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)

*Іл’ющенко А.С.*

(прізвище й ініціали студента)

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

(посада, науковий ступінь, прізвище й ініціали)

Національна шкала:

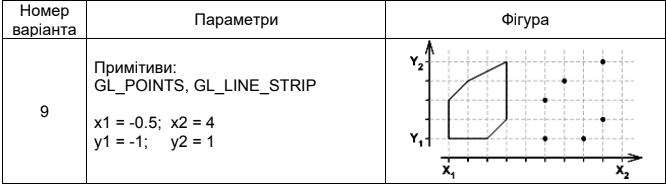
Кількість балів:

Оцінка ECTS:

# Практична робота 1. Основні принципи роботи з OpenGL

## Завдання, варіант № 9

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 Intel(R) Core(TM) i7-4790K CPU @ 4.00GHz 4.00 GHz

RAM 16.0 GB

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

Edition Windows 10 Pro Version 22H2

IDE Microsoft Visual Studio Community 2022 (64-bit) version 17.11.5

## 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.

### Команда glDrawArrays()

glDrawArrays() зчитує дані вершин з увімкнених масивів, проходячи прямо по масиву без пропусків або стрибків. Оскільки glDrawArrays() не дозволяє стрибати навколо масивів вершин, вам все одно доведеться повторювати спільні вершини один раз на кожну грань.

glDrawArrays() приймає 3 аргументи. По-перше, це примітивний тип. Другий параметр – це початковий виліт масиву. Останній параметр – це кількість вершин, які потрібно передати конвеєру рендерингу OpenGL.

Для наведеного вище прикладу для малювання куба першим параметром є GL\_TRIANGLES, другим - 0, що означає початок роботи масиву. І останній параметр - 36: у куба 6 сторін і кожній стороні потрібно 6 вершин, щоб намалювати 2 трикутника, 6 × 6 = 36.

GLfloat vertices[] = {...}; // 36 of vertex coords

...

// activate and specify pointer to vertex array

glEnableClientState(GL\_VERTEX\_ARRAY);

glVertexPointer(3, GL\_FLOAT, 0, vertices);

// draw a cube

glDrawArrays(GL\_TRIANGLES, 0, 36);

// deactivate vertex arrays after drawing

glDisableClientState(GL\_VERTEX\_ARRAY);

## Результати виконання практичної роботи

### Розв'язання завдання

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

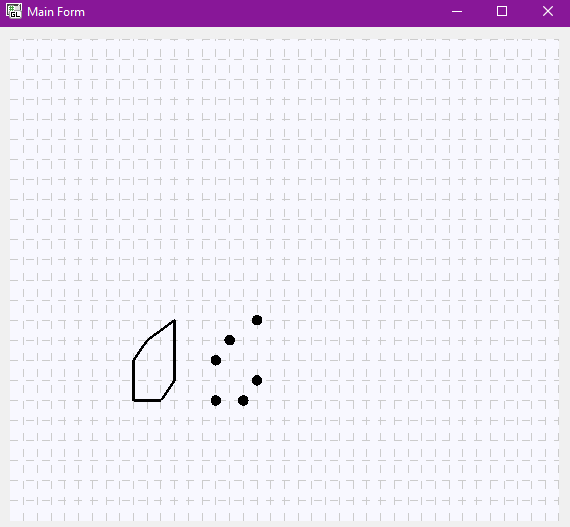
* колір, glColor3f() рядок 16 у файлі Task.cs;
* тип, glLineStipple(), glEnable()/glDisable(), рядок 18 у файлі Task.cs;
* товщина glLineWidth(), рядок 39 у файлі Task.cs

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

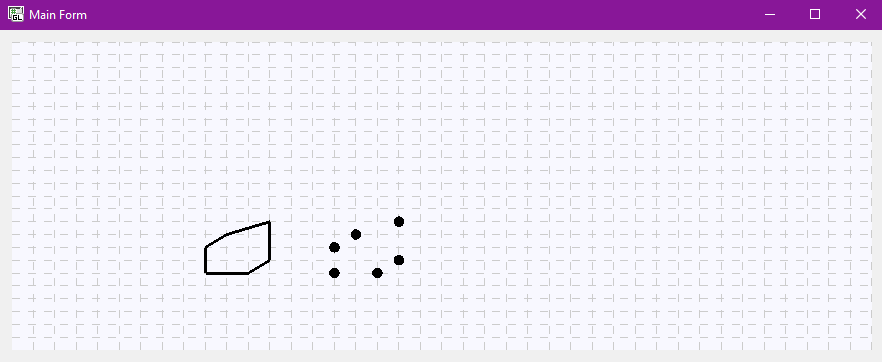
Розроблення підпрограм для виключення дублювання коду наведено у рядках 21 – 32 файлу Task.cs

