титульный

Содержание

[1 Формулировка задания 3](#_Toc466834821)

[2 Информационная модель 3](#_Toc466834822)

[3 Алгоритм 5](#_Toc466834823)

[4 Макет интерфейса 6](#_Toc466834824)

[5 Лицензия 7](#_Toc466834825)

[6 Листинг 8](#_Toc466834826)

[Приложения 13](#_Toc466834827)

# 1 Формулировка задания

Ознакомится с понятиями «фрактал» и « фрактальные изображения». Выбрать для программной реализации один из существующих фракталов. Разработать программный продукт в соответствии со следующими пунктами:

1. Осуществить вывод фрактального изображения
2. В зависимости от размера формы рассчитать порядок фрактала, а также длину его штриха и количество итераций
3. Сделать буферизацию
4. Создать структуру/класс
5. Реализовать поток

В процессе разработки использовать любую из систем управления версий. Создание схем и алгоритмов осуществить по средствам онлайн инструментов. К готовому продукту написать лицензию. Распределение командной работы на усмотрение.

# 2 Информационная модель

Фрактал - математическое множество, обладающее свойством самоподобия (объект, в точности или приближённо совпадающий с частью себя самого, то есть целое имеет ту же форму, что и одна или более частей). В математике под фракталами понимают множества точек в евклидовом пространстве, имеющие дробную метрическую размерность, либо метрическую размерность, отличную от топологической, поэтому их следует отличать от прочих геометрических фигур, ограниченных конечным числом звеньев. Самоподобные фигуры, повторяющиеся конечное число раз, называются предфракталами.

Самоподобные множества с необычными свойствами в математике:

1. Множество Кантора
2. Треугольник Серпинского
3. Кривая Коха
4. Кривая Гильберта
5. …

Кривая Гильберта (известная также как заполняющая пространство кривая Гильберта) — это непрерывная фрактальная заполняющая пространство кривая, впервые описанная немецким математиком Давидом Гильбертом в 1891 году, как вариант заполняющих пространство кривых Пеано, открытых итальянским математиком Джузеппе Пеано в 1890 году.

Поскольку кривая заполняет плоскость, её размерность Хаусдорфа равна 2 (в точности, её образ является единичным квадратом, размерность которого равна 2 при любом определении размерности, а её граф является компактным множеством, гомеоморфным замкнутому единичному интервалу с хаусдорфовой размерностью 2).

является *n*-м приближением к предельной кривой. Евклидова длина кривой равна , то есть растёт экспоненциально от *n*, будучи в то же время всегда в пределах квадрата с конечной площадью.

Кривая гильберта показана на рисунке 1.

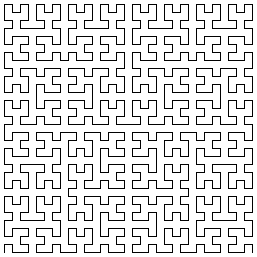


Рисунок 1 – кривая гильберта

# 3 Алгоритм

Основным элементом кривой Гильберта является П-образный элемент. Он показан на рисунке 2. Он является кривой Гильберта первого порядка.

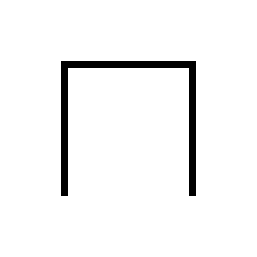


Рисунок 2 – элемент кривой

Кривая второго порядка строиться на 4 копиях кривой первого порядка, две одинаковые параллельно расположенные и две передвинутые и повернутые на четверть оборота. Соединены между собой тремя линиями, длина линии равна длине стороны элемента кривой. Кривая второго порядка представлена на рисунке 3.

