МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники (ФИТР)

Кафедра программного обеспечения информационных

систем и технологий

**КУРСОВАЯ РАБОТА**

по дисциплине: «Разработка приложений в визуальных средах»

на тему: ***«Разработка приложения “Фракталы”»***

|  |  |
| --- | --- |
| Выполнила: | ст. гр. 10701121 Панасевич А.А. |
| Принял: | доц. Гурский Н.Н. |

Минск 2023

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники (ФИТР)

Кафедра программного обеспечения информационных

систем и технологий

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовой работе**

**по дисциплине:** «Разработка приложений в визуальных средах»

Тема: «Разработка приложения “Фракталы”»

|  |  |
| --- | --- |
| Исполнитель: Панасевич А.А.  (подпись) | |
| Студентка 2 курса 10701121 группы |  |
| Руководитель: доц. Гурский Н.Н. | |

(подпись)

Минск 2023

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 4](#_1fob9te)

[1 МАТЕМАТИЧЕСКАЯ ФОРМУЛИРОВКА ЗАДАЧИ 5](#_3znysh7)

[2 ОПИСАНИЕ ПРОГРАММЫ 6](#_2et92p0)

[2.1 Структурная схема программы 6](#_tyjcwt)

[2.2 Описание разработанных классов 7](#_zexgku1xerl5)

[2.3 Описание динамических библиотек и используемых серверов 9](#_uquaagoadfsf)

[3 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 11](#_4d34og8)

[4 МЕТОДИКА ИСПЫТАНИЙ 21](#_17dp8vu)

[ЗАКЛЮЧЕНИЕ 24](#_3rdcrjn)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 25](#_26in1rg)

[ПРИЛОЖЕНИЕ 26](#_lnxbz9)

ВВЕДЕНИЕ

Фракталы – это геометрические фигуры, которые имеют самоподобную структуру на разных масштабах. То есть, они выглядят схожими сами с собой, при увеличении или уменьшении размера. Фракталы могут быть созданы с помощью итеративной процедуры, при которой более крупные копии фигуры состоят из более мелких копий, подобных ей.

Фракталы занимают важное место в математике, физике, информатике и других науках. Они имеют множество приложений, включая:

- Моделирование природных явлений, таких как облака, горы и реки.

- Создание компьютерных графиков и изображений высокой сложности.

- Разработка алгоритмов сжатия изображений и видео.

- Исследование хаотических систем и динамических процессов.

- Изучение фрактальных свойств многомерных объектов.

## 

Целью курсовой работы является закрепление и углубление знаний, полученных при изучении дисциплины «Разработка приложений в визуальных средах» посредством разработки программного обеспечения для вычисления и графического изображения фракталов Коха, Серпинского, Дракона, дерева Пифагора. Для выполнения курсовой работы был выбран язык программирования C#, который обладает рядом преимуществ и подходит для разработки приложений с графическим интерфейсом. C# предоставляет удобные инструменты для создания пользовательского интерфейса и реализации концепций объектно-ориентированного программирования. Благодаря своей интеграции с платформой .NET, C# обеспечивает широкий функционал и доступ к библиотекам, что упрощает разработку приложений и повышает их производительность.

# **1 МАТЕМАТИЧЕСКАЯ ФОРМУЛИРОВКА ЗАДАЧИ**

В программировании фракталы используются для создания визуальных эффектов, генерации сложных узоров и текстур, моделирования природных явлений, а также в компьютерной графике, компьютерной анимации и визуализации данных. Существуют различные алгоритмы и методы для генерации и отображения фракталов, и программисты могут использовать их для создания уникальных и привлекательных визуальных эффектов в своих приложениях.

В данной работе используются фракталы Коха, Серпинского, Дракона, дерево Пифагора.

Рекурсивное правило фрактала Коха: Заменить каждый отрезок на четыре отрезка, образующих равносторонний треугольник.

Рекурсивное правило фрактала Серпинского: Заменить каждый треугольник на три треугольника меньшего размера, образующих "вырезы" внутри исходного треугольника.

Рекурсивное правило фрактала Дракона: Заменить каждую линию на две линии под определенным углом, образующие "зигзагообразную" структуру.

Рекурсивное правило фрактала дерева Пифагора: Заменить каждую линию на две линии, одна из которых является продолжением предыдущей под углом θ, а другая отходит от середины предыдущей линии под углом -90° относительно нее.

# **2 ОПИСАНИЕ ПРОГРАММЫ**

## 2.1 Структурная схема программы

В процессе разработки программного продукта были реализованы следующие формы:

1. *Form\_intro.cs* – форма c оригинальной заставкой;
2. *Form1.cs* – основная форма, реализующая функционал программного обеспечения;
3. *About*.*cs* – форма, содержащая информацию о программном обеспечении.
4. *Export.cs* – форма, реализующая экспорт информации с основной формы;
5. *FormGraph.cs* – форма, содержащая дополнительную информацию о фрактале.

Также программа содержит компоненты, предназначенные для взаимодействия с Paint, Word и Excel, вызова информации о программе, выполнения вычислений и отображения процесса вычислений.

Ниже на рисунке 2.1 представлена структурная схема программы.

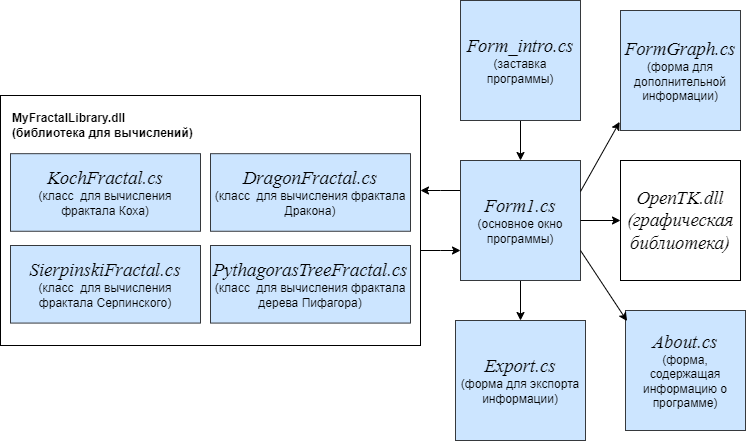


Рисунок 2.1 – Структурная схема программы

## 2.2 Описание разработанных классов

Для реализации основного интерфейса программы и изображения фрактала был разработан класс Form1.cs, в котором содержаться следующие методы и свойства:

Свойства:

public string NameFile – хранит название файла с изображением фрактала;

private Color drawColor – хранит цвет пера для отрисовки фрактала;

private Color backgroundColor – хранит цвет фона элемента для изображения фрактала;

Методы:

private void Form1\_Load(object sender, EventArgs e) – метод, вызывающий класс Form\_intro для показа заставки приложения, также задает параметры видимости элементам для ввода информации пользователем;

private void paintToolStripMenuItem\_Click(object sender, EventArgs e) – вызывает метод из класса Export для взаимодействия с Paint;

private void выходToolStripMenuItem\_Click(object sender, EventArgs e) – завершает работу приложения, дополнительно спрашивая подтверждение выхода;

private void руководствоПользователяToolStripMenuItem\_Click(object sender, EventArgs e) – открывает руководство пользователя;

private void оПриложенииToolStripMenuItem\_Click(object sender, EventArgs e) – вызывает форму About;

private void ToExcelNew\_Click(object sender, EventArgs e) – вызывает метод из класса Export для создания нового файла формата Excel;

private void ToExcelAdd\_Click(object sender, EventArgs e) – вызывает метод из класса Export для добавления информации в уже существующий файл формата Excel;

private void radioButtonKoch\_CheckedChanged(object sender, EventArgs e) – проверяет состояние radiobutton и устанавливает видимость элементов согласно этому состоянию;

private void radioButtonSerpinsky\_CheckedChanged(object sender, EventArgs e) – проверяет состояние radiobutton и устанавливает видимость элементов согласно этому состоянию;

private void radioButtonDragon\_CheckedChanged(object sender, EventArgs e) – проверяет состояние radiobutton и устанавливает видимость элементов согласно этому состоянию;

private void radioButtonPythagor\_CheckedChanged(object sender, EventArgs e) – проверяет состояние radiobutton и устанавливает видимость элементов согласно этому состоянию;

private List<Tuple<double, double>> GenerateFractalPoints() – записывает список координат точек фрактала, обращаясь к dll-библиотеке MyFractalLibrary, проверяет правильность вводимых данных, показывает подсказки по диапазону допустимых значений для пользователя;

private void buttonCreate\_Click(object sender, EventArgs e) – отвечает за отображение графика фрактала на основной форме и так же за вызов формы FromGraph для просмотра анимации и таблицы координат точек фрактала.

Для экспорта данных с основной формы был создан класс Export.cs, содержащий:

Свойства:

public static string NameFile – хранит название файла с изображением фрактала;

Методы:

public static void ToPaint(Control control, DateTime now) – создание и сохранение скриншота экрана с графиком;

public static void excelApp\_SheetFollowHyperlink(object Sh, Excel.Hyperlink Target) – создание гиперссылки в виде ссылки на изображение по адресу расположения изображения;

public static void ToExcelNew(DateTime now, string depth, string angle, string size, string ratio) – создание, запись и открытие нового Excel документа;

public static void ToExcelAdd(DateTime now, string depth, string angle, string size, string ratio) – добавление информации в уже существующий файл Excel и его открытие.

Для изображения анимации и просмотра и записи координат точек фрактала, был создан класс FormGraph.cs, содержащий:

Свойства:

private List<Tuple<double, double>> points - Список точек, используемых для отображения графика.

private double currentScale - Текущий масштаб.

Методы:

public FormGraph(List<Tuple<double, double>> points, Color drawColor, Color backgroundColor) - Конструктор класса FormGraph, принимающий параметры points, drawColor и backgroundColor.

private void buttonExport\_Click(object sender, EventArgs e) - Обработчик события нажатия на кнопку экспорта в Excel.

private void buttonWord\_Click(object sender, EventArgs e) - Обработчик события нажатия на кнопку экспорта в Word.

private void buttonAnimation\_Click(object sender, EventArgs e) - Обработчик события нажатия на кнопку создания и запуска анимации.

private void textBoxScale\_TextChanged(object sender, EventArgs e) - Обработчик изменения содержимого текстового поля масштаба.

## 2.3 Описание динамических библиотек

В данной программе была разработана динамическая библиотека MyFractalLibrary.dll, которая содержит четыре класса для вычисления координат точек фракталов. При подключении, библиотек возвращает список формата List<Tuple<double, double>>.

В классе DragonFractal для вычисления точек фрактала Драконы были разработаны:  
 Свойства:

public int Depth { get; set; } - Глубина фрактала.

public double Angle { get; set; } - Угол наклона.

public double X1 { get; set; } - Координата X начальной точки.

public double Y1 { get; set; } - Координата Y начальной точки.

Методы:

public DragonFractal(int depth, double angle, double x1, double y1) - Конструктор класса DragonFractal, принимающий параметры depth, angle, x1 и y1.

public List<Tuple<double, double>> GeneratePoints() - Метод для генерации точек фрактала.

private void GenerateDragonCurve(List<Tuple<double, double>> points, double x1, double y1, double x2, double y2, int depth) - Рекурсивный метод для генерации фрактала Дракона.

В классе KochFractal для вычисления точек фрактала Коха были разработаны:

Свойства:

public int Depth { get; set; } - Глубина фрактала.

public double Angle { get; set; } - Угол наклона.

public double X1 { get; set; } - Координата X начальной точки.

public double Y1 { get; set; } - Координата Y начальной точки.

Методы:

public KochFractal(int depth, double angle, double x1, double y1) - Конструктор класса KochFractal, принимающий параметры depth, angle, x1 и y1.

public List<Tuple<double, double>> GeneratePoints() - Метод для генерации точек фрактала.

private void GenerateKochCurve(List<Tuple<double, double>> points, double x1, double y1, double x2, double y2, int depth) - Рекурсивный метод для генерации фрактала Коха.

В классе PythagorasTreeFractal для вычисления точек фрактала дерева Пифагора были разработаны:

Свойства:

public int Depth { get; set; } - Глубина фрактала.

public double Size { get; set; } - Размер исходной линии.

public double X1 { get; set; } - Координата X начала линии.

public double Y1 { get; set; } - Координата Y начала линии.

public double Angle { get; set; } - Угол наклона исходной линии.

public double BranchRatio { get; set; } - Коэффициент масштабирования ветвей.

Методы:

public PythagorasTreeFractal(int depth, double size, double x1, double y1, double angle, double branchRatio) - Конструктор класса PythagorasTreeFractal, принимающий параметры depth, size, x1, y1, angle и branchRatio.

public List<Tuple<double, double>> GeneratePoints() - Метод для генерации точек фрактала.

private void GenerateTree(List<Tuple<double, double>> points, double x1, double y1, double size, double angle, int depth) - Рекурсивный метод для генерации фрактала Пифагорова дерева.

В классе SierpinskiFractal для вычисления точек фрактала Серпинского были разработаны:  
 Свойства:

public int Depth { get; set; } - Глубина фрактала.

public double Size { get; set; } - Размер.

public double X1 { get; set; } - Координата X начальной точки.

public double Y1 { get; set; } - Координата Y начальной точки.

Методы:

public SierpinskiFractal(int depth, double size, double x1, double y1) - Конструктор класса SierpinskiFractal, принимающий параметры depth, size, x1 и y1.

public List<Tuple<double, double>> GeneratePoints() - Метод для генерации точек фрактала.

private void GenerateSierpinski(List<Tuple<double, double>> points, double x1, double y1, double size, int depth) - Рекурсивный метод для генерации фрактала Серпинского.

Помимо этого была подключена динамическая библиотека OpenTK для графического изображение фрактала с помощью элемента OpenGL.  
Программа также поддерживает возможность вывода данных при помощи средств MSOffice, используя принципы СOM технологий, обеспечивая взаимодействие с серверами автоматизации приложений MSExcel и MSWord, позволяя результат вычислений сохранить в Excel и в Word.

# **3 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

Сначала нужно запустить файл Fractal.exe, который предствален на рис. 3.1:

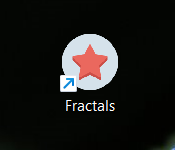


Рисунок 3.1 – Иконка файла формата .exe приложения Фракталы

После вызова приложения появится оригинальная заставка, изображённая на рисунке 3.2.

