|  |
| --- |
| ВШЭ |
| **Пояснительная записка** |
| **Программа для построения фракталов Кривая Гильберта, С-Кривая Леви, Множество Кантора** |
|  |
| **Герасименко Е.Р. БПИ-183 вариант 38** |
| **09.12.2018** |

|  |
| --- |
| [Введите аннотацию документа. Аннотация обычно представляет собой краткий обзор содержимого документа. Введите аннотацию документа. Аннотация обычно представляет собой краткий обзор содержимого документа.] |

**ТИТУЛЬНЫЙ ЛИСТ**

**Пояснительная записка**

**Программа для построения фракталов**

**Кривая Гильберта, С-Кривая Леви, Множество Кантора**

**Выполнил Герасименко Е.Р. БПИ-183 вариант 38**

Оглавление

[1. Условие задачи 3](#_Toc532261353)

[2. Функции разрабатываемого приложения 4](#_Toc532261354)

[2.1. Варианты использования 4](#_Toc532261355)

[2.2. Описание интерфейса пользователя 4](#_Toc532261356)

[3. Структура приложения 6](#_Toc532261357)

[4. Распределение исходного кода по файлам проекта 10](#_Toc532261358)

[5. Описание интерфейса 10](#_Toc532261359)

[6. Контрольный пример и описание результата 11](#_Toc532261360)

[7. Сообщения пользователю 15](#_Toc532261361)

# 1. Условие задачи

Разработать оконное приложение (Windows Forms Application (изучается на семинарах и лекциях) или WPF (изучается самостоятельно)), позволяющее:

1. Отрисовывать три вида фракталов, которые определены в индивидуальном варианте. Вариант 38: Кривая Гильберта, С-Кривая Леви, Множество Кантора.
2. Предоставлять пользователю выбор текущего фрактала для отрисовки.
3. Предоставлять пользователю возможность устанавливать количество шагов

рекурсии (её глубину - количество рекурсивных вызовов). При изменении глубины рекурсии фрактал должен быть автоматически перерисован. Следите за переполнением стека.

1. Автоматически перерисовывать фрактал при изменении размеров окна. Окно обязательно должно быть масштабируемым. Вы можете задать минимальный и максимальный размер окна. Максимальным считается размер окна, соответствующий размеру экрана, а минимальным размером окна считается половинный размер экрана (как по длине, так и по ширине).
2. Предоставлять пользователю возможность выбора двух цветов startColor и

endColor. Цвет startColor используется для отрисовки элементов первой итерации,

цвет endColor - для отрисовки элементов последней итерации. Цвета для

промежуточных итераций должны вычисляться с использованием линейного

градиента. http://qaru.site/questions/222000/generate-color-gradient-in-c

Например,



Рис. 1. Градиент цвета

Желтый цвет используется для отрисовки элементов первой итерации. Синий - для

отрисовки элементов последней итерации.

Промежуточные цвета вычисляются исходя из начального и конечного значений

цвета и номера итерации (не генерируются случайным образом).

1. Сообщать о некорректном вводе данных, противоречивых или недопустимых

значениях данных и других нештатных ситуациях во всплывающих окнах типа

MessageBox.

1. Должна быть предусмотрена возможность сохранения фрактала в виде картинки

(формат выбрать самостоятельно).

1. Предусмотреть возможность изменения масштаба фрактала для его детального

просмотра. Увеличение должно быть 2, 3 и 5-кратным

1. Предусмотреть возможность перемещения изображения, в т.ч. при увеличенном

изображении (пункт 8).

1. На интерфейсе может быть предусмотрена дополнительная функциональность на

Ваше усмотрение.

Для того, чтобы выделить базовую структуру алгоритма построения фракталов следует разбить процедуру рисования на небольшие действия, и те из них, которые окажутся одинаковыми для разных фракталов, поместить в базовый (родительский) класс.

Различающиеся действия поместить в производные классы, создав по одному производному классу для каждого фрактала.

Определить в базовом классе Fractal следующие члены:

* поля:
  + длина отрезка (сторона квадрата, треугольника, радиус окружности и т.д.)
  + на первой итерации
  + startColor (см. п. 5)
  + endColor (см. п. 5)
  + текущий уровень рекурсии
  + максимальный уровень рекурсии, заданный пользователем.
* метод Draw (может быть абстрактным или виртуальным, зависит от архитектуры ваших классов)

В зависимости от типа фрактала в классы-наследники следует добавить необходимые члены класса, указанные при описании фрактала. Допустимо добавлять другие члены класса.

Ограничения:

1. В программной реализации не использовать вспомогательные компоненты и сторонние библиотеки, не входящие в стандартную библиотеку.  
2. Не использовать массивы типа object[].

# 2. Функции разрабатываемого приложения

# 2.1. Варианты использования

Программа предназначена для использования на компьютере, содержащем пакеты, необходимые для использования С# и .NET, в образовательных, развлекательных и научных целях требующих построения фракталов Кривая Гильберта, С-Кривая Леви, Множество Кантора.

# 2.2. Описание интерфейса пользователя

Интерфейс представляет из себя оконный интерфейс Windows Forms и содержит:

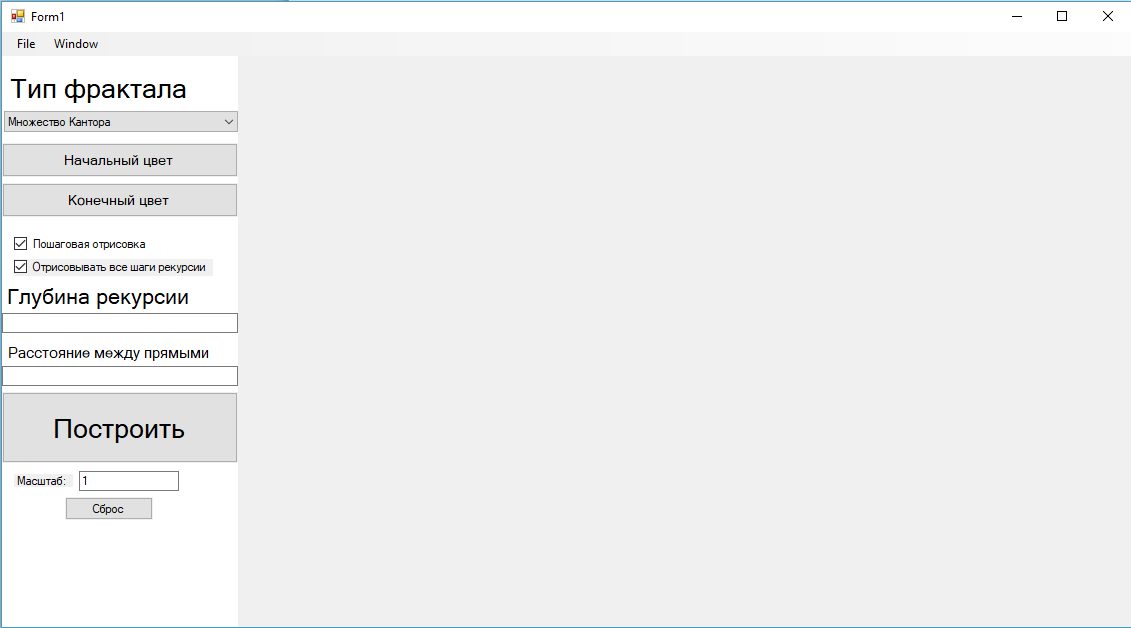


Рисунок 1 - Интерфейс программы

* Выпадающий список фракталов для выбора типа фрактала
* Кнопку для выбора начального цвета градиента
* Кнопку для выбора конечного цвета градиента
* Поле для выбора отрисовывать пошагово, или сразу всё
* Поле для выбора отрисовывать все шаги рекурсии или только видимые
* Поле для ввода глубины рекурсии
* Поле для ввода расстояния между элементами (Множество Кантора)
* Кнопка «Построить»
* Поле для ввода масштаба
* Кнопка «Сброс», которая устанавливает стартовую позицию и увеличение равное 1
* Меню File
  + Save
    - Save As – сохранить как - программа попросит выбрать путь и ввести имя
    - Save – сохранить - программа сохранит под указанным ранее именем, если имя не было у казано ранее – вызовется Save As
  + New - открывается новое окно фрактала, данное окно становится неактивным
* Меню Window
  + Over all windows – показывать окно поверх остальных (если галочка – да)

При отрисовке фрактала на экран выводится окно. Данное окно содержит:

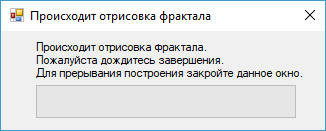


Рисунок 2 - Интерфейс окна отрисовки фрактала

* Текст «Происходит отрисовка фрактала. Пожалуйста дождитесь завершения. Для прерывания построения закройте данное окно.»
* Шкалу, отражающую, насколько завершено построение фрактала (далее - шкала прогресса).

# 3. Структура приложения

Класс:

* Fractal

Поля:

* + handle - используется для определения конца работы потока построения фрактала
  + \_vlock - используется для безопасного изменения значения количества вычисленных элементов фрактала
  + f - первая буква названия фрактала
  + drawall – поле отвечающее за то нужно ли строить все элементы фрактала (если истина) или нужно строить только те элементы, чьи линейные размеры по всем осям больше 0 пикселей (они будут изображены) (если ложь)
  + max\_length - количество элементов фрактала (при наличии в фрактале более 230 элементов max\_length = 230), после начала отрисовки шкалы прогресса принимает значение 10000
  + max\_length2 - количество элементов фрактала (при наличии в фрактале более 230 элементов max\_length = 230),
  + step - шаг заполнения шкалы прогресса, численно равен 1 части фрактала (1 часть фрактала соответствует max\_length/max\_length2, например если max\_length2=100000,(max\_length=10000),то 1 часть фрактала равна 0.1 (step=0.1) )
  + sum - поле отвечающее за то, на сколько заполнена шкала прогресса (при отрисовке каждого элемента фрактала увеличивается на step)
  + isdrawing - поле отвечающее за состояние процесса отрисовки фрактала (истина – происходит отрисовка, ложь - нет)
  + scale - коэффициент увеличения (приближения) фрактала
  + pb - ccылка на форму содержащую шкалу прогресса
  + pbm - максимальное значение шкалы прогресса
  + scf - установлен ли начальный цвет градиента (истина - установлен, ложь – нет)
  + ecf - установлен ли конечный цвет градиента (истина - установлен, ложь – нет)
  + colorarr - массив цветов (возможно вычисление цвета по ходу отрисовки элементов фрактала, но это усложняет реализацию отрисовки фрактала в параллельном потоке и убирает возможность последующего распараллеливания процесса отрисовки фрактала).
  + colorarrmax - длинна массива цветов
  + colorarriter - номер взятого на данной итерации элемента массива colorarr
  + colorarrstep - шаг при переходе на следующую итерацию (может требоваться для оптимизации используемой памяти при отрисовке более чем 256 рекурсий)
  + startColor - начальный цвет градиента
  + endColor - конечный цвет градиента
  + level\_of\_rec - текущий уровень рекурсии
  + max\_level\_of\_rec - максимальный уровень рекурсии
  + space - ширина полей от изображения фрактала, до границы закрашенной области
  + xspace - расстояние от левого края формы, до границы закрашенной области
  + yspace - расстояние от верхнего края формы, до границы закрашенной области
  + size - размеры квадрата ограничивающего фрактал
  + xsize - ширина фрактала
  + ysize - высота фрактала
  + (xleft,yleft) - координата левого верхнего угла области для рисования фрактала
  + pictureBoxXsize - ширина окна приложения
  + pictureBoxYsize - высота окна приложения
  + message - способ передачи информации о различных непредвиденных ситуациях в основное приложение из потока вычислений
  + pen - ручка для рисования линий
  + brush – кисть для заливки фигур

Свойства:

* + Max\_length - свойство для вычисления max\_length и max\_length2

Методы:

* + Конаструкторы - задаёт изначальные значения, затем те, которые поданы в него:
    - startColor = Color.White;
    - endColor = Color.White;
    - level\_of\_rec = 0;
    - max\_level\_of\_rec = 1;
    - size = 300;
    - space = 10;
    - xspace = 218;
    - yspace = 0;
    - xsize = ysize = 300;
  + binpow(x, step) - функция быстрого возведения x в степень step
  + bindrob(max,x,step) - функция точного деления max на x step раз (max/(x\*x\*...\*x))
  + setpictureBoxsize(x,y) - устанавливает pictureBoxXsize и pictureBoxYsize как x и y
  + set\_float(f) - устанавливает значение переменной = f (только множество Кантора)
  + Draw - метод, который запускает отрисовку фрактала
* Point

Поля:

* + x - координата по оси X
  + y - координата по оси Y

Методы:

* + Конструктор - задаёт точку (можно подать как (x,y), так и (Point p)
* Line

Поля:

* + start - точка (Point) начала прямой линии
  + end - точка (Point) конца прямой линии
  + wigth - толщина линии

Методы:

* + Конструктор - задаёт прямую (принимает (xн,yн,xк,yк) или (xн,yн,xк,yк,wigth))
* Cantor

Поля:

* + sizey - толщина элемента множества (прямоугольника)
  + dsizey - расстояние между элементами множества (прямоугольниками)

Методы:

* + Конструктор - задаёт sizey и dsizey
  + set\_float(f) - задаёт dsizey = f
  + count\_len - До какого уровня рекурсии длина элемента множества будет >= 1
  + Draw - инициализация отрисовки фрактала
  + Rec - рекурсивное вычисление и отрисовка элементов фрактала
* Levi

Поля:

Методы:

* + Draw - инициализация отрисовки фрактала
  + Rec - рекурсивное вычисление и отрисовка элементов фрактала
* Gilbert

Поля:

