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  *535а*

напряму підготовки (спеціальності):

*124 системне програмування*

(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)

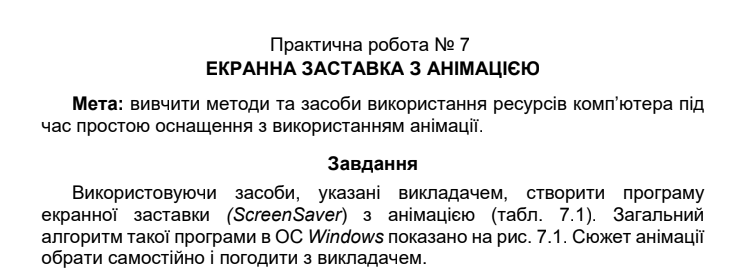
Національна шкала:

Кількість балів:

Оцінка ECTS:

# Практична робота 7. ГРАФІЧНІ ПРИМІТИВИ OPENGL

## Завдання, варіант № 1



## 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)

**Код**

using System;

using System.Drawing;

using System.Collections.Generic;

using System.Windows.Forms;

using Microsoft.Win32; // Для работы с реестром

namespace Lab\_7

{

public partial class RenderControl : OpenGL

{

private List<MovingShape> shapes = new List<MovingShape>();

private Random random = new Random();

private double boundaryX = 10.0;

private double boundaryY = 10.0;

private const double boundaryZ = 10.0;

private const string RegistryKey = "Software\\Lab\_7.exe";

public RenderControl()

{

InitializeComponent();

UpdateBoundaries();

Resize += OnResize;

}

#region Методы сохранения/загрузки настроек

public void SaveSettingsToRegistry(string key, string value)

{

using (var registryKey = Registry.CurrentUser.CreateSubKey(RegistryKey))

{

registryKey?.SetValue(key, value);

}

}

public string LoadSettingsFromRegistry(string key)

{

using (var registryKey = Registry.CurrentUser.OpenSubKey(RegistryKey))

{

return registryKey?.GetValue(key)?.ToString() ?? string.Empty;

}

}

public static void SaveSettingsToRegistryStatic(string key, string value)

{

using (var registryKey = Registry.CurrentUser.CreateSubKey(RegistryKey))

{

registryKey?.SetValue(key, value);

}

}

public static string LoadSettingsFromRegistryStatic(string key)

{

using (var registryKey = Registry.CurrentUser.OpenSubKey(RegistryKey))

{

return registryKey?.GetValue(key)?.ToString() ?? string.Empty;

}

}

#endregion

#region Инициализация OpenGL и загрузка фигур

private void OnContextCreated(object sender, EventArgs e)

{

// Включаем z-буфер

glEnable(GL\_DEPTH\_TEST);

// Фоновый цвет

glClearColor(Color.Black);

// Генерируем фигуры (учитывая настройки)

InitializeShapes();

}

private void InitializeShapes()

{

// Считываем ShapeCount

var shapeCountString = LoadSettingsFromRegistryStatic("ShapeCount");

int shapeCount = 300; // по умолчанию

if (!string.IsNullOrEmpty(shapeCountString) && int.TryParse(shapeCountString, out int parsedShapeCount))

{

shapeCount = parsedShapeCount;

}

// Считываем AnimationSpeed

var animationSpeedString = LoadSettingsFromRegistryStatic("AnimationSpeed");

double animationSpeed = 1.0;

if (!string.IsNullOrEmpty(animationSpeedString) && double.TryParse(animationSpeedString, out double parsedSpeed))

{

animationSpeed = parsedSpeed;

}

// Очищаем старые

shapes.Clear();

// Генерируем шары/кубы/пирамиды/фракталы/сложные треугольники

// enum ShapeType имеет 5 вариантов (0..4),

// так что random.Next(0, 5) даёт числа [0..4].

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

{

var st = (ShapeType)random.Next(0, 5);

// (Cube=0, Sphere=1, Pyramid=2, Fractal=3, ComplexTriangle=4)

var shape = new MovingShape

{

X = random.NextDouble() \* boundaryX \* 2 - boundaryX,

Y = random.NextDouble() \* boundaryY \* 2 - boundaryY,

Z = random.NextDouble() \* boundaryZ \* 2 - boundaryZ,

DX = (random.NextDouble() \* 0.2 - 0.1) \* animationSpeed,

DY = (random.NextDouble() \* 0.2 - 0.1) \* animationSpeed,

DZ = (random.NextDouble() \* 0.2 - 0.1) \* animationSpeed,

Size = random.NextDouble() \* 1.5 + 0.3,

Rotation = random.NextDouble() \* 360.0,

RotationSpeed = random.NextDouble() \* 3.0 - 1.5,

Color = new double[]

{

random.NextDouble(),

random.NextDouble(),

random.NextDouble()

},

ShapeType = st,

// Случайная ось вращения (RotAxisX/Y/Z).