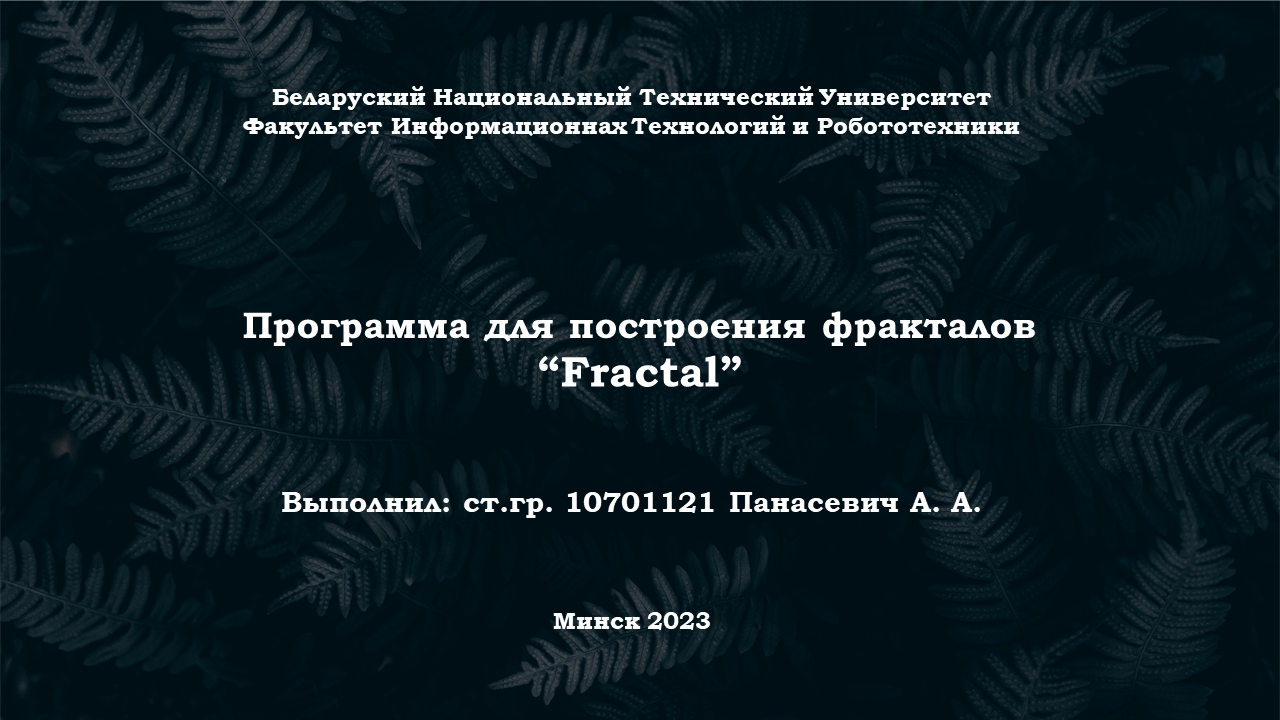


Рисунок 3.2 – Заставка приложения

После заставки появляется главное окно приложения, представленное ниже на рисунке 3.3.

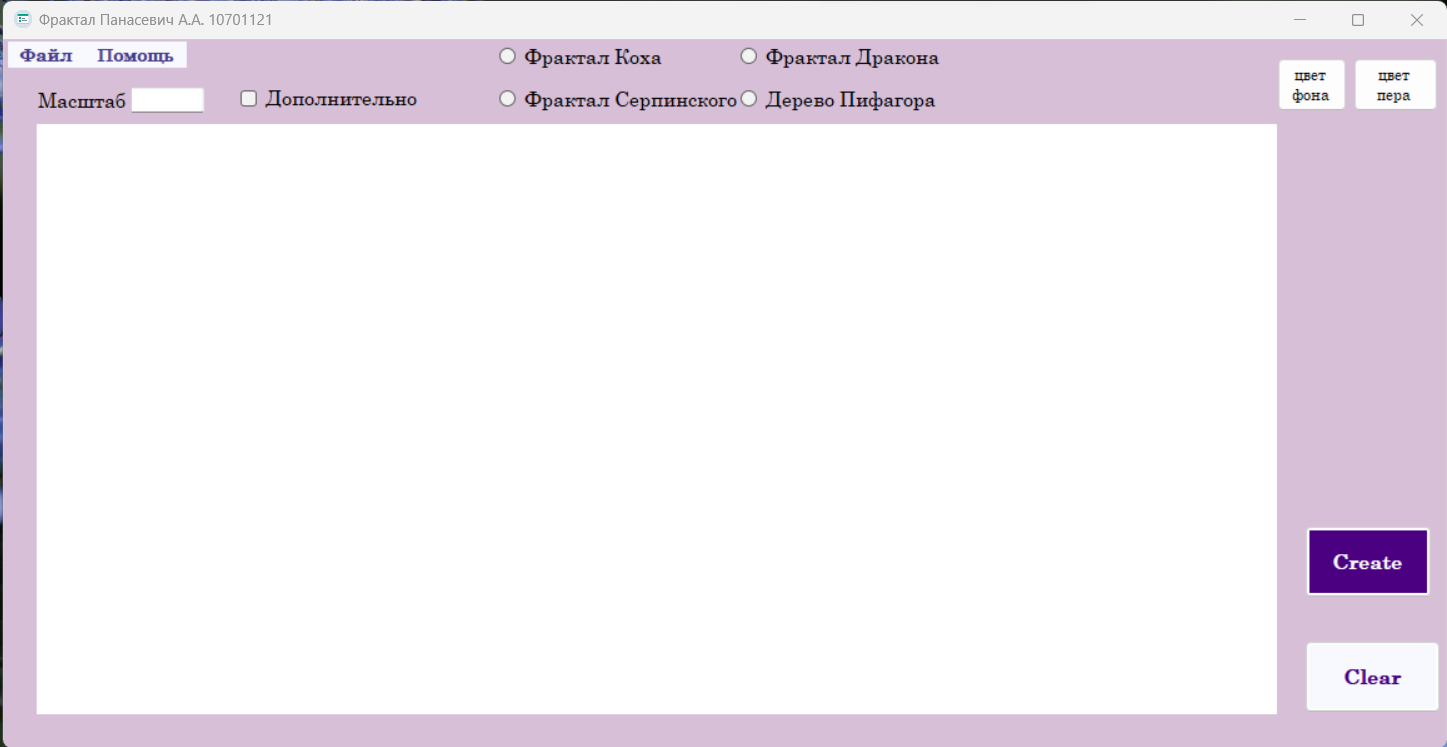


Рисунок 3.3 – Главное окно приложения

На главной форме имеется элемент menuStrip, представленный двумя ToolStripMenuItem: Файл и Помощь (Рис. 3.4 - 3.5).

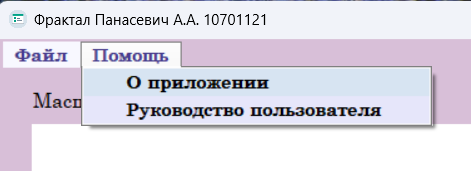


Рисунок 3.4 – Меню приложения, вкладка Помощь

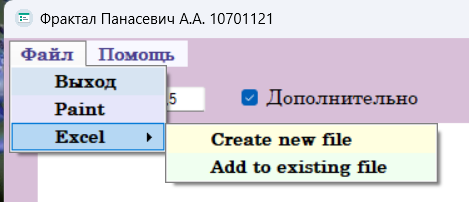


Рисунок 3.5 – Меню приложения, вкладка Файл

По нажатию пункта меню Помощь > О программе открывается окно, содержащее информацию о программе. Форма представлена на рисунке 3.6.1

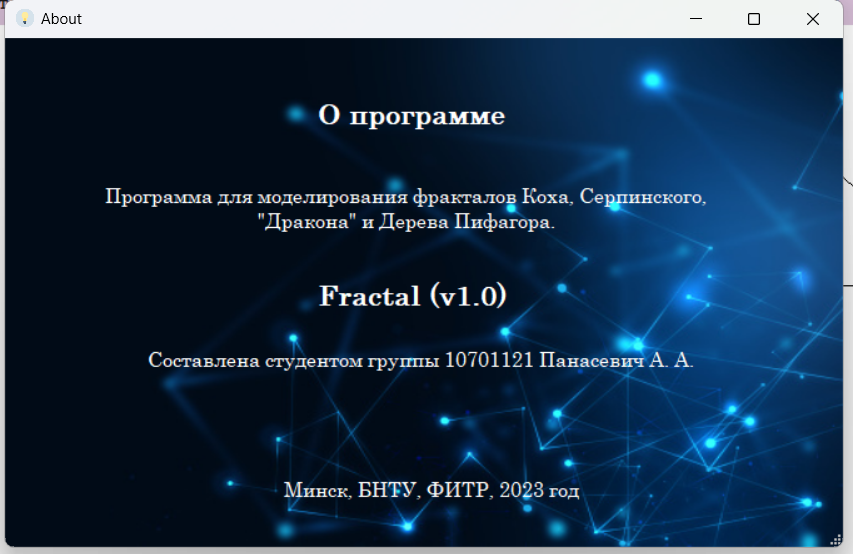


Рисунок 3.6.1 – Форма о программе

По нажатию пункта меню Помощь > Руководство пользователя открывается файл руководства пользователя.

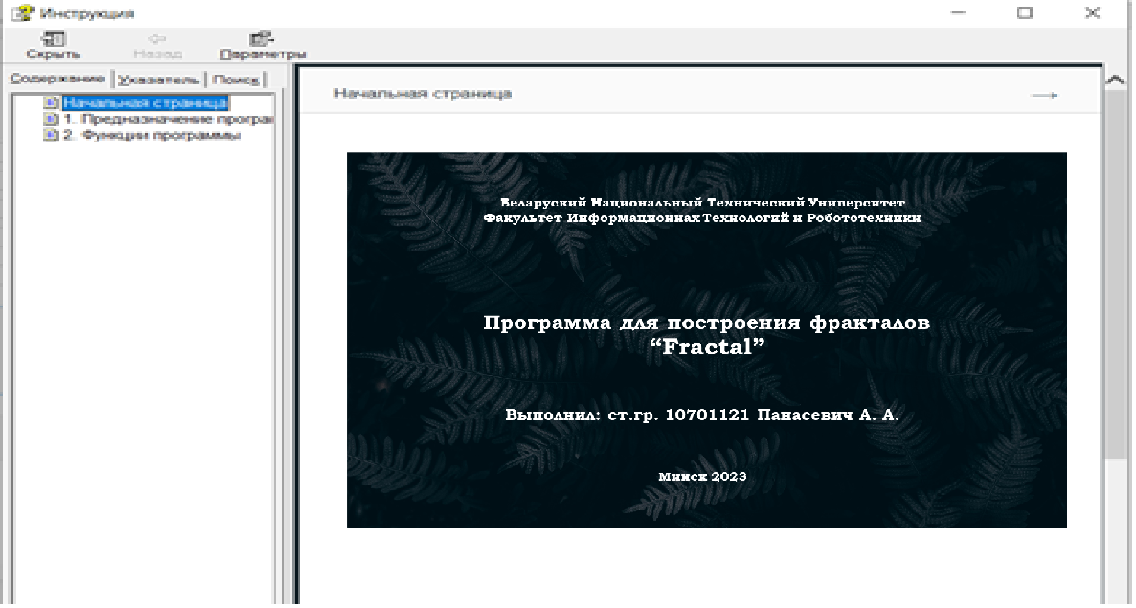


Рисунок 3.6.1 – help-file Руководство пользователя

Также программа позволяет сделать копию изображения фрактала, сохранить ее и открыть в приложении Paint, для удобной редакции изображения (рис. 3.7 - 3.8):

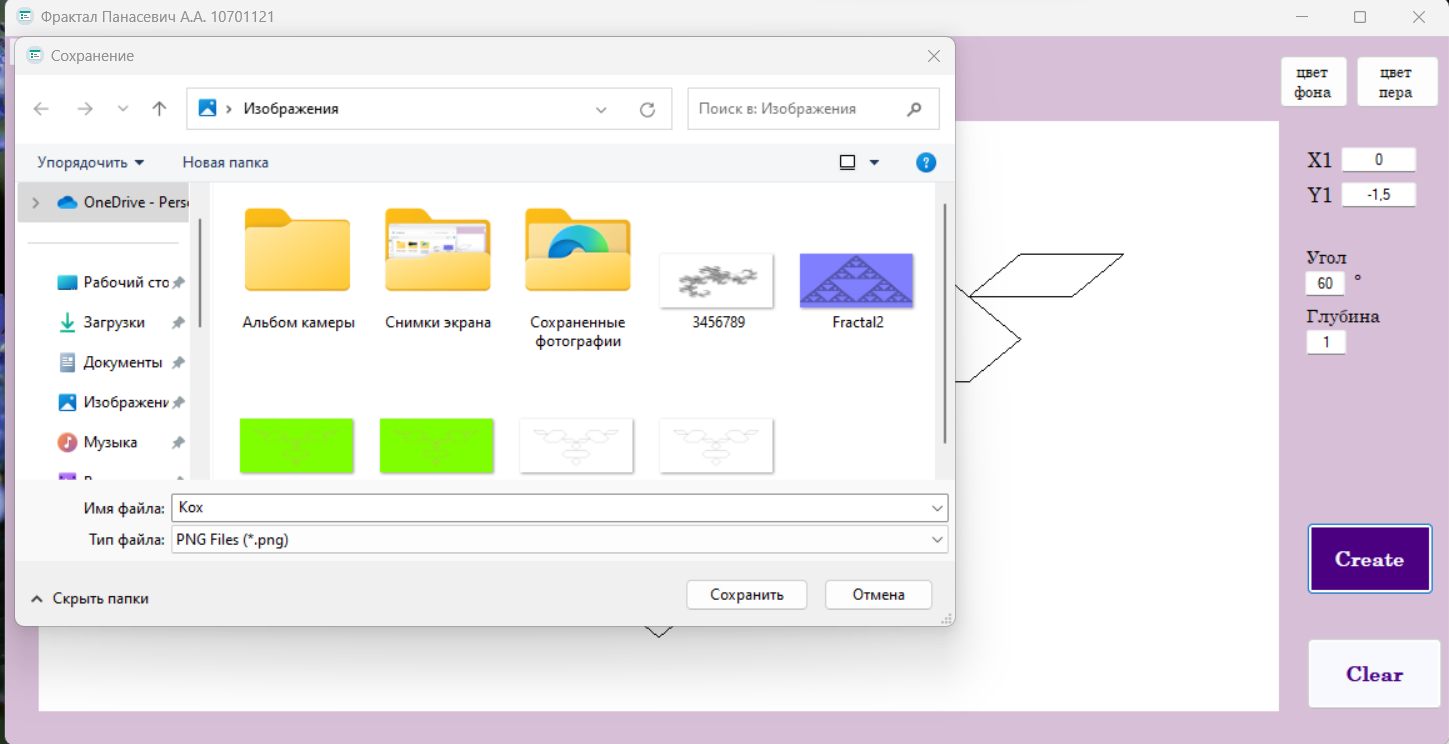


Рисунок 3.7 – Открытие диалогового окна для выбора места сохранения и наименования файла.

