**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

отчет

**по лабораторной работе №4**

**по дисциплине «Компьютерная графика»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 8383 |  | Киреев К.А. |
| Студент гр. 8383 |  | Муковский Д.В. |
| Преподаватель |  | Герасимова Т. В. |

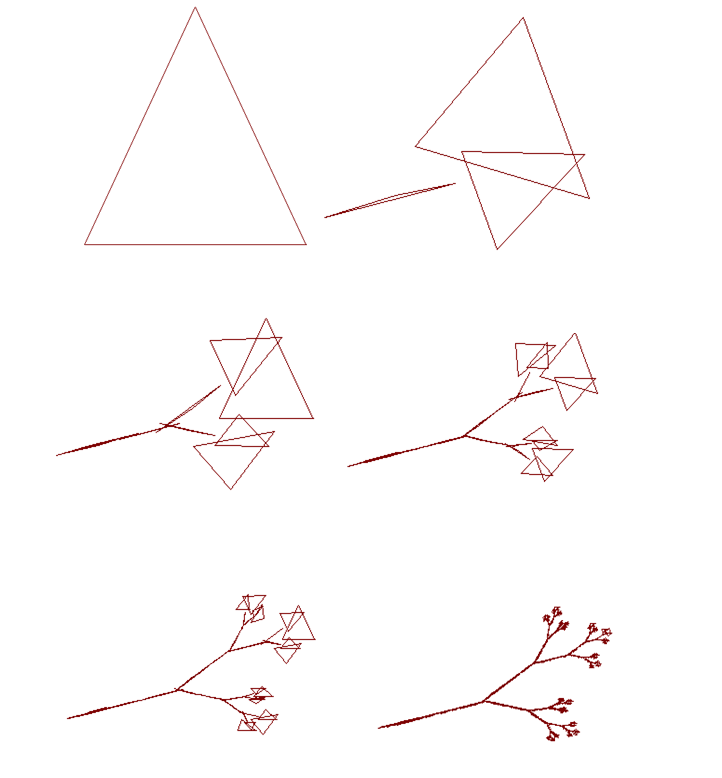
Санкт-Петербург

2021

**Задание.**

**Вариант 31.**

### *IFS*-фракталы “Ветка”



**Теоретические положения.**

### ***Система итерирующих функций IFC***

Система итерирующих функций IFC Применение таких преобразований, которые дают ту фигуру которую необходимо. Система итерирующих функ-ций - это совокупность сжимающих аффинных преобразований. Как извес-тно, аффинные преобразования включают в себя масштабирование, поворот и параллельный перенос. Афинное преобразование считается сжимающим, если коэффициент масштабирования меньше единицы.

Рассмотрим подробнее построение кривой Кох с использованием аф-финных преобразований. Каждый новый элемент кривой содержит четыре звена, полученных из образующего элемента использованием масштабиро-вания, поворота и переноса.

1. Для получения первого звена достаточно сжать исходный отрезок в три раза. Следует отметить, что тоже масштабирование применяется для всех звеньев.

2. Следующее звено строится с использованием всех возможных пре-образований, а именно: сжатие в три раза, поворот на - 60о и параллельный перенос на 1/3 по оси X.

3. Третье звено строится аналогично второму: сжатие в три раза, пово-рот на 60о, параллельный перенос на 2/3 по оси X.

4. Последнее звено: сжатие в три раза, параллельный перенос на 2/3 по оси X.

В дальнейшем правила построения кривой Кох будем называть IFS для кривой Кох.

На первой итерации кривая состоит из 4 фрагментов с коэффициентом сжатия r =1/3, два сегмента повернуты на 60 град. по час. и против час. ст.

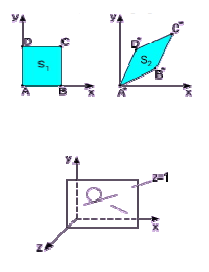
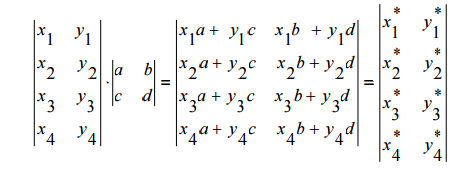
f1(x) -> масшт. на r

