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  *336*

напряму підготовки (спеціальності):

*126 комп'ютерні науки*

(шифр і назва напряму підготовки / спеціальності)

*Давиденко Костянтин Ігорович*

(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

Завдання, варіант № 30 3

Системна інформація 3

Теоретичні відомості 3

Результати виконання практичної роботи 5

Загальний перелік посилань 9

Додаток А. Лістинг програми до практичної роботи №1 10

**Практична робота 1.  
Основні принципи роботи з OpenGL**

**Завдання, варіант № 30**

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.

|  |  |  |
| --- | --- | --- |
| 30 | Примітиви:  GL\_POINTS, GL\_LINE\_LOOP  x1 = -7; x2 = 2  y1 = -1; y2 = 3 |  |

**Системна інформація**

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

Processor Intel(R) Core(TM) i5-10300H CPU @ 2.50GHz

RAM 16.0 GB

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

Edition Windows 11 Home Version 23H2

IDE Microsoft Visual Studio Community 2019 (64-bit) version 16.11.41

**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***



Рис. 1 Результат виконання роботи

***Контроль виконання вимог та елементів завдання***

В результаті виконання практичної роботи були частково виконані елементи базового 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 | + |

**Практична робота 2.  
ГРАФІЧНІ ПРИМІТИВИ OPENGL**

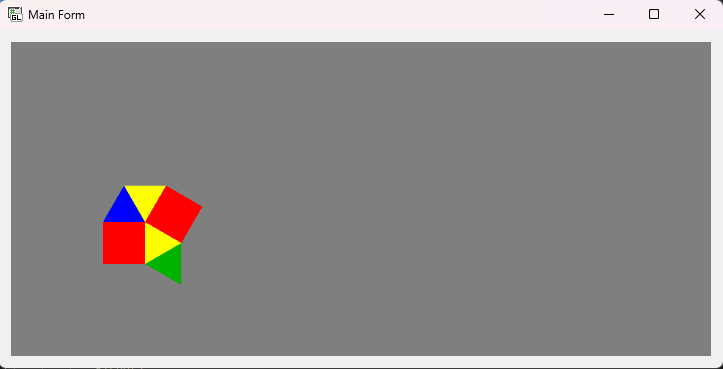
**Завдання, варіант № 5**

|  |  |  |
| --- | --- | --- |
| 5 | Сторона фігури a = 12  Примітив(и):  GL\_TRIANGLES,  GL\_QUADS |  |

Використовуючи інструментальні засоби, що вказані викладачем, і беручи до уваги вимого, що наведено в табл. 2.1, 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 (***tile***).

**Результати виконання практичної роботи**

***Розв'язання завдання***



***Контроль виконання вимог та елементів завдання***

В результаті виконання практичної роботи були частково виконані елементи базового рівня та частково підвищеного рівня складності, що відображено в таблиці 2.1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблиця 2.1 | | | | |
| № з/п | Складність | Вимоги | Бали | Зроблено |
| 1 | Базовий рівень | |  | | --- | | Під час запуску застосунку зображення відповідає варіанту завдання з однією плиткою (див. рис. 2.1) | | 1 | + |
| 2 | |  | | --- | | Багаторазове замощення плиткою (див. рис.2.2). Кратність замощення задається користувачем під час роботи застосунку. | | 1 | - |
| 3 | |  | | --- | | Коректне відображення завдання під час зміни як розмірів/положення вікна, так і параметрів замощення | | 1 | + |
| 4 | |  | | --- | | Організація взаємодії з користувачем одним зі стандартних засобів (клавіатура, «миша» та ін.) | | 1 | - |
| 5 | |  | | --- | | Застосування мінімальної (у рамках варіанту) кількості графічних примітивів для виконання завдання | | 1 | + |
| 6 | Підвищений рівень | Створення власних елементів інтерфейсу за допомогою OpenGL | 2 | - |
| 7 | Використання ООП (розроблення власних класів) | 1 | + |

**Практична робота 3.  
ГРАФІК ФУНКЦІЇ ОДНІЄЇ ЗМІННОЇ**

**Завдання, варіант № 5**

Використовуючи інструментальні засоби, що вказані викладачем, розробити програму для побудови графіка функції виду 𝑦=𝑓(𝑥) на довільному інтервалі від ***Xmin*** до ***Xmax*** і відображення точок перетину функції з віссю абсцис. Крім того, програма повинна мати такі можливості:

− дозволяти користувачу задавати інтервал від ***Xmin*** до ***Xmax*** з перевіркою ***Xmin < Xmax***;

− виконувати для завданого користувачем інтервалу від ***Xmin*** до ***Xmax*** автоматичне масштабування за віссю ***Y*** (додатково допускається наявність ручного режиму встановлення ***Ymin*** і ***Ymax***);

− відображати осі координат (та/або координатну сітку) з виводом значень меж видимої області ***Xmin***, ***Xmax***, ***Ymin*** і ***Ymax***, при цьому система координат повинна бути анізотропною;

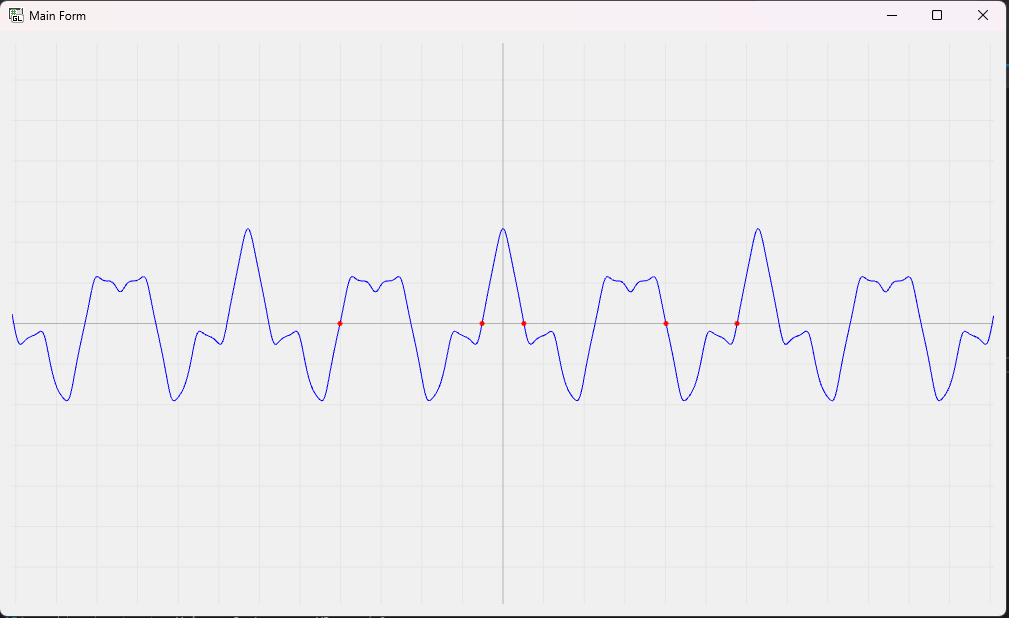
− відображати усі точки, де 𝑓(𝑥)=0, якщо вони є на завданому інтервалі від ***Xmin*** до ***Xmax***.

Для підвищеного рівня складності необхідно додатково реалізувати коректне виведення функції 𝑓2(𝑥) з урахуванням області визначення функції (варіанти наведені в табл. 3.3) і відобразити лінії розриву (рис. 3.2).

|  |  |
| --- | --- |
| 5 | f1(x) = tg (cos 2x) + (tg(cos 5x) / 2) |

**Результати виконання практичної роботи**

***Розв'язання завдання***



***Контроль виконання вимог та елементів завдання***

В результаті виконання практичної роботи були частково виконані елементи базового рівня та частково підвищеного рівня складності, що відображено в таблиці 3.1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблиця 3.1 | | | | |
| № з/п | Складність | Вимоги | Бали | Зроблено |
| 1 | Базовий рівень | |  |  | | --- | --- | | |  | | --- | | Осі координат і графік функції ***f1(x)*** виводяться на заданому користувачем інтервалі від *Xmin* до *Xmax* і від *Ymin* до *Ymax* | | | 1 | + |
| 2 | |  |  | | --- | --- | | |  | | --- | | Автоматичні обчислення *Ymin* і *Ymax* на завданому інтервалі від *Xmin* до *Xmax* функції ***f1(x)*** | | | 2 | - |
| 4 | |  |  | | --- | --- | | |  | | --- | | Обчислення і виведення на екран точок *f1(x)* = 0 | | | 2 | + |
| 6 | Підвищений рівень | |  | | --- | | Коректне виведення графіка ***f2(x)*** (без хибного виводу точок розриву як точок перетину з віссю абсцис) і з відображенням ліній розриву функції | | 2 | + |
| 7 | |  | | --- | | Використання ООП (наслідування, використання віртуальних і абстрактних методів) | | 1 | + |

