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  *631п*

напряму підготовки (спеціальності):

*121 інженерія програмного забезпечення*

(шифр і назва напряму підготовки / спеціальності)

*Бахшалієв А.Е.*

(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 № 2

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.

|  |  |  |
| --- | --- | --- |
| 2 | Примітиви:  GL\_POINTS, GL\_LINE\_LOOP  x1 = -8; x2 = 1  y1 = -3; y2 = 1 | W001-03 |

## Системна інформація

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

Processor AMD Ryzen 5 5600 6-Core Processor @ 3.50GHz 3.50GHz

RAM 32.0 GB (31.9 GB usable)

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

Edition Windows 11 Version 23H2

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[[1]](#footnote-1).

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 12 у файлі Figure.cs;
* тип, glLineStipple(), glEnable()/glDisable(), рядок 27 у файлі MyForm.cs;
* товщина glLineWidth(), рядок 41 у файлі Figures.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 28 – 55 файлу Figures.cs

Застосування циклів для створення зображень наведено у рядках 15 – 24 файлу Figures.cs.

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

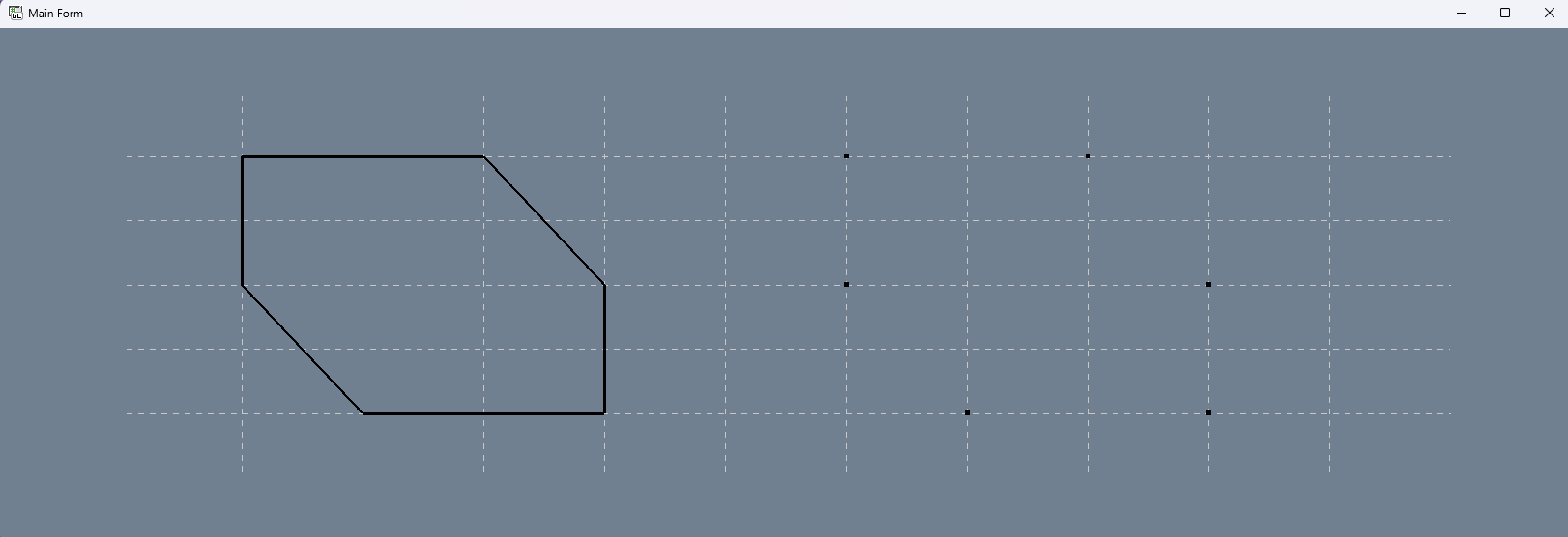


Рисунок 1.1 – Тестування програми при зміні ширини вікна

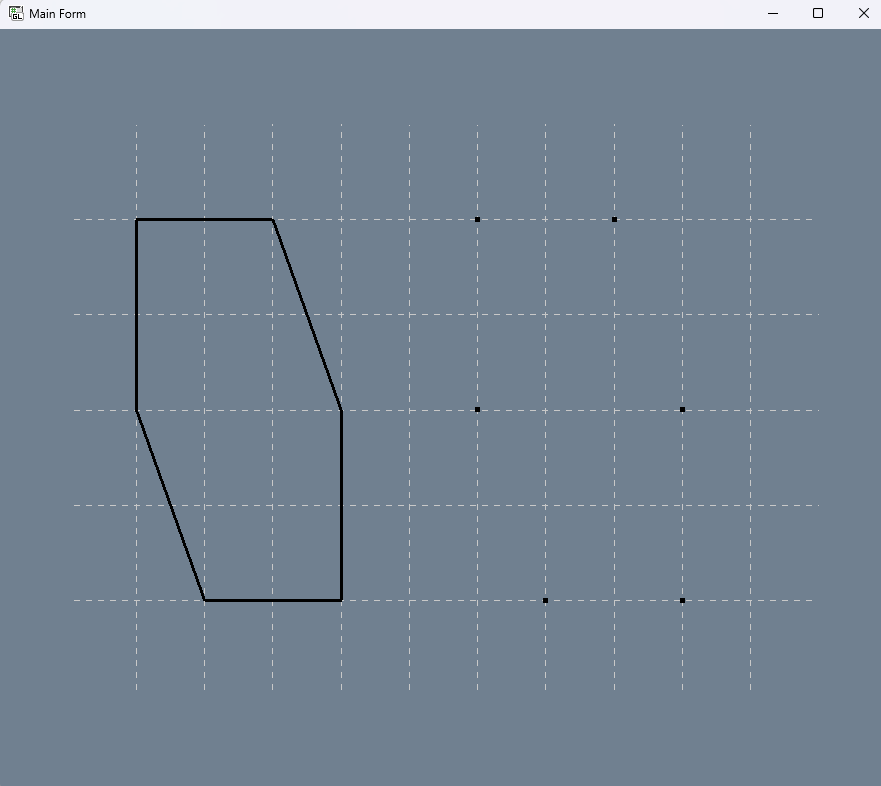


Рисунок 1.2 – Тестування програми при зміні висоти вікна

### Контроль виконання вимог та елементів завдання

В результаті виконання практичної роботи були 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 | 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. Назва роботи

## Завдання, варіант № 2

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. 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. Крім цього, користувач повинен мати можливість змінювати режим відображення графічних примітивів OpenGL: точкове (тільки вершини фігури), контурне і з заповненням кольором. 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.

|  |  |  |
| --- | --- | --- |
| 2 | Сторона фігури a = 4.25  Примітиви:   GL\_TRIANGLES,  GL\_QUADS |  |

## Системна інформація

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

Processor AMD Ryzen 5 5600 6-Core Processor @ 3.50GHz 3.50GHz