* + sizey - толщина элемента множества (прямоугольника)
  + dsizey - расстояние между элементами множества (прямоугольниками)

Методы:

* + Draw - инициализация отрисовки фрактала
  + Rec - рекурсивное вычисление и отрисовка элементов фрактала
  + Класс P:

Поля:

* + - 5 точек типа Point (p1,p2,p3,p4,p5)
    - w - расстояния между точками
    - direct - направление элемента фрактала

Методы:

* Конструктор - принимает:
  + Координата x начальной точки
  + Координата y начальной точки
  + Расстояние между точками
  + Направление элемента
  + Следующая точка
* Drow - нарисовать линии (p1,p2), (p2,p3), (p3,p4)
  + Класс L:

Поля:

* + - Координаты двух точек (x1,y1) и (x2,y2)
    - dir - направление элемента фрактала

Методы:

* Конструктор – принимает:
  + Координата x начальной точки
  + Координата y начальной точки
  + Координата x конечной точки
  + Координата y конечной точки
  + Направление
  + Класс Lines:

Поля:

* + - Массив линий (элементов класса L)

Методы:

* + - Конструктор без параметров: создаёт пустой массив линий l
* Конструктор: принимает массив line и строит по нему массив l
* ProgressBur

Поля:

* + frac - ссылка на фрактал, для получения информации об отрисовке фрактала
  + isexit - bool переменная, обозначающая нужно ли завершать выполнение
  + timer - таймер для обновления шкалы прогресса отрисовки фрактала
  + progressBar1 - шкала прогресса

Методы:

* + gfrac - Получение ссылки на фрактал
  + ProgressBur - Конструктор устанавливающий ссылку на фрактал
  + init - Инициализация шкалы прогресса
  + Draw - Обновление шкалы прогресса
  + ProgressBurClosed - Переопределение метода закрытия окна шкалы прогресса, для отключения возможности его закрыть
  + OnFormClosing - Отлов события Закрытие окна шкалы прогресса и (прекращение отрисовки или выход из программы (при команде от основного окна))
  + ProgressBur\_KeyDown – нажата ли клавиша Escape (да - вызвать OnFormClosing)
* Form1

Поля:

* + posx, posy - координаты позиции левого верхнего угла выводимого изображения
  + pox, poy - смещение относительно posx, posy соответственно
  + frac - ссылка на текущий фрактал, для отрисовки данного фрактала
  + message - строка для получения информации об отрисовке фрактала
  + bmp - BitMap в котором хранится текущий построенный фрактал
  + draw\_step\_by\_step - отрисовывать: истина - каждый шаг; ложь - только результат
  + pb – ссылка на шкалу прогресса
  + timer - таймер для обновления отрисовки фрактала
  + flagmb – выведено ли уже окно с сообщением об ошибке (да – истина, нет – ложь)

Методы:

* + Frac – Свойство: чтение – вернуть frac; запись – записать в frac, обнулить pox, poy
  + Form1 - Конструктор принимающий ссылку на окно шкалы прогресса
  + comboBox\_fractal\_SelectedIndexChanged – Выбор фрактала
  + button1\_Click – Нажали кнопку - построить – отрисовать фрактал
  + DrawFractal – Построение фрактала
  + OnPaint – Переопределение перерисовки окна
  + pictureBox\_fractal\_MouseDown – Левая Кнопка Мыши нажата
  + pictureBox\_fractal\_MouseUp – Левая Кнопка Мыши отпущена
  + pictureBox\_fractal\_MouseMove – Перемещение курсора (если ЛКМ нажата)
  + button2\_Click – Если кнопка Сброс нажата, то Init()
  + Form1\_KeyDown – Отлов нажатий клавиш
  + SetStart\_Click – Выбор начального цвета фрактала
  + SetEndColor\_Click – Выбор конечного цвета фрактала (можно объединить с предыдущим, но это снизит читаемость кода)
  + ZoomUp(e) – Приближение в точке, где находится курсор
  + ZoomUp() – Приближение в (posx, posy)
  + ZoomDown(e) - Удаление в точке, где находится курсор
  + ZoomDown() - Удаление в (posx, posy)
  + pictureBox\_fractal\_MouseWheel – Изменение масштаба с помощью колеса мыши
  + Rewrite – Перестроение фрактала
  + saveToolStripMenuItem1\_Click – сохранение изображения под «старым» именем
  + saveToolStripMenuItem\_Click – сохранение как изображения (старый интерфейс)
  + saveAsToolStripMenuItem\_Click – сохранение как изображения (новый интерфейс)
  + newToolStripMenuItem\_Click – Открытие нового окна фрактала
  + checkBox1\_CheckedChanged – Отрисовывать все уровни рекурсии?
  + textBox1\_TextChanged - Изменение масштаба через поле для ввода
  + DropExWindow – Вывод сообщения об ошибке (максимум 1 сообщение)
  + ProgressBur – Новое окно прогресса отрисовки фрактала
  + timer\_Tick - Отрисовка формы во время построения фрактала
  + checkBox\_buffer\_CheckedChanged – Отрисовывать постепенно или всё сразу?
  + end\_of\_Draw\_Fractal – Конец отрисовки фрактала
  + overAllWindowsToolStripMenuItem\_Click – Выводить поверх остальных окон?
  + Init() – Стартовые значения позиции и размера и перерасчёт фрактала
  + Init(draw) – Стартовые значения позиции и размера и если draw, то перерасчёт
  + Form1Closed – Закрытие приложения
* MultiFormContext

Поля:

* + openForms – Количество открытых форм

Методы:

* + Конструктор – принимает список форм, доинициализирует и запускает их
* ColorRGB – структура для перевода RGB2HSL и HSL2RGB
* Colorarr – класс, который собирает массив цветов (исходя из начального и конечного цветов)

# 4. Распределение исходного кода по файлам проекта

* Fractal.cs – Базовый класс Fractal
* Cantor.cs – Класс Cantor наследуемый от Fractal
* Gilbert.cs – Класс Gilbert наследуемый от Fractal
* Levi.cs – Класс Levi наследуемый от Fractal
* ProgressBur.cs – Класс ProgressBur (Интерфейс шкалы прогресса)
* Form1.cs – Класс Form1 (Интерфейс основного окна приложения)
* Program.cs – Класс Program (Точка запуска приложения и содержит MultiFormContext)
* Color.cs – Класс преобразования цветов RGB – HSL и получения градиента

# 5. Описание интерфейса

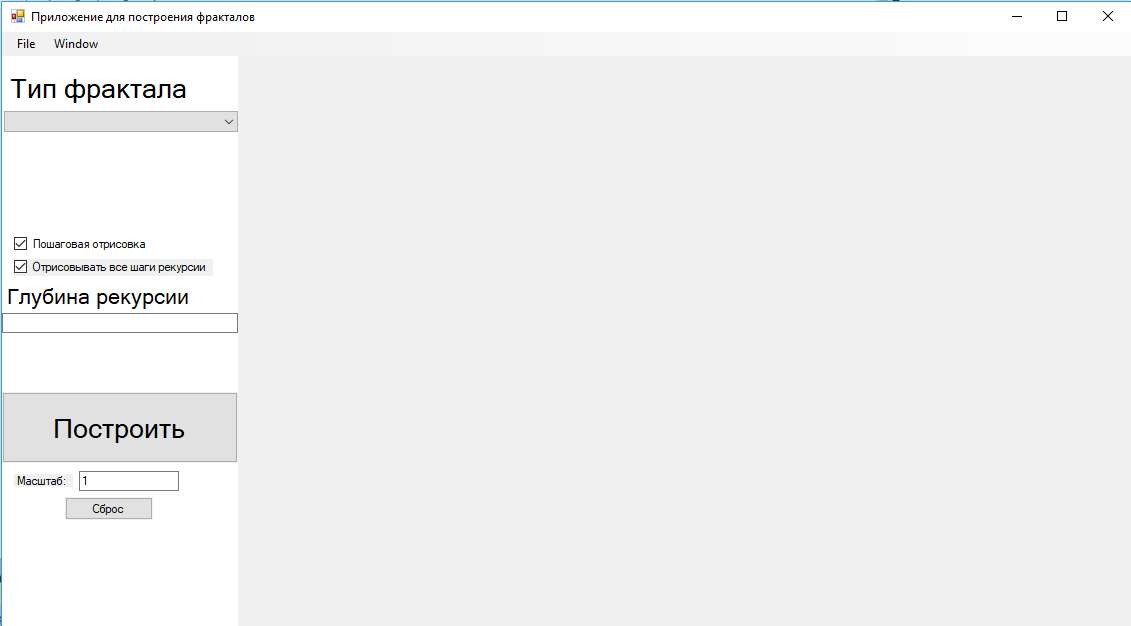
Состоит из 2-х частей:

* Шкала прогресса (показывает, насколько построен фрактал)
  + Закрыть для прерывания построения фрактала
* Форма:
  + Меню:
    - File:
      * Save:
        + Save As – выбрать директорию и имя и сохранить фрактал
        + Save – сохранить изображение по последнему пути, если пути нет, то вызвать Save As
      * New – открыть новое окно фрактала и заблокировать данное
    - Window:
      * Over all windows – Поверх всех окон (есть галочка – да, нет – нет)
  + Тип фрактала (под заголовком) – выпадающий список с типами фракталов
  + Кнопка выбора Начального цвета – открывается диалог выбора цвета
  + Кнопка выбора Конечного цвета – открывается диалог выбора цвета
  + Включить пошаговую отрисовку (галочка – да) (аналог – B)
  + Включить отрисовку всех шагов рекурсии (тех что не видны) (галочка – да)
  + Глубина рекурсии (под заголовком) (поле для ввода глубины рекурсии)
  + Расстояние между прямыми (под заголовком) (поле для ввода расстояний между элементами во множестве Кантора)
  + Построить – нажмите для построения фрактала (аналог – Enter)
  + Масштаб (правее заголовка) – введите масштаб (аналог – колесо мыши или Q и E)
  + Кнопка Сброс – обнулить позицию и увеличение фрактала (аналог – С)