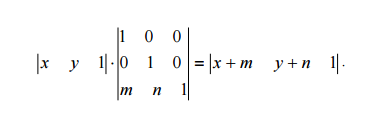
f2(x) -> масшт. на r, поворот на 600

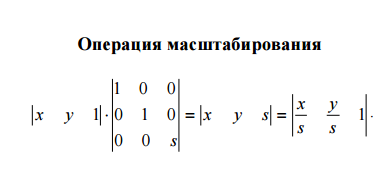
f3(x) -> масшт. на r, поворот на - 600

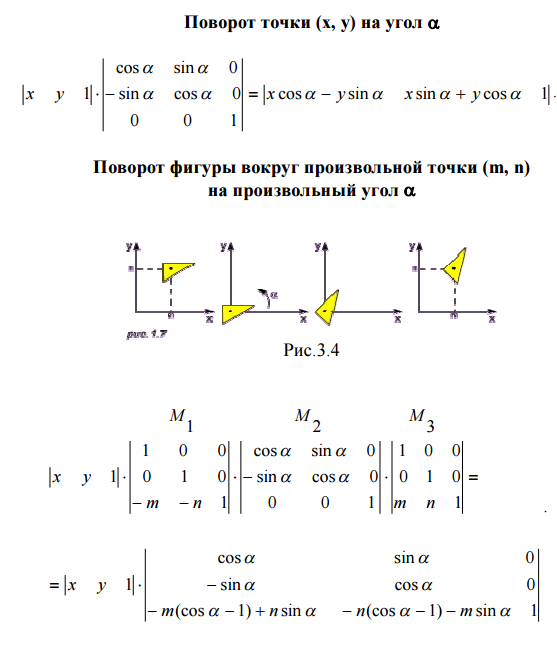
f4(x) -> масшт. на r

***Преобразования в двухмерном пространстве:***

******Деформация фигуры:

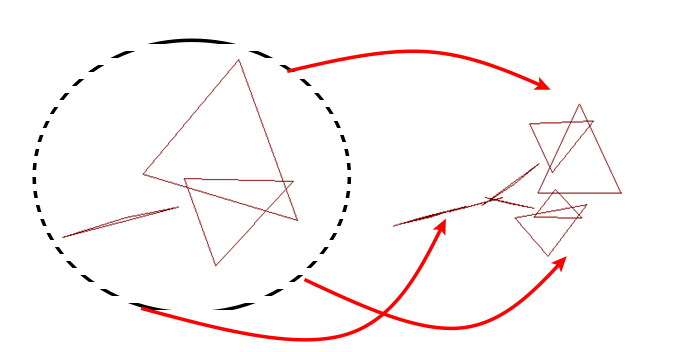
******Смещение фигуры:

******

******

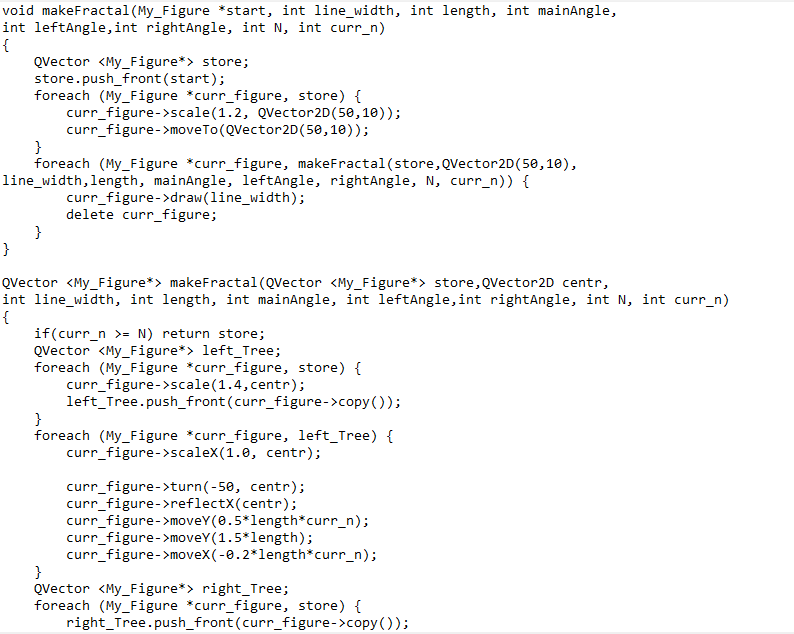
**Выполнение работы**

На каждом шаге множество фигур разбивается, путём поворота и изменения размера, преобразуются в подобие ветки дерева. Работа выполнена в среде разработки Qt.



Был написал класс GLWidget, в котором реализованы слоты для определения текущих настроек openGL.

Так же была написана рекурсивная формула для вычисления фрактала:



**Тестирование программы.**

На рисунках ниже приведены результаты тестирования программы с 1, 2, 3, 4 и 5 уровнями ветки соответственно.