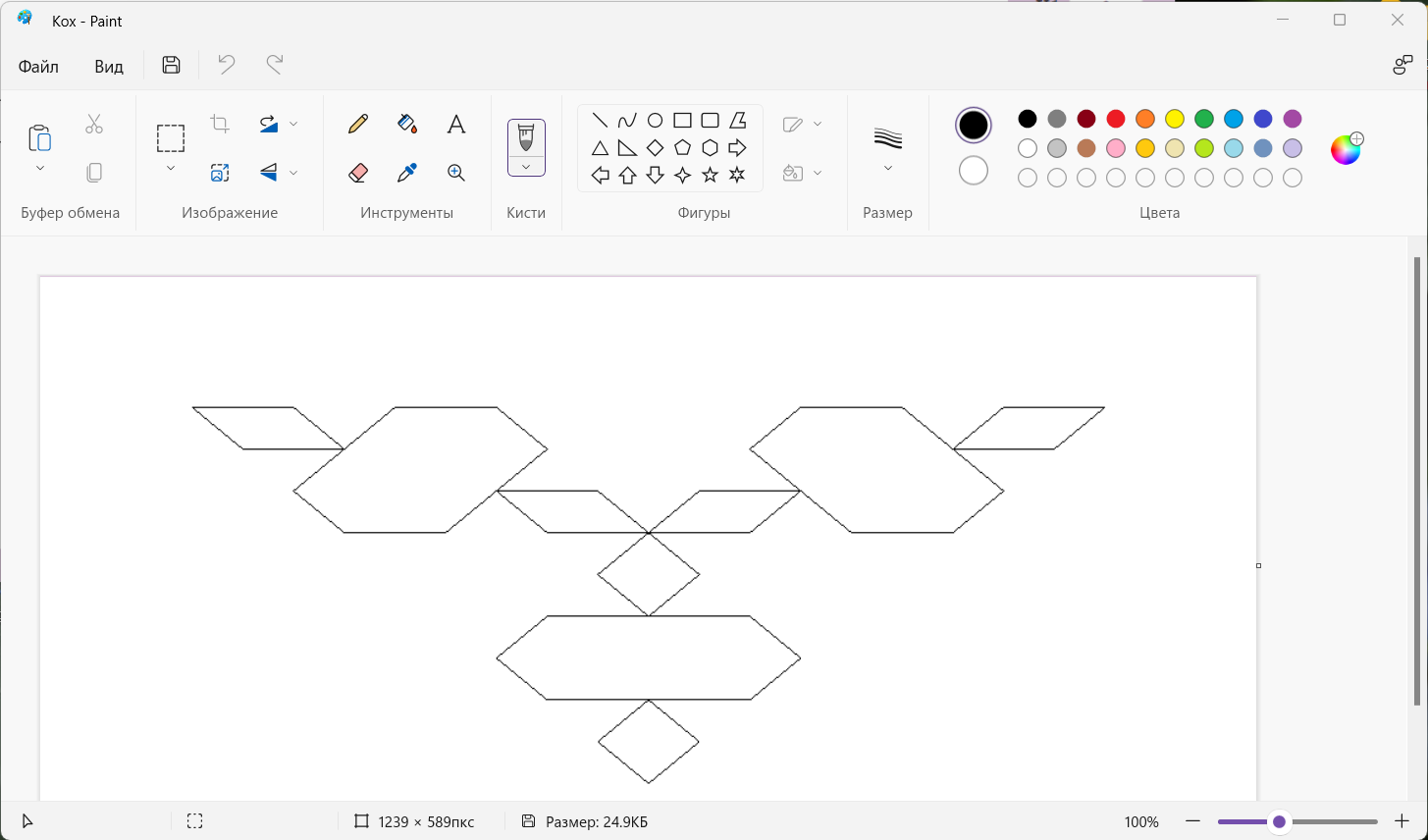


Рисунок 3.8 – Открытие файла в приложении Paint.

Далее есть возможность создать соответствующую запись о параметрах фрактала и изображении фрактала в таблице Excel, выбрав пункт создания нового файла данного приложения, после чего появляется диалоговое окно, сообщение об успешном создании файла и открывается приложение Excel, как показано на рис. 3.9 - 3.11.

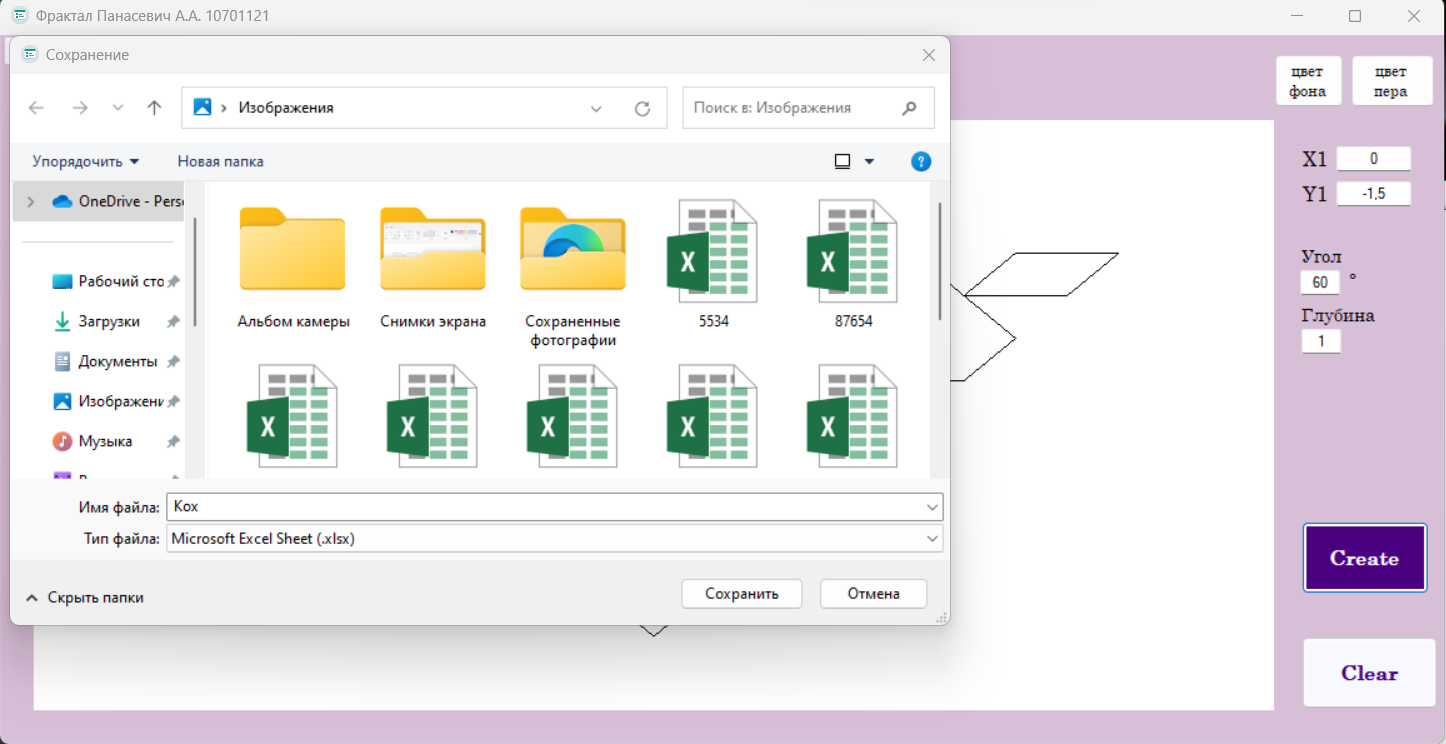


Рисунок 3.9 – Открытие диалогового окна для выбора месторасположения файла и его названия.

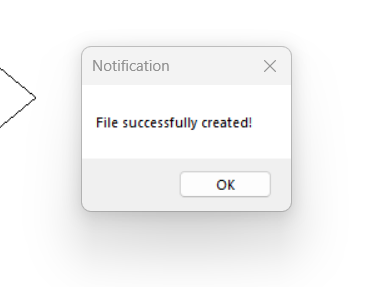


Рисунок 3.10 – Сообщение об успешном создании файла.

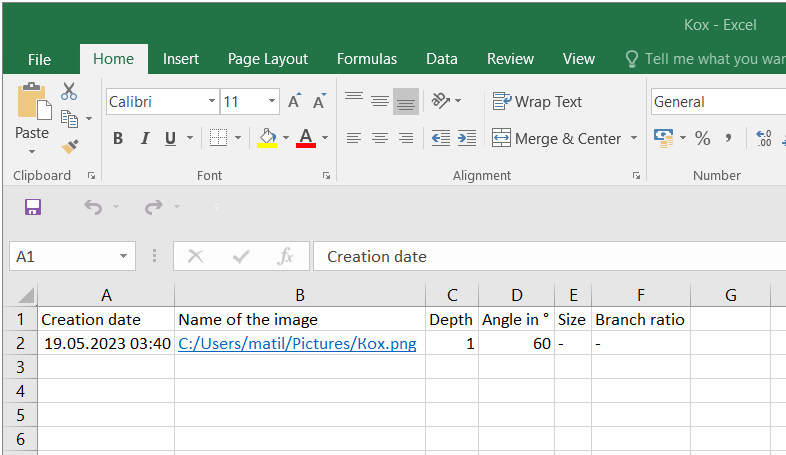


Рисунок 3.11 – Открытие файла в приложении Excel.

Если же пользователь хочет добавить запись в уже существующий файл, он выбирает пункт меню “Добавить в уже существующий”, в открывшемся диалоговом окне выбирает файл, после чего он открывается с уже добавленной записью, как продемонстрировано на рисунке 3.12:

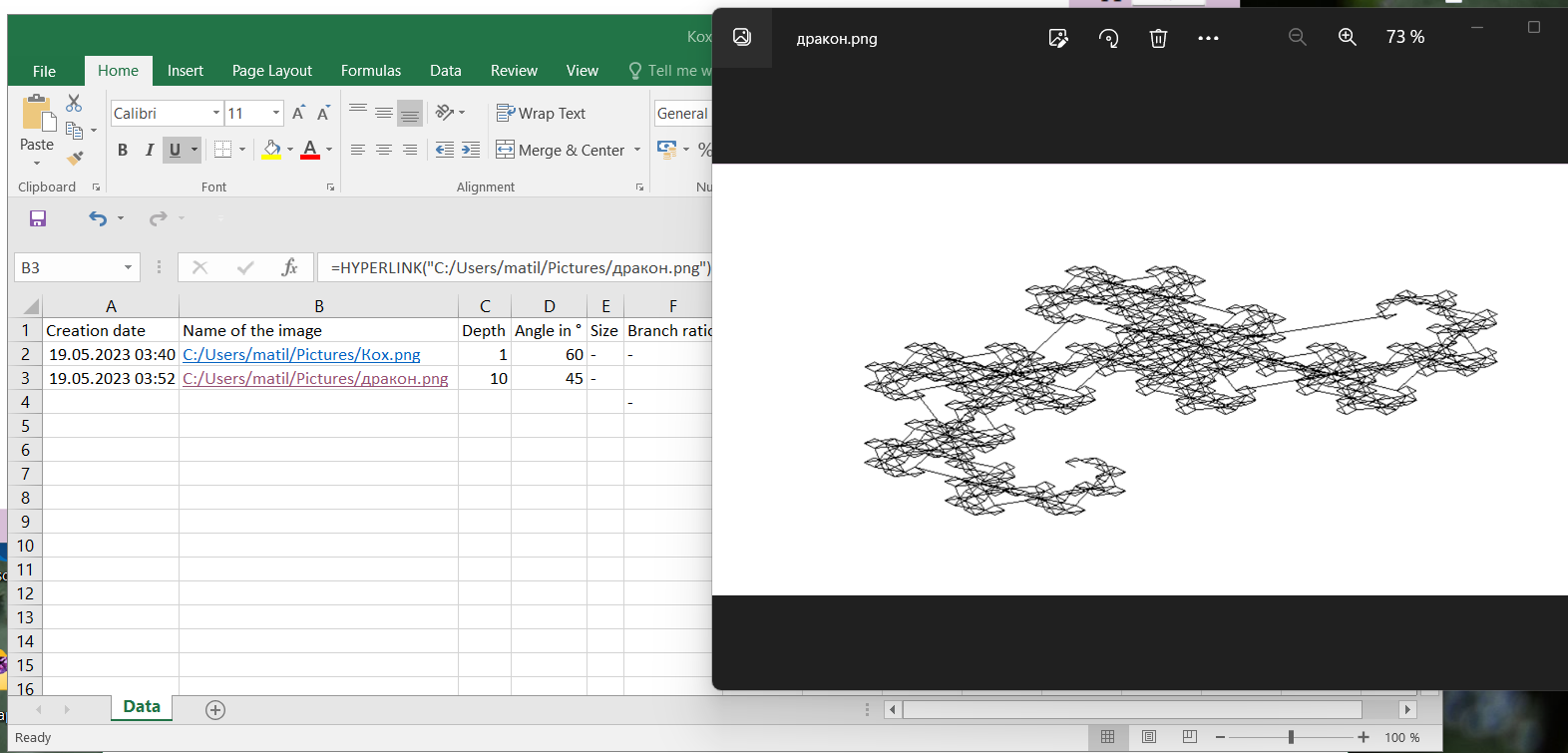


Рисунок 3.12 – Открытие файла в приложении Excel.

Как можно заметить из рисунка выше, если перейти по гиперссылке в графе “Name of the image”, открывается изображение фрактала, которое мы создали перед этим.

Также на главной форме имеются поля textBox для ввода масштаба, координат, глубины, угла, размера, разветвления, checkBox для вызова дополнительной формы, radioButton для выбора метода фрактала, кнопки для выбора цвета фона и пера, кнопки для создания фрактала и очистки экрана, элемент glControl для изображения фрактала. Форма представлена на рисунках 3.13.1 - 3.13.5 ниже.

