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.

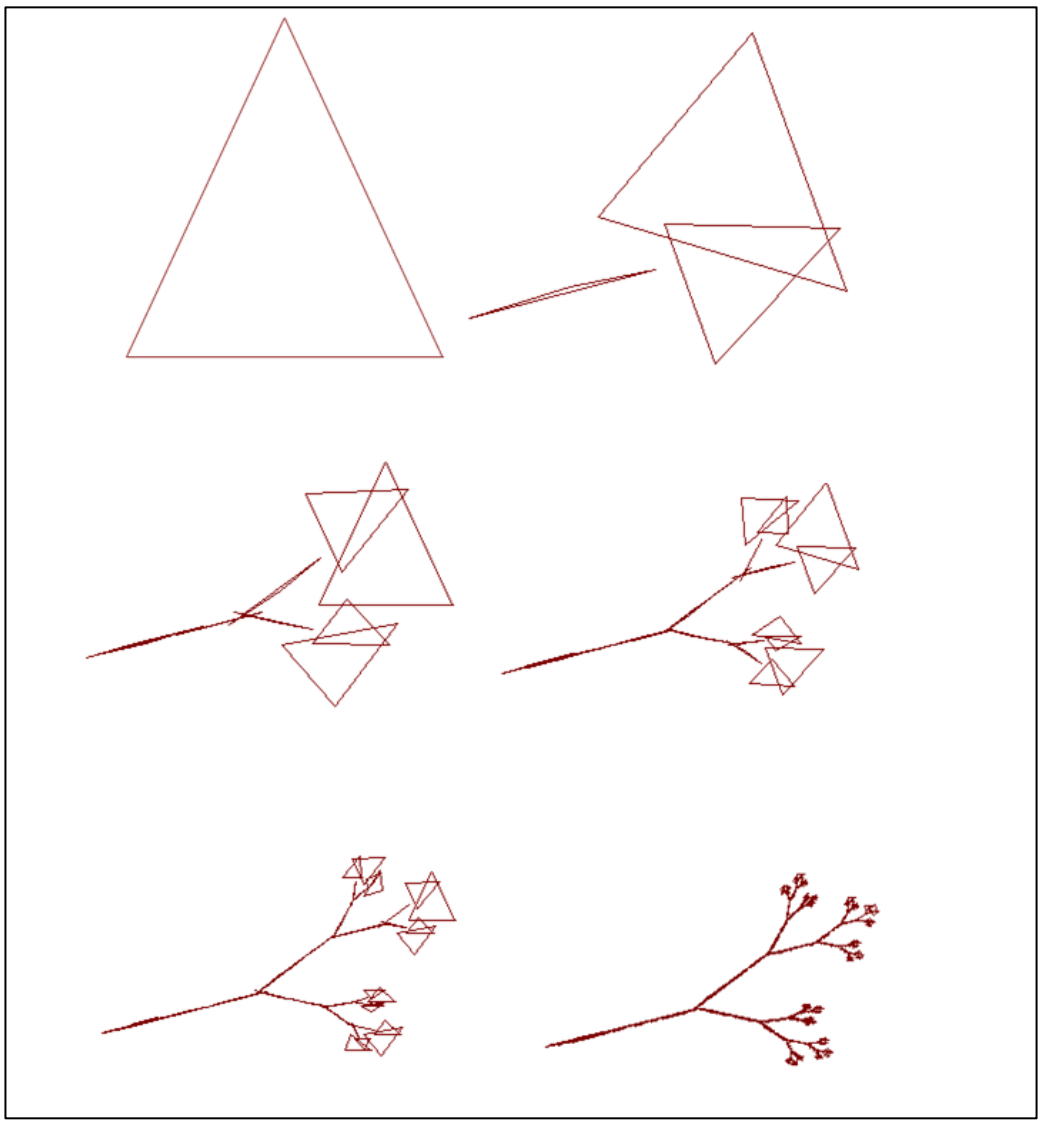
Санкт-Петербург

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 аффинных 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 преобразований, а именно: сжатие в три раза, поворот на - 60^o и параллельный перенос на 1/3 по оси X.

3. Третье звено строится аналогично второму: сжатие в три раза, пово-рот на 60^o, 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.