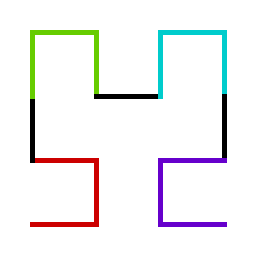


Рисунок 3 – кривая второго порядка

Кривая третьего порядка получается путем копирования кривой второго порядка и так далее. Кривые третьего и четвертого порядка представлены на рисунке 4 и 5.

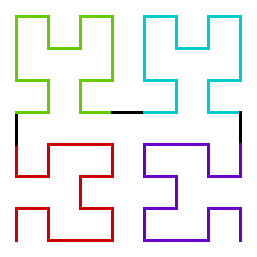


Рисунок 4 –кривая третьего порядка

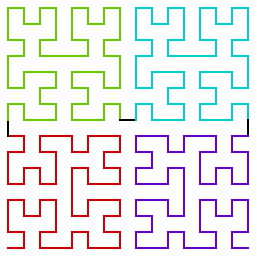


Рисунок 5 –кривая четвертого порядка

Алгоритмы функций, осуществляющих просчет и рисование кривой Гильберта, представлены в приложении а. Тесты в приложении б.

# 4 Макет интерфейса

Макет интерфейса представлен на рисунке 6. На нем вы можете видеть форму (окно) и изображение.

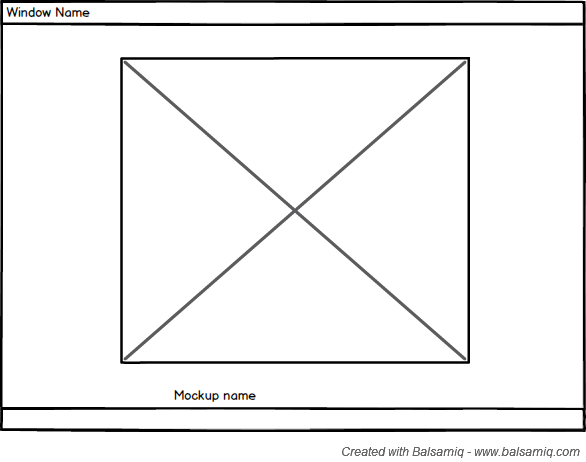


Рисунок 6 – макет

# 5 Лицензия

Текст лицензии: Программное обеспечение может распространяться на условии открытого кода (OpenWare), или же без такого условия.

При распространении программного обеспечения на условиях открытого кода правообладатель, предоставляя пользователю, право использования программного обеспечения, передает также исходные коды программы. При этом пользователю может предоставляться право модифицировать исходные тексты в целях их переработки и совершенствования.

Последующее использование полученных в результате такой переработки программных продуктов различается в зависимости от вида лицензии. В настоящее время существуют две группы типовых лицензий на передачу программного обеспечения с открытым кодом: GNU GPL и FreeBSD.

Основное их различие заключается в «наследуемости» свойства открытого кода: согласно условиям лицензии GNU GPL, все программные продукты, полученные в результате переработки или модернизации распространяемого на таких условиях программного кода, также могут распространяться далее только на условиях GNU GPL. Это, с одной стороны, способствует прогрессу в развитии программного обеспечения, с другой – нарушает имущественные интересы некоторых разработчиков, вложивших серьезные средства в модернизацию программного кода.

Бoльшую свободу в использовании передаваемого программного обеспечения предоставляет лицензия Free BSD. По условиям этой лицензии, программные продукты, полученные в результате переработки предоставленного программного кода, могут распространяться на любых условиях, в том числе и на возмездной основе.

Лицензии FreeBSD и GNU GPL получили широкое распространение, но они не являются единственно возможными лицензиями на передачу программного обеспечения с открытым кодом. Правообладателем, при необходимости, могут быть выработаны собственные условия предоставления прав на такое программное обеспечение, в большей или меньшей степени ограничивающие права пользователя.  
 Лицензии на распространение программного обеспечения, не содержащие условия об открытости исходных кодов, более разнообразны. Каждый правообладатель может выработать свои собственные условия предоставления прав на ПО. Практически во всех таких лицензиях содержится запрет на любую модификацию программного кода, если только такая модификация не разрешена прямо в законодательстве (например, адаптация программы).

