计算机图形学实验报告

姓名：[redacted]
学号：[redacted]

目录

前言

本次课程设计个人部分采用 QT5.8.0 编写，主要内容为：图元的生成：直线、圆、椭圆、区域填充；样条曲线的生成：Bezier 曲线的生成、B-样条曲线的生成；分形图形的生成：Koch 曲线、Julia 集、Mandelbrot 集和蕨类植物。附件中含有最终工程和可执行文件，各窗口有使用说明。

一、图元的生成

1.1 直线的生成

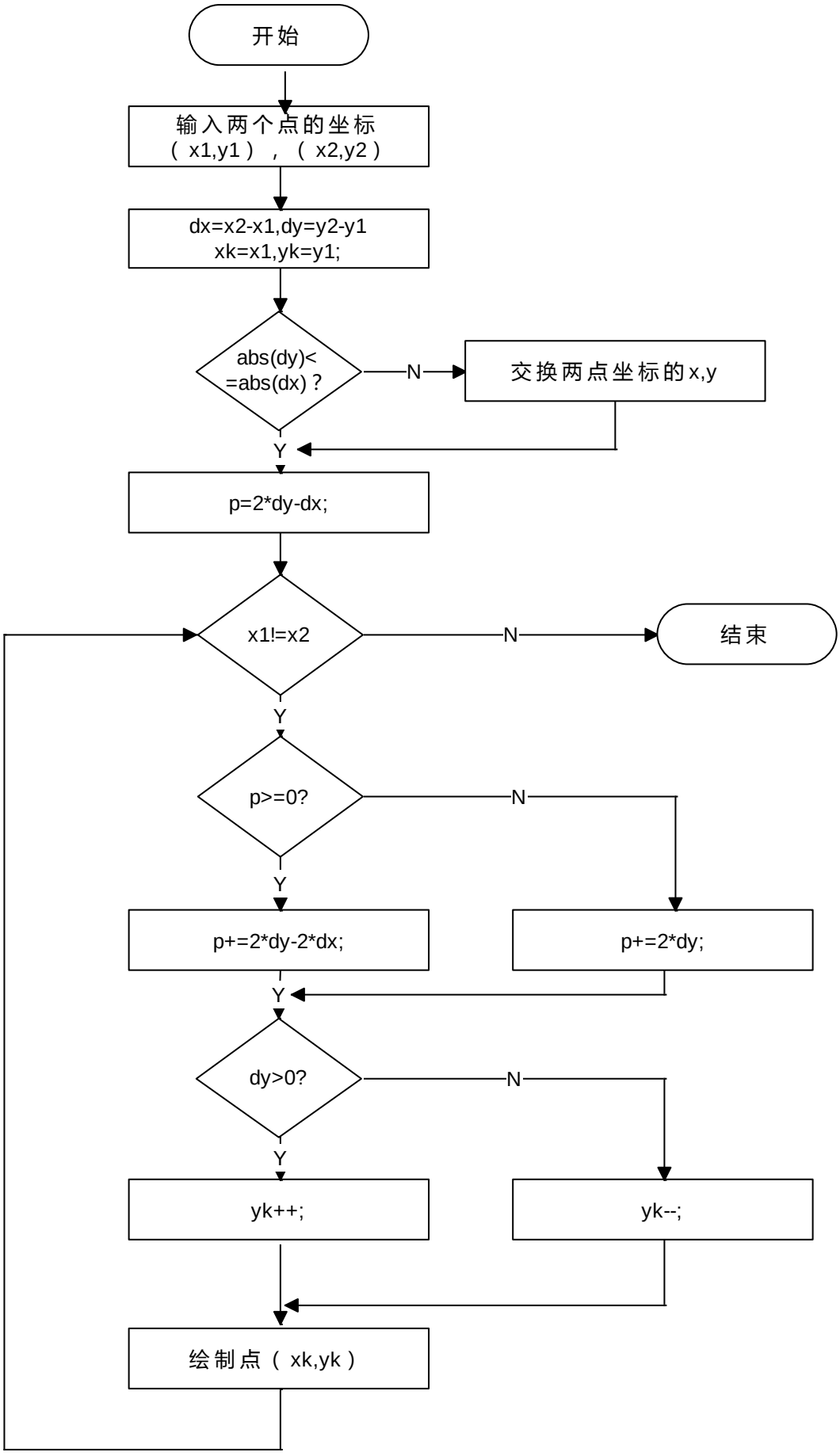
1.1.1 基本原理

由于我们现在使用的显示器都是栅格显示器，栅格显示器由一个个像素点组成的，故我们显示各种图形实际上都是由一个个离散点显示的。因此，我们可以将屏幕（或者画板）定义好坐标轴，对于特定的图形，只需计算其相应被点亮的栅格的坐标即可显示出所需图形。对于直线来说，只需知道直线两端的端点，用数学知识，即可计算出沿线各点的坐标。由此产生了很多算法，如 DDA 算法、Bresenham 算法、并行画线算法等。

本实验中选择 Bresenham 算法，该算法利用增量计算各点的坐标，无需计算浮点数，效率高，速度快。

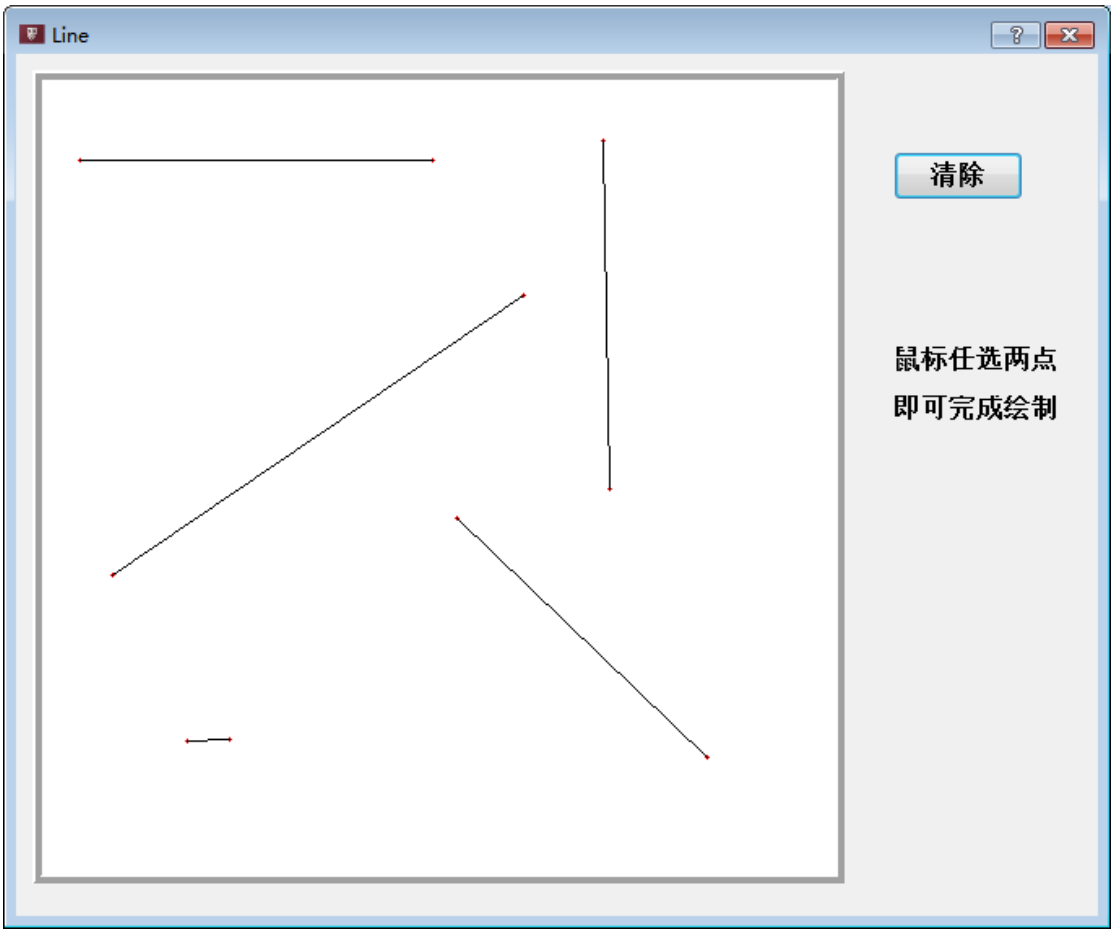
1.1.2 实现算法

Bresenham 直线算法流程图一所示。该算法只需要输入两个点的坐标即可完成直线的绘制。程序中，通过鼠标事件获取输入参数。本例中，通过 *mousepressEvent()*事件获得两次鼠标点击的点的坐标，每次将点的坐标存下，同时计数器加 1，当计数器等于 2 时清零，如此循环，可实现重复绘制点。



图一 直线绘制算法流程图

1.1.3 实验结果



图二 直线生成实验结果

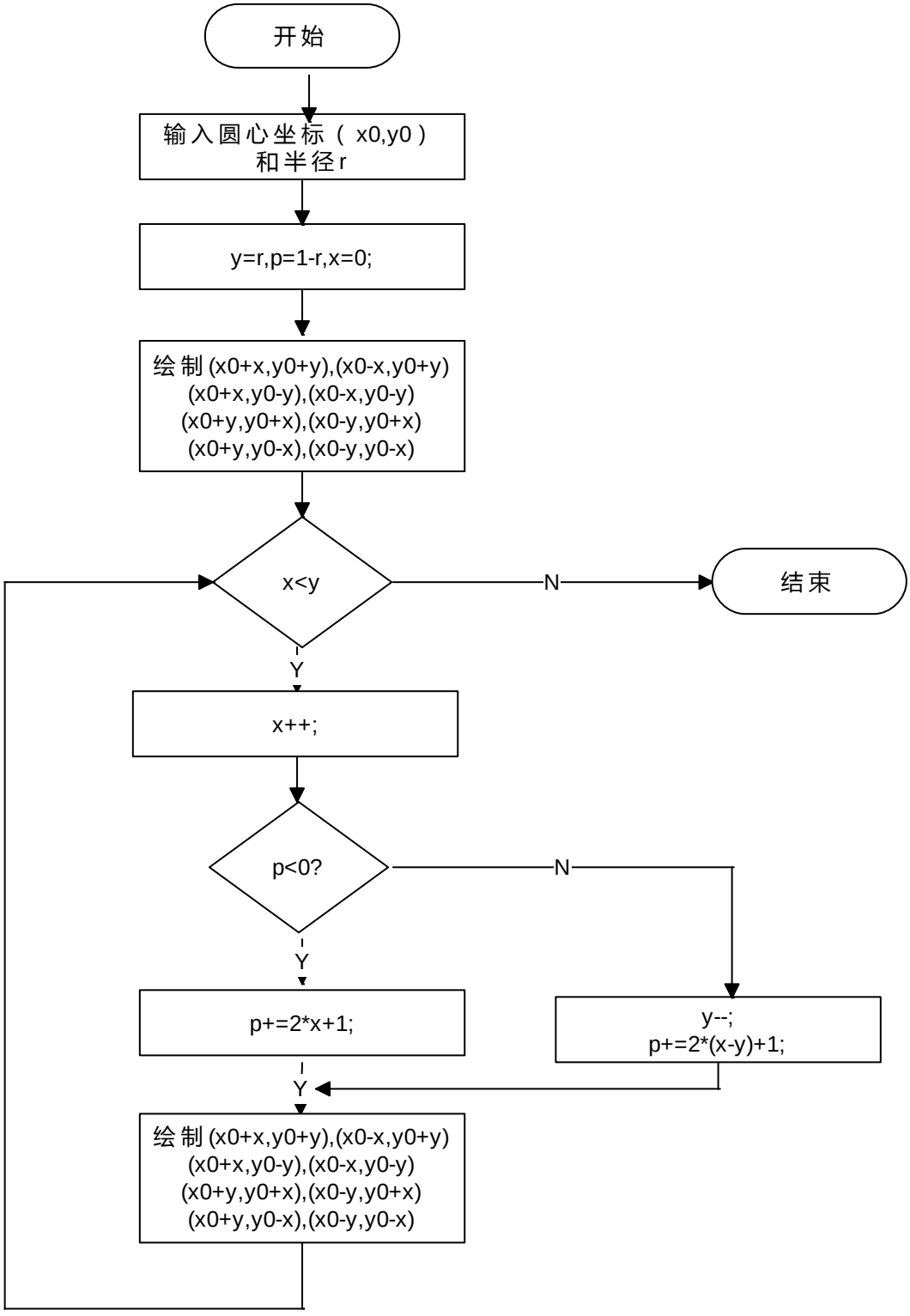
实验结果如图二所示，实验中，鼠标任选两个点（选取的两个点会用红点标注），即自动完成两点间直线的绘制。实验中，分别绘制了横线、竖线、斜率为正的直线、斜率为负的直线、短线等类型的线，由图二可知，实验成功。

