МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського

«Харківський авіаційний інститут»

Факультет програмної інженерії та бізнесу

Кафедра інженерії програмного забезпечення

Практична робота №1

Minor *«Розробник ігрових додатків»*

дисципліна *«Комп’ютерна графіка з OpenGL»*

(назва дисципліни)

Виконав: студент 3 курсу групи  *535б*

напряму підготовки (спеціальності):

*123 Комп’ютерна інженерія*

(шифр і назва напряму підготовки / спеціальності)

*Лашин Д.О.*

(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 1. Основні принципи роботи з OpenGL

**Мета**: навчитися створювати прості програмні додатки, які

використовують бібліотеку OpenGL. Оволодіти способом рисування

плоских опуклих фігур за допомогою геометричних примітивів та освоїти

встановлення системи координат командами OpenGL.

## Завдання, варіант № 12

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 AMD Ryzen 7 5800H with Radeon Graphics

RAM 16,0 ГБ

System type 64-розрядна операційна система, процесор з архітектурою x64

Edition Windows 11

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

## 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.

## Результати виконання практичної роботи

### Розв'язання завдання

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

* колір, glColor3f() рядок 16 у файлі Class1.cs;
* тип, glLineStipple(), glEnable()/glDisable(), рядок 18 у файлі Class1.cs;
* товщина glLineWidth(), рядок 39 у файлі Class1.cs

Коректне відображення завдання під час змінення розмірів/положення вікна наведено у рис. 1.1 та 1.2

Розроблення підпрограм для виключення дублювання коду наведено у рядках 21 – 32 файлу Class1.cs

Застосування циклів для створення зображень наведено у рядках 36 – 65 файлу Class1.cs.

Використання ООП реалізовано за допомогою розроблення власних класів, які наведено у файлі Class1.cs Додатку А.

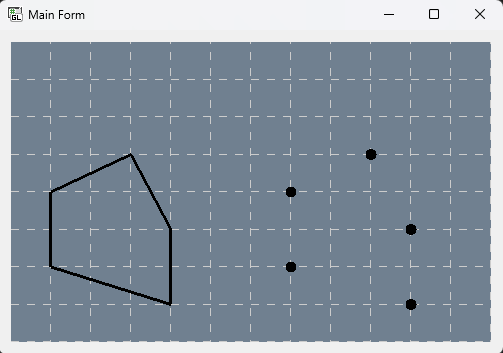


Рисунок 1.1 – Тестування програми після першого запуску

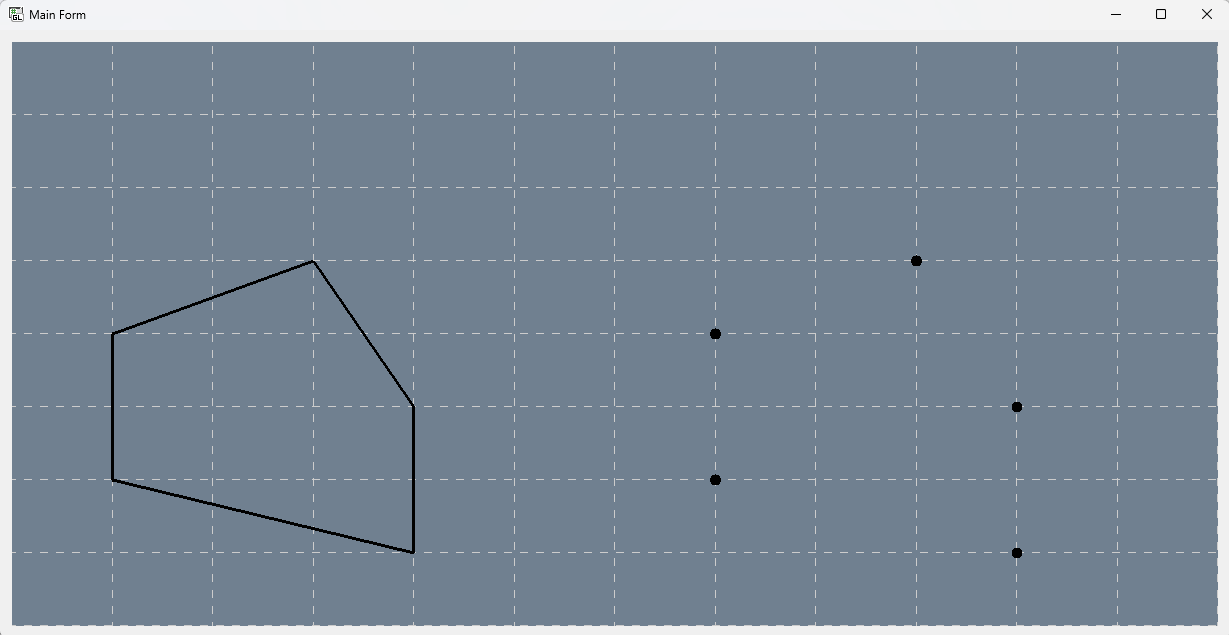


Рисунок 1.2 – Тестування програми при зміні розмірів вікна

### 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.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № 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 | **+** |

# Додаток А. Лістинг програми до практичної роботи №1

### Код файлу (Class1.cpp)

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Task01

{

public partial class RenderControl

{

internal class Class1

{

public void DrawGrid()

{

glColor3f(0.8f, 0.8f, 0.8f);

glLineWidth(1.0f); // Товщина ліній сітки

glEnable(GL\_LINE\_STIPPLE); // Включення режиму ліній

glLineStipple(1, 0x00FF);

glBegin(GL\_LINES);

for (double i = -8; i <= 12; i += 0.5)

{

glVertex2d(i, -4);

glVertex2d(i, 8);

}

for (double j = -4; j <= 8; j += 0.5)

{

glVertex2d(-8, j);

glVertex2d(12, j);

}

glEnd();

glDisable(GL\_LINE\_STIPPLE); // Відключення пунктирних ліній

}

// Метод зображення фігури

public void DrawFigure()

{

glColor3f(0.0f, 0.0f, 0.0f); // Колір контуру

glLineWidth(3.0f); // Товщина ліній фігури

glBegin(GL\_LINE\_STRIP);

Figure(-3);

glEnd();

}

public void Figure(double offsetX = 0)

{

glVertex2d(-0.5 + offsetX, -1); // Нижня ліва точка

glVertex2d(-0.5 + offsetX, 0); // Верхня ліва точка

glVertex2d(0.5 + offsetX, 0.5); // Верхня центральна точка

glVertex2d(1 + offsetX, -0.5); // Верхня права точка

glVertex2d(1 + offsetX, -1.5); // Нижня права точка

glVertex2d(-0.5 + offsetX, -1); // Нижня ліва точка

}

// Метод зображення точок

public void DrawPoints()

{

glEnable(GL\_POINT\_SMOOTH);

glColor3f(0.0f, 0.0f, 0.0f);

glPointSize(10.0f);

glBegin(GL\_POINTS);

Figure(0);

glEnd();

glDisable(GL\_POINT\_SMOOTH);

}

}

}

### }Код файлу (RenderControl.cs)

using System;

using System.Collections.Generic;

using System;

using System.Windows.Forms;

using System.Linq;

using System.Threading.Tasks;

namespace Task01

{

public partial class RenderControl : OpenGL

{

private Timer renderTimer;

public RenderControl()

{

InitializeComponent();

renderTimer = new Timer { Interval = 16 };

renderTimer.Tick += RenderGL;

renderTimer.Start();

}

private void RenderGL(object sender, EventArgs e)

{

glViewport(0, 0, Width, Height);

glLoadIdentity();

gluOrtho2D(-4, 2, -2, 2);

glClear(GL\_COLOR\_BUFFER\_BIT);

Class1 task = new Class1();

task.DrawGrid(); // Координатная сетка

task.DrawFigure(); // Фигура

task.DrawPoints(); // Точки

}

}

}

# додаток б посилання на репозиторій проєкту

<https://github.com/lashin-d/OpenGL>

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського

«Харківський авіаційний інститут»

Факультет програмної інженерії та бізнесу

Кафедра інженерії програмного забезпечення

Практична робота №2

Minor *«Розробник ігрових додатків»*

дисципліна *«Комп’ютерна графіка з OpenGL»*

(назва дисципліни)

Виконав: студент 3 курсу групи  *535б*

напряму підготовки (спеціальності):

*123 Комп’ютерна інженерія*

(шифр і назва напряму підготовки / спеціальності)

*Лашин Д.О.*

(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. Основні принципи роботи з OpenGL

**Мета**: вивчити поняття теселяції і навчитися використовувати графічні

примітиви OpenGL для створення поверхонь. Освоїти обробку подій

клавіатури і маніпулятора «миша» для створення інтерактивних застосунків..

## Завдання, варіант № 12

Використовуючи інструментальні засоби, що вказані викладачем, і беручи до уваги вимоги, що наведено в таблиці, 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). Приклад початкового стану застосунку показано на рис. 2.1.

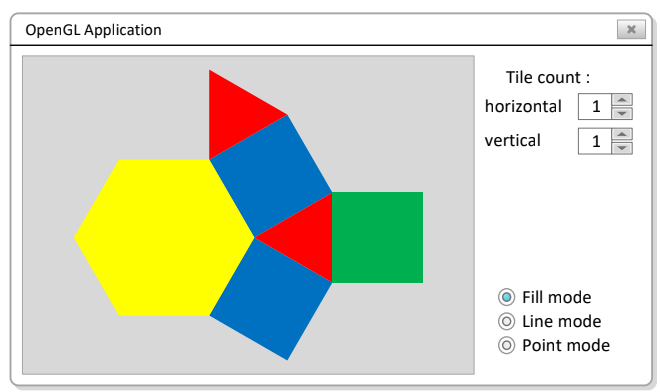


Рисунок 2.1 – Вигляд застосунку після старту

Усі варіанти заданій основані на правильних багатокутниках, розмір яких

визначається величиною одного ребра. Для зафарбування пропонується

використовувати шість кольорів: білий, сірий (35 %), червоний, зелений,

синій и жовтий.

За допомогою клавіатури або маніпулятора «миша» користувач повинен

мати можливість виконати замощення (tessellation, tilling) робочої області

по горизонталі і вертикалі. При цьому систему координат необхідно

скорегувати таким чином, щоб замощена поверхня розташовувалася у

центрі робочої області. Приклад замощення робочої області застосунку

показано на рис. 2.2.