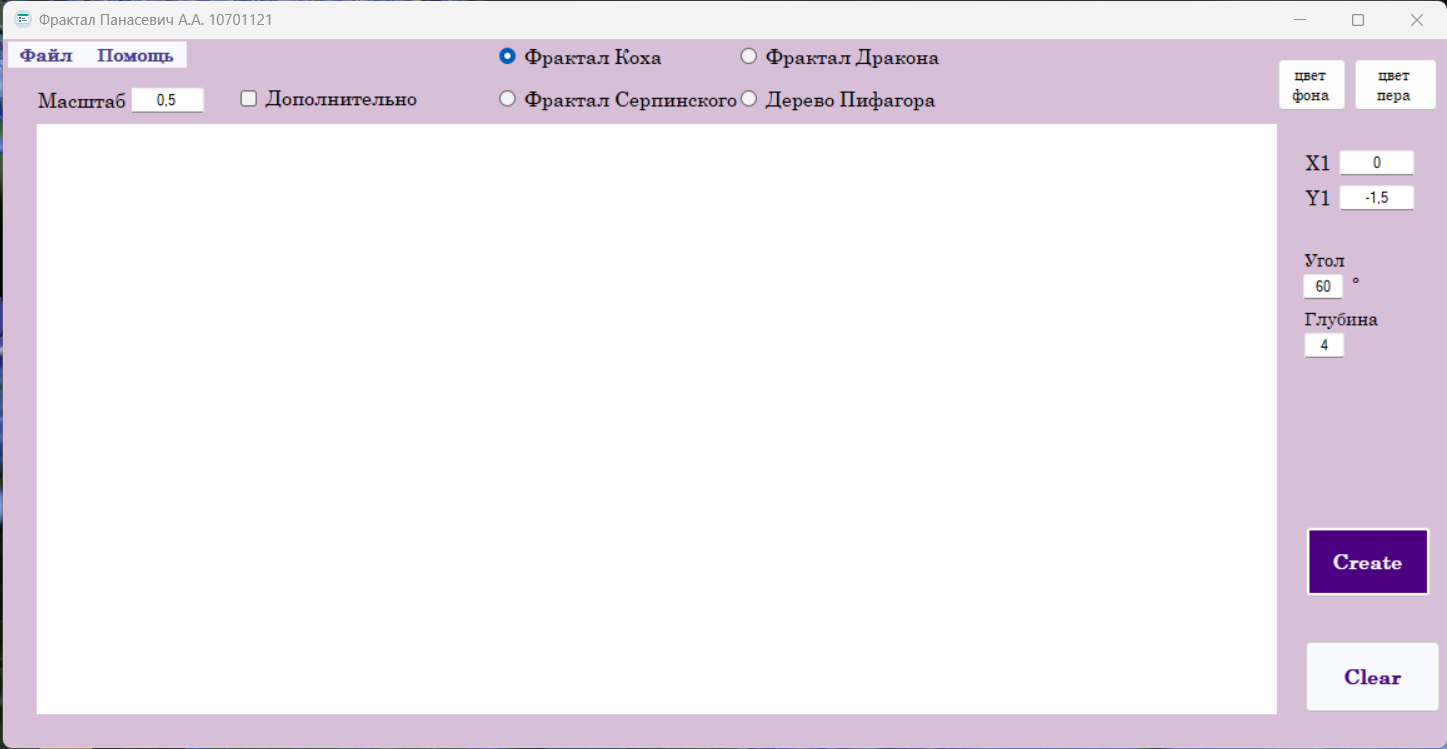


Рисунок 3.13.1 – Главное окно приложения, изменения после выбора метода фрактала.

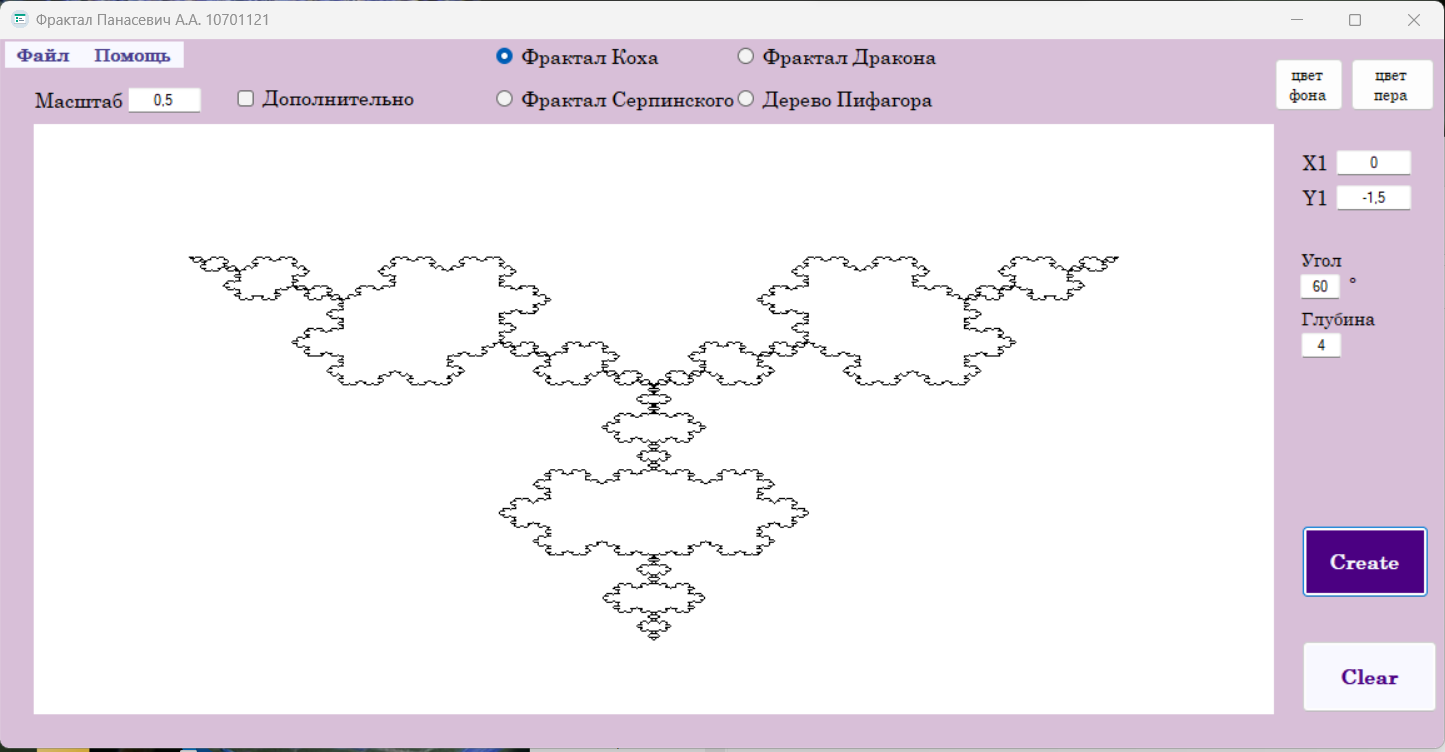


Рисунок 3.13.2 – Главное окно приложения, выбран для изображения фрактал Коха.

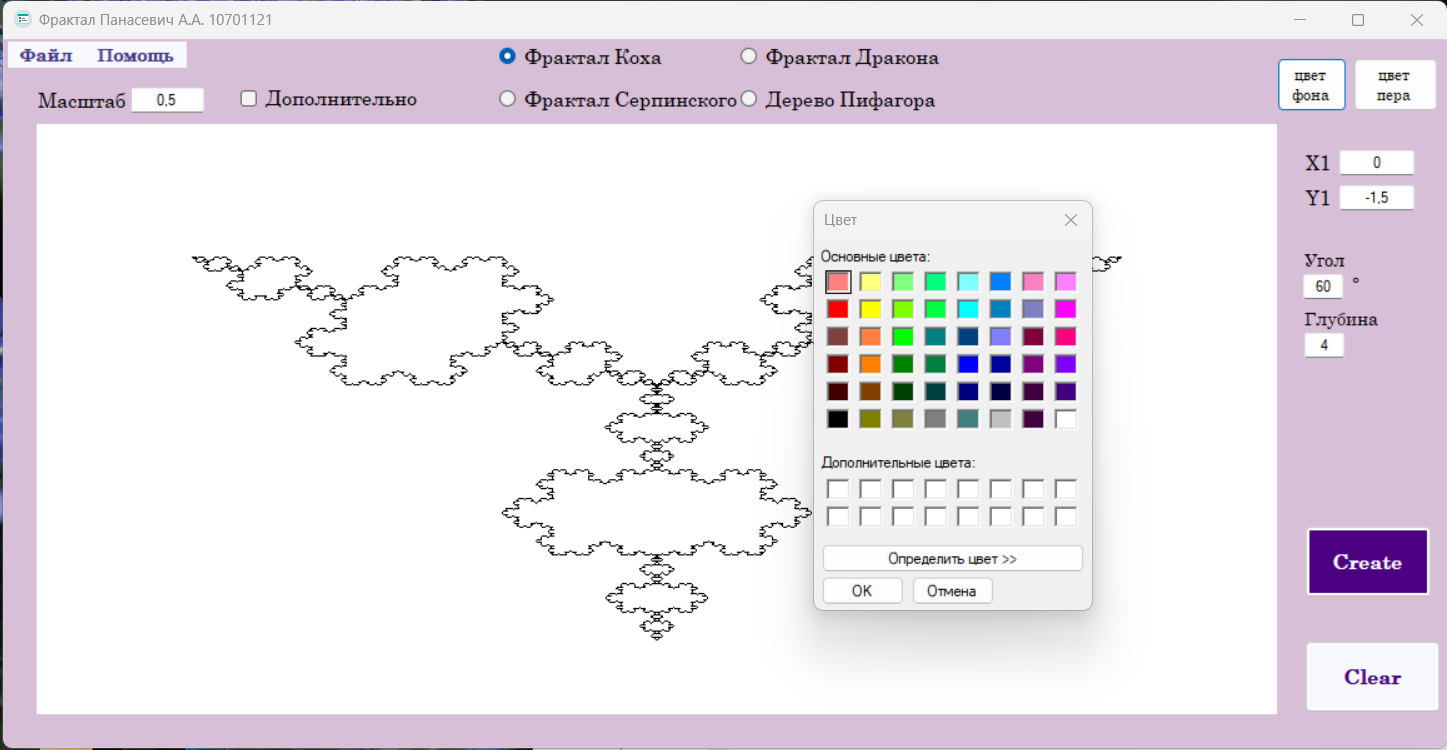


Рисунок 3.13.3 – Окно выбора цвета.

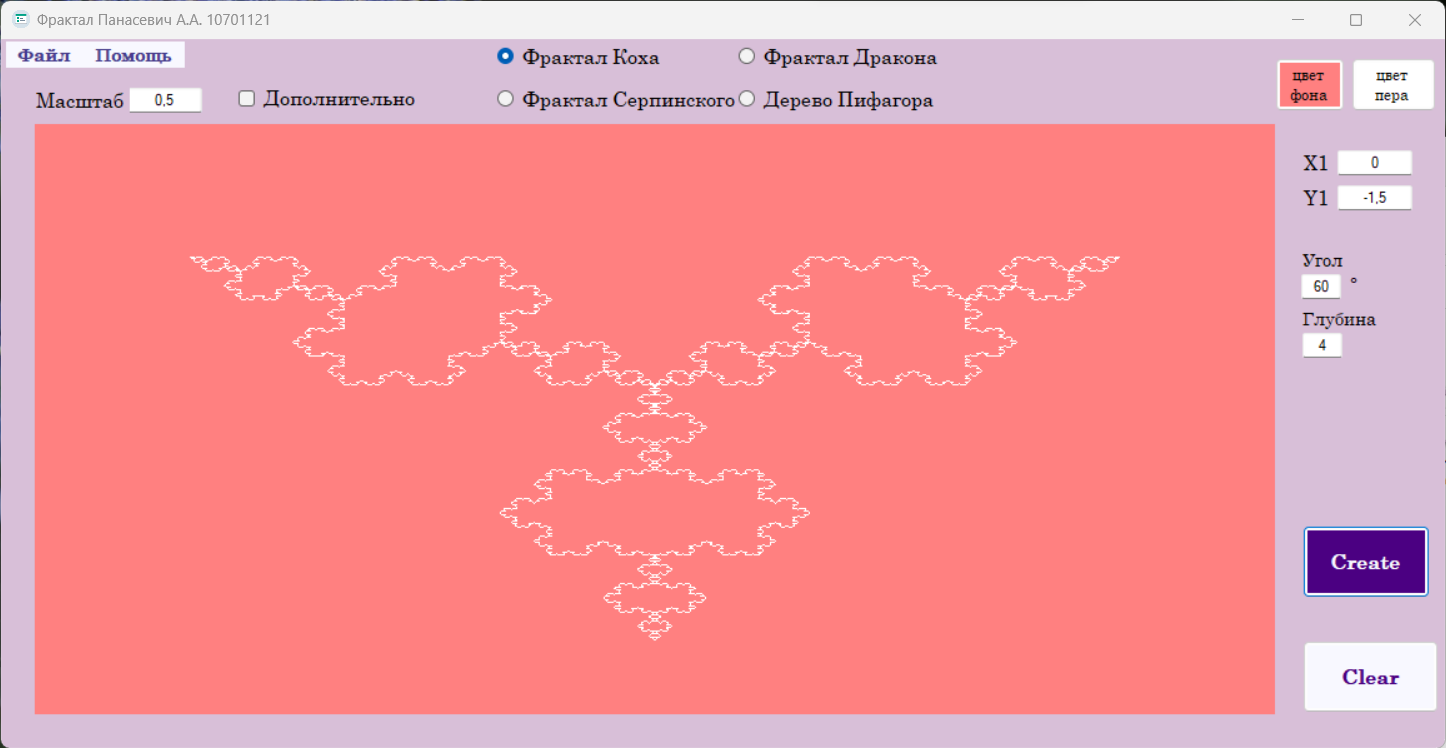


Рисунок 3.13.4 – Результат изображения с измененными цветами, цвета фона и пера соответсвуют цветам кнопок.

