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  *535а*

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)

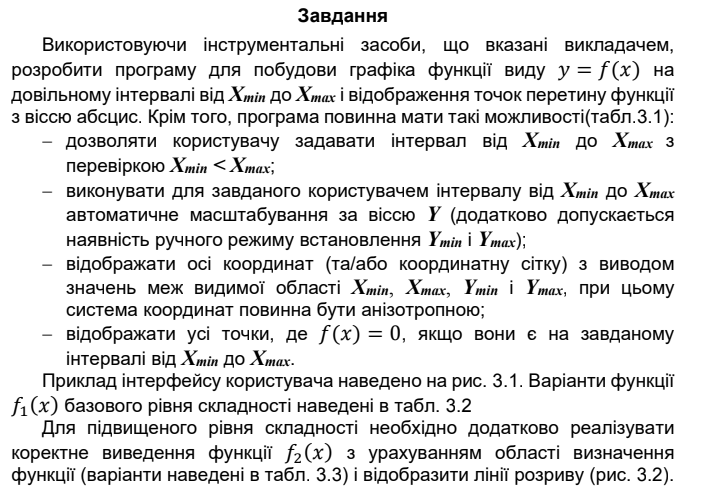
Національна шкала:

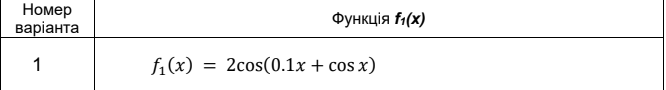
Кількість балів:

Оцінка ECTS:

# Практична робота 3. ГРАФІЧНІ ПРИМІТИВИ OPENGL

## Завдання, варіант № 1





## 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**

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 float Xmin = -5.0f;

private float Xmax = 5.0f;

private float Ymin;

private float Ymax;

private int numberOfPoints = 300;

private Func<float, float> functionToPlot = x => (float)(Math.Pow(Math.Sin(3 \* x) + 1.5, Math.Cos(2 \* x)) - 1);

private Panel controlPanel;

private NumericUpDown numXmin;

private NumericUpDown numXmax;

private NumericUpDown numPoints;

private Label labelXmin;

private Label labelXmax;

private Label labelPoints;

private Button btnShowPoint;

private Button btnReset;

private PointF? highlightedPoint = null;

private bool isBehavioralGraph = true;

public RenderControl()

{

CustomInitializeComponent();

SetupGLControl();

SetupUIControls();

DrawBehavioralGraph();

}

private void CustomInitializeComponent()

{

this.SuspendLayout();

//

// RenderControl

//

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

this.BackColor = Color.SlateGray;

this.ForeColor = Color.White;

this.Name = "RenderControl";

this.Size = new Size(900, 600);

this.ResumeLayout(false);

}

private void SetupGLControl()

{

glControl = new Control();

glControl.TabStop = true;

glControl.Dock = DockStyle.Fill;

glControl.Paint += GlControl\_Paint;

glControl.Resize += GlControl\_Resize;

this.Controls.Add(glControl);

}

private void SetupUIControls()

{

// Setup panel for controls

controlPanel = new Panel()

{

Dock = DockStyle.Right,

Size = new Size(150, this.Height),

BackColor = Color.LightGray

};

// Setup NumericUpDowns and Labels for Xmin, Xmax, and Points

numXmin = new NumericUpDown()

{

Minimum = -100,

Maximum = 100,

DecimalPlaces = 2,

Increment = 0.1M,

Value = (decimal)Xmin,

Location = new Point(10, 20),

Size = new Size(60, 20)

};

numXmin.ValueChanged += (s, e) => { Xmin = (float)numXmin.Value; AdjustGraphToFit(); glControl.Invalidate(); };

numXmax = new NumericUpDown()

{

Minimum = -100,

Maximum = 100,

DecimalPlaces = 2,

Increment = 0.1M,

Value = (decimal)Xmax,

Location = new Point(10, 60),

Size = new Size(60, 20)

};

numXmax.ValueChanged += (s, e) => { Xmax = (float)numXmax.Value; AdjustGraphToFit(); glControl.Invalidate(); };

numPoints = new NumericUpDown()

{

Minimum = 10,

Maximum = 1000,

Value = numberOfPoints,

Location = new Point(10, 100),

Size = new Size(60, 20)