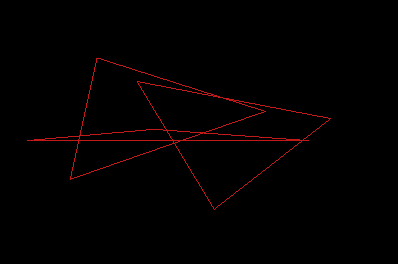


Рисунок 2 – “ветка” 1-го уровня

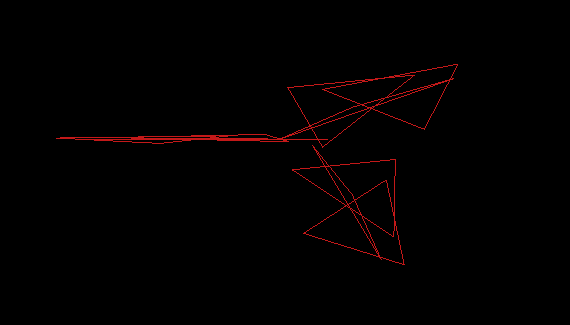


Рисунок 3 – “ветка” 2-го уровня

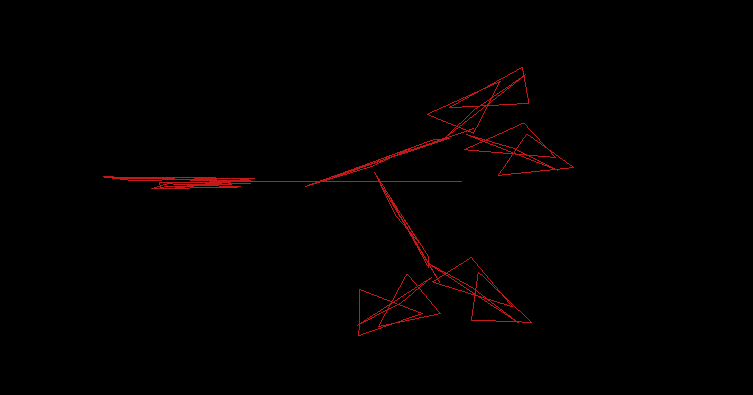


Рисунок 4 – “ветка” 3-го уровня

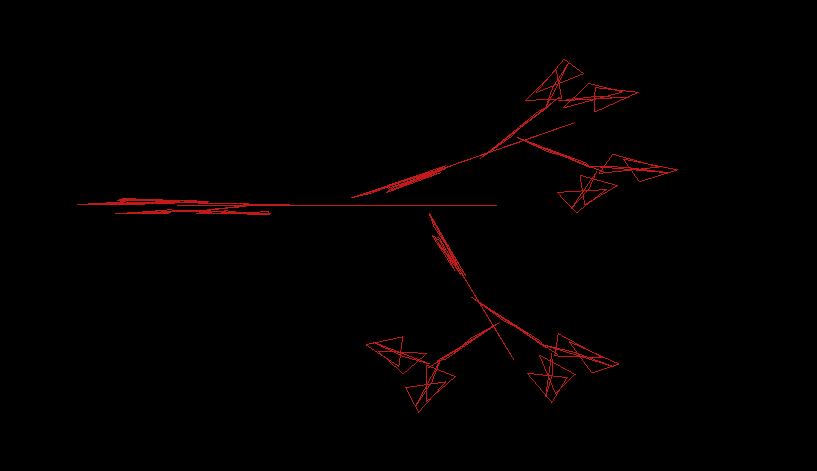


Рисунок 5 – “ветка” 4-го уровня



Рисунок 6 – “ветка” 5-го уровня

**Вывод.**

В результате выполнения лабораторной мной была написана программа, реализующая построение изображения с помощью фракталов.

**ПРИЛОЖЕНИЕ**

В данном приложении приведён исходный код виджета, реализующего работу с графикой.