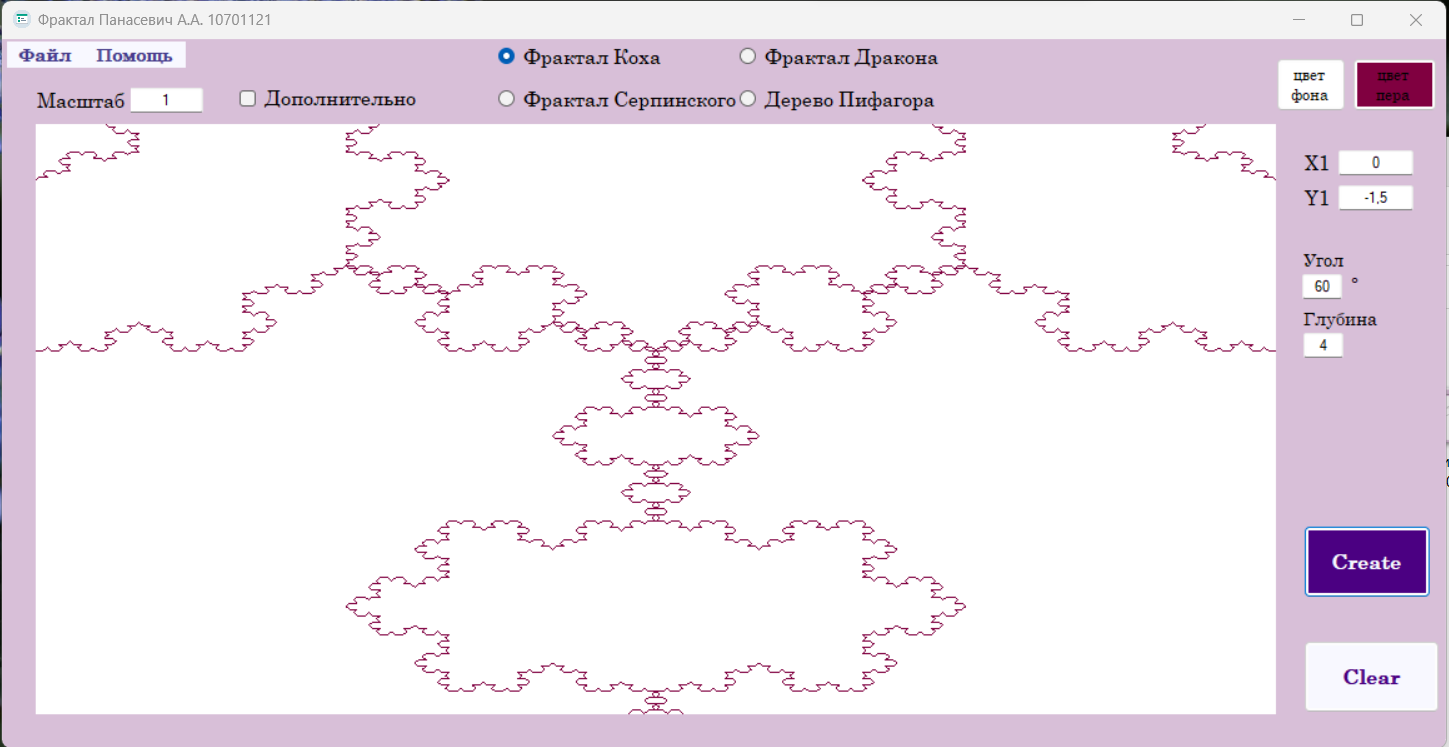


Рисунок 3.13.5 – Главное окно приложения, выбран для изображения фрактал Коха, изменены масштаб и цвет.

Для более подробных данных нужно отметить checkBox “Дополнительная информация” и заново нажать кнопку “Create”. После этого открывается форма с двумя вкладками, на первой есть возможность задавать масштаб и кнопка Анимации, что показано на рисунке 3.14:

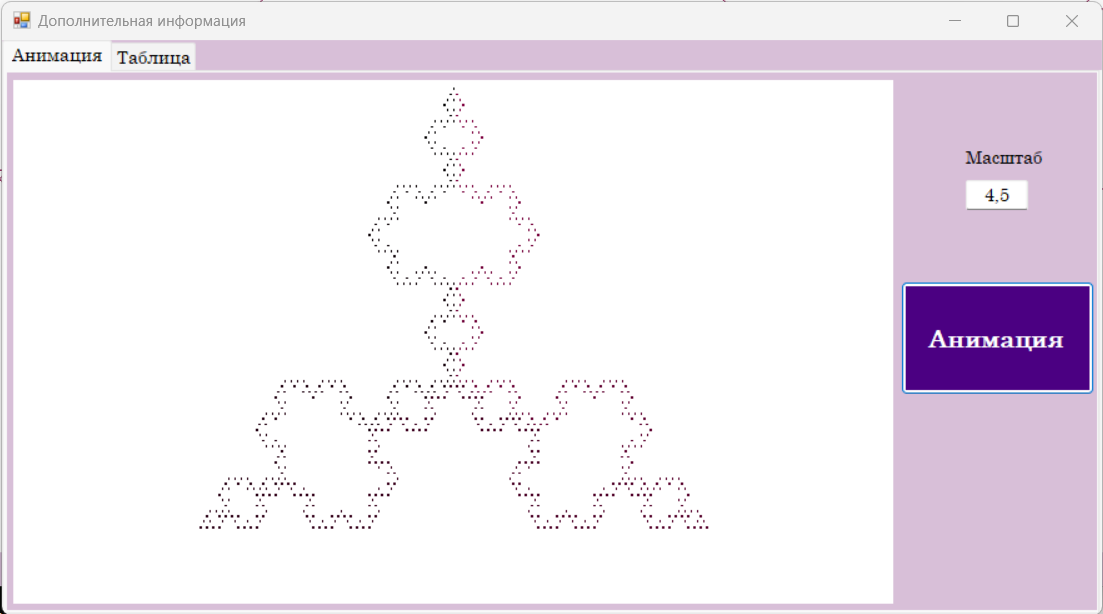


Рисунок 3.14 – Форма с анимацией.

На второй вкладке есть таблица и две кнопки для импорта ее в документ Word и Excel, что продемонстрировано на рис. 3.15.

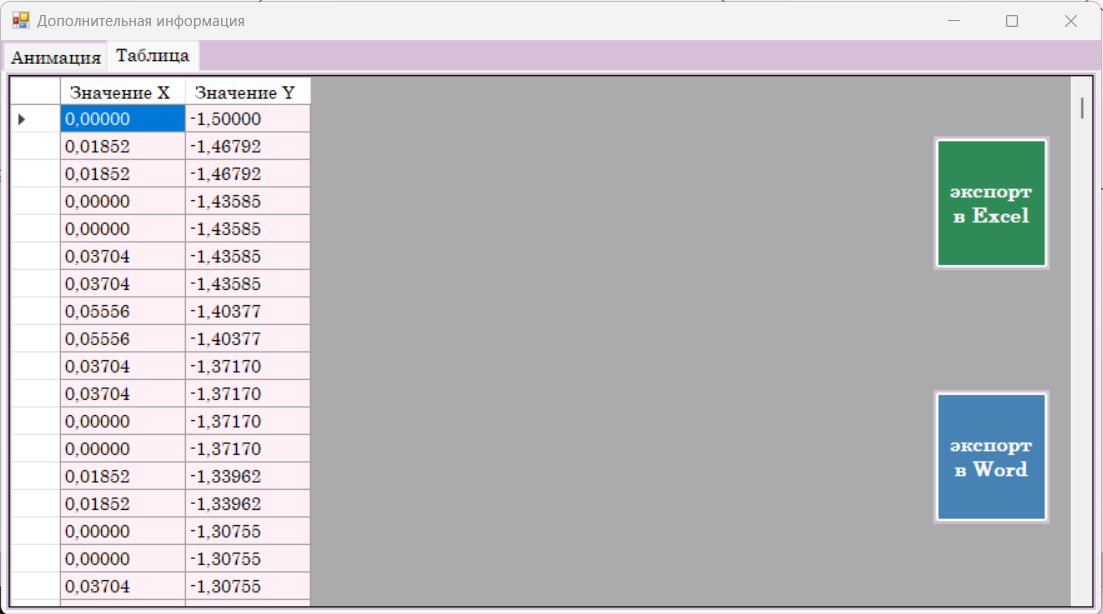


Рисунок 3.15 – Форма с таблицей.

Экспорт данных происходит по аналогии такого же процесса, описанного выше. По итогу создаются докумены, представленные на рисунках 3.16.1 и 3.16.2:

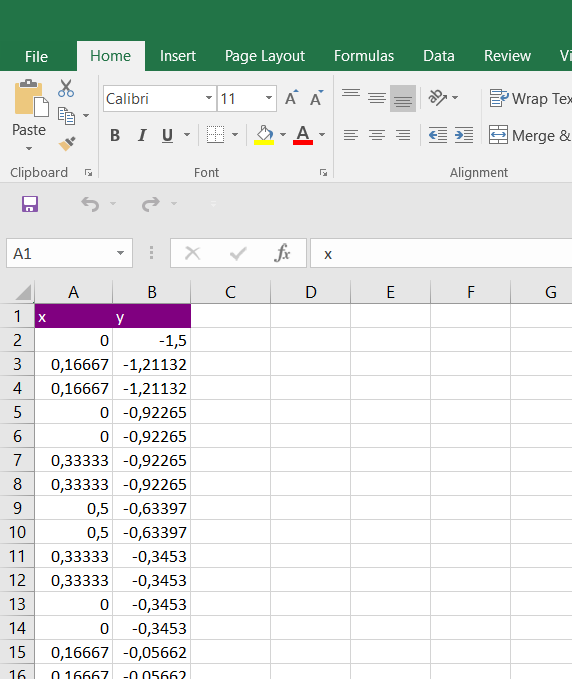


Рисунок 3.16.1 – Пример документа в Excel.

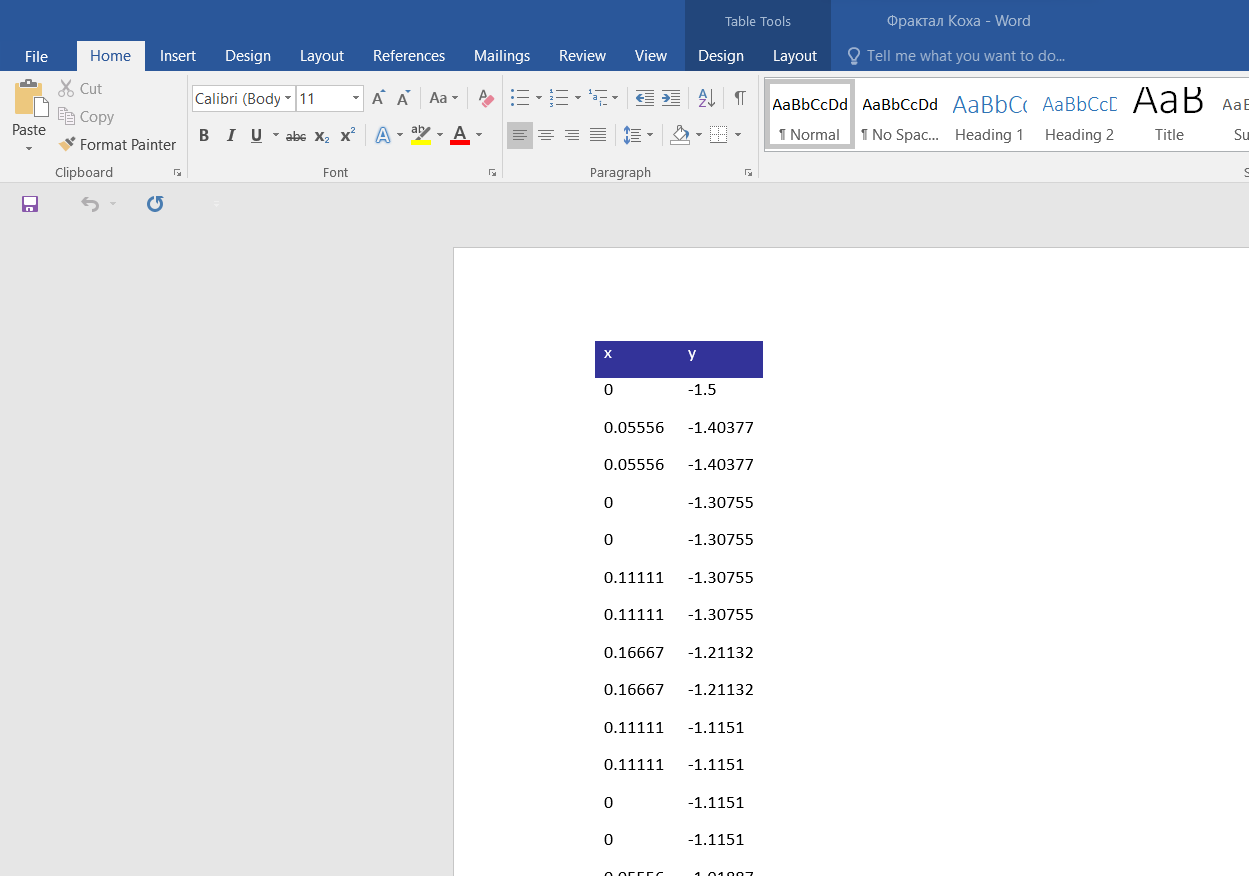


Рисунок 3.16.2 – Пример документа в Word .

# 

# **4 МЕТОДИКА ИСПЫТАНИЙ**

Важными аспектами испытаний проверки работоспособности и надежности программы в различных условиях ее функционирования являются корректность ввода исходных данных, проведение соответствующих проверок и информирование пользователя о возможных неточностях в работе программы, подсказки пользователю, а также получение последовательного и непротиворечивого результата.

Для обеспечения нормальной работы программы требуется наличие необходимых динамических библиотек, а также установленных приложений Paint, Word и Excel, которые могут быть использованы программой при необходимости.

При запуске приложения важно выбрать метод по какой формуле рассчитывать фрактал, поскольку изначально при загрузке этот параметр не задан. В таком случае, после нажатия кнопки «Create», появится предупреждение и создание фрактала не произойдёт. Результат тестирования можно увидеть на рисунке 4.1.