**Практична робота 7.  
ЕКРАННА ЗАСТАВКА З АНІМАЦІЄЮ**

**Завдання**

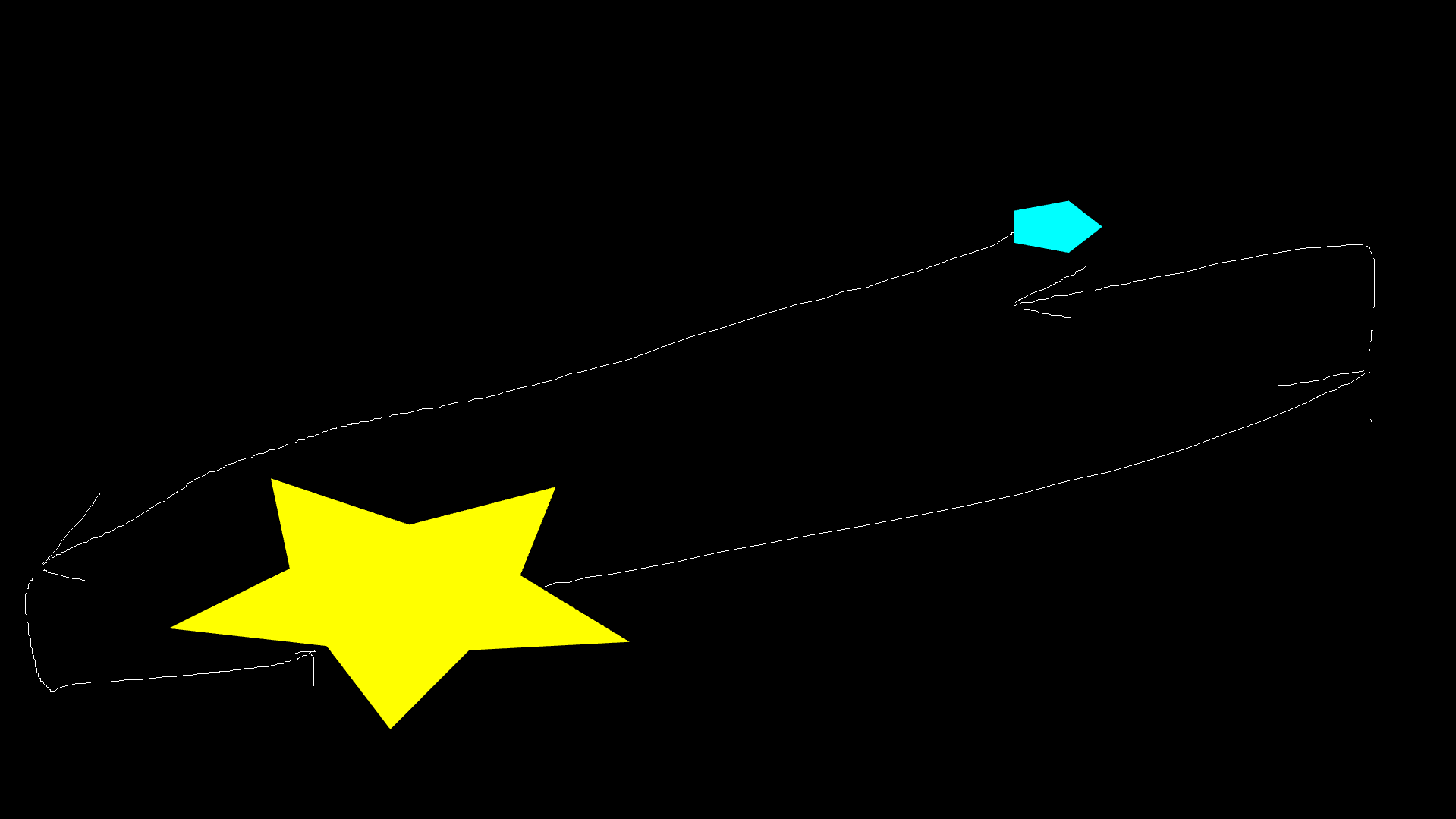
Використовуючи засоби, указані викладачем, створити програму

екранної заставки (ScreenSaver) з анімацією. Сюжет анімації

обрати самостійно і погодити з викладачем.