1.2 圆的生成

1.2.1 基本原理

本质上来说，圆的生成的基本原理和直线没有大区别，只是图案改变了。由于圆是弧形，故需要更巧妙的设计，本实验中采取中点圆算法绘制圆形，该算法在绘制时每次判断前后两点的中点，若中点在圆弧内，在下一个点外圆弧外方向前进，反之往圆弧内方向前进。且由于圆的对称性，无需计算所有点的坐标，本实验中，只计算第一象限的 1/8 圆弧的坐标，其余坐标利用对称性均可获得。

1.2.2 实现算法

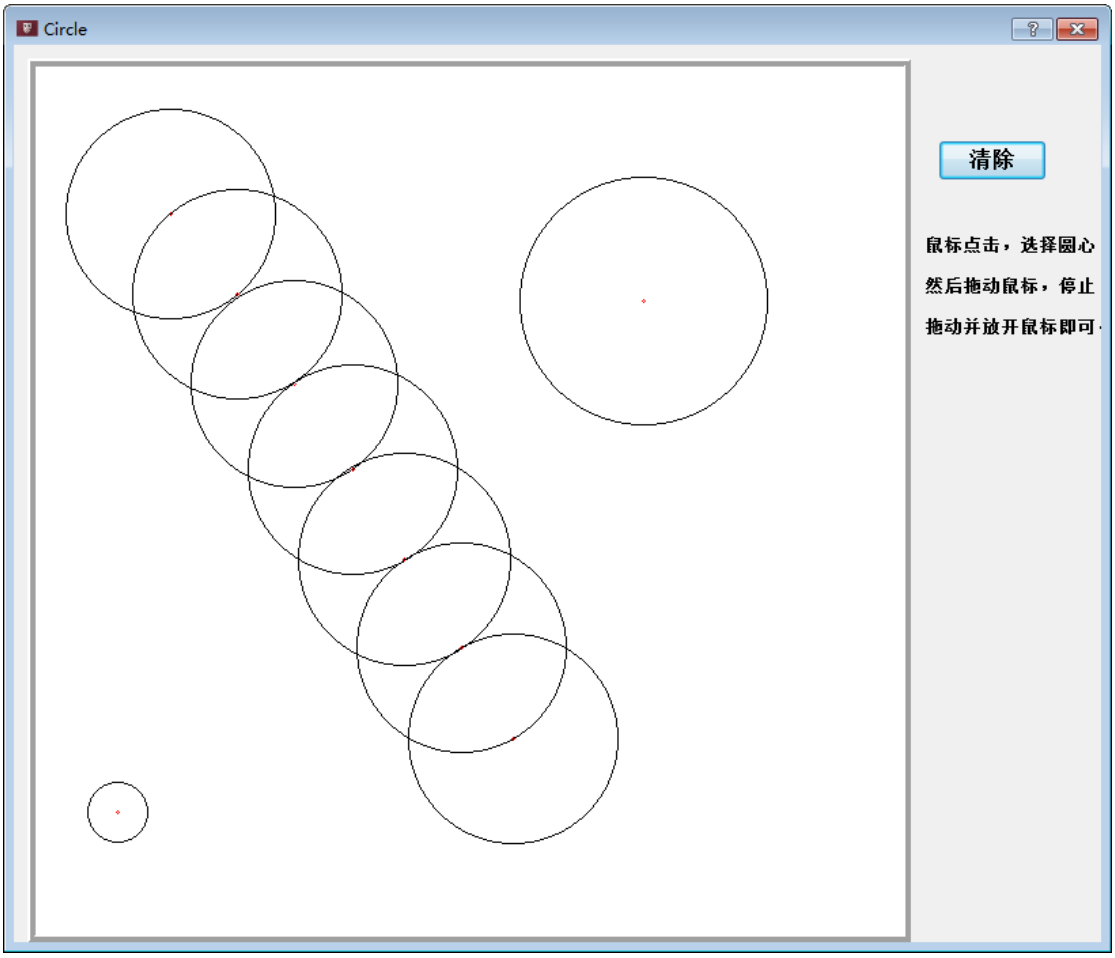


图三 圆的生成算法流程图

圆的生成算法流程图如图三所示，该算法中，输入为圆心坐标和半径，利用

圆的对对称性，只计算第一象限中上半部分（即八分之一圆），其余点的坐标利用对称性绘制。程序中，通过 *mousePressEvent()*获得圆心的坐标(x0,y0)，然后*mouseMoveEvent()*获取鼠标移动过程中的位置(x,y)，同时计算当前点距圆心的距离（即半径 r）当鼠标松开后，就得到了以上算法的两个输入参数，将圆心坐标和半径代入以上算法，并在 *mouseReleaseEvent()*事件中调用该算法，即实现圆的生成。

1.2.3 实验结果



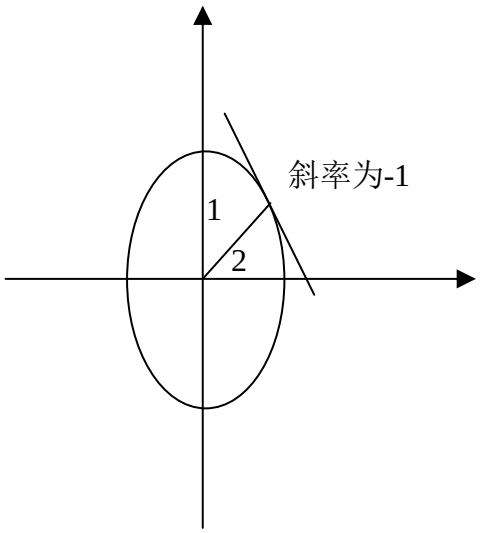
图四 圆生成算法实验结果

实验结果如图所示，画圆方法如下：首先鼠标点击一个点，按住不放，该点即为圆心，然后拖动鼠标，放开鼠标后即生成圆。由图四可知，该算法绘制的圆效果很好。

1.3 椭圆的生成

1.3.1 基本原理

与圆的生成相比，椭圆的生成过程类似，从算法上来说，椭圆比圆多一个参数，椭圆同时有短轴和长轴，故需要三个参数：长轴长，短轴长和圆心坐标。本设计中仍采用中点椭圆生成算法。



图五 椭圆处理区域

如图五所示，将椭圆分为两个区域处理，其中区域 1 斜率小于-1，区域 2 斜率大于-1，只需计算出第一象限的 1/4 椭圆，其余坐标即可利用对称性获得。利用椭圆方程，可定义如下函数，其中r_x，r_y分别为长轴长和短轴长。

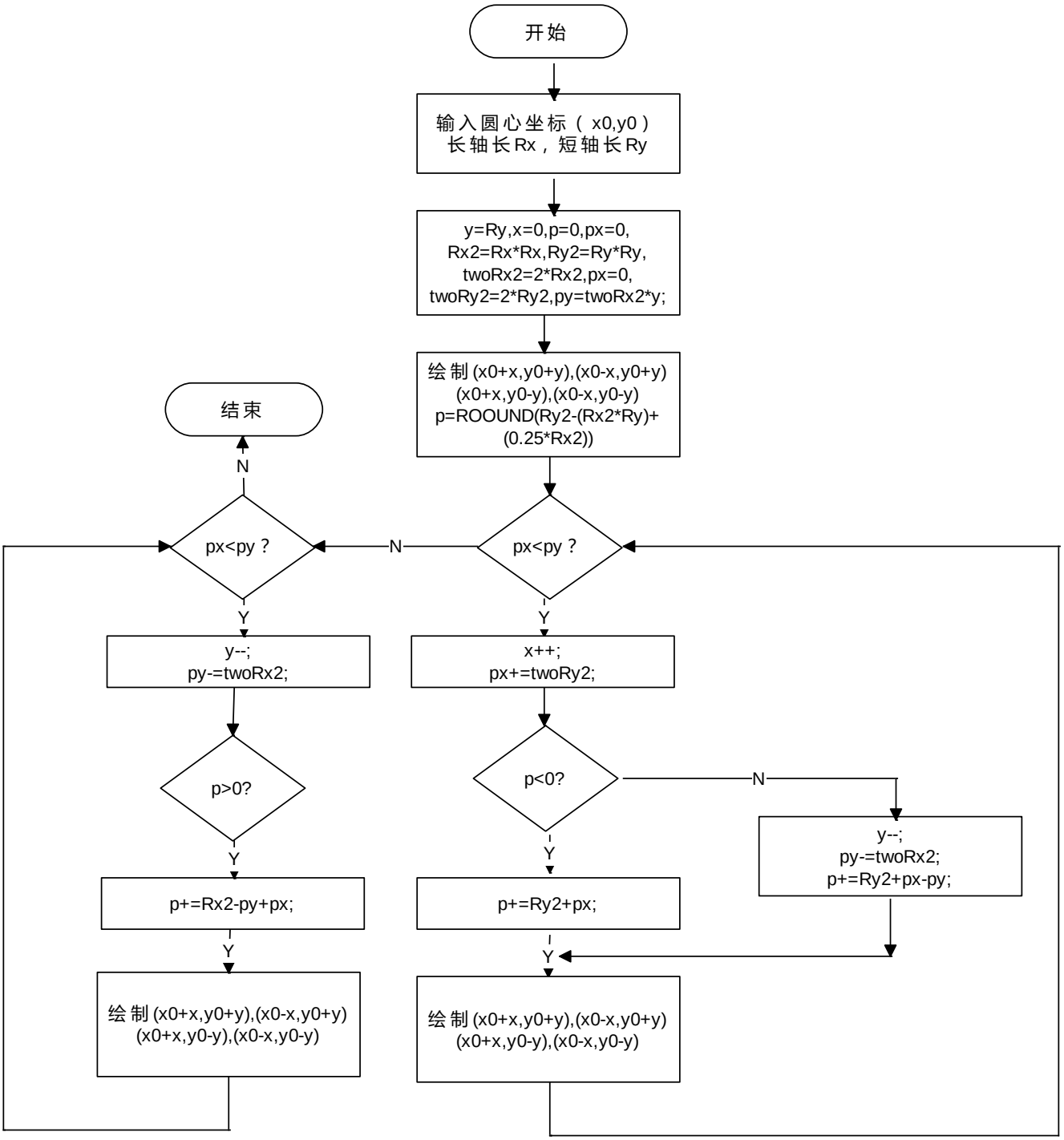
$$f_{ellipse}(x,y) = {r_y}^2x^2 + {r_x}^2y^2 - {r_x}^2{r_y}^2$$

上述函数具有下列特性：

$$f_{ellipse}(x,y) = \begin{cases} < 0 & \text{(x,y)在椭圆内} \\ = 0 & \text{(x,y)在椭圆上} \\ > 0 & \text{(x,y)在椭圆外} \end{cases}$$

通过上述关系，从$(0, r_y)$开始，在 x 方向取单位步长更新坐标，并计算每次的增量，当到达区域 1 和区域 2 边界时，更换成 y 方向的单位步长，从而将第一象限的点绘制完成，经过一系列简单的数学推导，可得出不同情形下各点的增量，具体形式见实现算法中的流程图部分。

