МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського

«Харківський авіаційний інститут»

Факультет програмної інженерії та бізнесу

Кафедра інженерії програмного забезпечення

Практичні роботи

Minor *«Розробник ігрових додатків»*

дисципліна *«Комп’ютерна графіка з OpenGL»*

(назва дисципліни)

Виконав: студент 3 курсу групи  *633п*

напряму підготовки (спеціальності):

*121 інженерія програмного забезпечення*

(шифр і назва напряму підготовки / спеціальності)

*Силенок Ю. А.*

(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:

**ЗМІСТ**

[Практична робота 1. Основні принципи роботи з OpenGL 3](#_Toc185706395)

[Завдання, варіант № 19 3](#_Toc185706396)

[Системна інформація 3](#_Toc185706397)

[Теоретичні відомості 3](#_Toc185706398)

[Результати виконання практичної роботи 5](#_Toc185706402)

[Практична робота 2. Графічні примітиви OpenGL 8](#_Toc185706405)

[Завдання, варіант № 19 8](#_Toc185706406)

[Результати виконання практичної роботи 8](#_Toc185706407)

[Загальний перелік посилань 14](#_Toc185706410)

[Додаток А. Лістинг програми до практичної роботи №1 15](#_Toc185706411)

[Додаток Б. Лістинг програми до практичної роботи №2 18](#_Toc185706414)

# Практична робота 1. Основні принципи роботи з OpenGL

## Завдання, варіант № 19

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.

|  |  |  |
| --- | --- | --- |
| 19 | Примітиви:  GL\_POINTS, GL\_LINES  x1 = -1.5; x2 = 7.5  y1 = -0.5; y2 = 3.5 |  |

## Системна інформація

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

Processor AMD Ryzen 5 5600H with Radeon Graphics 3.30 GHz

RAM 8.0 GB (5.86 GB usable)

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

Edition Windows 10 Home Version 22H2

IDE Microsoft Visual Studio Community 2022 (64-bit) version 17.11.2

## 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.

Для кожної заданої функції потрібні різні параметри. Будь ласка, перегляньте інструкції до API OpenGL. 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);

В результаті використання glDrawArrays() ви можете замінити 36 викликів glVertex\*() одним викликом glDrawArrays(). Однак нам все ще потрібно дублювати спільні вершини, тому кількість вершин, визначених у масиві, все ще становить 36 замість 8. glDrawElements() — це рішення для зменшення кількості вершин у масиві, тому воно дозволяє передавати менше даних до OpenGL.

### Команда glDrawElements()

glDrawElements() малює послідовність примітивів, перескакуючи навколо вершинних масивів з пов'язаними індексами масивів. При цьому зменшується як кількість викликів функцій, так і кількість вершин для передачі. Крім того, OpenGL може кешувати нещодавно оброблені вершини та повторно використовувати їх без повторного надсилання тих самих вершин у конвеєр перетворення вершин кілька разів.

## Результати виконання практичної роботи

### Розв'язання завдання

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

колір, glColor3d() рядок 15 у файлі Figure.cs;

тип, glLineStipple(), glEnable()/glDisable(), рядок 73, 74, 93 у файлі Figure.cs;

товщина glLineWidth(), рядках 7, 47, 76 у файлі Figure.cs.

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

Застосування циклів для створення зображень наведено у рядках 79 – 90 файлу Figure.cs.