RotAxisX = random.NextDouble() \* 2 - 1, // от -1 до +1

RotAxisY = random.NextDouble() \* 2 - 1,

RotAxisZ = random.NextDouble() \* 2 - 1

};

// Нормализуем ось, чтобы не было проблем с очень маленькой длиной

double length = Math.Sqrt(shape.RotAxisX \* shape.RotAxisX

+ shape.RotAxisY \* shape.RotAxisY

+ shape.RotAxisZ \* shape.RotAxisZ);

if (length < 0.0001) length = 1.0;

shape.RotAxisX /= length;

shape.RotAxisY /= length;

shape.RotAxisZ /= length;

shapes.Add(shape);

}

}

#endregion

#region Отрисовка

private void OnRender(object sender, EventArgs e)

{

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

glLoadIdentity();

// Настройка проекции

glViewport(0, 0, Width, Height);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(45.0, (double)Width / Height, 0.1, 100.0);

glMatrixMode(GL\_MODELVIEW);

// "Камеру" двинем подальше

glTranslated(0.0, 0.0, -50.0);

// Рисуем всё

DrawMovingShapes();

DrawBoundaryLimits();

}

private void DrawMovingShapes()

{

foreach (var shape in shapes)

{

// Двигаем фигуру

shape.X += shape.DX;

shape.Y += shape.DY;

shape.Z += shape.DZ;

shape.Rotation += shape.RotationSpeed;

// Проверяем границы (отражение)

if (shape.X - shape.Size < -boundaryX)

{

shape.X = -boundaryX + shape.Size;

shape.DX = -shape.DX;

}

if (shape.X + shape.Size > boundaryX)

{

shape.X = boundaryX - shape.Size;

shape.DX = -shape.DX;

}

if (shape.Y - shape.Size < -boundaryY)

{

shape.Y = -boundaryY + shape.Size;

shape.DY = -shape.DY;

}

if (shape.Y + shape.Size > boundaryY)

{

shape.Y = boundaryY - shape.Size;

shape.DY = -shape.DY;

}

if (shape.Z - shape.Size < -boundaryZ)

{

shape.Z = -boundaryZ + shape.Size;

shape.DZ = -shape.DZ;

}

if (shape.Z + shape.Size > boundaryZ)

{

shape.Z = boundaryZ - shape.Size;

shape.DZ = -shape.DZ;

}

// Отрисовка

glPushMatrix();

// Перемещаем в позицию

glTranslated(shape.X, shape.Y, shape.Z);

// Вращаем вокруг собственной оси (RotAxisX,Y,Z)

glRotated(shape.Rotation, shape.RotAxisX, shape.RotAxisY, shape.RotAxisZ);

// Цвет

glColor3d(shape.Color[0], shape.Color[1], shape.Color[2]);

// Тип фигуры

switch (shape.ShapeType)

{

case ShapeType.Cube:

DrawCube(shape.Size);

break;

case ShapeType.Sphere:

DrawSphere(shape.Size);

break;

case ShapeType.Pyramid:

DrawPyramid(shape.Size);

break;

case ShapeType.Fractal:

DrawFractal(shape.Size);

break;

case ShapeType.ComplexTriangle:

DrawComplexTriangle(shape.Size);

break;

}

glPopMatrix();

}

}

private void DrawBoundaryLimits()

{

glColor3d(1.0, 0.0, 0.0); // Красные "стенки"

glBegin(GL\_LINES);

// Линии по X

glVertex3d(-boundaryX, -boundaryY, 0);

glVertex3d(boundaryX, -boundaryY, 0);

glVertex3d(-boundaryX, boundaryY, 0);

glVertex3d(boundaryX, boundaryY, 0);

// Линии по Y

glVertex3d(-boundaryX, -boundaryY, 0);

glVertex3d(-boundaryX, boundaryY, 0);

glVertex3d(boundaryX, -boundaryY, 0);

glVertex3d(boundaryX, boundaryY, 0);

glEnd();

}

#endregion

#region Примитивы (куб, сфера, пирамида, фрактал, сложный треугольник)

private void DrawCube(double size)

{

glBegin(GL\_QUADS);

// Передняя грань

glVertex3d(-size, -size, size);

glVertex3d(size, -size, size);

glVertex3d(size, size, size);

glVertex3d(-size, size, size);

// Задняя грань

glVertex3d(-size, -size, -size);

glVertex3d(-size, size, -size);

glVertex3d(size, size, -size);

glVertex3d(size, -size, -size);

// Верх

glVertex3d(-size, size, -size);

glVertex3d(-size, size, size);

glVertex3d(size, size, size);

glVertex3d(size, size, -size);

// Низ

glVertex3d(-size, -size, -size);

glVertex3d(size, -size, -size);

glVertex3d(size, -size, size);

glVertex3d(-size, -size, size);