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

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



*Рисунок 1.1 - Тестування програми при зміні висоти вікна*



*Рисунок 1.2 - Тестування програми при зміні ширини 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 (RenderControl.cs)

1. using System;
2. using System.Windows.Forms;
3. namespace Lab1
4. {
5. public partial class RenderControl : OpenGL
6. {
7. private Timer renderTimer;
8. public RenderControl()
9. {
10. InitializeComponent();
11. }
12. private void RenderGL(object sender, EventArgs e)
13. {
14. glViewport(0, 0, Width, Height);
15. glLoadIdentity();
16. gluOrtho2D(-8, +12, -4, +8);
17. glClear(GL\_COLOR\_BUFFER\_BIT);
18. Task task = new Task();
19. task.DrawGrid(); // Зображення координатної сітки
20. task.DrawFigure(); // Зображення фігури
21. task.DrawPoints(); // Зображення точок
22. }
23. }
24. }

### Код файлу (Task.cs)

1. using System;
2. using System.Collections.Generic;
3. using System.Drawing;
4. using System.Linq;
5. using System.Text;
6. using System.Threading.Tasks;
7. namespace Lab1
8. {
9. public partial class RenderControl
10. {
11. public class Task
12. {
13. public void DrawGrid()
14. {
15. glColor3f(0.8f, 0.8f, 0.8f);
16. glLineWidth(1.0f); // Товщина ліній сітки
17. glEnable(GL\_LINE\_STIPPLE); // Включення режиму ліній
18. glLineStipple(1, 0x00FF);
19. glBegin(GL\_LINES);
20. for (double i = -8; i <= 12; i+=0.5)
21. {
22. glVertex2d(i, -4);
23. glVertex2d(i, 8);
24. }
25. for (double j = -4; j <= 8; j+=0.5)
26. {
27. glVertex2d(-8, j);
28. glVertex2d(12, j);
29. }
30. glEnd();
31. glDisable(GL\_LINE\_STIPPLE); // Відключення пунктирних ліній
32. }
33. // Метод зображення фігури
34. public void DrawFigure()
35. {
36. glColor3f(0.0f, 0.0f, 0.0f); // Колір контуру
37. glLineWidth(3.0f); // Товщина ліній фігури
38. glBegin(GL\_LINE\_STRIP);
39. Figure(-3);
40. glEnd();
41. }
42. public void Figure(double offsetX = 0)
43. {
44. glVertex2d(-0.5 + offsetX, -1); // Нижня ліва точка
45. glVertex2d(-0.5 + offsetX, 0); // Верхня ліва точка
46. glVertex2d(0 + offsetX, 0.5); // Верхня центральна точка
47. glVertex2d(1 + offsetX, 1); // Верхня права точка
48. glVertex2d(1 + offsetX, -0.5); // Права точка
49. glVertex2d(0.5 + offsetX, -1); // Нижня права точка
50. glVertex2d(-0.5 + offsetX, -1); // Нижня ліва точка
51. }
52. // Метод зображення точок
53. public void DrawPoints()
54. {
55. glEnable(GL\_POINT\_SMOOTH);
56. glColor3f(0.0f, 0.0f, 0.0f);
57. glPointSize(10.0f);
58. glBegin(GL\_POINTS);
59. Figure(0);
60. glEnd();
61. glDisable(GL\_POINT\_SMOOTH);
62. }
63. }
64. }
65. }

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)

*Іл’ющенко А.С.*

(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:

Харків - 2024

# Практична робота 2. ГРАФІЧНІ ПРИМІТИВИ 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.

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).

## Системна інформація

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

Processor Intel(R) Core(TM) i7-4790K CPU @ 4.00GHz 4.00 GHz

RAM 16.0 GB

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

Edition Windows 10 Pro Version 22H2

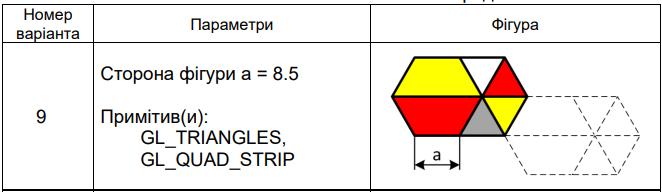
IDE Microsoft Visual Studio Community 2022 (64-bit) version 17.11.5