RAM 32.0 GB (31.9 GB usable)

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

Edition Windows 11 Version 23H2

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

## Теоретичні відомості

Графічні примітиви OpenGL — це основні елементи для побудови 2D та 3D графіки. OpenGL надає кілька видів примітивів, таких як точки, лінії, трикутники та багатокутники, за допомогою яких можна створювати складні геометричні об'єкти.

### Основні типи графічних примітивів:

* GL\_POINTS (Точки):

Використовуються для відображення окремих точок на екрані.

Кожна вершина, визначена у вершинному масиві, є точкою.

* GL\_LINES (Лінії):

Дві вершини створюють лінію.

Можна використовувати для побудови лінійних сегментів між точками.

* GL\_LINE\_STRIP і GL\_LINE\_LOOP (Лінійні смуги та петлі):

Використовують послідовність вершин для створення кількох підряд з'єднаних ліній.

Відрізняються тим, що у GL\_LINE\_LOOP остання вершина з'єднується з першою.

* GL\_TRIANGLES (Трикутники):

Найбільш використовуваний примітив для відображення поверхонь.

Кожен трикутник визначається трьома вершинами.

Також є варіанти: GL\_TRIANGLE\_STRIP і GL\_TRIANGLE\_FAN, які дозволяють оптимізувати кількість вершин.

* GL\_QUADS (Чотирикутники):

Використовується для створення заповнених чотирикутних поверхонь. Чотирикутник визначається точно чотирма вершинами, які задаються у правильному порядку (або за годинниковою, або проти годинникової стрілки).

### Команди для роботи з графічними примітивами в OpenGL:

* glBegin() і glEnd():

Команди, які огортають блоки коду, що описують примітиви.

Наприклад, glBegin(GL\_TRIANGLES) оголошує, що починається блок трикутників, а кожна трійка вершин буде малювати один трикутник.

* glVertex():

Визначає координати вершин для кожного примітиву.

Наприклад, glVertex2f(x, y) задає двовимірні координати вершини.

* glColor():

Додає колір до вершин або примітивів.

### Способи малювання примітивів:

* glDrawArrays():

Команда, яка дозволяє малювати примітиви, використовуючи масиви вершин.

Приймає три параметри: тип примітиву, індекс першої вершини і кількість вершин.

Це один з найефективніших способів малювання, оскільки скорочує кількість викликів функцій.

