = "RenderControl";

Size = new Size(480, 300);

Render += OnRender;

ResumeLayout(false);

}

#endregion

}

}