**Методичні вказівки**

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



## Результати виконання практичної роботи

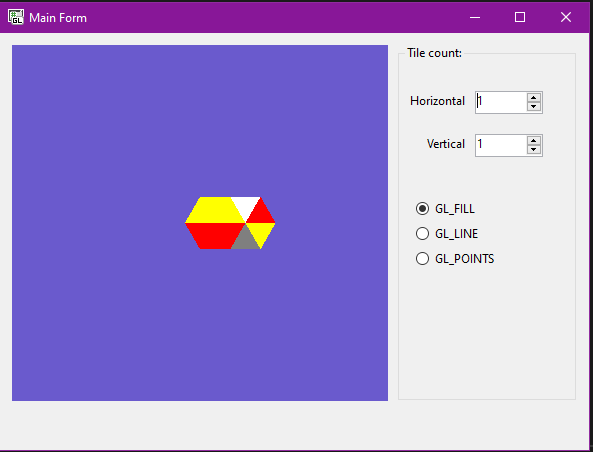
Розв'язання завдання

Для управління параметрами примітивів було використано наступні команди:

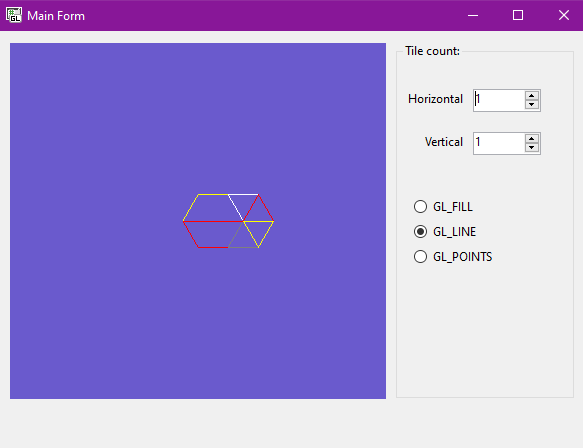
* колір, glColor3d() у рядках 25, 33, 41, 48, 55, 62 файлу figure.cs;

GL\_TRIANGLES у рядках 40, 47, 54, 61, GL\_QUAD\_STRIP у рядках 24, 32 файлу figure.cs

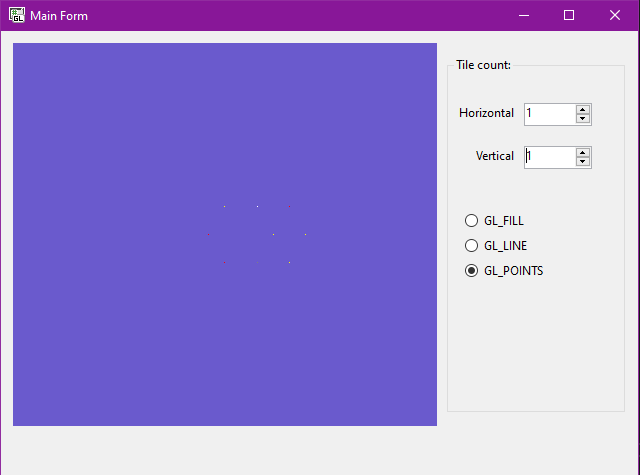
Головне меню



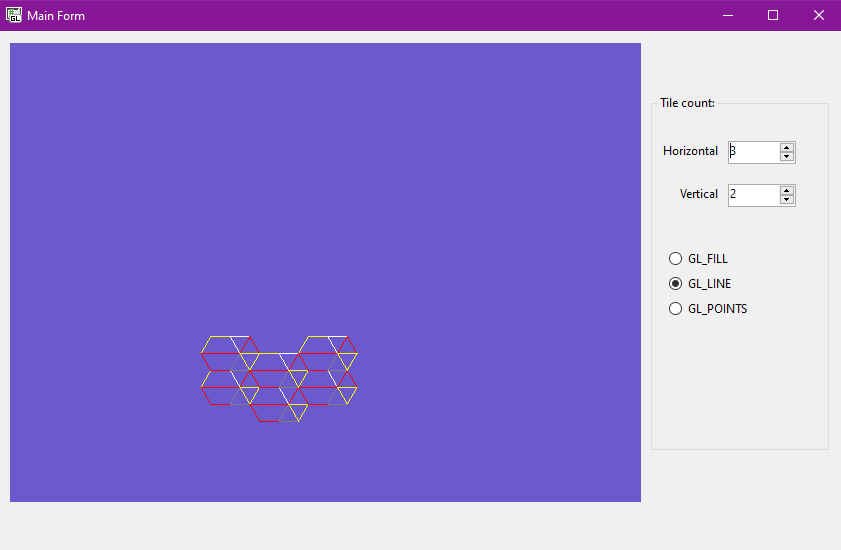
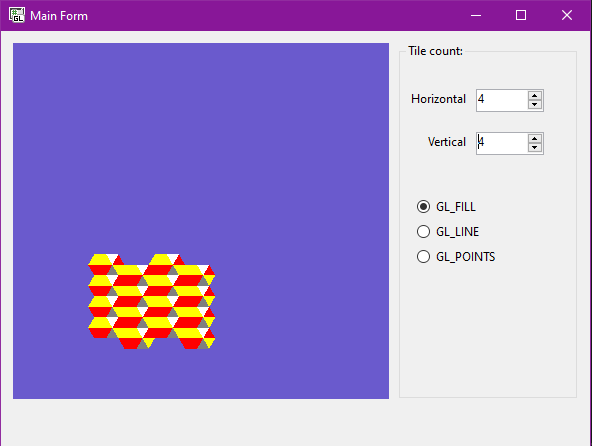
Тестування відображення ліній



