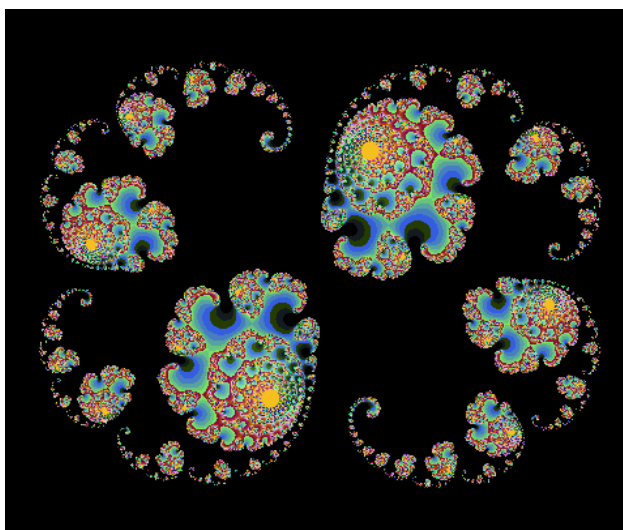


分形与混沌



1.思考三个问题:	3
2.绘制 3D 效果.....	5
3.迭代分形.....	5
4.多线程绘图.....	8
5.双缓冲动画.....	8
6.界面小组件的使用.....	10
7.递归	10
8.四步实现门格海绵.....	12
第一步: 画出一个立方体.....	12
第二步: 填充 Color 更具立体感	13
第三步: 绘制多个立方体形状, 如图:	15
第四步: 门格海绵的三部分组成.....	15
9.科赫曲线系列.....	22
10.分形树	23
11.L-System	25
12.梅塔特隆立方体.....	26
13.光线追踪	28
14.经典之作:曼德勃罗集.....	28
15.自然是随机的吗?	34
12.学长的 L-system	35
要看到现象的本质.....	39
混沌之迷.....	41
永恒之心.....	42
指定阅读书籍:	43

作品呈现：

A：创意图形集：绘制 5~6 种漂亮图形(菜单+JFrame)

A+：最美分形：在 A 的基础上，加上拉杆、输入框、按钮等调节参数变化。

A++：加上多线程实现的动画效果

重点文章：

多线程、队列、链表

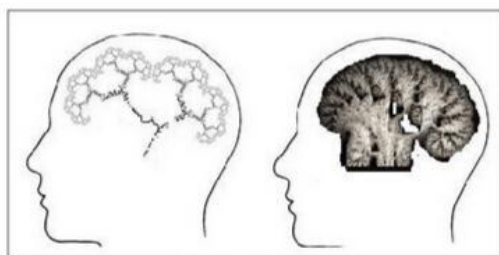
分形理论

混沌理论

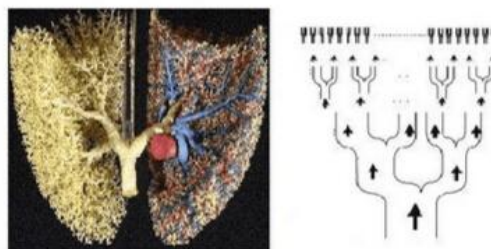
1. 思考三个问题：

1. 我们所谓“自然”，到底是什么意思
2. 美，是否有规律？
3. DNA 是否一段程序？
4. 阅读 <https://en.wikipedia.org/wiki/Fractal>

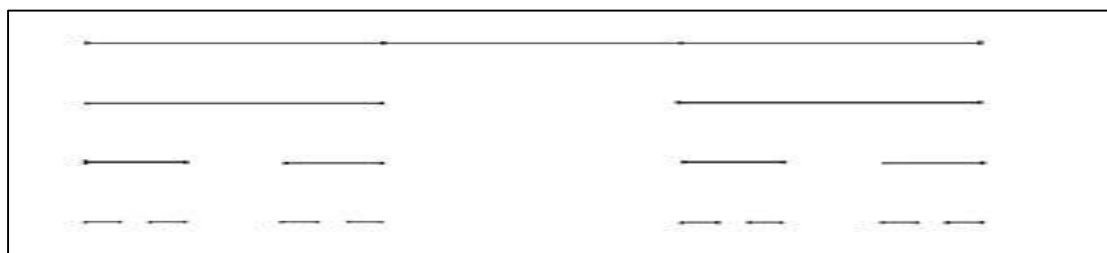
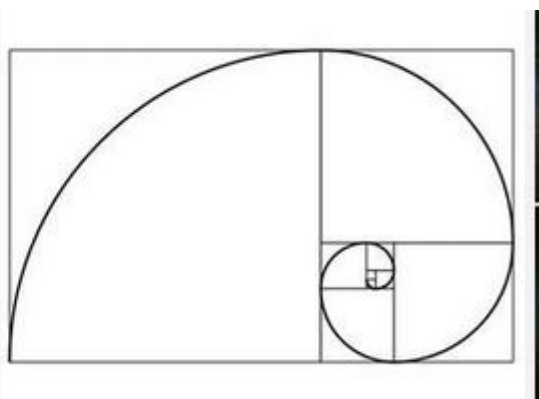
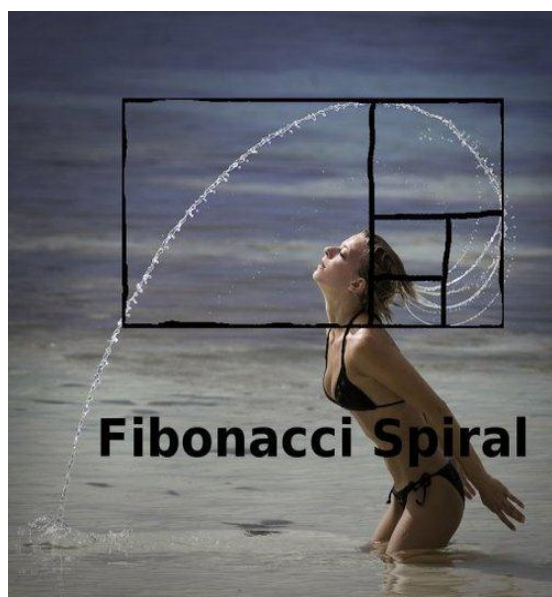




(a) 人脑的分形模型



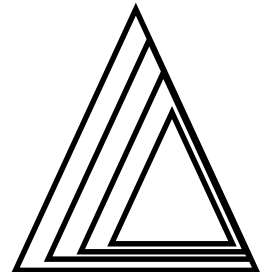
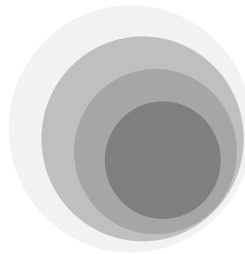
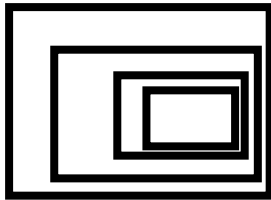
(b) 肺动脉床的分形模型



2. 绘制 3D 效果

用代码实现如下效果，理解“3 维=3 维的感觉”

思考问题：感觉就是真实的么？

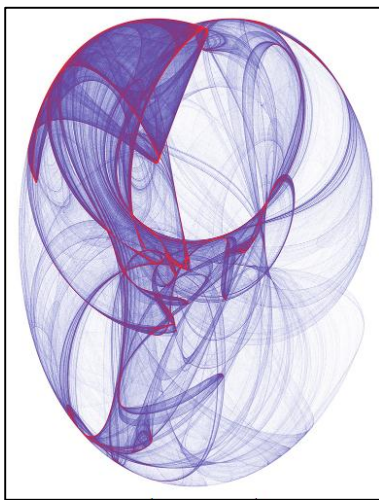


3. 迭代分形

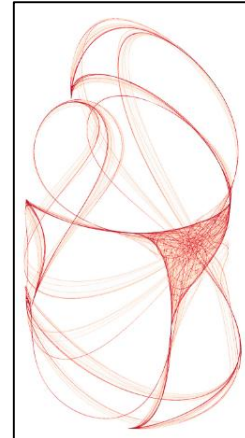
第一组公式：

$$\begin{aligned} \text{Definition} \\ x_{n+1} &= \sin(a y_n) + c \cos(a x_n) \\ y_{n+1} &= \sin(b x_n) + d \cos(b y_n) \end{aligned}$$

$$\begin{aligned} a &= -1.8, b = -2.0, \\ c &= -0.5, d = -0.9 \end{aligned}$$



$$\begin{aligned} a &= -1.7, b = 1.3, \\ c &= -0.1, d = -1.2 \end{aligned}$$



如下代码示例绘制一个图形：

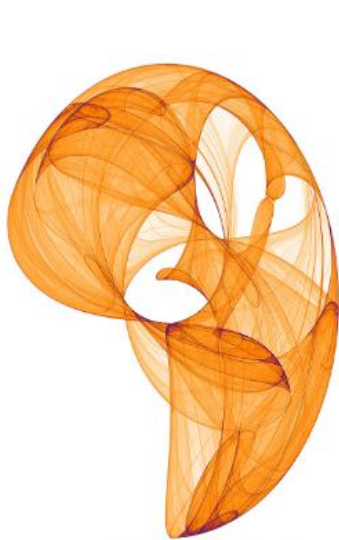
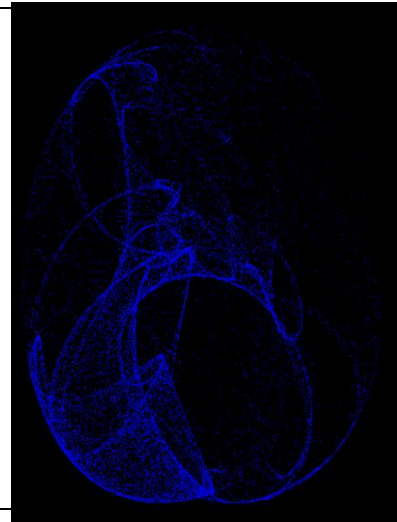
```
private void drawDream(Graphics g){
    double x=0f;
    double y=0f;
    //a,b,c,d等4个常量的值预设
    double a=-1.8, b=-2.0, c=-0.5, d=-0.9;
    for(int i=0;i<25500;i++){
        //公式:
        double temx= Math.sin(a*y)+c*Math.cos(a*x);
```



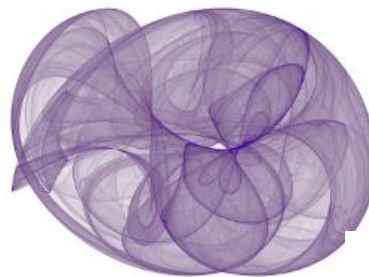
```

double temy=Math.sin(b*x)+d*Math.cos(b*y);
//对x1,y1转型，放大，移动到屏幕坐标系:
int x1= (int)(temx*130+500);
int y1= (int)(temy*130+400);
System.out.println("x1: "+x1+" y1: "+y1);
//颜色根据迭代次数加深
g.setColor(new Color(0,0,i/100));
g.drawLine(x1, y1, x1,y1);
x=temx;
y=temy;
}
}

```



$a = 1.7, b = 1.7, c = 0.6, d = 1.2$



$a = 1.5, b = -1.8, c = 1.6, d = 0.9$



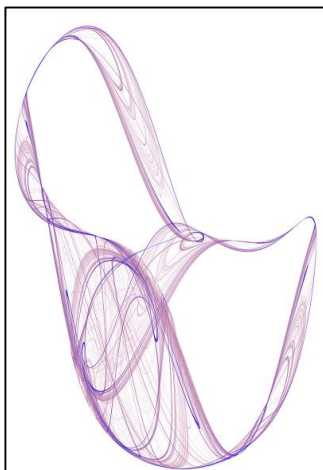
$a = -1.7, b = 1.8, c = -1.9, d = -0.4$

第二组公式:

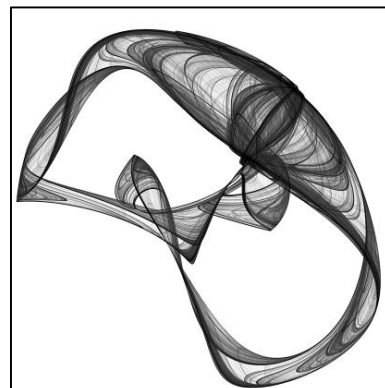
快速度上手，计算 x, y 的公式如下:

$$\begin{aligned} x_{n+1} &= \sin(a y_n) - \cos(b x_n) \\ y_{n+1} &= \sin(c x_n) - \cos(d y_n) \end{aligned}$$