// Правая грань

glVertex3d(size, -size, -size);

glVertex3d(size, size, -size);

glVertex3d(size, size, size);

glVertex3d(size, -size, size);

// Левая грань

glVertex3d(-size, -size, -size);

glVertex3d(-size, -size, size);

glVertex3d(-size, size, size);

glVertex3d(-size, size, -size);

glEnd();

}

private void DrawSphere(double size)

{

const int stacks = 18;

const int slices = 36;

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

{

double latitude1 = Math.PI \* (-0.5 + (double)i / stacks);

double latitude2 = Math.PI \* (-0.5 + (double)(i + 1) / stacks);

double sinLat1 = Math.Sin(latitude1);

double cosLat1 = Math.Cos(latitude1);

double sinLat2 = Math.Sin(latitude2);

double cosLat2 = Math.Cos(latitude2);

glBegin(GL\_QUAD\_STRIP);

for (int j = 0; j <= slices; j++)

{

double longitude = 2 \* Math.PI \* (double)j / slices;

double sinLon = Math.Sin(longitude);

double cosLon = Math.Cos(longitude);

double x1 = cosLon \* cosLat1;

double y1 = sinLon \* cosLat1;

double z1 = sinLat1;

double x2 = cosLon \* cosLat2;

double y2 = sinLon \* cosLat2;

double z2 = sinLat2;

glVertex3d(size \* x1, size \* y1, size \* z1);

glVertex3d(size \* x2, size \* y2, size \* z2);

}

glEnd();

}

}

private void DrawPyramid(double size)

{

// Треугольные стороны

glBegin(GL\_TRIANGLES);

// Перед

glVertex3d(0, size, 0);

glVertex3d(-size, -size, size);

glVertex3d(size, -size, size);

// Правая

glVertex3d(0, size, 0);

glVertex3d(size, -size, size);

glVertex3d(size, -size, -size);

// Задняя

glVertex3d(0, size, 0);

glVertex3d(size, -size, -size);

glVertex3d(-size, -size, -size);

// Левая

glVertex3d(0, size, 0);

glVertex3d(-size, -size, -size);

glVertex3d(-size, -size, size);

glEnd();

// Основание (квадрат)

glBegin(GL\_QUADS);

glVertex3d(-size, -size, size);

glVertex3d(size, -size, size);

glVertex3d(size, -size, -size);

glVertex3d(-size, -size, -size);

glEnd();

}

/// <summary>

/// Пример простого фрактала (рекурсивный треугольник).

/// Можно заменить на любой более «эффектный» алгоритм.

/// </summary>

private void DrawFractal(double size)

{

// Для красоты чуть уменьшим

size \*= 2.0;

// Рисуем "треугольник Серпинского" в 2D-плоскости

glPushMatrix();

glRotated(-90, 1, 0, 0); // Повернём "плашмя", чтобы смотреть "сверху"

// Примитивный рекурсивный метод

SierpinskiTriangle(0, 0, size, 3);

glPopMatrix();

}

private void SierpinskiTriangle(double x, double y, double size, int level)

{

if (level <= 0)

{

return;

}

if (level == 1)

{

// Рисуем простой равносторонний треугольник

glBegin(GL\_TRIANGLES);

glVertex3d(x, y, 0);

glVertex3d(x + size, y, 0);

glVertex3d(x + size / 2, y + (Math.Sqrt(3) \* size / 2), 0);

glEnd();

}

else

{

// Делим треугольник на 3 поменьше

double half = size / 2;

// Треугольник слева

SierpinskiTriangle(x, y, half, level - 1);

// Треугольник справа

SierpinskiTriangle(x + half, y, half, level - 1);

// Треугольник вверху

SierpinskiTriangle(x + half / 2, y + (Math.Sqrt(3) \* half / 2), half, level - 1);

}

}

/// <summary>

/// Отдельная фигура - "сложный треугольник".

/// Вы можете добавить туда ещё треугольников, чтобы сделать его по-настоящему "сложным".

/// </summary>

private void DrawComplexTriangle(double size)

{

glBegin(GL\_TRIANGLES);

// Первый треугольник

glVertex3d(0, size, 0);

glVertex3d(-size, -size, size);

glVertex3d(size, -size, size);

// Второй треугольник

glVertex3d(0, size, 0);

glVertex3d(size, -size, size);

glVertex3d(size, -size, -size);

// Третий - для симметрии

glVertex3d(0, size, 0);

glVertex3d(size, -size, -size);

glVertex3d(-size, -size, -size);

// Четвёртый

glVertex3d(0, size, 0);

glVertex3d(-size, -size, -size);

glVertex3d(-size, -size, size);

glEnd();

}

#endregion

#region Служебное

private void OnResize(object sender, EventArgs e)

{

UpdateBoundaries();

}

private void UpdateBoundaries()