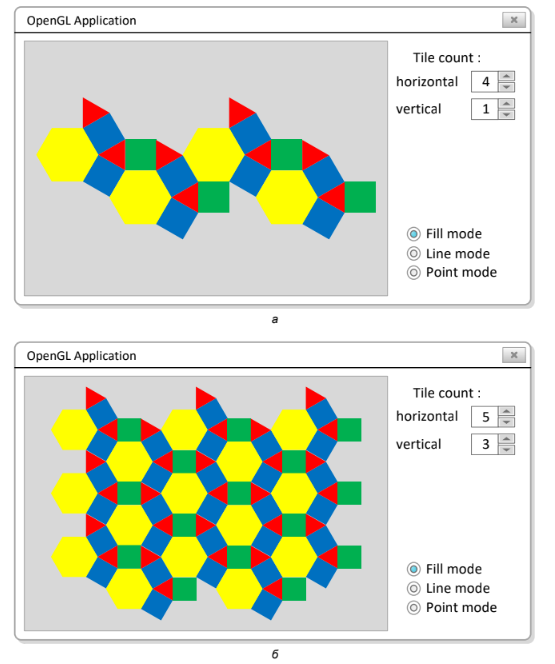


Рисунок 2.2 – Вигляд застосунку під час замощення:

а – тільки по горизонталі; б – по горизонталі та вертикалі

Крім цього, користувач повинен мати можливість змінювати режим

відображення графічних примітивів OpenGL: точкове (тільки вершини

фігури), контурне (рис. 2.3) і з заповненням кольором (див. рис. 2.2).

Передбачається, що перемикання між режимами виконують за подією від

клавіатури і/або маніпулятора «миша». При цьому можна використовувати

як стандартні елементи керування, так і власні, що реалізовані та

відображені засобами OpenGL (для підвищеного рівня складності,

див. табл. 2.1).

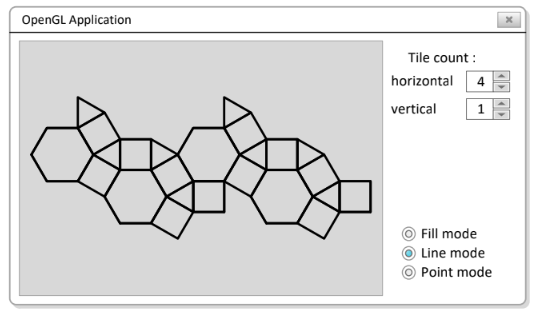


Рисунок 2.3 – Приклад керування режимом виводу графічних примітивів

|  |
| --- |
|  |
| Рисунок 2.4 – Варіант 12 |

## Системна інформація

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

Processor AMD Ryzen 7 5800H with Radeon Graphics

RAM 16,0 ГБ

System type 64-розрядна операційна система, процесор з архітектурою x64

Edition Windows 11

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

## Теоретичні відомості

Під час виводу зображення необхідно враховувати, що кожна поверхня

графічного примітиву OpenGL має дві сторони і режим виводу для кожної з

них може бути налаштований окремо за допомогою команди

glPolygonMode.

Для зміни режиму (моделі) розфарбування використовують команду

glShadeModel. Якщо режим зафарбування напівтонами вимкнений, то

колір примітиву визначається кольором тільки однієї вершини. Наприклад,

для GL\_TRIANGLE\_STRIP колір першого трикутника визначається

кольором третьої вершини, другого – четвертої вершини і т. д.

Для встановлення шаблону зафарбування необхідно задіяти команди-

перемикачі glEnable / glDisable (як і для шаблону лінії).

## Результати виконання практичної роботи

### Розв'язання завдання

Розв'язання завдання

Для управління параметрами примітивів було використано наступні команди:

* колір, glColor3d() у рядках 25, 32, 39, 46, 53, 60 файлу figure.cs;

GL\_TRIANGLES у рядках 24, 31, 38, 45, 52, 59, файлу figure.cs

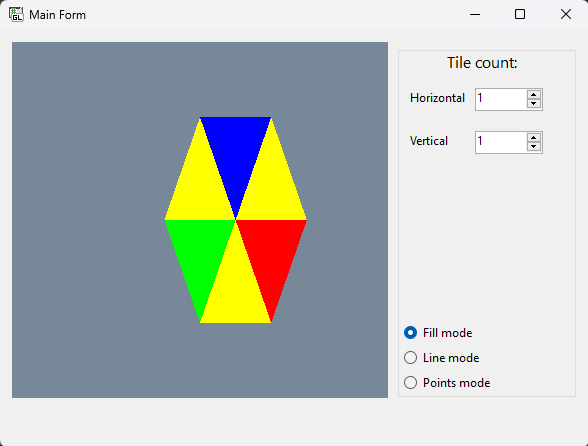


Рисунок 2.5 – Тестування програми після першого запуску

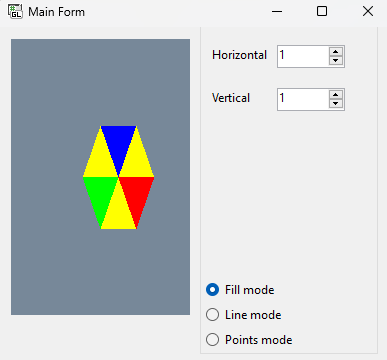


Рисунок 2.6 – Тестування програми при зміні розмірів вікна

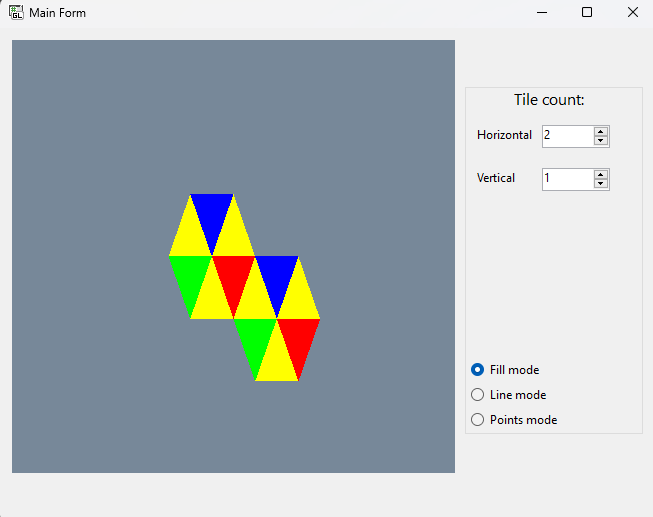
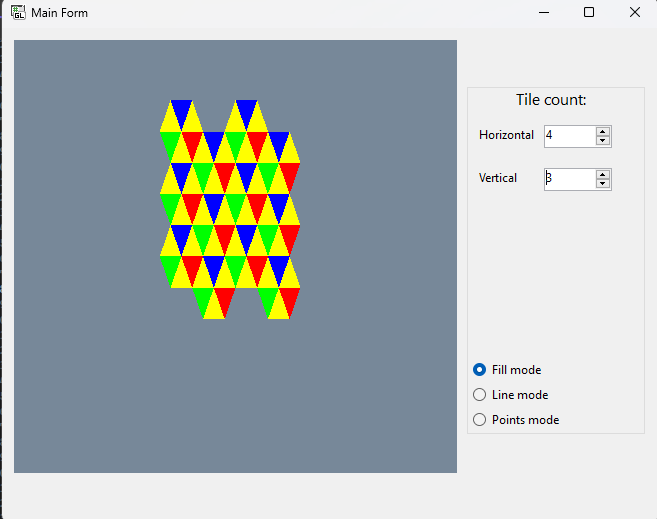
  


Рисунок 2.7-8 – Тестування додавання плиток

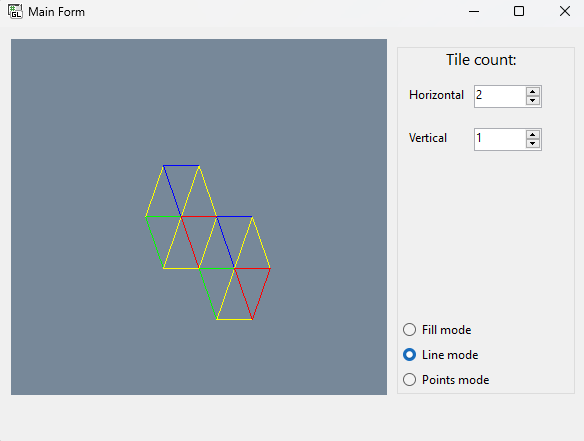


Рисунок 2.9 – Тестування відображення ліній

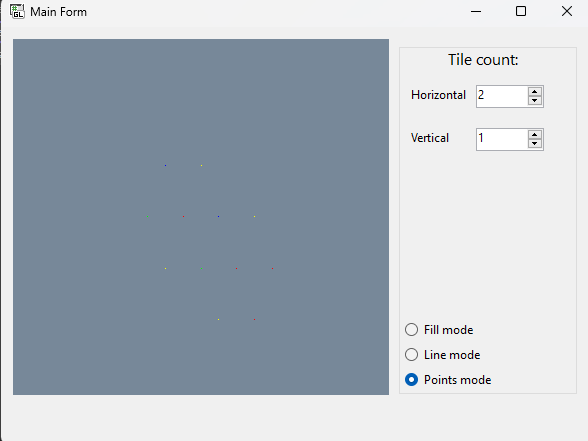


Рисунок 2.10 – Тестування відображення точок

### 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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № з/п | Складність | Вимоги | Бали | Зроблено |
| 1 | Базовий рівень | Під час запуску застосунку зображення відповідає варіанту завдання з однією плиткою. | 1 | **+** |
| 2 | Багаторазове 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 (розробка власних класів) | 1 | **+** |

# Додаток А. Лістинг програми до практичної роботи №1

### Код файлу (MainForm.cs)

using System.Windows.Forms;

using System;

using System.Drawing;

using static Task02.OpenGL;

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

namespace Task02

{

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();

}

}

private void groupBox1\_Enter(object sender, EventArgs e)

{

}

}

}

### 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 Task02.RenderControl;

namespace Task02

{

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)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glLoadIdentity();

// Установить квадратное окно просмотра

if (Width > Height)

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

else

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

// Рассчитать коэффициент масштаба

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

glOrtho(

-150 \* scaleFactor, 350 \* scaleFactor, // Горизонтальные границы

-150 \* scaleFactor, 150 \* scaleFactor, // Вертикальные границы

-1, 1 // Глубина

);

// Размер стороны фигуры и высота треугольника

double sideSize = 100;

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

// Рисование фигур в сетке

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

{

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

{

// Смещение по X и Y для текущей фигуры

double offsetX = col \* 1.5 \* sideSize;

double offsetY = row \* 2 \* height;

// Смещение для нечетных колонок

if (col % 2 != 0)

offsetY -= height;

// Отрисовка фигуры

\_f.DrawComplexFigure(sideSize, DrawMode, offsetX, offsetY);

}

}

glDisable(GL\_LINES);