1.3.2 实现算法

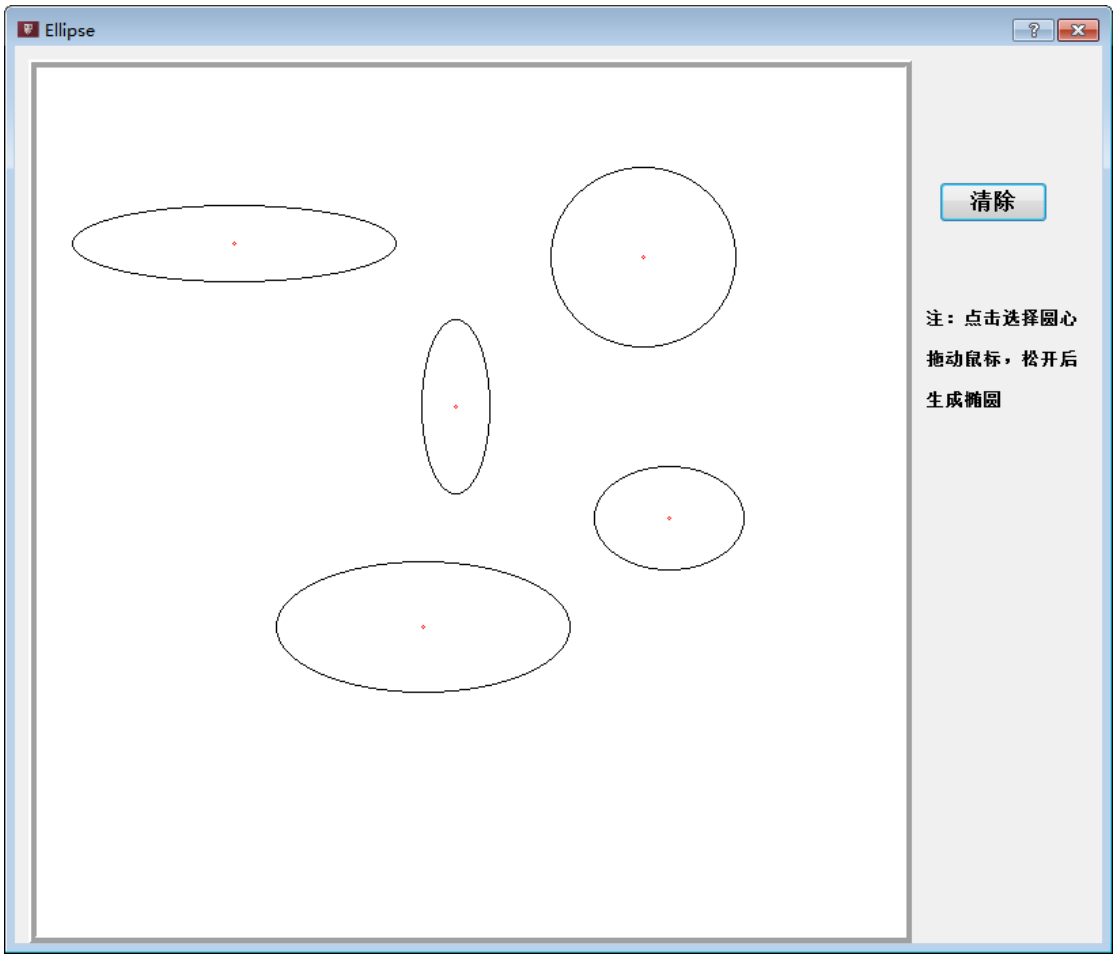


图六 椭圆生成算法流程图

椭圆生成算法流程图如图六所示，对比图四圆的生成算法，可见椭圆更为复杂，因为椭圆除了多一个参数外，还得分区域考虑，不同的区域参数更新规则不同，但是其基本原理相同。在椭圆生成算法中，圆心坐标(x0,y0)的获取和圆生成算法相同，利用 *mousePressEvent()*事件，至于长短轴的获取，为了简便起见，仍

利用 *mouseMoveEvent()*事件和 *mouseReleaseEvent()*事件获取鼠标拖动终点(x,y)，然后将长轴取为 x 的增量 dx=x-x0,短轴为 dy=y-y0，这种方法相对简单，且实验表明生成的椭圆效果也很好。

1.3.3 实验结果



图七 椭圆生成算法实验结果

实验结果如图七所示，图中绘制了几个不同方向的椭圆。绘制方法如下：首先点击鼠标左键选择圆心（以红点表示），然后按住鼠标不动，松开后即完成椭圆的绘制，由图七看见，本例中可实现不同形状的椭圆的绘制，且绘制效果很好。

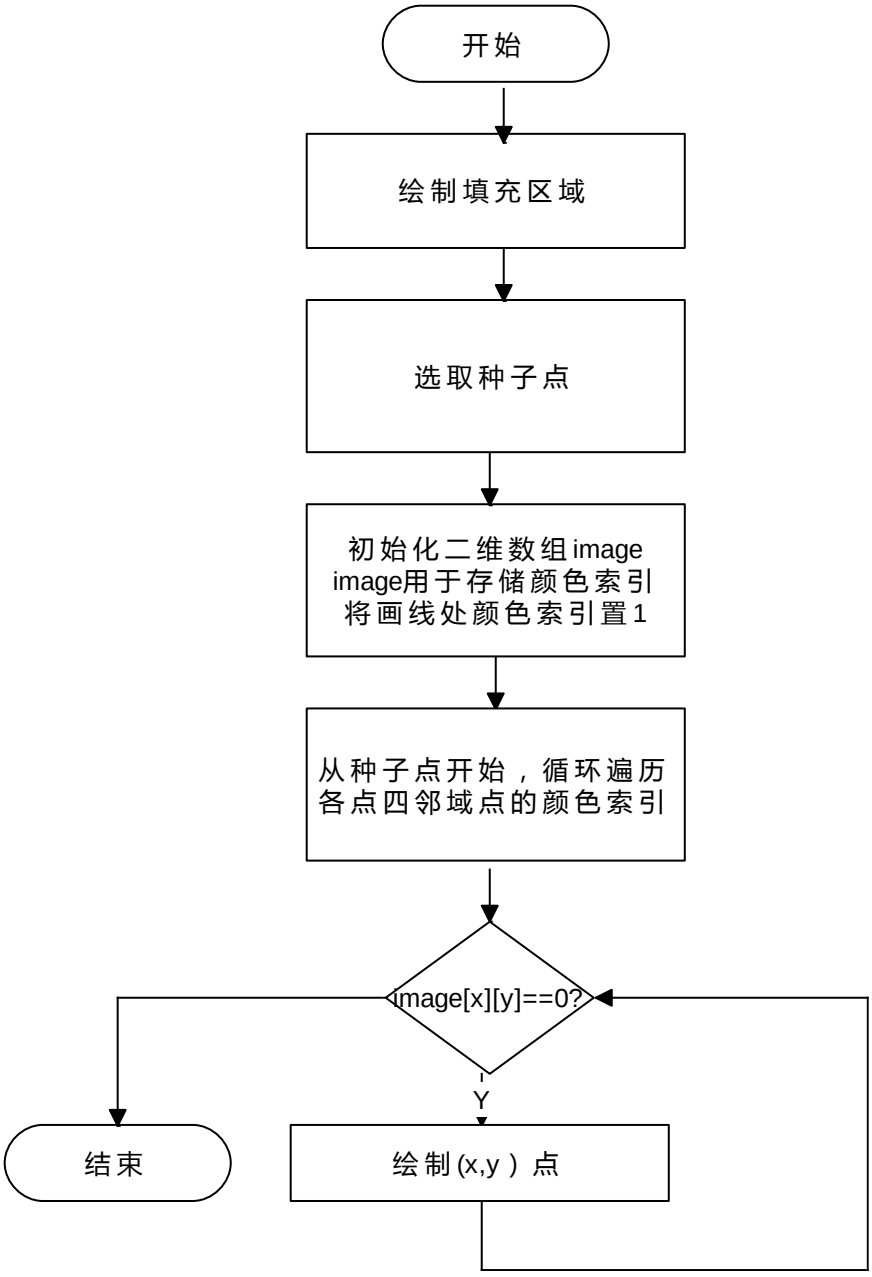
1.4 区域填充

1.4.1 基本原理

在光栅系统中有两种基本的区域填充方法，一种是通过确定横跨区域的扫描

线的覆盖间距进行填充；另一种是从给定的位置开始涂色直到指定的边界为止。由此产生了多种不同的填充算法，如扫描线转换法，边界填充算法，泛滥填充算法等，本设计采用泛滥填充算法。

1.4.2 实现算法

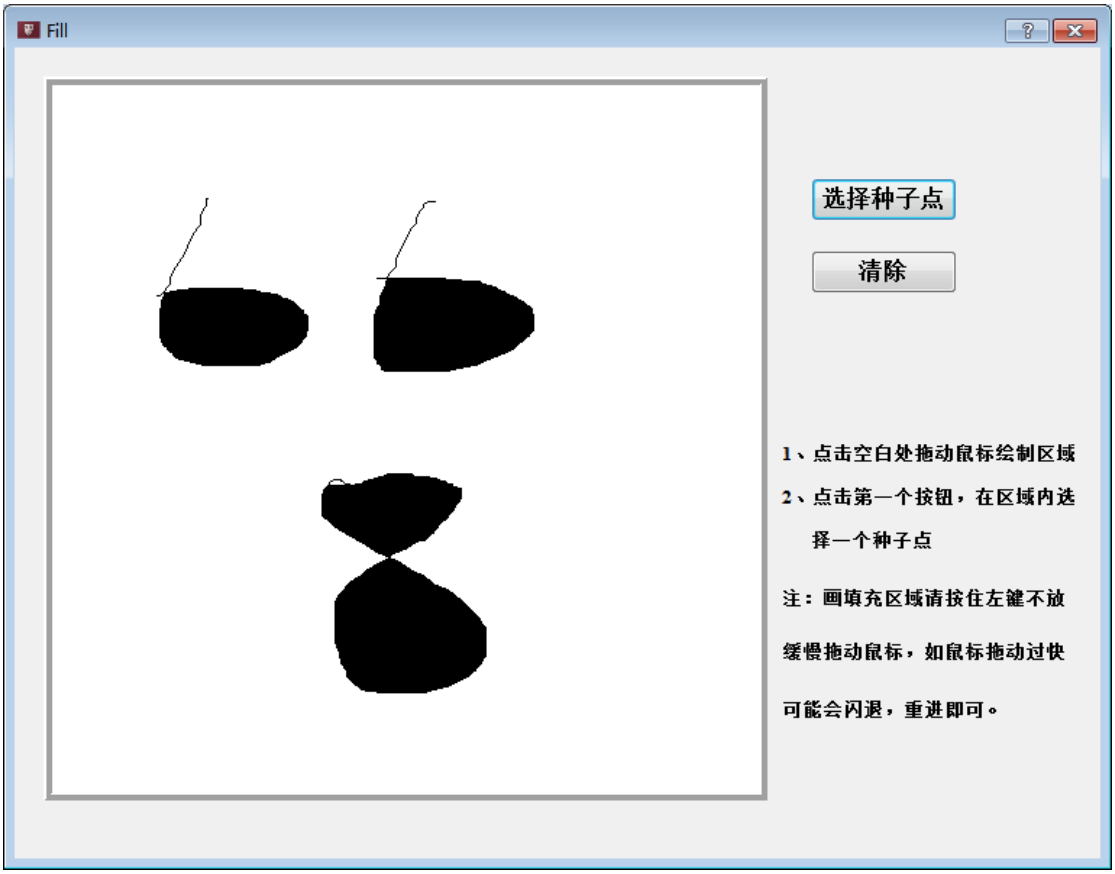


图八 区域填充算法流程图

区域填充算法的流程图如图八所示，其中区域绘制使用的是*mousePressEvent()*和 *mouseMoveEvent()*，首先 *mousePressEvent()*获取第一个点，记为 StartPoint，然后在 *mouseMoveEvent()*事件中，将当前坐标记为 EndPoint，在

StartPoint 和 Endpoint 两点间绘制直线，绘制完成后将 EndPoint 的值赋给 StartPoint，即不断地将上一时刻的终点值设为下一时刻的起点值。程序中定义了二维数组 image，该数组用来记录画板上各点的颜色索引值，若为白色，则索引记为 0，反之记为 1，在实验过程中，由于只有区域的边界出颜色为黑色，故出边界区域以外，颜色索引皆为 0，image 的行、列索引即画板上各点的坐标值。当选择种子点（x,y），首先判断当前点的颜色索引，若为白色则绘制该点，然后对（x,y）的四邻域点同样进行以上过程。若当前点的颜色索引为 1，则跳过该点，且不检查其周围点。

1.4.3 实验结果



图九 区域填充实验结果

实验结果如图九所示，实验过程如下：首先绘制填充区域（闭合），然后点击“选择种子点”按钮，在区域内单击，即可实现区域填充。本实验中，绘制了两个数字“6”，一个数字“8”，即产生四个封闭的填充区域，每次单击“选择种子点”按钮，以此将四个区域填充即可。