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

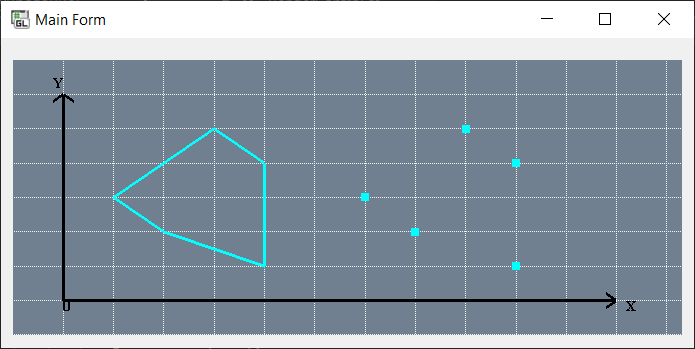


Рисунок 1.1 – Тестування програми при зміні ширини вікна

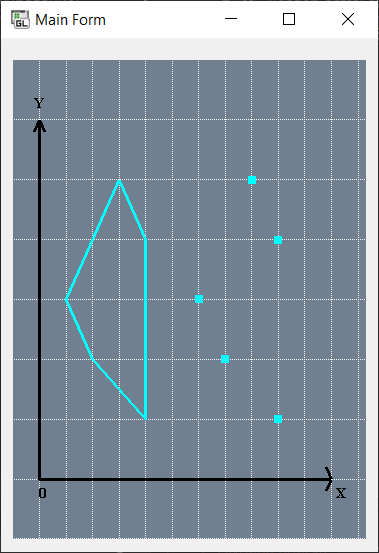


Рисунок 1.2 – Тестування програми при зміні висоти вікна

### Контроль виконання вимог та елементів завдання

В результаті виконання практичної роботи були частково 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 | 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 | **+** |

# Практична робота 2. ГРАФІЧНІ ПРИМІТИВИ oPENGL

## Завдання, варіант № 19

Використовуючи інструментальні засоби, що вказані викладачем, і беручи до уваги вимоги 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***).

Таблиця 2.1 – Варіаційна частина варіанту 19

|  |  |  |
| --- | --- | --- |
| № | Параметри | Фігура |
| 19 | Сторона фігури a = 0.15  Примітив(и):   GL\_TRIANGLE\_FAN, GL\_POLYGON |  |

## Результати виконання практичної роботи

### Розв'язання завдання

Під час запуску застосунку програми зображення відповідає варіанту завдання з однією плиткою це показано рисунку 2.1.

Багаторазове замощення плиткою та кратність замощення задається користувачем під час роботи застосунку показано на рисунку 2.2

Коректне відображення завдання під час зміни як розмірів/положення вікна, так і параметрів замощення показано на рисунках 2.3-2.6

Організовано взаємодію з користувачем за допомогою маніпулятора миші/клавіатури – взаємодія з інтерфейсом, а саме з елементами керування numericUpDown та radioButton.

Застосування мінімальної (у рамках варіанту) кількості графічних примітивів для виконання завдання – Додаток Б, рядок 33 у файлі HexagonRenderer.cs та рядок 19 у файлі TriangleRenderer.cs.

Використання ООП (розробка власних класів):

* RenderSettings – клас посередник між MainForm та RenderControl. MainForm передає налаштування елементів керування в об’єкт RenderSettings, звідки, через метод SetSettings класу посередника, RenderControl отримує відповідні налаштування.
* HexagonRenderer – клас, що виконує малювання шестикутників на основі налаштувань з RenderSettings, має виклик класу TriangleRenderer для малювання внутрішніх трикутників.
* TriangleRenderer – клас, що малює трикутники відповідно до координат, переданих з HexagonRenderer.