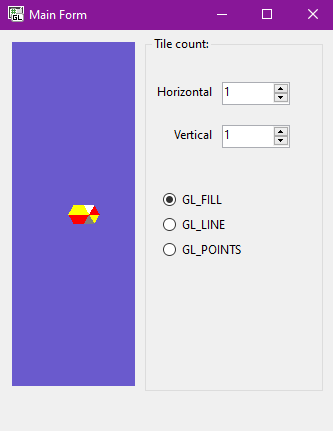
Тестування відображення точок



Тестування додавання плиток



Тестування зміни розміру вікна



### Контроль виконання вимог та елементів завдання

В результаті виконання практичної роботи були повністю виконані елементи базового рівня та частково підвищеного рівня складності, що відображено в таблиці 1.1.

Таблиця 1.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № з/п | Складність | Вимоги | Бали | Зроблено |
| 1 | Базовий рівень | 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 №2

### 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)

{

int ortho = 10;

glClear(GL\_COLOR\_BUFFER\_BIT);

glLoadIdentity();

if (Width > Height)

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

else

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

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

glOrtho(-50, 50 \* max, -50, 50 \* max, -1, 1);

double sideSize = 8.5;

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

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

{

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

{

double offsetX = col \* 2.5 \* sideSize;

double offsetY = row \* 2 \* height;

if (col % 2 != 0)

offsetY -= height;

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

}

}

glDisable(GL\_LINES);

}

}

}

***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 (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\_QUAD\_STRIP);

glColor3d(1, 1, 0); // Жовтий колір

glVertex2d(offsetX, offsetY + height); // Лівий верх

glVertex2d(offsetX + sidesize, offsetY + height); // Правий верх

glVertex2d(offsetX - 4.25, offsetY); // Правий низ

glVertex2d(offsetX + sidesize + 4.25, offsetY); // Лівий низ

glEnd();

glBegin(GL\_QUAD\_STRIP);

glColor3d(1, 0, 0); // Червоний колір

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

glVertex2d(offsetX, offsetY - height); // Правий верх

glVertex2d(offsetX + sidesize + 4.25, offsetY); // Правий низ

glVertex2d(offsetX - 4.25, offsetY); // Лівий низ

glEnd();

glBegin(GL\_TRIANGLES);

glColor3d(1, 1, 1); // Білий колір

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

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

glVertex2d(offsetX + sidesize + 4.25, offsetY); // Нижній кут

glEnd();

glBegin(GL\_TRIANGLES);

glColor3d(0.5, 0.5, 0.5); // Сірий колір

glVertex2d(offsetX + sidesize + 4.25, offsetY); // Верхній кут

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

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

glEnd();

glBegin(GL\_TRIANGLES);

glColor3d(1, 0, 0);

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

glVertex2d(offsetX + sidesize + 4.25, offsetY); // Лівий кут

glVertex2d(offsetX + sidesize \* 2 + 4.25, offsetY); // Правий кут

glEnd();

glBegin(GL\_TRIANGLES);

glColor3d(1, 1, 0);

glVertex2d(offsetX + sidesize + 4.25, offsetY); // Лівий кут

glVertex2d(offsetX + sidesize \* 2 + 4.25, offsetY); // Правий кут

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

glEnd();

}

}

}

}

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)

*Іл’ющенко А.С.*

(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.

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:

− 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.