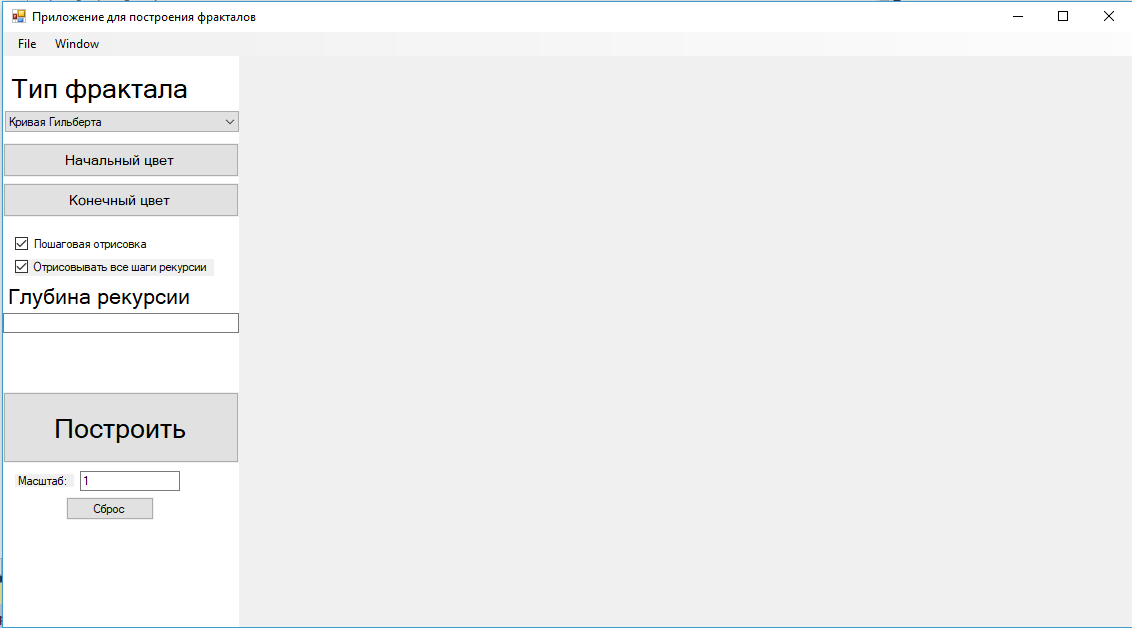
# 

# 6. Контрольный пример и описание результата

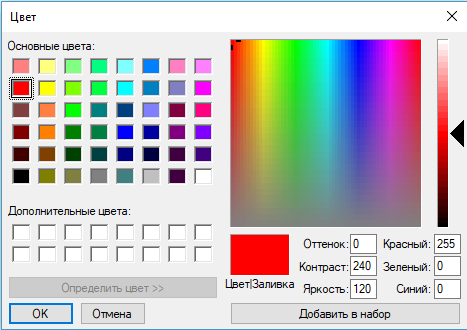
1) Запустить программу

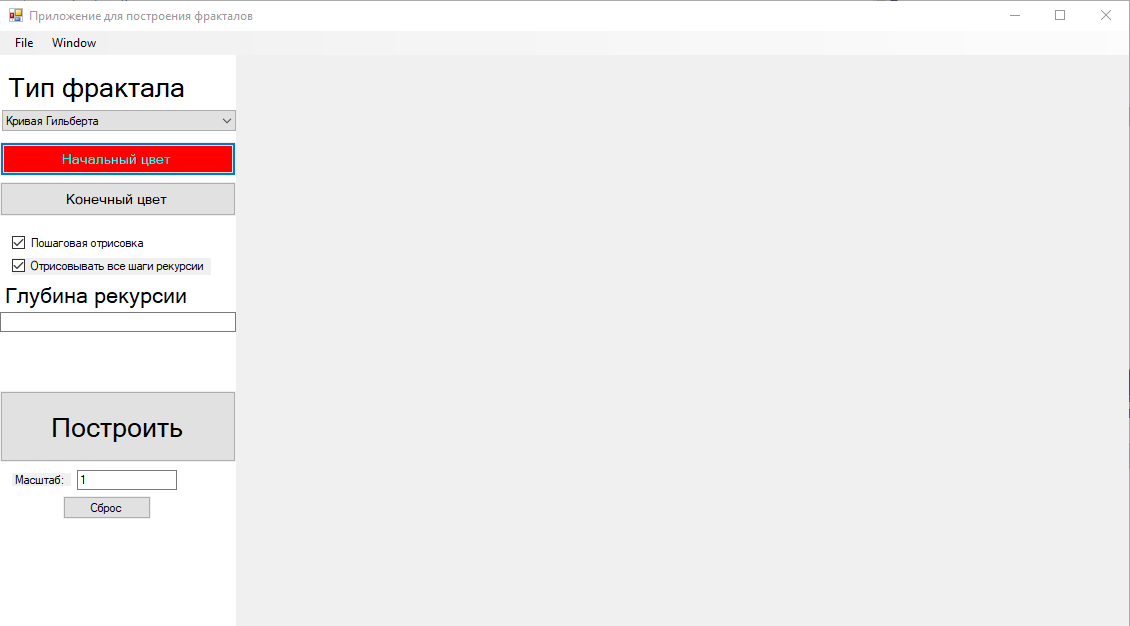


2) Выбрать «Кривая Гильберта

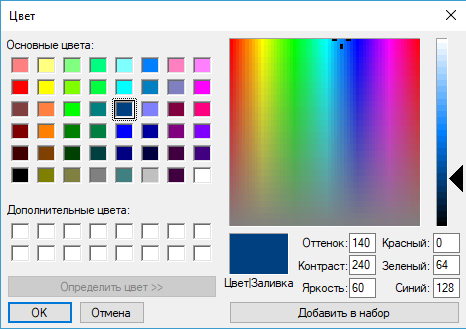


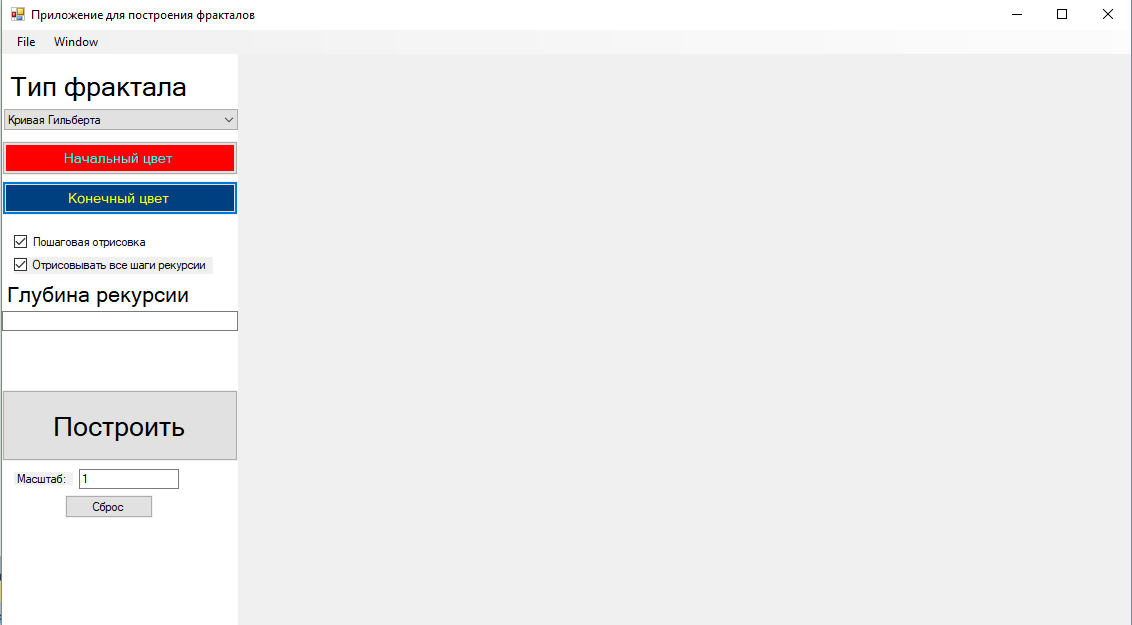
3) Установить начальный цвет



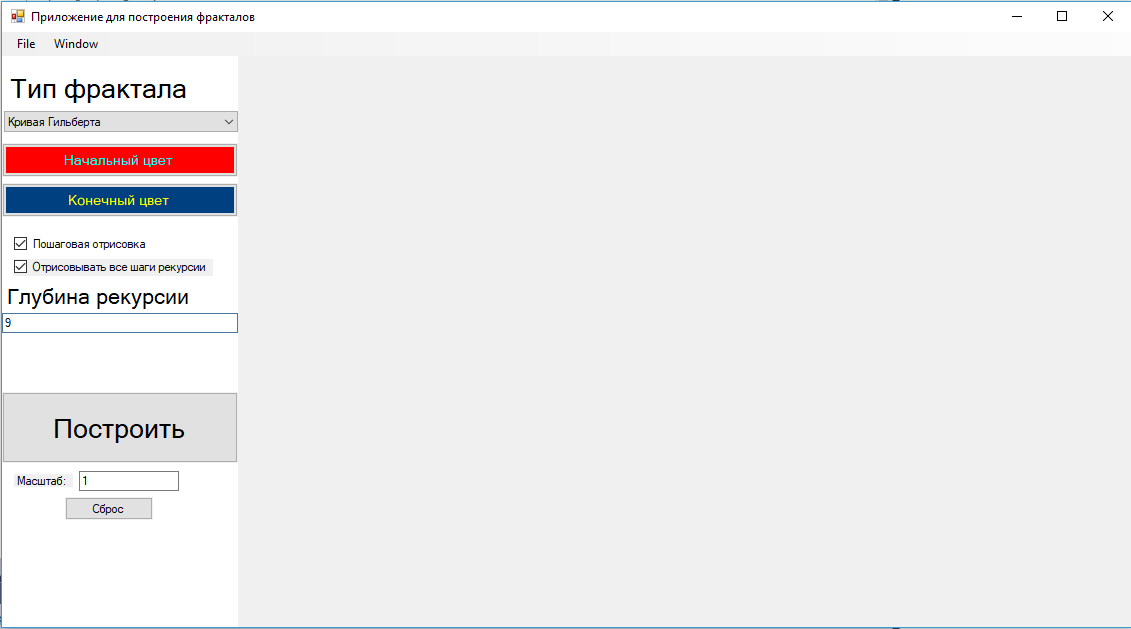


4) Установить конечный цвет

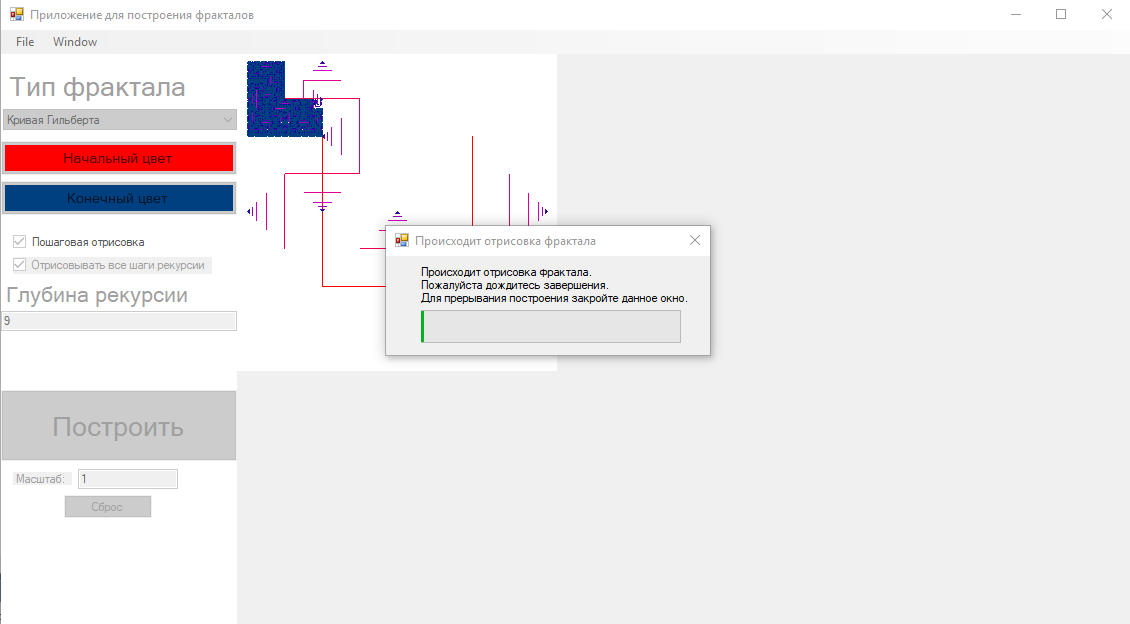




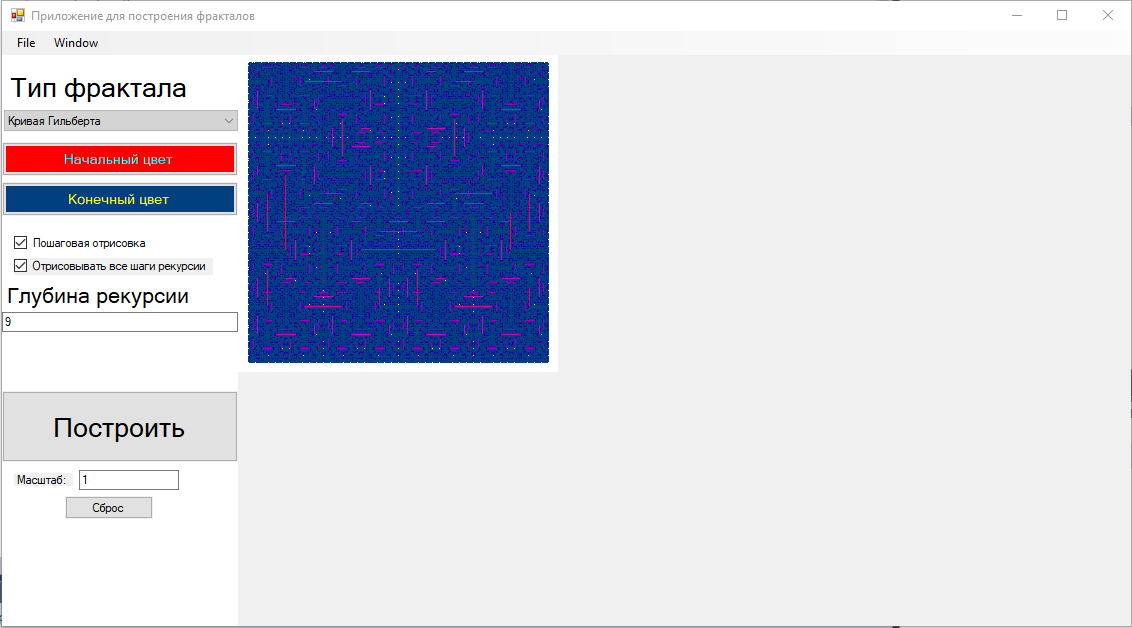
5) Установить глубину рекурсии



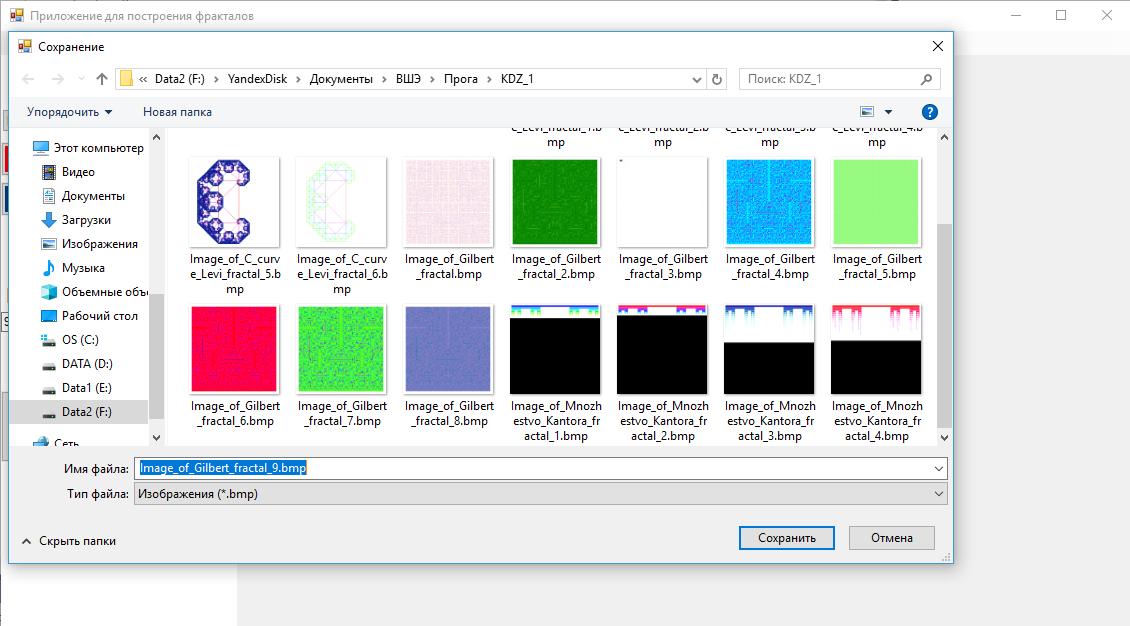
6) Нажать кнопку Построить или Enter (на клавиатуре)