#include "fractal.h"

#include "drawfunction.h"

#include "graf\_classes/my\_line.h"

#include "qmath.h"

void **makeFractal**(My\_Figure \*start, int line\_width, int length, int mainAngle, int leftAngle,int rightAngle, int N, int curr\_n)

{

QVector <My\_Figure\*> store;

store.push\_front(start);

foreach (My\_Figure \*curr\_figure, store) {

curr\_figure->*scale*(1.2, QVector2D(50,10));

curr\_figure->*moveTo*(QVector2D(50,10));

}

foreach (My\_Figure \*curr\_figure, makeFractal(store,QVector2D(50,10), line\_width,length, mainAngle, leftAngle, rightAngle, N, curr\_n)) {

curr\_figure->*draw*(line\_width);

*delete* curr\_figure;

}

}

QVector <My\_Figure\*> **makeFractal**(QVector <My\_Figure\*> store,QVector2D centr, int line\_width, int length, int mainAngle, int leftAngle,int rightAngle, int N, int curr\_n)

{

*if*(curr\_n >= N) *return* store;

QVector <My\_Figure\*> left\_Tree;

foreach (My\_Figure \*curr\_figure, store) {

curr\_figure->*scale*(1.4,centr);

left\_Tree.push\_front(curr\_figure->*copy*());

}

foreach (My\_Figure \*curr\_figure, left\_Tree) {

curr\_figure->*scaleX*(1.0, centr);

curr\_figure->*turn*(-50, centr);

curr\_figure->*reflectX*(centr);

curr\_figure->*moveY*(0.5\*length\*curr\_n);

curr\_figure->*moveY*(1.5\*length);

curr\_figure->*moveX*(-0.2\*length\*curr\_n);

}

QVector <My\_Figure\*> right\_Tree;

foreach (My\_Figure \*curr\_figure, store) {

right\_Tree.push\_front(curr\_figure->*copy*());

}

foreach (My\_Figure \*curr\_figure, right\_Tree) {

curr\_figure->*turn*(50, centr);

curr\_figure->*scale*(1.5, centr);

curr\_figure->*reflectX*(centr);

curr\_figure->*moveY*(0.5\*length\*curr\_n);

curr\_figure->*moveY*(1.5\*length);

curr\_figure->*moveX*(0.2\*length\*curr\_n);

}

QVector <My\_Figure\*> down\_left\_Tree;

foreach (My\_Figure \*curr\_figure, store) {

down\_left\_Tree.push\_front(curr\_figure->*copy*());

}

foreach (My\_Figure \*curr\_figure, down\_left\_Tree) {

curr\_figure->*scale*(0.8,centr);

curr\_figure->*turn*(-90, centr);

curr\_figure->*scaleX*(15, centr);

curr\_figure->*reflectX*(centr);

curr\_figure->*moveY*(0.4\*length);

curr\_figure->*moveX*(0.1\*length);

}

foreach (My\_Figure \*curr\_figure, store) {

*//curr\_figure->turn(mainAngle,centr);*

*//* *curr\_figure->scale(1.8,centr);*

curr\_figure->*moveY*(1.0\*length);

}

*//* *My\_Figure* *\** *line\_ptr* *=* *new* *My\_Line(centr,* *QVector2D(centr.x(),* *centr.y()-length),* *store.first()->getColorRGB(),* *store.first()->getAlpha());*

QVector <My\_Figure\*> Tree;

*//* *foreach* *(My\_Figure* *\*curr\_figure,* *store)* *{* *//соединяем* *получившиеся* *три* *части*

*//* *//curr\_figure->setColorRGB(QVector3D((float)196/255,* *(float)0/255,* *(float)0/255));*

*//* *Tree.push\_front(curr\_figure);*

*//* *}*

foreach (My\_Figure \*curr\_figure, left\_Tree) {

*//curr\_figure->setColorRGB(QVector3D((float)196/255,* *(float)36/255,* *(float)250/255));*

Tree.push\_front(curr\_figure);

}

foreach (My\_Figure \*curr\_figure, right\_Tree) {

*//curr\_figure->setColorRGB(QVector3D((float)196/255,* *(float)196/255,* *(float)3/255));*

Tree.push\_front(curr\_figure);

}

foreach (My\_Figure \*curr\_figure, down\_left\_Tree) {

*//curr\_figure->setColorRGB(QVector3D((float)120/255,* *(float)60/255,* *(float)200/255));*

Tree.push\_front(curr\_figure);

}

*//* *foreach* *(My\_Figure* *\*curr\_figure,* *down\_right\_Tree)* *{*

*//* *//curr\_figure->setColorRGB(QVector3D((float)240/255,* *(float)120/255,* *(float)240/255));*

*//* *Tree.push\_front(curr\_figure);*

*//* *}*

*//* *Tree.push\_front(line\_ptr);*

*return* makeFractal(Tree,QVector2D(centr.x(), centr.y()), line\_width,length, mainAngle, leftAngle, rightAngle, N, curr\_n+1);

}

*/\**

*void* *copyTreangle(My\_Triangle* *start,* *QVector2D* *sLine,* *QVector2D* *fLine,* *int* *line\_width,* *int* *N,* *int* *curr\_n)*