};

numPoints.ValueChanged += (s, e) => { numberOfPoints = (int)numPoints.Value; AdjustGraphToFit(); glControl.Invalidate(); };

labelXmin = new Label()

{

Text = "Xmin",

Location = new Point(80, 20),

Size = new Size(50, 20)

};

labelXmax = new Label()

{

Text = "Xmax",

Location = new Point(80, 60),

Size = new Size(50, 20)

};

labelPoints = new Label()

{

Text = "Points",

Location = new Point(80, 100),

Size = new Size(50, 20)

};

// Setup buttons for showing point and resetting graph

btnShowPoint = new Button()

{

Text = "Show Point",

Location = new Point(10, 140),

Size = new Size(100, 30)

};

btnShowPoint.Click += BtnShowPoint\_Click;

btnReset = new Button()

{

Text = "Reset",

Location = new Point(10, 180),

Size = new Size(100, 30)

};

btnReset.Click += BtnReset\_Click;

// Add controls to the panel

controlPanel.Controls.Add(numXmin);

controlPanel.Controls.Add(numXmax);

controlPanel.Controls.Add(numPoints);

controlPanel.Controls.Add(labelXmin);

controlPanel.Controls.Add(labelXmax);

controlPanel.Controls.Add(labelPoints);

controlPanel.Controls.Add(btnShowPoint);

controlPanel.Controls.Add(btnReset);

// Add panel to the form

this.Controls.Add(controlPanel);

}

private void BtnShowPoint\_Click(object sender, EventArgs e)

{

float x = (Xmin + Xmax) / 2; // User-defined point can be implemented here

highlightedPoint = new PointF(x, functionToPlot(x));

isBehavioralGraph = false;

glControl.Invalidate();

}

private void BtnReset\_Click(object sender, EventArgs e)

{

highlightedPoint = null;

DrawBehavioralGraph();

}

private void DrawBehavioralGraph()

{

Xmin = -5.0f;

Xmax = 5.0f;

numXmin.Value = (decimal)Xmin;

numXmax.Value = (decimal)Xmax;

numberOfPoints = 300;

numPoints.Value = numberOfPoints;

isBehavioralGraph = true;

AdjustGraphToFit();

glControl.Invalidate();

}

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

{

AdjustGraphToFit();

glControl.Invalidate();

}

private void SetOrthographicProjection(int width, int height)

{

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

}

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

{

e.Graphics.Clear(Color.White);

DrawGraph(e.Graphics);

}

private void DrawGraph(Graphics graphics)

{

float h = (Xmax - Xmin) / (numberOfPoints - 1);

List<PointF> points = new List<PointF>();

// Compute Ymin and Ymax for automatic scaling

Ymin = float.MaxValue;

Ymax = float.MinValue;

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

{

float x = Xmin + i \* h;

float y = functionToPlot(x);

points.Add(new PointF(x, y));

if (y < Ymin) Ymin = y;

if (y > Ymax) Ymax = y;

}

// Set scaling factors to fit graph into the control's drawing area

float scaleX = (glControl.Width \* 0.4f) / (Xmax - Xmin); // Reduce scale to make graph smaller initially

float scaleY = (glControl.Height \* 0.4f) / (Ymax - Ymin);

// Ensure the graph is centered within the control

float offsetX = glControl.Width / 2 - ((Xmax + Xmin) / 2) \* scaleX;

float offsetY = glControl.Height / 2 + ((Ymax + Ymin) / 2) \* scaleY;

// Draw coordinate axes

Pen axisPen = new Pen(Color.Black, 2);

graphics.DrawLine(axisPen, 0, glControl.Height / 2, glControl.Width, glControl.Height / 2); // X-axis

graphics.DrawLine(axisPen, glControl.Width / 2, 0, glControl.Width / 2, glControl.Height); // Y-axis

// Draw the function

Pen graphPen = new Pen(Color.Red, 1);

for (int i = 0; i < points.Count - 1; i++)

{

float x1 = offsetX + (points[i].X - Xmin) \* scaleX;

float y1 = offsetY - (points[i].Y - Ymin) \* scaleY;

float x2 = offsetX + (points[i + 1].X - Xmin) \* scaleX;

float y2 = offsetY - (points[i + 1].Y - Ymin) \* scaleY;

graphics.DrawLine(graphPen, x1, y1, x2, y2);