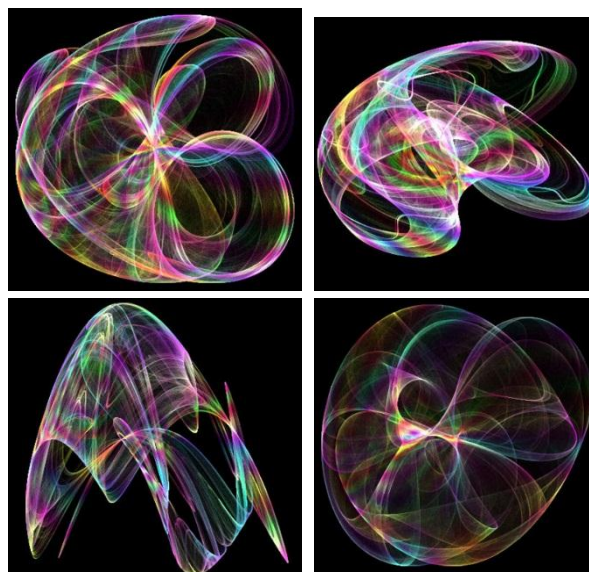
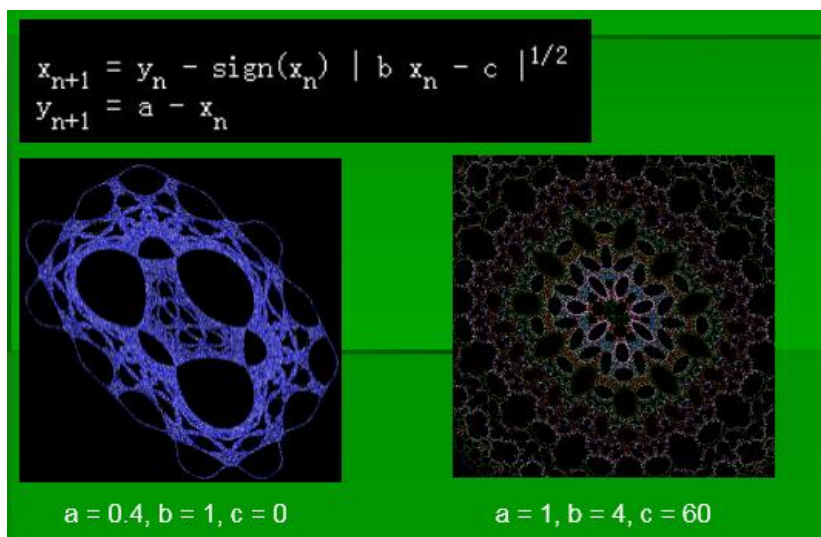
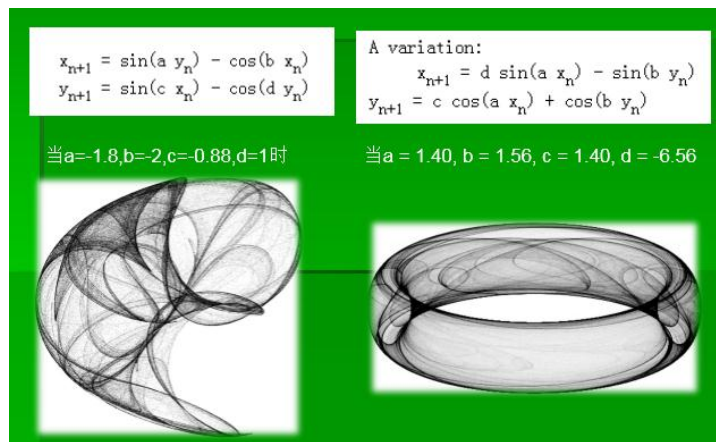
$$\begin{aligned} a &= 1.641, b = 1.902, \\ c &= 0.316, d = 1.525 \end{aligned}$$



$$\begin{aligned} a &= 0.970, b = -1.899, \\ c &= 1.381, d = -1.506 \end{aligned}$$



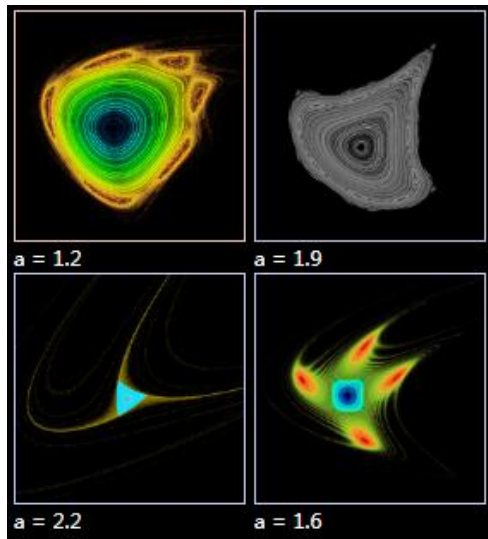
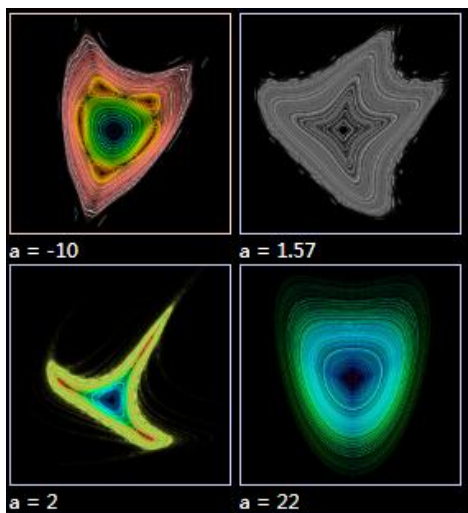
第三组：



第四组：

$$x_{n+1} = x_n \cos(a) - (y_n - x_n^2) \sin(a)$$

$$y_{n+1} = x_n \sin(a) + (y_n - x_n^2) \cos(a)$$



这里有所有你想要的：

<http://paulbourke.net/fractals/>

4. 多线程绘图

第一步：实现用 Mouse 点一下，画个小球(不能动)

第二步：用 for 循环让小球动起来，每间隔 100ms(能动，但只能动一个)

第三步：把小球的移动放到线程中去画~多个小球=多个线程

第四步：线程创建：extends Thread

第五步：线程控制：设置标志，在 run 方法里不作工

- 1.每个移动的物件，都是一个线程，并行
- 2.A 线程绘制+(装物件的队列)+B 线程移动

5. 双缓冲动画

动画效果=多图+快动=动画的感觉

1.用多线程

2.直接画，多了会闪烁

3.用双缓冲：在内存中画好（批量发给显卡），再显示

如下代码示例，实现双缓冲绘制一个不断放大图形的效果

红色代码为使用双缓冲的流程关键

```
//绘图线程
public class DreamControl extends Thread{
    public JFrame jf;//界面对象

    public DreamControl(JFrame jf){
        this.jf=jf;
    }

    public void run(){
        Graphics g=jf.getGraphics();
        drawDream(g);
    }

    private void drawDream(Graphics g){
        double x=0f;
        double y=0f;
        //a,b,c,d等4个常量的值预设
        double a= -1.8, b = -2.0, c=-0.5, d=-0.9;
        for(int j=0;j<180;j++){
            //1.创建一张和界面同样大小的缓冲区图像
            Image image = jf.createImage(jf.getWidth(), jf.getHeight());
            Graphics gBuffer = image.getGraphics();
            gBuffer.setColor(Color.BLACK);
            gBuffer.fillRect(0, 0, jf.getWidth(), jf.getHeight());

            for(int i=0;i<j*500;i++){
                //公式:
                double temx= Math.sin(a*y)+c*Math.cos(a*x);
                double temy=Math.sin(b*x)+d*Math.cos(b*y);
                //对x1,y1转型，放大，移动到屏幕坐标系:
                int x1= (int)(temx*j+600);
                int y1= (int)(temy*j+400);
                gBuffer.setColor(new Color(0,0,255));
                //2.画到缓冲区
                gBuffer.drawLine(x1, y1, x1,y1);
                x=temx;
                y=temy;
            }
            //3.将缓冲区的图像画到界面上
            jf.getGraphics().drawImage(image, 100, 100,800,600, null);
        }
    }
}
```

```
try{
    Thread.sleep(10);
}catch(Exception ef){ef.printStackTrace();}
}
}
```

6. 界面小组件的使用

菜单、滑块。。。

配合控制，

目标：实现一个展示创意想像力，

展示出令人惊叹效果的绘制效果软件

7. 递归

练习：

1，累加 $1+2+3+4+n$

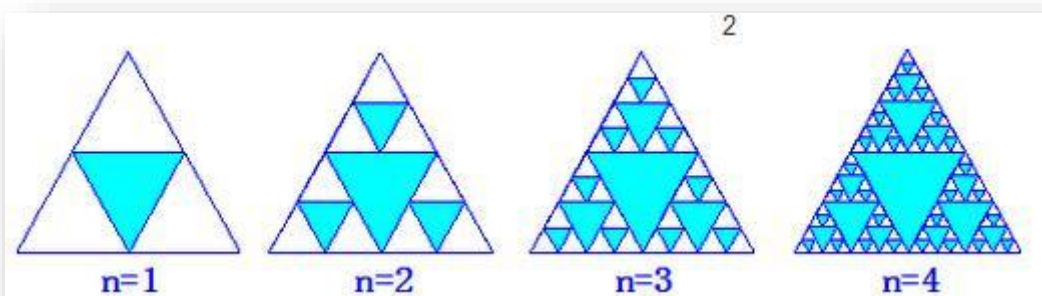
2.乘法表

3.FB 数序列

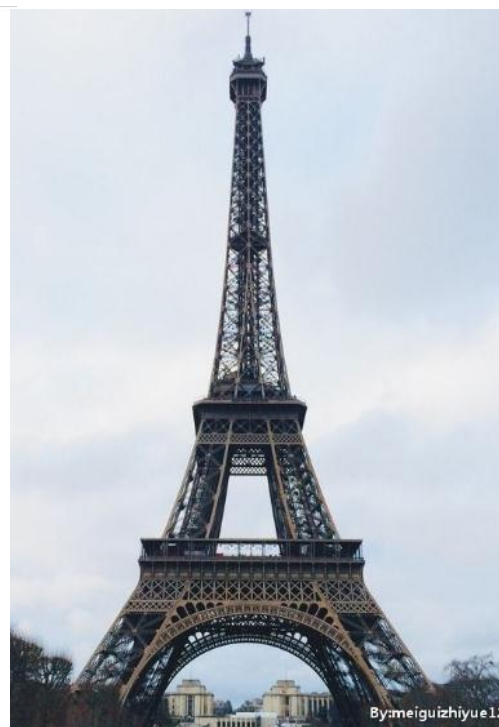
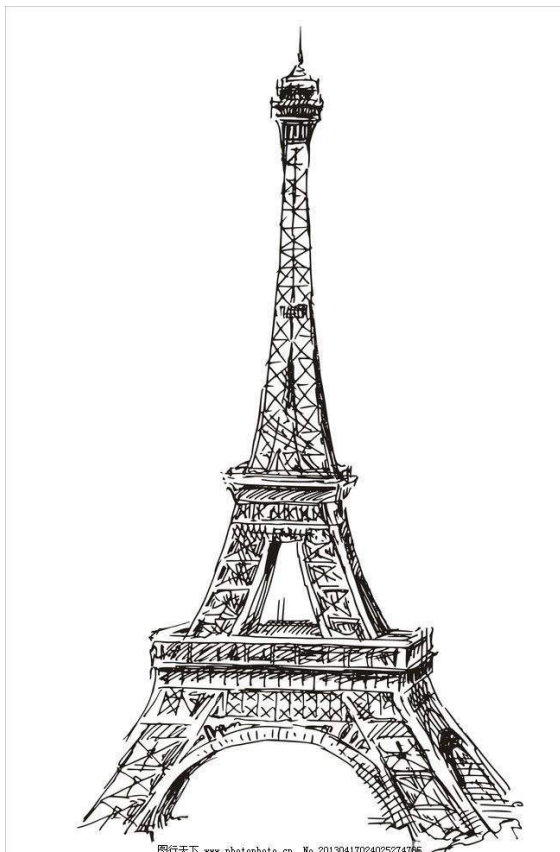
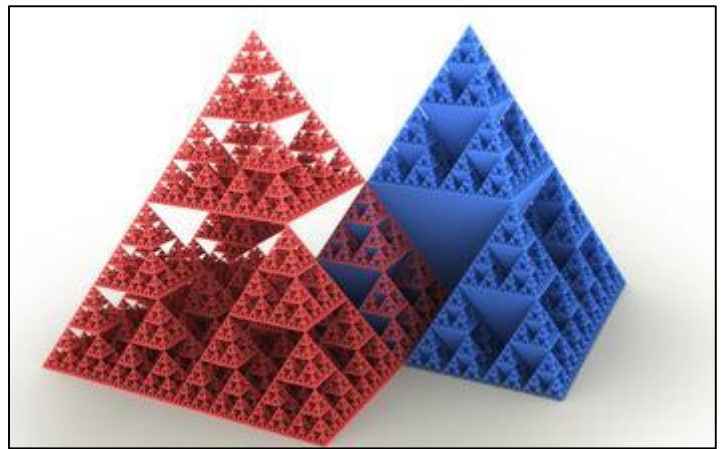
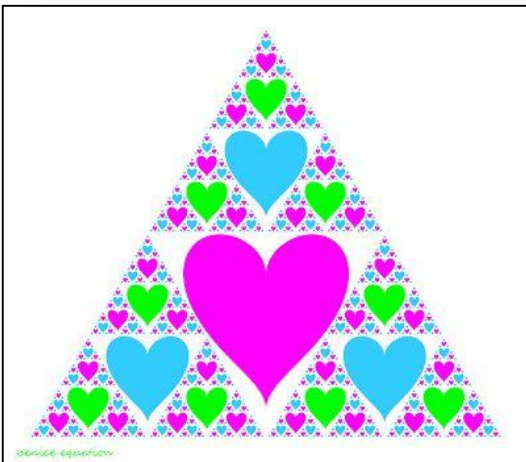
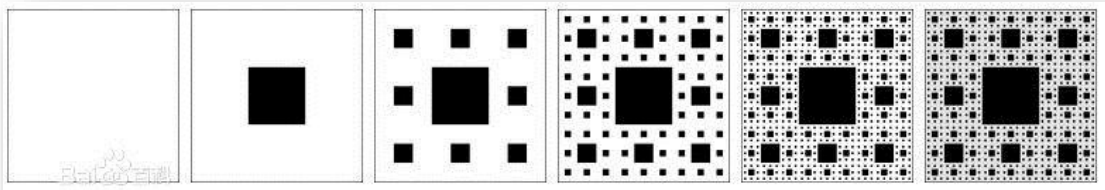
注意要点：