## Системна інформація

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

Processor Intel(R) Core(TM) i7-4790K CPU @ 4.00GHz 4.00 GHz

RAM 16.0 GB

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

Edition Windows 10 Pro Version 22H2

IDE Microsoft Visual Studio Community 2022 (64-bit) version 17.11.5

**Методичні вказівки**

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

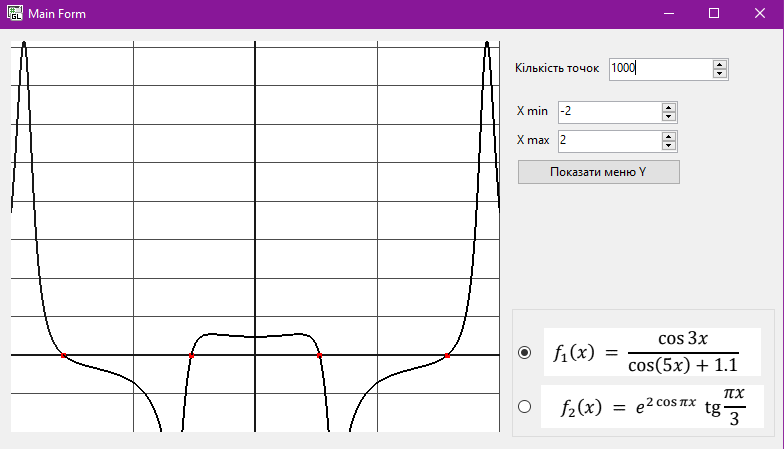
|  |  |  |
| --- | --- | --- |
| № варіанту | Функція f1(x) | Функція f2(x) |
| 9 |  |  |