7) Подождать



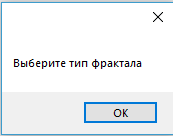
8) Сохранить изображение



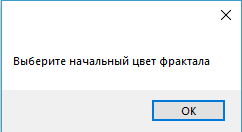
Готово

# 7. Сообщения пользователю

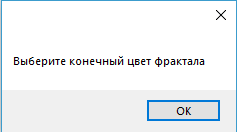
Необходимо выбрать тип фрактала



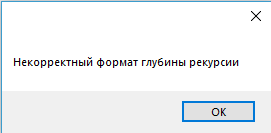
Необходимо выбрать начальный цвет фрактала



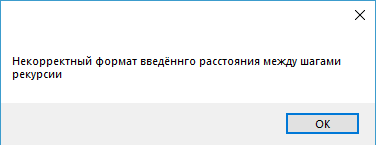
Необходимо выбрать конечный цвет фрактала



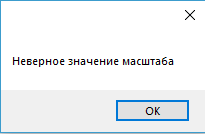
Глубина рекурсии должна быть натуральным числом (>0)

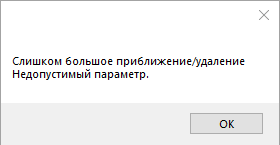


Расстояние между шагами рекурсии (прямыми) должно принадлежать [0,1000]

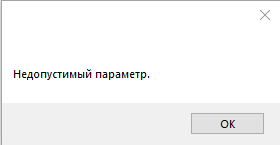


Значение масштаба должно принадлежать (0.05, 51.8]





При появлении окна Недопустимый параметр проверьте все поля ввода, если не помогло, то нажмите кнопку «Сброс» или (С – на клавиатуре)



# 8. Текст (код) программы

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Drawing;

using KDZ\_1;

namespace KDZ\_1

{

//class Color

//{

// public Color() { }

// public System.Drawing.Color color;

//}

class Fractal

{

public static System.Threading.EventWaitHandle handle = new System.Threading.AutoResetEvent(false);

protected static object \_vlock = new object();

public char f = 'f';

public bool drawall = true;

public int max\_length, max\_length2;

public virtual int Max\_length {

get

{

return max\_length;

}

set

{

max\_length = binpow(2, Math.Min(value,(int)30));

max\_length2 = max\_length;

}

}

public float summ = 0, step = 0;

public bool isdrawing = false;

public float scale = 1;

public ProgressBur pb;

public int pbm;

public bool scf = false, ecf = false;

protected Colorarr colorarr;

protected int colorarrmax;

protected float colorarriter = 0, colorarrstep;

//public float length;

public Color startColor;

public Color endColor;

protected int level\_of\_rec;

public int max\_level\_of\_rec;

public float space, xspace, yspace; //Rasstojanie do granitsi okna

public float size, xsize, ysize; //Size of fractal

public float xleft, yleft; //Left Upper point

protected float pictureBoxXsize, pictureBoxYsize; //Size of window

public string message = "";

public System.Drawing.Pen pen;

public System.Drawing.Brush brush;

public Fractal() {

//this.length = 100;

this.startColor = Color.White;

this.endColor = Color.White;

this.level\_of\_rec = 0;

this.max\_level\_of\_rec = 1;

size = 300;

space = 10;

xspace = 218;

yspace = 0;

xsize = ysize = 300;

}

/// <summary>

/// Быстрое возведение в степень

/// </summary>

/// <param name="x">Число возводимое в степень</param>

/// <param name="step">Степень</param>

/// <returns></returns>

public int binpow(int x, int step)

{

if (step == 0) return 1;

if (step % 2 == 0)

{

int a = binpow(x, step / 2);

if (a < 0)

{

return -1;

}

return a \* a;

}

else

{

int a = binpow(x, step - 1);

if (a < 0)

{

return -1;

}

return x \* a;

}

}

/// <summary>

/// Деление числа

/// </summary>

/// <param name="max">Делимое</param>

/// <param name="x">Делитель</param>

/// <param name="step">Количество делений Делимого на делитель ( max/(x\*x\*x\*x) для step = 4)</param>

/// <returns></returns>

public float bindrob(float max, int x, int step)

{

if (step == 1) return (float)(1.0 \* max) / x;

return bindrob((float)(max) / x, x, step - 1);

}

/// <summary>

/// Установить новые размеры окна для фрактала

/// </summary>

/// <param name="x">Новая ширина окна</param>

/// <param name="y">Новая высота окна</param>

public void setpictureBoxsize(float x, float y)

{

this.pictureBoxXsize = x;

this.pictureBoxYsize = y;

}

/// <summary>

/// Функция для возможности установки значения в protected поле

/// </summary>

/// <param name="f"></param>

public virtual void set\_float(float f)

{

}

public Fractal(int mlor) : this()

{

this.max\_level\_of\_rec = mlor;

}

public Fractal(int mlor, float length) : this()

{

//this.length = length;

this.max\_level\_of\_rec = mlor;

}

public Fractal(int mlor, Color sc, Color ec) : this()

{

this.startColor = sc;

this.endColor = ec;

this.max\_level\_of\_rec = mlor;

}

public Fractal(int mlor, float length, Color sc, Color ec) : this()

{

//this.length = length;

this.startColor = sc;

this.endColor = ec;

this.max\_level\_of\_rec = mlor;

}

public Fractal(int mlor, float length, Color sc, Color ec, float xs, float ys) : this()

{

//this.length = length;

this.startColor = sc;

this.endColor = ec;

this.max\_level\_of\_rec = mlor;

this.xsize = xs;

this.ysize = ys;

}

public Fractal(int mlor, float length, Color sc, Color ec, float xs, float ys, float size) : this()

{

//this.length = length;

this.startColor = sc;

this.endColor = ec;

this.max\_level\_of\_rec = mlor;

this.xsize = xs;

this.ysize = ys;

this.size \*= size / this.size;

this.space \*= size / this.size;

}

public virtual void Draw(System.Drawing.Graphics graph)

{

}

}

class Line

{

public Point start, end;

public int wigth;

public Line() { }

public Line(Point start, Point end)

{

this.start = new Point(start);

this.end = new Point(end);

}

public Line(int x\_start, int y\_start, int x\_end, int y\_end)

{

this.start = new Point(x\_start, y\_start);

this.end = new Point(x\_end, y\_end);

}

public Line(int x\_start, int y\_start, int x\_end, int y\_end, int wigth)

{

this.start = new Point(x\_start, y\_start);

this.end = new Point(x\_end, y\_end);

this.wigth = wigth;

}

}

class Point

{

public float x, y;

public Point() { }

public Point(float x, float y)

{

this.x = x;

this.y = y;

}

public Point(Point p)

{

this.x = p.x;

this.y = p.y;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace KDZ\_1

{

class Cantor : Fractal

{

//protected Line[] l;

public float sizey = 10, dsizey=10;

public Cantor() : base(){

xsize = ysize = 600;

}

public Cantor(float dwigth, float wigth=10) : base(){

this.sizey = wigth;

this.dsizey = dwigth;

}

/// <summary>

/// Задать dsizey

/// </summary>

/// <param name="f">Значение передаваемое dsizey</param>

public override void set\_float(float f)

{

this.dsizey = f;

}

/// <summary>

/// До какого уровня рекурсии длина элемента множества Кантора будет >= 1

/// </summary>

/// <param name="mlor">Максимальный уровень рекурсии</param>

/// <param name="s">Начальная длина элемента множества Кантора</param>

/// <returns></returns>

public int count\_len(int mlor, float s)

{

int counter = 0;

while (s / 3 >= 1) {

s /= 3;

counter++;

}

return Math.Min(counter+2, mlor);

}

/// <summary>

/// Инициализация отрисовки фрактала

/// </summary>

/// <param name="graph">Куда отрисовывать</param>

public override void Draw(System.Drawing.Graphics graph)

{

int mlor = count\_len(max\_level\_of\_rec, size \* scale);

if (this.drawall)

{

if (max\_level\_of\_rec > 0)

{

this.colorarrmax = (max\_level\_of\_rec + 1);

this.colorarr = new Colorarr(colorarrmax, startColor, endColor);

this.colorarrstep = bindrob(colorarrmax, 2, max\_level\_of\_rec);

//this.colorarrmax = (mlor < 0 ? int.MaxValue : mlor);

//this.colorarr = new Colorarr(colorarrmax);

//this.colorarrstep = colorarrmax/mlor;

}

//graph.FillRectangle(System.Drawing.Brushes.White, 0, 0, (xsize + space \* 2) \* scale, (sizey \* max\_level\_of\_rec + dsizey \* max\_level\_of\_rec + space \* 2) \* scale);

}

else

{

if (mlor > 0)

{

this.colorarrmax = (mlor + 1);

this.colorarr = new Colorarr(colorarrmax, startColor, endColor);

this.colorarrstep = bindrob(colorarrmax, 2, mlor);

//this.colorarrmax = (mlor < 0 ? int.MaxValue : mlor);

//this.colorarr = new Colorarr(colorarrmax);

//this.colorarrstep = colorarrmax/mlor;

}

//graph.FillRectangle(System.Drawing.Brushes.White, 0, System.Math.Max(((pictureBoxYsize - (sizey \* scale \* mlor + dsizey \* scale \* Math.Max(mlor - 1, 0) + space \* scale \* 2) + yspace \* scale) / 2) - yleft, 0), (xsize + space \* 2) \* scale, (sizey \* mlor + dsizey \* Math.Max(mlor - 1, 0) + space \* 2) \* scale);

}

graph.FillRectangle(System.Drawing.Brushes.White, 0, 0, (xsize + space \* 2) \* scale, (sizey + dsizey + space \* 2) \* scale);

//message = "" + pictureBoxXsize + " " + pictureBoxYsize + " " + (pictureBoxXsize - xsize + xspace) / 2 + " " + (pictureBoxYsize - ysize + yspace) / 2 + " " + (xsize + space \* 2) + " " + (ysize + space \* 2);

//return;

//message = "" + mlor + " " + max\_level\_of\_rec;

//this.colorarr = new Colorarr(mlor);

try

{

rec(graph, 0 + space \* scale, (System.Math.Max(((pictureBoxYsize - (sizey \* scale \* mlor + dsizey \* scale \* Math.Max(mlor - 1, 0)) + yspace \* scale) / 2) - 0, 0) + space) \* scale, xsize \* scale, 0);

}

catch (StackOverflowException)

{

message = "Слишком большая глубина рекусии (установите количество итераций для построения фрактала на меньшее значение";

}

}

/// <summary>

/// Рекурсивное вычисление и отрисовка фрактала

/// </summary>

/// <param name="g">Куда отрисовывать</param>

/// <param name="x">Координата x верхнего левого угла</param>

/// <param name="y">Координата y верхнего левого угла</param>

/// <param name="w">Длинна элемента</param>

/// <param name="lor">Текущий уровень рекурсии</param>

void rec(System.Drawing.Graphics g, float x, float y, float w, float lor)

{

if (!isdrawing) return;

this.summ += this.step;

try

{

lock (\_vlock) { this.pb.progressBar1.Value = Math.Min(this.pbm - 3, (int)(this.summ)); }

}

catch (System.InvalidOperationException ex)

{

}

//this.pb.progressBar1.Value = Math.Min(pb.progressBar1.Maximum - 3, (int)(this.summ));

level\_of\_rec = (int)lor;

if (lor == max\_level\_of\_rec)

{

return;

}

if (w < 1 && !drawall && lor != 0)

{

return;

}

g.FillRectangle(System.Drawing.Brushes.White, 0, y, space \* scale, (sizey + dsizey \* Math.Min(lor, 1) + space) \* scale);

g.FillRectangle(System.Drawing.Brushes.White, x, y, (xsize + space \* 2) \* scale - x, (sizey + dsizey + space) \* scale);

//Start another recursions

rec(g, x, y + sizey \* scale + dsizey \* scale, w / 3, lor+1);

rec(g, x + 2 \* w / 3, y + sizey \* scale + dsizey \* scale, w / 3, lor+1);

//Print figure

if (max\_level\_of\_rec > 0)

{

this.brush = new System.Drawing.SolidBrush(this.colorarr.colorarr[(int)(lor)]);

}

g.FillRectangle(this.brush, x, y, w, this.sizey \* scale);

if (lor == 0)

{

this.isdrawing = false;

Fractal.handle.Set();

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace KDZ\_1

{

class Gilbert : Fractal

{

public override int Max\_length

{

get

{

return max\_length;

}

set

{

max\_length = binpow(4, Math.Min(value, (int)16)-1);

max\_length2 = max\_length;

}

}

//public char f = 'g';

//protected Line[] l;

/// <summary>

/// Инициализация отрисовки фрактала

/// </summary>

/// <param name="graph">Куда отрисовывать</param>

public override void Draw(System.Drawing.Graphics graph)

{

graph.FillRectangle(System.Drawing.Brushes.White, 0, 0, (xsize + space \* 2) \* scale, (ysize + space \* 2) \* scale);

if (max\_level\_of\_rec > 0)

{

this.colorarrmax = (max\_level\_of\_rec + 1);

this.colorarr = new Colorarr(colorarrmax, startColor, endColor);

this.colorarrstep = bindrob(colorarrmax, 2, max\_level\_of\_rec);

}

try

{

P p = new P((0 + this.xsize / 4 + space) \* scale, (0 + this.xsize / 4 + space) \* scale, (this.xsize / 2) \* scale, "ulr", new Point((0 + this.xsize / 4 + this.xsize / 2 + space) \* scale, (0 + this.xsize / 4 + space) \* scale));

//this.length = 1;

Lines l = new Lines();

rec(graph, p, 0, l);

}

catch (StackOverflowException)

{

message = "Слишком большая глубина рекусии (установите количество итераций для построения фрактала на меньшее значение";

}

}

class P

{

public Point p1, p2, p3, p4, p5;

public float w=0;

public string direct="nnn";

