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)

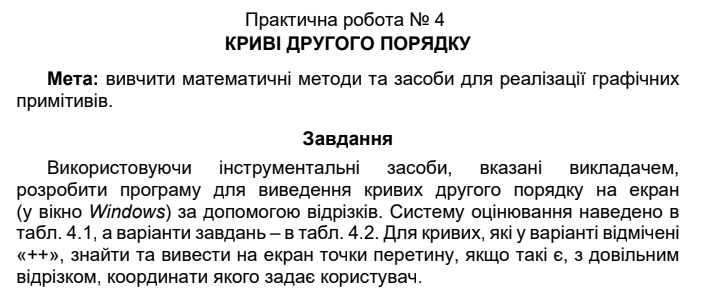
Національна шкала:

Кількість балів:

Оцінка ECTS:

# Практична робота 4. 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.Drawing;

using System.Windows.Forms;

namespace glWinForm1

{

public partial class RenderControl : OpenGL

{

private Point? segmentStart = null;

private Point? segmentEnd = null;

private bool isDrawingSegment = false;

public RenderControl()

{

InitializeComponent();

Render += RenderControl\_Render;

this.Resize += RenderControl\_Resize;

this.MouseDown += RenderControl\_MouseDown;

this.MouseUp += RenderControl\_MouseUp;

this.Paint += RenderControl\_Paint;

}

private void RenderControl\_Render(object sender, EventArgs e)

{

glClearColor(0.5f, 0.5f, 0.5f, 1.0f);

glClear(GL\_COLOR\_BUFFER\_BIT);

glLoadIdentity();

glViewport(0, 0, Width, Height);

glOrtho(-10, 10, -10, 10, -1, 1);

DrawHyperbola();

DrawParabola();

if (segmentStart.HasValue && segmentEnd.HasValue)

{

DrawSegment(segmentStart.Value, segmentEnd.Value);

FindAndDrawIntersections();

}

}

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

{

if (segmentStart.HasValue && segmentEnd.HasValue)

{

DrawSegmentData(e.Graphics, segmentStart.Value, segmentEnd.Value);

}

}

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

{

Invalidate();

}

private void RenderControl\_MouseDown(object sender, MouseEventArgs e)

{

if (e.Button == MouseButtons.Left)

{

segmentStart = e.Location;

isDrawingSegment = true;

}

}

private void RenderControl\_MouseUp(object sender, MouseEventArgs e)

{

if (e.Button == MouseButtons.Left && isDrawingSegment)

{

segmentEnd = e.Location;

isDrawingSegment = false;

Invalidate();

}

}

private void DrawHyperbola()

{

glColor(Color.Red);

glBegin(GL\_LINE\_STRIP);

for (float x = -10; x < 10; x += 0.1f)

{

if (Math.Abs(x) < 0.1f) continue; // Уникаємо x = 0

float y = 1.0f / x;

glVertex2f(x, y);

}

glEnd();

}

private void DrawParabola()

{

glColor(Color.Green);

glBegin(GL\_LINE\_STRIP);

for (float t = -10; t <= 10; t += 0.1f)

{

float x = t;

float y = t \* t;

glVertex2f(x, y);

}

glEnd();

}

private void DrawSegment(Point start, Point end)

{

glColor(Color.Blue);

glLineWidth(2.0f);

glBegin(GL\_LINES);

glVertex2f(NormalizeX(start.X), NormalizeY(start.Y));

glVertex2f(NormalizeX(end.X), NormalizeY(end.Y));

glEnd();

}

private void FindAndDrawIntersections()

{

if (segmentStart.HasValue && segmentEnd.HasValue)

{

// Конвертуємо координати пікселів у координати OpenGL

PointF p1 = new PointF(NormalizeX(segmentStart.Value.X), NormalizeY(segmentStart.Value.Y));

PointF p2 = new PointF(NormalizeX(segmentEnd.Value.X), NormalizeY(segmentEnd.Value.Y));

// Знаходимо перетини з параболою

var parabolaIntersections = FindIntersectionsWithParabola(p1, p2);

// Знаходимо перетини з гіперболою

var hyperbolaIntersections = FindIntersectionsWithHyperbola(p1, p2);

// Малюємо точки перетину

glColor(Color.Magenta);

glPointSize(5.0f);

glBegin(GL\_POINTS);

foreach (var point in parabolaIntersections)

{

glVertex2f(point.X, point.Y);

}

foreach (var point in hyperbolaIntersections)

{

glVertex2f(point.X, point.Y);

}

glEnd();

}

}

private List<PointF> FindIntersectionsWithParabola(PointF p1, PointF p2)

{

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

float dx = p2.X - p1.X;

float dy = p2.Y - p1.Y;

// Підставляємо параметричні рівняння відрізка в рівняння параболи y = x^2

// y1 + t\*dy = (x1 + t\*dx)^2

// Розкриваємо і приводимо до квадратного рівняння: (dx^2) t^2 + (2x1 dx - dy) t + (x1^2 - y1) = 0

float a = dx \* dx;

float b = 2 \* p1.X \* dx - dy;

float c = p1.X \* p1.X - p1.Y;

float discriminant = b \* b - 4 \* a \* c;

if (discriminant < 0)