}

}

}

***Код файлу (figure.cs)***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Task02

{

public partial class RenderControl

{

public class figure

{

readonly double cursorCenterX = -3;

readonly double cursorCenterY = 0;

public void DrawComplexFigure(double sidesize = 8.5, uint DrawMode = GL\_FILL, double offsetX = 0, double offsetY = 0)

{

double height = Math.Sqrt(3) / 2 \* sidesize; // Висота трикутника

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

// Лівий верхній жовтий чотирикутник (трапеція)

glBegin(GL\_TRIANGLES);

glColor3d(1, 1, 0); // Жовтий колір

glVertex2d(offsetX + sidesize, offsetY + height); // Верхній кут

glVertex2d(offsetX + 50, offsetY); // Лівий кут

glVertex2d(offsetX + sidesize + 50, offsetY); // Правий кут

glEnd();

glBegin(GL\_TRIANGLES);

glColor3d(0, 1, 0); // Зелений колір

glVertex2d(offsetX + sidesize, offsetY - height); // Нижній кут

glVertex2d(offsetX + sidesize + 50, offsetY); // Правий кут

glVertex2d(offsetX + 50, offsetY); // Лівий кут

glEnd();

glBegin(GL\_TRIANGLES);

glColor3d(0, 0, 1); // Синій колір

glVertex2d(offsetX + sidesize, offsetY + height); // Лівий кут

glVertex2d(offsetX + sidesize \* 2 , offsetY + height); // Правий кут

glVertex2d(offsetX + sidesize + 50, offsetY); // Нижній кут

glEnd();

glBegin(GL\_TRIANGLES);

glColor3d(1, 1, 0); // Жовтий колір

glVertex2d(offsetX + sidesize + 50, offsetY); // Верхній кут

glVertex2d(offsetX + sidesize, offsetY - height); // Лівий кут

glVertex2d(offsetX + sidesize \* 2, offsetY - height); // Правий кут

glEnd();

glBegin(GL\_TRIANGLES);

glColor3d(1, 1, 0); // Жовтий колір

glVertex2d(offsetX + sidesize \* 2, offsetY + height); // Правий кут

glVertex2d(offsetX + sidesize + 50, offsetY); // Лівий кут

glVertex2d(offsetX + sidesize \* 2 + 50, offsetY); // Правий кут

glEnd();

glBegin(GL\_TRIANGLES);

glColor3d(1, 0, 0);

glVertex2d(offsetX + sidesize + 50, offsetY); // Лівий кут

glVertex2d(offsetX + sidesize \* 2 + 50, offsetY); // Правий кут

glVertex2d(offsetX + sidesize \* 2, offsetY - height); // Нижній кут

glEnd();

}

}

}

}

# додаток б посилання на репозиторій проєкту

<https://github.com/lashin-d/OpenGL>

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського

«Харківський авіаційний інститут»

Факультет програмної інженерії та бізнесу

Кафедра інженерії програмного забезпечення

Практична робота №3

Minor *«Розробник ігрових додатків»*

дисципліна *«Комп’ютерна графіка з OpenGL»*

(назва дисципліни)

Виконав: студент 3 курсу групи  *535б*

напряму підготовки (спеціальності):

*123 Комп’ютерна інженерія*

(шифр і назва напряму підготовки / спеціальності)

*Лашин Д.О.*

(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. ГРАФІК ФУНКЦІЇ ОДНІЄЇ ЗМІННОЇ

**Мета**: вивчити основні поняття і принципи перетворення координат для

побудови двомірного графіка.

## Завдання, варіант № 12

Використовуючи інструментальні засоби, що вказані викладачем,

розробити програму для побудови графіка функції виду y = f(x) на

довільному інтервалі від Xmin до Xmax і відображення точок перетину функції

з віссю абсцис. Крім того, програма повинна мати такі можливості(табл.3.1):

− дозволяти користувачу задавати інтервал від Xmin до Xmax з

перевіркою Xmin < Xmax;

− виконувати для завданого користувачем інтервалу від Xmin до Xmax

автоматичне масштабування за віссю Y (додатково допускається

наявність ручного режиму встановлення Ymin і Ymax);

− відображати осі координат (та/або координатну сітку) з виводом

значень меж видимої області Xmin, Xmax, Ymin і Ymax, при цьому

система координат повинна бути анізотропною;

− відображати усі точки, де f(x) = 0, якщо вони є на завданому

інтервалі від Xmin до Xmax.

Приклад інтерфейсу користувача наведено на рис. 3.1. Варіанти функції

f1(x) базового рівня складності наведені в табл. 3.2

Для підвищеного рівня складності необхідно додатково реалізувати

коректне виведення функції f2(x) з урахуванням області визначення

функції (варіанти наведені в табл. 3.3) і відобразити лінії розриву (рис. 3.2).

**Методичні вказівки**

Після того, як користувач у режимі діалогу визначив інтервал по осі X,

слід задати кількість точок N, необхідно для побудови графіка функції. Ця

кількість може бути задана користувачем явно чи отримана програмно,

наприклад, відповідати ширині (кількість пікселів) робочої області. На основі

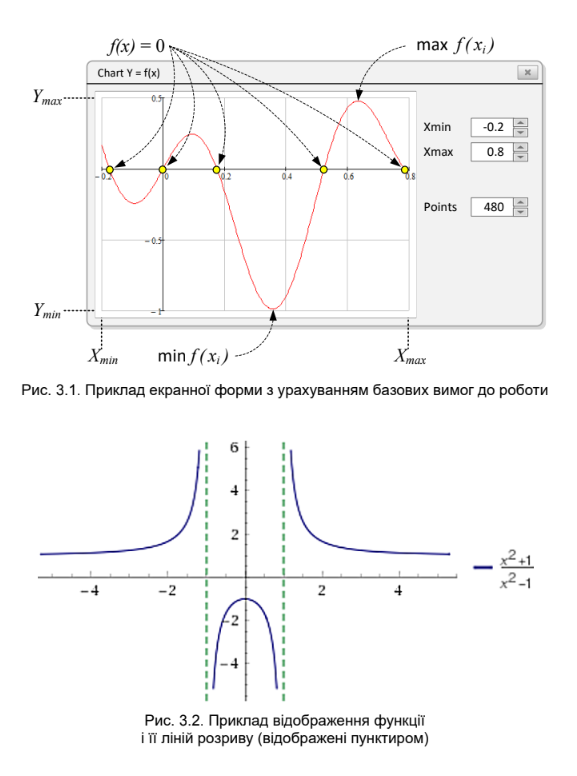
цієї інформації вираховується крок аргументу функції:



і розраховуються координати точок функції в межах завданого

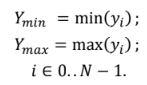
користувачем інтервалу:





Використовуючи ці розрахунки, можна знайти межі робочої області по

вертикалі:



Таким чином, знаючи значення меж інтервалу по осі X і обчисливши

значення меж по осі Y, можна встановити систему координат

(за допомогою команди glOrtho(...)) для виведення графіка y = f(x) на

екран.

Знайти корні функції x0 (точки, де f(x0) = 0) можна на основі наступної

властивості: якщо на інтервалі від xi до xi+1 є перетин з віссю абсцис, то в

результаті примноження відповідних ординат буде виконуватися умова

f(xi)f(xi+1) ≤ 0 (рис. 3.3). В цьому випадку координати точок перетину

функції з віссю X вираховують спрощеним методом половинного ділення:

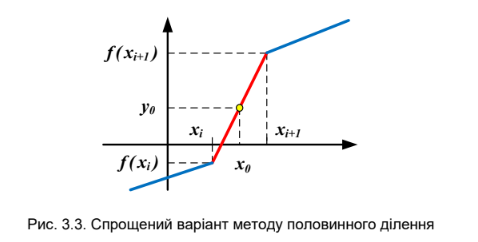
x0 = (xi + xi+1)⁄2, y0 = f(x0). Якщо брати до уваги дискретність екрану

і використовувати для побудови графіка функції кількість точок, що близьке

до значення ширини робочої області (в пікселях) чи більше за неї, то

спрощений метод половинного ділення дозволить отримати рішення, що

візуально не відрізняється від точного.



Алгоритм коректного відображення на екрані функції f2(x), що має

розриви в області визначення, студенти розробляють самостійно.

Додатковий аналіз особливостей свого варіанта функції f2(x) можна

отримати за допомогою наступних електронних ресурсів [1,2].

|  |
| --- |
|  |
| Рисунок 3.4-5 – Варіант 12 |

## Системна інформація

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

Processor AMD Ryzen 7 5800H with Radeon Graphics

RAM 16,0 ГБ

System type 64-розрядна операційна система, процесор з архітектурою x64

Edition Windows 11

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

## Результати виконання практичної роботи

Розв'язання завдання

Для реалізації завдання було розроблено програму, що будує графік функцій  та на заданому інтервалі з можливістю масштабування і ручного налаштування осей. Програма дозволяє користувачу вводити координати Xmin та Xmax, де координати Y обчислюються автоматично