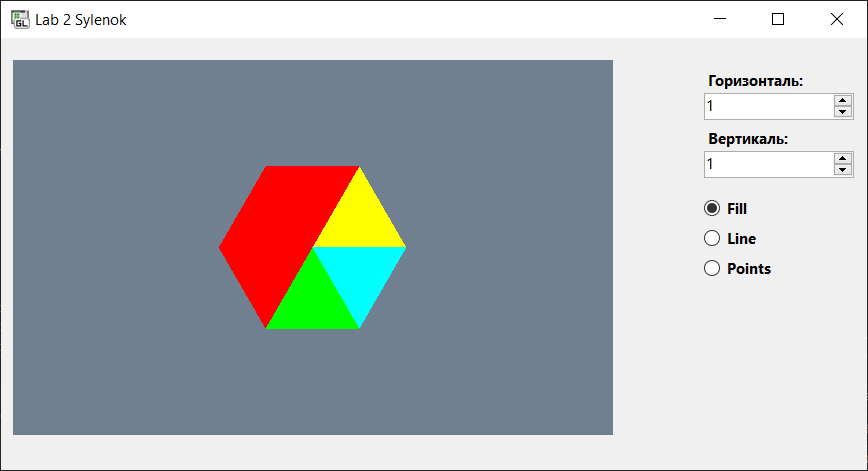


Рисунок 2.1 – Відображення при запуску програми

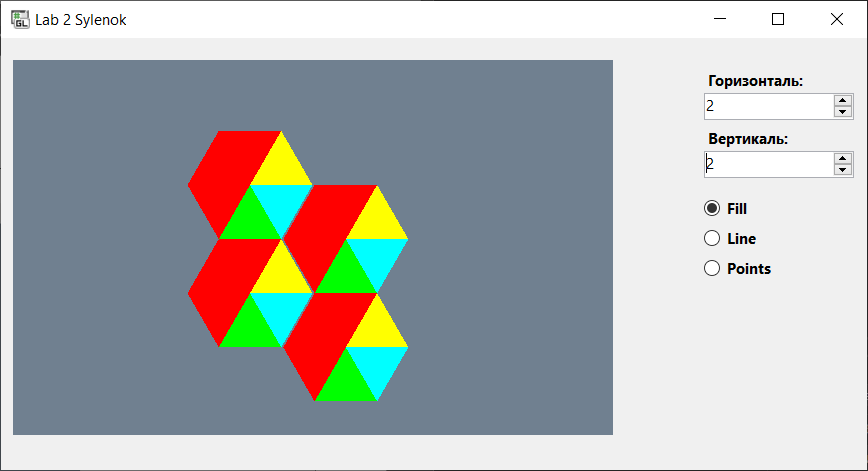


Рисунок 2.2 – Багаторазове замощення плитки та тестування інтерфейсу користувача

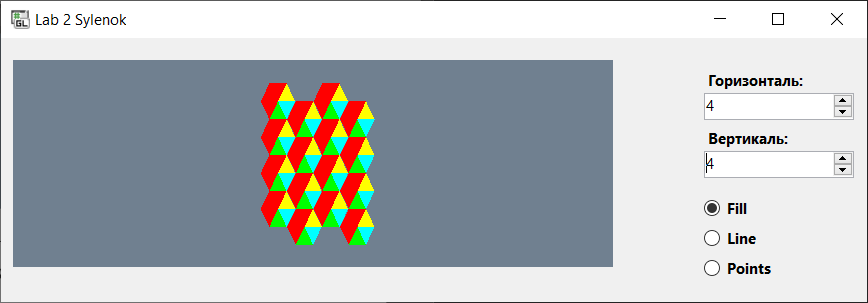


Рисунок 2.3 – Тестування багаторазового замощення вікна при зміні розмірів вікна по горизонталі

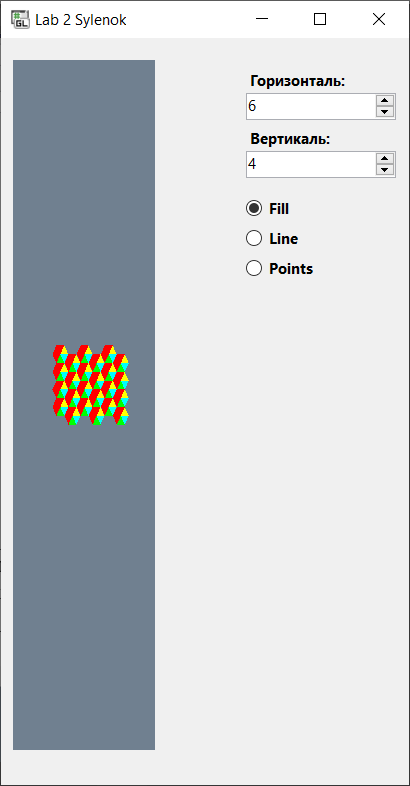


Рисунок 2.4 – Тестування багаторазового замощення вікна при зміні розмірів вікна по вертикалі