## Результати виконання практичної роботи

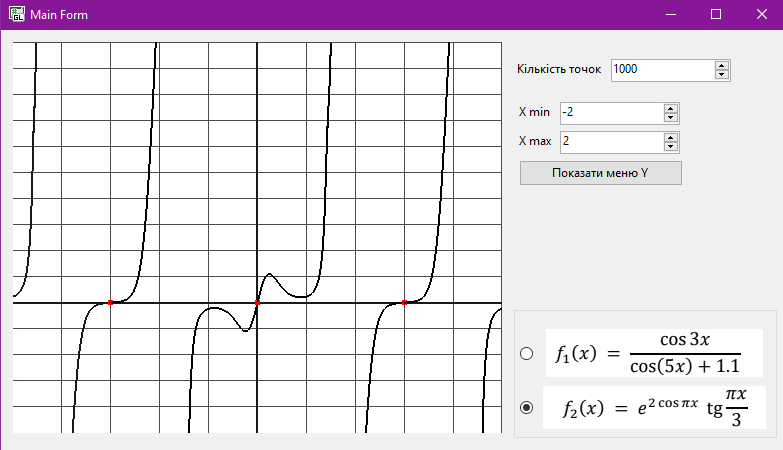
Розв'язання завдання

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

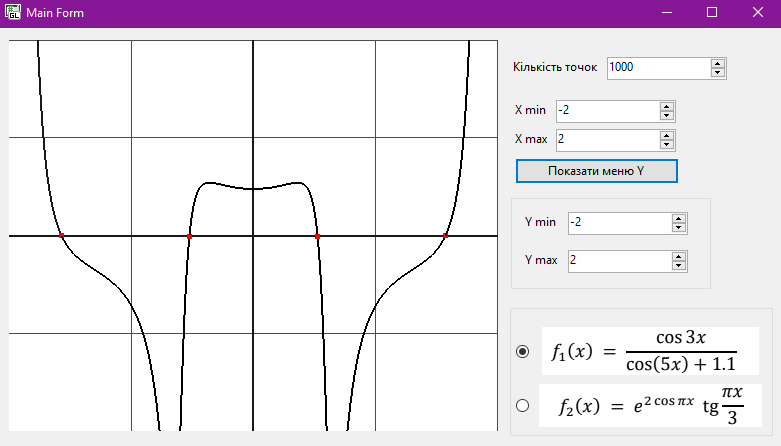
Головне меню з ***f1(x)***



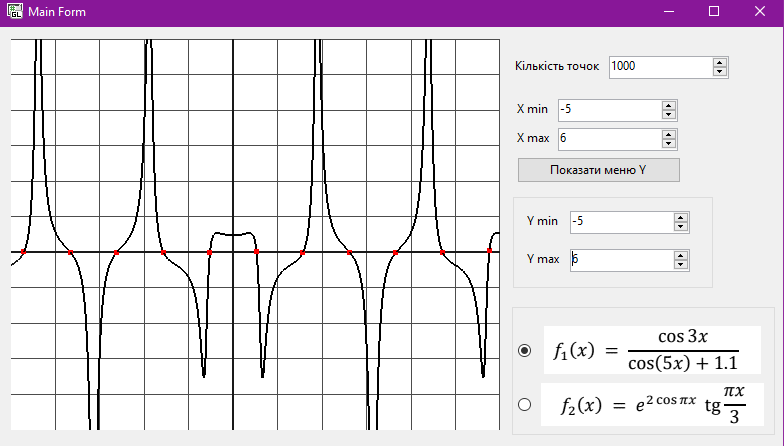
Головне меню з ***f2(x)***

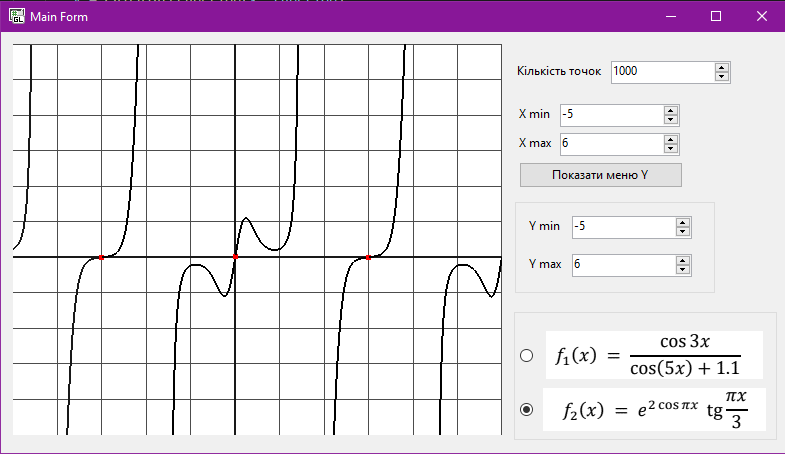


Робота меню Y



Редагування Xmin та Xmax





### Контроль виконання вимог та елементів завдання

В результаті виконання практичної роботи були повністю виконані елементи базового рівня та частково підвищеного рівня складності, що відображено в таблиці 1.1.

Таблиця 1.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № з/п | Складність | Вимоги | Бали | Зроблено |
| 1 | Базовий рівень | 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 №3

### 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 (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 (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(3 \* x) / (Math.Cos(5 \* x) + 1.1);

case 1:

return Math.Exp(2 \* Math.Cos(Math.PI \* x)) \* Math.Tan((Math.PI \* x) / 3);

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();

}

}

}

}

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)

*Іл’ющенко А.С.*

(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 КРИВІ ДРУГОГО ПОРЯДКУ

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

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. 1.1. Для кривих, які у варіанті відмічені «++», знайти та вивести на екран точки перетину, якщо такі є, з довільним відрізком, координати якого задає користувач.

## Системна інформація

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

Processor Intel(R) Core(TM) i7-4790K CPU @ 4.00GHz 4.00 GHz

RAM 16.0 GB

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

Edition Windows 10 Pro Version 22H2

IDE Microsoft Visual Studio Community 2022 (64-bit) version 17.11.5

**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 t1 першого 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].

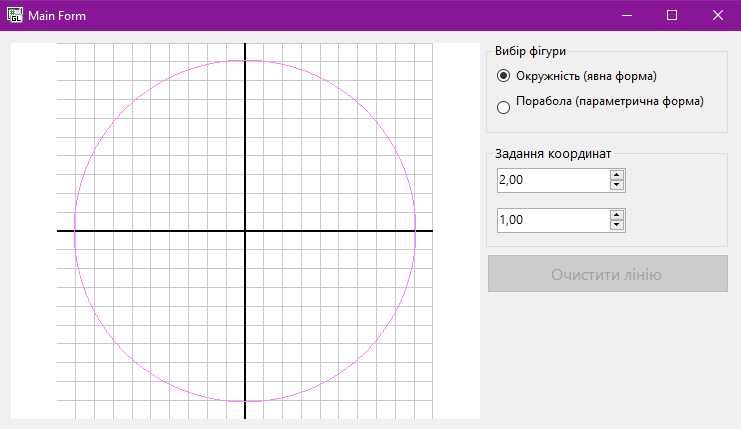
|  |  |  |
| --- | --- | --- |
| № варіанту | Окружність | Парабола |
| 9 | Подання | |
| + (Явне) | ++ (Параметричне) |

