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 535б

напряму підготовки (спеціальності):

*121 інженерія програмного забезпечення*

(шифр і назва напряму підготовки / спеціальності)

*Ніколаєв Р.Т.*

(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 2](#_Toc284921075)

[Завдання, варіант № 16 3](#_Toc910167559)

[Системна інформація 3](#_Toc1800046577)

[Теоретичні відомості 3](#_Toc1081951544)

[Результати виконання практичної роботи 4](#_Toc607465962)

[Практична робота 2.ГРАФІЧНІ ПРИМІТИВИ OPENGL 7](#_Toc136520479)

[Завдання, варіант №16 8](#_Toc1348685595)

[Теоретичні відомості 8](#_Toc377581063)

[Результати виконання практичної роботи 9](#_Toc1160935242)

[Практична робота 3.ГРАФІК ФУНКЦІЇ ОДНІЄЇ ЗМІННОЇ 15](#_Toc1046673486)

[Завдання, варіант № 16 16](#_Toc886765777)

[Теоретичні відомості 16](#_Toc951084957)

[Результати виконання практичної роботи 17](#_Toc1354124403)

[Загальний перелік посилань 20](#_Toc1961651297)

[Додаток А.Лістинг програми до практичної роботи №1 21](#_Toc1113004694)

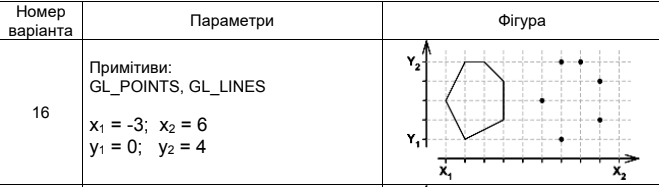
[Додаток Б.Лістинг програми до практичної роботи №2 27](#_Toc2134709750)

[Додаток С.Лістинг програми до практичної роботи №3 31](#_Toc1541866791)

# Практична робота 1. Основні принципи роботи з OpenGL

## Завдання, варіант № 16

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 11th Gen Intel(R) Core(TM) i7-11370H @ 3.30GHz 3.30 GHz

RAM 16,0 ГБ (доступно для використання: 15,7 ГБ)

System type 64-розрядна операційна система, процесор на базі архітектури x64

Edition Windows 11 Home Version

IDE Microsoft Visual Studio Enterprise 2022 (64-bit) version 17.12.3

## Теоретичні відомості

Використання вершинних масивів зменшує кількість викликів функцій та надмірне використання спільних вершин. Це дозволяє підвищити продуктивність рендерингу для завдань, таких як побудова фігури згідно з варіантом 16.

Для побудови фігури в вашому варіанті використовуються примітиви: GL\_POINTS та GL\_LINES.

Фігура побудована на основі масиву координат вершин, які визначаються у коді за допомогою класу Figura. Для відображення осей, сітки та фігури використовуються класи Layout та Figura.

Функції OpenGL для роботи з масивами

OpenGL надає функції, які дозволяють оптимізувати роботу з масивами даних:

glEnableClientState() та glDisableClientState() — для активації та деактивації масивів.

glVertexPointer() — вказує на масив координат вершин.

glDrawArrays() — дозволяє малювати примітиви за масивом вершин.

У вашому коді використовується масив координат вершин для побудови фігури та функція glDrawArrays(). Це зменшує кількість викликів функцій glVertex\*().

Приклад використання glDrawArrays()

Функція glDrawArrays() приймає три аргументи:

Тип примітива (у вашому випадку це GL\_LINES або GL\_POINTS).

Початок читання масиву (індекс у масиві).

Кількість вершин, які потрібно передати до конвеєра рендерингу.

Використання glDrawArrays() дозволяє оптимізувати побудову фігури за допомогою масиву вершин. У вашому варіанті ця функція застосовується для побудови як фігури (лінії), так і окремих точок.

Оптимізація з glDrawElements()

Хоча glDrawArrays() є ефективним методом, він вимагає повторення координат спільних вершин. Для зменшення кількості даних, які передаються, можна використовувати glDrawElements(). Ця функція дозволяє передавати унікальні вершини та використовувати індекси для посилань на них. Це може зменшити обсяг даних у масиві, але у вашому завданні це необов'язково.

## Результати виконання практичної роботи

### Розв'язання завдання

Результати виконання практичної роботи

Розв'язання завдання

Для управління параметрами графічних примітивів у варіанті 16 було використано наступні команди (Додаток А):

Колір: glColor3d(), рядки 68–69 у файлі Figura.cs.