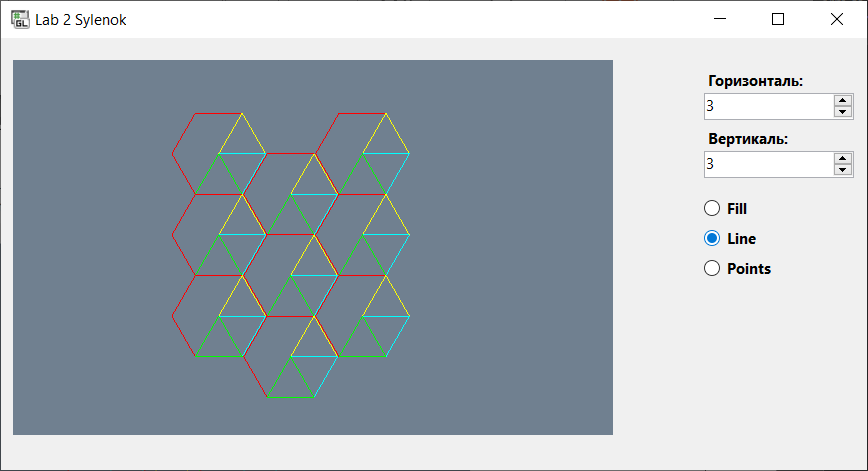


Рисунок 2.5 – Тестування зміни типу відображення на Line

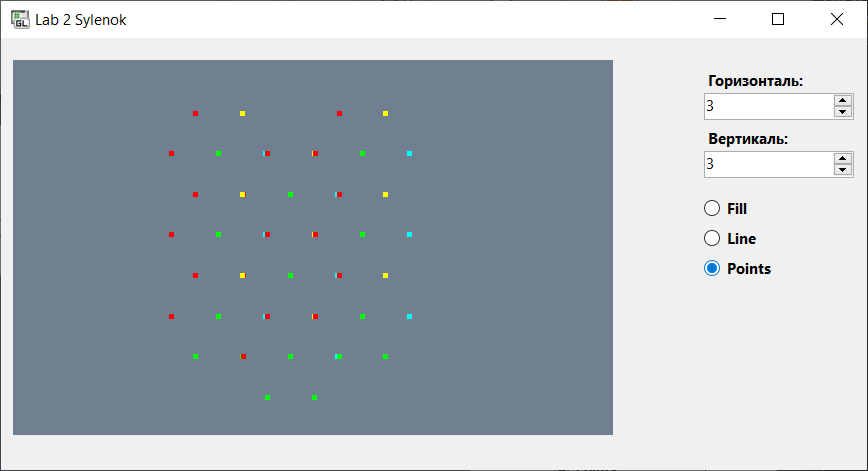


Рисунок 2.6 – Тестування зміни типу відображення на Points

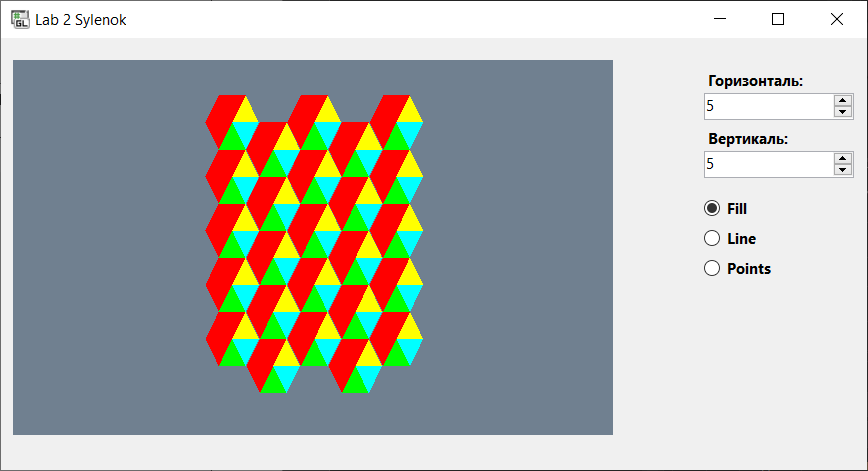


Рисунок 2.7 – Тестування зміни типу відображення на Fill

### 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 |
|  | Базовий рівень | Під час запуску застосунку зображення відповідає варіанту завдання з однією плиткою | 1 | **+** |
|  | Багаторазове замощення плиткою. Кратність замощення задається користувачем під час роботи застосунку | 1 | **+** |
|  | Коректне відображення завдання під час зміни як розмірів/положення вікна, так і параметрів замощення | 1 | **+** |
|  | Організація взаємодії з користувачем одним зі стандартних засобів (клавіатура, «миша» та ін.) | 1 | **+** |
|  | Застосування мінімальної (у рамках варіанту) кількості графічних примітивів для виконання завдання | 1 | **+** |
|  | Підвищений рівень | Створення власних елементів інтерфейсу за допомогою OpenGL | 2 | **-** |
|  | Використання ООП (розробка власних класів) | 1 | **+** |

# Загальний перелік посилань

1. Microsoft. glDrawArrays function [Електронний ресурс] / Microsoft – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/windows/win32/opengl/gldrawarrays>.

# Додаток А. Лістинг програми до практичної роботи №1

### Код файлу (RenderControl.cs)

1. public partial class RenderControl : OpenGL
2. {
3. Figure figure = new Figure();
4. public RenderControl()
5. {
6. InitializeComponent();
7. }
8. private void RenderControl\_Render(object sender, EventArgs e)
9. {
10. glClear(GL\_COLOR\_BUFFER\_BIT);
11. glLoadIdentity();
12. glViewport(0, 0, Width, Height);
13. gluOrtho2D(-3.5, 9.8, -2.5, 5.5);
14. figure.DrawGrid();