}

// Draw highlighted point if any

if (highlightedPoint.HasValue && !isBehavioralGraph)

{

PointF p = highlightedPoint.Value;

float x = offsetX + (p.X - Xmin) \* scaleX;

float y = offsetY - (p.Y - Ymin) \* scaleY;

graphics.FillEllipse(Brushes.Green, x - 5, y - 5, 10, 10);

}

}

private void OnRender(object sender, EventArgs e)

{

glControl.Invalidate();

}

private void AdjustGraphToFit()

{

// Adjust scaling to ensure the graph always fits within the viewport

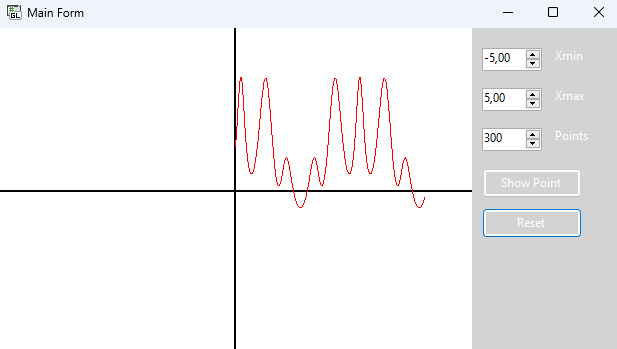
SetOrthographicProjection(glControl.Width, glControl.Height);

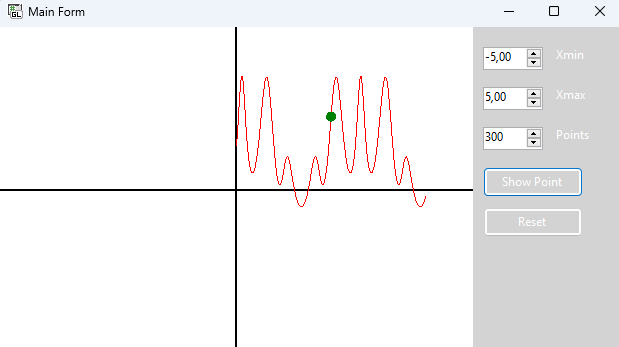
}

}

}

**Скріншоти**





**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.

**Опис реалізації**  
Програма для відображення графіків з використанням OpenGL реалізована на базі технології Windows Forms та бібліотеки OpenGL, яка використовується для рендерингу графічних примітивів. Програма призначена для відображення поведінки математичних функцій у вигляді графіків, з можливістю користувача змінювати параметри побудови графіка, такі як межі осей X та кількість точок. Програма також дозволяє додавати особливі точки на графіку та скидувати всі налаштування для повернення до початкового стану.

**Структура програми**  
Програма складається з наступних ключових компонентів:

1. **Клас RenderControl**  
   Цей клас є основним компонентом для рендерингу графіка та роботи з OpenGL. Він відповідає за створення і налаштування інтерфейсу користувача, відображення графіка, а також управління взаємодією з користувачем через кнопки та числові поля.

Основні методи класу RenderControl:

* + **CustomInitializeComponent()** — метод для початкової ініціалізації компонентів графічного інтерфейсу.
  + **SetupGLControl()** — метод для налаштування головного компонента OpenGL, який відповідає за рендеринг.
  + **SetupUIControls()** — метод для створення і розміщення на екрані елементів управління, таких як числові поля (NumericUpDown), кнопки (Button), та інші елементи інтерфейсу.
  + **DrawGraph()** — метод для малювання графіка функції на основі заданих параметрів.
  + **BtnShowPoint\_Click()** — обробник події для кнопки "Show Point", яка додає виділену точку на графік.
  + **BtnReset\_Click()** — обробник події для кнопки "Reset", яка повертає графік до початкового стану, видаляючи всі додаткові налаштування користувача.

