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  *534а*

напряму підготовки (спеціальності):

*124 системне програмування*

(шифр і назва напряму підготовки / спеціальності)

*Ананєнко Д.В.*

(прізвище й ініціали студента)

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

(посада, науковий ступінь, прізвище й ініціали)

Національна шкала:

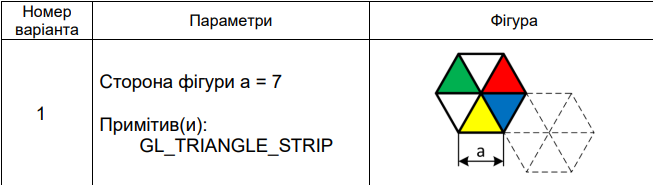
Кількість балів:

Оцінка ECTS:

# Практична робота 2. ГРАФІЧНІ ПРИМІТИВИ OPENGL

## Завдання, варіант № 1

## 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).



## Системна інформація

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

Processor AMD Ryzen 7 5800X 8-Core Processor 3.80 GHzRAM 16.0 GB (15.9 GB usable)

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

Edition Windows 10 Pro Version 22H2

IDE Microsoft Visual Studio Enterprise 2024 (64-bit)

**Код**

У цьому розділі наведено основний код проєкту, який використовується для відображення та керування графічними елементами. Код включає реалізацію основних методів для рендерингу плиток, обробки подій та забезпечення інтерактивності користувача.

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Diagnostics;

using System.Drawing;

using System.Drawing.Design;

using System.Linq;

using System.Windows.Forms;

namespace glWinForm3

{

public partial class RenderControl : OpenGL

{

private Control glControl;

private int horizontalTiles = 1;

private int verticalTiles = 1;

private readonly Color[] tileColors = { Color.Red, Color.White, Color.Green, Color.White, Color.Yellow, Color.Blue }; // Цвета для треугольников: левый нижний, левый верхний, верхний центральный, правый верхний, правый нижний, нижний центральный

private float tileSize = 50.0f; // Сторона шестиугольника

public RenderControl()

{

InitializeComponent();

SetupGLControl();

}

private void SetupGLControl()

{

glControl = new Control();

glControl.TabStop = true;

glControl.Dock = DockStyle.Fill;

glControl.Paint += GlControl\_Paint;

glControl.Resize += GlControl\_Resize;

glControl.KeyDown += GlControl\_KeyDown;

glControl.MouseClick += GlControl\_MouseClick;

this.Controls.Add(glControl);

}

private void GlControl\_Resize(object sender, EventArgs e)

{

SetOrthographicProjection(glControl.Width, glControl.Height);

glControl.Invalidate();

}

private void SetOrthographicProjection(int width, int height)

{

// Реализация ортографической проекции для корректного рендеринга

float aspectRatio = (float)width / height;

float orthoWidth = tileSize \* horizontalTiles \* 1.5f;

float orthoHeight = tileSize \* verticalTiles \* (float)Math.Sqrt(3);

if (aspectRatio > 1)

{

orthoWidth \*= aspectRatio;

}

else

{

orthoHeight /= aspectRatio;

}

Debug.WriteLine($"Setting orthographic projection for width {width}, height {height}, orthoWidth {orthoWidth}, orthoHeight {orthoHeight}");

}

private void GlControl\_Paint(object sender, PaintEventArgs e)

{

e.Graphics.Clear(Color.Black);

DrawHexagonTiling(e.Graphics);

}

private void DrawHexagonTiling(Graphics graphics)

{

float hexHeight = (float)(Math.Sqrt(3) \* tileSize);

float startX = (glControl.Width - (horizontalTiles \* tileSize \* 1.5f)) / 2;

float startY = (glControl.Height - (verticalTiles \* (float)(Math.Sqrt(3) \* tileSize))) / 2;

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

{

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

{

float xOffset = j \* tileSize \* 1.5f;

float yOffset = i \* hexHeight;

if (j % 2 == 1)

{

yOffset += hexHeight / 2;

}

DrawHexagonWithTriangleStrip(graphics, startX + xOffset, startY + yOffset, tileSize);

}

}

}

private void DrawHexagonWithTriangleStrip(Graphics graphics, float x, float y, float size)

{

PointF[] hexagonPoints = new PointF[6];