{

boundaryX = Width / 20.0;

boundaryY = Height / 20.0;

}

/// <summary>

/// Закрытие заставки по Esc

/// </summary>

protected override bool ProcessCmdKey(ref Message msg, Keys keyData)

{

if (keyData == Keys.Escape)

{

Application.Exit();

}

return base.ProcessCmdKey(ref msg, keyData);

}

// Класс, описывающий двигающуюся и вращающуюся фигуру.

private class MovingShape

{

public double X { get; set; }

public double Y { get; set; }

public double Z { get; set; }

public double DX { get; set; }

public double DY { get; set; }

public double DZ { get; set; }

public double Size { get; set; }

public double Rotation { get; set; }

public double RotationSpeed { get; set; }

public double[] Color { get; set; }

public ShapeType ShapeType { get; set; }

// Добавляем ось вращения

public double RotAxisX { get; set; }

public double RotAxisY { get; set; }

public double RotAxisZ { get; set; }

}

// Расширяем enum ShapeType:

private enum ShapeType

{

Cube,

Sphere,

Pyramid,

Fractal,

ComplexTriangle // новый пункт

}

#endregion

}

}

Program.cs

using System;

using System.Windows.Forms;

using System.Diagnostics;

namespace Lab\_7

{

/// <summary>

/// Главный класс скринсейвера (partial в исходном шаблоне).

/// Теперь собран в одном файле, чтобы не было конфликтов.

/// </summary>

public static partial class Program

{

/// <summary>

/// Режимы работы скринсейвера (из исходного шаблона).

/// </summary>

public enum Mode

{

Unknown,

Settings,

ModalSettings,

Preview,

Run

}

/// <summary>

/// Текущее состояние работы (устанавливается после парсинга командной строки).

/// </summary>

public static Mode mode = Mode.Unknown;

/// <summary>

/// Хендл родительского окна при /p <HWND> (Windows передаёт).

/// </summary>

public static IntPtr parentWnd = IntPtr.Zero;

/// <summary>

/// Хендл owner-окна при /c:<HWND> (Windows передаёт).

/// </summary>

public static IntPtr ownerWnd = IntPtr.Zero;

/// <summary>

/// Нужен, чтобы в OpenGL-контроле обращаться к Program.DesignMode.

/// Если не нужно — убираем все ссылки из OpenGL-кода и отсюда.

/// </summary>

public static bool DesignMode { get; set; } = false;

/// <summary>

/// Точка входа приложения (ScreenSaver).

/// </summary>

[STAThread]

static void Main(string[] cmd)

{

// Обычно для скринсейвера не включаем дизайн-режим

DesignMode = false;

// Стандартная настройка WinForms

Application.SetHighDpiMode(HighDpiMode.SystemAware);

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

// Парсим командную строку и определяем режим

mode = Selector(cmd);

// В зависимости от режима — делаем свою логику

switch (mode)

{

case Mode.Run:

// /s: запустить заставку (полноэкранную)

ShowSaver();

break;

case Mode.Preview:

// /p <HWND>: превью в родительском окошке

// (или своя логика превью)

ShowPreview();

break;

case Mode.Settings:

// /c без параметров: просто открыть окно настроек

Application.Run(new SettingsForm());

break;

case Mode.ModalSettings:

// /c:<HWND>: открыть настройки модально к ownerWnd

ShowModalSetting();

break;

default:

// Unknown — ругаемся или делаем что-то по умолчанию

if (cmd.Length > 0)

{

MessageBox.Show(

"Unknown command: " + cmd[0]

+ Environment.NewLine

+ "Use /s, /c, /p, etc.",

"Info"

);

}

else

{

MessageBox.Show(

"No command specified. Use /s, /c, /p, etc.",

"Info"

);

}

break;

}

}

/// <summary>

/// Разбор аргументов командной строки.

/// </summary>

internal static Mode Selector(string[] cmd)

{

// Если нет аргументов, пусть будет Settings

if (cmd == null || cmd.Length == 0)

return Mode.Settings;

string arg = cmd[0].Trim().ToLower();

// Если меньше 2 символов, тоже в Unknown

if (arg.Length < 2) return Mode.Unknown;

// Смотрим первые 2 символа: /s, /p, /c, ...

switch (arg.Substring(0, 2))

{

case "/s": return Mode.Run;

case "/p": return ModePreview(cmd);

case "/c": return ModeSetting(cmd);

// /a и т.д. (если нужно) — по аналогии

default: return Mode.Unknown;

}

}

/// <summary>

/// Обработка /p <HWND>

/// </summary>

private static Mode ModePreview(string[] cmd)

{

// Если просто "/p" без HWND — даём своё превью, без встраивания

if (cmd.Length == 1)

{

parentWnd = IntPtr.Zero;

return Mode.Preview;

}

// Иначе, если передали HWND, делаем стандартное встраивание

if (long.TryParse(cmd[1], out long val))