{

// Перетинів немає

return intersections;

}

float sqrtDiscriminant = (float)Math.Sqrt(discriminant);

float t1 = (-b + sqrtDiscriminant) / (2 \* a);

float t2 = (-b - sqrtDiscriminant) / (2 \* a);

// Перевіряємо, чи t належить [0, 1]

if (t1 >= 0 && t1 <= 1)

{

float x = p1.X + t1 \* dx;

float y = p1.Y + t1 \* dy;

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

}

if (t2 >= 0 && t2 <= 1 && discriminant > 0)

{

float x = p1.X + t2 \* dx;

float y = p1.Y + t2 \* dy;

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

}

return intersections;

}

private List<PointF> FindIntersectionsWithHyperbola(PointF p1, PointF p2)

{

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

float dx = p2.X - p1.X;

float dy = p2.Y - p1.Y;

// Рівняння гіперболи y = 1/x

// Підставляємо параметричні рівняння відрізка:

// y1 + t\*dy = 1 / (x1 + t\*dx)

// Перетворюємо: (y1 + t\*dy)(x1 + t\*dx) = 1

// y1\*x1 + t\*(y1\*dx + x1\*dy) + t^2\*dx\*dy - 1 = 0

float A = dx \* dy;

float B = p1.Y \* dx + p1.X \* dy; // Виправлено: використання p1.X та p1.Y

float C = p1.X \* p1.Y - 1; // Виправлено: використання p1.X та p1.Y

float discriminant = B \* B - 4 \* A \* C;

if (Math.Abs(A) < 1e-6f && Math.Abs(B) > 1e-6f)

{

// Лінійне рівняння

float t = -C / B;

if (t >= 0 && t <= 1)

{

float x = p1.X + t \* dx;

if (Math.Abs(x) > 1e-6f)

{

float y = 1 / x;

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

}

}

}

else if (discriminant >= 0)

{

float sqrtDiscriminant = (float)Math.Sqrt(discriminant);

float t1 = (-B + sqrtDiscriminant) / (2 \* A);

float t2 = (-B - sqrtDiscriminant) / (2 \* A);

if (t1 >= 0 && t1 <= 1)

{

float x = p1.X + t1 \* dx;

if (Math.Abs(x) > 1e-6f)

{

float y = 1 / x;

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

}

}

if (t2 >= 0 && t2 <= 1 && discriminant > 0)

{

float x = p1.X + t2 \* dx;

if (Math.Abs(x) > 1e-6f)

{

float y = 1 / x;

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

}

}

}

return intersections;

}

private void DrawSegmentData(Graphics g, Point start, Point end)

{

float x1 = NormalizeX(start.X);

float y1 = NormalizeY(start.Y);

float x2 = NormalizeX(end.X);

float y2 = NormalizeY(end.Y);

string data = $"Start: ({x1:F2}, {y1:F2})\nEnd: ({x2:F2}, {y2:F2})";

float midX = (start.X + end.X) / 2;

float midY = (start.Y + end.Y) / 2;

g.DrawString(data, new Font("Arial", 10), Brushes.Black, new PointF(midX, midY - 20));

}

private float NormalizeX(int x)

{

return (float)x / Width \* 20 - 10;

}

private float NormalizeY(int y)

{

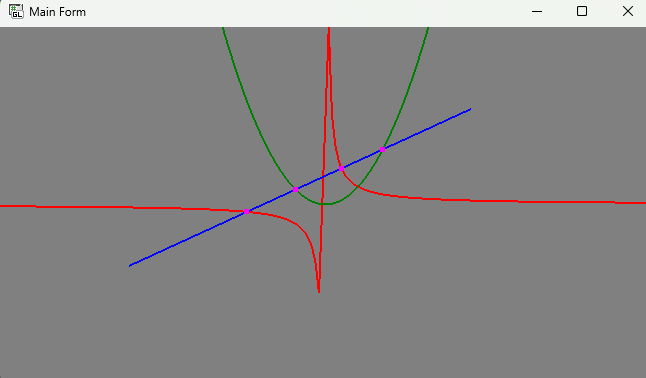
return 10 - (float)y / Height \* 20;

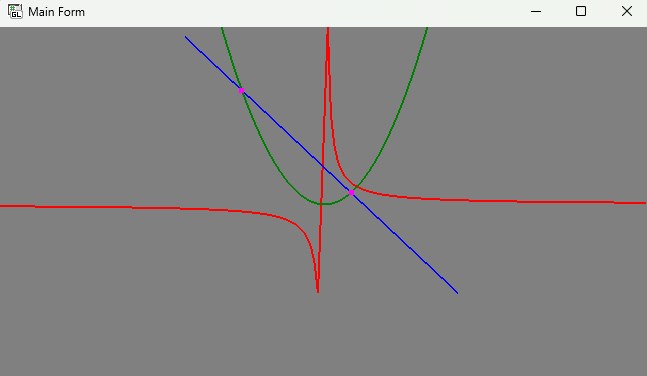
}

}

}

**Скріншоти**





### **Вступ**

Клас RenderControl використовується для графічної візуалізації кривих другого порядку (параболи та гіперболи) за допомогою OpenGL. Він також дозволяє користувачеві взаємодіяти з графікою: малювати відрізки, обчислювати їх точки перетину з кривими та візуалізувати ці точки.