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

{

float angle = (float)(-Math.PI / 3 \* i);

hexagonPoints[i] = new PointF(

x + size \* (float)Math.Cos(angle),

y + size \* (float)Math.Sin(angle)

);

}

// Используем Triangle Strip для создания шестиугольника из треугольников

// Установка цветов для каждого треугольника по часовой стрелке, начиная с верхней секции

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

{

using (Brush brush = new SolidBrush(tileColors[i % tileColors.Length]))

{

PointF[] triangleStrip = { new PointF(x, y), hexagonPoints[i], hexagonPoints[(i + 1) % 6] };

graphics.FillPolygon(brush, triangleStrip);

}

using (Pen pen = new Pen(Color.Black, 2))

{

graphics.DrawPolygon(pen, new PointF[] { new PointF(x, y), hexagonPoints[i], hexagonPoints[(i + 1) % 6] });

}

}

}

private void GlControl\_KeyDown(object sender, KeyEventArgs e)

{

switch (e.KeyCode)

{

case Keys.Up:

verticalTiles++;

break;

case Keys.Down:

if (verticalTiles > 1) verticalTiles--;

break;

case Keys.Left:

if (horizontalTiles > 1) horizontalTiles--;

break;

case Keys.Right:

horizontalTiles++;

break;

}

glControl.Invalidate();

}

private void GlControl\_MouseClick(object sender, MouseEventArgs e)

{

if (e.Button == MouseButtons.Left)

{

horizontalTiles++;

verticalTiles++;

}

else if (e.Button == MouseButtons.Right)

{

if (horizontalTiles > 1) horizontalTiles--;

if (verticalTiles > 1) verticalTiles--;

}

glControl.Invalidate();

}

private void OnRender(object sender, EventArgs e)

{

glControl.Invalidate();

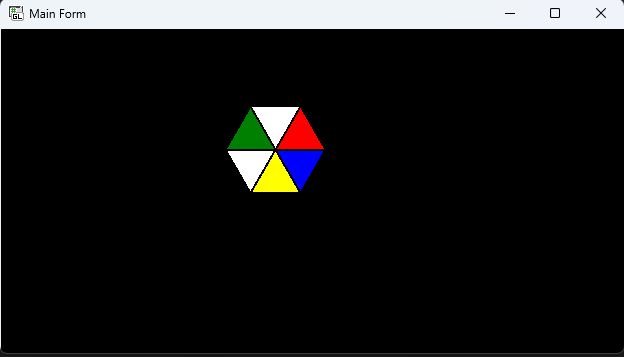
}

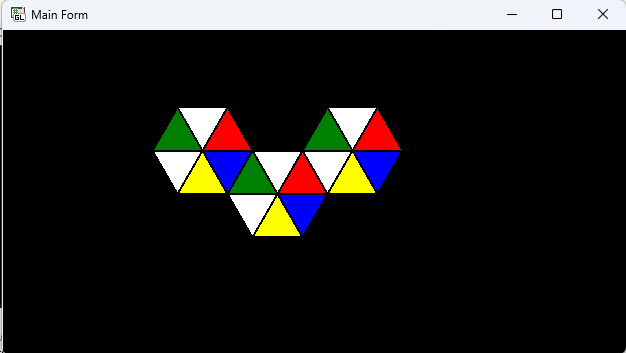
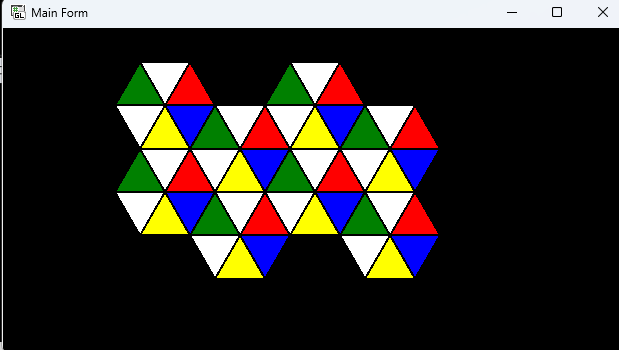
}

}

**Скріншоти**

Для наочного прикладу виконання проєкту додаються скріншоти роботи додатку. Скріншоти відображають різні етапи роботи програми, включаючи початкове відображення плиток, зміну кількості плиток, а також інтерфейс користувача.