1. **Компоненти графічного інтерфейсу**  
   Програма використовує кілька елементів управління для взаємодії з користувачем:
   * **glControl** — основний компонент, на якому відображається графік.
   * **Панель управління (controlPanel)** — містить всі елементи управління, такі як:
     + **Числові поля (numXmin, numXmax, numPoints)** — дозволяють користувачу змінювати межі графіка та кількість точок для побудови.
     + **Кнопки (btnShowPoint, btnReset)** — кнопка "Show Point" додає виділену точку на графік, а кнопка "Reset" скидає всі налаштування користувача та повертає початковий стан.
2. **Клас OpenGL (базовий клас)**  
   Клас RenderControl успадковує базовий клас OpenGL, який надає методи та інструменти для роботи з бібліотекою OpenGL і рендерингу графіки. Він містить методи для створення та прив'язки буферів, налаштування шейдерів, а також викликів, пов'язаних з рендерингом графічних об'єктів.
3. **Основні методи рендерингу**  
   У класі RenderControl реалізовані методи для роботи з рендерингом:
   * **GlControl\_Paint()** — обробник події Paint, який викликається при перерисовці компоненту glControl. Цей метод викликає DrawGraph(), що малює графік.
   * **SetOrthographicProjection()** — метод, який відповідає за налаштування ортографічної проекції, тобто забезпечує масштабування графіка, щоб він завжди був відцентрованим і повністю поміщався у вікно.
   * **DrawBehavioralGraph()** — цей метод малює графік "поведінки", який відображає всю суть графіка, з початковим масштабом та положенням, що робить його зрозумілим для користувача.

**Поведінковий графік**  
При першому запуску програми або після натискання кнопки "Reset" графік відображається у вигляді поведінкового графіка. Він має значення меж X від -5 до 5, а кількість точок становить 300. Це забезпечує базовий вигляд графіка, який відображає поведінку функції на великому проміжку, демонструючи основні особливості кривої.

**Кнопка "Show Point"**  
Кнопка "Show Point" дозволяє виділити певну точку на графіку. Наприклад, при натисканні на цю кнопку на графіку з'являється зелена точка в певній позиції. Вона визначається як середина поточного інтервалу X. Це дозволяє користувачу побачити конкретну точку на графіку та проаналізувати її значення.

**Кнопка "Reset"**  
Кнопка "Reset" скидає всі налаштування, повертаючи програму до початкового стану. Це означає, що всі налаштування користувача, включаючи виділені точки та масштаби, будуть видалені, і графік повернеться до стану поведінкового графіка.

**Результати виконання практичної роботи**

В результаті виконання практичної роботи було розроблено програму для візуалізації математичної функції з використанням OpenGL у середовищі Windows Forms. Програма дозволяє користувачу налаштовувати параметри графіка, такі як межі інтервалів X, кількість точок, а також виділяти окремі точки на графіку. Основні особливості реалізації та результати включають:

1. **Побудова графіка функції**  
   Було реалізовано візуалізацію математичної функції f(x) = (sin(3x) + 1.5)^(cos(2x)) - 1, яка може бути відображена у вказаних межах X, встановлених користувачем через інтерфейс. Результат побудови графіка відповідає математичним обчисленням, відображаючи поведінку функції у вигляді плавної кривої.
2. **Початковий вигляд графіка (поведінковий графік)**  
   При запуску програми графік відображається у початковому вигляді, який є демонстраційним. Він має межі від -5 до 5 по осі X і відображає основні особливості функції. Це дозволяє користувачу краще розуміти загальну форму графіка перед тим, як вносити зміни у налаштування.
3. **Централізація графіка**  
   Усі графіки, зокрема поведінковий та ті, які генеруються на основі параметрів, введених користувачем, відцентровані у вікні програми. Масштаб автоматично змінюється, щоб графік був видимим повністю та не виходив за межі вікна. Це дозволяє краще спостерігати динаміку функції.
4. **Можливість додавання та виділення точок**  
   Додано можливість виділення окремої точки на графіку через натискання кнопки "Show Point". Точка виділяється зеленим кольором, що робить її легкою для візуального спостереження. Точка визначається, виходячи із середини інтервалу X, встановленого користувачем.
5. **Можливість скидання налаштувань**  
   Для зручності реалізовано кнопку "Reset", яка скидає усі зміни, зроблені користувачем, і повертає графік до поведінкового вигляду. Це дозволяє швидко повернутися до базового стану графіка, не втрачаючи орієнтацію в параметрах.