*{*

*if(curr\_n* *>=* *N)* *return;*

*My\_Triangle* *finish(* *start.getPoint1()* *-* *(sLine* *-* *fLine)/2,*

*start.getPoint2()* *-* *(sLine* *-* *fLine)/2,*

*start.getPoint3()* *-* *(sLine* *-* *fLine)/2,* *start.getColorRGB(),1* *);*

*}\*/*

*/\*QVector2D* *getLineCenter(QVector2D* *p1,* *QVector2D* *p2)*

*{*

*return* *QVector2D(* *(p1.x()<p2.x()?* *p1.x()+(p2.x()-p1.x())/2* *:* *p1.x()-(p1.x()-p2.x())/2)* *,*

*(p1.y()<p2.y()?* *p1.y()+(p2.y()-p1.y())/2* *:* *p1.y()-(p1.y()-p2.y())/2));;*

*}\*/*

QVector2D **findScale**(QVector2D p1, QVector2D p2,float n) *//* *уменьшает* *в* *сторону* *первой* *точки*

{

*return* QVector2D( (p1.x()<p2.x()? p1.x()+(p2.x()-p1.x())/n : p1.x()-(p1.x()-p2.x())/n) ,

(p1.y()<p2.y()? p1.y()+(p2.y()-p1.y())/n : p1.y()-(p1.y()-p2.y())/n)); *//* *т.к.* *действия* *в* *положительной* *плоскости,* *если* *нужно* *в* *отрицательной,* *то* *есно* *переделать.*

}

*//* *доп* *штуки*

void **makeLabirint**(My\_Figure \*start, int line\_width, int length, int mainAngle, int leftAngle,int rightAngle, int N, int curr\_n)

{

QVector <My\_Figure\*> store;

store.push\_front(start);

My\_Figure \* save =start->*copy*();\

save->*turn*(90);

store.push\_front(save);

foreach (My\_Figure \*curr\_figure, store) {

curr\_figure->*scale*(1.2, QVector2D(50,50));

curr\_figure->*moveTo*(QVector2D(50,50));

}

foreach (My\_Figure \*curr\_figure, makeLabirint(store,QVector2D(50,50), line\_width,length, mainAngle, leftAngle, rightAngle, N, curr\_n)) {

curr\_figure->*draw*(line\_width);

*delete* curr\_figure;

}

}

QVector <My\_Figure\*> **makeLabirint**(QVector <My\_Figure\*> store,QVector2D centr, int line\_width, int length, int mainAngle, int leftAngle,int rightAngle, int N, int curr\_n)

{

*if*(curr\_n >= N) *return* store;

QVector <My\_Figure\*> left\_Tree;

foreach (My\_Figure \*curr\_figure, store) {

*//curr\_figure->turn(mainAngle,centr);*

curr\_figure->*scale*(2, centr);

left\_Tree.push\_front(curr\_figure->*copy*());

}

foreach (My\_Figure \*curr\_figure, left\_Tree) {

curr\_figure->*turn*(90, centr);

*//curr\_figure->scale(2,* *centr);*

}

*/\*QVector* *<My\_Figure\*>* *right\_Tree;*

*foreach* *(My\_Figure* *\*curr\_figure,* *store)* *{*

*right\_Tree.push\_front(curr\_figure->copy());*

*}*

*foreach* *(My\_Figure* *\*curr\_figure,* *right\_Tree)* *{*

*curr\_figure->scale(3,* *centr);*

*curr\_figure->reflectX(centr);*

*curr\_figure->turn(-rightAngle,* *centr);*

*curr\_figure->moveY(-length\*2/3);*

*}\*/*

*//* *My\_Figure* *\** *line\_ptr* *=* *new* *My\_Line(centr,* *QVector2D(centr.x(),* *centr.y()-length),* *store.first()->getColorRGB(),* *store.first()->getAlpha());*

QVector <My\_Figure\*> Tree;

foreach (My\_Figure \*curr\_figure, store) { *//соединяем* *получившиеся* *три* *части*

Tree.push\_front(curr\_figure);

}

foreach (My\_Figure \*curr\_figure, left\_Tree) {

Tree.push\_front(curr\_figure);

}

*/\*foreach* *(My\_Figure* *\*curr\_figure,* *right\_Tree)* *{*

*Tree.push\_front(curr\_figure);*

*}*

*Tree.push\_front(line\_ptr);\*/*

*return* makeLabirint(Tree,QVector2D(centr.x(), centr.y()-length), line\_width,length, mainAngle, leftAngle, rightAngle, N, curr\_n+1);

}