## Результати виконання практичної роботи

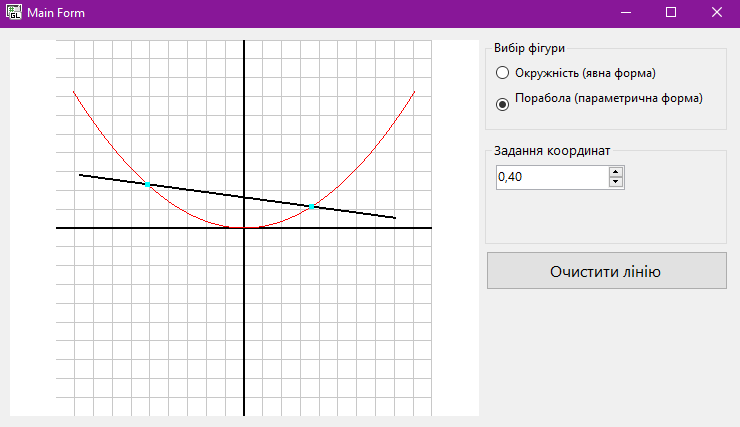
Розв'язання завдання

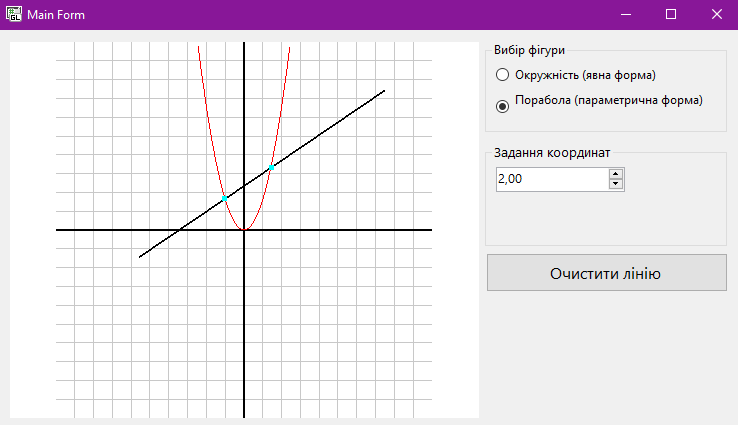
Для реалізації завдання було розроблено програму, що будує фігури окружності та параболи, які були зроблені завдяки кривим другого порядку та представлені в явної та параметричній формі.

Зображення окружності



Зображення параболи





### Контроль виконання вимог та елементів завдання

В результаті виконання практичної роботи були повністю виконані елементи базового рівня та підвищеного рівня складності, що відображено в таблиці 1.1.

Таблиця 1.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № 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 №4

### 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.ParametricPorabola(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 (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 (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 ParametricPorabola(double a)

{

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

glBegin(GL\_LINE\_STRIP);

int segments = 100;

double range = 2.0; // Діапазон значень для параметру t

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

{

double t = i \* range / segments; // Параметр t

double x = t;

double y = a \* t \* t;

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(b);

}

else

{

double range = 2.0;

maxX = range;

maxY = a \* range \* range;

maxY = maxY < 0 ? -maxY : maxY;

}

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();

}

}

}

}

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)

*Іл’ющенко А.С.*

(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 АФІННІ ПЕРЕТВОРЕННЯ У ПРОСТОРІ

**Мета:** навчитися працювати з тривимірними графічними примітивами OpenGL та застосовувати афінні перетворення для розміщення об’єктів у просторі.

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:

− 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.

## Системна інформація

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

Processor Intel(R) Core(TM) i7-4790K CPU @ 4.00GHz 4.00 GHz

RAM 16.0 GB

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

Edition Windows 10 Pro Version 22H2

IDE Microsoft Visual Studio Community 2022 (64-bit) version 17.11.5

**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.

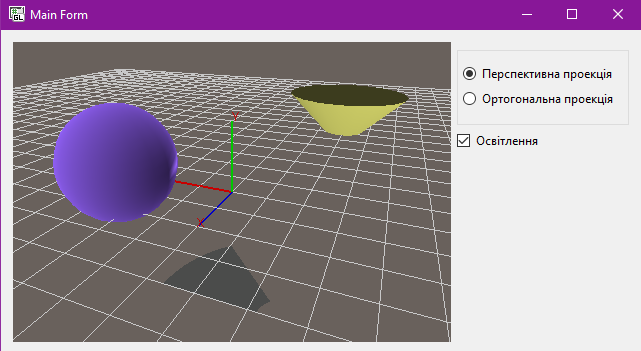
|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Варіант | Grid | Фігура | Ось | x0 | y0 | z0 | R | r | H | ∠start | ∠sweep |
| 9 | X0Z | Сфера | || 0Y | +1.5 | +1.0 | +2.5 | 1.5 | - | - | - | - |
| Усічений конус | ↑↓ 0Z | -2.0 | +1.5 | -2.5 | 1.5 | 0.5 | 1.0 | - | - |
| Частинний диск | || 0X | +3.5 | -0.5 | -2.5 | 2.5 | 0.5 | - | 90° | 45° |

