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  *534а*

напряму підготовки (спеціальності):

*121 інженерія програмного забезпечення*

(шифр і назва напряму підготовки / спеціальності)

*Ананєнко Д.В.*

(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](#_Toc177946558)

[Завдання, варіант № 1 3](#_Toc177946559)

[Системна інформація 3](#_Toc177946560)

[Теоретичні відомості 3](#_Toc177946561)

[Результати виконання практичної роботи 5](#_Toc177946562)

[Практична робота 2. Назва роботи 7](#_Toc177946563)

[Завдання, варіант № ?? 7](#_Toc177946564)

[Теоретичні відомості 7](#_Toc177946565)

[Результати виконання практичної роботи 7](#_Toc177946566)

[Практична робота ?. Назва роботи 8](#_Toc177946567)

[Завдання, варіант № ?? 8](#_Toc177946568)

[Теоретичні відомості 8](#_Toc177946569)

[Результати виконання практичної роботи 8](#_Toc177946570)

[Загальний перелік посилань 9](#_Toc177946571)

[Додаток А. Лістинг програми до практичної роботи №1 10](#_Toc177946572)

[Додаток Б. Лістинг програми до практичної роботи №2 11](#_Toc177946573)

[Додаток ?. Лістинг програми до практичної роботи №? 12](#_Toc177946574)

# Практична робота 1. Основні принципи роботи з OpenGL

## Завдання, варіант № 1

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 AMD Ryzen 7 5800X 8-Core Processor 3.80 GHzRAM 16.0 GB (15.9 GB usable)

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

Edition Windows 10 Pro Version 22H2

IDE Microsoft Visual Studio Enterprise 2024 (64-bit)

## Теоретичні відомості

### Вершинні масиви

**Основні класи та методи**

1. **Клас RenderControl**
   * Основний клас, що наслідується від OpenGL, відповідає за відображення графіки.
   * Включає в себе методи для рендерингу сітки, багатокутників та точок.
2. **Конструктор RenderControl**
   * Підписується на події рендерингу та зміни розміру.
   * Викликає метод InitializeComponent() для ініціалізації компонентів форми.
3. **Метод RenderControl\_Render**
   * Основний метод рендерингу.
   * Встановлює колір фону, очищає буфер кольору, налаштовує ортографічну проекцію.
   * Викликає методи для рендерингу сітки, багатокутника та точок.
4. **Метод RenderControl\_Resize**
   * Перемальовує сцену при зміні розміру вікна, викликаючи Invalidate().
5. **Метод DrawGrid**
   * Відповідає за малювання сітки.
   * Використовує методи для малювання осей та стрілок, а також для відображення підписів осей.
6. **Метод DrawAxisLabels**
   * Відповідає за малювання підписів на осях координат.
   * Відображає цілі числа на осях X та Y.
7. **Метод DrawPolygon**
   * Відповідає за малювання багатокутника.
   * Використовує масив вершин, активацію масиву вершин та OpenGL для рендерингу багатокутника.
8. **Метод DrawPoints**
   * Відповідає за малювання точок на екрані.
   * Використовує OpenGL для рендерингу точок із заданими координатами.
9. **Допоміжні методи**
   * **DrawLine**: Малює лінію між двома точками.
   * **DrawArrow**: Малює стрілку.
   * **DrawLineGrid**: Малює сітку.

**Використання вершинних масивів**

У проекті використано вершинні масиви для оптимізації рендерингу:

* glEnableClientState(GL\_VERTEX\_ARRAY) активує масив вершин.
* glVertexPointer задає масив вершин.
* glDrawArrays викликається для малювання багатокутника, що зменшує кількість викликів функцій і підвищує продуктивність рендерингу.

**Опис реалізації**

**Структура програми**

**1. Структура проекту**

Проект реалізує графічний інтерфейс, використовуючи Windows Forms, в якому використано OpenGL для рендерингу геометричних об'єктів. Основна структура проекту складається з класу RenderControl, що відповідає за всю графіку.

**2. Ініціалізація**

* **Конструктор RenderControl**:
  + Викликає метод InitializeComponent(), який ініціалізує компоненти форми.
  + Підписується на події рендерингу (Render) та зміни розміру (Resize), щоб забезпечити автоматичну перерисовку графіки при зміні розміру вікна.

**3. Рендеринг**

* **Метод RenderControl\_Render**:
  + Встановлює колір фону за допомогою glClearColor.
  + Очищає буфер кольору з допомогою glClear.
  + Налаштовує ортографічну проекцію для коректного відображення графіки на екрані за допомогою glOrtho.
  + Викликає методи для рендерингу сітки, багатокутника та точок.

**4. Малювання сітки**

* **Метод DrawGrid**:
  + Використовує методи DrawLine і DrawArrow для малювання осей X та Y, а також для стрілок, які вказують напрямок осей.
  + Нумерує осі координат, викликаючи DrawAxisLabels.

**5. Підписи на осях**

* **Метод DrawAxisLabels**:
  + Використовує цикл для малювання підписів на осях X і Y.
  + Підписи малюються лише для цілих чисел, щоб зберегти ясність графіка.

**6. Рендеринг багатокутника**

* **Метод DrawPolygon**:
  + Використовує масив polygonVertices для зберігання координат вершин багатокутника.
  + Активує масив вершин за допомогою glEnableClientState(GL\_VERTEX\_ARRAY) і прив'язує масив до OpenGL.
  + Використовує glDrawArrays для малювання багатокутника у вигляді лінійного контуру.