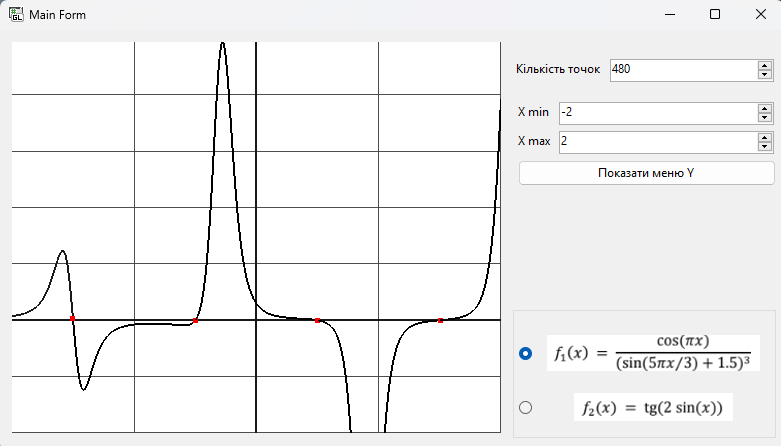


Рисунок 3.6 – Тестування програми після першого запуску

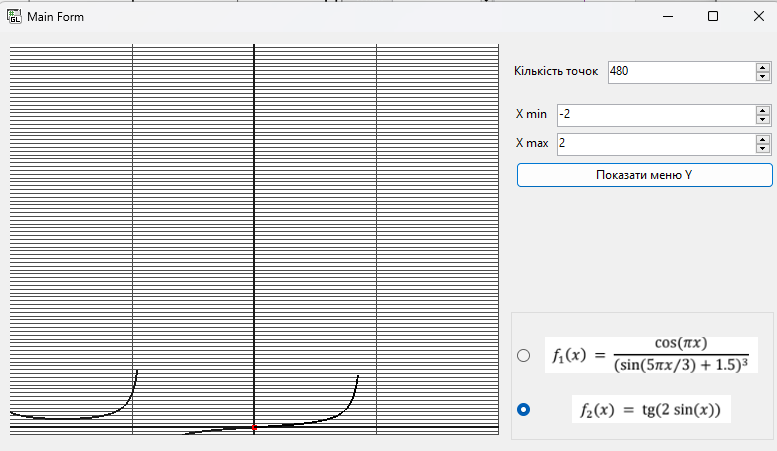


Рисунок 3.7 – Тестування *f2(x)*

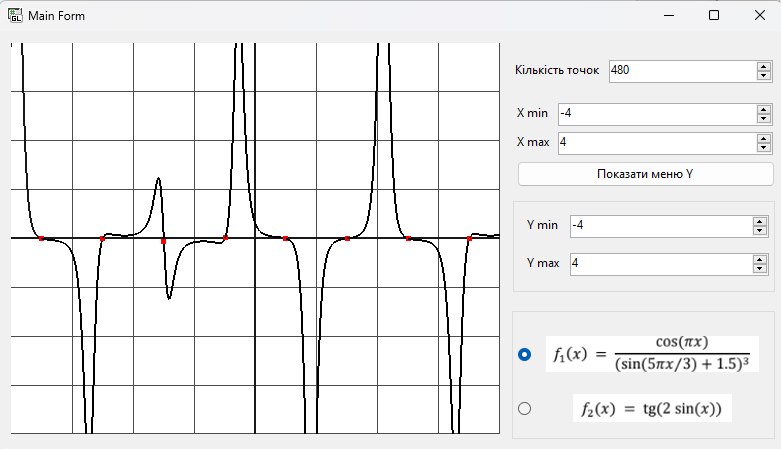
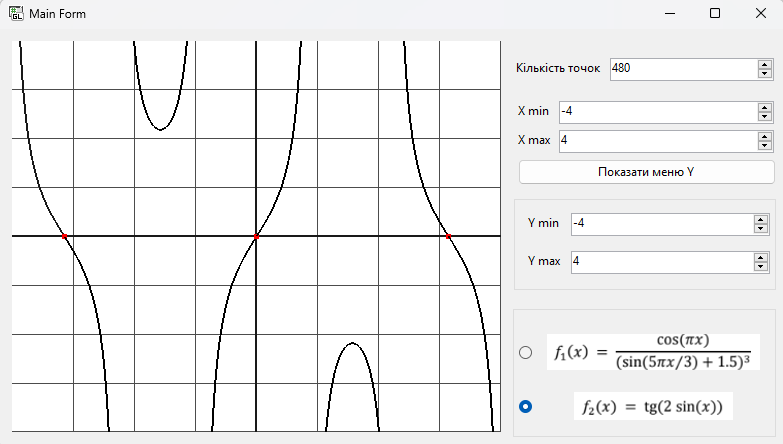
  


Рисунок 2.7-8 – Тестування роботи меню Y

### 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 3.1.

Таблиця 3.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № з/п | Складність | Вимоги | Бали | Зроблено |
| 1 | Базовий рівень | Осі координат і графік функції ***f1(x)***  виводяться на заданому користувачем  інтервалі від Xmin до Xmax і від Ymin до Ymax | 1 | **+** |
| 2 | Автоматичні обчислення Ymin і Ymax на  завданому інтервалі від Xmin до Xmax функції ***f1(x)*** | 2 | **+** |
| 3 | Обчислення і виведення на екран точок  ***f1(x)*** = 0 | 2 | **+** |
| 4 | Підвищений рівень | Коректне виведення графіка ***f2(x)*** (без  хибного виводу точок розриву як точок  перетину з віссю абсцис) і з відображенням ліній розриву функції | 2 | **+** |
| 5 | Використання ООП (наслідування,  використання віртуальних і абстрактних  методів) | 1 | **+** |

# Додаток А. Лістинг програми до практичної роботи №1

### Код файлу (MainForm.cs)

using System.Windows.Forms;

using static Task03.OpenGL;

using System.Drawing;

namespace Task03

{

public partial class MainForm : Form

{

public MainForm()

{

InitializeComponent();

this.MinimumSize = new Size(766, 435);

renderControl1.MinX = (double)MinXvalue.Value;

renderControl1.MaxX = (double)MaxXvalue.Value;

renderControl1.MinY = (double)MinYvalue.Value;

renderControl1.MaxY = (double)MaxYvalue.Value;

renderControl1.showY = false;

}

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

{

renderControl1.points = (double)NumberOfPoints.Value;

renderControl1.Invalidate();

}

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

{

renderControl1.MinX = (double)MinXvalue.Value;

renderControl1.Invalidate();

}

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

{

renderControl1.MaxX = (double)MaxXvalue.Value;

renderControl1.Invalidate();

}

private void buttonShowY\_Click(object sender, System.EventArgs e)

{

if (!renderControl1.showY)

{

MenuY.Visible = true;

renderControl1.MinY = (double)MinYvalue.Value;

renderControl1.MaxY = (double)MaxYvalue.Value;

}

else

{

MenuY.Visible = false;

renderControl1.MinY = -2.0;

renderControl1.MaxY = 2.0;

}

renderControl1.showY = !renderControl1.showY;

renderControl1.Invalidate();

}

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

{

renderControl1.MinY = (double)MinYvalue.Value;

renderControl1.Invalidate();

}

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

{

renderControl1.MaxY = (double)MaxYvalue.Value;

renderControl1.Invalidate();

}

private void Function1\_CheckedChanged(object sender, System.EventArgs e)

{

renderControl1.function = 0;

renderControl1.Invalidate();

}

private void Function2\_CheckedChanged(object sender, System.EventArgs e)

{

renderControl1.function = 1;

renderControl1.Invalidate();

}

}

}

### 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 Task03.RenderControl;

namespace Task03

{

public partial class RenderControl : OpenGL

{

Class1 \_l = new Class1();

Functions \_dF = new Functions();

public double points = 1000;

public bool showY { get; set; } = false;

public double MinX { get; set; } = -2.0;

public double MaxX { get; set; } = 2.0;

public double MinY { get; set; } = -2.0;

public double MaxY { get; set; } = 2.0;

public int function { get; set; } = 0;

public RenderControl()

{

InitializeComponent();

}

private void OnRender(object sender, EventArgs e)

{

glClearColor(1, 1, 1, 1);

glClear(GL\_COLOR\_BUFFER\_BIT);

glLoadIdentity();

glViewport(0, 0, Width, Height);

if (!showY)

(MinY, MaxY) = \_dF.ShowY(MinX, MaxX, MinY, MaxY, points, function);

MaxY = Math.Min(MaxY, 100);

MinY = Math.Max(MinX, -100);

gluOrtho2D(MinX, MaxX, MinY, MaxY);

\_l.DrawGrid(MinX, MaxX, MinY, MaxY);

\_l.DrawAxis(MinX, MaxX, MinY, MaxY);

\_dF.Function(MinX, MaxX, MinY, MaxY, points, function);

}

}

}

***Код файлу (Class1.cs)***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Task03

{

public partial class RenderControl

{

public class Class1

{

public void DrawGrid(double MinX, double MaxX, double MinY, double MaxY)

{

glLineWidth(0.5f);

glColor3d(0.3, 0.3, 0.3);

glBegin(GL\_LINES);

for (double i = MinX; i <= MaxX; i++)

{

glVertex2d(i, MinY);

glVertex2d(i, MaxY);

}

for (double i = MinY; i <= MaxY; i++)

{

glVertex2d(MinX, i);

glVertex2d(MaxX, i);

}

glEnd();

}

public void DrawAxis(double MinX, double MaxX, double MinY, double MaxY)

{

glLineWidth(2);

glColor3d(0.1, 0.1, 0.1);

glBegin(GL\_LINES);

glVertex2d(MinX, 0);

glVertex2d(MaxX, 0);

glVertex2d(0, MinY);

glVertex2d(0, MaxY);

glEnd();

}

}

}

}

***Код файлу (Functions.cs)***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Task03

{

public partial class RenderControl

{

public class Functions

{

public void Function(double MinX, double MaxX, double MinY, double MaxY, double points, int function)

{

glLineWidth(2);

glBegin(GL\_LINE\_STRIP);

double previousY = double.NaN;

double x = MinX;

double step = (MaxX - MinX) / (points - 1);

double y = CalculateFunction(x, function);

glVertex2d(x, y);

for (int i = 1; i < points; i++)

{

previousY = y;

x = MinX + i \* step;

y = CalculateFunction(x, function);

if (!double.IsNaN(previousY) && Math.Abs(y - previousY) > 1.0)

{

glEnd();

glBegin(GL\_LINE\_STRIP);

continue;

}

glVertex2d(x, y);

if ((previousY \* y) <= 0 && previousY != 0)

{

glEnd();

DrawPointsOnX(previousY, x, step, y);

glBegin(GL\_LINE\_STRIP);

glColor3d(0, 0, 0);

glVertex2d(x, y);

}

}

glEnd();

}

public (double, double) ShowY(double MinX, double MaxX, double MinY, double MaxY, double points, int function)

{

double previousY = double.NaN;

double x = MinX;

double step = (MaxX - MinX) / (points - 1);

double y = CalculateFunction(x, function);

double min;

double max;

min = max = y;

for (int i = 1; i < points; i++)

{

previousY = y;

x = MinX + i \* step;

y = CalculateFunction(x, function);

if (y > max)

max = y;

if (y < min)

min = y;

}

return (min, max);

}

public double CalculateFunction(double x, int function)

{

switch (function)

{

case 0:

return Math.Cos(Math.PI \* x) / Math.Pow((Math.Sin(5 \* Math.PI \* x / 3) + 1.5), 3);

case 1:

return Math.Tan(2 \* Math.Sin(x));

default:

return 0;

}

}

private void DrawPointsOnX(double previousY, double x, double step, double y)

{

glPointSize(5);

glColor3d(1, 0, 0);

glBegin(GL\_POINTS);

glVertex2d(x - step / 2, (previousY + y) / 2);

glEnd();

}

}

}

}

# додаток б посилання на репозиторій проєкту

<https://github.com/lashin-d/OpenGL>

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського

«Харківський авіаційний інститут»

Факультет програмної інженерії та бізнесу

Кафедра інженерії програмного забезпечення

Практична робота 4

Minor *«Розробник ігрових додатків»*

дисципліна *«Комп’ютерна графіка з OpenGL»*

(назва дисципліни)

Виконав: студент 3 курсу групи  *535б*

напряму підготовки (спеціальності):

*123 Комп’ютерна інженерія*

(шифр і назва напряму підготовки / спеціальності)

*Лашин Д.О.*

(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. ГРАФІК ФУНКЦІЇ ОДНІЄЇ ЗМІННОЇ

**Мета**: вивчити математичні методи та засоби для реалізації графічних

примітивів.