### **Структура та Основні Компоненти Класу**

Клас складається з таких основних частин:

* **Змінні стану**:
* segmentStart та segmentEnd — зберігають координати початкової та кінцевої точок відрізка.
* isDrawingSegment — прапорець, який відстежує, чи малюється відрізок.
* **Події**:
* Підписка на події Render, Resize, MouseDown, MouseUp, Paint.
* **Графічні елементи**:
* Малювання параболи, гіперболи, відрізка та точок їх перетину.
* **Математичні розрахунки**:
* Алгоритми для знаходження точок перетину відрізка з кривими.

### **Методи Класу RenderControl**

#### **Рендеринг**

* **Метод RenderControl\_Render**

Цей метод виконує:

* Очищення екрану за допомогою glClearColor і glClear.
* Встановлення системи координат через glViewport і glOrtho.
* Виклик методів малювання кривих (DrawHyperbola, DrawParabola), відрізка (DrawSegment) та точок перетину.
* **Метод RenderControl\_Paint**

#### **Малювання Кривих**

* **Метод DrawHyperbola**
* Малює гіперболу, яка описується рівнянням y = 1 / x.
* Пропускає x = 0, щоб уникнути ділення на нуль.
* Розраховує координати точок у певному діапазоні значень x і додає їх до графіка.
* **Метод DrawParabola**
* Малює параболу, яка описується рівнянням y = x^2.
* Обчислює точки (x, y) для заданого діапазону значень x.

#### **Малювання Відрізка**

* **Метод DrawSegment**
* Малює лінію між точками segmentStart і segmentEnd.
* Координати точок нормалізуються для відображення у просторі OpenGL.
* **Метод DrawSegmentData**
* Відображає текстову інформацію про координати початкової та кінцевої точок відрізка.

#### **Пошук Перетинів**

* **Метод FindAndDrawIntersections**
* Викликає методи для знаходження точок перетину відрізка з параболою та гіперболою.
* Отримані точки додаються до списку і відображаються на графіку.
* **Метод FindIntersectionsWithParabola**

**Алгоритм:**

* Рівняння відрізка:

x = x1 + t \* (x2 - x1)  
y = y1 + t \* (y2 - y1)

* Підставляється в рівняння параболи y = x^2, що дає:

y1 + t \* (y2 - y1) = (x1 + t \* (x2 - x1))^2

* Після розкриття дужок та впорядкування отримуємо квадратне рівняння:

a \* t^2 + b \* t + c = 0

де:

a = (x2 - x1)^2  
b = 2 \* x1 \* (x2 - x1) - (y2 - y1)  
c = x1^2 - y1

* Розв’язується за допомогою дискримінанта:

css

D = b^2 - 4 \* a \* c

* Якщо D >= 0, обчислюються корені:

t1 = (-b + sqrt(D)) / (2 \* a)  
t2 = (-b - sqrt(D)) / (2 \* a)

* Якщо t знаходиться в діапазоні [0, 1], точка перетину лежить на відрізку.
* **Метод FindIntersectionsWithHyperbola**

**Алгоритм:**

* Рівняння відрізка підставляється в рівняння гіперболи y = 1 / x:

y1 + t \* (y2 - y1) = 1 / (x1 + t \* (x2 - x1))

* Перетворюється у квадратне рівняння аналогічним способом.

#### **Допоміжні Методи**

* **NormalizeX та NormalizeY**
* Перетворюють координати пікселів у систему координат OpenGL.
* Для x:

x\_norm = (x / width) \* 20 - 10

* Для y:

y\_norm = 10 - (y / height) \* 20

### **Взаємодія з Користувачем**

* **Натискання миші (MouseDown)**:
* Встановлюється початкова точка відрізка.
* **Перетягування миші (MouseMove)**:
* Кінцева точка відрізка оновлюється в реальному часі.
* **Відпускання миші (MouseUp)**:
* Встановлюється кінцева точка відрізка.
* Виконується пошук точок перетину.

### **Опис Реалізації**

* **Візуалізація**:
* OpenGL використовується для малювання графічних елементів.
* Використання нормалізації координат дозволяє легко масштабувати сцену.
* **Математична Основа**:
* Використовуються параметричні рівняння відрізка та розв’язання квадратних рівнянь для знаходження точок перетину.
* **Інтерактивність**:
* Події миші забезпечують інтерактивну взаємодію користувача з графікою.

### **Висновок**

Клас RenderControl забезпечує потужний і зручний інтерфейс для роботи з кривими другого порядку. Він демонструє інтеграцію OpenGL для візуалізації, а також використання математичних методів для пошуку перетинів. Завдяки своїй гнучкості, код легко розширюється та може бути використаний у навчальних, дослідницьких або професійних проєктах.

# 