二、样条曲线的生成

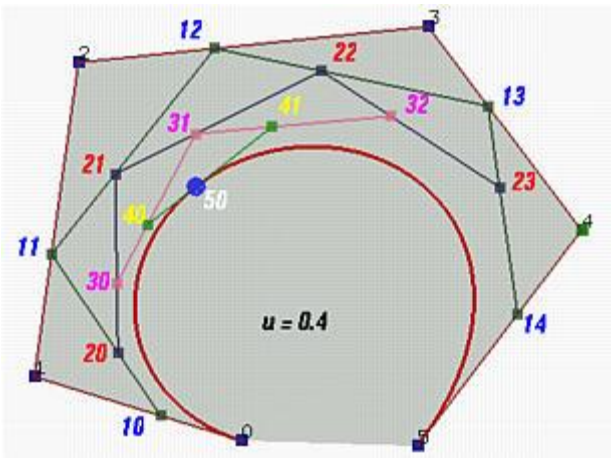
2.1 Bezier 曲线

2.1.1 基本原理

贝塞尔曲线(Bézier curve)，又称贝兹曲线或贝济埃曲线，是应用于二维图形应用程序的数学曲线。由于用计算机画图大部分时间是操作鼠标来掌握线条的路径，与手绘的感觉和效果有很大的差别。即使是一位精明的画师能轻松绘出各种图形，拿到鼠标想随心所欲的画图也不是一件容易的事。这一点是计算机万万不能代替手工的工作，所以到目前为止人们只能颇感无奈。使用贝塞尔工具画图很大程度上弥补了这一缺憾。贝塞尔曲线是计算机图形图像造型的基本工具，是图形造型运用得最多的基本线条之一。一般来说，贝塞尔曲线可以拟合任何数目的控制点，本设计实现了可拟合任意控制点的功能。

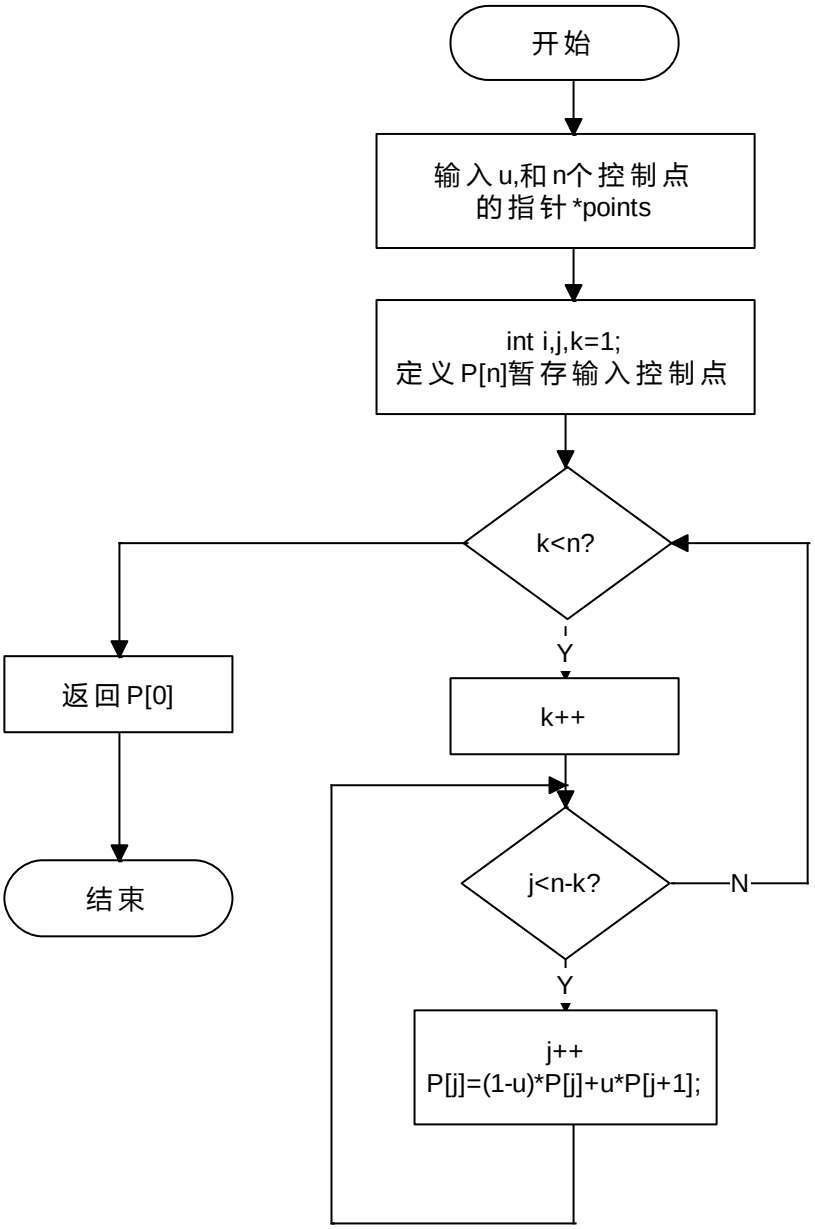
2.1.2 算法实现

本设计中，对贝塞尔曲线的绘制采用 DeCasteljau 算法，该算法是一个递归算法，可以快速得到贝塞尔曲线上的点，该算法的输入是 n 个控制点，输出是贝塞尔曲线上的一个点。DeCasteljau 算法的思想如下：为了计算 n 次贝塞尔曲线上的点 C(u), u∈[0,1]，首先将控制点连接形成一条折线 00-01-02……0(n - 1)-0n。利用上述方法，计算出折线中每条线段 0j-0(j+1)上的一个点 1j,使得点 1j 分该线段的比为 u:1-u。



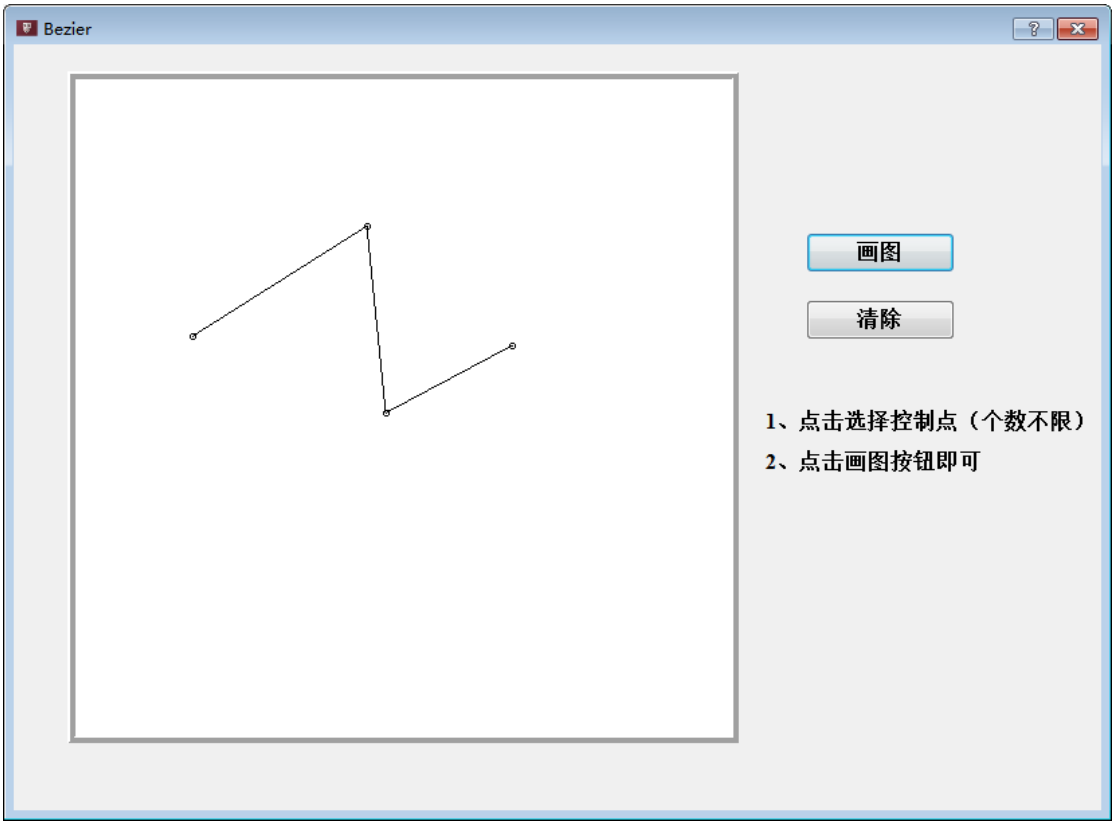
图十 DeCasteljau 算法示意图

然后在折线 10-11-·····-1(n-1)上递归调用该算法，以此类推。最终，求得最后一个点 n0。DeCasteljau 证明了，点 n0 一定是曲线上的点。如图十所示，曲线控制点是 00、01、02、03、04、05。线段 00-01 上取点 10，10 分该线段的比为 u:1-u，类似地取点 11、12、13、14，然后第二次迭代在线段 10-11 上取点 20，点 20 分该线段的比为 u:1-u，类似地取点 21、22、23。然后进行下一次迭代，依次类推，直到最后在线段 40-41 上取点 50，50 是最终惟一的点，也是在曲线上的点。对于 u∈[0,1]，若对其进行足够小的划分，便可得到足够多的贝塞尔曲线上的点，绘制到屏幕上看到的效果就是一条连续的曲线。该算法的流程图如图十一所示。



图十一 DeCasteljau 算法的流程图

程序中，通过 *mousePressEvent()*获取当前鼠标点击的点的坐标，并存在 points 结构体中，该结构体的成员是一个 QPoint point[20]数组，每个成员可保存 20 个点的信息，若实际中需要更多点可以再添加。每次点击一个点后，在该点处绘制一个小椭圆，用来标注该点，如果当前点不是第一个点，还会将当前点和上一个点以直线的方式连接起来。（如图十二所示）

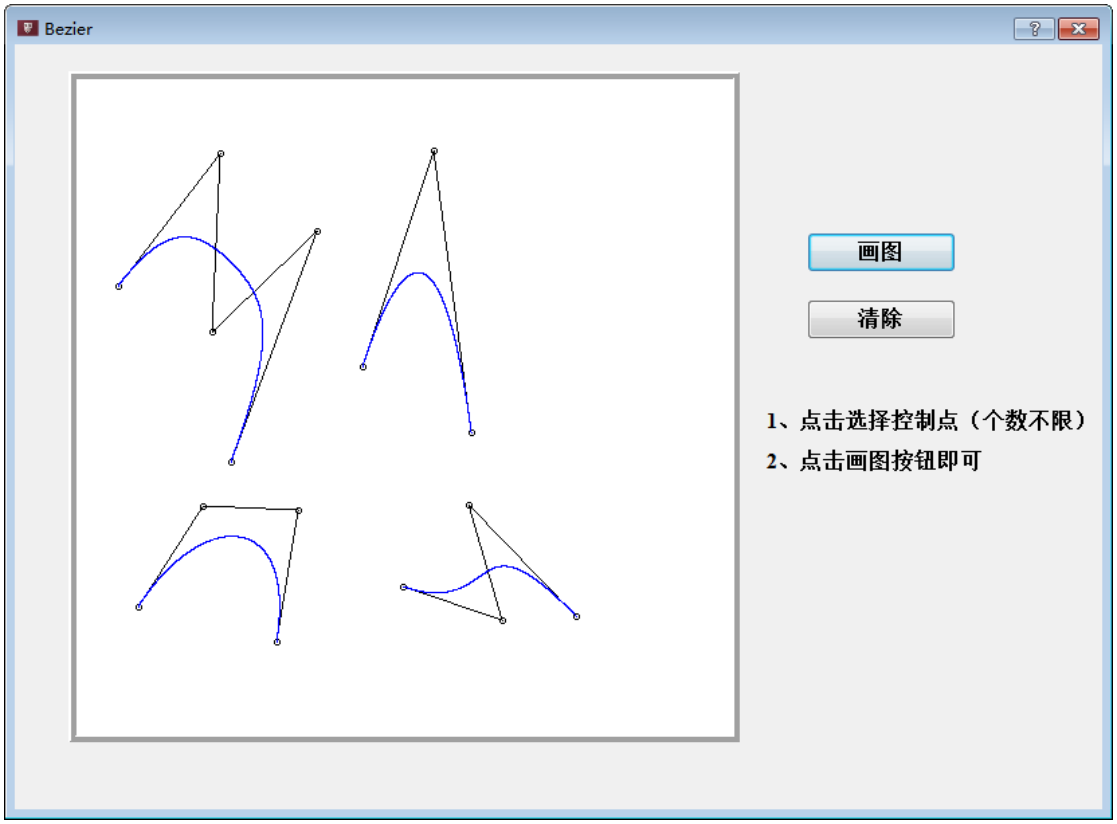


图十二 控制点绘制方式

得到各控制点坐标后，将 u（程序中为 t）在(0,1)范围内等分，然后每次传入一个 u 值和绘制的 n 个控制点的坐标，得到当前 u 值下贝塞尔曲线上的一个点，并将给点画出来，如此循环，直到 u=1。绘制完成后，将控制点的计数器清零，以便于继续下一次绘制。