{

parentWnd = new IntPtr(val);

return Mode.Preview;

}

// Если не Parse-ится, считаем Unknown

return Mode.Unknown;

}

/// <summary>

/// Обработка /c[:<HWND>]

/// </summary>

private static Mode ModeSetting(string[] cmd)

{

// /c:123456

var c = cmd[0].Split(':');

if (c.Length == 1)

{

// Просто /c

return Mode.Settings;

}

// Пытаемся прочесть owner-окно

if (long.TryParse(c[1], out long h))

{

ownerWnd = new IntPtr(h);

return Mode.ModalSettings;

}

else

{

return Mode.Unknown;

}

}

/// <summary>

/// Запуск заставки (пример: полноэкранно на всех мониторах)

/// </summary>

internal static void ShowSaver()

{

Cursor.Hide();

foreach (Screen screen in Screen.AllScreens)

{

var screensaver = new MainForm();

screensaver.Bounds = screen.Bounds;

screensaver.Show();

}

// Запускаем обработку сообщений

Application.Run();

}

/// <summary>

/// Показываем настройки в «модальном» режиме к ownerWnd (при /c:<HWND>)

/// </summary>

internal static void ShowModalSetting()

{

WindowHandleWrapper wnd = new WindowHandleWrapper(ownerWnd);

var d = new SettingsForm();

if (d is Form)

d.ShowDialog(wnd);

}

/// <summary>

/// Логика для /p <HWND> (превью)

/// </summary>

private static void ShowPreview()

{

// Если parentWnd == IntPtr.Zero, это значит: "/p" без второго аргумента

if (parentWnd == IntPtr.Zero)

{

// 1) Покажем "маленькое окно-превью",

// например, размером 400×300, с рамкой и заголовком.

using (var previewForm = new Form()) // или MainForm(), как удобнее

{

previewForm.Text = "Preview ScreenSaver";

previewForm.Width = 400;

previewForm.Height = 300;

previewForm.StartPosition = FormStartPosition.CenterScreen;

// Вставим ваш RenderControl (или что там рисует заставку)

var previewControl = new RenderControl();

previewControl.Dock = DockStyle.Fill;

previewForm.Controls.Add(previewControl);

// Модально, чтобы пользователь не переключался

previewForm.ShowDialog();

}

}

else

{

// 2) Если parentWnd != IntPtr.Zero,

// это классическая схема "встраивания" в окно с хендлом parentWnd.

var previewForm = new MainForm

{

// Чтобы не иметь рамки

FormBorderStyle = FormBorderStyle.None,

TopLevel = false, // делаем форму не-верхнеуровневой

};

// Покажем, как дочерний элемент для parentWnd.

WindowHandleWrapper w = new WindowHandleWrapper(parentWnd);

previewForm.Show(w);

// Обычно для /p не вызывается Application.Run() второй раз,

// но если вы хотите запустить цикл сообщений, можно:

Application.Run();

}

}

}

}

SettingsForm.cs

using System;

using System.Diagnostics;

using System.Windows.Forms;

using Microsoft.Win32;

namespace Lab\_7

{

public partial class SettingsForm : Form

{

private const string RegistryKey = "Software\\Lab\_7.exe";

public SettingsForm()

{

InitializeComponent();

}

private void SettingsForm\_Load(object sender, EventArgs e)

{

Debug.WriteLine("Loading screen saver's settings...");

// Загрузка настроек из реестра

string animationSpeed = LoadSettingsFromRegistry("AnimationSpeed") ?? "1.0";

string shapeCount = LoadSettingsFromRegistry("ShapeCount") ?? "300";

// Создаём элементы управления (TextBox'ы) динамически

var txtAnimationSpeed = new TextBox

{

Text = animationSpeed,

Name = "txtAnimationSpeed",

Location = new System.Drawing.Point(20, 20),

Width = 200

};

var txtShapeCount = new TextBox

{

Text = shapeCount,

Name = "txtShapeCount",

Location = new System.Drawing.Point(20, 60),

Width = 200

};

// Надписи

this.Controls.Add(new Label

{

Text = "Animation Speed:",

Location = new System.Drawing.Point(20, 0),

AutoSize = true

});

this.Controls.Add(txtAnimationSpeed);

this.Controls.Add(new Label

{

Text = "Shape Count:",

Location = new System.Drawing.Point(20, 40),

AutoSize = true

});

this.Controls.Add(txtShapeCount);

// Кнопка "Save"

var btnSave = new Button

{

Text = "Save",

Location = new System.Drawing.Point(20, 100),

Width = 80

};

btnSave.Click += (s, ev) =>

{

SaveSettingsToRegistry("AnimationSpeed", txtAnimationSpeed.Text);

SaveSettingsToRegistry("ShapeCount", txtShapeCount.Text);