# 6 Листинг

#pragma once

#include <stdlib.h>

#include <time.h>

#include <math.h>

namespace ComandProjectInst {

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

using namespace System::Threading;

bool kl;

public ref class MyForm : public System::Windows::Forms::Form

{

public:

Bitmap^bmp = gcnew Bitmap(1000, 1000);

Graphics^gp = Graphics::FromImage(bmp);

Pen^pen = gcnew Pen(Brushes::DarkRed);

public:

ref class piy

{

public:

int x1;

int y1;

int x2;

int y2;

int u;

int Kol\_shtrihov;

int shtrih;

int kol\_p;

static void ThreadProc()

{

Form ^Form1 = gcnew Form;

Form1->Size = System::Drawing::Size(250, 150);

Form1->BackColor = Color::YellowGreen;

Label ^label = gcnew Label;

Form1->Controls->Add(label);

label->Location = System::Drawing::Point(0, 0);

label->Size = System::Drawing::Size(250, 150);

label->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 28, FontStyle::Regular, GraphicsUnit::Point));

label->Text = System::Convert::ToString("Поток");

Form1->ShowDialog();

if (kl == false)

{

System::Windows::Forms::MessageBox::Show("Поток будет остановлен");

Thread::Sleep(0);

}

}

void a(int i, Graphics^gp, Pen^pen)

{

if (i > 0)

{

d(i - 1, gp, pen);

gp->DrawLine(pen, x1, y1, x2 = x2 + u, y2);

x1 = x2;

y1 = y2;

a(i - 1, gp, pen);

gp->DrawLine(pen, x1, y1, x2, y2 = y2 + u);

x1 = x2;

y1 = y2;

a(i - 1, gp, pen);

gp->DrawLine(pen, x1, y1, x2 = x2 - u, y2);

x1 = x2;

y1 = y2;

c(i - 1, gp, pen);

}

}

void b(int i, Graphics^gp, Pen^pen)

{

if (i > 0)

{

c(i - 1, gp, pen);

gp->DrawLine(pen, x1, y1, x2 = x2 - u, y2);

x1 = x2;

y1 = y2;

b(i - 1, gp, pen);

gp->DrawLine(pen, x1, y1, x2, y2 = y2 - u);

x1 = x2;

y1 = y2;

b(i - 1, gp, pen);

gp->DrawLine(pen, x1, y1, x2 = x2 + u, y2);

x1 = x2;

y1 = y2;

d(i - 1, gp, pen);

}

}

void c(int i, Graphics^gp, Pen^pen)

{

if (i > 0)

{

b(i - 1, gp, pen);

gp->DrawLine(pen, x1, y1, x2, y2 = y2 - u);

x1 = x2;

y1 = y2;

c(i - 1, gp, pen);

gp->DrawLine(pen, x1, y1, x2 = x2 - u, y2);

x1 = x2;

y1 = y2;

c(i - 1, gp, pen);

gp->DrawLine(pen, x1, y1, x2, y2 = y2 + u);

x1 = x2;

y1 = y2;

a(i - 1, gp, pen);

}

}

void d(int i, Graphics ^gp, Pen^pen)

{

if (i > 0)

{

a(i - 1, gp, pen);

gp->DrawLine(pen, x1, y1, x2, y2 = y2 + u);

x1 = x2;

y1 = y2;

d(i - 1, gp, pen);

gp->DrawLine(pen, x1, y1, x2 = x2 + u, y2);

x1 = x2;

y1 = y2;

d(i - 1, gp, pen);

gp->DrawLine(pen, x1, y1, x2, y2 = y2 - u);

x1 = x2;

y1 = y2;

b(i - 1, gp, pen);

}

}

private:

};

piy^Piy = gcnew piy;

MyForm(void)

{

InitializeComponent();

}

protected:

~MyForm()

{

if (components)

{

delete components;

}

}

private: System::Windows::Forms::PictureBox^ pictureBox1;

protected:

protected:

private:

System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code

void InitializeComponent(void)

{

this->pictureBox1 = (gcnew System::Windows::Forms::PictureBox());

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->pictureBox1))->BeginInit();

this->SuspendLayout();

//

// pictureBox1

//

this->pictureBox1->Anchor = static\_cast<System::Windows::Forms::AnchorStyles>((System::Windows::Forms::AnchorStyles::Top | System::Windows::Forms::AnchorStyles::Bottom));

this->pictureBox1->BackColor = System::Drawing::Color::LightBlue;

this->pictureBox1->Location = System::Drawing::Point(106, 0);

this->pictureBox1->Name = L"pictureBox1";

this->pictureBox1->Size = System::Drawing::Size(256, 256);

this->pictureBox1->TabIndex = 1;

this->pictureBox1->TabStop = false;

//

// MyForm

//

this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);

this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;

this->ClientSize = System::Drawing::Size(488, 256);

this->Controls->Add(this->pictureBox1);

this->Name = L"MyForm";

this->Text = L"MyForm";

this->FormClosing += gcnew System::Windows::Forms::FormClosingEventHandler(this, &MyForm::MyForm\_FormClosing);

this->Load += gcnew System::EventHandler(this, &MyForm::MyForm\_Load);

this->Resize += gcnew System::EventHandler(this, &MyForm::MyForm\_Resize);

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->pictureBox1))->EndInit();

this->ResumeLayout(false);

}

#pragma endregion

//существует 4 вида позиции стандартной формы "ѕ"

private: System::Void MyForm\_Load(System::Object^ sender, System::EventArgs^ e) {

kl = true;

pictureBox1->Height = MyForm::Height - 40;

pictureBox1->Width = MyForm::Height - 40;

pictureBox1->Top = ClientSize.Height / 2 - pictureBox1->Height / 2;

pictureBox1->Left = ClientSize.Width / 2 - pictureBox1->Width / 2;

Piy->x1 = 1;

Piy->y1 = 1;

Piy->x2 = 1;

Piy->y2 = 1;

Piy->u = 2;

Piy->a(8, gp, pen);

Thread^ oThread = gcnew Thread(gcnew ThreadStart(&piy::ThreadProc));

oThread->Start();

this->pictureBox1->Image = bmp;

}

private: System::Void MyForm\_Resize(System::Object^ sender, System::EventArgs^ e) {

pictureBox1->Height = MyForm::Height - 40;

pictureBox1->Width = MyForm::Height - 40;

pictureBox1->Top = ClientSize.Height / 2 - pictureBox1->Height / 2;

pictureBox1->Left = ClientSize.Width / 2 - pictureBox1->Width / 2;

Piy->shtrih = 2; //минимальна¤ длина штриха

gp->Clear(Color::LightBlue);

int pr = pictureBox1->Height - 2; // рабоча¤ переменна¤ хран¤ща¤ размерность

int p = pictureBox1->Height - 2;

while (pr > 0)

{

if ((pr & (pr - 1)) == 0) break;

pr = pr - 1;

}

int kol\_p = (log10(double(pr)) / log10(double(2)));// kol\_p-пор¤док

int k = pow(2, double(kol\_p)) - 1 / pow(2, double(kol\_p));//k=2^n-1/2^n минимально зан¤та¤ площадь

Piy->x1 = 1;

Piy->y1 = 1;

Piy->x2 = 1;

Piy->y2 = 1;

Piy->u = k / ((3 + kol\_p - 1) \* kol\_p);

if (Piy->u <3) Piy->u = 2;

Piy->a(kol\_p, gp, pen);

this->pictureBox1->Image = bmp;

}

private: System::Void MyForm\_FormClosing(System::Object^ sender, System::Windows::Forms::FormClosingEventArgs^ e) {

kl = false;

}

};

}

# Приложение А

**Алгоритмы функций**

Алгоритмы функций a,b,c,d представлены на рисунках А.1, А.2, А.3, А.4.