2.1.3 实验结果

实验结果如图十二所示，本实验中，首先用鼠标在白板处选取若干个控制点，选取完成后单击“画图”按钮即可实现画图功能，每次绘制完成后，可以不清除继续绘制。由图十二可知，该算法效果很好，生成的曲线很优美。



图十二 Bezier 曲线实验结果

2.2 B 样条的生成

2.2.1 基本原理

B 样条曲线具有两个 Bezier 曲线不具有的优点：（1）B 样条多项式的次数独立于控制点的数目，（2）B 样条允许局部控制曲线或曲面。本设计中，主要绘制的是三次周期性 B 样条，通过 B 样条的通用公式，可将三次周期 B 样条以矩阵的形式写出。

$$\mathrm{P}(u) = [u^3\ u^2\ \ u\ \ 1] \cdot M_B \cdot [P_0\ \ P_1\ P_2\ P_3]^T$$

其中 u∈[0,1], P_0 , P_1 , P_2 , P_3为四个控制点，M_B是周期性三次多项式的 B-样条矩阵。M_B如下所示。

$$M_B = \frac{1}{6}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}$$

2.2.2 实现算法

对于 B 样条的生成，其算法非常简单，只需要输入四个控制点，并将四个控制点按照其生成公式计算即可。此处只对程序进行解释，并介绍多于四个控制点的绘制方法。首先，对于控制点的坐标获取方式，和 Bezier 曲线中相同，此处不再赘述。当控制点大于 4 个时，不妨假设为 6 个，那么首先绘制前四个，然后将第二个点视为第一个点，以此类推，第五个点视为第四个点，绘制这四个点的 B 样条曲线，然后继续后移，直至所有点绘制完成。由于 d-1 次 B 样条曲线具有C^{d-2}连续性，因此曲线处处连续且光滑。具体到程序中，做法如下：

（1）定义一个 points 结构体，其成员为 point[20]数组，用以存储鼠标点击的控制点。定义一个 QPointF 数组 a[4]用以存储当前需要绘制的四个点。

（2）假设选取了 n 个控制点，定义一个计数器 j，将 j 初始化为 0，对于 j<=n-4 执行以下计算：

a[0] = $f_1(u)$*points.point[j];

a[1] = $f_2(u)$*points.point[j+1];

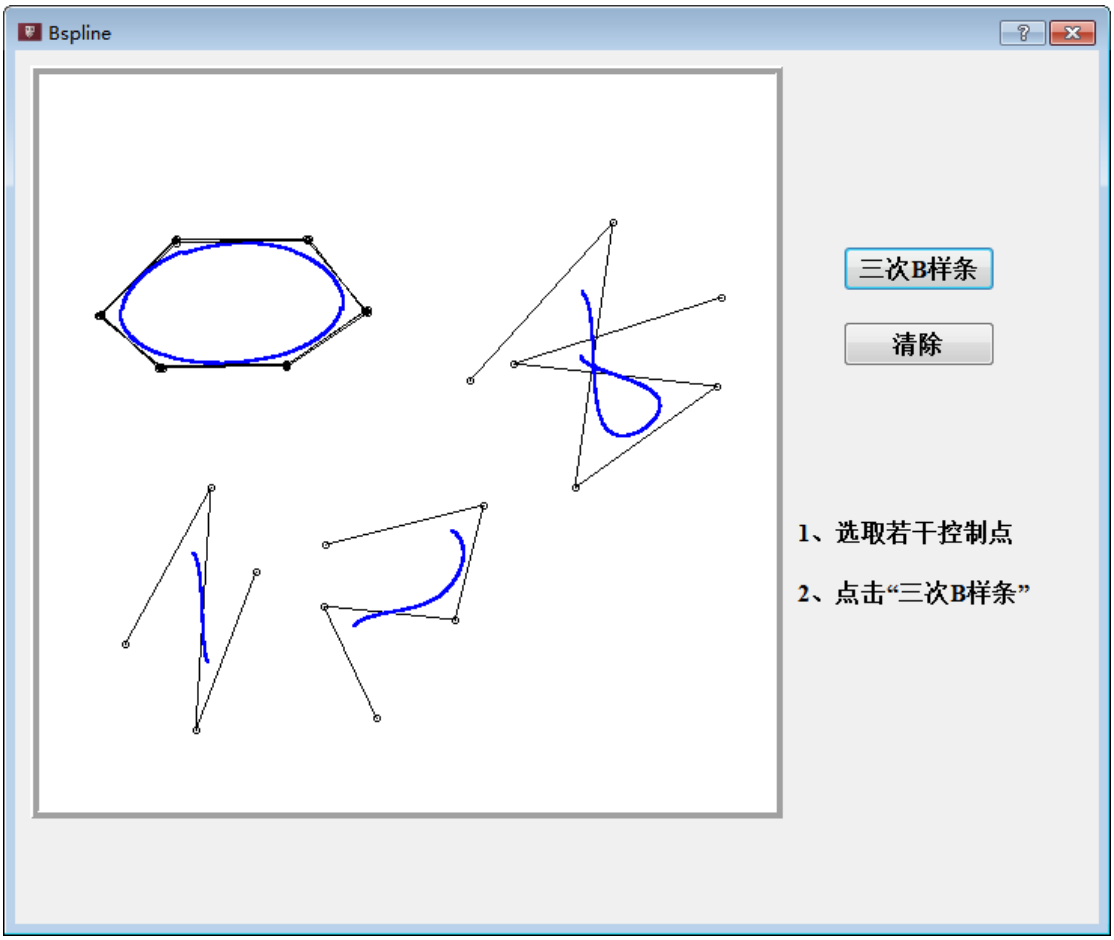
a[2] = $f_3(u)$*points.point[j+2];

a[3] = $f_4(u)$*points.point[j+3];

计算完成后，绘制四个点，然后 j=j+1，如此每次移动一个点，实现了绘制多于四个控制点的三次周期性 B 样条曲线的绘制。

2.2.3 实验结果

实验结果如图十三所示，其中左上角为通过循环选取六个控制点中的四个生成的，其余为分别选取 4、5、6 个控制点生成的。实验过程如下：先选取若干个点，选取完成后点击“三次 B 样条”按钮生成 B 样条曲线，本实验同样不需要清除，可继续在已经画了 B 样条的画板上继续绘制。由图十三可见，本设计生成的 B 样条曲线是很光滑的。



图十三 B 样条曲线实验结果

三、分形图形的生成

3.1 Koch 曲线的生成

3.1.1 基本原理

Koch 曲线是这样一条曲线，对于一条直线，将其三等分，然后去掉中间的那 1/3，以一个只有腰而没有底边的等腰三角形代替，如图十四所示，对于生成的这条曲线中的每一段（共四段），再次进行上述操作，得出如图十五所示曲线。其中图十四经过一次迭代，图十五经过两次迭代。以此类推，经过若干次迭代后，

会生成更复杂的图案。

图十四 一次迭代的 Koch 曲线

图十五 两次迭代的 Koch 曲线

3.1.2 实现算法

对于 Koch 曲线，首先观察其第一次迭代过程，第一次迭代时，已知两个点，利用几何知识可以计算出新出现的三个点的坐标，然后对于其中的每一段，重复这个过程。具体计算过程如下：

（1） 输入两点 point1，point2，迭代次数 n。

（2） 计算 dx=point2.x-point1.x,dy=point2.y-point1.y；

（3） dx1=dx/3.0,dy1=dy/3.0,dx2=2*dx/3.0,dy2=2*dy/3.0;

（4） 定义点 P1,P2,P3（即生成的三个点，其中 P3 为中间的点）；

（5） P1.x=point1.x+dx1,P1.y=point1.y+dy1,P2.x=point1.x+dx2,P2.y=point1.y+dy2；

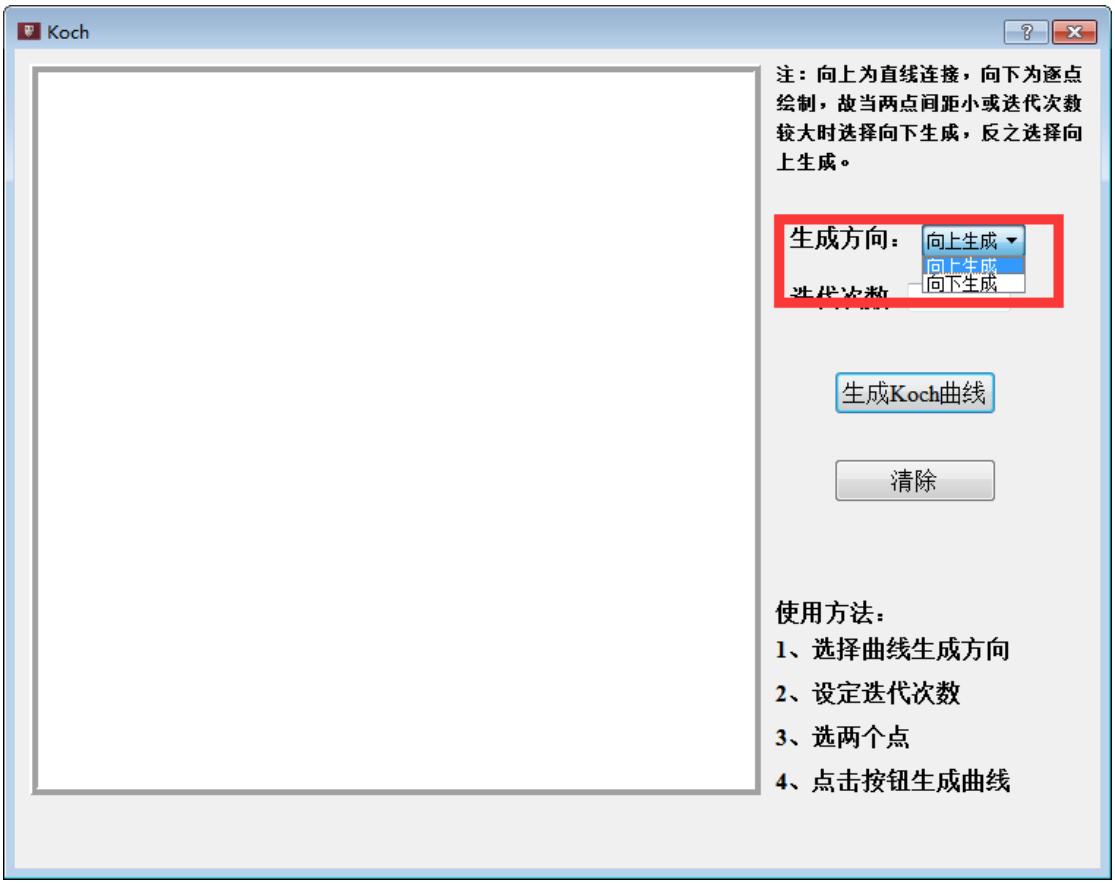
（6） mid_dx=0.5*dx,mid_dy=0.5*dy,dx3=(sqrt(3.0)/6.0)*(-dy) dy3=(sqrt(3.0)/6.0)*(dx),mid_pont.x=point1.x+mid_dx,mid_point.y=point1.y+mid_dy,P3.x=mid_point.x+dx3,P3.y=mid_point+dy3;

（7） 相邻的点递归，并每次将最后面的点保存，当 n=0 时结束递归。

实际程序中，把以上（1）-（6）过程包装成一个函数 KochCalculate(QPointF point1,QPointF point2 ,int n)，则(7)可写成如下形式：