15. figure.DrawAxes();
16. figure.DrawPentagon();
17. figure.DrawPoints();
19. glColor3f(0, 0, 0);
20. DrawText("X", 8.7, -1.8);
21. DrawText("Y", -2.7, 4.7);
22. DrawText("0", -2.5, -1.8);
24. }
25. }

### Код файлу (Figure.cs)

1. internal class Figure
2. {
3. public void DrawPentagon()
4. {
5. glColor3d(0, 1, 1);
7. glLineWidth(2.5f);
8. glBegin(GL\_LINES);
9. glVertex2d(-1.5, 1.5);
10. glVertex2d(0.5, 3.5);
11. glVertex2d(0.5, 3.5);
12. glVertex2d(1.5, 2.5);
13. glVertex2d(1.5, 2.5);
14. glVertex2d(1.5, -0.5);
15. glVertex2d(1.5, -0.5);
16. glVertex2d(-0.5, 0.5);
17. glVertex2d(-0.5, 0.5);
18. glVertex2d(-1.5, 1.5);
19. glEnd();
20. }
21. public void DrawPoints()
22. {
23. glPointSize(8.0f);
25. glBegin(GL\_POINTS);
26. glVertex2d(3.5, 1.5);
27. glVertex2d(4.5, 0.5);
28. glVertex2d(5.5, 3.5);
29. glVertex2d(6.5, 2.5);
30. glVertex2d(6.5, -0.5);
31. glEnd();
32. }
33. public void DrawAxes()
34. {
35. glColor3f(0.0f, 0.0f, 0.0f);
36. glLineWidth(3.0f);
37. glBegin(GL\_LINES);
38. glVertex2d(-2.5, -1.5);
39. glVertex2d(8.5, -1.5);
40. glVertex2d(8.3, -1.3);
41. glVertex2d(8.5, -1.5);
42. glVertex2d(8.3, -1.7);
43. glVertex2d(8.5, -1.5);
44. glVertex2d(-2.5, -1.5);
45. glVertex2d(-2.5, 4.5);
46. glVertex2d(-2.3, 4.3);
47. glVertex2d(-2.5, 4.5);
48. glVertex2d(-2.7, 4.3);
49. glVertex2d(-2.5, 4.5);
50. glEnd();
51. }
52. public void DrawGrid()
53. {
54. glLineStipple(1, 0xAAAA);
55. glEnable(GL\_LINE\_STIPPLE);
56. glLineWidth(1.0f);
57. glBegin(GL\_LINES);
58. for (double x = -3.5; x <= 9.8; x += 1.0)
59. {
60. glVertex2d(x, -2.5);
61. glVertex2d(x, 5.5);
63. }
64. for (double y = -2.5; y <= 5.5; y += 1.0)
65. {
66. glVertex2d(-3.5, y);
67. glVertex2d(9.8, y);
68. }
69. glEnd();
70. glDisable(GL\_LINE\_STIPPLE);
72. }
73. }