* glDrawElements():

Дозволяє малювати примітиви, використовуючи індексовані масиви вершин.

Дозволяє уникати дублювання спільних вершин для складних фігур.

### Управління масштабом та координатною системою:

* glOrtho() і gluOrtho2D():

Використовуються для встановлення проекції, яка контролює, як об'єкти відображаються на екрані.

* gluOrtho2D(left, right, bottom, top) встановлює двовимірну ортогональну проекцію, де left, right, bottom, і top — це межі координатної площини.
* glViewport():

Визначає частину вікна, на якій буде відображатися зображення.

Це дозволяє масштабувати або змінювати розмір області, де рендеряться примітиви.

## Результати виконання практичної роботи

### Розв'язання завдання

У даній практичній роботі було розроблено застосунок з використанням бібліотеки OpenGL для відображення правильних багатокутника та можливості замощення області екрану користувачем. Програма реалізована з використанням команд OpenGL для керування примітивами, налаштування координатної системи, відображення фігур та взаємодії з користувачем.

Налаштування координатної системи:

Використовуються функції glOrtho() та glViewport() для встановлення ізотропної системи координат, що дозволяє відображати багатокутники в центрі вікна незалежно від розмірів області рендерингу. Параметри масштабування визначаються розмірами фігури та кількістю плиток, які буде відображено на екрані.

Коректне відображення завдання під час змінення розмірів/положення вікна наведено у рис. 2.1 та 2.2

Відображення багатокутника:

Для відображення правильного багатокутника використано примітиви GL\_TRIANGLE\_STRIP та GL\_QUADS. Після старту програми у робочій області відображається одна плитка. Розмір плитки визначено згідно з варіантом, де сторона фігури дорівнює 4.25.

Реалізовано три режими відображення фігур:

* Точковий режим (відображення лише вершин фігури) за допомогою примітиву GL\_POINTS.
* Контурний режим (відображення лише контуру фігури) за допомогою примітиву GL\_LINE.
* Режим із заливкою (заповнення кольором) за допомогою примітиву GL\_TRIANGLE\_STRIP та GL\_QUADS.

Колірна схема:

Для зафарбування фігур використано 4 кольори: сірий, червоний, синій та жовтий. Фарба накладається відповідно до положення багатокутників на екрані.