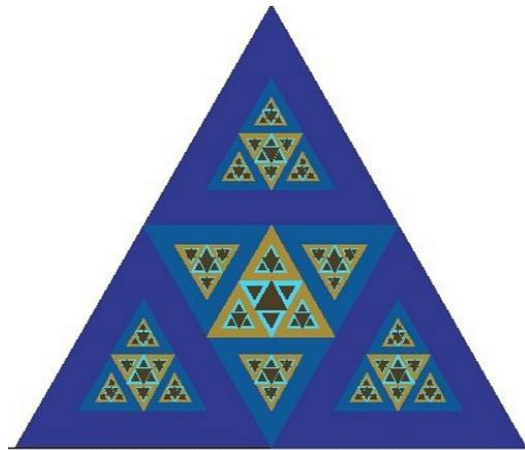
- 1.退出条件的控制
- 2.递归时，变量的传递和值的变化
- 3.进入时的参数值变化和返回时返回值的变化

Sierpinski 三角形:

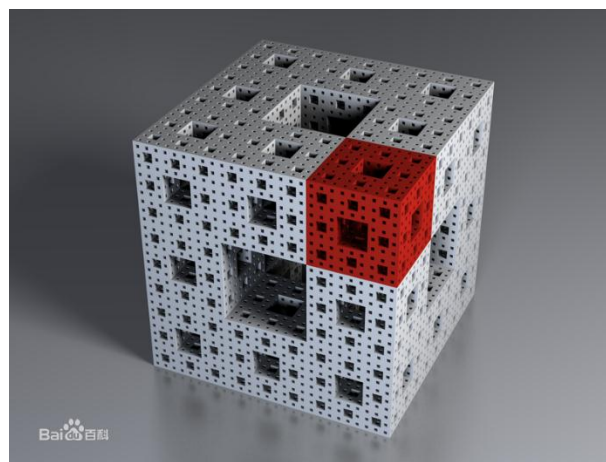


<http://baike.baidu.com/item/%E8%B0%A2%E5%B0%94%E5%AE%BE%E6%96%AF%E5%9F%BA%E4%B8%89%E8%A7%92%E5%BD%A2>

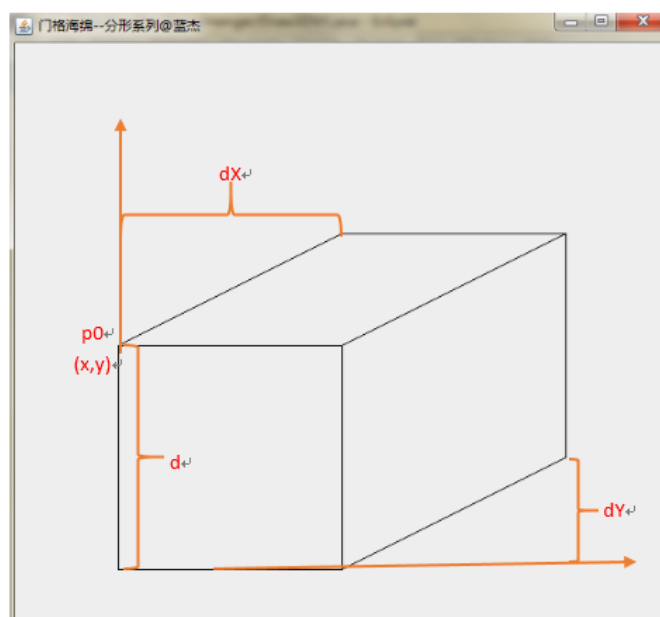




8. 四步实现门格海绵



第一步：画出一个立方体



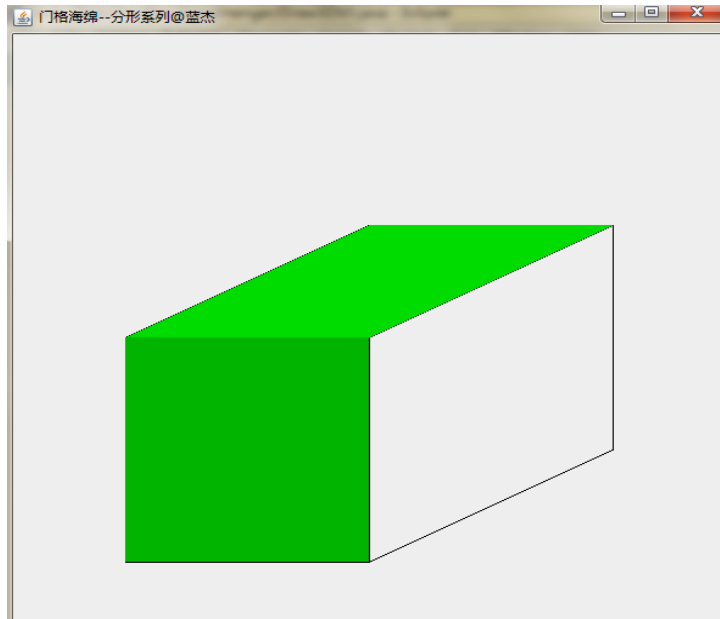
如上图示，给定立方体的顶点 $p_0(x,y)$ ，定义常量立方体的高度 d 、和透视偏移量 dx ， dy ，即可画出一个立方体。

```
//重写界面的paint方法
public void paint(Graphics g){
    super.paint(g);
    int x=300,y=300; //正方体的顶点坐标
    int d=200,dx=200,dy=100; //正方体的高,宽深度
    //画出来
    Point p0=new Point(x,y);
    draw(g,p0.x,p0.y,d,dx,dy,0);
}

private void draw(Graphics g,int x,int y,int d,int dx,int dy){
    //6个顶点位置
    Point p0=new Point(x,y);
    Point p1=new Point(x-d,y);
    Point p2=new Point(x-d,y+d);
    Point p3=new Point(x,y+d);
    Point p4=new Point(p3.x+dx,p3.y-dy);
    Point p5=new Point(p0.x+dx,p0.y-dy);
    Point p6=new Point(p0.x-(d-dx),p0.y-dy);
    //9条线
    g.drawLine(p0.x, p0.y, p1.x, p1.y);
    g.drawLine(p1.x, p1.y, p2.x, p2.y);
    g.drawLine(p2.x, p2.y, p3.x, p3.y);
    g.drawLine(p3.x, p3.y, p0.x, p0.y);
    g.drawLine(p1.x, p1.y, p6.x, p6.y);
    g.drawLine(p6.x, p6.y, p5.x, p5.y);
    g.drawLine(p0.x, p0.y, p5.x, p5.y);
    g.drawLine(p5.x, p5.y, p4.x, p4.y);
    g.drawLine(p4.x, p4.y, p3.x, p3.y);
}
```

第二步：填充 Color 更具立体感

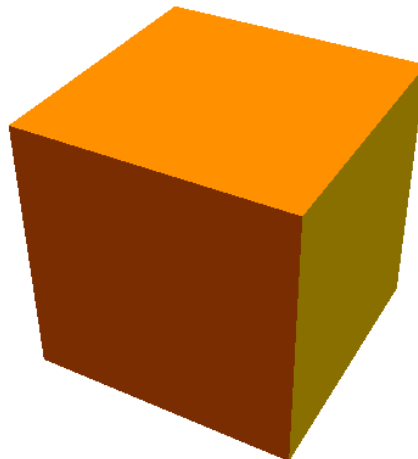
每个面用不同深度的 Color 填充，则更有立体感，如图：



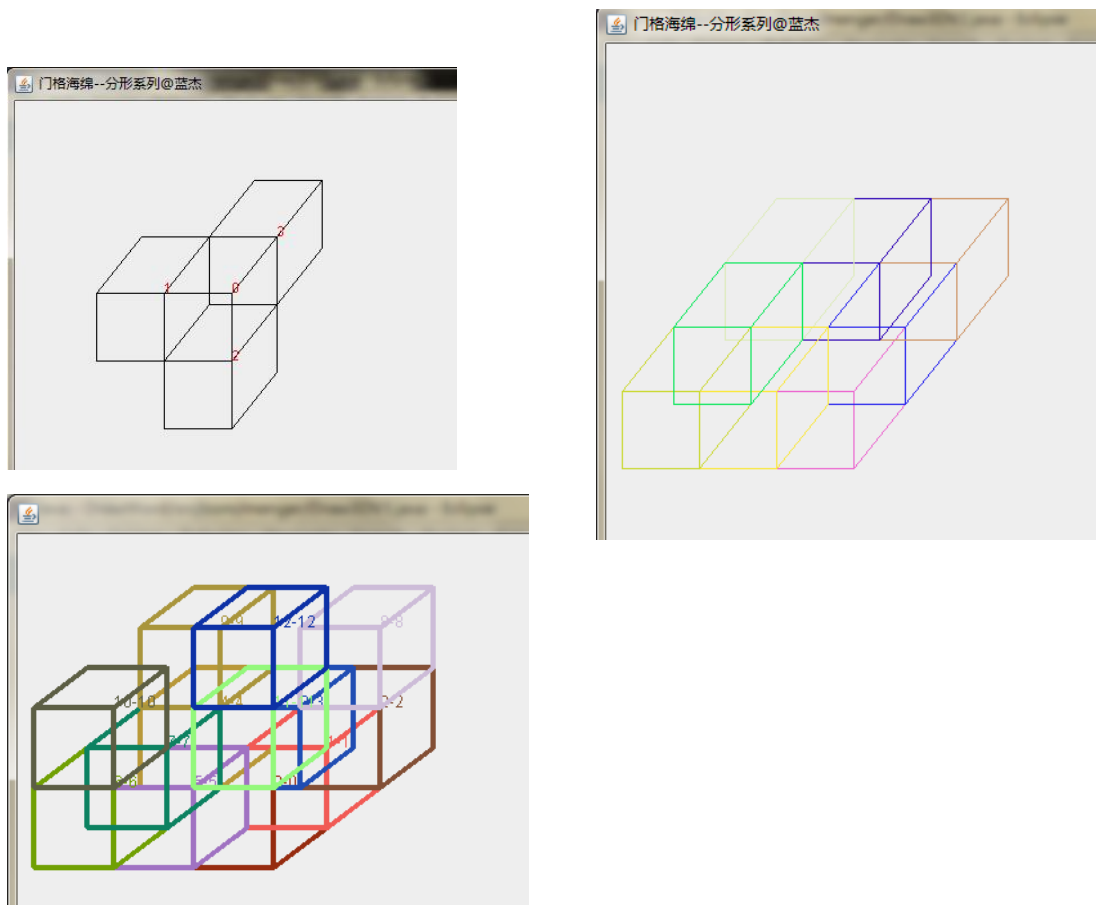
填充时，用 `java.awt.Polygon` 类，代码如下

```
//填充第一个面
Polygon pon1=new Polygon();
pon1.addPoint(p0.x,p0.y);
pon1.addPoint(p1.x,p1.y);
pon1.addPoint(p2.x,p2.y);
pon1.addPoint(p3.x,p3.y);
g.setColor(new Color(0,150,0));
g.fillPolygon(pon1);
//填充第二个面
Polygon pon2=new Polygon();
pon2.addPoint(p0.x,p0.y);
pon2.addPoint(p1.x,p1.y);
pon2.addPoint(p6.x,p6.y);
pon2.addPoint(p5.x,p5.y);
g.setColor(new Color(0,250,0));
g.fillPolygon(pon2);
```