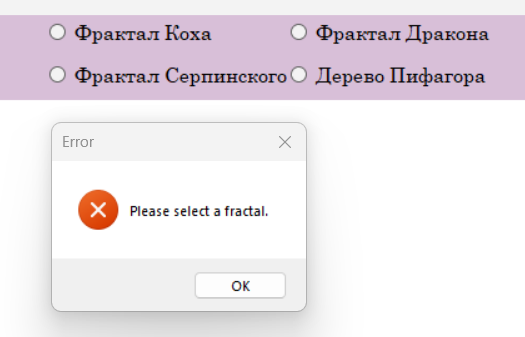


Рисунок 4.1 – Предупреждение, что не выбран тип фрактала

Так же важно проверяеть правильный диапазон данных и наличие посторонних символов, если они введены некорректно, то приложение показывает уведомление с соответсвующими подсказками, как показано на рисунках 4.2.1 - 4.2.3:

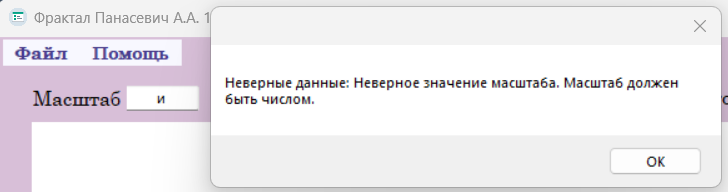


Рисунок 4.2.1 – Поведение программы при неправильном вводе типа данных

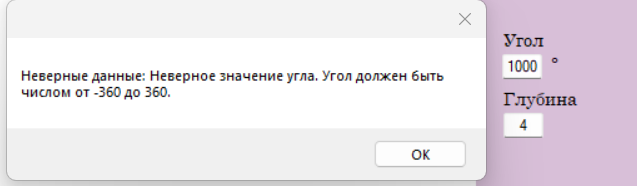


Рисунок 4.2.2 – Поведение программы при неправильном вводе диапазона значений угла

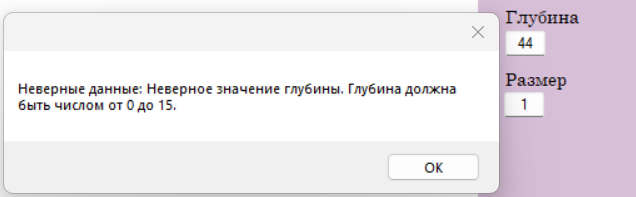


Рисунок 4.2.3 – Поведение программы при неправильном вводе значений глубины

Тестирование данного программного продукта пройдено успешно.

# **ЗАКЛЮЧЕНИЕ**

В ходе выполнения курсовой работы был закреплен опыт разработки объектно-ориентированных программ, закреплены знания, полученные при прохождении курса «Разработка приложений в визуальных средах», было изучено взаимодействие с серверами автоматизации приложений Microsoft Office, а также создание динамических библиотек.

Разработанная программа имеет удобный пользовательский интерфейс, различные формы вывода информации. Программа может использоваться для генерации фракталов для наглядной демонстрации их изменений в зависимости от вводимых параметров. Созданная в процессе разработки данного приложения библиотека может в дальнейшем использоваться как маска для генерации сложных и подробных геометрических узоров, создания компьютерной графики, например, генерации ландшафта для карты видеоигры, так же как маска для шифрования сигналов и тому подобное.

# **СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**

1. Л. Аммерал Программирование графики на СИ. Пер.с англ.-М.:"Сол Систем", 1992.-221с.
2. Биллиг В.A. Основы объектного программирования на С# (C# 3.0, Visual Studio 2008): учебное пособие / В.A. Биллиг. — Электрон. текстовые данные. — Москва, Саратов: Интернет-Университет Информационных Технологий (ИНТУИТ), Вузовское образование, 2017. — 583 c.
3. Windows Forms. Программирование на C# [Электронный ресурс]. — Режим доступа: http://csharpcoding.org/category/windows-forms/ (дата обращения: 05.3.2023).

4) Некоммерческий научно-популярный проект «Элементы большой науки» . Фракталы [Электронный ресурс]. — Режим доступа: https://elementy.ru/posters/fractals (дата обращения: 05.3.2023).

# **ПРИЛОЖЕНИЕ**

**Form1.cs**

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Drawing;

using System.Windows.Forms;

using OpenTK.Graphics.OpenGL;

using System.Threading;

using MyFractalsLibrary;

namespace Fractals

{

public partial class Form1 : Form

{

public string NameFile = "";

DateTime now;

private Color drawColor;

private Color backgroundColor = Color.White;

public Form1()

{

InitializeComponent();

}

private void Form1\_Load(object sender, EventArgs e)

{

this.Visible = false;

//создание обьекта Form\_intro(заставки)

Form\_intro newForm = new Form\_intro();

newForm.Visible = true;

newForm.Size = new Size(1205, 644);

newForm.Location = new Point(0, 0);

newForm.Show();

Thread.Sleep(2000);

newForm.Close();

this.Visible = true;

// Устанавливаем координаты окна в левый верхний угол экрана

this.Location = new System.Drawing.Point(0, 0);

GL.ClearColor(Color.White); // устанавливаем белый фон

GL.Color3(Color.Black); // устанавливаем черный цвет для рисунка

// устанавливаем невидимость параметров для каждого фрактала

label1.Visible = false;

label2.Visible = false;

label3.Visible = false;

label4.Visible = false;

label5.Visible = false;

label6.Visible = false;

label7.Visible = false;

textBoxX1.Visible = false;

textBoxY1.Visible = false;

textBoxAngle.Visible = false;

textBoxDepth.Visible = false;

textBoxSize.Visible = false;

textBoxBranchRatio.Visible = false;

}

private void paintToolStripMenuItem\_Click(object sender, EventArgs e) // пункт меню вызова Paint

{

now = DateTime.Now;

Export.ToPaint(glControl1, now);

}

private void выходToolStripMenuItem\_Click(object sender, EventArgs e) // пункт меню выход

{

DialogResult result = MessageBox.Show("Вы точно хотите выйти из программы?", "Подтвердите выход", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

if (result == DialogResult.Yes)

{

this.Close();

}

}

private void руководствоПользователяToolStripMenuItem\_Click(object sender, EventArgs e) // пункт меню руководство пользователя

{

string guidePath = @"Guide.docx";

if (System.IO.File.Exists(guidePath))

{

Process.Start(guidePath);

}

else

{

MessageBox.Show("Файл руководства пользователя не найден.");

}

}

private void оПриложенииToolStripMenuItem\_Click(object sender, EventArgs e) // пункт меню о приложении

{

About about = new About();

about.ShowDialog();

}

private void ToExcelNew\_Click(object sender, EventArgs e) // подпункт меню для создания нового файла эксель

{

if (radioButtonKoch.Checked)

{

Export.ToExcelNew(now, textBoxDepth.Text, textBoxAngle.Text, "-", "-");

}

else if (radioButtonSerpinsky.Checked)

{

Export.ToExcelNew(now, textBoxDepth.Text, "-", textBoxSize.Text, "-");

}

else if (radioButtonDragon.Checked)

{

Export.ToExcelNew(now, textBoxDepth.Text, textBoxAngle.Text, "-", "-");

}

else if (radioButtonPythagor.Checked)

{

Export.ToExcelNew(now, textBoxDepth.Text, textBoxAngle.Text, textBoxSize.Text, textBoxBranchRatio.Text);

}

else

{

MessageBox.Show("Please select a fractal.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

}

private void ToExcelAdd\_Click(object sender, EventArgs e) // подпункт меню для добавления информации в эксель

{

if (radioButtonKoch.Checked)

{

Export.ToExcelAdd(now, textBoxDepth.Text, textBoxAngle.Text, "-", "-");

}

else if (radioButtonSerpinsky.Checked)

{

Export.ToExcelAdd(now, textBoxDepth.Text, "-", textBoxSize.Text, "-");

}

else if (radioButtonDragon.Checked)

{

Export.ToExcelAdd(now, textBoxDepth.Text, textBoxAngle.Text, "-", "-");

}

else if (radioButtonPythagor.Checked)

{

Export.ToExcelAdd(now, textBoxDepth.Text, textBoxAngle.Text, textBoxSize.Text, textBoxBranchRatio.Text);

}

else

{

MessageBox.Show("Please select a fractal.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

}

private void radioButtonKoch\_CheckedChanged(object sender, EventArgs e) //выбор фрактала коха

{

// установление видимости для нужных элементов и задание параметров для примера фрактала

textBoxX1.Visible = true;

textBoxY1.Visible = true;

label4.Visible = true;

label5.Visible = true;

label1.Visible = true;

label3.Visible = true;

label2.Visible = true;

textBoxAngle.Visible = true;

label6.Visible = false;

textBoxSize.Visible = false;

label7.Visible = false;

textBoxBranchRatio.Visible = false;

textBoxDepth.Visible = true;

textBoxScale.Text = "0,5";

textBoxX1.Text = "0";

textBoxY1.Text = "-1,5";

textBoxDepth.Text = "4";

textBoxAngle.Text = "60";

}

private void radioButtonSerpinsky\_CheckedChanged(object sender, EventArgs e) //выбор фрактала серпинского

{

// установление видимости для нужных элементов и задание параметров для примера фрактала

textBoxX1.Visible = true;

textBoxY1.Visible = true;

label4.Visible = true;

label5.Visible = true;

label1.Visible = true;

label6.Visible = true;

textBoxSize.Visible = true;

label3.Visible = false;

label2.Visible = false;

textBoxAngle.Visible = false;

label7.Visible = false;

textBoxBranchRatio.Visible = false;

textBoxDepth.Visible = true;

textBoxScale.Text = "2";

textBoxX1.Text = "-0,5";

textBoxY1.Text = "-0,4";

textBoxDepth.Text = "8";

textBoxSize.Text = "1";

}

private void radioButtonDragon\_CheckedChanged(object sender, EventArgs e) //выбор фрактала дракона

{

// установление видимости для нужных элементов и задание параметров для примера фрактала

textBoxX1.Visible = true;

textBoxY1.Visible = true;

label4.Visible = true;

label5.Visible = true;

label1.Visible = true;

label6.Visible = false;

textBoxSize.Visible = false;

label3.Visible = false;

label2.Visible = true;

textBoxAngle.Visible = true;

label3.Visible = true;

label7.Visible = false;

textBoxBranchRatio.Visible = false;

textBoxDepth.Visible = true;

textBoxScale.Text = "1";

textBoxX1.Text = "-0,2";

textBoxY1.Text = "-0,4";

textBoxDepth.Text = "10";

textBoxAngle.Text = "45";

}

private void radioButtonPythagor\_CheckedChanged(object sender, EventArgs e) // выбор фрактала дерева пифагора

{

// установление видимости для нужных элементов и задание параметров для примера фрактала

textBoxX1.Visible = true;

textBoxY1.Visible = true;

label4.Visible = true;

label5.Visible = true;

label1.Visible = true;

label6.Visible = true;

textBoxSize.Visible = true;

label3.Visible = true;

label2.Visible = true;

textBoxAngle.Visible = true;

label3.Visible = true;

label7.Visible = true;

textBoxBranchRatio.Visible = true;

textBoxDepth.Visible = true;

textBoxScale.Text = "0,5";

textBoxX1.Text = "0,035";

textBoxY1.Text = "-1,5";

textBoxDepth.Text = "15";

textBoxAngle.Text = "0";

textBoxSize.Text = "1";

textBoxBranchRatio.Text = "0,7";

}

private List<Tuple<double, double>> GenerateFractalPoints() // запись в список координат для соответствующего фрактала, проверка введенных значений

{

if (!double.TryParse(textBoxX1.Text, out double x1) ||

!double.TryParse(textBoxY1.Text, out double y1))

{

throw new ArgumentException("Неверный формат ввода. Пожалуйста, введите только числа.");

}

if (radioButtonKoch.Checked)

{

if (!int.TryParse(textBoxDepth.Text, out int depth) || depth < 0 || depth > 10)

{

throw new ArgumentException("Неверное значение глубины. Глубина должна быть числом от 0 до 10.");

}

if (!double.TryParse(textBoxAngle.Text, out double angle) || angle < -360.0 || angle > 360.0)

{

throw new ArgumentException("Неверное значение угла. Угол должен быть числом от -360 до 360.");

}

KochFractal kochFractal = new KochFractal(depth, angle, x1, y1);

List<Tuple<double, double>> points = kochFractal.GeneratePoints();

// Генерация следующего фрактала Коха с координатами начала,

// которые являются координатами конца предыдущей снежинки,

// и углом, который на 120 градусов больше предыдущего, таким образом он стротся во внутрь треугольника

double x2 = points[points.Count - 1].Item1;

double y2 = points[points.Count - 1].Item2;

kochFractal = new KochFractal(depth, angle + 120.0, x2, y2);

points.AddRange(kochFractal.GeneratePoints());

// Генерация третьей снежинки Коха аналогичным образом

x2 = points[points.Count - 1].Item1;

y2 = points[points.Count - 1].Item2;

kochFractal = new KochFractal(depth, angle + 240.0, x2, y2);

points.AddRange(kochFractal.GeneratePoints());

return points;

}

else if (radioButtonSerpinsky.Checked)

{

if (!int.TryParse(textBoxDepth.Text, out int depth) || depth < 0 || depth > 15)

{

throw new ArgumentException("Неверное значение глубины. Глубина должна быть числом от 0 до 15.");

}

if (!double.TryParse(textBoxSize.Text, out double size) || size <= 0.0)

{

throw new ArgumentException("Неверное значение размера. Размер должен быть положительным числом.");

}

SierpinskiFractal serpinskyFractal = new SierpinskiFractal(depth, size, x1, y1);

return serpinskyFractal.GeneratePoints();

}

else if (radioButtonDragon.Checked)

{

if (!int.TryParse(textBoxDepth.Text, out int depth) || depth < 0 || depth > 20)

{

throw new ArgumentException("Неверное значение глубины. Глубина должна быть числом от 0 до 20.");

}

if (!double.TryParse(textBoxAngle.Text, out double angle) || angle < -360.0 || angle > 360.0)

{

throw new ArgumentException("Неверное значение угла. Угол должен быть числом от -360 до 360.");

}

DragonFractal dragonFractal = new DragonFractal(depth, angle, x1, y1);

return dragonFractal.GeneratePoints();

}

else if (radioButtonPythagor.Checked)

{

if (!int.TryParse(textBoxDepth.Text, out int depth) || depth < 0 || depth > 20)

{

throw new ArgumentException("Неверное значение глубины. Глубина должна быть числом от 0 до 20.");

}

if (!double.TryParse(textBoxAngle.Text, out double angle) || angle < -360.0 || angle > 360.0)

{

throw new ArgumentException("Неверное значение угла. Угол должен быть числом от -360 до 360.");

}

if (!double.TryParse(textBoxSize.Text, out double size) || size <= -10.0 || size >= 10.0)

{

throw new ArgumentException("Неверное значение размера. Размер должен быть числом от -10 до 10.");

}

if (!double.TryParse(textBoxBranchRatio.Text, out double branchRatio) || branchRatio <= 0.0 || branchRatio >= 1.0)

{

throw new ArgumentException("Неверное значение коэффициента ветвления. Коэффициент ветвления должен быть числом от 0 до 1.");

}

PythagorasTreeFractal pythagorasFractal = new PythagorasTreeFractal(depth, size, x1, y1, angle, branchRatio);

return pythagorasFractal.GeneratePoints();

}

else

{

throw new ArgumentException("Пожалуйста, выберите фрактал.");

}

}

private void buttonCreate\_Click(object sender, EventArgs e) // вызов отрисовки графика для фрактала