# Додаток Б. Лістинг програми до практичної роботи №2

### Код файлу (RenderControl.cs)

1. public partial class RenderControl : OpenGL
2. {
3. private RenderSettings settings;
4. public RenderControl()
5. {
6. InitializeComponent();
7. }
8. public void SetSettings(RenderSettings renderSettings)
9. {
10. settings = renderSettings;
11. }
13. private void OnRender(object sender, EventArgs e)
14. {
15. glClear(GL\_COLOR\_BUFFER\_BIT);
16. glLoadIdentity();
17. float aspectRatio = (float)Width / Height;
18. int horizont = settings.Horizont;
19. int vertical = settings.Vertical;
20. if (Width > Height)
21. glViewport((Width - Height) / 2, 0, Height, Height);
22. else glViewport(0, (Height - Width) / 2, Width, Width);
23. if (vertical > horizont)
24. gluOrtho2D(-0.3 \* (vertical), +0.3 \* (vertical), -0.3 \* (vertical / 1), 0);
25. else if (vertical == 1 && horizont == 1)
26. gluOrtho2D(-0.3, +0.3, -0.3, +0.3);
27. else gluOrtho2D(-0.3 \* (horizont / 2), +0.3 \* (horizont), -0.3 \* (horizont / 1), +0.3);
28. for (int v = 0; v < vertical; v++)
29. {
30. for (int h = 0; h < horizont; h++)
31. {
32. double offsetX = 0.23 \* h;
33. double offsetY = -0.1299 \* (h % 2) - 0.2598 \* v;
34. DrawHexagonAndTriangles(offsetX, offsetY);
35. }
36. }
37. }
38. private void DrawHexagonAndTriangles(double offsetX, double offsetY)
39. {
40. HexagonRenderer hexagonRenderer = new HexagonRenderer(0.15, settings.Fill, settings.Line, settings.Points);
41. hexagonRenderer.DrawHexagon(offsetX, offsetY);
42. }
43. }

### Код файлу (HexagonRenderer.cs)

1. public class HexagonRenderer
2. {
3. private double radius;
4. private bool fill;
5. private bool line;
6. private bool points;
7. public HexagonRenderer(double radius, bool fill, bool line, bool points)
8. {
9. this.radius = radius;
10. this.fill = fill;
11. this.line = line;
12. this.points = points;
13. }
14. public void DrawHexagon(double offsetX, double offsetY)
15. {
16. int numVertices = 6;
17. double[,] vertices = new double[numVertices, 2];
18. for (int i = 0; i < numVertices; i++)
19. {
20. double angle = 2.0 \* Math.PI \* i / numVertices;
21. double x = radius \* Math.Cos(angle) + offsetX;
22. double y = radius \* Math.Sin(angle) + offsetY;
23. vertices[i, 0] = x;
24. vertices[i, 1] = y;
25. }
26. if (fill)
27. {
28. glColor3f(1.0f, 0.0f, 0.0f);
29. glBegin(GL\_POLYGON);
30. for (int i = 0; i < numVertices; i++)
31. {
32. glVertex2d(vertices[i, 0], vertices[i, 1]);
33. }
34. glEnd();
35. }
36. if (line)
37. {
38. glColor3f(1.0f, 0.0f, 0.0f);
39. glBegin(GL\_LINE\_LOOP);
40. for (int i = 0; i < numVertices; i++)
41. {
42. glVertex2d(vertices[i, 0], vertices[i, 1]);
43. }
44. glEnd();
45. }
46. if (points)
47. {
48. glPointSize(5.0f);
49. glColor3f(1.0f, 0.0f, 0.0f);
50. glBegin(GL\_POINTS);
51. for (int i = 0; i < numVertices; i++)
52. {
53. glVertex2d(vertices[i, 0], vertices[i, 1]);
54. }
55. glEnd();
56. }
57. TriangleRenderer triangleRenderer = new TriangleRenderer(fill, line, points);
58. triangleRenderer.DrawTriangle(vertices, 0, 1, offsetX, offsetY, 0, 1, 1);
59. triangleRenderer.DrawTriangle(vertices, 0, 5, offsetX, offsetY, 0, 1, 0);
60. triangleRenderer.DrawTriangle(vertices, 4, 5, offsetX, offsetY, 1, 1, 0);
61. }
62. }