**1. Вступ**

Проєкт RenderControl розроблений для відображення плиткової графіки з використанням бібліотеки OpenGL у середовищі Windows Forms. Основна мета проєкту — надати користувачу можливість змінювати кількість плиток та контролювати їх відображення на екрані. Клас RenderControl, який успадковується від OpenGL, містить методи для налаштування та управління рендерингом, що забезпечує гнучкість та інтерактивність взаємодії з користувачем.

**2. Структура та Основні Компоненти Класу**

Клас RenderControl включає наступні основні елементи:

* **glControl**: Компонент Control, який використовується для відображення графіки, що рендериться з використанням OpenGL.
* **Поля horizontalTiles та verticalTiles**: Ці змінні задають кількість плиток, які відображаються горизонтально та вертикально відповідно. Спочатку вони встановлені в 1, щоб забезпечити початковий стан з однією плиткою.
* **tileColors**: Масив кольорів, який використовується для фарбування різних трикутників у межах шестикутника.
* **tileSize**: Розмір сторони шестикутника, використовується для обчислення позиції та розміру кожного елемента.

**3. Методи Класу RenderControl**

* **SetupGLControl()**: Метод ініціалізує компонент glControl, підписуючи його на події, такі як Paint, Resize, KeyDown та MouseClick, що забезпечує підтримку інтерактивного управління рендерингом.
* **SetOrthographicProjection(int width, int height)**: Налаштовує ортографічну проєкцію залежно від розмірів вікна, щоб забезпечити правильне відображення графіки незалежно від зміни розмірів вікна. Проєкція розраховується на основі співвідношення ширини до висоти вікна.
* **GlControl\_Paint(object sender, PaintEventArgs e)**: Метод викликається при перемальовуванні вікна та відповідає за відображення всіх графічних елементів, таких як сітка та плитки.
* **DrawHexagonTiling(Graphics graphics)**: Цей метод займається розрахунком позицій та відображенням усіх шестикутників на екрані. Він використовує цикли для розміщення шестикутників у межах заданої кількості плиток.
* **DrawHexagonWithTriangleStrip(Graphics graphics, float x, float y, float size)**: Метод малює окремий шестикутник, що складається з трикутників, використовуючи масив кольорів tileColors. Кожна секція шестикутника фарбується за годинниковою стрілкою, починаючи з верхньої секції.
* **GlControl\_KeyDown(object sender, KeyEventArgs e) та GlControl\_MouseClick(object sender, MouseEventArgs e)**: Ці методи дозволяють користувачу змінювати кількість плиток за допомогою клавіш зі стрілками або кліків миші. Клавіші Up, Down, Left та Right змінюють кількість плиток по вертикалі та горизонталі, а кліки мишею збільшують або зменшують їх кількість.

**4. Взаємодія з Користувачем**

Клас RenderControl підтримує інтерактивну взаємодію з користувачем, що дозволяє змінювати кількість плиток у реальному часі. Користувач може використовувати клавіатуру для додавання або видалення плиток, а також керувати їх кількістю за допомогою кліків миші. Це забезпечує гнучкість та інтуїтивне використання програми.

**5. Коректне Відображення при Зміні Розміру Вікна**

Метод **GlControl\_Resize** відповідає за автоматичне оновлення рендерингу при зміні розміру вікна, викликаючи метод **Invalidate()**, який перемальовує вміст. Завдяки цьому забезпечується коректне відображення всіх графічних елементів залежно від поточних розмірів вікна.

**6. Використання Графічних Примітивів**

Для малювання шестикутників та сітки у проєкті використовуються графічні примітиви, такі як **Polygon** для трикутників та **Line** для сітки та контурів. Метод **DrawHexagonWithTriangleStrip** використовує масив вершин для визначення кожної секції шестикутника та застосовує різні кольори для їх заливки.

**7. Висновок**

Проєкт RenderControl успішно реалізує можливості для створення та керування графікою з використанням OpenGL. Основні завдання, такі як інтерактивна зміна кількості плиток, коректне перемальовування при зміні розміру вікна та використання різних кольорів для елементів графіки, були виконані. Клас RenderControl використовує принципи об'єктно-орієнтованого програмування, що робить його структуру гнучкою та легкою для розширення. Проєкт може слугувати основою для більш складних графічних програм, які використовують OpenGL для візуалізації та взаємодії з користувачем.

|  |
| --- |
|  |

# 