/// <summary>

/// Конструктор без параметров

/// </summary>

public P() { }

/// <summary>

/// Конструктор

/// </summary>

/// <param name="x">Координата x начальной точки</param>

/// <param name="y">Координата y начальной точки</param>

/// <param name="w">Расстояние между точками</param>

/// <param name="direct">Направление элемента</param>

/// <param name="o">Следующая точка</param>

public P(float x, float y, float w, string direct, Point o)

{

this.p5 = o;

this.w = w;

this.direct = direct;

p1 = new Point(x, y);

switch (direct)

{

case "ulr":

p2 = new Point(x, y + w);

p3 = new Point(x + w, y + w);

p4 = new Point(x + w, y);

break;

case "url":

p2 = new Point(x, y + w);

p3 = new Point(x - w, y + w);

p4 = new Point(x - w, y);

break;

case "lud":

p2 = new Point(x + w, y);

p3 = new Point(x + w, y + w);

p4 = new Point(x, y + w);

break;

case "ldu":

p2 = new Point(x + w, y);

p3 = new Point(x + w, y - w);

p4 = new Point(x, y - w);

break;

case "rud":

p2 = new Point(x - w, y);

p3 = new Point(x - w, y + w);

p4 = new Point(x, y + w);

break;

case "rdu":

p2 = new Point(x - w, y);

p3 = new Point(x - w, y - w);

p4 = new Point(x, y - w);

break;

case "dlr":

p2 = new Point(x, y - w);

p3 = new Point(x + w, y - w);

p4 = new Point(x + w, y);

break;

case "drl":

p2 = new Point(x, y - w);

p3 = new Point(x - w, y - w);

p4 = new Point(x - w, y);

break;

}

}

/// <summary>

/// Нарисовать линии (p1,p2), (p2,p3), (p3,p4)

/// </summary>

/// <param name="g">Куда рисовать</param>

/// <param name="pen">Чем рисовать</param>

public void Drow(System.Drawing.Graphics g, System.Drawing.Pen pen)

{

g.DrawLine(pen, p1.x, p1.y, p2.x, p2.y);

g.DrawLine(pen, p2.x, p2.y, p3.x, p3.y);

g.DrawLine(pen, p3.x, p3.y, p4.x, p4.y);

//g.DrawLine(pen, p4.x, p4.y, p5.x, p5.y);

}

}

class Line {

public float x1, y1, x2, y2;

public char dir = ' ';

/// <summary>

/// Конструктор без параметров

/// </summary>

public Line() { }

/// <summary>

/// Конструктор

/// </summary>

/// <param name="x1">Координата x начальной точки</param>

/// <param name="y1">Координата y начальной точки</param>

/// <param name="x2">Координата x конечной точки</param>

/// <param name="y2">Координата y конечной точки</param>

/// <param name="dir"></param>

public Line(float x1, float y1, float x2, float y2, char dir)

{

this.x1 = x1;

this.y1 = y1;

this.x2 = x2;

this.y2 = y2;

this.dir = dir;

}

}

class Lines

{

public List<Line> l;

/// <summary>

/// Конструктор без параметров: создаёт пустой массив линий l

/// </summary>

public Lines()

{

l = new List<Line>();

}

/// <summary>

/// Конструктор: принимает массив line и строит по нему массив l

/// </summary>

/// <param name="line">Массив линий - соединителей</param>

public Lines(List<Line> line)

{

//l = line;

l = new List<Line>();

foreach (Line li in line)

{

switch (li.dir)

{

case 'u':

float wu = Math.Abs(li.x2 - li.x1);

l.Add(new Line(li.x1 + wu / 4, li.y1 - wu / 4, li.x2 - wu / 4, li.y2 - wu / 4, 'u'));

break;

case 'l':

float wl = Math.Abs(li.y2 - li.y1);

l.Add(new Line(li.x1 - wl / 4, li.y1 + wl / 4, li.x2 - wl / 4, li.y2 - wl / 4, 'l'));

break;

case 'd':

float wd = Math.Abs(li.x2 - li.x1);

l.Add(new Line(li.x1 - wd / 4, li.y1 + wd / 4, li.x2 + wd / 4, li.y2 + wd / 4, 'd'));

break;

case 'r':

float wr = Math.Abs(li.y2 - li.y1);

l.Add(new Line(li.x1 + wr / 4, li.y1 - wr / 4, li.x2 + wr / 4, li.y2 + wr / 4, 'r'));

break;

}

}

}

}

/// <summary>

/// Рекурсивное вычисление и отрисовка фрактала

/// </summary>

/// <param name="g">Куда отрисовывать</param>

/// <param name="p">Точки для отрисовки следующего шага</param>

/// <param name="lor">Текущий уровень рекурсии</param>

/// <param name="l">Массив линий</param>

void rec(System.Drawing.Graphics g, P p, float lor, Lines l)

{

if (!isdrawing) return;

if (lor == max\_level\_of\_rec - 1)

{

this.summ += this.step;

try

{

lock (\_vlock) { this.pb.progressBar1.Value = Math.Min(this.pbm - 3, (int)(this.summ)); }

}

catch (System.InvalidOperationException ex)

{

}

}

//this.pb.progressBar1.Value = Math.Min(pb.progressBar1.Maximum-3, (int)(this.summ));

//System.Threading.Interlocked.Exchange(ref pb.progressBar1.Value, 1);

level\_of\_rec = (int)lor;

//this.length = binpow(2,level\_of\_rec) + 1;

if (lor == max\_level\_of\_rec)

{

//message = "" + xs + " " + ys + " " + xe + " " + ye;

return;

}

if (max\_level\_of\_rec > 0)

{

this.pen.Color = this.colorarr.colorarr[(int)(lor)];

//colorarriter += colorarrstep;

//g.DrawLine(this.pen, xs, ys, xe, ye);

}

//Print figure

p.Drow(g,pen);

if (l.l.Count>0)

{

foreach (Line line in l.l)

{

g.DrawLine(pen, line.x1, line.y1, line.x2, line.y2);

}

}

if (p.w < 2 && !drawall && lor!=0)

{

return;

}

//Start another recursions

switch (p.direct[0]){

case 'u':

Lines liu = l;

l = new Lines(liu.l);

l.l.Add(new Line(p.p1.x - p.w / 4, p.p1.y + p.w / 4, p.p2.x - p.w / 4, p.p2.y - p.w / 4, 'l'));

l.l.Add(new Line(p.p2.x + p.w / 4, p.p2.y - p.w / 4, p.p3.x - p.w / 4, p.p3.y - p.w / 4, 'u'));

l.l.Add(new Line(p.p3.x + p.w / 4, p.p3.y - p.w / 4, p.p4.x + p.w / 4, p.p4.y + p.w / 4, 'r'));

rec(g, new P(p.p1.x - p.w / 4, p.p1.y - p.w / 4, p.w / 2, "lud", new Point(p.p2.x - p.w / 4, p.p2.y - p.w / 4)), lor + 1, l);

rec(g, new P(p.p2.x - p.w / 4, p.p2.y - p.w / 4, p.w / 2, "ulr", new Point(p.p3.x - p.w / 4, p.p3.y - p.w / 4)), lor + 1, l);

rec(g, new P(p.p3.x - p.w / 4, p.p3.y - p.w / 4, p.w / 2, "ulr", new Point(p.p4.x + p.w / 4, p.p4.y + p.w / 4)), lor + 1, l);

rec(g, new P(p.p4.x + p.w / 4, p.p4.y + p.w / 4, p.w / 2, "rdu", new Point(p.p5.x + p.w / 4, p.p5.y - p.w / 4)), lor + 1, l);

break;

case 'l':

Lines lil = l;

l = new Lines(lil.l);

l.l.Add(new Line(p.p1.x + p.w / 4, p.p1.y - p.w / 4, p.p2.x - p.w / 4, p.p2.y - p.w / 4, 'u'));

l.l.Add(new Line(p.p2.x - p.w / 4, p.p2.y + p.w / 4, p.p3.x - p.w / 4, p.p3.y - p.w / 4, 'l'));

l.l.Add(new Line(p.p3.x - p.w / 4, p.p3.y + p.w / 4, p.p4.x + p.w / 4, p.p4.y + p.w / 4, 'd'));

rec(g, new P(p.p1.x - p.w / 4, p.p1.y - p.w / 4, p.w / 2, "ulr", new Point(p.p2.x - p.w / 4, p.p2.y - p.w / 4)), lor + 1, l);

rec(g, new P(p.p2.x - p.w / 4, p.p2.y - p.w / 4, p.w / 2, "lud", new Point(p.p3.x - p.w / 4, p.p3.y - p.w / 4)), lor + 1, l);

rec(g, new P(p.p3.x - p.w / 4, p.p3.y - p.w / 4, p.w / 2, "lud", new Point(p.p4.x + p.w / 4, p.p4.y + p.w / 4)), lor + 1, l);

rec(g, new P(p.p4.x + p.w / 4, p.p4.y + p.w / 4, p.w / 2, "drl", new Point(p.p5.x - p.w / 4, p.p5.y - p.w / 4)), lor + 1, l);

break;

case 'r':

Lines lir = l;

l = new Lines(lir.l);

l.l.Add(new Line(p.p1.x - p.w / 4, p.p1.y + p.w / 4, p.p2.x + p.w / 4, p.p2.y + p.w / 4, 'd'));

l.l.Add(new Line(p.p2.x + p.w / 4, p.p2.y - p.w / 4, p.p3.x + p.w / 4, p.p3.y + p.w / 4, 'r'));

l.l.Add(new Line(p.p3.x + p.w / 4, p.p3.y - p.w / 4, p.p4.x - p.w / 4, p.p4.y - p.w / 4, 'u'));

rec(g, new P(p.p1.x + p.w / 4, p.p1.y + p.w / 4, p.w / 2, "drl", new Point(p.p2.x + p.w / 4, p.p2.y + p.w / 4)), lor + 1, l);

rec(g, new P(p.p2.x + p.w / 4, p.p2.y + p.w / 4, p.w / 2, "rdu", new Point(p.p3.x + p.w / 4, p.p3.y + p.w / 4)), lor + 1, l);

rec(g, new P(p.p3.x + p.w / 4, p.p3.y + p.w / 4, p.w / 2, "rdu", new Point(p.p4.x - p.w / 4, p.p4.y - p.w / 4)), lor + 1, l);

rec(g, new P(p.p4.x - p.w / 4, p.p4.y - p.w / 4, p.w / 2, "ulr", new Point(p.p5.x + p.w / 4, p.p5.y + p.w / 4)), lor + 1, l);

break;

case 'd':

Lines lid = l;

l = new Lines(lid.l);

l.l.Add(new Line(p.p1.x + p.w / 4, p.p1.y - p.w / 4, p.p2.x + p.w / 4, p.p2.y + p.w / 4, 'r'));

l.l.Add(new Line(p.p2.x - p.w / 4, p.p2.y + p.w / 4, p.p3.x + p.w / 4, p.p3.y + p.w / 4, 'd'));

l.l.Add(new Line(p.p3.x - p.w / 4, p.p3.y + p.w / 4, p.p4.x - p.w / 4, p.p4.y - p.w / 4, 'l'));

rec(g, new P(p.p1.x + p.w / 4, p.p1.y + p.w / 4, p.w / 2, "rdu", new Point(p.p2.x + p.w / 4, p.p2.y + p.w / 4)), lor + 1, l);

rec(g, new P(p.p2.x + p.w / 4, p.p2.y + p.w / 4, p.w / 2, "drl", new Point(p.p3.x + p.w / 4, p.p3.y + p.w / 4)), lor + 1, l);

rec(g, new P(p.p3.x + p.w / 4, p.p3.y + p.w / 4, p.w / 2, "drl", new Point(p.p4.x - p.w / 4, p.p4.y - p.w / 4)), lor + 1, l);

rec(g, new P(p.p4.x - p.w / 4, p.p4.y - p.w / 4, p.w / 2, "lud", new Point(p.p5.x - p.w / 4, p.p5.y - p.w / 4)), lor + 1, l);

break;

}

if (lor == 0)

{

this.isdrawing = false;

Fractal.handle.Set();

}

}

}

public struct ColorRGB

{

public byte R;

public byte G;

public byte B;

public ColorRGB(Color value)

{

this.R = value.R;

this.G = value.G;

this.B = value.B;

}

public static implicit operator Color(ColorRGB rgb)

{

Color c = Color.FromArgb(rgb.R, rgb.G, rgb.B);

return c;

}

public static explicit operator ColorRGB(Color c)

{

return new ColorRGB(c);

}

public static ColorRGB HSL2RGB(double h, double sl, double l)

{

double v;

double r, g, b;

h = Math.Min(h, 1);

sl = Math.Min(sl, 1);

l = Math.Min(l, 1);

r = l; // default to gray

g = l;

b = l;

v = (l <= 0.5) ? (l \* (1.0 + sl)) : (l + sl - l \* sl);

if (v > 0)

{

double m;

double sv;

int sextant;

double fract, vsf, mid1, mid2;

m = l + l - v;

sv = (v - m) / v;

h \*= 6.0;

sextant = (int)h;

fract = h - sextant;

vsf = v \* sv \* fract;

mid1 = m + vsf;

mid2 = v - vsf;

switch (sextant)

{

case 0:

r = v;

g = mid1;

b = m;

break;

case 1:

r = mid2;

g = v;

b = m;

break;

case 2:

r = m;

g = v;

b = mid1;

break;

case 3:

r = m;

g = mid2;

b = v;

break;

case 4:

r = mid1;

g = m;

b = v;

break;

case 5:

r = v;

g = m;

b = mid2;

break;