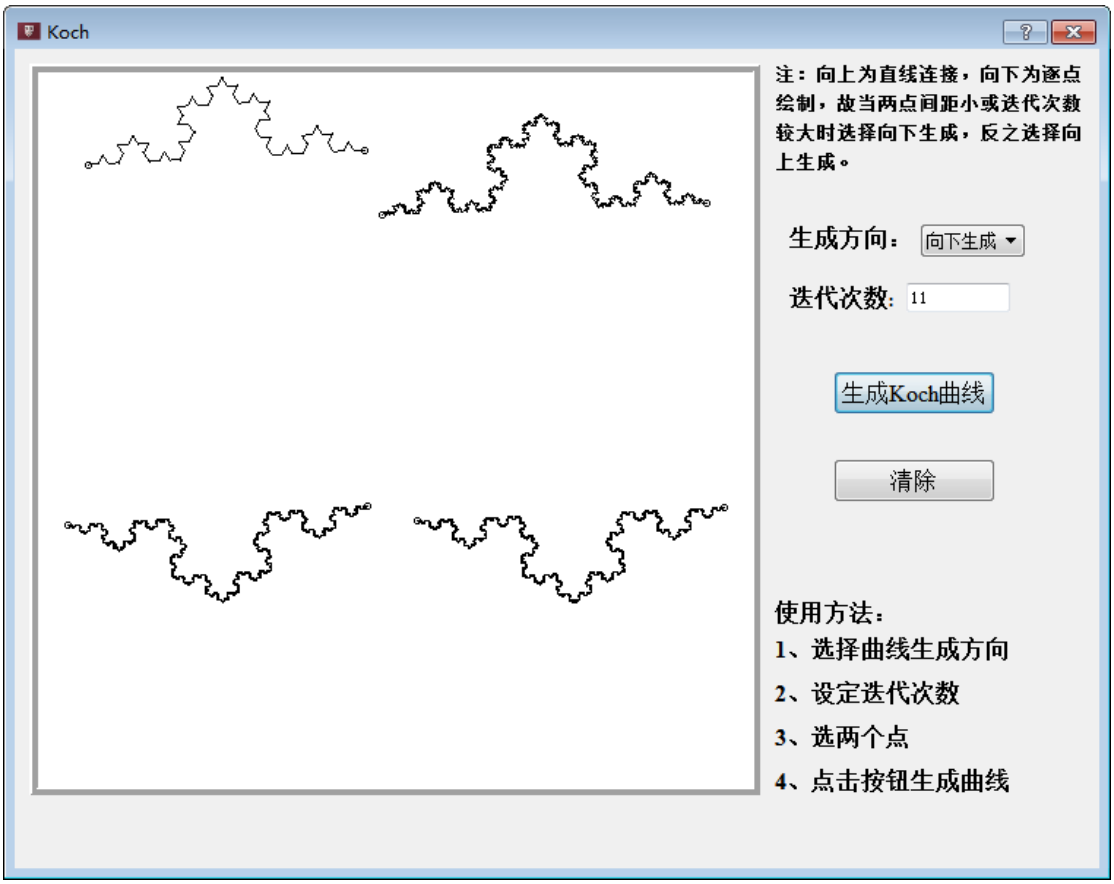
```
KochCalculate(point1,P1,n-1);
pointarray.push_back(P1);
KochCalculate(P1,P3,n-1);
pointarray.push_back(P3);
KochCalculate(P3,P2,n-1);
pointarray.push_back(P2);
KochCalculate(P2,point2,n-1);
pointarray.push_back(point2);
```

其中 pointarray 为一个 vector，用以保存生成的所有 Koch 曲线上的点，本设计精心设计了两种 Koch 曲线绘制方式，一种针对与迭代次数少，当迭代次数少时，生成的点很少，看起来是一些离散的点，为此，将这类情况采用两点间用直线连接的方法，从而时曲线看起来美观；另外一种是针对迭代次数较多的情况，在这种情况下，生成的点很少，点与点之间的间距已经足够小了，再用直线连接就没必要了。在本设计中，第一种情况设计成向上生成，第二中情况设计为向下生成，有一个下拉菜单可以切换两种情况，如图十六所示。



图十六 生成方式的选择

3.1.3 实验结果



图十七 Koch 曲线实验结果

实验结果如图十七所示，其中左上角为向上生成 3 次迭代的结果，右上角为向上生成 5 次迭代的结果，左下角为向下生成 10 次迭代的结果，右下角为向下生成 11 次迭代的结果。实验发现，在我实验室的 PC（4GB）上，12 次迭代软件就会出错。由此可见，Koch 曲线的计算量很大，但是由图十七也可以看出 Koch 曲线是一种很优美的曲线。

3.2 Julia 集的生成

3.2.1 基本原理

Julia 集合是一个在复平面上形成分形的点的集合。以法国数学家加斯顿·朱利亚（Gaston Julia）的名字命名。Julia 集合可由下式反复迭代生成：

$$f(z) = z^2 + c$$

在 Julia 集中，c 是固定的值，取某一 z 的初始值，按照上式不断迭代，可生成一个序列，由于 z 的初始值的不同，该序列可能发散也可能收敛，而发散的序列可能经过不同的迭代次数迭代，数列收敛或者到达发散时的不同迭代次数的差异，在复平面上对应不同的区域，对不同的区域添加不同的颜色，便得到了缤纷多彩的 Julia 集图案了。对于复数 z，若|z|>2，都会有$|z^2|=|z|^2>2*|z|$，也就是说，对这样的 z 进行迭代，一定会发散，因此本程序中把 z 的实部和虚部的取值范围设为[-1.5，1.5]。

3.2.2 实现算法

对于复数z_n=a+bi，c=d+ei，在实际编程实现中，做以下处理即可得出$f(z)=z^2+c$的更新法则。

（1） 令zx_n = a，zy_n = b，cx = d，cy = e；

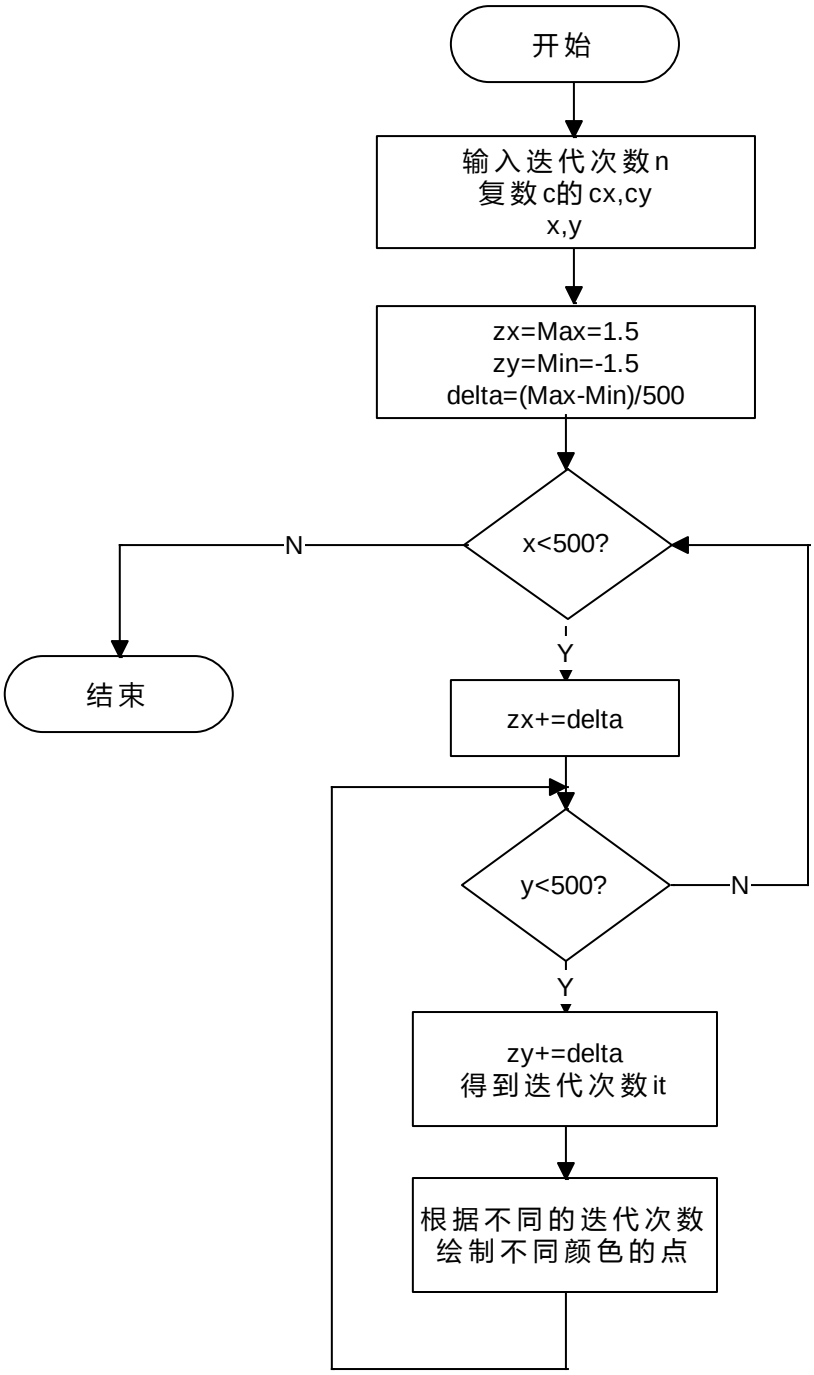
（2） 则对于z_n可通过下式子得到z_{n+1}

$$\begin{cases} zx_{n+1} = {zx_n}^2 - {zy_n}^2 + cx \\ zy_{n+1} = 2zx_n zy_n + cy \end{cases}$$

（3） 每经过一次迭代，迭代计数器 it 加 1，退出迭代条件为|z|>2 或达到最大迭代次数，以上过程返回迭代次数。

通过这种数学推导，可以避免复数的运算，在程序中也无需定义复数类型，而只需使用常用的浮点型即可完成 Julia 集的计算。

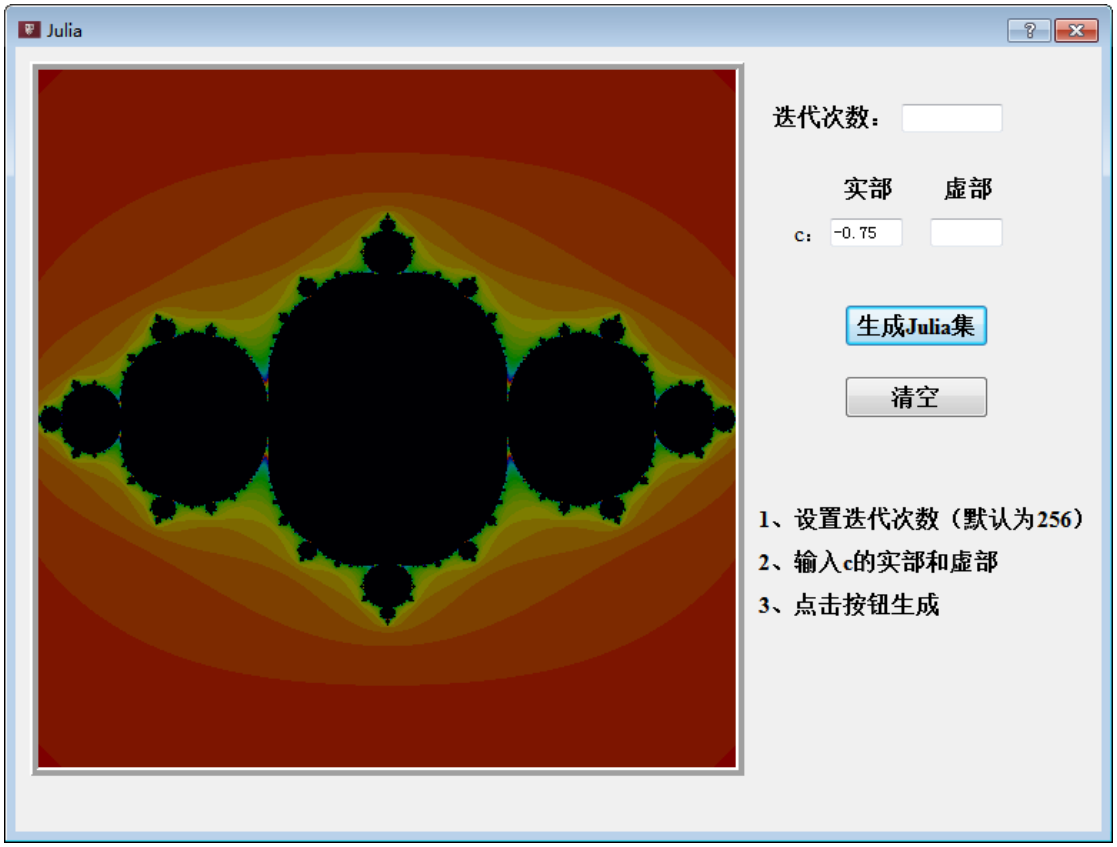
Julia 集的绘制算法如图十八所示，其中 cx,cy 的值由 LineEdit 控件输入，对于 z，将整个画板（500*500），在[-1.5，1.5]范围内对 zx 和 zy 均分成 500 份，对于不同的 zx，zy 组合计算其迭代次数，根据返回的迭代次数，对不同的点给以不同的颜色，最后得到美丽的 Julia 集图案。