考虑如何绘制如下图立方体：

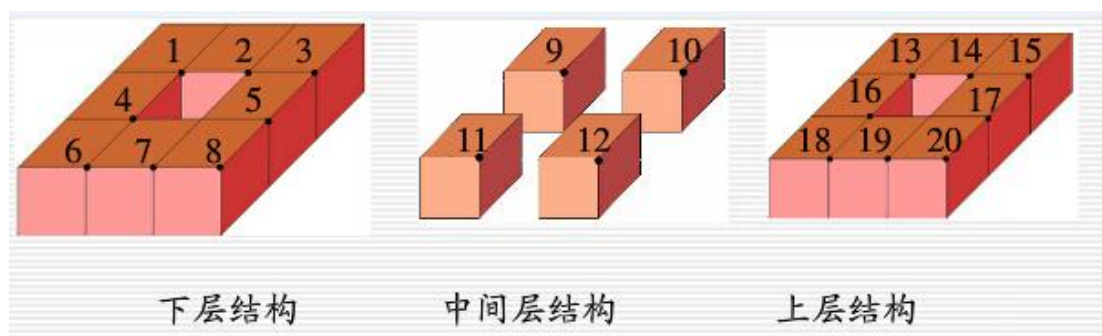


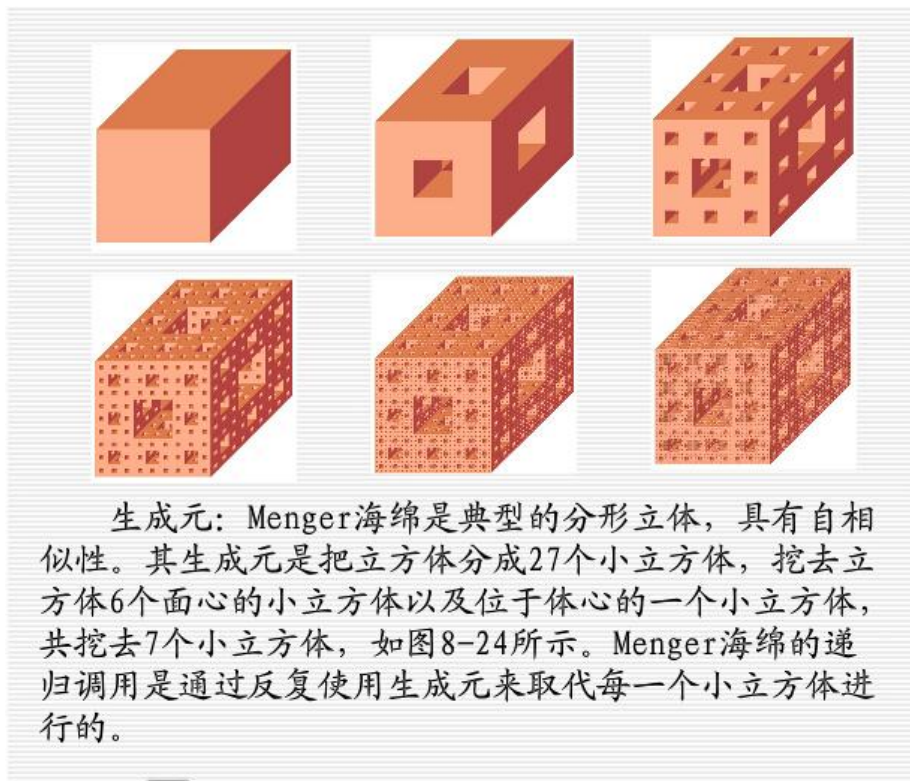
第三步：绘制多个立方体形状，如图：



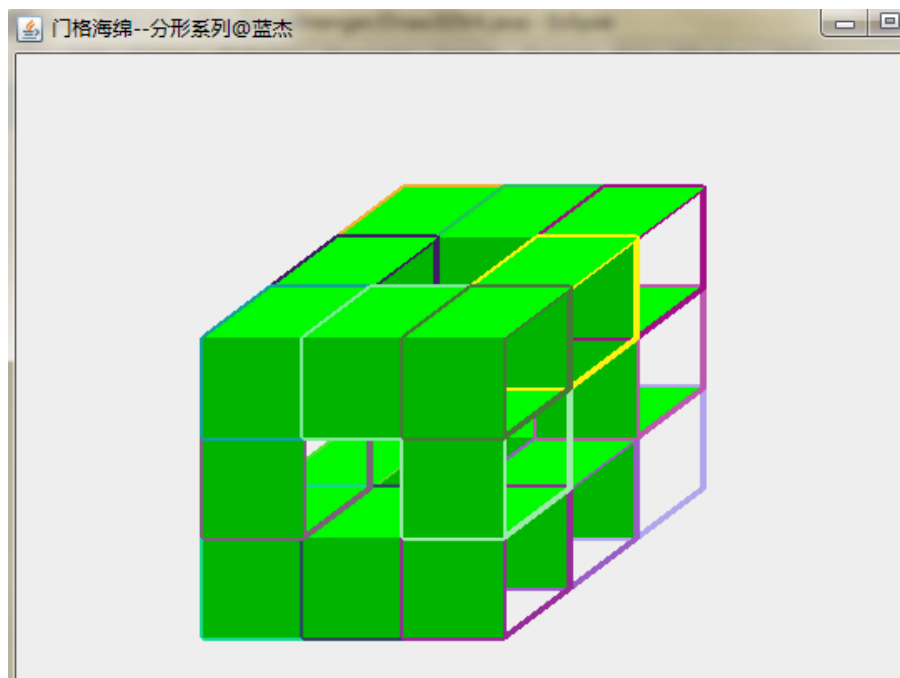
第四步：门格海绵的三部分组成

每一个门格海绵的单元由三部分组成：



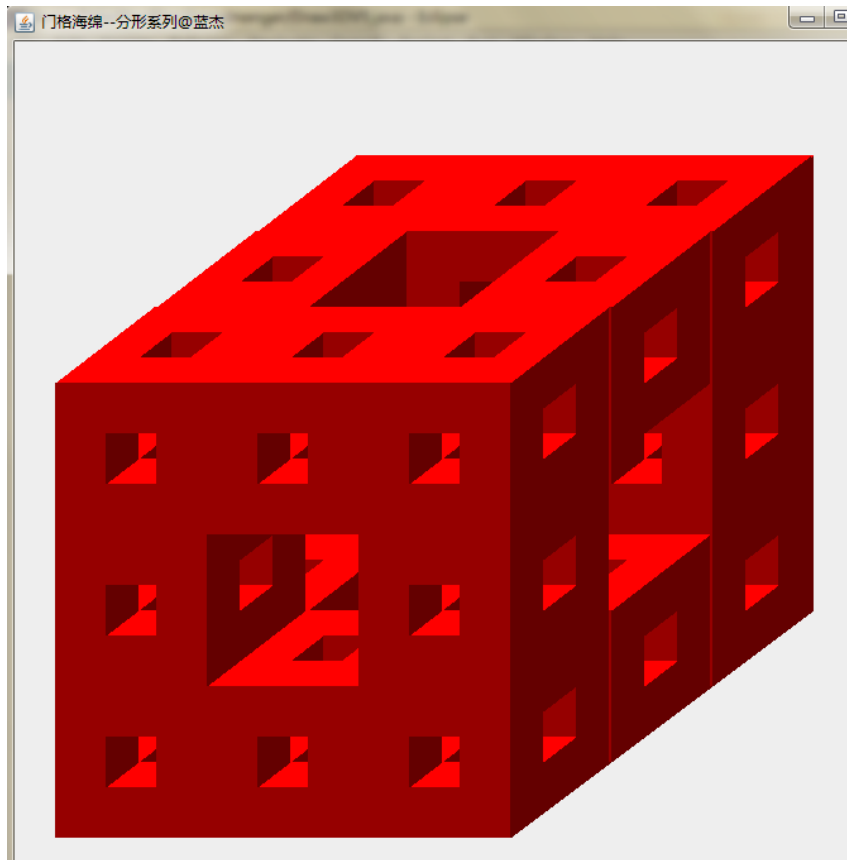


如下图所示：我们分三层，画出每一层的结构：



然后，对每一单元块，做递归变化，即可生成图形：

最后，实现我们的成品，如下图示：



上图中，一共有多少个小立方体？如果再迭代一次，是多少个？

```
//门格海绵绘制
public class DrawMGV5 extends JFrame{

    public void initUI(){
        this.setSize(600,800);
        this.setVisible(true);
        this.setTitle("门格海绵--分形系列@蓝杰");
        this.setDefaultCloseOperation(3);
    }

    public void paint(Graphics g){
        super.paint(g);
        int x=400,y=300,d=120,dx=80,dy=60;
        Point p0=new Point(x,y);//起点

        drawButtonLevel(g,p0,d,dx,dy);//画底层
```

```

        drawMidLevel(g,p0,d,dx,dy);//画中层
        drawTopLevel(g,p0,d,dx,dy);//画顶层
        for(int i=0;i<pn.length;i++){
            Point ptem=pn[i];
            if(ptem!=null){
                drawButtonLevel(g,ptem,d/3,dx/3,dy/3);//画底层
                drawMidLevel(g,ptem,d/3,dx/3,dy/3);//画中层
                drawTopLevel(g,ptem,d/3,dx/3,dy/3);//画顶层
            }
        }
    }
    //保存下一次的
    Point[] pn=new Point[20];//起点
    private void drawMidLevel(Graphics g,Point p0,int d,int dx,int dy){
        Point[] ps1=getPointByP0(p0,d,dx,dy);//根据起点得到另外6个点
        Point[] ps2=getPointByP0(ps1[3],d,dx,dy);//这个点放到最后画

        Point[] ps3=getPointByP0(ps2[5],d,dx,dy);//这个不需要画出来
        Point[] ps4=getPointByP0(ps3[5],d,dx,dy);
        draw(g,ps4[0],ps4[1],ps4[2],ps4[3],ps4[4],ps4[5],ps4[6]);

        Point[] ps5=getPointByP0(ps4[1],d,dx,dy);//这个不需要画出来
        Point[] ps6=getPointByP0(ps5[1],d,dx,dy);//这个不需要画出来
        draw(g,ps6[0],ps6[1],ps6[2],ps6[3],ps6[4],ps6[5],ps6[6]);
        Point[] ps7=getPointByP0(ps2[1],d,dx,dy);//这个不需要画出来
        Point[] ps8=getPointByP0(ps7[1],d,dx,dy);//这个不需要画出来
        draw(g,ps8[0],ps8[1],ps8[2],ps8[3],ps8[4],ps8[5],ps8[6]);
        draw(g,ps2[0],ps2[1],ps2[2],ps2[3],ps2[4],ps2[5],ps2[6]); }

    private void drawTopLevel(Graphics g,Point p0,int d,int dx,int dy){
        Point[] ps1=getPointByP0(p0,d,dx,dy);//根据起点得到另外6个点
        Point[] ps2=getPointByP0(ps1[5],d,dx,dy);//根据起点得到另外6个点

        Point[] ps3=getPointByP0(ps2[5],d,dx,dy);//根据起点得到另外6个点
        Point[] ps4=getPointByP0(ps3[1],d,dx,dy);//根据起点得到另外6个点
        Point[] ps5=getPointByP0(ps4[1],d,dx,dy);//根据起点得到另外6个点
        Point[] ps7=getPointByP0(ps1[1],d,dx,dy);//根据起点得到另外6个点
        Point[] ps8=getPointByP0(ps7[1],d,dx,dy);//根据起点得到另外6个点
        Point[] ps9=getPointByP0(ps8[5],d,dx,dy);//根据起点得到另外6个点

        //画出第一个个立方体
        draw(g,ps5[0],ps5[1],ps5[2],ps5[3],ps5[4],ps5[5],ps5[6]);
    }

```

```

        draw(g,ps4[0],ps4[1],ps4[2],ps4[3],ps4[4],ps4[5],ps4[6]);
        draw(g,ps9[0],ps9[1],ps9[2],ps9[3],ps9[4],ps9[5],ps9[6]);
        draw(g,ps8[0],ps8[1],ps8[2],ps8[3],ps8[4],ps8[5],ps8[6]);
        draw(g,ps7[0],ps7[1],ps7[2],ps7[3],ps7[4],ps7[5],ps7[6]);
        draw(g,ps3[0],ps3[1],ps3[2],ps3[3],ps3[4],ps3[5],ps3[6]);
        draw(g,ps2[0],ps2[1],ps2[2],ps2[3],ps2[4],ps2[5],ps2[6]);
        draw(g,ps1[0],ps1[1],ps1[2],ps1[3],ps1[4],ps1[5],ps1[6]); }

    private void drawButtonLevel(Graphics g,Point p0,int d,int dx,int
dy){

        Point[] ps1=getPointByP0(p0,d,dx,dy);//根据起点得到另外6个点
        ps1=getPointByP0(ps1[3],d,dx,dy);//根据起点得到另外6个点
        ps1=getPointByP0(ps1[3],d,dx,dy);//根据起点得到另外6个点

        Point[] ps2=getPointByP0(ps1[5],d,dx,dy);//根据起点得到另外6个点

        Point[] ps3=getPointByP0(ps2[5],d,dx,dy);//根据起点得到另外6个点
        Point[] ps4=getPointByP0(ps3[1],d,dx,dy);//根据起点得到另外6个点
        Point[] ps5=getPointByP0(ps4[1],d,dx,dy);//根据起点得到另外6个点
        Point[] ps7=getPointByP0(ps1[1],d,dx,dy);//根据起点得到另外6个点
        Point[] ps8=getPointByP0(ps7[1],d,dx,dy);//根据起点得到另外6个点
        Point[] ps9=getPointByP0(ps8[5],d,dx,dy);//根据起点得到另外6个点

        draw(g,ps5[0],ps5[1],ps5[2],ps5[3],ps5[4],ps5[5],ps5[6]);
        draw(g,ps4[0],ps4[1],ps4[2],ps4[3],ps4[4],ps4[5],ps4[6]);
        draw(g,ps9[0],ps9[1],ps9[2],ps9[3],ps9[4],ps9[5],ps9[6]);
        draw(g,ps8[0],ps8[1],ps8[2],ps8[3],ps8[4],ps8[5],ps8[6]);
        draw(g,ps7[0],ps7[1],ps7[2],ps7[3],ps7[4],ps7[5],ps7[6]);
        draw(g,ps3[0],ps3[1],ps3[2],ps3[3],ps3[4],ps3[5],ps3[6]);
        draw(g,ps2[0],ps2[1],ps2[2],ps2[3],ps2[4],ps2[5],ps2[6]);
        draw(g,ps1[0],ps1[1],ps1[2],ps1[3],ps1[4],ps1[5],ps1[6]);
    }

    //根据0号点，得到另外几个点的坐标
    private Point[] getPointByP0(Point p0,int d,int dx,int dy){
        Point p1=new Point(p0.x-d,p0.y);
        Point p2=new Point(p0.x-d,p0.y+d);
        Point p3=new Point(p0.x,p0.y+d);

        Point p4=new Point(p3.x+dx,p3.y-dy);
        Point p5=new Point(p0.x+dx,p0.y-dy);
        Point p6=new Point(p0.x-(d-dx),p0.y-dy);
    }