Тип примітивів: glBegin(), glEnd(), glEnable()/glDisable() — реалізовано у файлі Figura.cs для GL\_POINTS та GL\_LINES.

Товщина ліній та розмір точок: glLineWidth() та glPointSize(), рядки 66 та 67 у файлі Figura.cs.

Коректність відображення

Коректне відображення завдання під час змінення розмірів/положення вікна продемонстровано на рисунках 1.1 та 1.2 (див. Додаток Б).

Оптимізація коду

Розроблення підпрограм для виключення дублювання коду наведено у рядках 78–92 файлу Figura.cs.

Застосування циклів для створення зображень реалізовано у рядках 84–92 файлу Layout.cs.

Використання ООП

Об’єктно-орієнтований підхід реалізовано шляхом створення власних класів:

Layout (для побудови осей координат та сітки);

Figura (для малювання фігури).

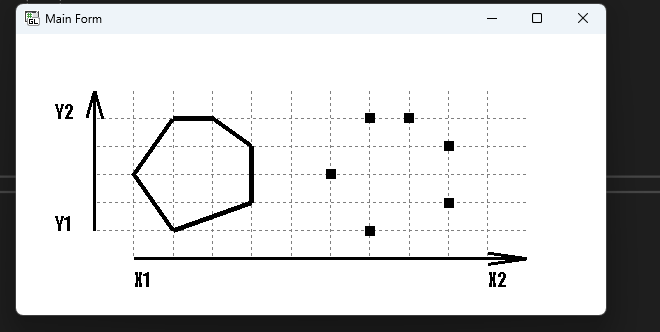


Рисунок 1.1 – Тестування програми при зміні ширини вікна

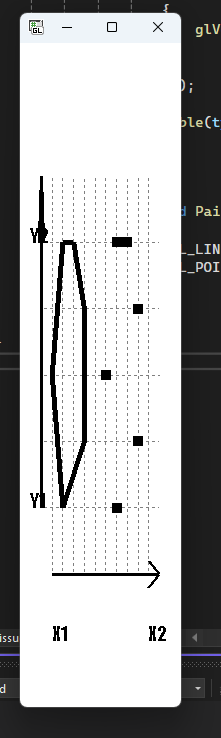


Рисунок 1.2 – Тестування програми при зміні висоти вікна

### 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 | **+** |

# Загальний перелік посилань

**Загальний перелік посилань**

1 Репозіторій GitHub <https://github.com/RostyslavNikolaiev/Task01/tree/main>

2 Офіційна документація OpenGL:

3 Методичні матеріали з основ OpenGL:

# Додаток А. Лістинг програми до практичної роботи №1

### Код файлу (MainForm.cs)

1. namespace Nikolaiev\_task01

* {
* partial class MainForm
* {
* private System.ComponentModel.IContainer components = null;
* /// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>
* protected override void Dispose(bool disposing)
* {
* if (disposing && (components != null))
* {
* components.Dispose();
* }
* base.Dispose(disposing);
* }
* #region Windows Form Designer generated code
* /// <summary>
* /// Required method for Designer support - do not modify
* /// the contents of this method with the code editor.
* /// </summary>
* private void InitializeComponent()
* {
* System.ComponentModel.ComponentResourceManager resources = new System.ComponentModel.ComponentResourceManager(typeof(MainForm));
* renderControl1 = new RenderControl();
* SuspendLayout();
* //
* // renderControl1
* //
* renderControl1.BackColor = System.Drawing.Color.Transparent;
* renderControl1.Dock = System.Windows.Forms.DockStyle.Fill;
* renderControl1.Font = new System.Drawing.Font("Impact", 14.25F, System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point);
* renderControl1.ForeColor = System.Drawing.Color.White;
* renderControl1.Location = new System.Drawing.Point(0, 0);
* renderControl1.Name = "renderControl1";
* renderControl1.Size = new System.Drawing.Size(543, 281);
* renderControl1.TabIndex = 0;
* renderControl1.TextCodePage = 65001;
* //
* // MainForm
* //
* AutoScaleDimensions = new System.Drawing.SizeF(10F, 19F);
* AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
* ClientSize = new System.Drawing.Size(543, 281);
* Controls.Add(renderControl1);
* Font = new System.Drawing.Font("Play", 12F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point);
* Icon = (System.Drawing.Icon)resources.GetObject("$this.Icon");
* Margin = new System.Windows.Forms.Padding(4);
* Name = "MainForm";
* Text = "Main Form";
* ResumeLayout(false);
* }
* #endregion
* private RenderControl renderControl1;
* }
* }