**Результати виконання практичної роботи**

***Розв'язання завдання***



***Контроль виконання вимог та елементів завдання***

В результаті виконання практичної роботи були частково виконані елементи базового рівня та частково підвищеного рівня складності, що відображено в таблиці 3.1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблиця 4.1 | | | | |
| № з/п | Складність | Вимоги | Бали | Зроблено |
| 1 | Базовий рівень | |  | | --- | | Прив’язка анімації до механізму простою  операційної системи | | 2 | + |
| 2 | |  | | --- | | Реалізація повноекранного режиму  (команда «/s») | | 1 | + |
| 3 | |  | | --- | | Реалізація налаштування програми Screen  Saver (команда «/c») | |  | | 1 | + |
| 4 | |  | | --- | | Реалізація попереднього перегляду (команда  «/p») | | 1 | + |
| 6 | Підвищений рівень | Збереження конфігурації і налаштувань  програми ScreenSaver у реєстрі ОС | 1 | - |
| 7 | Використання складних і видовищних  алгоритмів для формування зображення  (наприклад, фрактали) | 2 | - |

**Лістинг програми до практичної роботи №1**

***Код файлу (MainForm.cs)***

public partial class RenderControl : OpenGL

{

public RenderControl()

{

InitializeComponent();

}

private void CoordinateSystem(object sender, EventArgs e)

{

glClearColor(0.95f, 0.95f, 0.95f, 0f);

glClear(GL\_COLOR\_BUFFER\_BIT);

glLoadIdentity();

glViewport(0, 0, Width, Width);

glOrtho2d(-0.1, +1, -0.1, +0.6);

DrawGrid();

DrawAxes();

DrawPointsFigure();

DrawLineStripFigure();

}

private void DrawGrid()

{

glEnable(GL\_LINE\_STIPPLE);

glLineStipple(1, 0x00FF);

glColor3f(0.7f, 0.7f, 0.7f);

glLineWidth(2);

for (int i = 1; i < 11; ++i)

{

DrawLine(0, 0.1f \* i, 1, 0.1f \* i);

}

for (int i = 1; i < 11; ++i)

{

DrawLine(0.1f \* i, 1, 0.1f \* i, 0);

}

glDisable(GL\_LINE\_STIPPLE);

}

private void DrawAxes()

{

glLineWidth(1);

glColor3i(0, 0, 0);

DrawLine(-0.1f, 0, 1, 0);

DrawLine(0, -0.1f, 0, 1);

for (int i = 1; i < 11; ++i)

{

DrawLine(0.1f \* i, -0.03f, 0.1f \* i, 0);

}

for (int i = 1; i < 11; ++i)

{

DrawLine(-0.03f, 0.1f \* i, 0f, 0.1f \* i);

}

}

private void DrawLine(float x1, float y1, float x2, float y2)

{

glBegin(GL\_LINES);

glVertex2f(x1, y1);

glVertex2f(x2, y2);

glEnd();

}

private void DrawLineStripFigure()

{

glEnable(GL\_LINE\_STRIP);

glLineWidth(2);

glColor3f(0, 0, 0);

glBegin(GL\_LINE\_STRIP);

glVertex2d(0.2, 0.2);

glVertex2d(0.3, 0.1);

glVertex2d(0.4, 0.2);

glVertex2d(0.3, 0.4);

glVertex2d(0.2, 0.5);

glVertex2d(0.1, 0.4);

glVertex2d(0.2, 0.2);

glEnd();

}

private void DrawPointsFigure()

{

glEnable(GL\_POINTS);

glPointSize(7);

glColor3f(0, 0, 0);

glBegin(GL\_POINTS);

glVertex2d(0.7, 0.2);

glVertex2d(0.8, 0.1);

glVertex2d(0.9, 0.2);

glVertex2d(0.6, 0.4);

glVertex2d(0.7, 0.5);

glVertex2d(0.8, 0.4);

glEnd();