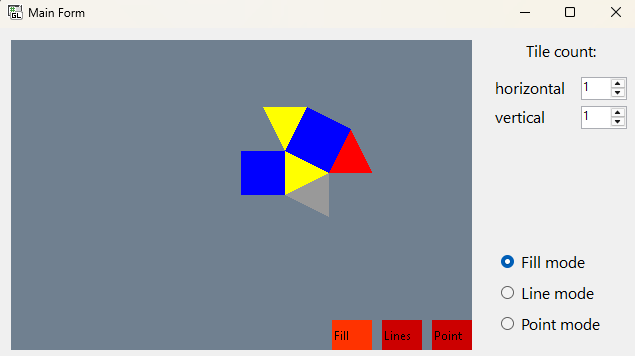


Рисунок 2.1 – Тестування програми при запуску

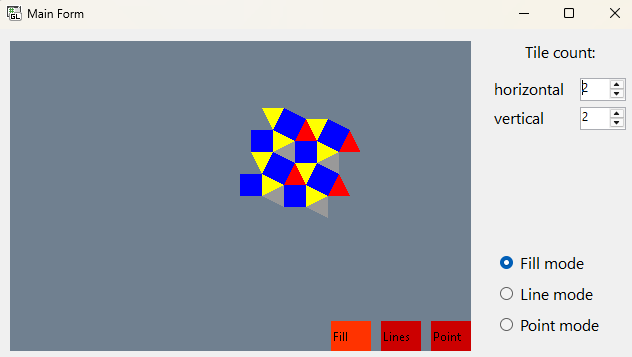


Рисунок 2.2 – Тестування програми при додаванні плиток

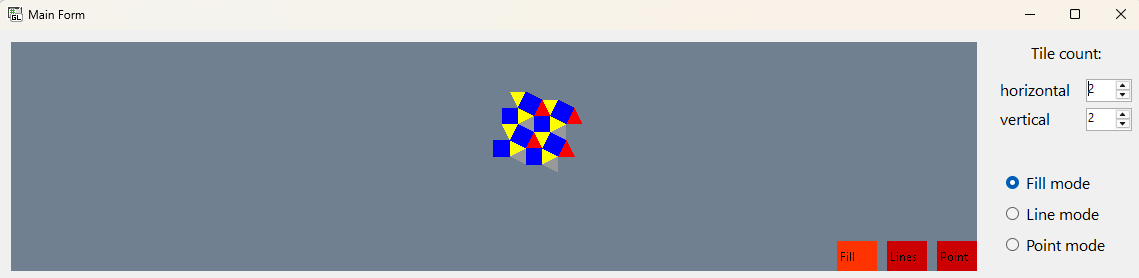


Рисунок 2.3 – Тестування програми при зміні ширини вікна

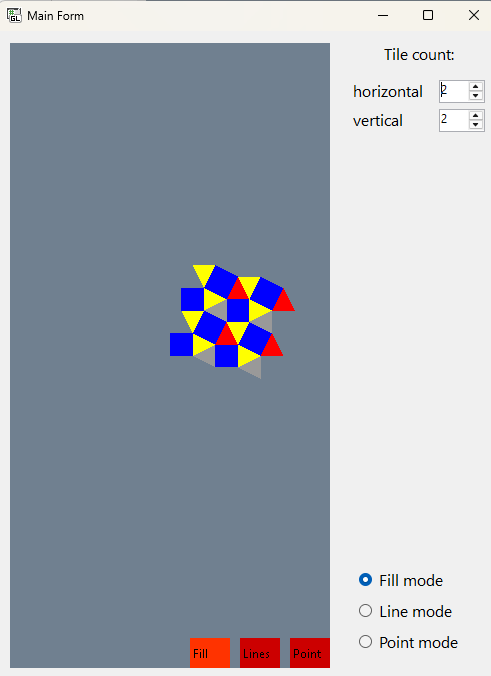


Рисунок 2.4 – Тестування програми при зміні висоти вікна

### 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 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 | **+** |

# Практична робота ?. Назва роботи

## Завдання, варіант № ??

## Теоретичні відомості

## Результати виконання практичної роботи

### Розв'язання завдання

### Контроль виконання вимог та елементів завдання

# 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 (MainForm.cs)

1. using System;
2. using System.Collections.Generic;
3. using System.ComponentModel;
4. using System.Diagnostics;
5. using System.Drawing;
6. using System.Drawing.Design;
7. using System.Linq;
8. namespace Lab1
9. {
10. public partial class RenderControl : OpenGL
11. {
12. public RenderControl()
13. {
14. InitializeComponent();
15. }
16. private void RenderControl\_Render(object sender, EventArgs e)

19. {

1. Figures figures = new Figures();

21. glClear(GL\_COLOR\_BUFFER\_BIT);

1. glLoadIdentity();
2. glViewport(0, 0, Width, Height);
3. gluOrtho2D(-10, +3, -5, +3);

25.

26. glLineWidth(1);

27. glEnable(GL\_LINE\_STIPPLE);

28. glLineStipple(6, 0xAAAA);

29. glColor3ub(200, 200, 200);

1. glBegin(GL\_LINES);
2. figures.Grid();
3. figures.DrawFigure();
4. figures.DrawPoints();
5. }
6. }
7. }

### Код файлу (Figures.cs)

1 using System;

2 using System.Collections.Generic;

3 using System.Linq;

4 using System.Text;

5 using System.Threading.Tasks;

6

7 namespace e Lab1

8 {

9 public partial class RenderControl

10 {

11 public class Figures

12 {

13 public void Grid()

14 {

15 for (int i = -8; i <= 1; i++)

16 {

17 glVertex2d(i, -4);

18 glVertex2d(i, 2);

19 }

20 for (int j = -3; j <= 1; j++)

21 {

22 glVertex2d(-9, j);

23 glVertex2d(2, j);

24 }

25 glEnd();

26 glDisable(GL\_LINE\_STIPPLE);

27 }

28 public void Form(int OffsetX = 0)

29 {

30 glBegin(GL\_LINE\_LOOP);

31 glVertex2d(-8 + OffsetX, 1);

32 glVertex2d(-6 + OffsetX, 1);

33 glVertex2d(-5 + OffsetX, -1);

34 glVertex2d(-5 + OffsetX, -3);

35 glVertex2d(-7 + OffsetX, -3);

36 glVertex2d(-8 + OffsetX, -1);

37 glEnd();

38 }

39 public void DrawFigure()

40 {

41 glLineWidth(3);

42 glColor3ub(0, 0, 0);

43 glBegin(GL\_LINE\_LOOP);

44 Form(0);

45 glEnd();

46 }

47 public void DrawPoints()

48 {

49 glEnable(GL\_POINT);

50 glPointSize(5);

51 glBegin(GL\_POINTS);

52 Form(5);

53 glEnd();

54 glDisable(GL\_POINT);

55 }

56 }

57 }

58 }

# Додаток Б. Лістинг програми до практичної роботи №2

### Код файлу (???)

### Код файлу (???)

# Додаток ?. Лістинг програми до практичної роботи №?

### Код файлу (???)

### Код файлу (???)

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