}

}

ColorRGB rgb;

rgb.R = Convert.ToByte(Math.Max(0, r) \* 255.0f);

rgb.G = Convert.ToByte(Math.Max(0, g) \* 255.0f);

rgb.B = Convert.ToByte(Math.Max(0, b) \* 255.0f);

return rgb;

}

public static void RGB2HSL(ColorRGB rgb, out double h, out double s, out double l)

{

double r = rgb.R / 255.0;

double g = rgb.G / 255.0;

double b = rgb.B / 255.0;

double v;

double m;

double vm;

double r2, g2, b2;

h = 0; // default to black

s = 0;

l = 0;

v = Math.Max(r, g);

v = Math.Max(v, b);

m = Math.Min(r, g);

m = Math.Min(m, b);

l = (m + v) / 2.0;

if (l <= 0.0)

{

return;

}

vm = v - m;

s = vm;

if (s > 0.0)

{

s /= (l <= 0.5) ? (v + m) : (2.0 - v - m);

}

else

{

return;

}

r2 = (v - r) / vm;

g2 = (v - g) / vm;

b2 = (v - b) / vm;

if (r == v)

{

h = (g == m ? 5.0 + b2 : 1.0 - g2);

}

else if (g == v)

{

h = (b == m ? 1.0 + r2 : 3.0 - b2);

}

else

{

h = (r == m ? 3.0 + g2 : 5.0 - r2);

}

h /= 6.0;

}

}

class Colorarr

{

//public Color() { }

public Color[] colorarr;

public Colorarr(int len, Color start, Color end)

{

double hs, he, ss, se, ls, le;

//len -= 2;

ColorRGB.RGB2HSL((ColorRGB)(start), out hs, out ss, out ls);

ColorRGB.RGB2HSL((ColorRGB)(end), out he, out se, out le);

double sh = (se - ss) / (Math.Max(len - 2, 1));

double lh = (le - ls) / (Math.Max(len - 2, 1));

double hh = (he - hs) / (Math.Max(len - 2, 1));

bool rev = false;

//if (hh < 0)

//{

// hh = (hh)\*(0-1.0);

// double h = hs;

// hs = he;

// he = h;

// sh = (sh) \* (0 - 1.0);

// h = ss;

// ss = se;

// se = h;

// lh = (lh) \* (0 - 1.0);

// h = ls;

// ls = le;

// le = h;

// rev = true;

//}

//colorarr = new Color[] { Color.Red, Color.Green, Color.Blue };

this.colorarr = new Color[0];

int i = 0;

for (double hi = hs, li = ls, si = ss; i < len; hi += hh, li += lh, si += sh )

{

i++;

ColorRGB c = ColorRGB.HSL2RGB(hi, si, li);

Color[] ncolorarr = this.colorarr;

this.colorarr = new Color[ncolorarr.GetLength(0) + 1];

for (int j = 0; j < ncolorarr.GetLength(0); j++)

{

this.colorarr[j] = ncolorarr[j];

}

this.colorarr[ncolorarr.GetLength(0)] = (Color)(c);

//do something with the color

}

//if (rev) Array.Reverse(colorarr);

this.colorarr[Math.Max(colorarr.GetLength(0)-2,0)] = end;

colorarr[0] = start;

//Color[] ncolorarr2 = this.colorarr;

//this.colorarr = new Color[ncolorarr2.GetLength(0) + 1];

//for (int j = 0; j < ncolorarr2.GetLength(0); j++)

//{

// this.colorarr[j] = ncolorarr2[j];

//}

int vivod = 0;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace KDZ\_1

{

class Levi : Fractal

{

//protected Line[] l;

/// <summary>

/// Инициализация отрисовки фрактала

/// </summary>

/// <param name="graph">Куда отрисовывать</param>

public override void Draw(System.Drawing.Graphics graph)

{

graph.FillRectangle(System.Drawing.Brushes.White, 0, 0, (xsize \* 8 / 8 + space \* 2)\*scale, (ysize + space \* 2)\*scale);

//message = "" + pictureBoxXsize + " " + pictureBoxYsize + " " + (pictureBoxXsize - xsize + xspace) / 2 + " " + (pictureBoxYsize - ysize + yspace) / 2 + " " + (xsize + space \* 2) + " " + (ysize + space \* 2);

//return;

if (max\_level\_of\_rec>0)

{

//this.colorarrmax = (binpow(2, max\_level\_of\_rec) < 0 ? int.MaxValue : binpow(2, max\_level\_of\_rec));

this.colorarrmax = (max\_level\_of\_rec+1);

this.colorarr = new Colorarr(colorarrmax, startColor, endColor);

this.colorarrstep = bindrob(colorarrmax, 2, max\_level\_of\_rec);

}

try

{

rec(graph, (0 + space + xsize \* 9 / 16)\*scale, (0 + space + ysize \* 3 / 4)\*scale, (0 + space + xsize \* 9 / 16)\*scale, (0 + space + ysize / 4)\*scale, 0);

}

catch (StackOverflowException)

{

message = "Слишком большая глубина рекусии (установите количество итераций для построения фрактала на меньшее значение";

}

}

/// <summary>

/// Рекурсивное вычисление и отрисовка фрактала

/// </summary>

/// <param name="g">Куда отрисовывать</param>

/// <param name="xs">Координата x начальной точки</param>

/// <param name="ys">Координата y начальной точки</param>

/// <param name="xe">Координата x конечной точки</param>

/// <param name="ye">Координата y конечной точки</param>

/// <param name="lor">Текущий уровень рекурсии</param>

void rec(System.Drawing.Graphics g, float xs, float ys, float xe, float ye, float lor)

{

if (!isdrawing) return;

this.summ += this.step;

try

{

lock (\_vlock) { this.pb.progressBar1.Value = Math.Min(this.pbm - 3, (int)(this.summ)); }

}

catch (System.InvalidOperationException ex)

{

}

//this.pb.progressBar1.Value = Math.Min(pb.progressBar1.Maximum - 3, (int)(this.summ));

level\_of\_rec = (int)lor;

if (lor == max\_level\_of\_rec)

{

//message = "" + xs + " " + ys + " " + xe + " " + ye;

return;

}

if (Math.Abs((xe - xs)) < 1 && Math.Abs((ye - ys)) < 1 && !drawall && lor != 0)

{

return;

}

//Start another recursions

float dx = Math.Abs(xe - xs) / 4, dy = Math.Abs(ye - ys) / 4;

if (lor % 2 == 0)

{

if (Math.Abs(xs-xe)<0.1)

{

if (ys < ye)

{

rec(g, xs, ys, xs + 2 \* dy, ys + 2 \* dy, lor + 1);

rec(g, xs + 2 \* dy, ys + 2 \* dy, xe, ye, lor + 1);

}

else

{

rec(g, xs, ys, xs - 2 \* dy, ys - 2 \* dy, lor + 1);

rec(g, xs - 2 \* dy, ys - 2 \* dy, xe, ye, lor + 1);

}

}

else

{

if (xs < xe)

{

rec(g, xs, ys, xs + dx \* 2, ys - dx \* 2, lor + 1);

rec(g, xs + 2 \* dx, ys - 2 \* dx, xe, ye, lor + 1);

}

else

{

rec(g, xs, ys, xs - dx \* 2, ys + dx \* 2, lor + 1);

rec(g, xs - 2 \* dx, ys + 2 \* dx, xe, ye, lor + 1);

}

}

}

else

{

float xl = Math.Min(xs, xe), xr = Math.Max(xs, xe), yu = Math.Min(ys, ye), yd = Math.Max(ys, ye);

//if (ye < ys)

//{

// rec(g, xl, yu, xr, yu, lor + 1);

// rec(g, xl, yu, xl, yd, lor + 1);

//}

//else

//{

// rec(g, xl, yd, xr, yd, lor + 1);

// rec(g, xl, yu, xl, yd, lor + 1);

//}

//rec(g, xs, ye, xe, ye, lor + 1);

//rec(g, xs, yu, xs, yd, lor + 1);

if(xs < xe && ys > ye)

{

rec(g, xs, ys, xs, ye, lor + 1);

rec(g, xs, ye, xe, ye, lor + 1);

}

if (xs < xe && ys < ye)

{

rec(g, xs, ys, xe, ys, lor + 1);

rec(g, xe, ys, xe, ye, lor + 1);

}

if (xs > xe && ys < ye)

{

rec(g, xs, ys, xs, ye, lor + 1);

rec(g, xs, ye, xe, ye, lor + 1);

}

if (xs > xe && ys > ye)

{

rec(g, xs, ys, xe, ys, lor + 1);

rec(g, xe, ys, xe, ye, lor + 1);

}

}

//Print figure

if (max\_level\_of\_rec > 0)

{

this.pen.Color = this.colorarr.colorarr[(int)(lor)];

//colorarriter += colorarrstep;

}

g.DrawLine(this.pen, xs, ys, xe, ye);

if (lor == 0)

{

this.isdrawing = false;

Fractal.handle.Set();

}

}

}

}

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 System.Windows.Forms;

namespace KDZ\_1

{

public partial class Form1 : Form

{

private float posx, posy, pox, poy;

Fractal frac;

private string name = "";

Fractal Frac{

get{

return frac;

}

set

{

pox = poy = 0;

frac = value;

}

}

//private string Cantor\_dspace = "";

private string message = "";

private Bitmap bmp = null;

private bool draw\_step\_by\_step = true;

ProgressBur pb;

Timer timer = new Timer();

bool fenableformwhendrawing = false;

bool flagmb = false;

/// <summary>

/// Конструктор принимающий ссылку на окно шкалы прогресса

/// </summary>

/// <param name="pb">Ссылка на окно шкалы прогресса</param>

internal Form1(ProgressBur pb)

{

this.pb = pb;

this.pb.Visible = false;

this.pb.Enabled = false;

DoubleBuffered = true;

InitializeComponent();

Invalidate();

Init();

SetStartColor.Visible = false;

SetEndColor.Visible = false;

this.colorDialog1.FullOpen = true;

this.colorDialog1.Color = Color.White;

this.Closed += Form1Closed;

this.TopMost = overAllWindowsToolStripMenuItem.Checked;//true;

//Draw();

timer.Interval = 10; //интервал между срабатываниями 10 миллисекунд

timer.Tick += new EventHandler(timer\_Tick);

}

/// <summary>

/// Выбор фрактала

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void comboBox\_fractal\_SelectedIndexChanged(object sender, EventArgs e)

{

switch (this.comboBox\_type\_of\_fractal.SelectedIndex)

{

case 2:

this.textBox\_dspace.Visible = true;

this.label\_dspace.Visible = true;

Fractal Frac2 = Frac;

Frac = new Cantor();

if (Frac2 != null)

{

Frac.startColor = Frac2.startColor;

Frac.endColor = Frac2.endColor;

Frac.scf = Frac2.scf;

Frac.ecf = Frac2.ecf;

Frac.drawall = Frac2.drawall;

Frac.scale = Frac2.scale;

}

else

Init(false);

//name = name.Length>0?name:"Cantor";

//Init(false);

break;

case 1:

this.textBox\_dspace.Visible = false;

this.label\_dspace.Visible = false;

Fractal Frac21 = Frac;

Frac = new Levi();

if (Frac21 != null)

{

Frac.startColor = Frac21.startColor;

Frac.endColor = Frac21.endColor;

Frac.scf = Frac21.scf;

Frac.ecf = Frac21.ecf;

Frac.drawall = Frac21.drawall;

Frac.scale = Frac21.scale;

}

else

Init(false);

//name = name.Length > 0 ? name : "Levi";

//Init(false);

break;

case 0:

this.textBox\_dspace.Visible = false;

this.label\_dspace.Visible = false;

Fractal Frac22 = Frac;

Frac = new Gilbert();

if (Frac22 != null)

{

Frac.startColor = Frac22.startColor;

Frac.endColor = Frac22.endColor;

Frac.scf = Frac22.scf;

Frac.ecf = Frac22.ecf;

Frac.drawall = Frac22.drawall;

Frac.scale = Frac22.scale;

}

else

Init(false);

//Init(false);

//name = name.Length > 0 ? name : "Gilbert";

break;

case -1:

//DropExWindow("Выберите тип фрактала");

return;

}

pb.gfrac(Frac);

SetStartColor.Visible = true;

SetEndColor.Visible = true;

//InitializeComponent();

}

/// <summary>

/// Отрисовка фрактала

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void button1\_Click(object sender, EventArgs e)

{

if (this.Frac == null) { DropExWindow("Выберите тип фрактала"); return; }

if (this.Frac.isdrawing) return;

//this.Draw();

DrawFractal();

}

/// <summary>

/// Построение фрактала

/// </summary>

void DrawFractal()

{

if (Frac!=null && Frac.isdrawing) return;

bool f = false;

message = "";

if (Frac == null)

{

f = true;

}

//Frac = null;

this.textBox\_dspace.Visible = false;

this.label\_dspace.Visible = false;

switch (this.comboBox\_type\_of\_fractal.SelectedIndex)

{

case 0:

//Frac = new Gilbert();

break;

case 1:

//Frac = new Levi();

break;

case 2:

this.textBox\_dspace.Visible = true;

this.label\_dspace.Visible = true;

//Frac = new Cantor();

int dspace;

if (this.textBox\_dspace.TextLength == 0)

{

//Вывод окна с сообщением об ошибке

DropExWindow("Введите расстояния между шагами рекурсии");

return;

}

if (this.textBox\_dspace.TextLength == 0 || !int.TryParse(this.textBox\_dspace.Text, out dspace) || dspace < 0 || dspace > 1000)

{

//Вывод окна с сообщением об ошибке

DropExWindow("Некорректный формат введённго расстояния между шагами рекурсии");

return;

}

Frac.set\_float(dspace);

break;

}

if (Frac == null)

{

//Вывод окна с сообщением об ошибке

message = "Выберите тип фрактала";

DropExWindow(message);

return;

}

//switch (this.comboBox\_start\_color.SelectedIndex)