## Завдання, варіант № 12

Використовуючи інструментальні засоби, вказані викладачем,

розробити програму для виведення кривих другого порядку на екран

(у вікно Windows) за допомогою відрізків. Систему оцінювання наведено в

табл. 4.1, а варіанти завдань – в табл. 4.2. Для кривих, які у варіанті відмічені

«++», знайти та вивести на екран точки перетину, якщо такі є, з довільним

відрізком, координати якого задає користувач.

**Методичні вказівки**

Кожну криву другого порядку можна подати як послідовність відрізків. У

цьому випадку перетин кривої другого порядку та довільного відрізка можна

розглядати як пошук спільної точки [x0, y0] двох відрізків[x1, y1], [x2, y2] и

[x3, y3], [x4, y4], заданих у параметричному вигляді (один з яких є

фрагментом кривої). Цю задачу можна подати у вигляді системи, що

складається з двох лінійних рівнянь з невідомими параметрами t1першого і

t2 другого відрізків:



І за умови, що результат рішення буде задовольняти наступні дві умови:

0 ≤ t1 ≤ 1 і 0 ≤ t2 ≤ 1, інакше відрізки або паралельні, або перетинаються

тільки прямі, на яких вони лежать [1, 2].

|  |
| --- |
|  |
| Рисунок 4.1 – Варіант 12 |

## Системна інформація

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

Processor AMD Ryzen 7 5800H with Radeon Graphics

RAM 16,0 ГБ

System type 64-розрядна операційна система, процесор з архітектурою x64

Edition Windows 11

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

## Результати виконання практичної роботи

Розв'язання завдання

Для реалізації завдання було розроблено програму, що будує фігури окружності та гиперболи, які були зроблені завдяки кривим другого порядку та представлені в явної та параметричній формі.

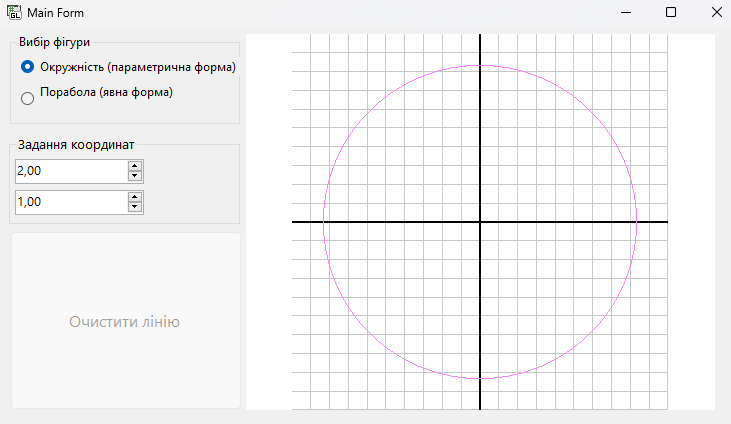


Рисунок 4.1 – Тестування програми після першого запуску

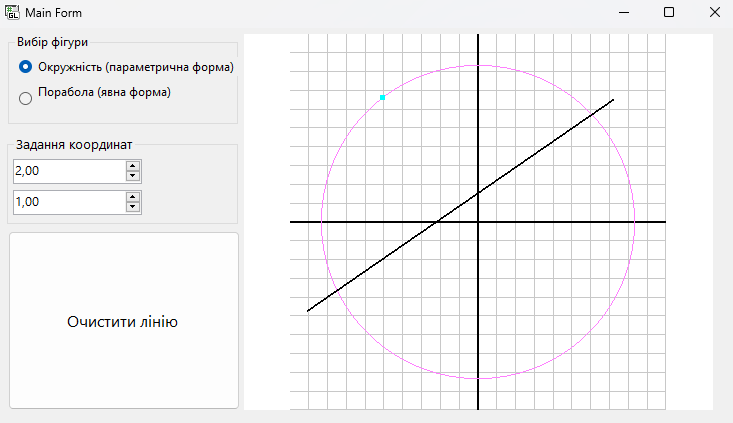


Рисунок 4.2 – Тестування окружності

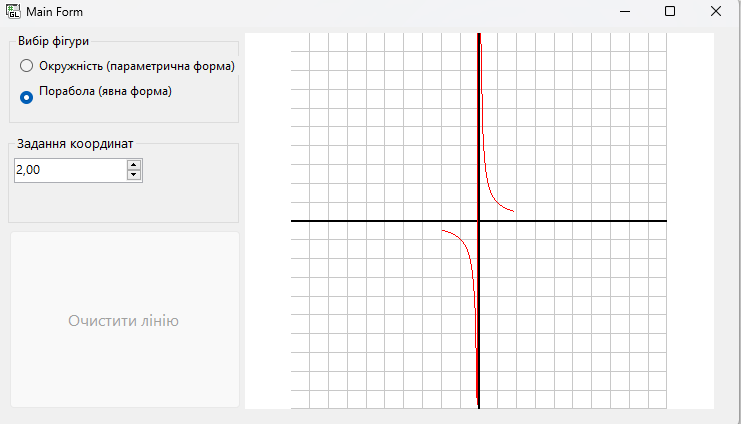


Рисунок 4.2 – Тестування зображення параболи

### 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 4.1.

Таблиця 4.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № з/п | Складність | Вимоги | Бали | Зроблено |
| 1 | Базовий рівень | Установлення ізотропної системи координат для вікна з змінюваними розмірами | 1 | **+** |
| 2 | Виведення кривих другого порядку  відповідно до варіанту завдання | 2 | **+** |
| 3 | Виведення відрізка та обчислення його точок перетину з кривою другого порядку відповідно до варіанту | 2 | **+** |
| 4 | Підвищений рівень | Вказання положення точок, що формують відрізок, в області графічного виведення програми за допомогою маніпулятора «миш» | 2 | **+** |
| 5 | Використання ООП | 1 | **+** |

# Додаток А. Лістинг програми до практичної роботи №1

### Код файлу (MainForm.cs)

using System;

using System.Windows.Forms;

using static Task04.OpenGL;

namespace Task04

{

public partial class MainForm : Form

{

public MainForm()

{

InitializeComponent();

}

private void ClearLinec(object sender, System.EventArgs e)

{

renderControl1.LineExist = false;

renderControl1.StartX = renderControl1.EndX = renderControl1.StartY = renderControl1.EndY = 0;

ClearLine.Enabled = false;

renderControl1.Invalidate();

}

private void ChosedCircle(object sender, System.EventArgs e)

{

renderControl1.figur = false;

Coordinat2.Visible = true;

renderControl1.Invalidate();

}

private void ChosedPorabola(object sender, System.EventArgs e)

{

renderControl1.figur = true;

Coordinat2.Visible = false;

renderControl1.Invalidate();

}

private void Mouse\_Click(object sender, MouseEventArgs e)

{

if (e.Button == MouseButtons.Left)

{

double difference = Math.Max(renderControl1.ClientRectangle.Width, renderControl1.ClientRectangle.Height) - Math.Min(renderControl1.ClientRectangle.Width, renderControl1.ClientRectangle.Height);

if (difference != 0)

{

renderControl1.LineExist = renderControl1.ClientRectangle.Width > renderControl1.ClientRectangle.Height ?

difference / 2 < e.X && e.X < Math.Min(renderControl1.ClientRectangle.Width, renderControl1.ClientRectangle.Height) + difference / 2 :

difference / 2 < renderControl1.ClientRectangle.Height - e.Y && renderControl1.ClientRectangle.Height - e.Y < Math.Min(renderControl1.ClientRectangle.Width, renderControl1.ClientRectangle.Height) + difference / 2;

if (renderControl1.LineExist)

{

renderControl1.IsUp = false;

double xLength = renderControl1.ClientRectangle.Width > renderControl1.ClientRectangle.Height ?

renderControl1.ClientRectangle.Width / (double)renderControl1.ClientRectangle.Height :

1;

double yLength = renderControl1.ClientRectangle.Height > renderControl1.ClientRectangle.Width ?

renderControl1.ClientRectangle.Height / (double)renderControl1.ClientRectangle.Width :

1;

renderControl1.StartX = renderControl1.EndX = (renderControl1.Length \* xLength) \* ((e.X - (renderControl1.ClientRectangle.Width / 2.0)) / (double)(renderControl1.ClientRectangle.Width / 2.0));

renderControl1.StartY = renderControl1.EndY = (renderControl1.Length \* yLength) \* (((renderControl1.ClientRectangle.Height / 2.0) - e.Y) / (double)(renderControl1.ClientRectangle.Height / 2.0));

}

else

{

ClearLine.Enabled = false;

}

}

else

{

renderControl1.IsUp = false;

renderControl1.LineExist = true;

double xLength = renderControl1.ClientRectangle.Width > renderControl1.ClientRectangle.Height ?

renderControl1.ClientRectangle.Width / (double)renderControl1.ClientRectangle.Height :

1;

double yLength = renderControl1.ClientRectangle.Height > renderControl1.ClientRectangle.Width ?

renderControl1.ClientRectangle.Height / (double)renderControl1.ClientRectangle.Width :

1;

renderControl1.StartX = renderControl1.EndX = (renderControl1.Length \* xLength) \* ((e.X - (renderControl1.ClientRectangle.Width / 2.0)) / (double)(renderControl1.ClientRectangle.Width / 2.0));

renderControl1.StartY = renderControl1.EndY = (renderControl1.Length \* yLength) \* (((renderControl1.ClientRectangle.Height / 2.0) - e.Y) / (double)(renderControl1.ClientRectangle.Height / 2.0));

}

}

renderControl1.Invalidate();

}

private void oCoordinat1(object sender, EventArgs e)

{

if (Coordinat1.Value == Coordinat2.Value)

{

Coordinat2.Value = (double)Coordinat1.Value > renderControl1.A ? (decimal)((double)Coordinat2.Value - 0.1) : (decimal)((double)Coordinat2.Value + 0.1);

}

renderControl1.A = (double)Coordinat1.Value;

renderControl1.Invalidate();

}

private void oCoordinat2(object sender, EventArgs e)

{

if (Coordinat1.Value == Coordinat2.Value)

{

Coordinat1.Value = (double)Coordinat2.Value > renderControl1.B ? (decimal)((double)Coordinat1.Value - 0.1) : (decimal)((double)Coordinat1.Value + 0.1);

}

renderControl1.B = (double)Coordinat2.Value;

renderControl1.Invalidate();

}

private void ReleaseTheMouse(object sender, MouseEventArgs e)

{

if (e.Button == MouseButtons.Left && renderControl1.LineExist)

{

double xLength = renderControl1.ClientRectangle.Width > renderControl1.ClientRectangle.Height ?

renderControl1.ClientRectangle.Width / (double)renderControl1.ClientRectangle.Height :