MessageBox.Show("Settings saved!", "Info", MessageBoxButtons.OK, MessageBoxIcon.Information);

this.Close();

};

this.Controls.Add(btnSave);

}

private void SettingsForm\_FormClosed(object sender, FormClosedEventArgs e)

{

Debug.WriteLine("Saving screen saver's settings...");

}

// Сохранение в реестр

public static void SaveSettingsToRegistry(string key, string value)

{

using (var registryKey = Registry.CurrentUser.CreateSubKey(RegistryKey))

{

if (registryKey != null)

{

registryKey.SetValue(key, value);

}

else

{

throw new Exception("Unable to access registry key.");

}

}

}

// Загрузка из реестра

public static string LoadSettingsFromRegistry(string key)

{

using (var registryKey = Registry.CurrentUser.OpenSubKey(RegistryKey))

{

if (registryKey != null && registryKey.GetValue(key) != null)

{

return registryKey.GetValue(key).ToString();

}

else

{

return null;

}

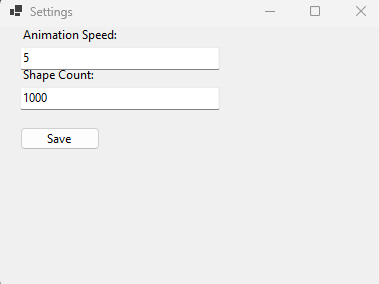
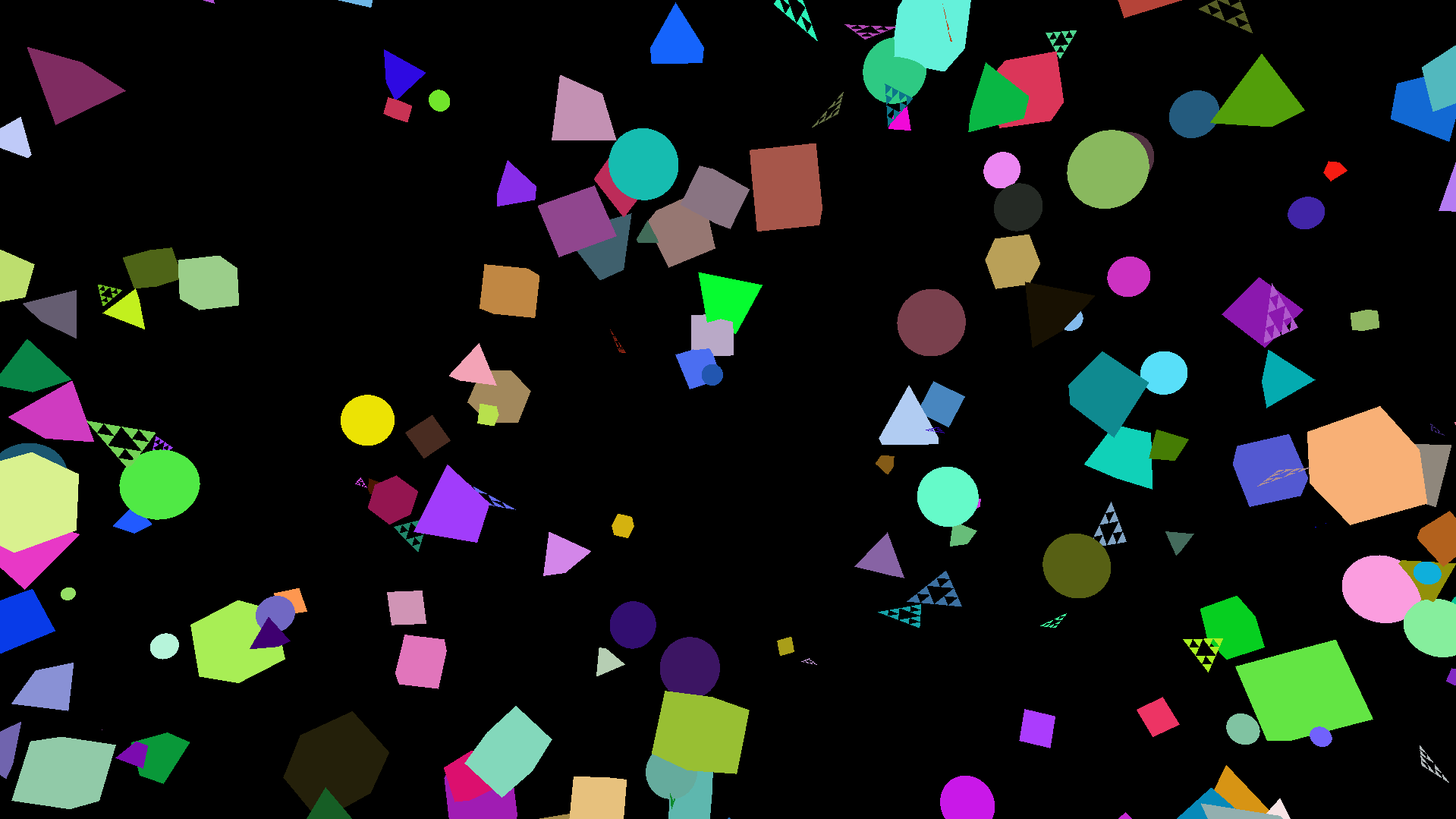
}

