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:

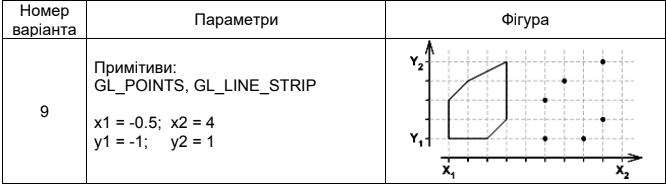
Кількість балів:

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):

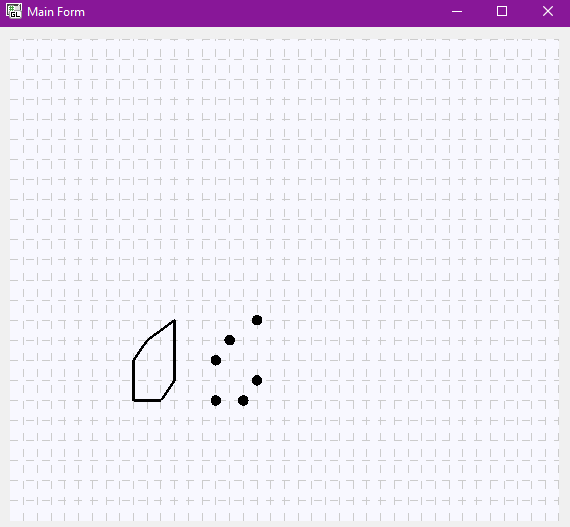
* колір, glColor3f() рядок 16 у файлі Task.cs;
* тип, glLineStipple(), glEnable()/glDisable(), рядок 18 у файлі Task.cs;
* товщина glLineWidth(), рядок 39 у файлі Task.cs

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

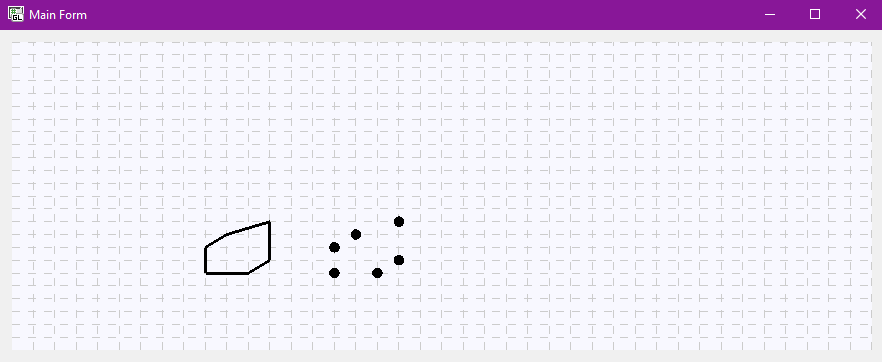
Розроблення підпрограм для виключення дублювання коду наведено у рядках 21 – 32 файлу Task.cs

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. }