1;

double yLength = renderControl1.ClientRectangle.Height > renderControl1.ClientRectangle.Width ?

renderControl1.ClientRectangle.Height / (double)renderControl1.ClientRectangle.Width :

1;

renderControl1.EndX = (renderControl1.Length \* xLength) \* ((e.X - (renderControl1.ClientRectangle.Width / 2.0)) / (double)(renderControl1.ClientRectangle.Width / 2.0));

renderControl1.EndY = (renderControl1.Length \* yLength) \* (((renderControl1.ClientRectangle.Height / 2.0) - e.Y) / (double)(renderControl1.ClientRectangle.Height / 2.0));

renderControl1.IsUp = true;

if (renderControl1.EndX == renderControl1.StartX && renderControl1.EndY == renderControl1.StartY)

{

renderControl1.LineExist = false;

ClearLine.Enabled = false;

}

else

{

ClearLine.Enabled = true;

}

}

renderControl1.Invalidate();

}

private void Mouse\_Movement(object sender, MouseEventArgs e)

{

if (renderControl1.LineExist)

{

if (!renderControl1.IsUp)

{

double xLength = renderControl1.ClientRectangle.Width > renderControl1.ClientRectangle.Height ?

renderControl1.ClientRectangle.Width / (double)renderControl1.ClientRectangle.Height :

1;

double yLength = renderControl1.ClientRectangle.Height > renderControl1.ClientRectangle.Width ?

renderControl1.ClientRectangle.Height / (double)renderControl1.ClientRectangle.Width :

1;

renderControl1.EndX = (renderControl1.Length \* xLength) \* ((e.X - (renderControl1.ClientRectangle.Width / 2.0)) / (double)(renderControl1.ClientRectangle.Width / 2.0));

renderControl1.EndY = (renderControl1.Length \* yLength) \* (((renderControl1.ClientRectangle.Height / 2.0) - e.Y) / (double)(renderControl1.ClientRectangle.Height / 2.0));

renderControl1.Invalidate();

}

}

}

}

}

### 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 System.Security.Policy;

namespace Task04

{

public partial class RenderControl : OpenGL

{

Figures draw;

public double StartX { get; set; }

public double EndX { get; set; }

public double StartY { get; set; }

public double EndY { get; set; }

public double Length { get; set; } = 1.5;

public bool figur { get; set; } = false;

public bool IsUp { get; set; } = true;

public bool LineExist { get; set; } = false;

public double A { get; set; } = 2.0;

public double B { get; set; } = 1.0;

public RenderControl()

{

InitializeComponent();

}

private void OnRender(object sender, EventArgs e)

{

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

glClear(GL\_COLOR\_BUFFER\_BIT);

glLoadIdentity();

int size = Math.Min(Width, Height);

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

Length = draw.FindMax(A, B, figur);

gluOrtho2D(-Length, Length, -Length, Length);

draw.DrawGrid(-Length, Length, -Length, Length);

draw.DrawCoordinateGrid(-Length, Length, -Length, Length);

if (!figur)

draw.Circle(A);

else

draw.Hyperbola(A);

if (LineExist)

{

CreateLine();

draw.SearchPoint(A, B, StartX, StartY, EndX, EndY, figur);

}

}

private void CreateLine()

{

if (!IsUp)

{

draw.DrawLineWait(StartX, EndX, StartY, EndY);

}

else

{

draw.DrawLine(StartX, EndX, StartY, EndY);

}

}

private void Created(object sender, EventArgs e)

{

draw = new Figures();

}

}

}

***Код файлу (Figures.cs)***

using System;

using System.Collections.Generic;

using System.Drawing.Printing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

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

namespace Task04

{

public partial class RenderControl

{

public class Figures

{

public void SearchPoint(double a, double b, double StartX, double EndX, double StartY, double EndY, bool isChoosed)

{

glPointSize(5.0f);

glColor3f(0.0f, 1.0f, 1.0f);

glBegin(GL\_POINTS);

double dx = EndX - StartX;

double dy = EndY - StartY;

if (!isChoosed)

{

double A = dx \* dx + dy \* dy;

double B = 2 \* (StartX \* dx + StartY \* dy);

double C = StartX \* StartX + StartY \* StartY - a \* a;

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

if (discriminant >= 0)

{

double sqrtD = Math.Sqrt(discriminant);

double t1 = (-B + sqrtD) / (2 \* A);

double t2 = (-B - sqrtD) / (2 \* A);

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

{

double x1 = StartX + t1 \* dx;

double y1 = StartY + t1 \* dy;

glVertex2d(x1, y1);

}

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

{

double x2 = StartX + t2 \* dx;

double y2 = StartY + t2 \* dy;

glVertex2d(x2, y2);

}

}

}

else

{

double A = a \* dx \* dx;

double B = 2 \* a \* dx \* StartX - dy;

double C = a \* StartX \* StartX - StartY;

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

if (discriminant >= 0)

{

double sqrtD = Math.Sqrt(discriminant);

double t1 = (-B + sqrtD) / (2 \* A);

double t2 = (-B - sqrtD) / (2 \* A);

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

{

double x1 = StartX + t1 \* dx;

double y1 = StartY + t1 \* dy;

glVertex2d(x1, y1);

}

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

{

double x2 = StartX + t2 \* dx;

double y2 = StartY + t2 \* dy;

glVertex2d(x2, y2);

}

}

}

glEnd();

}

public void Circle(double radius)

{

glColor3f(1.0f, 0.5f, 1.0f);

glBegin(GL\_LINE\_LOOP);

int segments = 100;

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

{

double theta = 2.0 \* Math.PI \* i / segments; // Кут в радіанах

double x = radius \* Math.Cos(theta);

double y = radius \* Math.Sin(theta);

glVertex2d(x, y);

}

glEnd();

}

public void Hyperbola(double a)

{

glColor3f(1.0f, 0.0f, 0.0f);

glBegin(GL\_LINE\_STRIP);

double step = 0.1; // Крок для x

for (double x = -2.0; x <= 2.0; x += step)

{

if (x == 0) continue; // Уникнення розриву в 0

double y = a / x;

glVertex2d(x, y);

}

glEnd();

}

public double FindMax(double a, double b, bool isChoosed)

{

double maxX, maxY;

if (!isChoosed)

{

maxX = Math.Abs(a);

maxY = Math.Abs(a);

}

else

{

maxX = 2.0;

maxY = Math.Abs(a / 0.1); // Максимальне значення гіперболи

}

return Math.Max(maxX, maxY) + 0.2;

}

public void DrawGrid(double minX, double maxX, double minY, double maxY)

{

glLineWidth(1.0f);

glColor3ub(200, 200, 200);

glBegin(GL\_LINES);

for (double i = minX; i <= maxX; i += maxX / 10)

{

glVertex2d(i, minY);

glVertex2d(i, maxY);

}

for (double j = minY; j <= maxY; j += maxY / 10)

{

glVertex2d(minX, j);

glVertex2d(maxX, j);

}

glEnd();

}

public void DrawCoordinateGrid(double minX, double maxX, double YMin, double YMax)

{

glLineWidth(2.0f);

glBegin(GL\_LINES);

glColor3ub(0, 0, 0);

glVertex2d(minX, 0);

glVertex2d(maxX, 0);

glVertex2d(0, YMin);

glVertex2d(0, YMax);

glEnd();

glLineWidth(1.0f);

}

public void DrawLineWait(double minX, double maxX, double YMin, double YMax)

{

glColor3f(0.5f, 0.5f, 0.5f);

glLineWidth(1.4f);

glPointSize(3.0f);

glBegin(GL\_POINTS);

glVertex2d(minX, YMin);

glVertex2d(maxX, YMax);

glEnd();

glBegin(GL\_LINES);

glVertex2d(minX, YMin);

glVertex2d(maxX, YMax);

glEnd();

}

public void DrawLine(double minX, double maxX, double YMin, double YMax)

{

glColor3f(0.0f, 0.0f, 0.0f);

glLineWidth(1.7f);

glBegin(GL\_LINES);

glVertex2d(minX, YMin);

glVertex2d(maxX, YMax);

glEnd();

}

}

}

}

# додаток б посилання на репозиторій проєкту

<https://github.com/lashin-d/OpenGL>

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського

«Харківський авіаційний інститут»

Факультет програмної інженерії та бізнесу

Кафедра інженерії програмного забезпечення

Практична робота 5

Minor *«Розробник ігрових додатків»*

дисципліна *«Комп’ютерна графіка з OpenGL»*

(назва дисципліни)

Виконав: студент 3 курсу групи  *535б*

напряму підготовки (спеціальності):

*123 Комп’ютерна інженерія*

(шифр і назва напряму підготовки / спеціальності)

*Лашин Д.О.*

(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 5. ГРАФІК ФУНКЦІЇ ОДНІЄЇ ЗМІННОЇ

**Мета**: навчитися працювати з тривимірними графічними примітивами

OpenGL та застосовувати афінні перетворення для розміщення об’єктів у

просторі..

## Завдання, варіант № 12

Використовуючи інструментальні засоби, вказані викладачем,

розробити програму з використання засобів OpenGL, яка встановлює

ізотропну систему координат, створює і виводить зображення тривимірної

сцени з такими елементами (систему оцінки наведено в табл. 5.1, а варіанти

завдань – в табл.5.2):

− осі координат з нулем у центрі екрана та вказанням осі та додатного

напрямку;

− координатна сітка (grid) в одній з площин (X0Y, X0Z чи Y0Z);

− три квадратичні фігури – gluDisk / gluPartialDisk, gluSphere,

gluCylinder в режимі відображення каркаса і з спрощеною моделлю

освітлення glEnable(GL\_COLOR\_MATERIAL) для базового рівня

складності;

− площина відтину для однієї з фігур (сфера, циліндр чи конус);

− повноцінна модель освітлення та/або текстурами для реалізації

завдання з підвищеною складністю.

Мінімальний інтерфейс користувача повинен забезпечувати можливості

повороту сцени відносно осей OX і OY за допомогою маніпулятора «миш» і

керування параметрами площини відтину [1, 2]. Параметри деталізації

об’єктів (slices, stacks), кольору, товщини і типу ліній обирають самостійно.

Приклад сцени показано на рис. 5.1.

**Методичні вказівки**

Ізотропну систему координат можна встановлювати двома способами.