}

}

**Лістинг програми до практичної роботи №2**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Diagnostics;

using System.Drawing;

using System.Drawing.Design;

using System.Linq;

namespace GLR2

{

public partial class RenderControl : OpenGL

{

public RenderControl()

{

InitializeComponent();

}

float a = 0.12f;

private void OnRender(object sender, EventArgs e)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glClearColor(0.5f, 0.5f, 0.5f, 0f);

glLoadIdentity();

glViewport(-300, -300, Width, Width);

DrawQuads();

glPushMatrix();

glTranslatef(a \* 2, a \* 2, 0);

glRotatef(150, 0, 0, 1);

glTranslatef(-a \* 2, -a \* 2, 0);

DrawQuads();

glPopMatrix();

glPushMatrix();

glColor3f(1, 1, 0);

DrawTriangles();

glPopMatrix();

glPushMatrix();

glColor3d(0, 0, 1);

glTranslatef(a \* 1.5f, a \* 1.5f, 0);

glRotatef(90, 0, 0, 1);

glTranslatef(-a \* 1.5f, -a \* 1.5f, 0);

DrawTriangles();

glPopMatrix();

glPushMatrix();

glColor3f(1, 1, 0);

glTranslatef(a \* 2, a \* 2, 0);

glRotatef(150, 0, 0, 1);

glTranslatef(-a \* 2, -a \* 2, 0);

DrawTriangles();

glPopMatrix();

glPushMatrix();

glColor3f(0, 0.7f, 0);

glTranslatef(a \* 2, a, 0);

glRotatef(-60, 0, 0, 1);

glTranslatef(-a \* 2, -a, 0);

DrawTriangles();

glPopMatrix();

}

private void DrawQuads()

{

glBegin(GL\_QUADS);

glColor3f(1, 0, 0);

glVertex2f(a, a);

glVertex2f(2 \* a, a);

glVertex2f(2 \* a, 2 \* a);

glVertex2f(a, 2 \* a);

glEnd();

}

private void DrawTriangles()

{

glBegin(GL\_TRIANGLES);

glVertex2f(a \* 2, a \* 2);

glVertex2f(a \* 2, a);

glVertex2f(a \* 2 + a \* (float)Math.Sqrt(3) / 2, a \* 1.5f);

glEnd();

}

}

}

**Лістинг програми до практичної роботи №3**

using System;

using System.Drawing;

using System.Windows.Forms;

namespace GLR3

{

public partial class RenderControl : OpenGL

{

private double Xmin = -10, Xmax = 10;

private double Ymin, Ymax;

private bool autoScale = true;

private double step = 0.01; // Крок для побудови графіка

public RenderControl()

{

InitializeComponent();

Resize += (s, e) => Refresh();

}

public void SetInterval(double xmin, double xmax)

{

if (xmin >= xmax)

throw new ArgumentException("Xmin має бути менше за Xmax!");

Xmin = xmin;

Xmax = xmax;

if (autoScale)

CalculateYBounds();

}

public void SetYBounds(double ymin, double ymax)

{

if (ymin >= ymax)

throw new ArgumentException("Ymin має бути менше за Ymax!");

Ymin = ymin;

Ymax = ymax;

autoScale = false;

}

public void EnableAutoScale(bool enable)

{

autoScale = enable;

if (enable)

CalculateYBounds();

}

private void CalculateYBounds()

{

double ymin = double.MaxValue, ymax = double.MinValue;

for (double x = Xmin; x <= Xmax; x += step)

{

double y = Function(x);

if (y < ymin) ymin = y;

if (y > ymax) ymax = y;

}

Ymin = ymin - 1; // Трохи відступів для масштабу

Ymax = ymax + 1;

}

private double Function(double x)

{

return Math.Tan(Math.Cos(2 \* x)) + 0.5 \* Math.Tan(Math.Cos(5 \* x));

}

private void DrawAxes()

{

glColor3d(0.7, 0.7, 0.7);

glBegin(GL\_LINES);

// X-Axis

glVertex2d(Xmin, 0);

glVertex2d(Xmax, 0);

// Y-Axis

glVertex2d(0, Ymin);

glVertex2d(0, Ymax);

glEnd();

}

private void DrawGrid()