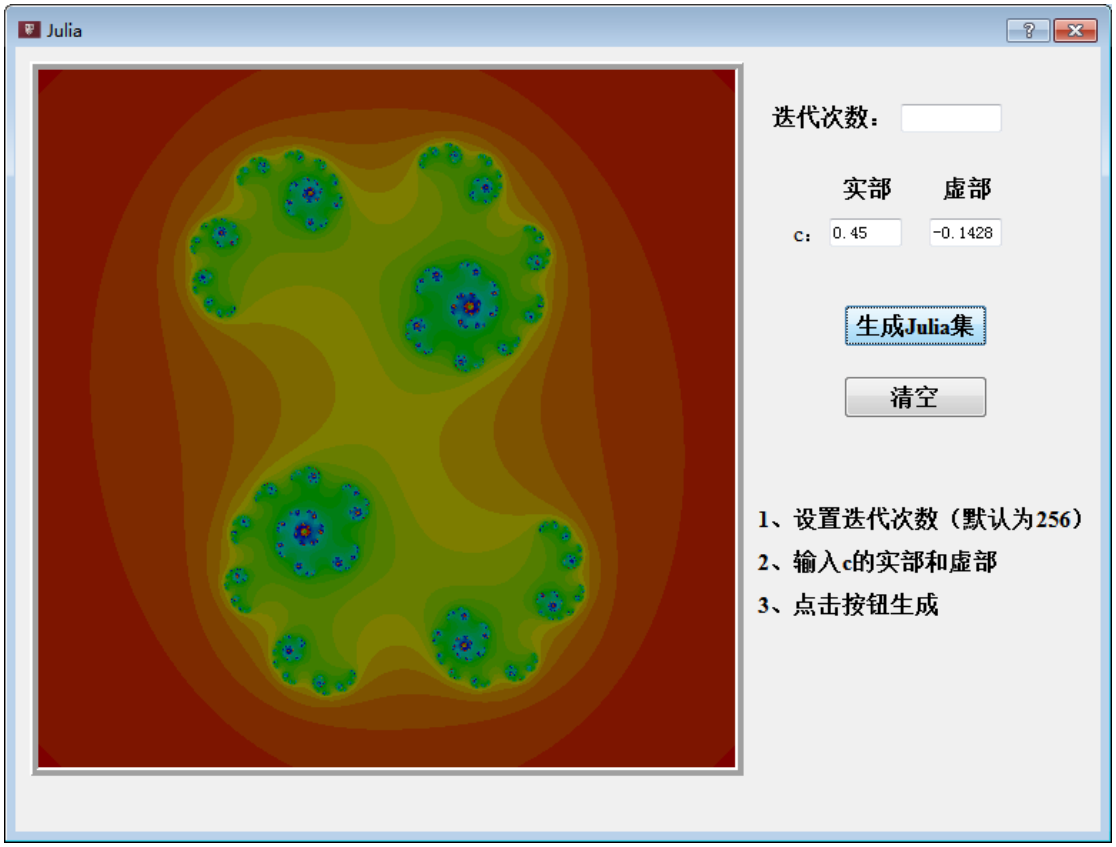
图十八 Julia 集生成算法流程图

3.2.3 实验结果

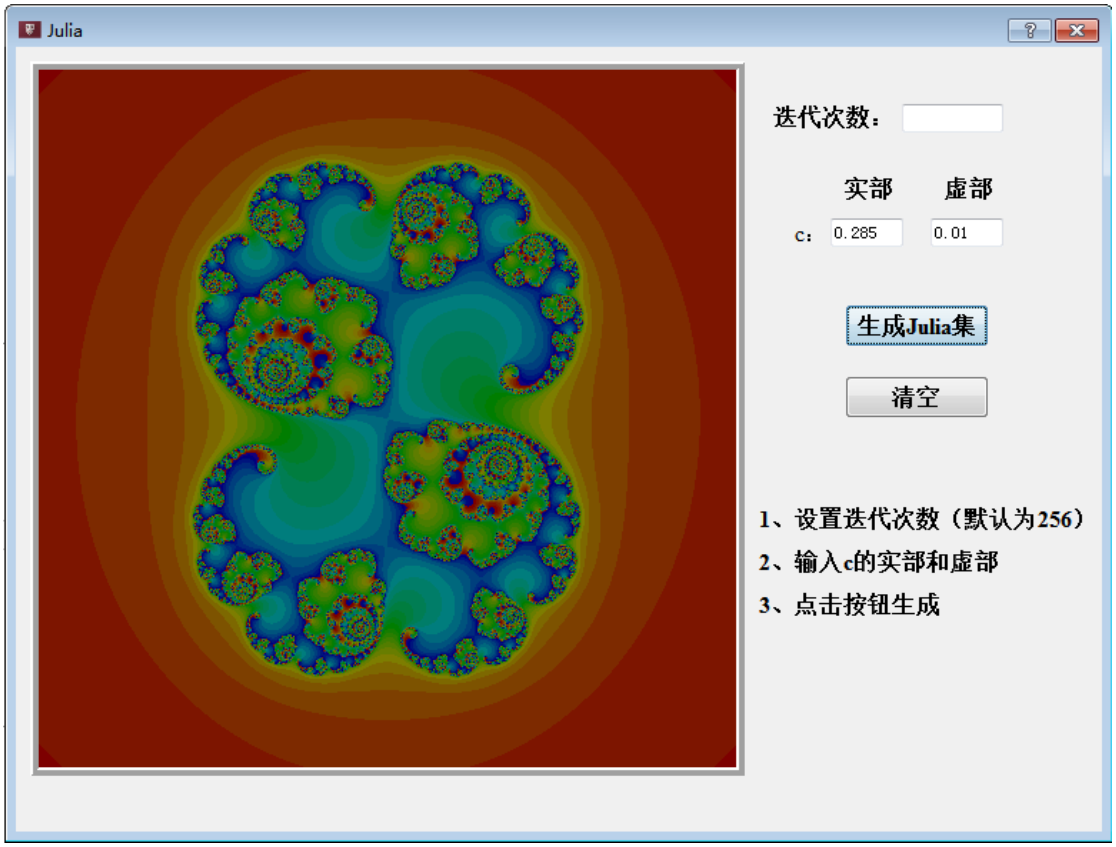
如图十九至图二十四为不同的 c 取值下 Julia 图案，由这些图可见，Julia 集生成的图案很漂亮。本程序中，采用的是 HSV 色彩空间绘制的图案，故其色彩更加绚丽。