Рисунок А.1 – алгоритм *a*



Рисунок А.2 – алгоритм *b*



Рисунок А.3 – алгоритм *c*



Рисунок А.4 – алгоритм *d*

Граф состояний показан на рисунке А.5.



Рисунок А.5 – граф состояний

# Приложение Б

**Тесты**

Ни рисунке Б.1, Б.2 представлены примеры отрисованной кривой Гильберта.

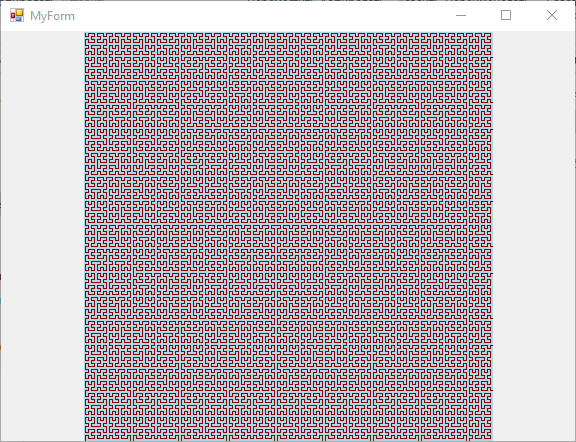


Рисунок Б.1 – кривая Гильберта

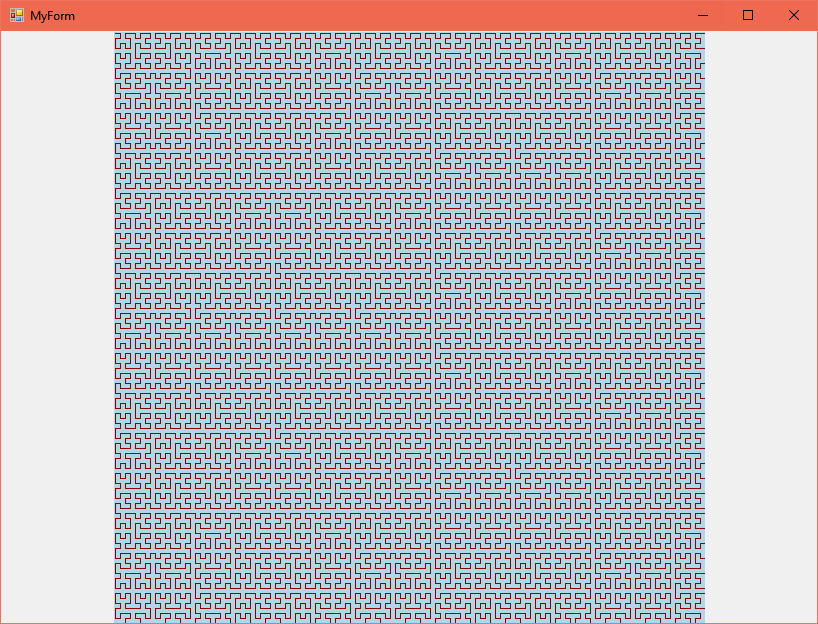


Рисунок Б.2 – кривая Гильберта

На рисунке Б.3 показано окно потока.

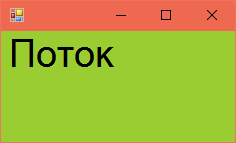


Рисунок Б.3 – поток

На рисунке Б.4 показано сообщение о закрытии потока.

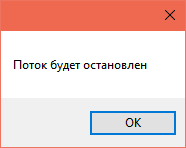


Рисунок Б.4 - сообщение