$f_1(x)$ -> масшт. на r

$f_2(x)$ -> масшт. на r, поворот на 60^0

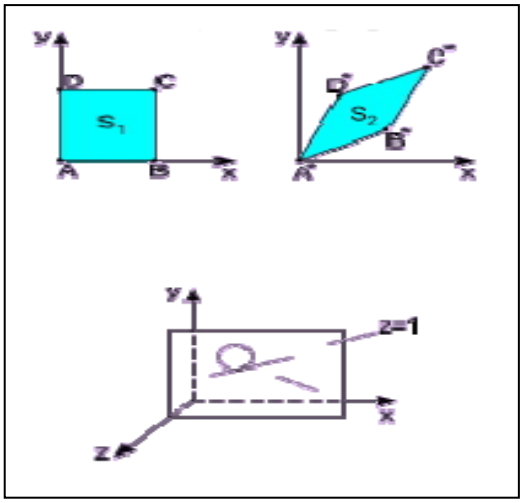
$f_3(x)$ -> масшт. на r, поворот на - 60^0

$f_4(x)$ -> масшт. на r

Преобразования в двухмерном пространстве:

Деформация фигуры:

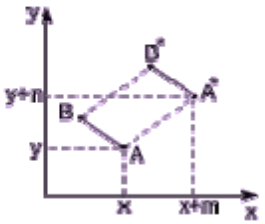
$$\begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ x_4 & y_4 \end{vmatrix} \cdot \begin{vmatrix} a & b \\ c & d \end{vmatrix} = \begin{vmatrix} x_1 a + y_1 c & x_1 b + y_1 d \\ x_2 a + y_2 c & x_2 b + y_2 d \\ x_3 a + y_3 c & x_3 b + y_3 d \\ x_4 a + y_4 c & x_4 b + y_4 d \end{vmatrix} = \begin{vmatrix} x_1^* & y_1^* \\ x_2^* & y_2^* \\ x_3^* & y_3^* \\ x_4^* & y_4^* \end{vmatrix}$$



Смещение фигуры:

$$\begin{vmatrix} x & y & 1 \end{vmatrix} \cdot \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ m & n & 1 \end{vmatrix} = \begin{vmatrix} x+m & y+n & 1 \end{vmatrix}.$$

Операция смещения



Операция масштабирования

$$\begin{vmatrix} x & y & 1 \end{vmatrix} \cdot \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & s \end{vmatrix} = \begin{vmatrix} x & y & s \end{vmatrix} = \begin{vmatrix} \frac{x}{s} & \frac{y}{s} & 1 \end{vmatrix}.$$

Поворот точки (x, y) на угол α

$$\begin{vmatrix} x & y & 1 \end{vmatrix} \cdot \begin{vmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} x\cos\alpha - y\sin\alpha & x\sin\alpha + y\cos\alpha & 1 \end{vmatrix}.$$