//{

// case 0:

// Frac.startColor = Color.Red; break;

// case 1:

// Frac.startColor = Color.Green; break;

// case 2:

// Frac.startColor = Color.Blue; break;

//}

if (!Frac.scf)

{

//Вывод окна с сообщением об ошибке

message = "Выберите начальный цвет фрактала";

DropExWindow(message);

return;

}

//switch (this.comboBox\_end\_color.SelectedIndex)

//{

// case 0:

// Frac.endColor = Color.Red; break;

// case 1:

// Frac.endColor = Color.Green; break;

// case 2:

// Frac.endColor = Color.Blue; break;

//}

if (!Frac.ecf)

{

//Вывод окна с сообщением об ошибке

message = "Выберите конечный цвет фрактала";

DropExWindow(message);

return;

}

if (this.textBox\_max\_depth\_of\_rec.TextLength == 0 || !int.TryParse(this.textBox\_max\_depth\_of\_rec.Text, out Frac.max\_level\_of\_rec) || Frac.max\_level\_of\_rec <= 0 /\*|| Frac.max\_level\_of\_rec > 1000\*/)

{

//Вывод окна с сообщением об ошибке

this.message = "Некорректный формат глубины рекурсии";

DropExWindow(message);

return;

}

textBox1\_TextChanged();

draw\_step\_by\_step = checkBox\_buffer.Checked;

if (f) Init();

try

{

this.checkBox1.Enabled = false;

this.comboBox\_type\_of\_fractal.Enabled = false;

this.SetStartColor.Enabled = false;

this.SetEndColor.Enabled = false;

this.textBox\_max\_depth\_of\_rec.Enabled = false;

this.textBox\_dspace.Enabled = false;

this.button1.Enabled = false;

this.textBox1.Enabled = false;

this.button2.Enabled = false;

this.toolStripMenuItem1.Enabled = false;

this.bmp = new Bitmap((int)((Frac.xsize + Frac.space \* 2) \* Frac.scale), (int)((Frac.ysize + Frac.space \* 2) \* Frac.scale));

//Frac.setpictureBoxsize(Width, Height);

Graphics graph = Graphics.FromImage(bmp);

//this.Draw(bmp);

Frac.drawall = checkBox1.Checked;

Frac.pen = new Pen(Frac.startColor);

Frac.brush = new SolidBrush(Frac.startColor);

this.Enabled = false || fenableformwhendrawing;

this.TopMost = false;

this.pb.Show();

this.pb.TopMost = true;

ProgressBur();

timer.Start();

Frac.pb.timer.Start();

System.Threading.Thread thr = new System.Threading.Thread(delegate() { Frac.Draw(graph); });

thr.Start();

//Frac.Draw(graph);

if (Frac.message.Length > 0)

{

//Вывод окна с сообщением об ошибке

message = Frac.message;

//Text = message;

DropExWindow(message);

}

//pictureBox\_fractal.Image = bmp;

}

catch(NullReferenceException ex)

{

DropExWindow(ex.Message);

}

catch(OverflowException ex)

{

DropExWindow(ex.Message);

}

catch(ArgumentNullException ex)

{

DropExWindow("Введёно слишком большое приближение/удаление\n" + ex.Message);

Init();

}

catch(Exception ex)

{

DropExWindow(""+ex.Message);

Init();

}

finally

{

//Frac.pb.Visible = false;

//this.Frac.pb.TopMost = true;

//Frac.pb.Enabled = false;

//this.TopMost = true;

}

//Invalidate();

}

/// <summary>

/// Переопределение перерисовки окна

/// </summary>

/// <param name="e"></param>

protected override void OnPaint(PaintEventArgs e)

{

try

{

if (bmp != null && Frac != null)

{

e.Graphics.DrawImage(bmp, posx, posy);

}

base.OnPaint(e);

}

catch(ArgumentNullException ex)

{

//Вывод окна с сообщением об ошибке

DropExWindow("Попытка вывести несуществующий фрактал" + ex.Message);

//Данное сообщение можно было получить в предыдущих версиях программы

}

catch(OverflowException ex)

{

//Вывод окна с сообщением об ошибке

//Text = "" + (posx + Frac.xspace) + " " + (posy + Frac.yspace) + " " + (Frac.scale);

DropExWindow("" + ex.Message);

}

catch(Exception ex)

{

DropExWindow("" + ex.Message);

}

}

bool f = false;

/// <summary>

/// ЛКМ нажата

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void pictureBox\_fractal\_MouseDown(object sender, MouseEventArgs e)

{

if (Frac == null || (Frac.isdrawing && !fenableformwhendrawing)) return;

if (!f)

{

pox = e.X;

poy = e.Y;

f = true;

return;

}

else

{

}

}

/// <summary>

/// ЛКМ отпущена

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void pictureBox\_fractal\_MouseUp(object sender, MouseEventArgs e)

{

if (Frac == null || (Frac.isdrawing && !fenableformwhendrawing)) return;

if (f)

{

f = false;

}

}

/// <summary>

/// Перемещение курсора

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void pictureBox\_fractal\_MouseMove(object sender, MouseEventArgs e)

{

if (Frac == null || (Frac.isdrawing && !fenableformwhendrawing)) return;

if (f)

{

float dx = e.X - pox, dy = e.Y - poy;

pox += dx;

poy += dy;

posx += dx;

posy += dy;

Invalidate();

//pictureBox\_fractal.Location = new System.Drawing.Point((int)(posx + dx), (int)(posy + dy));

}

}

/// <summary>

/// Отцентровать изображение и сбросить масштаб

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void button2\_Click(object sender, EventArgs e)

{

if (Frac == null || Frac.isdrawing) return;

if (this.Frac.isdrawing) return;

Init();

}

/// <summary>

/// Отлов нажатий клавиш

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void Form1\_KeyDown(object sender, KeyEventArgs e)

{

if (Frac == null || Frac.isdrawing) return;

if (e.KeyCode == Keys.C)

{

if (this.Frac.isdrawing) return;

Init();

}

if (e.KeyCode == Keys.E)

{

if (this.Frac.isdrawing) return;

ZoomUp();

if (Frac == null)

{

this.textBox1.Text = "1";

}

else

{

this.textBox1.Text = $"{this.Frac.scale:f3}";

}

}

if (e.KeyCode == Keys.Q)

{

if (this.Frac.isdrawing) return;

ZoomDown();

this.label5.Text = $"Масштаб: ";

if (Frac == null)

{

this.textBox1.Text = "1";

}

else

{

this.textBox1.Text = $"{this.Frac.scale:f3}";

}

}

if (e.KeyCode == Keys.B)

{

if (this.Frac.isdrawing) return;

if (checkBox\_buffer.Checked)

{

DoubleBuffered = false;

checkBox\_buffer.Checked = false;

}

else

{

DoubleBuffered = true;

checkBox\_buffer.Checked = true;

}

}

if (e.KeyCode == Keys.Enter)

{

if (this.Frac.isdrawing) return;

Rewrite();

}

////Only for developer

if (e.KeyCode == Keys.L && this.textBox1.Text=="42" && this.textBox\_max\_depth\_of\_rec.Text=="42")

{

fenableformwhendrawing ^= true;

DropExWindow("В чём заключается смысл Жизни: "+(fenableformwhendrawing?"42":"I dont know"));

}

//Text = "" + (e.KeyValue);

}

/// <summary>

/// Выбор начального цвета фрактала

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void SetStart\_Click(object sender, EventArgs e)

{

if (this.Frac.isdrawing) return;

colorDialog1.FullOpen = true;

colorDialog1.Color = Frac.startColor;

if (colorDialog1.ShowDialog() == DialogResult.OK)

{

Frac.startColor = colorDialog1.Color;

((Button)sender).BackColor = colorDialog1.Color;

((Button)sender).ForeColor = Color.FromArgb(((Button)sender).ForeColor.A,

colorDialog1.Color.R < 128 ? 255 : 0,

colorDialog1.Color.G < 128 ? 255 : 0,

colorDialog1.Color.B < 128 ? 255 : 0);

Frac.scf = true;

}

}

/// <summary>

/// Выбор конечного цвета фрактала

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void SetEndColor\_Click(object sender, EventArgs e)

{

if (this.Frac.isdrawing) return;

colorDialog1.FullOpen = true;

colorDialog1.Color = Frac.endColor;

if (colorDialog1.ShowDialog() == DialogResult.OK)

{

Frac.endColor = colorDialog1.Color;

((Button)sender).BackColor = colorDialog1.Color;

((Button)sender).ForeColor = Color.FromArgb(((Button)sender).ForeColor.A,

colorDialog1.Color.R < 128 ? 255 : 0,

colorDialog1.Color.G < 128 ? 255 : 0,

colorDialog1.Color.B < 128 ? 255 : 0);

Frac.ecf = true;

}

}

/// <summary>

/// Приближение в конкретной точке

/// </summary>

/// <param name="e"></param>

void ZoomUp(MouseEventArgs e)

{

if (this.Frac.isdrawing) return;

pox -= (posx - e.X) - (posx - e.X) \* (float)(1.5);

poy -= (posy - e.Y) - (posy - e.Y) \* (float)(1.5);

posx -= (posx - e.X) - (posx - e.X) \* (float)(1.5);

posy -= (posy - e.Y) - (posy - e.Y) \* (float)(1.5);

Frac.scale \*= (float)1.5;

//Rewrite();

}

/// <summary>

/// Приближение в левом верхнем углу

/// </summary>

void ZoomUp()

{

if (this.Frac.isdrawing) return;

Frac.scale \*= (float)1.5;

this.textBox1.Text = Frac.scale.ToString();

Rewrite();

}

/// <summary>

/// Удаление в конкретной точке

/// </summary>

/// <param name="e"></param>

void ZoomDown(MouseEventArgs e)

{

if (this.Frac.isdrawing) return;

Frac.scale /= (float)1.5;

pox -= (posx - e.X) - (posx - e.X) / (float)(1.5);

poy -= (posy - e.Y) - (posy - e.Y) / (float)(1.5);

posx -= (posx - e.X) - (posx - e.X) / (float)(1.5);

posy -= (posy - e.Y) - (posy - e.Y) / (float)(1.5);

//Rewrite();

}

/// <summary>

/// Удаление в левом вехнем углу экрана

/// </summary>

void ZoomDown()

{

if (this.Frac.isdrawing) return;

Frac.scale /= (float)1.5;

this.textBox1.Text = Frac.scale.ToString();

Rewrite();

}

/// <summary>

/// Масштабирование изображения с помощью колёсика мыши

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

void pictureBox\_fractal\_MouseWheel(object sender, MouseEventArgs e)

{

if (this.Frac.isdrawing) return;

if (Frac == null || Frac.isdrawing) return;

if (Frac == null || bmp == null) return;

if (Frac.isdrawing) return;

if (e.Delta > 0)

{

ZoomUp(e);

//Text = "Вверх";

}

else

{

ZoomDown(e);

//Text = "Вниз";

}

this.label5.Text = $"Масштаб: ";

if (Frac == null)

{

this.textBox1.Text = "1";

}

else

{

this.textBox1.Text = $"{this.Frac.scale:f3}";

}

//Text = "" + posx + " " + posy;

Rewrite();

}

/// <summary>

/// Перерисовываем фрактал

/// </summary>

void Rewrite()

{

if (this.Frac.isdrawing) return;

try

{

this.bmp = new Bitmap((int)(((Frac.xsize + Frac.space \* 2) \* Frac.scale)), (int)((Frac.ysize + Frac.space \* 2) \* Frac.scale));

//Frac.setpictureBoxsize((int)(Width\*Frac.scale), (int)(Height\*Frac.scale));

Graphics graph = Graphics.FromImage(bmp);

//Frac.Draw(graph);

DrawFractal();

}

catch (ArgumentNullException ex)

{

//Исключение выбрасываемое в предыдущих версиях программы

}

catch (System.ArgumentException ex)

{

//Вывод окна с сообщением об ошибке

DropExWindow("Слишком большое приближение/удаление \n" + ex.Message);

Init();

}

catch (OverflowException ex)

{

//Text = "" + (posx + Frac.xspace) + " " + (posy + Frac.yspace) + " " + (Frac.scale);

//Вывод окна с сообщением об ошибке

DropExWindow("Слишком большая глубина рекурсии \n" + ex.Message);

}

catch (Exception ex)

{

DropExWindow("\n" + ex.Message);

}

//Init();

Invalidate();

}

/// <summary>

/// Сохранение изображения

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void saveToolStripMenuItem1\_Click(object sender, EventArgs e)

{

try

{

if (name.Length == 0)

{

saveAsToolStripMenuItem\_Click(sender, e);

}

else

{

bmp.Save("" + name, System.Drawing.Imaging.ImageFormat.Bmp);

}

}

catch (NullReferenceException ex)

{

DropExWindow("Невозможно сохранить несуществующий оъект\n" + ex.Message);

}

catch (ArgumentNullException ex)

{

DropExWindow("Невозможно сохранить оъект\n" + ex.Message);

}

catch (System.Runtime.InteropServices.ExternalException ex)

{

DropExWindow("Невозможно сохранить оъект\n" + ex.Message);

}

catch (Exception ex)

{

DropExWindow("" + ex.Message);

}

}

/// <summary>

/// Сохранение изображения как (старый интерфейс)

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void saveToolStripMenuItem\_Click(object sender, EventArgs e)

{

try

{

FolderBrowserDialog FBD = new FolderBrowserDialog();

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

{

name = FBD.SelectedPath;

bmp.Save("" + name, System.Drawing.Imaging.ImageFormat.Bmp);

}

}

catch (NullReferenceException ex)

{

DropExWindow("Невозможно сохранить несуществующий оъект\n" + ex.Message);

}

catch (ArgumentNullException ex)