В першому випадку за допомогою команди glViewport задають робочу

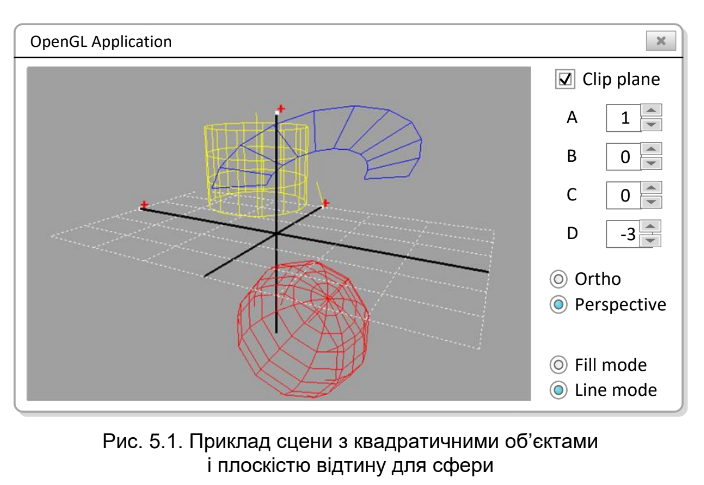
область відповідного меншого значення ширини/висоти вікна, а у другому –

вводять множник (або дільник), що коректує і рівний відношенню ширини і

висоти вікна під час встановлення системи координат (наприклад,командою glOrtho). Глибину потрібно задавати таким чином, щоб за будь-

якого розміщення заданої сцени усі об’єкти знаходилися всередині видимої

області.



Для відображення квадратичних примітивів використовують точку

прив’язки об’єкта. В загальному випадку за відсутності афінних перетворень

ця точка прив’язки об’єкта збігається з початком системи координат. Точкою

прив’язки для сфери і диска (повного і часткового) виступає їхній центр, а

для циліндра (конуса) – центр однієї з основ. Для розміщення кожного з

трьох квадратичних об’єктів відповідно варіанту завдання необхідно

скористатися одним з афінних перетворень або їхніх комбінацій: поворот,

перенос, масштабування. Після виконання перетворень точка прив’язки

повинна знаходитися на координатах x0, y0, z0, а вісь об’єкта паралельна (∥)

для сфери/диска або колінеарна (⇈, ⇅) для циліндра / конуса з урахуванням

варіанта. Під час формування зображення квадратичних примітивів

використовують такі параметри:

x0, y0, z0 – координати точки прив’язки фігури;

R – радіус сфери або радіус основи циліндра/конуса з

центром у точці прив’язки або зовнішній радіус диска;

r – радіус другої основи циліндра / усіченого конуса;

внутрішній радіус диска;

h – висота циліндра / конуса / усіченого конуса;

∠start – початковий кут частинного диска;

∠sweep – кут розгортки для частинного диска;

ось ∥,⇈,⇅ – паралельність / колінеарність осей координат і фігури.

|  |
| --- |
|  |
|  |
| Рисунок 5.2 – Варіант 12 |

## Системна інформація

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

Processor AMD Ryzen 7 5800H with Radeon Graphics

RAM 16,0 ГБ

System type 64-розрядна операційна система, процесор з архітектурою x64

Edition Windows 11

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

## Результати виконання практичної роботи

Розв'язання завдання

Для реалізації завдання було розроблено програму, що встановлює ізотропну систему координат, створює і виводить зображення тривимірної сцени з осями координат, координатною сіткою та фігурами

Зображення програми з освітленням

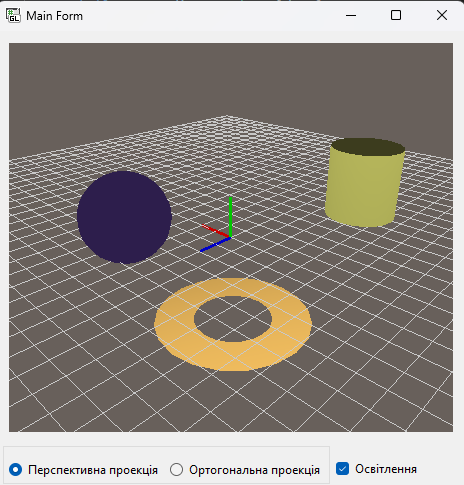


Рисунок 5.2 – Тестування програми після першого запуску

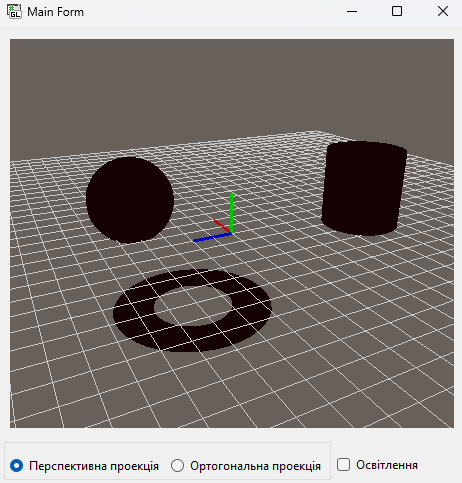


Рисунок 5.3 – Тестування без освітлення

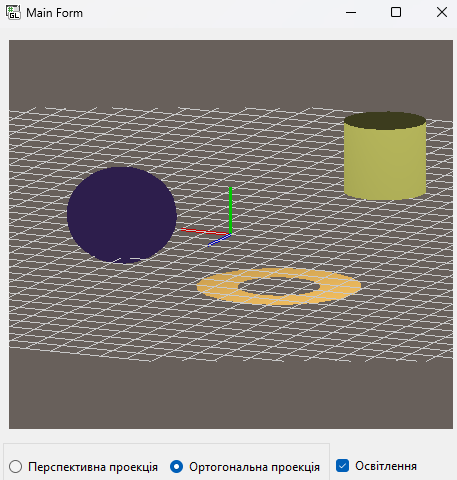


Рисунок 5.4 – Тестування зображення ортогональної проекції

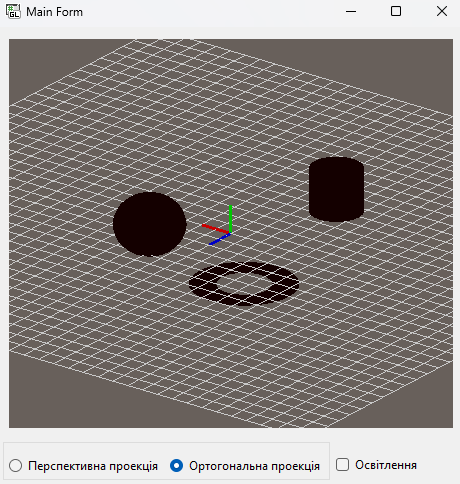


Рисунок 5.5 – Тестування зображення ортогональної проекції без освітлення

### 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 5.1.

Таблиця 5.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № з/п | Складність | Вимоги | Бали | Зроблено |
| 1 | Базовий рівень | Коректне (ізотропне) відображення завдання (під час зміни розмірів вікна) у ортографічній проекції | 1 | **+** |
| 2 | Під час запуску застосунку відображаються осі 0X, 0Y, 0Z, координатна сітка і каркас квадратичних об’єктів | 1 | **+** |
| 3 | Інтерфейс керування параметрами площини відтину | 1 | **+** |
| 4 | Використання джерел світла для освітлення об’єктів сцени сумісно з командою glColorMaterial | 1 | **+** |
| 5 | Використання списків відображення (Display Lists) | 1 | **+** |
| 6 | Підвищений рівень | Створення зображення сцени в перспективній проекції | 1 | **+** |
| 7 | Накладення текстури на поверхню завданих у варіанті фігур | 1 | **-** |
| 8 | Застосування команди glMaterial для налаштування параметрів відбиття поверхонь об’єктів сцени | 1 | **+** |

# Додаток А. Лістинг програми до практичної роботи №1

### Код файлу (MainForm.cs)

using System;

using System.Windows.Forms;

using static Task05.OpenGL;

namespace Task05

{

public partial class MainForm : Form

{

public MainForm()

{

InitializeComponent();

}

private void OnWheel(object sender, MouseEventArgs e)

{

if (e.Delta < 0)

{

renderControl1.Multi += 0.1;

}

else

{

if (renderControl1.Multi > 0.2)

{

renderControl1.Multi -= 0.1;

}

}

renderControl1.Invalidate();

}

private void MouseDown(object sender, MouseEventArgs e)

{

if (renderControl1.NotRotating && e.Button == MouseButtons.Left)

{

renderControl1.NotRotating = false;

renderControl1.LastLeftMouseX = e.X;

renderControl1.LastLeftMouseY = e.Y;

}

if (renderControl1.NotPanning && e.Button == MouseButtons.Right)

{

renderControl1.NotPanning = false;

renderControl1.LastRightMouseX = e.X;

renderControl1.LastRightMouseY = e.Y;

}

}

private void MouseMovement(object sender, MouseEventArgs e)

{

if (renderControl1.Mode)

{

if (!renderControl1.NotRotating)

{

double deltaX = e.X - renderControl1.LastLeftMouseX;

double deltaY = e.Y - renderControl1.LastLeftMouseY;

renderControl1.CameraTheta += deltaX \* 0.01;

renderControl1.CameraPhi -= deltaY \* 0.01;

renderControl1.CameraPhi = Math.Clamp(renderControl1.CameraPhi, 0.1, Math.PI - 0.1);

renderControl1.LastLeftMouseX = e.X;

renderControl1.LastLeftMouseY = e.Y;

renderControl1.Invalidate();

}

}

else

{

if (!renderControl1.NotRotating)

{

double deltaX = e.X - renderControl1.LastLeftMouseX;

double deltaY = e.Y - renderControl1.LastLeftMouseY;

renderControl1.AngleX += deltaX \* 0.2;

renderControl1.AngleY += deltaY \* 0.2;

renderControl1.LastLeftMouseX = e.X;

renderControl1.LastLeftMouseY = e.Y;

renderControl1.Invalidate();

}

}

if (!renderControl1.NotPanning)

{

double deltaX = e.X - renderControl1.LastRightMouseX;

double deltaY = e.Y - renderControl1.LastRightMouseY;

renderControl1.PanOffsetX += deltaX \* 0.01;

renderControl1.PanOffsetY -= deltaY \* 0.01;

renderControl1.LastRightMouseX = e.X;

renderControl1.LastRightMouseY = e.Y;

renderControl1.Invalidate();

}

}

private void MouseClick(object sender, MouseEventArgs e)

{

if (!renderControl1.NotRotating && e.Button == MouseButtons.Left)

{

renderControl1.NotRotating = true;

}

if (!renderControl1.NotPanning && e.Button == MouseButtons.Right)

{

renderControl1.NotPanning = true;

}

}

private void OrthogonalSystem(object sender, EventArgs e)

{

if (OrthogonalSys.Checked)

{

renderControl1.Mode = false;

renderControl1.Invalidate();

}

}

private void PerspectiveSystem(object sender, EventArgs e)

{

if (PerspectiveSys.Checked)

{

renderControl1.Mode = true;

renderControl1.Invalidate();

}

}

private void Light(object sender, EventArgs e)