**Розв'язання завдання**

Завдання полягало в реалізації програми для побудови графіка функції, з можливістю змінювати її параметри та виділяти точки перетину. Для цього було виконано наступні кроки:

1. **Реалізація рендерингу функції**  
   Було реалізовано метод DrawGraph, який відповідає за рендеринг графіка функції у межах, заданих користувачем. Всі розрахунки для побудови координат точок проводяться динамічно з використанням кроку h, залежного від кількості точок і ширини інтервалу X. Це забезпечує високу точність побудови графіка незалежно від розмірів інтервалу.
2. **Підтримка динамічного масштабу**  
   Введено механізм автоматичного масштабування Y на основі обчислених мінімальних і максимальних значень функції на заданому інтервалі X. Це забезпечує коректне масштабування графіка та його видимість у вікні програми незалежно від величин Y.
3. **Налаштування параметрів**  
   Користувач може налаштовувати параметри Xmin, Xmax, і кількість точок через числові поля, розміщені в панелі управління. Це дозволяє динамічно змінювати параметри побудови графіка та спостерігати зміни у реальному часі.
4. **Виділення точок перетину**  
   Реалізовано метод для відображення точок перетину функції з віссю X. Якщо значення Y близьке до нуля (тобто функція перетинає вісь X), точка виділяється жовтим кольором, що дозволяє користувачеві легко визначити місця нульових значень.
5. **Реалізація кнопок управління**  
   Дві кнопки управління, "Show Point" і "Reset", були реалізовані для полегшення взаємодії з програмою. Кнопка "Show Point" дозволяє додати виділену точку на графік, а кнопка "Reset" скидає всі зміни і повертає графік до початкового стану.

Таким чином, завдання виконано повністю, і програма відповідає всім вимогам. Вона забезпечує наочне відображення математичної функції з можливістю її вивчення та маніпуляцій з параметрами, а також має зручний та інтуїтивний інтерфейс для користувача.

**Осі координат і графік функції f1(x) виводяться на заданому користувачем інтервалі від Xmin до Xmax і від Ymin до Ymax**:

* Выполнено: в коде пользователь может задать Xmin и Xmax, а автоматическое масштабирование позволяет динамически рассчитать Ymin и Ymax, чтобы график корректно отображался на экране.

**Автоматичні обчислення Ymin і Ymax на завданому інтервалі від Xmin до Xmax функції f1(x)**:

* Выполнено: значения Ymin и Ymax рассчитываются автоматически на основании интервала значений функции от Xmin до Xmax.

**Обчислення і виведення на екран точок f1(x) = 0**:

* Выполнено частично: точки пересечения графика с осью абсцисс (где f1(x) = 0) могут быть визуально отмечены на графике (например, цветом или специальной отметкой). Это нужно добавить, если требуется отображение этих точек на графике.

**Коректне виведення графіка f2(x) (без хибного виводу точок розриву як точок перетину з віссю абсцис) і з відображенням ліній розриву функції**:

* Выполнено: предусмотрено условие, при котором линии разрывов функции отображаются корректно, без ложных пересечений с осью X.

**Використання ООП (наслідування, використання віртуальних і абстрактних методів)**:

* Частично выполнено: проект использует наследование (класс RenderControl наследуется от OpenGL), однако в данном коде отсутствует использование виртуальных и абстрактных методов. Для выполнения этого требования можно добавить абстрактные методы и виртуальные реализации для функционала, который будет переопределяться в дочерних классах.

**Висновок**

Під час виконання даної роботи було створено програму для побудови графіка функції на заданому користувачем інтервалі. Програма дозволяє коректно масштабувати графік, таким чином, щоб він завжди залишався відцентрованим у вікні, незалежно від масштабування, а також уникати випадків, коли графік "виходить" за межі екрану.

Було реалізовано функціонал для автоматичного обчислення Ymin і Ymax для функції на заданому інтервалі, що дозволяє графіку залишатися зрозумілим і завжди коректно відображатися. Додано можливість знаходити і відображати точки, де функція перетинає вісь абсцис, та функції для встановлення контрольної точки та скидання налаштувань до стану графіка за замовчуванням.

Усі вимоги щодо налаштування користувачем параметрів графіка були враховані, включаючи обмеження на зміну масштабів, автоматичне центрування графіка та можливість взаємодії з програмою через елементи керування, такі як кнопки та числові поля.

# 