Поворот фигуры вокруг произвольной точки (m, n) на произвольный угол α

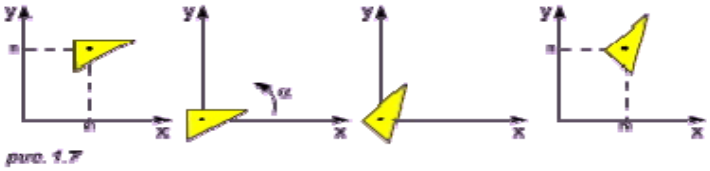


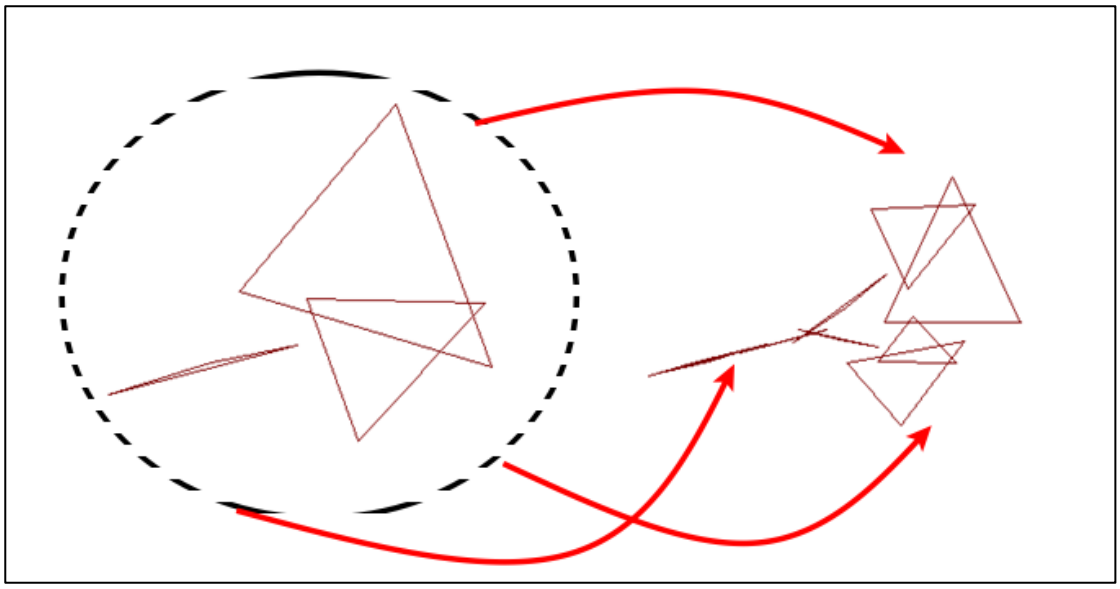
Рис.3.4

$$\underset{M_1}{\begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -m & -n & 1 \end{vmatrix}} \cdot \underset{M_2}{\begin{vmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{vmatrix}} \cdot \underset{M_3}{\begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ m & n & 1 \end{vmatrix}} =$$

$$= \begin{vmatrix} x & y & 1 \end{vmatrix} \cdot \begin{vmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ -m(\cos\alpha - 1) + n\sin\alpha & -n(\cos\alpha - 1) - m\sin\alpha & 1 \end{vmatrix}$$

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:

```
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());
```

Тестирование программы.