{

glColor3d(0.9, 0.9, 0.9);

glBegin(GL\_LINES);

for (double x = Math.Ceiling(Xmin); x <= Xmax; x++)

{

glVertex2d(x, Ymin);

glVertex2d(x, Ymax);

}

for (double y = Math.Ceiling(Ymin); y <= Ymax; y++)

{

glVertex2d(Xmin, y);

glVertex2d(Xmax, y);

}

glEnd();

}

private void DrawFunction()

{

glColor3d(0, 0, 1);

glBegin(GL\_LINE\_STRIP);

for (double x = Xmin; x <= Xmax; x += step)

{

double y = Function(x);

glVertex2d(x, y);

}

glEnd();

}

private void DrawZeros()

{

glColor3d(1, 0, 0);

glPointSize(5);

glBegin(GL\_POINTS);

for (double x = Xmin; x <= Xmax; x += step)

{

double y = Function(x);

if (Math.Abs(y) < 1e-2) // Умовно вважаємо нульовим

{

glVertex2d(x, 0);

}

}

glEnd();

}

private void OnRender(object sender, EventArgs e)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

// Розрахунок ізотропного масштабу

int viewportWidth = Width;

int viewportHeight = Height;

double aspect = (double)viewportWidth / viewportHeight;

double xRange = Xmax - Xmin;

double yRange = Ymax - Ymin;

double xCenter = (Xmax + Xmin) / 2;

double yCenter = (Ymax + Ymin) / 2;

// Визначення масштабу для ізотропності

if (xRange / yRange > aspect)

{

double newYRange = xRange / aspect;

Ymin = yCenter - newYRange / 2;

Ymax = yCenter + newYRange / 2;

}

else

{

double newXRange = yRange \* aspect;

Xmin = xCenter - newXRange / 2;

Xmax = xCenter + newXRange / 2;

}

glViewport(0, 0, viewportWidth, viewportHeight);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(Xmin, Xmax, Ymin, Ymax);

DrawGrid();

DrawAxes();

DrawFunction();

DrawZeros();

}

}

}

**Лістинг програми до практичної роботи №3**

**using** System;

using System.Drawing;

using System.Windows.Forms;

namespace LR7

{

public partial class RenderControl : OpenGL

{

public RenderControl()

{

InitializeComponent();

}

private double angle = 0;

private double dx = 0.01, dy = 0.01;

private double xPos = 0, yPos = 0;

private void OnContextCreated(object sender, EventArgs e)

{

glClearColor(Color.Black);

glEnable(GL\_DEPTH\_TEST);

glShadeModel(GL\_SMOOTH);

}

private void OnRender(object sender, EventArgs e)

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glLoadIdentity();

glViewport(0, 0, Width, Height);

glOrtho(-1.5, 1.5, -1.5, 1.5, -1, 1);

angle += 1;

if (angle > 360) angle -= 360;

xPos += dx;

yPos += dy;

if (xPos > 1 || xPos < -1) dx = -dx;

if (yPos > 1 || yPos < -1) dy = -dy;

glPushMatrix();

glTranslated(xPos, yPos, 0);

glRotated(angle, 0, 0, 1);

DrawStar(0.5, 5);

glPopMatrix();

glPushMatrix();

glTranslated(-xPos, -yPos, 0);

glScaled(0.5 + 0.25 \* Math.Sin(angle \* Math.PI / 180), 0.5 + 0.25 \* Math.Sin(angle \* Math.PI / 180), 1);

DrawPolygon(0.4, 5);

glPopMatrix();

}

private void DrawStar(double radius, int points)

{

glColor3d(1, 1, 0);

glBegin(GL\_TRIANGLE\_FAN);

glVertex2d(0, 0);

for (int i = 0; i <= points \* 2; i++)

{

double angle = Math.PI \* i / points;

double r = (i % 2 == 0) ? radius : radius / 2;

glVertex2d(r \* Math.Cos(angle), r \* Math.Sin(angle));

}

glEnd();

}

private void DrawPolygon(double radius, int sides)

{

glColor3d(0, 1, 1);

glBegin(GL\_POLYGON);

for (int i = 0; i < sides; i++)

{

double angle = 2 \* Math.PI \* i / sides;

glVertex2d(radius \* Math.Cos(angle), radius \* Math.Sin(angle));

}

glEnd();

}

}

}