```

```
Point[] ps=new Point[7];
ps[0]=p0;
ps[1]=p1;
ps[2]=p2;
ps[3]=p3;
ps[4]=p4;
ps[5]=p5;
ps[6]=p6;
        return ps;
}

int count=0;//递归次数
//给定6个点，画出一个正方体
private void draw(Graphics gg,Point p0,Point p1,Point p2,
        Point p3,Point p4,Point p5,Point p6){
    if(count<20){
        pn[count]=p0;
    }
    count++;
    int cc1=new java.util.Random().nextInt(255);
    int cc2=new java.util.Random().nextInt(255);
    int cc3=new java.util.Random().nextInt(255);

    Graphics2D g=(Graphics2D)gg;

    Polygon ponlygon3=new Polygon();
    ponlygon3.addPoint(p0.x,p0.y);
    ponlygon3.addPoint(p3.x,p3.y);
    ponlygon3.addPoint(p4.x,p4.y);
    ponlygon3.addPoint(p5.x,p5.y);
    g.setColor(new Color(100,0,0));
    g.fillPolygon(ponlygon3);

    Polygon ponlygon1=new Polygon();
    ponlygon1.addPoint(p0.x,p0.y);
    ponlygon1.addPoint(p1.x,p1.y);
    ponlygon1.addPoint(p2.x,p2.y);
    ponlygon1.addPoint(p3.x,p3.y);
    g.setColor(new Color(150,0,0));
    g.fillPolygon(ponlygon1);

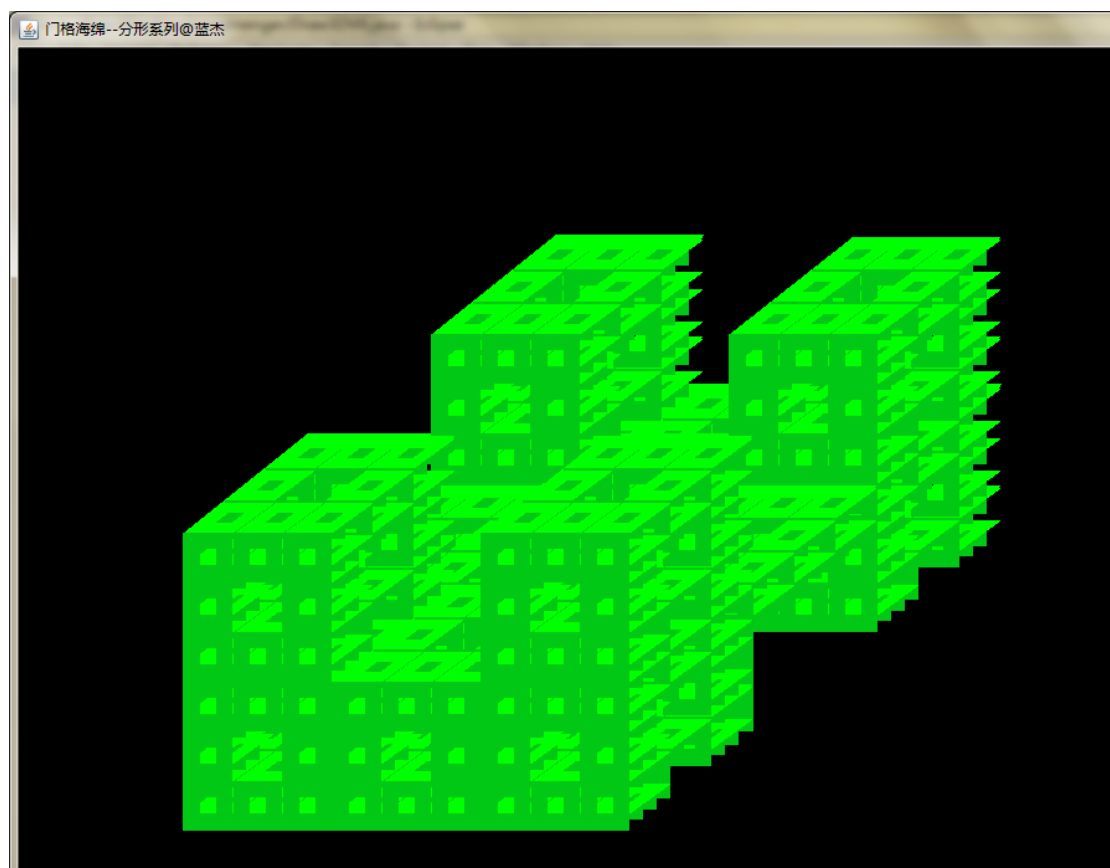
    Polygon ponlygon2=new Polygon();
    ponlygon2.addPoint(p0.x,p0.y);
    ponlygon2.addPoint(p5.x,p5.y);
```



```
ponlygon2.addPoint(p6.x,p6.y);
ponlygon2.addPoint(p1.x,p1.y);
g.setColor(new Color(255,0,0));
g.fillPolygon(ponlygon2);
}

public static void main(String[] args) {
    DrawMGV5 d3=new DrawMGV5();
    d3.initUI();
}
}
```

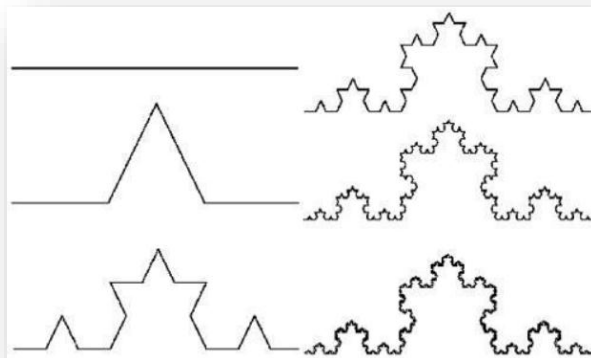
如下图示：可以随机不绘制某些单元格，或配成随机 Color，如下效果：



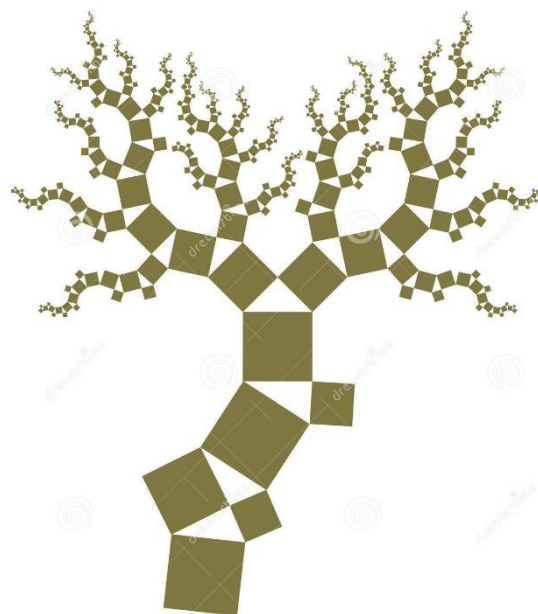
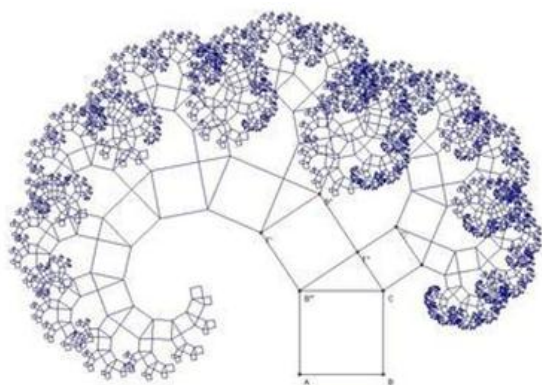
任务：

1. 生成门格海绵
2. 生成动画的穿越海绵中空格子的效果

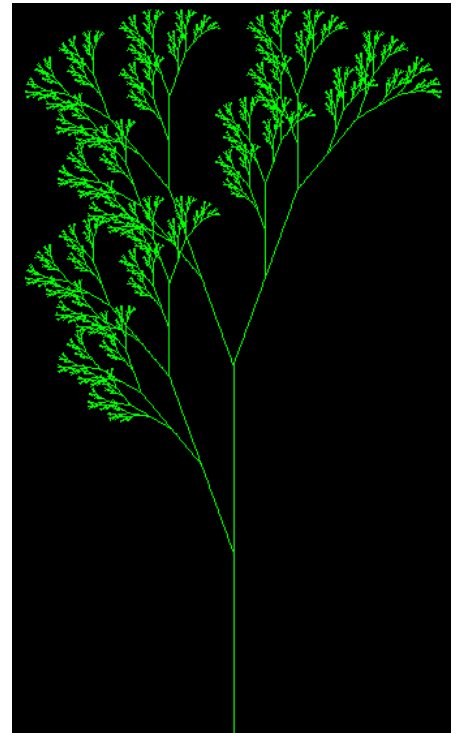
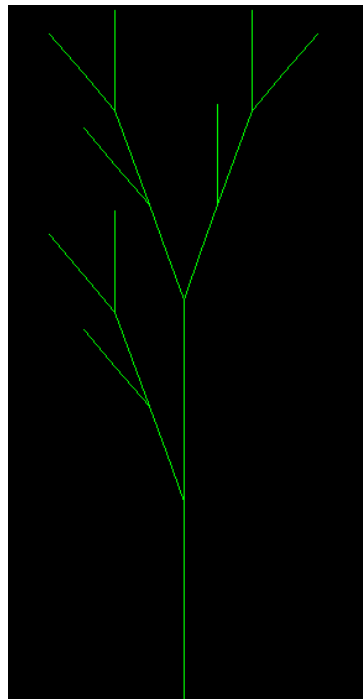
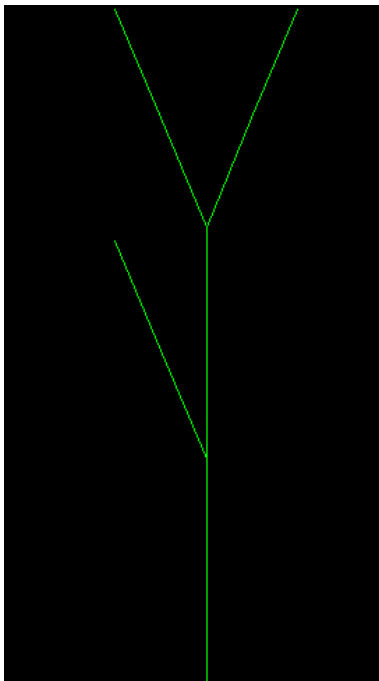
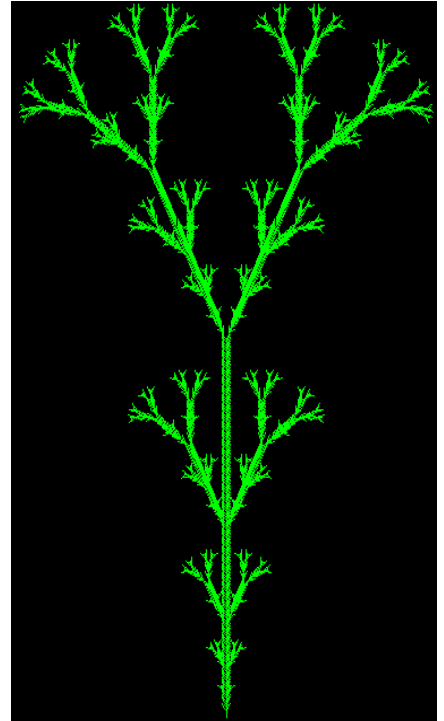
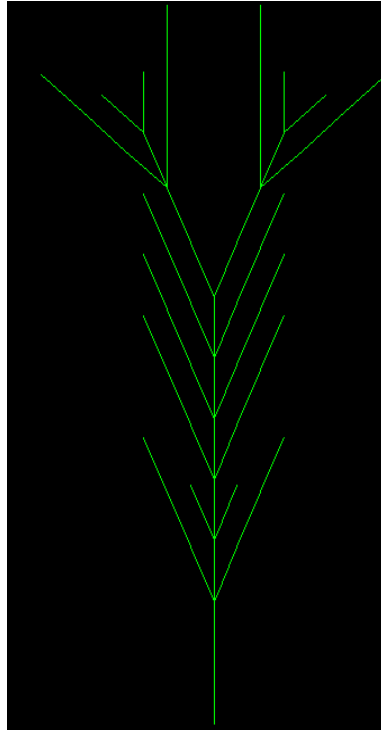
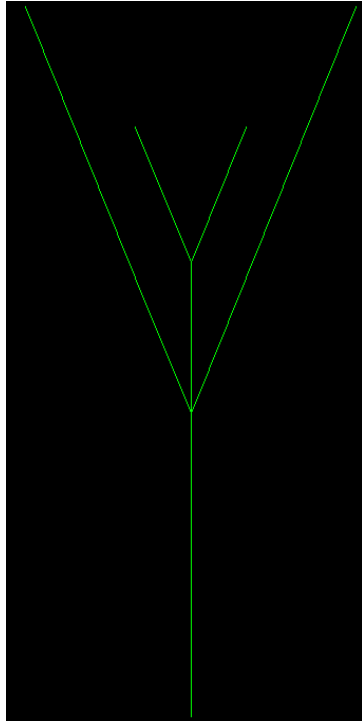
9. 科赫曲线系列