### Код файлу (Figura.cs)

1. using System;

* using System.Buffers;
* using System.Collections.Generic;
* using System.Diagnostics;
* using System.Drawing;
* using System.Linq;
* using System.Text;
* using System.Threading.Tasks;
* namespace Nikolaiev\_task01
* {
* public partial class RenderControl
* {
* internal class Figura
* {
* private readonly int[] \_cordX;
* private readonly int[] \_\_cordY;
* public Figura(int[] coordinatesX, int[] coordinatesY)
* {
* \_cordX = coordinatesX;
* \_\_cordY = coordinatesY;
* }
* public void Paint(uint type, int difference = 0, bool complex = false)
* {
* glLineWidth(5);
* glPointSize(10);
* glColor3d(0, 0, 0);
* glBegin(type);
* if (type == GL\_LINES)
* {
* for (int i = 0; i < \_cordX.Length; i++)
* {
* glVertex2d(\_cordX[i]+difference, \_\_cordY[i]);
* glVertex2d(\_cordX[(i + 1) % \_cordX.Length]+difference, \_\_cordY[(i + 1) % \_\_cordY.Length]);
* }
* }
* else
* {
* for (int i = 0; i < \_cordX.Length; i++)
* {
* glVertex2d(\_cordX[i]+difference, \_\_cordY[i]);
* }
* }
* glEnd();
* glDisable(type);
* }

* public void Paintall()
* {
* Paint(GL\_LINES);
* Paint(GL\_POINTS, 5);
* }
* }
* }
* }

**Layout.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Numerics;

using System.Security.Cryptography.X509Certificates;

using System.Text;

using System.Threading.Tasks;

using static Nikolaiev\_task01.OpenGL;

namespace Nikolaiev\_task01

{

public partial class RenderControl

{

internal class Layout

{

private readonly int \_x1;

private readonly int \_y1;

private readonly int \_x2;

private readonly int \_y2;

private readonly float \_step;

public Layout(int x1, int y1, int x2, int y2, float step)

{

\_x1 = x1;

\_y1 = y1;

\_x2 = x2;

\_y2 = y2;

\_step = step;

}

public void DrawAxis()

{

glLineWidth(3);

glColor3d(0, 0, 0);

glBegin(GL\_LINES);

glVertex2d(\_x1, \_y1-1);

glVertex2d(\_x2, \_y1-1);

glVertex2d(\_x1-1, \_y1);

glVertex2d(\_x1-1, \_y2);

PaintArrowY();

PaintArrowX();

glEnd();

}

public void PaintArrowY()

{

glLineWidth(3);

glColor3d(0, 0, 0);

glBegin(GL\_LINES);

glVertex2d(\_x1-1, \_y2);

glVertex2d(\_x1-1, \_y2+1);

glVertex2d(\_x1-1.2, \_y2);

glVertex2d(\_x1-1, \_y2+1);

glVertex2d(\_x1-0.8, \_y2);

glVertex2d(\_x1-1, \_y2+1);

glEnd();

}

public void PaintArrowX()

{

glLineWidth(3);

glColor3d(0, 0, 0);

glBegin(GL\_LINES);

glVertex2d(\_x2, \_y1-1);

glVertex2d(\_x2+1, \_y1-1);

glVertex2d(\_x2, \_y1-0.8);

glVertex2d(\_x2+1, \_y1-1);

glVertex2d(\_x2, \_y1-1.2);

glVertex2d(\_x2+1, \_y1-1);

glEnd();

}

public void PaintGrid()

{

glColor3d(0.5, 0.5, 0.5);

glLineWidth(1);

glEnable(GL\_LINE\_STIPPLE);

glLineStipple(3, 0xAAAA);

glBegin(GL\_LINES);

for (float x = \_x1; x <= \_x2; x += \_step)

{

glVertex2f(x, \_y1-1);

glVertex2f(x, \_y2+1);

}

for (float y = \_y1; y <= \_y2; y += \_step)

{

glVertex2f(\_x1-1, y);

glVertex2f(\_x2+1, y);

}

glEnd();

glDisable(GL\_LINE\_STIPPLE);

}

public void LoadLayout()

{

PaintGrid();

DrawAxis();

}

}

}

}