{

try

{

double scale;

if (!double.TryParse(textBoxScale.Text, out scale))

{

throw new ArgumentException("Неверное значение масштаба. Масштаб должен быть числом.");

}

List<Tuple<double, double>> points = GenerateFractalPoints();

DrawFractal(points, scale);

if (checkBox1.Checked)

{

// Создание нового окна с графиком

FormGraph newGraphForm = new FormGraph(points, drawColor, backgroundColor);

newGraphForm.Show();

}

}

catch (Exception ex)

{

MessageBox.Show("Неверные данные: " + ex.Message);

}

}

private void DrawFractal(List<Tuple<double, double>> points, double scale) //отрисовка фрактала на glControl

{

GL.Clear(ClearBufferMask.ColorBufferBit | ClearBufferMask.DepthBufferBit);

GL.LoadIdentity();

GL.Scale(scale, scale, 1.0);

GL.Begin(PrimitiveType.LineStrip);

foreach (Tuple<double, double> point in points)

{

GL.Vertex2(point.Item1, point.Item2);

}

GL.End();

glControl1.SwapBuffers();

}

private void Clear\_Click(object sender, EventArgs e) // очистка glControl

{

glControl1.MakeCurrent();

GL.ClearColor(backgroundColor.R / 255f, backgroundColor.G / 255f, backgroundColor.B / 255f, 1.0f);

GL.Clear(ClearBufferMask.ColorBufferBit | ClearBufferMask.DepthBufferBit);

glControl1.SwapBuffers();

}

private void glControl1\_Load(object sender, EventArgs e) // загрузка glControl

{

GL.Enable(EnableCap.DepthTest);

GL.Viewport(0, 0, glControl1.Width, glControl1.Height);

GL.MatrixMode(MatrixMode.Projection);

GL.LoadIdentity();

GL.Ortho(-1, 1, -1, 1, -1, 1);

}

private void glControl1\_Resize(object sender, EventArgs e)

{

GL.Viewport(0, 0, glControl1.Width, glControl1.Height);

}

private void Form1\_Resize(object sender, EventArgs e)

{

// Force a redraw of the glControl1 control

glControl1.Invalidate();

}

private void buttonChooseDrawColor\_Click(object sender, EventArgs e) //выбор цвета пера

{

if (colorDialogDraw.ShowDialog() == DialogResult.OK)

{

drawColor = colorDialogDraw.Color;

GL.Color3(drawColor.R / 255f, drawColor.G / 255f, drawColor.B / 255f);

// Установка цвета кнопки выбранным цветом

buttonChooseDrawColor.BackColor = drawColor;

}

}

private void buttonChooseBackgroundColor\_Click(object sender, EventArgs e) //выбор цвета фона

{

if (colorDialogBackground.ShowDialog() == DialogResult.OK)

{

backgroundColor = colorDialogBackground.Color;

GL.ClearColor(backgroundColor.R / 255f, backgroundColor.G / 255f, backgroundColor.B / 255f, 1.0f);

// Установка цвета кнопки выбранным цветом

buttonChooseBackgroundColor.BackColor = backgroundColor;

}

}

}

}

**Export.cs**

**using** System;

**using** System.Diagnostics;

**using** System.Drawing.Imaging;

**using** System.Drawing;

**using** System.Runtime.InteropServices;

**using** System.Windows.Forms;

**using** Excel = Microsoft.Office.Interop.Excel;

**namespace** Fractals

{

**public** **partial** **class** **Export** : Form

{

**public** **static** **string** NameFile = "";

**public** **static** **void** ToPaint(Control control, DateTime now) // создание и сохранение скриншота экрана с графиком

{

// Захват области клиентской области элемента управления GLControl

**var** glControl = (OpenTK.GLControl)control;

glControl.MakeCurrent();

Form parentForm = control.FindForm();

System.Threading.Thread.Sleep(500); // Пауза

**var** bmp = **new** Bitmap(1239, 589);

**var** gfx = Graphics.FromImage(bmp);

gfx.CopyFromScreen(parentForm.PointToScreen(**new** Point(control.Location.X + 9, control.Location.Y + 24)), **new** Point(0, 0), bmp.Size);

// Сохранение скриншота в файл

**using** (SaveFileDialog dialog = **new** SaveFileDialog())

{

dialog.Filter = "PNG Files (\*.png)|\*.png";

dialog.DefaultExt = "png";

dialog.AddExtension = **true**;

**if** (dialog.ShowDialog() == DialogResult.OK)

{

bmp.Save(dialog.FileName, ImageFormat.Png);

NameFile = dialog.FileName;

now = DateTime.Now;

// Открытие скриншота в Paint

Process.Start("mspaint.exe", dialog.FileName);

}

}

}

**public** **static** **void** excelApp\_SheetFollowHyperlink(**object** Sh, Excel.Hyperlink Target) // создание гипперссылки в виде ссылки на изображение по адресу расположения изображения

{

**if** (Target != **null**)

{

**string** address = Target.Address;

**if** (!**string**.IsNullOrEmpty(address))

{

**try**

{

System.Diagnostics.Process.Start(address);

}

**catch** (Exception ex)

{

MessageBox.Show($"An error occurred while trying to open the file: {ex.Message}");

}

}

}

}

**public** **static** **void** ToExcelNew(DateTime now, **string** depth, **string** angle, **string** size, **string** ratio) // создание и открытие нового эксель документа

{

Excel.Application excelApp = **new** Excel.Application();

excelApp.SheetFollowHyperlink += excelApp\_SheetFollowHyperlink;

Excel.Workbook workbook = **null**;

Excel.Worksheet worksheet = **null**;

**try**

{

workbook = excelApp.Workbooks.Add(Type.Missing);

worksheet = workbook.Sheets[1];

worksheet.Name = "Data";

worksheet.Cells[1, 1] = "Creation date";

worksheet.Cells[1, 2] = "Name of the image";

worksheet.Cells[1, 3] = "Depth";

worksheet.Cells[1, 4] = "Angle in °";

worksheet.Cells[1, 5] = "Size";

worksheet.Cells[1, 6] = "Branch ratio";

**int** lastRow = worksheet.Cells.SpecialCells(Excel.XlCellType.xlCellTypeLastCell).Row + 1;

worksheet.Cells[lastRow, 1] = now.ToString("yyyy-MM-dd HH:mm:ss");

worksheet.Cells[lastRow, 2] = $"=HYPERLINK(\"{NameFile.Replace("\\", "/")}\")";

worksheet.Cells[lastRow, 3] = depth;

worksheet.Cells[lastRow, 4] = angle;

worksheet.Cells[lastRow, 5] = size;

worksheet.Cells[lastRow, 6] = ratio;

worksheet.Columns.AutoFit();

SaveFileDialog sfd = **new** SaveFileDialog();

sfd.Filter = "Microsoft Excel Sheet (.xlsx)|\*.xlsx";

sfd.FileName = "File Name.xlsx";

**if** (sfd.ShowDialog() == DialogResult.OK)

{

workbook.SaveAs(sfd.FileName);

MessageBox.Show("File successfully created!", "Notification");

Process.Start(sfd.FileName);

workbook.Save();

workbook.Close();

excelApp.Quit();

Process.Start(sfd.FileName);

}

NameFile = "";

}

**catch** (Exception ex)

{

MessageBox.Show($"An error occurred while trying to create the Excel file: {ex.Message}");

}

**finally**

{

// Освобождение COM-объектов и закрытие приложения Excel

**if** (worksheet != **null**) Marshal.ReleaseComObject(worksheet);

**if** (workbook != **null**) Marshal.ReleaseComObject(workbook);

**if** (excelApp != **null**) Marshal.ReleaseComObject(excelApp);

GC.Collect();

GC.WaitForPendingFinalizers();

}

}

**public** **static** **void** ToExcelAdd(DateTime now, **string** depth, **string** angle, **string** size, **string** ratio) // добавление информации в уже существующий файл эксель

{

Excel.Application excelApp = **new** Excel.Application();

excelApp.SheetFollowHyperlink += excelApp\_SheetFollowHyperlink;

OpenFileDialog ofd = **new** OpenFileDialog();

ofd.Filter = "Microsoft Excel Sheet (.xlsx)|\*.xlsx";

**if** (ofd.ShowDialog() == DialogResult.OK)

{

Excel.Workbook workbook = **null**;

Excel.Worksheet worksheet = **null**;

**try**

{

workbook = excelApp.Workbooks.Open(ofd.FileName);

worksheet = workbook.Sheets[1];

**int** lastRow = worksheet.Cells.SpecialCells(Excel.XlCellType.xlCellTypeLastCell).Row + 1;

worksheet.Cells[lastRow, 1] = now.ToString("yyyy-MM-dd HH:mm:ss");

worksheet.Cells[lastRow, 2] = $"=HYPERLINK(\"{NameFile.Replace("\\", "/")}\")";

worksheet.Cells[lastRow, 3] = depth;

worksheet.Cells[lastRow, 4] = angle;

worksheet.Cells[lastRow, 5] = size;

worksheet.Cells[lastRow, 6] = ratio;

worksheet.Columns.AutoFit();

workbook.Save();

workbook.Close();

excelApp.Quit();

Process.Start(ofd.FileName);

}

**catch** (Exception ex)

{

MessageBox.Show($"An error occurred while trying to add data to the Excel file: {ex.Message}");

}

**finally**

{

// Освобождение COM-объектов и закрытие приложения Excel

**if** (worksheet != **null**) Marshal.ReleaseComObject(worksheet);

**if** (workbook != **null**) Marshal.ReleaseComObject(workbook);

**if** (excelApp != **null**) Marshal.ReleaseComObject(excelApp);

GC.Collect();

GC.WaitForPendingFinalizers();

}

}

NameFile = "";

}

}

}

**FormGraph.cs**

**using** System;

**using** System.Collections.Generic;

**using** System.ComponentModel;

**using** System.Data;

**using** System.Drawing;

**using** System.Linq;

**using** System.Text;

**using** System.Threading.Tasks;

**using** Excel = Microsoft.Office.Interop.Excel;

**using** Word = Microsoft.Office.Interop.Word;

**using** System.Windows.Forms;

**using** System.Diagnostics;

**using** System.Runtime.InteropServices;

**using** System.Globalization;

**using** System.Threading;

**namespace** Fractals

{

**public** **partial** **class** **FormGraph** : Form

{

**private** List<Tuple<**double**, **double**>> points;

**private** **double** currentScale;

Color drawColor, backgroundColor;

**public** FormGraph(List<Tuple<**double**, **double**>> points, Color drawColor, Color backgroundColor)

{

InitializeComponent();

**this**.points = points;

**this**.drawColor = drawColor;

**this**.backgroundColor = backgroundColor;

// Создаем таблицу и добавляем ее на вкладку tabPageTable

DataGridView dataGridView1 = **new** DataGridView();

dataGridView1.Dock = DockStyle.Fill;

dataGridView1.ScrollBars = ScrollBars.Vertical;

// Задаем цвет фона для таблицы

dataGridView1.DefaultCellStyle.BackColor = Color.LavenderBlush;

// Задаем цвет фона для заголовков столбцов

dataGridView1.ColumnHeadersDefaultCellStyle.BackColor = Color.MediumOrchid;

tabPageTable.Controls.Add(dataGridView1);

// Добавляем столбцы "x" и "y" в таблицу

dataGridView1.Columns.Add("xColumn", "Значение X");

dataGridView1.Columns.Add("yColumn", "Значение Y");

// Добавляем значения точек в таблицу

**foreach** (**var** point **in** points)

{

dataGridView1.Rows.Add(point.Item1.ToString("N5"), point.Item2.ToString("N5"));

}

}

**private** **void** buttonExport\_Click(**object** sender, EventArgs e) //кнопка экспорта в эксель

{

Excel.Application excelApp = **null**;

Excel.Workbook workbook = **null**;

Excel.Worksheet worksheet = **null**;

**try**

{

// Создаем экземпляр приложения Excel

excelApp = **new** Excel.Application();

// Создаем новую рабочую книгу

workbook = excelApp.Workbooks.Add();

// Получаем первый лист рабочей книги

worksheet = (Excel.Worksheet)workbook.Sheets[1];

worksheet.Cells[1, 1].Interior.Color = 8388736; // Фиолетовый цвет заливки

worksheet.Cells[1, 2].Interior.Color = 8388736;

worksheet.Cells[1, 1].Font.Color = 16777215; // Белый цвет шрифта

worksheet.Cells[1, 2].Font.Color = 16777215;

// Добавляем заголовки столбцов

worksheet.Cells[1, 1] = "x";

worksheet.Cells[1, 2] = "y";

// Заполняем лист данными из DataGridView

**for** (**int** i = 0; i < points.Count; i++)

{

worksheet.Cells[i + 2, 1] = points[i].Item1.ToString("0.#####", CultureInfo.InvariantCulture);

worksheet.Cells[i + 2, 2] = points[i].Item2.ToString("0.#####", CultureInfo.InvariantCulture);

}

// Автоматически подгоняем ширину столбцов

worksheet.Columns.AutoFit();

// Отображаем диалог сохранения файла, чтобы получить имя файла и путь для сохранения Excel-файла

SaveFileDialog sfd = **new** SaveFileDialog();

sfd.Filter = "Microsoft Excel Sheet (.xlsx)|\*.xlsx";

sfd.FileName = "File Name.xlsx";

**if** (sfd.ShowDialog() == DialogResult.OK)

{

// Сохраняем рабочую книгу

workbook.SaveAs(sfd.FileName);

// Закрываем рабочую книгу

workbook.Close();

// Завершаем работу приложения Excel

excelApp.Quit();

// Открываем сохраненный Excel-файл в приложении Excel

Process.Start(sfd.FileName);

}

}

**catch** (Exception ex)

{

MessageBox.Show($"Произошла ошибка при создании Excel-файла: {ex.Message}");

}

**finally**

{

// Освобождаем COM-объекты и закрываем приложение Excel

**if** (worksheet != **null**) Marshal.ReleaseComObject(worksheet);

**if** (workbook != **null**) Marshal.ReleaseComObject(workbook);

**if** (excelApp != **null**) Marshal.ReleaseComObject(excelApp);

GC.Collect();

GC.WaitForPendingFinalizers();

}

}

**private** **void** buttonWord\_Click(**object** sender, EventArgs e) //экспорт в ворд

{

Word.Application wordApp = **null**;

Word.Document document = **null**;

**try**

{

// Создаем экземпляр приложения Word

wordApp = **new** Word.Application();

// Создаем новый документ

document = wordApp.Documents.Add();