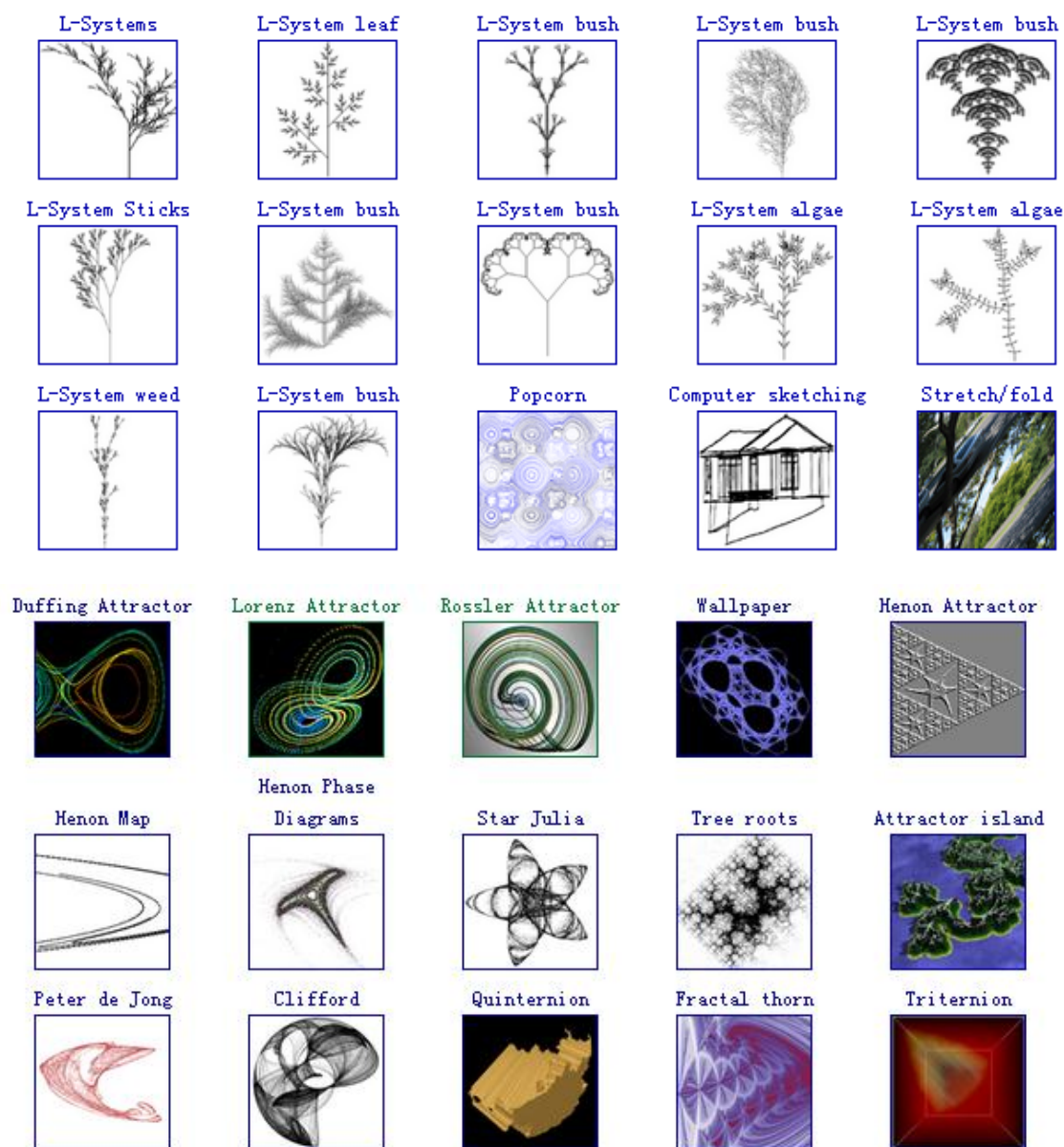


毕达哥拉斯树



10. 分形树



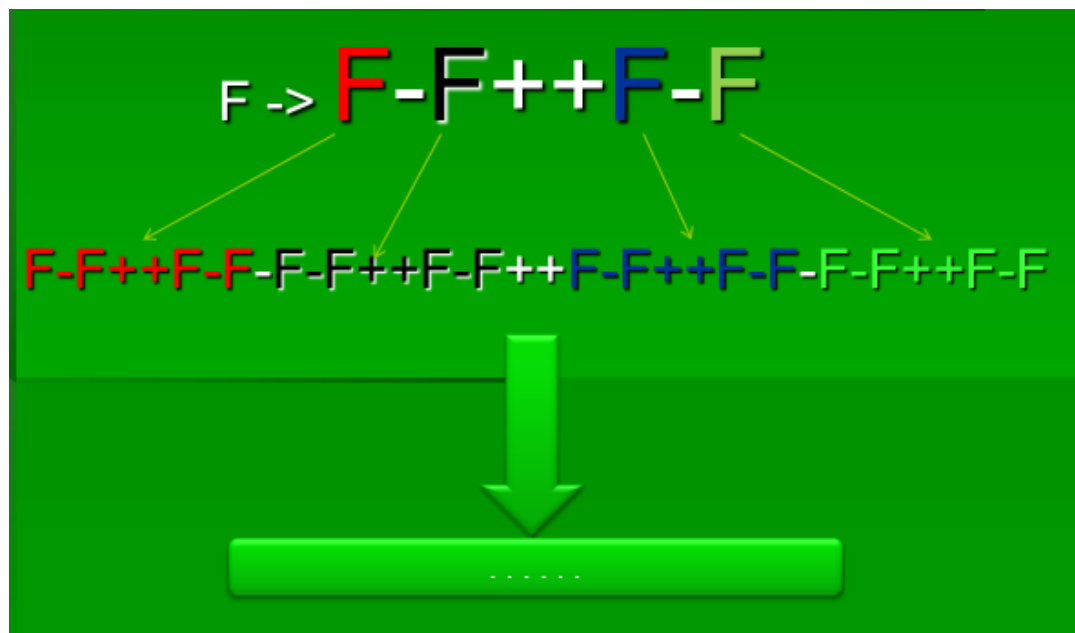


这里有所有你想要的：

<http://paulbourke.net/fractals/>

11. L-System

任何一棵树，终级方法：L-System，都可表示成如下公式：



axiom

axiom = F-F++F-F

F 表示向前走一个单位长度

+表示向右转 60 度

-表示向左转 60 度

F-F++F-F 画出的是什么？如果

$F(n+1) = F(n)-F(n)++F(n)-F(n)$?—

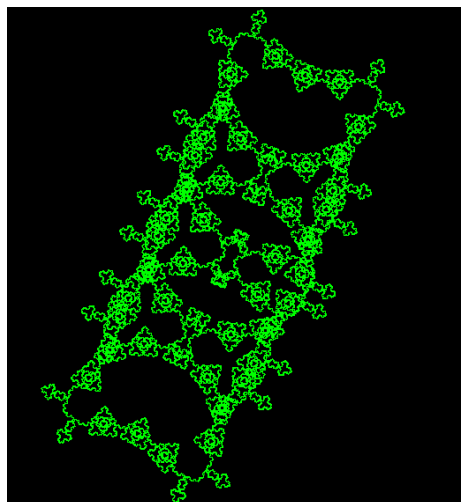
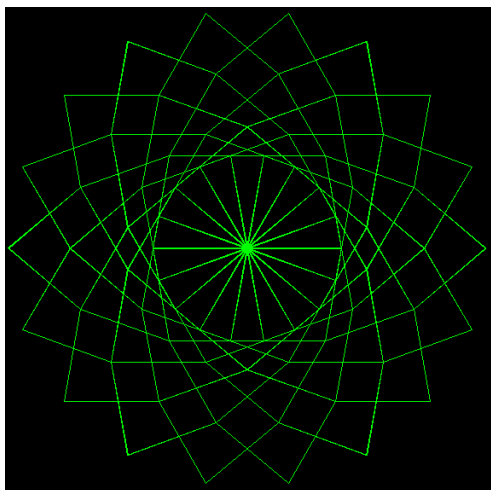
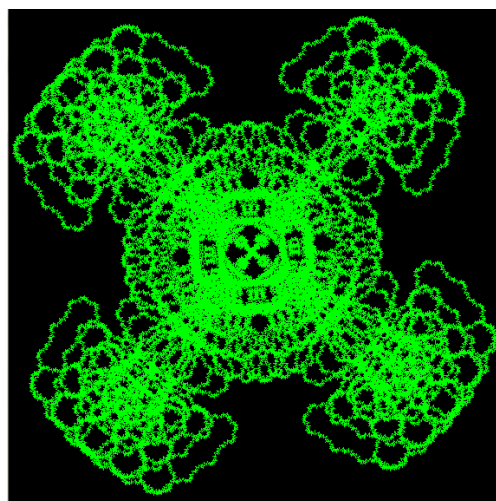
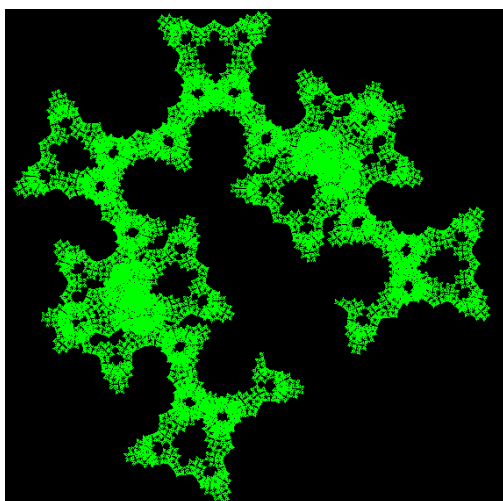
+表示向右转 90 度

-表示向左转 90 度

1.递归

2.预先生成各点坐标

3. 执行 axiom



12. 梅塔特隆立方体

<http://baike.baidu.com/item/%E6%A2%85%E5%A1%94%E7%89%B9%E9%9A%86%E7%AB%8B%E6%96%B9%E4%BD%93>

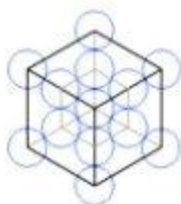
k. 星形四面体 (Star Tetrahedron)



=



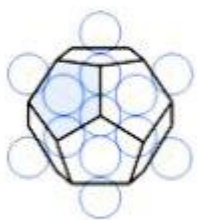
l. 立方体 (Metratron's Cube)



=



m. 十二面体 (Dodecahedron)



=



n. 二十面体 (Icosahedron)



=



o. 八面体 (Octahedron)



=



13. 光线追踪

<http://www.cnblogs.com/miloyip/archive/2010/03/29/1698953.html>



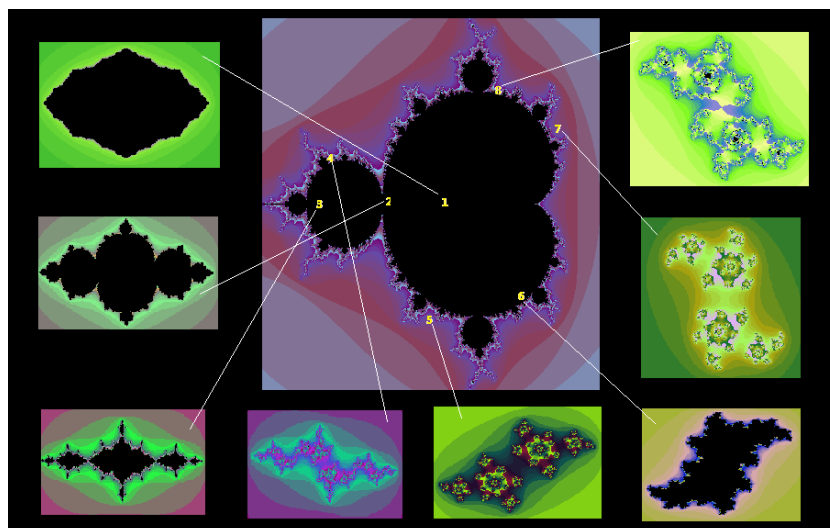
14. 经典之作:曼德勃罗集

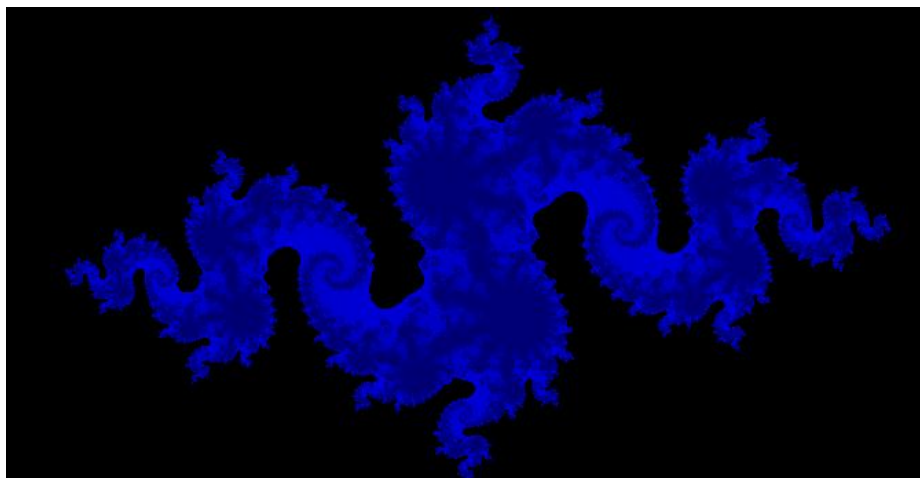
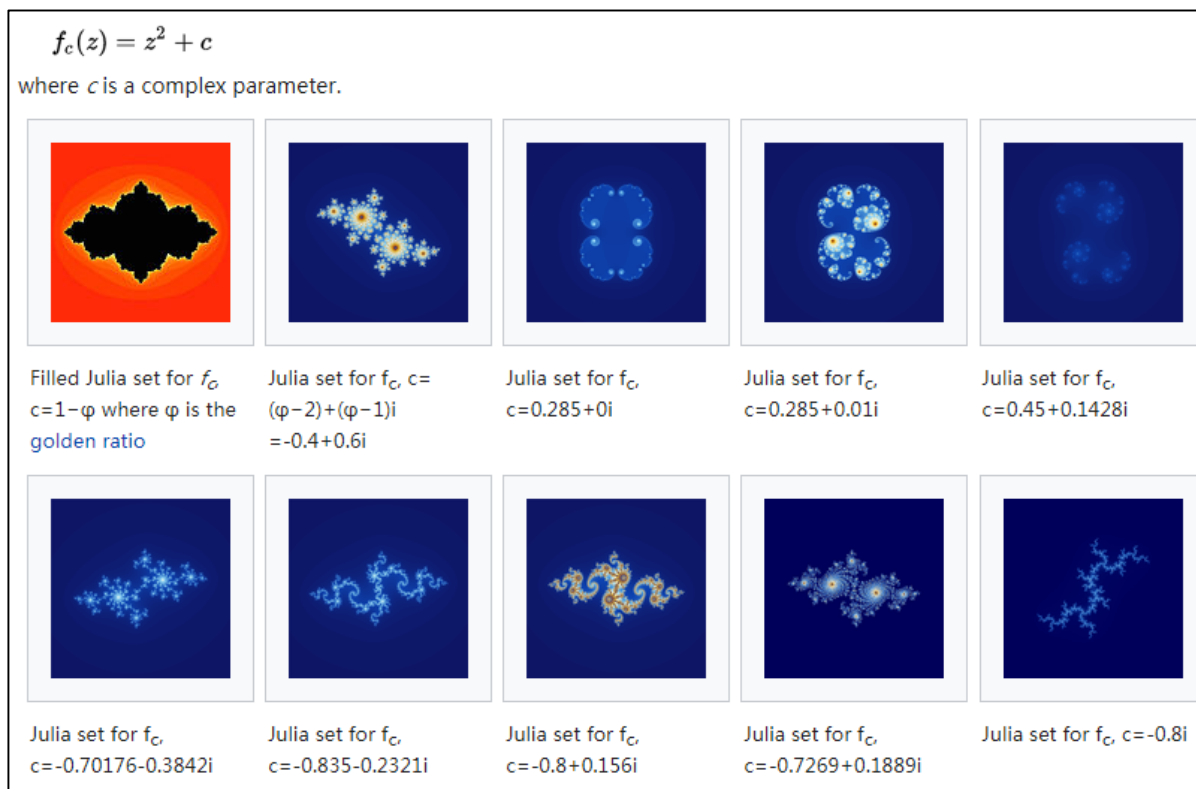
朱利亚集

https://en.wikipedia.org/wiki/Julia_set

曼德勃罗集

<http://baike.baidu.com/item/%E6%9B%BC%E5%BE%B7%E5%8B%83%E7%BD%97%E9%9B%86%E5%90%88>