{

renderControl1.LightOn = Lighting.Checked;

renderControl1.Invalidate();

}

}

}

### 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 Task05

{

public partial class RenderControl : OpenGL

{

public double CenterX { get; set; } = 0.0;

public double CenterY { get; set; } = 0.0;

public double CenterZ { get; set; } = 0.0;

public double PanOffsetX { get; set; } = 0.0;

public double PanOffsetY { get; set; } = 0.0;

public double PanOffsetZ { get; set; } = 0.0;

public double Multi { get; set; } = 1.0;

public bool Mode { get; set; } = true;

public bool LightOn { get; set; } = true;

public bool NotRotating { get; set; } = true;

public bool NotPanning { get; set; } = true;

public bool Fill { get; set; } = true;

public bool IsClipPlane { get; set; } = true;

public double LastRightMouseX { get; set; } = 0.0;

public double LastRightMouseY { get; set; } = 0.0;

public double LastLeftMouseX { get; set; } = 0.0;

public double LastLeftMouseY { get; set; } = 0.0;

public double CameraRadius { get; set; } = 20.0;

public double CameraTheta { get; set; } = 0.0;

public double CameraPhi { get; set; } = Math.PI / 2;

public double AngleX { get; set; } = 0.0;

public double AngleY { get; set; } = 0.0;

Figures draw;

private uint coordinatesDisplayList;

private uint gridDisplayList;

private uint sphereDisplayList;

private uint cylinderDisplayList;

private uint discDisplayList;

public RenderControl()

{

InitializeComponent();

}

private void Rendering(object sender, EventArgs e)

{

glClearColor(0.41f, 0.38f, 0.36f, 1.0f);

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glEnable(GL\_DEPTH\_TEST);

glViewport(0, 0, Width, Height);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

double size = 20;

double aspect = (double)Width / Height;

if (Mode)

{

gluPerspective(45.0, aspect, 0.1, 100.0);

}

else

{

glOrtho(-size \* Multi, size \* Multi, -size \* Multi, size \* Multi, -size \* 2.5 \* Multi, size \* 2.5 \* Multi);

}

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glTranslated(PanOffsetX, PanOffsetY, 0.0);

if (Mode)

{

double cameraX = CameraRadius \* Math.Sin(CameraPhi) \* Math.Cos(CameraTheta);

double cameraY = CameraRadius \* Math.Cos(CameraPhi);

double cameraZ = CameraRadius \* Math.Sin(CameraPhi) \* Math.Sin(CameraTheta);

gluLookAt(

Multi \* cameraX, Multi \* cameraY, Multi \* cameraZ,

CenterX, CenterY, CenterZ,

0.0, 1.0, 0.0

);

}

else

{

glRotatef((float)AngleY, 1.0f, 0.0f, 0.0f);

glRotatef((float)AngleX, 0.0f, 1.0f, 0.0f);

}

glCallList(coordinatesDisplayList);

if (LightOn)

InitializeLighting();

else

{

glDisable(GL\_LIGHTING);

glDisable(GL\_LIGHT0);

glDisable(GL\_COLOR\_MATERIAL);

}

glCallList(gridDisplayList);

double scale = CameraRadius;

float lineWidth = Math.Clamp((float)(1.0 / scale), 0.1f, 5.0f);

glLineWidth(lineWidth);

draw.Sphere(4.0, 1.5, 2.5, 2.0);

draw.Cylinder(-3.5, 1.5, -4.5, 1.5, 3.0);

draw.Disc(3.5, -1.0, -3.5, 3.0, 1.5);

}

// Освітлення

public void InitializeLighting()

{

glEnable(GL\_LIGHTING);

float[] lightPosition = { 10.0f, 10.0f, 10.0f, 1.0f };

float[] lightAmbient = { 0.2f, 0.2f, 0.2f, 1.0f };

float[] lightDiffuse = { 0.8f, 0.8f, 0.8f, 1.0f };

float[] lightSpecular = { 1.0f, 1.0f, 1.0f, 1.0f };

glLightfv(GL\_LIGHT0, GL\_POSITION, lightPosition);

glLightfv(GL\_LIGHT0, GL\_AMBIENT, lightAmbient);

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, lightDiffuse);

glLightfv(GL\_LIGHT0, GL\_SPECULAR, lightSpecular);

glEnable(GL\_LIGHT0);

glEnable(GL\_COLOR\_MATERIAL);

glColorMaterial(GL\_FRONT\_AND\_BACK, GL\_AMBIENT\_AND\_DIFFUSE);

float[] globalAmbient = { 0.1f, 0.1f, 0.1f, 1.0f };

glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, globalAmbient);

glLightModeli(GL\_LIGHT\_MODEL\_LOCAL\_VIEWER, (int)GL\_TRUE);

glLightModeli(GL\_LIGHT\_MODEL\_TWO\_SIDE, (int)GL\_TRUE);

}

// Список відображення

private void InitializeDisplayLists()

{

coordinatesDisplayList = glGenLists(1);

glNewList(coordinatesDisplayList, GL\_COMPILE);

draw.CoordinateLines(CenterX, CenterY, CenterZ);

glEndList();

gridDisplayList = glGenLists(1);

glNewList(gridDisplayList, GL\_COMPILE);

draw.Grid(20);

glEndList();

sphereDisplayList = glGenLists(1);

glNewList(sphereDisplayList, GL\_COMPILE);

draw.Sphere(4.0, 1.5, 2.5, 2.0);

glEndList();

cylinderDisplayList = glGenLists(1);

glNewList(cylinderDisplayList, GL\_COMPILE);

draw.Cylinder(-3.5, 1.5, -4.5, 1.5, 3.0);

glEndList();

discDisplayList = glGenLists(1);

glNewList(discDisplayList, GL\_COMPILE);

draw.Disc(3.5, -1.0, -3.5, 3.0, 1.5);

glEndList();

}

private void DeleteDisplayLists()

{

glDeleteLists(coordinatesDisplayList, 1);

glDeleteLists(gridDisplayList, 1);

glDeleteLists(sphereDisplayList, 1);

glDeleteLists(cylinderDisplayList, 1);

glDeleteLists(discDisplayList, 1);

}

private void Creating(object sender, EventArgs e)

{

draw = new Figures();

draw.Print = DrawText;

InitializeDisplayLists();

}

private void Deleting(object sender, EventArgs e)

{

DeleteDisplayLists();

}

// Інтерфейс керування параметрами площини відтину

public void ConfigureClippingPlane(double a, double b, double c, double d)

{

glEnable(GL\_CLIP\_PLANE0);

double[] planeEquation = { a, b, c, d };

glClipPlane(GL\_CLIP\_PLANE0, planeEquation);

}

public void DisableClippingPlane()

{

glDisable(GL\_CLIP\_PLANE0);

}

// Накладення текстури

public void ApplyTexture(uint textureId)

{

glEnable(GL\_TEXTURE\_2D);

glBindTexture(GL\_TEXTURE\_2D, textureId);

}

public void DisableTexture()

{

glDisable(GL\_TEXTURE\_2D);

}

}

}

***Код файлу (Figures.cs)***

using System;

using System.Runtime.InteropServices;

namespace Task05

{

public partial class RenderControl

{

public class Figures

{

public delegate void outText(string s, double x, double y, double z);

public outText Print;

public void Grid(double size, double step = 1.0)

{

glDisable(GL\_LIGHTING);

glLineWidth(0.5f);

glColor3ub(200, 200, 200);

glBegin(GL\_LINES);

for (double i = -size; i <= size; i += step)

{

glVertex3d(i, 0, -size);

glVertex3d(i, 0, size);

glVertex3d(-size, 0, i);

glVertex3d(size, 0, i);

}

glEnd();

glEnable(GL\_LIGHTING);

}

public void CoordinateLines(double x, double y, double z)

{

glDisable(GL\_LIGHTING);

glLineWidth(2.5f);

glBegin(GL\_LINES);

glColor3ub(0, 0, 200);

glVertex3d(x, y, z);

glVertex3d(x + 2, y, z);

glColor3ub(0, 200, 0);

glVertex3d(x, y, z);

glVertex3d(x, y + 2, z);

glColor3ub(200, 0, 0);

glVertex3d(x, y, z);

glVertex3d(x, y, z + 2);

glEnd();

Print?.Invoke("X", x + 2, y, z);

Print?.Invoke("Y", x, y + 2, z);

Print?.Invoke("Z", x, y, z + 2);

}

public void Sphere(double x0, double y0, double z0, double radius, int slices = 20, int stacks = 20)

{

glPushMatrix();

glTranslated(x0, y0, z0);

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

{

double phi1 = Math.PI \* i / stacks;

double phi2 = Math.PI \* (i + 1) / stacks;

glBegin(GL\_QUAD\_STRIP);

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

{

double theta = 2.0 \* Math.PI \* j / slices;

double x1 = radius \* Math.Sin(phi1) \* Math.Cos(theta);

double y1 = radius \* Math.Cos(phi1);

double z1 = radius \* Math.Sin(phi1) \* Math.Sin(theta);

double x2 = radius \* Math.Sin(phi2) \* Math.Cos(theta);

double y2 = radius \* Math.Cos(phi2);

double z2 = radius \* Math.Sin(phi2) \* Math.Sin(theta);

glColor3ub(150, 100, 255);

glVertex3d(x1, y1, z1);

glVertex3d(x2, y2, z2);

}

glEnd();

}

glPopMatrix();

}

public void Cylinder(double x0, double y0, double z0, double radius, double height, int slices = 20)

{

glPushMatrix();

glTranslated(x0, y0, z0);

glBegin(GL\_QUAD\_STRIP);

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

{

double theta = 2.0 \* Math.PI \* i / slices;

double x1 = radius \* Math.Cos(theta);

double z1 = radius \* Math.Sin(theta);

glColor3ub(200, 200, 100);

glVertex3d(x1, 0, z1);

glVertex3d(x1, height, z1);

}

glEnd();

glPopMatrix();

}

public void Disc(double x0, double y0, double z0, double radiusOuter, double radiusInner, int slices = 20)

{

glPushMatrix();

glTranslated(x0, y0, z0);

glBegin(GL\_QUAD\_STRIP);

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

{

double theta = 2.0 \* Math.PI \* i / slices;

double xOuter = radiusOuter \* Math.Cos(theta);

double zOuter = radiusOuter \* Math.Sin(theta);

double xInner = radiusInner \* Math.Cos(theta);

double zInner = radiusInner \* Math.Sin(theta);

glColor3ub(255, 200, 100);

glVertex3d(xOuter, 0, zOuter);

glVertex3d(xInner, 0, zInner);

}

glEnd();

glPopMatrix();

}

}

}

}

# додаток б посилання на репозиторій проєкту

<https://github.com/lashin-d/OpenGL>