// Добавляем заголовки столбцов

Word.Table table = document.Tables.Add(document.Range(), points.Count + 1, 2);

table.Cell(1, 1).Range.Text = "x";

table.Cell(1, 2).Range.Text = "y";

// Задаем цвет фона и цвет шрифта для заголовков

table.Cell(1, 1).Range.Shading.BackgroundPatternColor = Word.WdColor.wdColorIndigo;

table.Cell(1, 2).Range.Shading.BackgroundPatternColor = Word.WdColor.wdColorIndigo;

table.Cell(1, 1).Range.Font.Color = Word.WdColor.wdColorWhite;

table.Cell(1, 2).Range.Font.Color = Word.WdColor.wdColorWhite;

// Заполняем таблицу данными из points

**for** (**int** i = 0; i < points.Count; i++)

{

table.Cell(i + 2, 1).Range.Text = points[i].Item1.ToString("0.#####", CultureInfo.InvariantCulture);

table.Cell(i + 2, 2).Range.Text = points[i].Item2.ToString("0.#####", CultureInfo.InvariantCulture);

}

// Автоматически подгоняем ширину столбцов

table.AutoFitBehavior(Word.WdAutoFitBehavior.wdAutoFitContent);

// Отображаем диалог сохранения файла, чтобы получить имя файла и путь для сохранения Word-документа

SaveFileDialog sfd = **new** SaveFileDialog();

sfd.Filter = "Microsoft Word Document (.docx)|\*.docx";

sfd.FileName = "File Name.docx";

**if** (sfd.ShowDialog() == DialogResult.OK)

{

// Сохраняем документ

document.SaveAs2(sfd.FileName);

// Закрываем документ

document.Close();

// Завершаем работу приложения Word

wordApp.Quit();

// Открываем сохраненный Word-документ в приложении Word

Process.Start(sfd.FileName);

}

}

**catch** (Exception ex)

{

MessageBox.Show($"Произошла ошибка при создании Word-документа: {ex.Message}");

}

**finally**

{

// Освобождаем COM-объекты и закрываем приложение Word

**if** (document != **null**) Marshal.ReleaseComObject(document);

**if** (wordApp != **null**) Marshal.ReleaseComObject(wordApp);

GC.Collect();

GC.WaitForPendingFinalizers();

}

}

**private** **void** buttonAnimation\_Click(**object** sender, EventArgs e) //кнопка создания и запуска анимации

{

Graphics graphics = PanelDraw.CreateGraphics();

// Очищаем панель перед началом анимации

graphics.Clear(Color.White);

// Определяем задержку между кадрами анимации (в миллисекундах) в зависимости от количества точек

**int** totalPoints = points.Count;

**int** maxDelay = 25;

**int** delay = maxDelay / totalPoints;

// Получаем значение масштаба из текстового поля

**double** scale;

**if** (!**double**.TryParse(textBoxScale.Text, **out** scale))

{

MessageBox.Show("Пожалуйста, введите допустимое значение масштаба.");

**return**;

}

// Обновляем текущий масштаб

currentScale = scale;

// Размер точки

**int** pointSize = 2;

// Определение градиента цветов

Color startColor = drawColor;

Color endColor;

**if** (backgroundColor == Color.White)

{

endColor = Color.Black;

}

**else**

{

endColor = backgroundColor;

}

**for** (**int** i = 0; i < points.Count; i++)

{

Tuple<**double**, **double**> point = points[i];

// Интерполяция цвета между начальным и конечным цветом на основе позиции

**double** position = (**double**)i / (points.Count - 1); // Позиция точки в списке (от 0 до 1)

**int** red = (**int**)(startColor.R + (endColor.R - startColor.R) \* position);

**int** green = (**int**)(startColor.G + (endColor.G - startColor.G) \* position);

**int** blue = (**int**)(startColor.B + (endColor.B - startColor.B) \* position);

Color pointColor = Color.FromArgb(red, green, blue);

Brush brush = **new** SolidBrush(pointColor);

// Рисуем точку с учетом текущего масштаба и цвета

graphics.FillEllipse(brush, (**float**)((point.Item1 \* currentScale \* 30) - pointSize / 2 + PanelDraw.Width / 2), (**float**)((point.Item2 \* currentScale \* 30) - pointSize / 2 + PanelDraw.Height / 2), pointSize, pointSize);

// Задержка между кадрами

Thread.Sleep(delay);

}

}

// Обработчик изменения содержимого текстового поля масштаба

**private** **void** textBoxScale\_TextChanged(**object** sender, EventArgs e) //изменение масштаба

{

// Обновляем текущий масштаб

**double** scale;

**if** (**double**.TryParse(textBoxScale.Text, **out** scale))

{

currentScale = scale;

}

}

}

}

**SierpinskiFractal.cs**

**using** System;

**using** System.Collections.Generic;

**namespace** MyFractalsLibrary

{

**public** **class** **SierpinskiFractal**

{

**public** **int** Depth { **get**; **set**; } // Глубина фрактала

**public** **double** Size { **get**; **set**; } // Размер

**public** **double** X1 { **get**; **set**; } // Координата X начальной точки

**public** **double** Y1 { **get**; **set**; } // Координата Y начальной точки

**public** SierpinskiFractal(**int** depth, **double** size, **double** x1, **double** y1)

{

Depth = depth;

Size = size;

X1 = x1;

Y1 = y1;

}

**public** List<Tuple<**double**, **double**>> GeneratePoints()

{

**var** points = **new** List<Tuple<**double**, **double**>>();

**double** size = Size;

**double** x1 = X1;

**double** y1 = Y1;

// Генерация фрактала Серпинского

GenerateSierpinski(points, x1, y1, size, Depth);

**return** points;

}

**private** **void** GenerateSierpinski(List<Tuple<**double**, **double**>> points, **double** x1, **double** y1, **double** size, **int** depth)

{

**if** (depth == 0)

{

// Рисуем треугольник

points.Add(**new** Tuple<**double**, **double**>(x1, y1));

points.Add(**new** Tuple<**double**, **double**>(x1 + size, y1));

points.Add(**new** Tuple<**double**, **double**>(x1 + size / 2.0, y1 + size \* Math.Sqrt(3.0) / 2.0));

}

**else**

{

**double** size2 = size / 2.0;

**double** x2 = x1 + size2;

**double** y2 = y1;

**double** x3 = x1 + size2 / 2.0;

**double** y3 = y1 + size2 \* Math.Sqrt(3.0) / 2.0;

// Рекурсивно рисуем три меньших треугольника

GenerateSierpinski(points, x1, y1, size2, depth - 1);

GenerateSierpinski(points, x2, y2, size2, depth - 1);

GenerateSierpinski(points, x3, y3, size2, depth - 1);

}

}

}

}

**PythagorasTreeFractal.cs**

**using** System.Collections.Generic;

**using** System;

**namespace** MyFractalsLibrary

{

**public** **class** **PythagorasTreeFractal**

{

**public** **int** Depth { **get**; **set**; } // Глубина фрактала

**public** **double** Size { **get**; **set**; } // Размер исходной линии

**public** **double** X1 { **get**; **set**; } // Координата X начала линии

**public** **double** Y1 { **get**; **set**; } // Координата Y начала линии

**public** **double** Angle { **get**; **set**; } // Угол наклона исходной линии

**public** **double** BranchRatio { **get**; **set**; } // Коэффициент масштабирования ветвей

**public** PythagorasTreeFractal(**int** depth, **double** size, **double** x1, **double** y1, **double** angle, **double** branchRatio)

{

Depth = depth;

Size = size;

X1 = x1;

Y1 = y1;

Angle = angle;

BranchRatio = branchRatio;

}

**public** List<Tuple<**double**, **double**>> GeneratePoints()

{

**var** points = **new** List<Tuple<**double**, **double**>>();

**double** size = Size;

**double** x1 = X1;

**double** y1 = Y1;

**double** angle = Angle;

GenerateTree(points, x1, y1, size, angle, Depth);

**return** points;

}

**private** **void** GenerateTree(List<Tuple<**double**, **double**>> points, **double** x1, **double** y1, **double** size, **double** angle, **int** depth)

{

**if** (depth == 0)

{

// Рисуем лист

points.Add(**new** Tuple<**double**, **double**>(x1, y1));

}

**else**

{

**double** x2 = x1 + size \* Math.Sin(angle \* Math.PI / 180.0);

**double** y2 = y1 + size \* Math.Cos(angle \* Math.PI / 180.0);

// Рисуем ствол

points.Add(**new** Tuple<**double**, **double**>(x1, y1));

points.Add(**new** Tuple<**double**, **double**>(x2, y2));

// Вычисляем две ветви

**double** size1 = size \* BranchRatio;

**double** size2 = size \* Math.Sqrt(1 - BranchRatio \* BranchRatio);

**double** angle1 = angle + 45;

**double** angle2 = angle - 45;

// Рекурсивно рисуем две ветви

GenerateTree(points, x2, y2, size1, angle1, depth - 1);

GenerateTree(points, x2, y2, size2, angle2, depth - 1);

}

}

}

}

**KochFractal.cs**

**using** System;

**using** System.Collections.Generic;

**namespace** MyFractalsLibrary

{

**public** **class** **KochFractal**

{

**public** **int** Depth { **get**; **set**; } // Глубина фрактала

**public** **double** Angle { **get**; **set**; } // Угол наклона

**public** **double** X1 { **get**; **set**; } // Координата X начальной точки

**public** **double** Y1 { **get**; **set**; } // Координата Y начальной точки

**public** KochFractal(**int** depth, **double** angle, **double** x1, **double** y1)

{

Depth = depth;

Angle = angle;

X1 = x1;

Y1 = y1;

}

**public** List<Tuple<**double**, **double**>> GeneratePoints()

{

**var** points = **new** List<Tuple<**double**, **double**>>();

**double** length = 1.0; // Длина изначальной линии

**double** theta = Angle \* Math.PI / 180.0; // Угол наклона в радианах

// Начальная точка

**double** x1 = X1;

**double** y1 = Y1;

// Конечная точка

**double** x2 = length \* Math.Cos(theta) + X1;

**double** y2 = length \* Math.Sin(theta) + Y1;

// Генерация фрактала Коха

GenerateKochCurve(points, x1, y1, x2, y2, Depth);

x1 = x2;

y1 = y2;

x2 = length \* Math.Cos(theta + 60.0 \* Math.PI / 180.0) + x1;

y2 = length \* Math.Sin(theta + 60.0 \* Math.PI / 180.0) + y1;

GenerateKochCurve(points, x1, y1, x2, y2, Depth);

x1 = x2;

y1 = y2;

x2 = length \* Math.Cos(theta - 60.0 \* Math.PI / 180.0) + x1;

y2 = length \* Math.Sin(theta - 60.0 \* Math.PI / 180.0) + y1;

GenerateKochCurve(points, x1, y1, x2, y2, Depth);

x1 = x2;

y1 = y2;

x2 = length \* Math.Cos(theta) + x1;

y2 = length \* Math.Sin(theta) + y1;

GenerateKochCurve(points, x1, y1, x2, y2, Depth);

**return** points;

}

**private** **void** GenerateKochCurve(List<Tuple<**double**, **double**>> points, **double** x1, **double** y1, **double** x2, **double** y2, **int** depth)

{

**if** (depth == 0)

{

// Добавляем начальную и конечную точки линии

points.Add(**new** Tuple<**double**, **double**>(x1, y1));

points.Add(**new** Tuple<**double**, **double**>(x2, y2));

}

**else**

{

// Вычисляем промежуточные точки линии

**double** deltaX = (x2 - x1) / 3.0;

**double** deltaY = (y2 - y1) / 3.0;

**double** x1n = x1 + deltaX;

**double** y1n = y1 + deltaY;

**double** x2n = x2 - deltaX;

**double** y2n = y2 - deltaY;

**double** x3 = x1n + (x2n - x1n) \* Math.Cos(Math.PI / 3) - (y2n - y1n) \* Math.Sin(Math.PI / 3);

**double** y3 = y1n + (y2n - y1n) \* Math.Cos(Math.PI / 3) + (x2n - x1n) \* Math.Sin(Math.PI / 3);

// Рекурсивно генерируем фрактал для каждой из четырех новых линий

GenerateKochCurve(points, x1, y1, x1n, y1n, depth - 1);

GenerateKochCurve(points, x1n, y1n, x3, y3, depth - 1);

GenerateKochCurve(points, x3, y3, x2n, y2n, depth - 1);

GenerateKochCurve(points, x2n, y2n, x2, y2, depth - 1);

}

}

}

}

**DragonFractal.cs**

**using** System;

**using** System.Collections.Generic;

**namespace** MyFractalsLibrary

{

**public** **class** **DragonFractal**

{

**public** **int** Depth { **get**; **set**; } // Глубина фрактала

**public** **double** Angle { **get**; **set**; } // Угол наклона

**public** **double** X1 { **get**; **set**; } // Координата X начальной точки

**public** **double** Y1 { **get**; **set**; } // Координата Y начальной точки

**public** DragonFractal(**int** depth, **double** angle, **double** x1, **double** y1)

{

Depth = depth;

Angle = angle;

X1 = x1;

Y1 = y1;

}

**public** List<Tuple<**double**, **double**>> GeneratePoints()

{

**var** points = **new** List<Tuple<**double**, **double**>>();

**double** length = 1.0; // Длина изначальной линии

**double** theta = Angle \* Math.PI / 180.0; // Угол наклона в радианах

// Начальная точка

**double** x1 = X1;

**double** y1 = Y1;

// Конечная точка

**double** x2 = length \* Math.Cos(theta) + X1;

**double** y2 = length \* Math.Sin(theta) + Y1;

// Генерация фрактала Дракона

GenerateDragonCurve(points, x1, y1, x2, y2, Depth);

**return** points;

}

**private** **void** GenerateDragonCurve(List<Tuple<**double**, **double**>> points, **double** x1, **double** y1, **double** x2, **double** y2, **int** depth)

{

**if** (depth == 0)

{

// Добавляем начальную и конечную точки линии

points.Add(**new** Tuple<**double**, **double**>(x1, y1));

points.Add(**new** Tuple<**double**, **double**>(x2, y2));

}

**else**

{

// Вычисляем координаты промежуточной точки

**double** deltaX = x2 - x1;

**double** deltaY = y2 - y1;

**double** x3 = x1 + deltaX / 2.0 - deltaY / 2.0;

**double** y3 = y1 + deltaY / 2.0 + deltaX / 2.0;

// Рекурсивно генерируем фрактал для двух новых линий

GenerateDragonCurve(points, x1, y1, x3, y3, depth - 1);

GenerateDragonCurve(points, x2, y2, x3, y3, depth - 1);

}

}

}

}