画上上图的代码如下：

```
/**
 * 朱利亚集
 */
public class MDBLhdf extends JFrame{
    public static void main(String args[]){
        MDBLhdf dm=new MDBLhdf();
        dm.initUI();
    }

    public void initUI(){
```

```
        this.setSize(1800,800);
        this.setDefaultCloseOperation(3);
        this.setTitle("神奇的朱利亚集");
        this.getContentPane().setBackground(Color.BLACK);
        this.setVisible(true);
    }

    public void paint(Graphics g){
        super.paint(g);
        drawOne(g);
    }

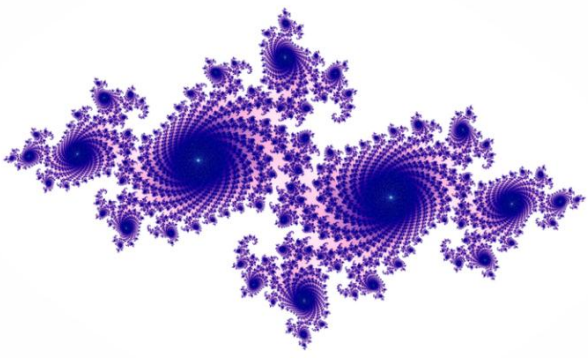
    private void drawOne(Graphics g){
        Complex c=new Complex();
        // z=z2+c
        c.rel = -0.8f;c.img=0.156f;
        //      c.rel = 0.285f;c.img=0.01f;
        //      c.rel = -0.835f;c.img=0.2321f;
        //      c.rel= -0.835f;c.img=-0.2321f;
        //      c.rel= 0.285f;c.img=0.0111f;
        Complex z=new Complex();
        for(float i=-300;i<300;i++){
            for(float j=-300;j<300;j++){
                z.rel=i/200.0f;
                z.img=j/200.0f;
                for(int k=0;k<120;k++){
                    double r= Math.sqrt(z.rel*z.rel+z.img*z.img);
                    if(r>2)
                    {
                        break;
                    }
                    else{
                        z=Complex.multiply(z, z);
                        z=Complex.add(z, c);
                        int x=(int)(i+400);
                        int y=(int)(j+300);
                        if(k>20){
                            //Color color= new Color(k*2000);
                            Color color=new Color(0,0,255-k);
                            g.setColor(color);
                            g.drawLine(x, y, x, y);
                        }
                    }
                }
            }
        }
    }
}
```

```

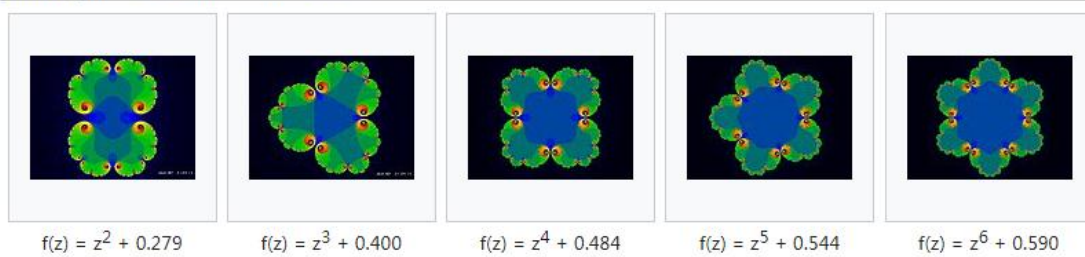
    }
    }
}

//进行复数运算的工具类
class Complex{
    public float rel=0.0f;
    public float img=0.0f;
    public Complex(){ }
    //复数相加
    public static Complex add(Complex a,Complex b){
        Complex c=new Complex();
        c.rel=a.rel+b.rel;
        c.img=a.img+b.img;
        return c;
    }
    //复数相乘
    public static Complex multiply(Complex a,Complex b){
        Complex c=new Complex();
        c.rel=a.rel*b.rel-a.img*b.img;
        c.img=a.img*b.rel+a.rel*b.img;
        return c;
    }
}

```



Examples of Julia sets [\[edit\]](#)



考虑：如何绘制出不同参数、不同放大程度、不同 Color 的图形？

如下代码示例：根据参数 c 变化，绘制动态图：

```

/**
 * 用双缓冲渐变放大的朱利亚集
 */
public class MDBLhdfControl extends JFrame{

```



```
private JFrame temf=this;

public static void main(String args[]){
    MDBLhdfControl dm=new MDBLhdfControl();
    dm.initUI();
}

public void initUI(){
    this.setSize(1800,800);
    this.setDefaultCloseOperation(3);
    // this.setTitle("神奇的朱利亚集");
    this.getContentPane().setBackground(Color.BLACK);
    this.setLayout(new FlowLayout());
    JButton bu=new JButton("Draw");
    bu.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent e) {
            toDraw();
        }
    });
    this.add(bu);
    this.setVisible(true);
}

private void toDraw(){
    Thread t=new Thread(){
        public void run(){
            Image bufferImage=temf.createImage(temf.getWidth(),temf.getHeight());
            Graphics bufferG=bufferImage.getGraphics();
            bufferG.fillRect(0, 0, temf.getWidth(), temf.getHeight());
            //绘制到缓冲区中
            for(float i=-2.0f;i<2;i+=0.01f){
                drawOne(bufferG,i);
                //System.out.println("buffer draw OK. . .");
                temf.getGraphics().drawImage(bufferImage, 0, 0, null);
                if(i>1.9){
                    i=-2.0f;
                }
            }
        }
    };
    t.start();
}
```

```

private void drawOne(Graphics g,float img){
    Complex c=new Complex();
        // z=z2+c
    //      c.rel = -0.8f;c.img=0.156f;
    //      c.rel= 0.285f;c.img=0.0111f;
    c.rel = -0.75f;c.img=img;
    Complex z=new Complex();
    for(float i=-200;i<200;i++){
        for(float j=-100;j<100;j++){
            z.rel=i/100.0f;
            z.img=j/100.0f;
            for(int k=0;k<280;k++){
                double r= Math.sqrt(z.rel*z.rel+z.img*z.img);
                if(r>2) {break; }
                else{
                    z=Complex.multiply(z, z);
                }
            }
            //      z=Complex.multiply(z, z);
            z=Complex.add(z, c);
            // z=Complex.add(z, c);
            int x=(int)(i+400);
            int y=(int)(j+300);
            if(k>0){
                Color color= new Color(k*500000);
                g.setColor(color);
                g.drawLine(x, y, x, y);
            }
        }
    }
}
}

```

//进行复数运算的工具类

```

class Complex{
    public float rel=0.0f;
    public float img=0.0f;
    public Complex(){ }
    //复数相加
    public static Complex add(Complex a,Complex b){
        Complex c=new Complex();
        c.rel=a.rel+b.rel;
        c.img=a.img+b.img;
        return c;
    }
}

```

```
}  
//复数相乘  
public static Complex multiply(Complex a,Complex b){  
    Complex c=new Complex();  
    c.rel=a.rel*b.rel-a.img*b.img;  
    c.img=a.img*b.rel+a.rel*b.img;  
    return c;  
}  
}
```

15. 自然是随机的吗？

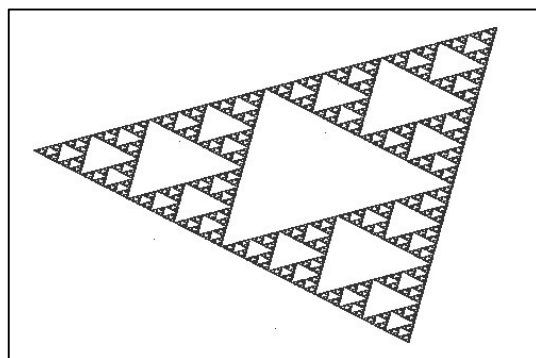
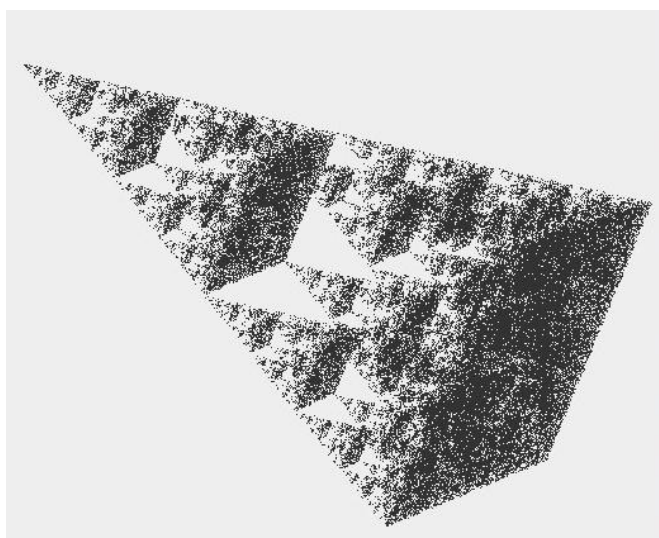
给我一个点，我能画出整个世界！这正是分形的魅力所在。

今天我们不说什么很枯燥的内容，我们先来看神奇的色子。

题目这样说：

- 1.平面上随机选 A,B,C 三个点。再随机选一个点，记为 P。
 - 2.有一个三面色子，每丢一次，则选中 ABC 三个中一点。
- 开始游戏：
- 1.重复丢色子，如果选中 A，则取 A 和 P 的中点 P1，画黑。
 - 2.如果选中 B，则取 B 和 P1 的中点 P2，画黑。
 - 3.如果选中 A，则取 A 和 P2 的中点 P3，画黑。
 - 4.一直重复，如每点一下鼠标，丢 100 次色子。

这个游戏看似有点规律，但是当你画出图片时会发生非常奇妙的东西！



12.学长的 L-system

也许是不小心又翻到了去年暑假的那个分形 PPT，让我想起来还有一个没有完成的任务，就是 L-system。当时刚接触 java，还是属于很年轻的，但是经过了那么久的积淀，我觉得我可以解决这个问题。

于是，我开始了探求 L-system 之旅。

首先我们还是来回顾一下 Koch 雪花，当时我们是直接用递归来解决的问题：

Java 代码 ☆

```
1.  /**
2.   * 画雪花的方法
3.   *
4.   * @param g 图形对象
5.   * @param x1 左边点的横坐标
6.   * @param y1 左边点的纵坐标
7.   * @param x2 右边点的横坐标
8.   * @param y2 右边点的纵坐标
9.   * @param count 画线的次数
10.  */
11. public void drawkoch(Graphics g, double x1, double y1, double x2,
12.    double y2, int count) {
13.    if (count <= 1) {
14.        g.drawLine((int) x1, (int) y1, (int) x2, (int) y2); // 画线
15.    } else {
16.        double x3 = (2 * x1 + x2) / 3; // 第一个三等份点的 x 坐标
17.        double y3 = (2 * y1 + y2) / 3; // 第一个三等份点的 y 坐标
18.        double x4 = (x1 + 2 * x2) / 3; // 第二个三等份点的 x 坐标
19.        double y4 = (y1 + 2 * y2) / 3; // 第二个三等份点的 y 坐标
20.        double k = (x1 - x2) * (y1 - y2); // 线的斜率
21.        double x5 = x1, y5 = y1; // 第一个三等份点的 x 坐标
22.        if (y3 == y4) // 直线
23.        {
24.            x5 = (x3 + x4) / 2;
25.            y5 = y3 - (x4 - x3) * Math.sqrt(3) / 2;
26.        } else if (k < 0) // 左斜线
27.        {
28.            x5 = x1;
29.            y5 = y4;
```

```

30.         } else if (k > 0) // 右斜线
31.         {
32.             x5 = x2;
33.             y5 = y3;
34.         }
35.         if (x3 == x4) // 如果斜线为竖线
36.         {
37.             x5 = x3;
38.             y5 = y3;
39.         }
40.         // 画尖端的左面那条线
41.         drawkoch(g, x3, y3, x5, y5, count - 1);
42.         // 画尖端的右面那条线
43.         drawkoch(g, x5, y5, x4, y4, count - 1);
44.         // 画左边那条直线
45.         drawkoch(g, x1, y1, x3, y3, count - 1);
46.         // 画右边那条直线
47.         drawkoch(g, x4, y4, x2, y2, count - 1);
48.     }
49. }