{

DropExWindow("Невозможно сохранить оъект\n" + ex.Message);

}

catch (System.Runtime.InteropServices.ExternalException ex)

{

DropExWindow("Невозможно сохранить оъект\n" + ex.Message);

}

catch (Exception ex)

{

DropExWindow("" + ex.Message);

}

}

/// <summary>

/// Сохранение изображения как (новый интерфейс)

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void saveAsToolStripMenuItem\_Click(object sender, EventArgs e)

{

try

{

if (Frac == null) throw (new NullReferenceException());

SaveFileDialog FBD = new SaveFileDialog();

FBD.Filter = "Изображения (\*.bmp)|\*.bmp|Все файлы (\*.\*)|\*.\*";

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

{

name = FBD.FileName;

bmp.Save("" + name, System.Drawing.Imaging.ImageFormat.Bmp);

}

}

catch (NullReferenceException ex)

{

DropExWindow("Невозможно сохранить несуществующий оъект\n" + ex.Message);

}

catch (ArgumentNullException ex)

{

DropExWindow("Невозможно сохранить оъект\n" + ex.Message);

}

catch (System.Runtime.InteropServices.ExternalException ex)

{

DropExWindow("Невозможно сохранить оъект\n" + ex.Message);

}

catch (ArgumentException ex)

{

DropExWindow("Невозможно сохранить оъект\n" + ex.Message);

}

catch (Exception ex)

{

DropExWindow("" + ex.Message);

}

}

/// <summary>

/// Новое окно фрактала

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void newToolStripMenuItem\_Click(object sender, EventArgs e)

{

if (this.Frac!=null && this.Frac.isdrawing) return;

(new Form1(pb)).ShowDialog(new Form1(pb));

}

/// <summary>

/// Сколько уровней рекурсии отрисовывать

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void checkBox1\_CheckedChanged(object sender, EventArgs e)

{

if (Frac != null && !this.Frac.isdrawing) Frac.drawall = checkBox1.Checked;

//if (this.Frac.isdrawing) return;

}

/// <summary>

/// Изменение масштаба через поле для ввода

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void textBox1\_TextChanged()

{

//if (this.Frac.isdrawing) return;

float sc;

if (!(float.TryParse(this.textBox1.Text, out sc) && sc > 0.05 && sc <= 51.8)) { DropExWindow("Неверное значение масштаба"); Init(false); return; }

if (Frac == null) return;

Frac.xspace = this.pictureBox1.Width;

Frac.yspace = 22;

//posx = Frac.xspace;//(Width - (Frac.xsize + Frac.space \* 2) + Frac.xspace) / 2;

//posy = Frac.yspace;// (Height - (Frac.ysize + Frac.space \* 2) + Frac.yspace) / 2;

Frac.xleft = posx;

Frac.yleft = posy;

Frac.scale = sc;

if (!Frac.scf || !Frac.ecf) return;

//DrawFractal();

//Rewrite();

//Invalidate();

}

/// <summary>

/// Вывод сообщения об ошибке

/// </summary>

/// <param name="s"></param>

void DropExWindow(string s)

{

if (flagmb) return;

flagmb = true;

if(MessageBox.Show(s) == DialogResult.OK)

{

flagmb = false;

}

}

/// <summary>

/// Новое окно ожидания

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void ProgressBur()

{

Frac.isdrawing = true;

Frac.Max\_length = (Frac.max\_level\_of\_rec);

Frac.step = (float)(10000.0) / Frac.Max\_length;

Frac.max\_length = 10000;

this.Frac.pbm = this.Frac.max\_length;

Frac.summ = 0;

Frac.pb = this.pb;

Frac.pb.init();

Frac.pb.gfrac(Frac);

//Frac.pb.Visible=true;

//System.Threading.Thread thr = new System.Threading.Thread(Frac.pb.Draw);

//Frac.pb.Draw();

//thr.Start();

}

/// <summary>

/// Отрисовка формы во время построения фрактала

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

void timer\_Tick(object sender, EventArgs e)

{

if(draw\_step\_by\_step) Invalidate();

if (!Frac.isdrawing) end\_of\_Draw\_Fractal();

}

/// <summary>

/// Отрисовывать все уровни постепенно?

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void checkBox\_buffer\_CheckedChanged(object sender, EventArgs e)

{

draw\_step\_by\_step = checkBox\_buffer.Checked;

}

/// <summary>

/// Конец отрисовки фрактала

/// </summary>

void end\_of\_Draw\_Fractal() {

//this.Activate();

this.Enabled = true;

Frac.isdrawing = false;

this.pb.Hide();

this.pb.TopMost = false;

this.Frac.pb.Hide();

this.Frac.pb.TopMost = false;

this.TopMost = overAllWindowsToolStripMenuItem.Checked;

timer.Stop();

Frac.pb.timer.Stop();

this.checkBox1.Enabled = true;

this.comboBox\_type\_of\_fractal.Enabled = true;

this.SetStartColor.Enabled = true;

this.SetEndColor.Enabled = true;

this.textBox\_max\_depth\_of\_rec.Enabled = true;

this.textBox\_dspace.Enabled = true;

this.button1.Enabled = true;

this.textBox1.Enabled = true;

this.button2.Enabled = true;

this.toolStripMenuItem1.Enabled = true;

Invalidate();

}

/// <summary>

/// Изменение выбранности пункта меню Поверх остальных окон и статуса Поверх остальных окон основного окна

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void overAllWindowsToolStripMenuItem\_Click(object sender, EventArgs e)

{

overAllWindowsToolStripMenuItem.Checked ^= true;

TopMost = overAllWindowsToolStripMenuItem.Checked;

}

/// <summary>

/// Стартовые значения позиции и размера и перерисование фрактала

/// </summary>

private void Init()

{

if (Frac == null) return;

if (this.Frac.isdrawing) return;

Frac.xspace = this.pictureBox1.Width;

Frac.yspace = 22;

posx = Frac.xspace;//(Width - (Frac.xsize + Frac.space \* 2) + Frac.xspace) / 2;

posy = Frac.yspace;// (Height - (Frac.ysize + Frac.space \* 2) + Frac.yspace) / 2;

Frac.xleft = posx;

Frac.yleft = posy;

Frac.scale = 1;

this.textBox1.Text = Frac.scale.ToString();

if (!Frac.scf || !Frac.ecf) return;

DrawFractal();

Invalidate();

}

/// <summary>

/// Стартовые значения позиции и размера и если draw, то перерисование фрактала

/// </summary>

/// <param name="draw">Перерисовать фрактал?Да:Нет</param>

private void Init(bool draw)

{

if (this.Frac.isdrawing) return;

if (Frac == null) return;

Frac.xspace = this.pictureBox1.Width;

Frac.yspace = 22;

posx = Frac.xspace;//(Width - (Frac.xsize + Frac.space \* 2) + Frac.xspace) / 2;

posy = Frac.yspace;// (Height - (Frac.ysize + Frac.space \* 2) + Frac.yspace) / 2;

Frac.xleft = posx;

Frac.yleft = posy;

Frac.scale = 1;

this.textBox1.Text = Frac.scale.ToString();

if (!Frac.scf || !Frac.ecf) return;

if (draw)

{

DrawFractal();

Invalidate();

}

}

/// <summary>

/// Закрытие приложения

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

void Form1Closed(object sender, EventArgs e)

{

if (Frac == null || Frac.pb == null)

{

pb.isexit = true;

}

else

{

Frac.isdrawing = false;

Frac.pb.isexit = true;

Fractal.handle.WaitOne();

}

Dispose();

Application.Exit();

}

#region Old features

//public Form1() : base()

//{

// DoubleBuffered = true;

// //Init();

//}

//private void label1\_Click(object sender, EventArgs e)

//{

//}

//private void pictureBox\_fractal\_Click(object sender, EventArgs e)

//{

//}

/// <summary>

/// Not using method

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void textBox\_dspace\_TextChanged(object sender, EventArgs e)

{

if (this.Frac.isdrawing) return;

}

/// <summary>

/// Not using method

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void comboBox\_start\_color\_SelectedIndexChanged(object sender, EventArgs e)

{

if (this.Frac.isdrawing) return;

}

/// <summary>

/// Not using method

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void comboBox\_end\_color\_SelectedIndexChanged(object sender, EventArgs e)

{

if (this.Frac.isdrawing) return;

}

/\*private void checkBox\_buffer\_CheckedChanged(object sender, EventArgs e)

{

DoubleBuffered = checkBox\_buffer.Checked;

Text = DoubleBuffered?"True":"False";

}\*/

/// <summary>

/// Изменение масштаба через поле для ввода

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

//private void textBox1\_TextChanged(object sender, EventArgs e)

//{

// if (this.Frac.isdrawing) return;

// float sc;

// if (!(float.TryParse(this.textBox1.Text, out sc) && sc > 0 && sc <= 51.8)) { DropExWindow("Неверное значение масштаба"); return; }

// if (Frac == null) return;

// Frac.xspace = this.pictureBox1.Width;

// Frac.yspace = 22;

// //posx = Frac.xspace;//(Width - (Frac.xsize + Frac.space \* 2) + Frac.xspace) / 2;

// //posy = Frac.yspace;// (Height - (Frac.ysize + Frac.space \* 2) + Frac.yspace) / 2;

// Frac.xleft = posx;

// Frac.yleft = posy;

// Frac.scale = sc;

// if (!Frac.scf || !Frac.ecf) return;

// //DrawFractal();

// //Rewrite();

// //Invalidate();

//}

#endregion

}

}

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 System.Windows.Forms;

using KDZ\_1;

namespace KDZ\_1

{

/// <summary>

/// Класс для вывода шкалы прогресса построения фрактала

/// </summary>

public partial class ProgressBur : Form

{

/// <summary>

/// ссылка на фрактал, для получения информации об отрисовке фрактала

/// </summary>

private Fractal frac;

/// <summary>

/// bool переменная, обозначающая нужно ли завершать выполнение

/// </summary>

public bool isexit = false;

/// <summary>

/// таймер для обновления шкалы прогресса отрисовки фрактала

/// </summary>

public Timer timer = new Timer();

/// <summary>

/// Шкала прогресса

/// </summary>

public System.Windows.Forms.ProgressBar progressBar1 = new System.Windows.Forms.ProgressBar();

/// <summary>

/// Получение ссылки на фрактал

/// </summary>

/// <param name="f"></param>

public void gfrac(object f)

{

this.frac = (Fractal)f;

init();

}

//private Bitmap bmp = null;

/// <summary>

/// Конструктор устанавливающий ссылку на фрактал

/// </summary>

/// <param name="f">Ссылка на фрактал</param>

public ProgressBur(Object f)

{

this.Closed += ProgressBurClosed;

frac = (Fractal)f;

InitializeComponent();

init();

//this.TopMost = true;

this.Hide();

//this.Enabled = false;

this.Visible = false;

timer.Interval = 10; //интервал между срабатываниями 10 миллисекунд

timer.Tick += new EventHandler(Draw);

}

/// <summary>

/// Инициализация шкалы прогресса

/// </summary>

public void init()

{

this.progressBar1.Minimum = 0;

this.progressBar1.Maximum = frac.max\_length;

//this.progressBar1.Step = 1;

}

/// <summary>

/// Обновление шкалы прогресса

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

public void Draw(object sender, EventArgs e)

{

//init();

//this.Activate();

//this.Show();

//this.TopMost = true;

//this.TopMost = true;

//while (frac.isdrawing)

//{

//if (!frac.isdrawing) break;

//this.progressBar1.Value = frac.pb.progressBar1.Value;

//Invalidate();

//}

//this.Hide();

//this.TopMost = false;

this.Enabled = true;

Invalidate();

}

/// <summary>

/// Переопределение метода закрытия окна шкалы прогресса, для отключения возможности его закрыть

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

void ProgressBurClosed(object sender, EventArgs e)

{

//Do nothing

return;

}

/// <summary>

/// Отлов события Закрытие окна шкалы прогресса и прекращение отрисовки или выход из программы

/// </summary>

/// <param name="e"></param>

protected override void OnFormClosing(FormClosingEventArgs e)

{

if (!isexit)

{

frac.isdrawing=false;

e.Cancel = true;

}

//Hide();

}

/// <summary>

/// Отлов нажатий клавиш

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void ProgressBur\_KeyDown(object sender, KeyEventArgs e)

{

if (e.KeyCode == Keys.Escape)

{

this.OnFormClosing(new FormClosingEventArgs(new CloseReason(), false));

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Threading;

using System.Windows.Forms;

namespace KDZ\_1

{

static class Program

{

public class MultiFormContext : ApplicationContext

{

private int openForms;

public MultiFormContext(params Form[] forms)

{

openForms = forms.Length;

foreach (var form in forms)

{

form.FormClosed += (s, args) =>

{

//When we have closed the last of the "starting" forms,

//end the program.

if (Interlocked.Decrement(ref openForms) == 0)

ExitThread();

};

form.Show();

}

}

}

/// <summary>

/// The main entry point for the application.

/// </summary>

[STAThread]

static void Main()

{

//for developer only

bool debug = false;

//Заставляем всех использовать английскую локаль

System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");

while (true)

{

try

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

//Application.Run(new Form1());

ProgressBur pb = new ProgressBur(new Fractal());

Application.Run(new MultiFormContext(new Form1(pb), pb));

break;

}

catch (Exception ex)

{

if (debug) // For developers only

{

Console.WriteLine("Произошла непредвиденная ошибка\n" + ex.Message + '\n' + ex.Source + '\n' + ex.StackTrace + '\n' + ex.ToString());

Console.WriteLine("\nДля выхода из программы нажмите ESC\n Для перезапуска программы - клавишу Enter");

if (Console.ReadKey(true).Key != ConsoleKey.Escape) continue;

}

}

if(debug) break;

}

}

}

}

# 9. Список литературы