## Результати виконання практичної роботи

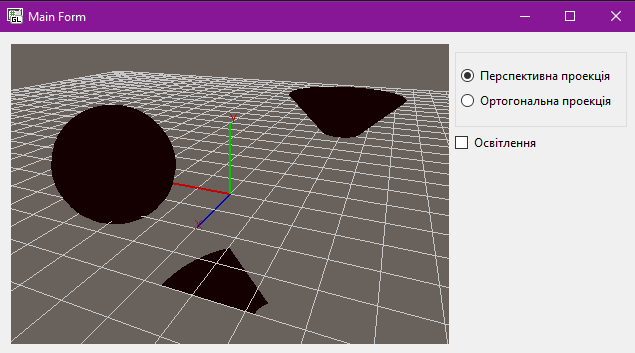
Розв'язання завдання

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

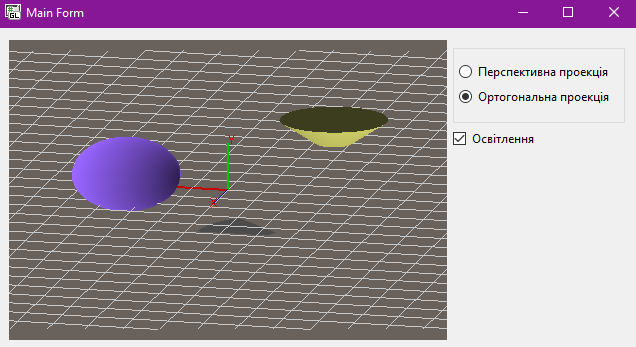
Зображення програми з освітленням



Сцена без освітлення



Зображення ортогональної проекції



### Контроль виконання вимог та елементів завдання

В результаті виконання практичної роботи були повністю виконані елементи базового рівня та підвищеного рівня складності, що відображено в таблиці 1.1.

Таблиця 1.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № 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 №5

### 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 coneDisplayList;

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, // Позиція камери

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(1.5, 1.0, 2.5, 1.5);

draw.TruncatedCone(-2.0, 1.5, -2.5, 0.5, 1.5, 1.0);

draw.PartialDisc(3.5, -0.5, -2.5, 0.5, 2.5, 90, 45);

}

// Освітлення

public void InitializeLighting()

{

glEnable(GL\_LIGHTING);

// Налаштування джерела світла 0 (позиційне світло)

float[] lightPosition = { 10.0f, 10.0f, 10.0f, 10.0f }; // Позиція світла

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

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);

// Встановлення глобальних параметрів освітлення

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

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

// Встановлення моделі відображення

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

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

}

// 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(1.5, 1.0, 2.5, 1.5);

glEndList();

coneDisplayList = glGenLists(1);

glNewList(coneDisplayList, GL\_COMPILE);

draw.TruncatedCone(-2.0, 1.5, -2.5, 0.5, 1.5, 1.0);

glEndList();

discDisplayList = glGenLists(1);

glNewList(discDisplayList, GL\_COMPILE);

draw.PartialDisc(3.5, -0.5, -2.5, 0.5, 2.5, 90, 45);

glEndList();

}

private void DeleteDisplayLists()

{

glDeleteLists(coordinatesDisplayList, 1);

glDeleteLists(gridDisplayList, 1);

glDeleteLists(sphereDisplayList, 1);

glDeleteLists(coneDisplayList, 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();

}

}

}

***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 (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;

// Координатна сітка

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)

{

// Лінії параллельно осі Z

glVertex3d(i, 0, -size);

glVertex3d(i, 0, size);

// Лінії параллельно осі X

glVertex3d(-size, 0, i);

glVertex3d(size, 0, i);

}

glEnd();

glEnable(GL\_LIGHTING);

}

// Зображення координат

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

{

glDisable(GL\_LIGHTING);

glLineWidth(2.5f);

glBegin(GL\_LINES);

glColor3ub(0, 0, 200); // Координата X

glVertex3d(x, y, z);

glVertex3d(x + 2, y, z);

glColor3ub(0, 200, 0); // Координата Y

glVertex3d(x, y, z);

glVertex3d(x, y + 2, z);

glColor3ub(200, 0, 0); // Координата Z

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);

}

// Сфера

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();

}

// Усіченний конус

public void TruncatedCone(double x0, double y0, double z0, double radius1, double radius2, 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 = radius1 \* Math.Cos(theta);

double z1 = radius1 \* Math.Sin(theta);

double x2 = radius2 \* Math.Cos(theta);

double z2 = radius2 \* Math.Sin(theta);

glColor3ub(200, 200, 100);

glVertex3d(x1, 0, z1);

glVertex3d(x2, height, z2);

}

glEnd();

glPopMatrix();

}

// Частинний диск

public void PartialDisc(double x0, double y0, double z0, double radiusInner, double radiusOuter, double startAngle, double sweepAngle, int slices = 40)

{

glPushMatrix();

glTranslated(x0, y0, z0);

glBegin(GL\_TRIANGLE\_STRIP);

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

{

double angle = startAngle + sweepAngle \* i / slices;

angle = angle \* Math.PI / 180.0;

double x1 = radiusInner \* Math.Cos(angle);

double z1 = radiusInner \* Math.Sin(angle);

double x2 = radiusOuter \* Math.Cos(angle);

double z2 = radiusOuter \* Math.Sin(angle);

glColor3ub(250, 255, 250);

glVertex3d(x1, 0, z1);

glVertex3d(x2, 0, z2);

}

glEnd();

glPopMatrix();

}

}

}

}