```

但是这一次，我们使用的并不仅仅是递归，而是用字母表和符号串来表达生成的对象的初始形式，称之为公理（axiom）。

现在我们可以定义如下字符规则：

F：在当前方向前进一个单位，并画线

+：逆时针旋转 α 角度


-：顺时针旋转 α 角度

我们再定义一个字符转换规则：**F -> F-F++F-F**

那么下一步将会转成：**F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F**

以此类推，我们取 $\alpha=60$ 度，便得到我们以前画的 Koch 雪花。

那么如果我们在每画完一条直线时，稍微附带着偏转一个角度，结果会更加的令人惊讶的！

Java 代码 

```

1. public static final String strF = "F-F++F-F";// 基础字符串
2. private double theta = Math.PI * 60 / 180;// 偏转固定的角度
3. String axiom = "F-F++F-F";// "公理"字符串
4. double d = 50;// 单位长度
5. double garma = 3;// 微调时的偏转角度
6. double x1, y1;// 画线的初始点
7.
8. /**
9.  * 根据字符串 axiom, 画出图形
10.  * @param axiom
11.  * @param g
12.  */
13. public void drawLS(String axiom, Graphics g) {
14.     // 把中点坐标定到屏幕正中心
15.     x1 = getWidth() / 2;
16.     y1 = getHeight() / 2;
17.     double x2 = x1, y2 = y1;// 用于存取下一步的位置
18.     double alpha = 0;// 当前所指向的角度
19.     // 循环遍历字符串，根据字符串所给出的提示，F 表示前进一个单位，+表示逆时针旋转
    α, -表示顺时针旋转 α
20.     for (int i = 0; i < axiom.length(); i++) {
21.         switch (axiom.charAt(i)) {
22.             case 'F':// 如果是 F 就走一部
23.                 // 计算下一步的坐标
24.                 x2 = x1 + d * Math.cos(alpha);
25.                 y2 = y1 + d * Math.sin(alpha);
26.                 // 根据坐标画线
27.                 g.drawLine((int) x1, (int) y1, (int) x2, (int) y2);
28.                 // x1, y1 变到新的坐标点
29.                 x1 = x2;
30.                 y1 = y2;
31.                 break;
32.             case '-':// -表示顺时针旋转 α
33.                 alpha -= theta;
34.                 // 在两个 strF 中间，要进行一个小角度的偏转，使得图形偏转角度趋于混乱
35.                 if (i < axiom.length() - 2) {
36.                     if (axiom.charAt(i + 1) == 'F'
37.                         && axiom.charAt(i + 2) == '+') {
38.                         double a = Math.PI * garma / 180;// 根据 γ 计算出偏转角度 a
39.                         alpha -= a;// α 再逆时针旋转 a
40.                     }
41.                 }
42.                 break;
43.             case '+':// +表示逆时针旋转 α

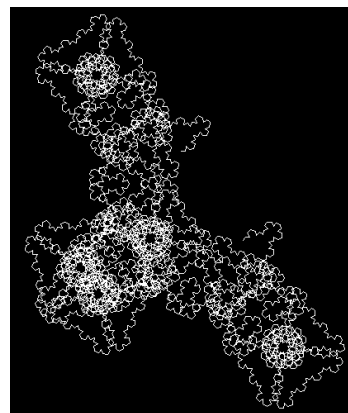
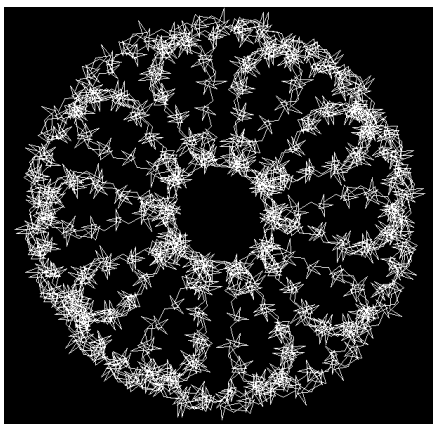
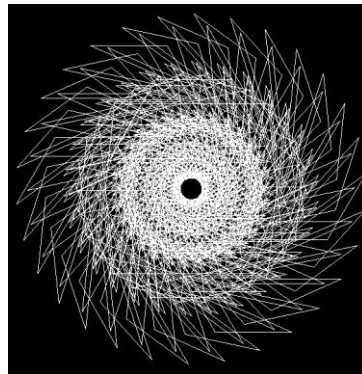
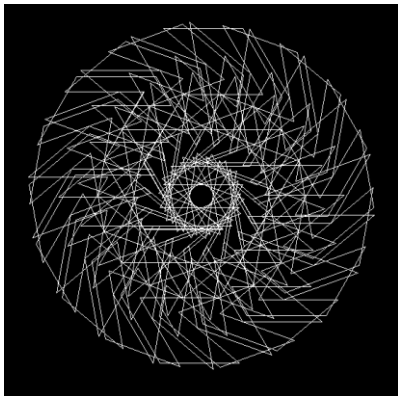
```



```

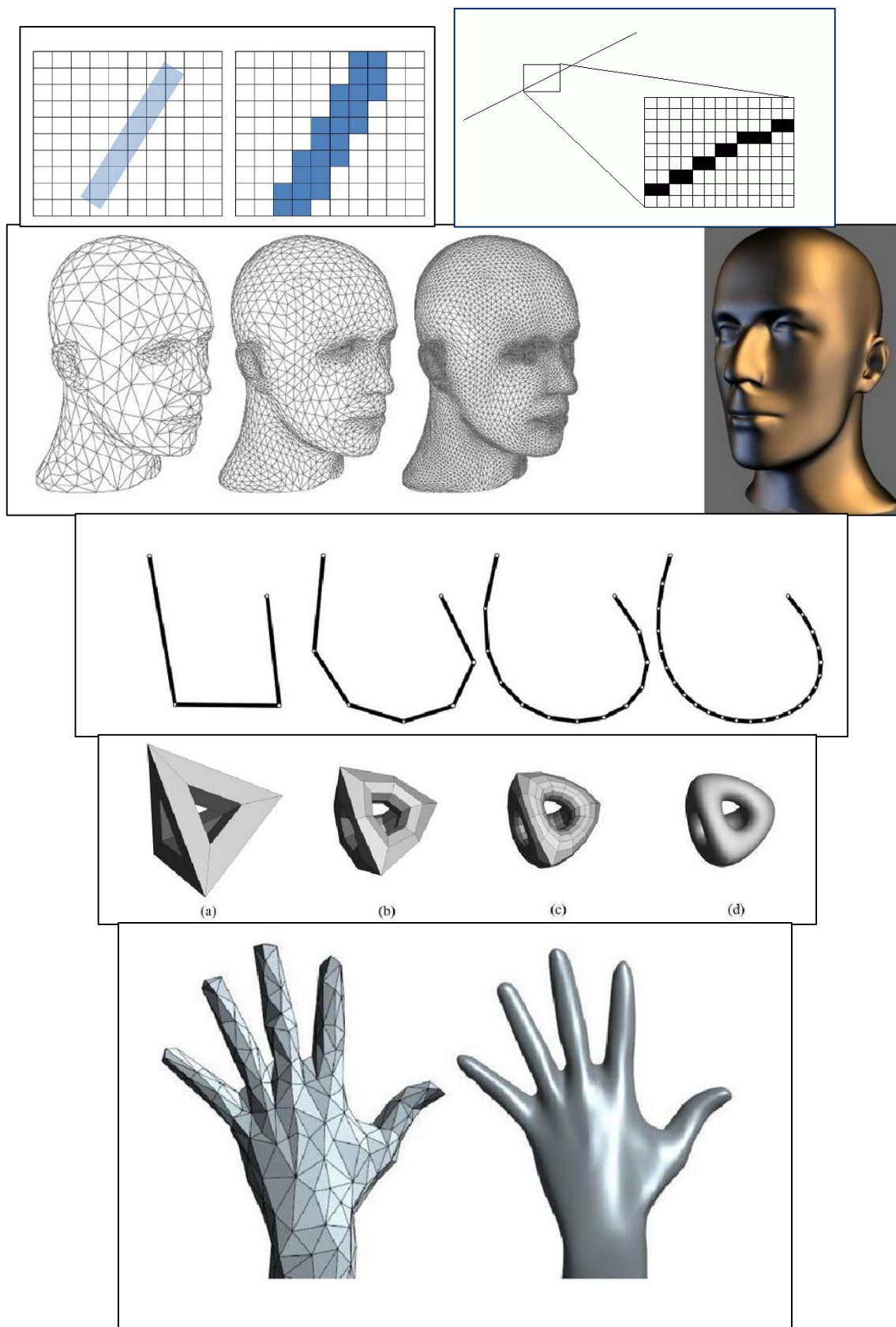
44.         alpha += theta;
45.         break;
46.     }
47. }
48. }
49. /**
50.  * 字符串递归的方法
51.  * @param str 如果是 small 就是变小
52.  *           , 如果是 large 就是变大
53.  */
54. public void dealAxiom(String str) {
55.     if (str.equals("small")) {
56.         if (!axiom.equals("F")) {
57.             axiom = axiom.replace(strF, "F");// 用"F"代替 strF
58.         }
59.     }
60.     if (str.equals("large")) {
61.         axiom = axiom.replace("F", strF);// 用 strF 代替"F"
62.     }
63. }

```



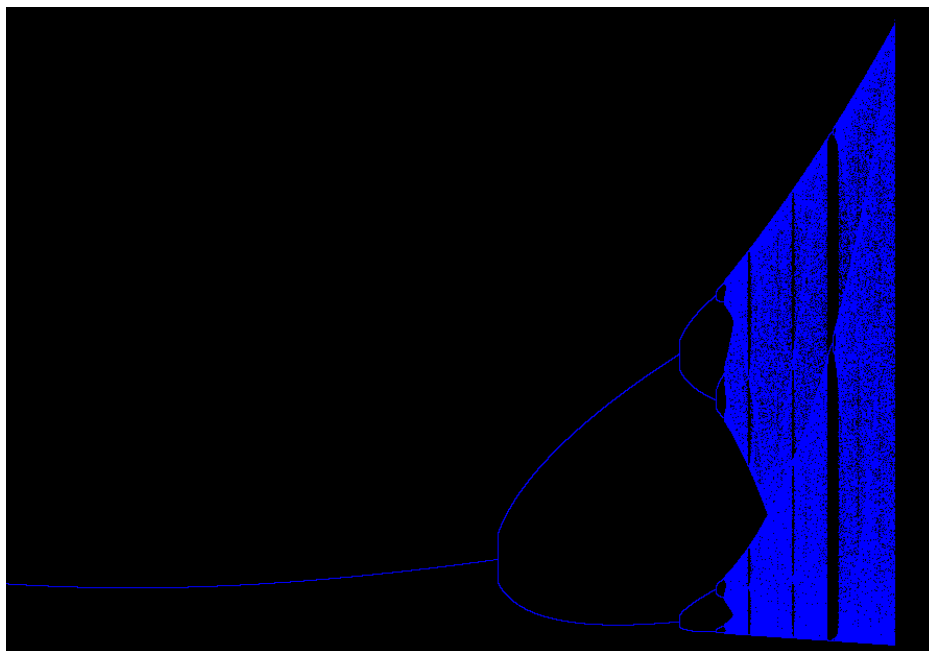
学习永远没有尽头，to be continue...

要看到现象的本质





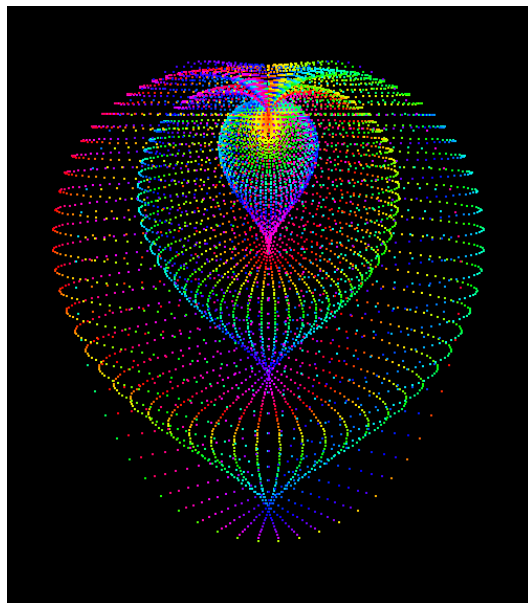
混沌之迷



逻辑斯蒂方程，代码如下：

```
// 逻辑斯蒂方程:  $x = RX(1-x)$ 
private void draw(Graphics2D g){
    double x=0.00d;
    double k=0.00d;
    g.setColor(Color.BLUE);
    for(double f=1.01d;f<2.00f;f+=0.000001d){
        k=f;
        x+=f;
        double xn=k*x*(1-x);
        x=xn;
        if(Double.isInfinite(x))break;
        // if(x>0.5d)break;
        double xi= (k*800)-200;
        double yi= (xn*220)+600;
        g.draw(new Line2D.Double(xi, yi, xi, yi));
    }
}
```

永恒之心



```
//r=a(1-sinθ)
private void draw(Graphics2D g){
    for(int i=0;i<=180;i++){
        for(int j=0;j<=180;j++){
            double r=Math.PI/45*i*(1-Math.sin(Math.PI/45*j))*20;
            double x=r*Math.cos(Math.PI/45*j)*Math.sin(Math.PI/45*i)+300;
            double y=-r*Math.sin(Math.PI/45*j)+200;
            Color c=Color.getHSBColor(i*j/8100.0f, 0.9999f,0.9999f);
            g.setColor(c);

            g.drawOval((int)x, (int)y, 1,1);
            try{
                Thread.sleep(10);
            }catch(Exception e){}

        }
    }
}
```


指定阅读书籍：

能让未来不可避免的，只有思想——哈耶克

我们的理论根基：



《系统论》 贝塔朗菲

《信息论》 香农

《博弈论》 冯·诺意曼

《控制论》 维纳

图灵机

奥派经济学： 冯塞斯

哈耶克