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

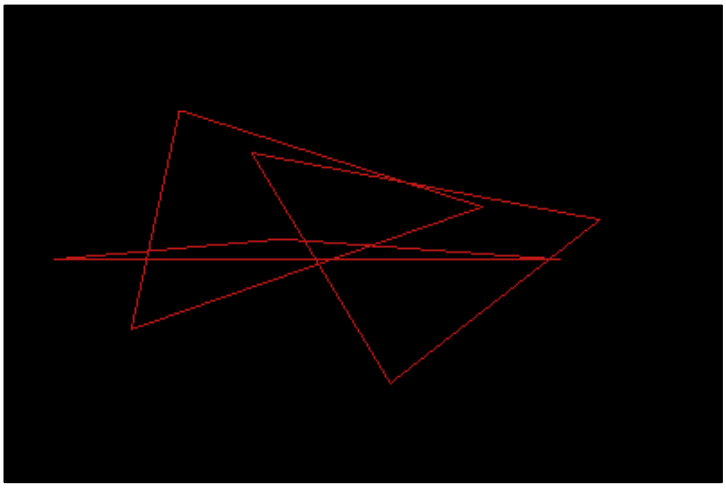


Рисунок 2 – “ветка” 1-го уровня

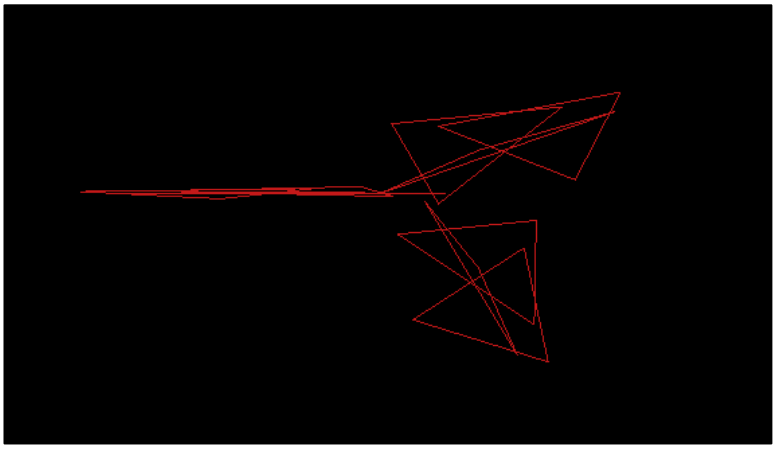


Рисунок 3 – “ветка” 2-го уровня

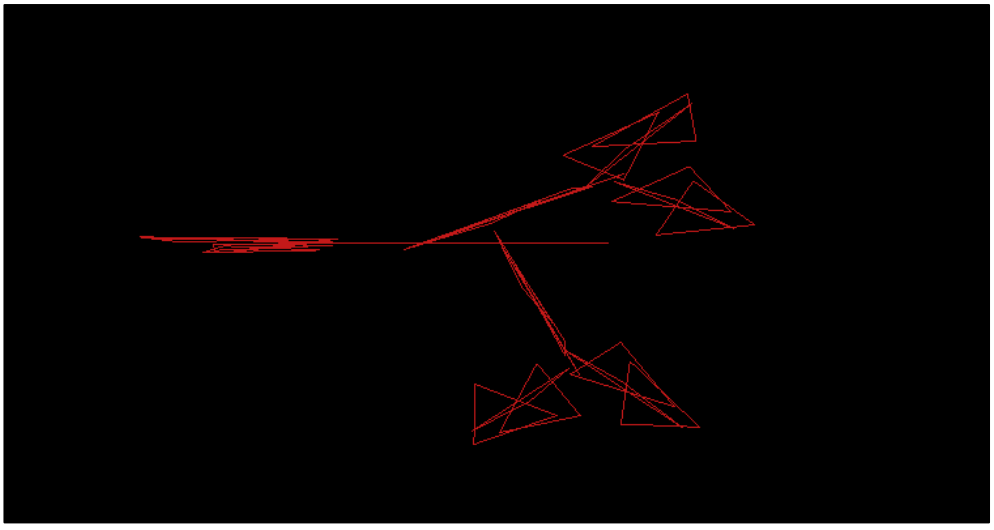


Рисунок 4 – “ветка” 3-го уровня

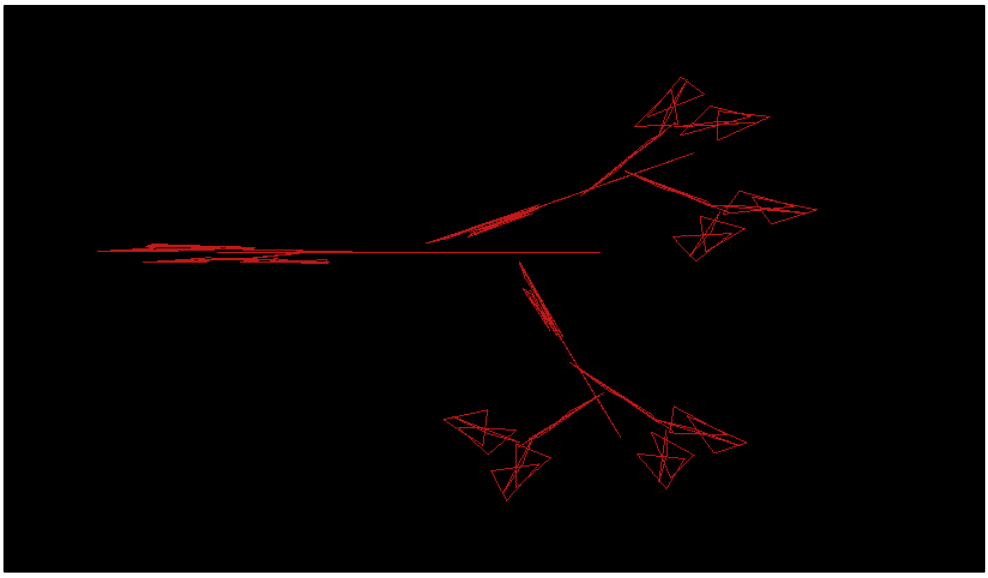


Рисунок 5 – “ветка” 4-го уровня

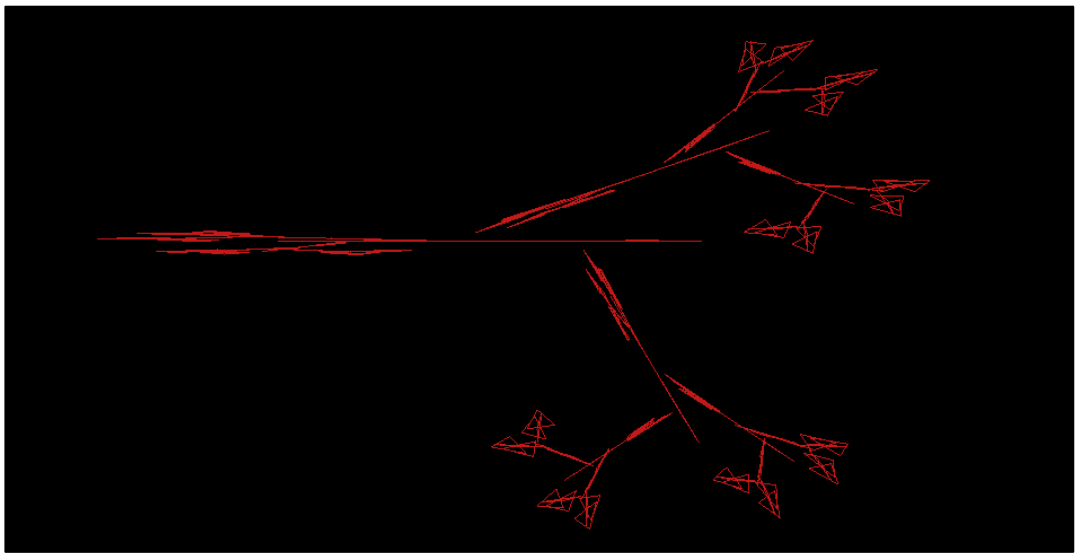


Рисунок 6 – “ветка” 5-го уровня

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);

}
```