### Код файлу (RenderSettings.cs)

1. public class RenderSettings
2. {
3. public int Horizont { get; set; } = 1;
4. public int Vertical { get; set; } = 1;
5. public bool Fill { get; set; } = true;
6. public bool Line { get; set; } = false;
7. public bool Points { get; set; } = false;
8. }

### Код файлу (TriangleRenderer.cs)

1. public class TriangleRenderer
2. {
3. private bool fill;
4. private bool line;
5. private bool points;
6. public TriangleRenderer(bool fill, bool line, bool points)
7. {
8. this.fill = fill;
9. this.line = line;
10. this.points = points;
11. }
12. public void DrawTriangle(double[,] vertices, int v1, int v2, double offsetX, double offsetY, float r, float g, float b)
13. {
14. if (fill)
15. {
16. glColor3f(r, g, b);
17. glBegin(GL\_TRIANGLE\_FAN);
18. glVertex2d(offsetX, offsetY);
19. glVertex2d(vertices[v1, 0], vertices[v1, 1]);
20. glVertex2d(vertices[v2, 0], vertices[v2, 1]);
21. glEnd();
22. }
23. if (line)
24. {
25. glColor3f(r, g, b);
26. glBegin(GL\_LINE\_LOOP);
27. glVertex2d(offsetX, offsetY);
28. glVertex2d(vertices[v1, 0], vertices[v1, 1]);
29. glVertex2d(vertices[v2, 0], vertices[v2, 1]);
30. glEnd();
31. }
32. if (points)
33. {
34. glPointSize(5.0f);
35. glColor3f(r, g, b);
36. glBegin(GL\_POINTS);
37. glVertex2d(offsetX, offsetY);
38. glVertex2d(vertices[v1, 0], vertices[v1, 1]);
39. glVertex2d(vertices[v2, 0], vertices[v2, 1]);
40. glEnd();
41. }
42. }
43. }

### Код файлу (TriangleRenderer.cs)

1. public partial class MainForm : Form
2. {
3. private RenderSettings renderSettings;
4. public MainForm()
5. {
6. InitializeComponent();
7. renderSettings = new RenderSettings
8. {
9. Horizont = 1,
10. Vertical = 1,
11. Fill = true,
12. Line = false,
13. Points = false
14. };
16. numericUpDown\_Horizont.Value = renderSettings.Horizont;
17. numericUpDown\_Vertical.Value = renderSettings.Vertical;
18. radioButton\_Fill.Checked = renderSettings.Fill;
19. renderControl1.SetSettings(renderSettings);
20. }
21. private void numericUpDown\_Horizont\_ValueChanged(object sender, EventArgs e)
22. {
23. renderSettings.Horizont = (int)numericUpDown\_Horizont.Value;
24. renderControl1.Invalidate();
25. }
26. private void numericUpDown\_Vertical\_ValueChanged(object sender, EventArgs e)
27. {
28. renderSettings.Vertical = (int)numericUpDown\_Vertical.Value;
29. renderControl1.Invalidate();
30. }
31. private void radioButton\_Fill\_CheckedChanged(object sender, EventArgs e)
32. {
33. renderSettings.Fill = radioButton\_Fill.Checked;
34. renderControl1.Invalidate();
35. }
36. private void radioButton\_Line\_CheckedChanged(object sender, EventArgs e)
37. {
38. renderSettings.Line = radioButton\_Line.Checked;
39. renderControl1.Invalidate();
40. }
41. private void radioButton\_Points\_CheckedChanged(object sender, EventArgs e)
42. {
43. renderSettings.Points = radioButton\_Points.Checked;
44. renderControl1.Invalidate();
45. }
46. }