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](#_Toc185705707)

[Завдання, варіант № 19 3](#_Toc185705708)

[Системна інформація 3](#_Toc185705709)

[Теоретичні відомості 3](#_Toc185705710)

[Результати виконання практичної роботи 5](#_Toc185705714)

[Загальний перелік посилань 8](#_Toc185705717)

[Додаток А. Лістинг програми до практичної роботи №1 9](#_Toc185705718)

# 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.

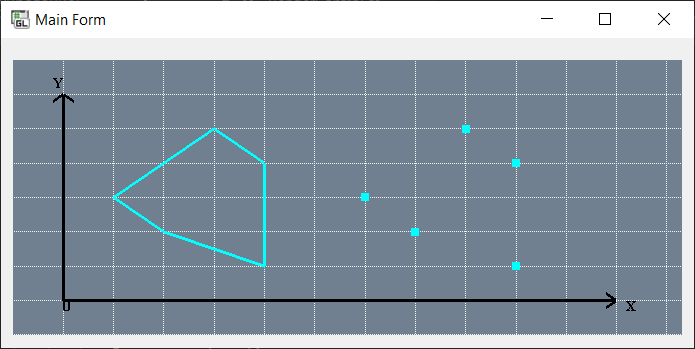


Рисунок 1.1 – Тестування програми при зміні ширини вікна

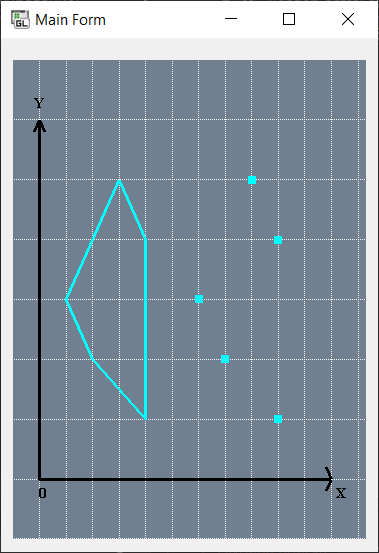


Рисунок 1.2 – Тестування програми при зміні висоти вікна

### 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 | **+** |

# 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. }