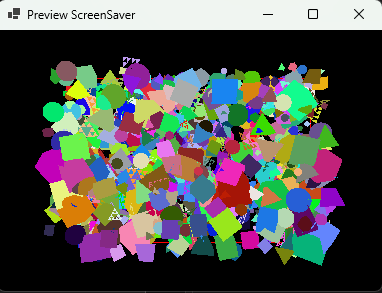
}

}

}

**Скріншоти**





### **Вступ**

Даний проект реалізує скрінсейвер для операційної системи Windows з використанням технологій WinForms та OpenGL. Скрінсейвер підтримує різні режими роботи, включаючи запуск заставки, перегляд попереднього перегляду, а також налаштування параметрів через графічний інтерфейс користувача. Налаштування зберігаються в реєстрі Windows, що дозволяє зберігати параметри між сеансами використання.

### **Структура та Основні Компоненти**

Проект складається з двох основних частин:

* **Клас Program**:
* Відповідає за обробку командних рядків, визначення режиму роботи скрінсейвера та запуск відповідних форм.
* Реалізує основну логіку запуску, прев’ю та налаштувань.
* **Клас SettingsForm**:
* Забезпечує інтерфейс для налаштування параметрів скрінсейвера.
* Відповідає за збереження та завантаження налаштувань з реєстру Windows.

### **Детальний Опис Класу Program**

#### **Основні Компоненти**

* **Перелічення Mode**:
* Визначає можливі режими роботи скрінсейвера:
* Unknown — невідомий режим.
* Settings — відкриття вікна налаштувань.
* ModalSettings — відкриття налаштувань модально до іншого вікна.
* Preview — перегляд попереднього зображення скрінсейвера.
* Run — запуск скрінсейвера.
* **Змінні Стану**:
* mode — поточний режим роботи.
* parentWnd — дескриптор батьківського вікна для режиму попереднього перегляду.
* ownerWnd — дескриптор власника вікна для модального режиму налаштувань.
* DesignMode — прапорець, який визначає, чи знаходиться додаток у режимі дизайну.

#### **Метод Main**

Точка входу додатку, яка виконує наступні кроки:

* **Ініціалізація**:
* Встановлює DesignMode у false.
* Налаштовує високий DPI режим та візуальні стилі для WinForms.
* **Парсинг Командного Рядка**:
* Викликає метод Selector для визначення режиму роботи на основі аргументів командного рядка.
* **Виконання Дій Залежно від Режиму**:
* Mode.Run: Викликає ShowSaver для запуску заставки.
* Mode.Preview: Викликає ShowPreview для показу попереднього перегляду.
* Mode.Settings: Запускає форму налаштувань SettingsForm.
* Mode.ModalSettings: Викликає ShowModalSetting для модального відкриття налаштувань.
* Unknown: Відображає повідомлення про невідому команду або відсутність аргументів.

#### **Метод Selector**

Відповідає за розбір аргументів командного рядка та визначення відповідного режиму роботи.

* Якщо аргументів немає, повертає Mode.Settings.
* Аналізує перший аргумент і повертає відповідний режим:
* /s: Mode.Run
* /p: Викликає ModePreview
* /c: Викликає ModeSetting
* Інші аргументи: Mode.Unknown

#### **Методи Обробки Режимів**

* **ModePreview**:
* Обробляє аргумент /p <HWND>.
* Якщо дескриптор вікна не передано, повертає Mode.Preview з parentWnd = IntPtr.Zero.
* Якщо дескриптор передано, зберігає його та повертає Mode.Preview.
* **ModeSetting**:
* Обробляє аргумент /c[:<HWND>].
* Якщо дескриптор не передано, повертає Mode.Settings.
* Якщо дескриптор передано, зберігає його та повертає Mode.ModalSettings.

#### **Метод ShowSaver**

Відповідає за запуск заставки у повноекранному режимі на всіх доступних екранах.

* Ховає курсор миші.
* Для кожного екрану створює та показує екземпляр MainForm.
* Запускає цикл обробки повідомлень WinForms через Application.Run().

#### **Метод ShowModalSetting**

Відкриває форму налаштувань модально до вказаного вікна власника (ownerWnd).

* Створює обгортку для дескриптора вікна.
* Відкриває SettingsForm як модальне вікно.

#### **Метод ShowPreview**

Відповідає за показ попереднього перегляду скрінсейвера.