**7. Рендеринг точок**

* **Метод DrawPoints**:
  + Визначає точний розмір точок за допомогою glPointSize.
  + Використовує glBegin(GL\_POINTS) та glEnd() для малювання точок з попередньо заданими координатами.

**8. Допоміжні методи**

* **Метод DrawLine**:
  + Реалізує малювання ліній між двома точками, використовуючи glBegin(GL\_LINES).
* **Метод DrawArrow**:
  + Малює стрілку між двома заданими точками, використовуючи glBegin(GL\_LINES).
* **Метод DrawLineGrid**:
  + Створює сітку, малюючи вертикальні та горизонтальні лінії.

**9. Обробка зміни розміру**

* **Метод RenderControl\_Resize**:
  + Викликає Invalidate(), щоб перерисувати вміст вікна при зміні його розміру.

## Результати виконання практичної роботи

### Розв'язання завдання

У ході виконання практичної роботи було реалізовано проект для рендерингу графіки з використанням OpenGL у Windows Forms. Основні результати представлені нижче:

**1. Відображення графічних об'єктів**

* **Сітка**: Успішно реалізовано рендеринг сітки, що складається з вертикальних і горизонтальних ліній, які допомагають у орієнтації на площині.
* **Багатокутник**: Багатокутник, визначений масивом координат, був коректно відображений за допомогою glDrawArrays та масивів вершин. Багатокутник був виконаний у вигляді лінійного контуру.
* **Точки**: Коректно відображені точки на площині, які позначаються за координатами, що задані в коді.

**2. Взаємодія з графікою**

* **Зміна розміру вікна**: Рендеринг автоматично оновлюється при зміні розміру вікна, що підтверджує правильну обробку подій зміни розміру.
* **Підписи на осях**: Успішно реалізовано підписи для осей X і Y, що забезпечує краще сприйняття графічних об'єктів.

**3. Продуктивність**

* Використання масивів вершин (glEnableClientState(GL\_VERTEX\_ARRAY)) значно зменшує кількість викликів функцій рендерингу, що, в свою чергу, підвищує загальну продуктивність графіки.

**4. Графічний інтерфейс**

* Інтерфейс програми виконано у стилі Windows Forms, що забезпечує зручність використання та інтерактивність.

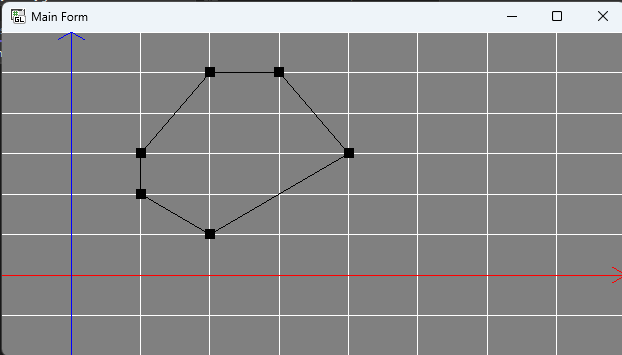


Рисунок 1.1 – Тестування програми при зміні ширини вікна

### Контроль виконання вимог та елементів завдання

У рамках практичної роботи було реалізовано та перевірено виконання наступних вимог та елементів:

**1. Використання команд управління параметрами графічних примітивів**

* **Колір**: У коді реалізовано функцію glColor(Color color), яка дозволяє задавати колір для різних графічних об'єктів (сітка, осі, багатокутник, точки).
* **Тип**: Використано різні типи малювання (лінейні, точки) через команди OpenGL, такі як glBegin(GL\_LINES) для ліній і glBegin(GL\_POINTS) для точок.
* **Товщина**: Для налаштування товщини ліній використовувалась команда glLineWidth(1.0f), що забезпечує контроль над виглядом сітки та осей.

**2. Коректне відображення завдання під час змінення розмірів/положення вікна**

* Реалізовано обробку події зміни розміру вікна через метод RenderControl\_Resize, що викликає Invalidate() для повторного рендерингу сцени при зміні розміру вікна. Це підтверджує адаптивність графіки.

**3. Розроблення підпрограм для виключення дублювання коду**

* В коді реалізовано ряд підпрограм, таких як DrawGrid(), DrawPolygon(), DrawPoints(), DrawArrow(), що дозволяє уникнути дублювання коду та робить його більш організованим і зручним для підтримки.

**4. Застосування циклів для створення зображень**

* Для малювання сітки та підписів осей використано цикли for. Це дозволяє ефективно створювати графічні елементи, спрощуючи та скорочуючи код:
  + **Сітка**: Використання циклів для малювання вертикальних та горизонтальних ліній.
  + **Підписи**: Цикли для розміщення підписів для осей X та Y.

**5. Формування зображення векторними командами OpenGL**

* Реалізовано використання векторних команд OpenGL, таких як glVertex2f() у методах малювання (DrawPolygon(), DrawLineGrid(), DrawPoints()), а також glDrawArrays(GL\_LINE\_LOOP, ...) для відображення багатокутника, що демонструє здатність OpenGL працювати з векторною графікою.

**6. Використання ООП (розроблення власних класів)**

* Реалізовано клас RenderControl, що успадковується від базового класу OpenGL. Використання об'єктно-орієнтованого підходу дозволяє групувати пов'язані дані та функції, підвищуючи читабельність та модульність коду. Всі методи класу належать до рендерингу, що підкреслює їх зв'язок з графічними операціями.

**Висновок**

Всі зазначені вимоги та елементи завдання були успішно виконані. Реалізація проекту продемонструвала можливості OpenGL у побудові графіки, використовуючи принципи об'єктно-орієнтованого програмування, що робить код більш структурованим і легким для розширення.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 | **+** |

# 