* Якщо parentWnd дорівнює IntPtr.Zero, створює маленьку форму з попереднім переглядом.
* Якщо parentWnd не дорівнює IntPtr.Zero, вбудовує форму попереднього перегляду в батьківське вікно.

### **Детальний Опис Класу SettingsForm**

#### **Основні Компоненти**

* **Константа RegistryKey**:
* Визначає шлях до ключа реєстру, де зберігаються налаштування скрінсейвера: "Software\\Lab\_7.exe".
* **Конструктор SettingsForm**:
* Ініціалізує компоненти форми (визначено в InitializeComponent).
* **Метод SettingsForm\_Load**:
* Викликається при завантаженні форми.
* Завантажує налаштування з реєстру або встановлює значення за замовчуванням.
* Динамічно створює елементи керування (TextBox, Label, Button) для редагування налаштувань.
* Реєструє обробник події Click для кнопки "Save".
* **Метод SettingsForm\_FormClosed**:
* Виводить повідомлення в відладчик при закритті форми.
* **Методи SaveSettingsToRegistry та LoadSettingsFromRegistry**:
* Відповідають за збереження та завантаження налаштувань з реєстру Windows.

#### **Метод SettingsForm\_Load**

* **Завантаження Налаштувань**:
* Викликає LoadSettingsFromRegistry для отримання значень параметрів AnimationSpeed та ShapeCount.
* Якщо значення відсутні, встановлює значення за замовчуванням: "1.0" для швидкості анімації та "300" для кількості фігур.
* **Створення Елементів Керування**:
* Динамічно створює TextBox для введення швидкості анімації та кількості фігур.
* Додає відповідні Label для опису полів вводу.
* Додає кнопку "Save", яка зберігає налаштування при натисканні.
* **Обробник Події Click для Кнопки "Save"**:
* Зберігає введені значення в реєстр за допомогою SaveSettingsToRegistry.
* Відображає повідомлення про успішне збереження налаштувань.
* Закриває форму налаштувань.

#### **Методи Збереження та Завантаження Налаштувань**

* **SaveSettingsToRegistry**:
* Відкриває або створює підключ до вказаного ключа реєстру.
* Записує значення параметра за заданим ключем.
* Викидає виняток, якщо не вдалося отримати доступ до ключа реєстру.
* **LoadSettingsFromRegistry**:
* Відкриває підключ до вказаного ключа реєстру.
* Повертає значення параметра за заданим ключем або null, якщо параметр відсутній.

### **Взаємодія між Класами Program та SettingsForm**

* **Запуск Налаштувань**:
* При запуску скрінсейвера з аргументом /c або /c:<HWND>, клас Program відкриває форму SettingsForm для налаштування параметрів.
* **Збереження та Завантаження Налаштувань**:
* SettingsForm відповідає за завантаження існуючих налаштувань при завантаженні форми та за збереження змін при натисканні кнопки "Save".
* Налаштування зберігаються в реєстрі Windows, що дозволяє зберігати параметри між сеансами використання.
* **Використання Налаштувань**:
* Інші частини скрінсейвера (наприклад, RenderControl) можуть звертатися до параметрів, збережених у реєстрі, для налаштування поведінки та вигляду заставки.

### **Опис Реалізації**

* **Режими Роботи**:
* Скринсейвер підтримує різні режими роботи, які визначаються через аргументи командного рядка.
* Основні режими включають запуск заставки (/s), перегляд попереднього зображення (/p <HWND>), та відкриття налаштувань (/c або /c:<HWND>).
* **Обробка Командного Рядка**:
* Клас Program аналізує аргументи командного рядка для визначення режиму роботи.
* В залежності від режиму, виконується відповідна логіка, наприклад, показ заставки або відкриття налаштувань.
* **Збереження Налаштувань**:
* Налаштування скрінсейвера зберігаються в реєстрі Windows за допомогою класу Registry.
* Це забезпечує збереження параметрів між перезавантаженнями системи та використаннями скрінсейвера.
* **Інтерфейс Користувача**:
* Використовується технологія WinForms для створення графічного інтерфейсу користувача.
* Динамічне створення елементів керування у SettingsForm дозволяє гнучко змінювати інтерфейс залежно від потреб.
* **Інтеграція з OpenGL**:
* Хоча основний код OpenGL не наданий, згадуються методи малювання кривих та інших графічних елементів.
* RenderControl (з попереднього звіту) відповідає за візуалізацію графіки за допомогою OpenGL.

### **Висновок**

Даний проект демонструє ефективну реалізацію скрінсейвера з підтримкою різних режимів роботи та налаштувань користувача. Використання WinForms для інтерфейсу та інтеграція з реєстром Windows дозволяють забезпечити зручність налаштування та стабільність роботи додатку. Структура коду забезпечує гнучкість та можливість подальшого розширення функціональності, що робить його придатним для навчальних, дослідницьких або професійних проектів.