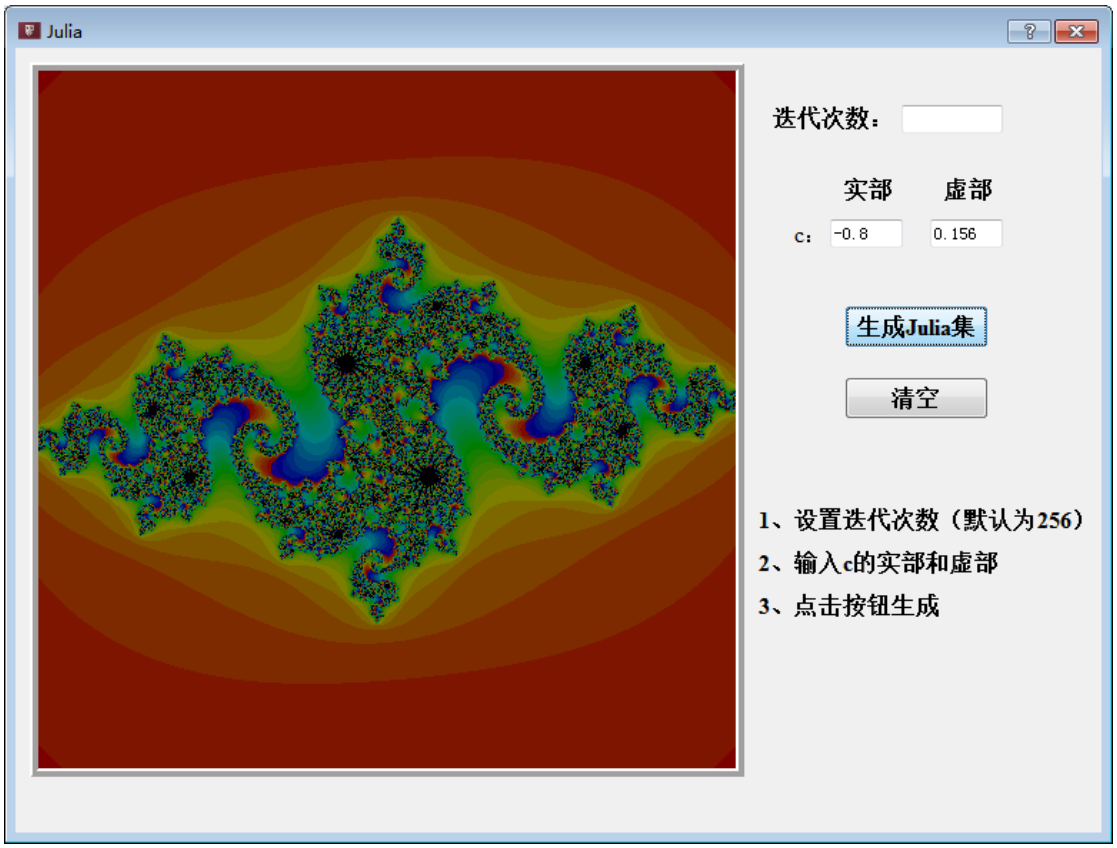
图十九 c=-0.75 生成的 Julia 集图案



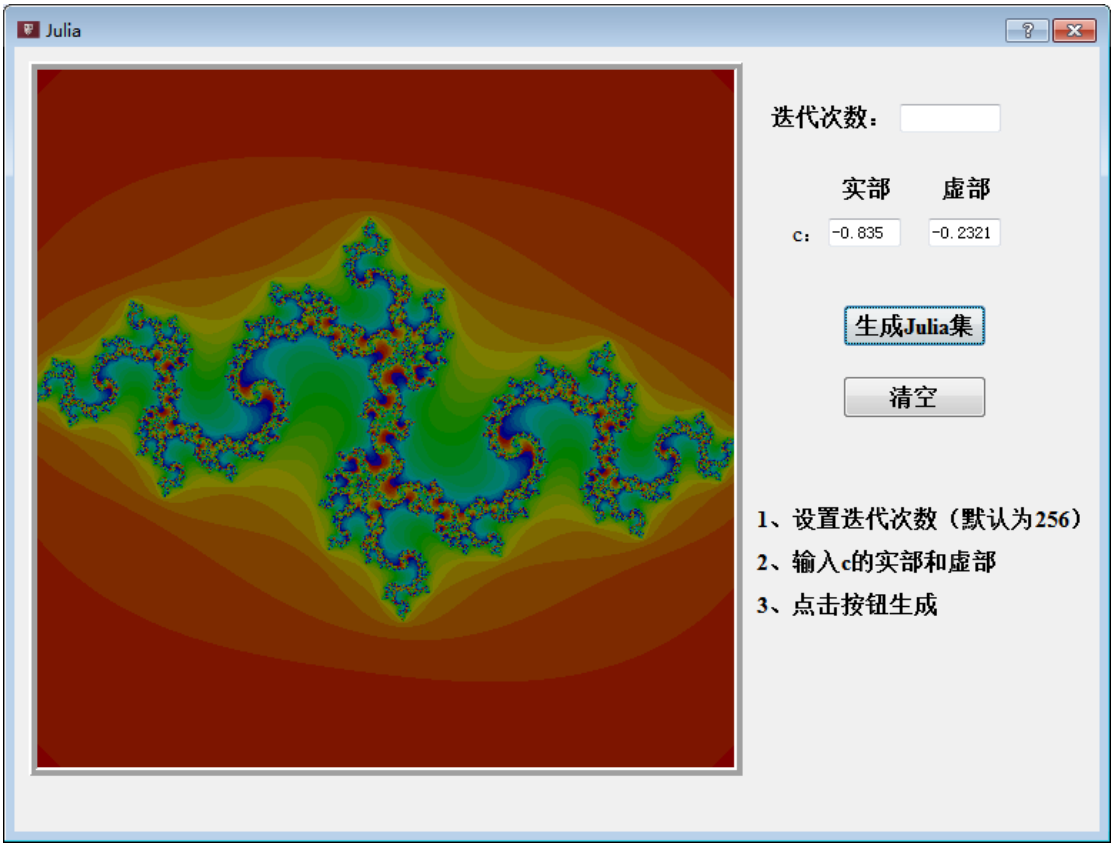
图二十 c=0.45-0.1428i 生成的 Julia 集图案



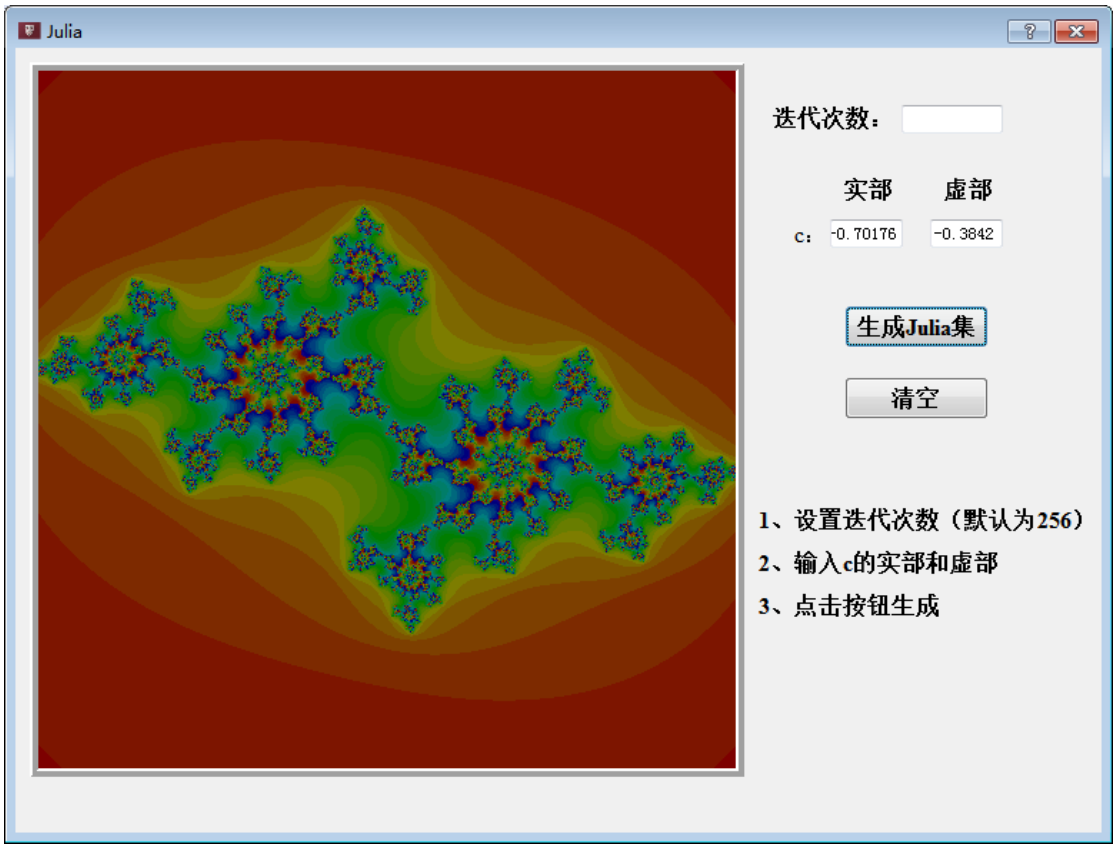
图二十一 c=0.285-0.01i 生成的 Julia 集图案



图二十二 c=-0.8+0.156i 生成的 Julia 集图案



图二十三 c=-0.835-0.2321i 生成的 Julia 集图案



图二十四 c=-0.70176-0.3842 生成的 Julia 集图案

3.3 Mandelbrot 集的生成

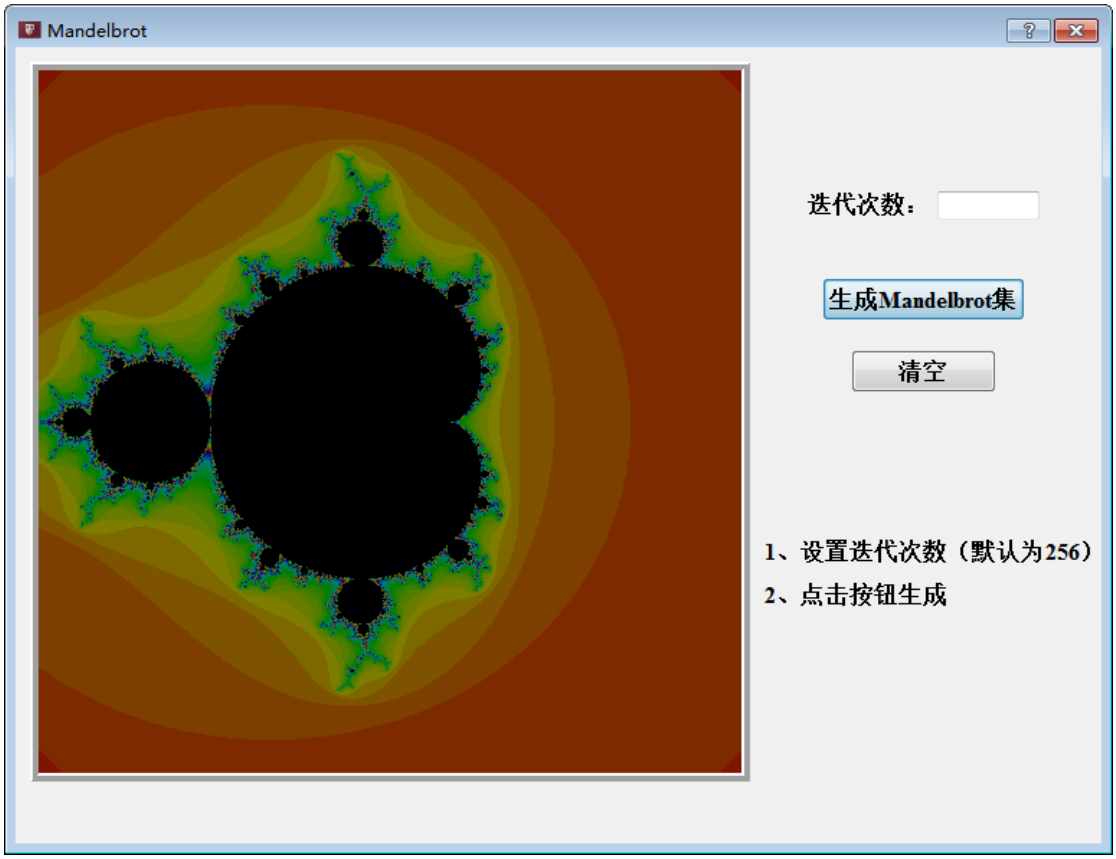
3.3.1 基本原理

Mandelbrot 集的基本原理和 Julia 集相同，只是在 Mandelbrot 集中，是固定 z0 的，而改变的是 c，即对于给定的 z0，cx,cy 从最小变为最大，在此过程中，不断更新$f(z) = z^2 + c$，同样可以得到不同的发散次数。

3.3.2 生成算法

Mandelbrot 集的生成算法和 Julia 集基本一致，其流程图和 Julia 集流程图只有一点小差异，即将 Julia 集中的 zx+=delta,zy+=delta 改成 cx+=delta,cy+=delta 即可，故其流程图在此不再绘制，直接给出实验结果。

3.3.3 实验结果



图二十五 Mandelbrot 集实验结果

3.4 蕨类植物的生成

3.4.1 基本原理

对于蕨类植物，采用仿射变换法构造，其过程中各参数如下：

i	a	b	c	d	e	f	P
1	0	0	0	0.16	0	0	0.01
2	0.85	0.04	-0.04	0.85	0	1.6	0.85
3	0.2	-0.25	0.23	0.22	0	1.6	0.07
4	-0.15	0.28	0.26	0.24	0	0.44	0.07

表中 a,b,c,d,e,f 为仿射变换中的参数，每个参数有四种取值，P 为 i 取 1，2，3，4 的概率。仿射变换过程如下式：

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x_k \\ y_k \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix}$$

由上式可知，迭代次数的不同，获得的点数也不同，最终绘制出来的蕨类植物的疏密程度也不同。

3.4.2 算法实现

蕨类植物生成的算法非常非常简单，就是不断迭代以上公式，对于概率的生成，采用以下方式：

（1） 随机生成一个 0-100 的数 Num

（2） 若 Num<1，i=1,即 P=0.01；

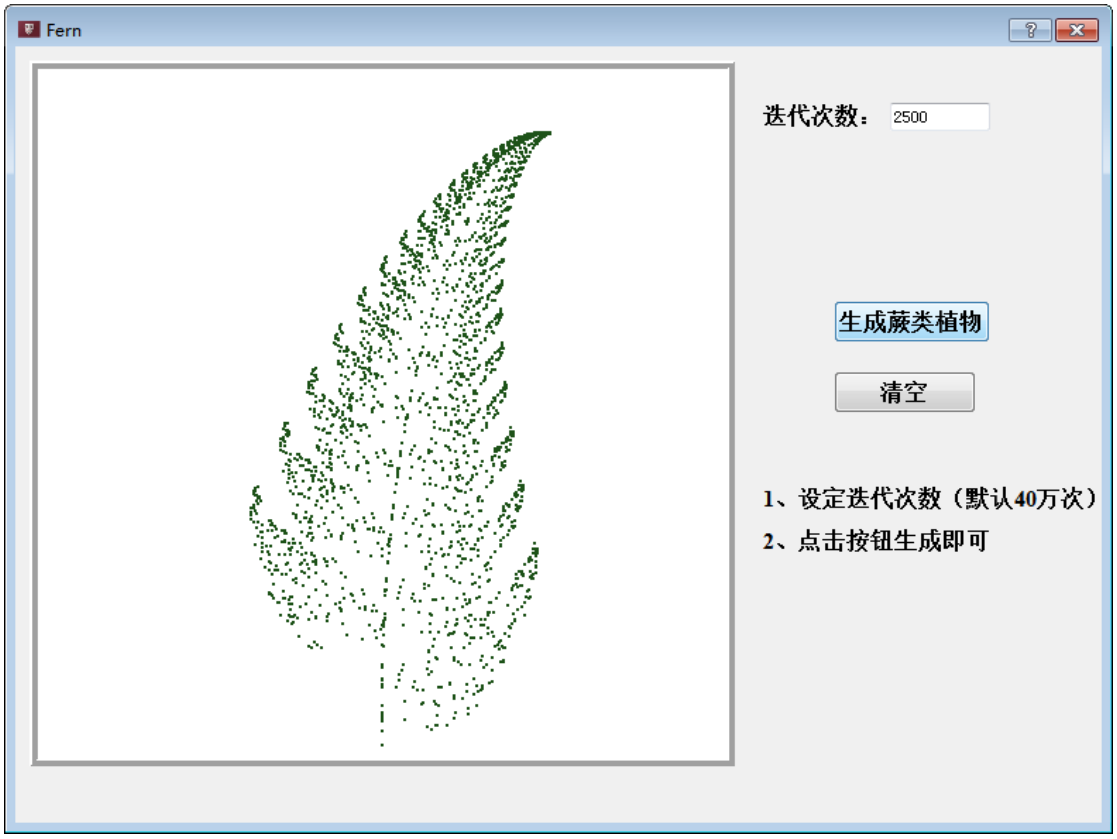
（3） 若 1<=Num<8,i=3, 即 P=0.07;

（4） 若 8<=Num<15,i=4, 即 P=0.07;

（5） 若 Num>=15,i=2, 即 P=0.85;

当给定迭代次数后，在迭代次数内，不断生成随机数，并根据 i 取值的不同，采用不同的 a,b,c,d,e,f 计算下一次的坐标，每次计算完成后都画点。在实验中发现，x 的取值范围大致为[-2.5~2.5]，y 的取值范围大致为[0~10]，因此对 x,y 进行一个简单的数学变换即可将其映射到[0,500]的范围中，即可完美地在画板中显示。

3.4.3 实验结果



图二十六 迭代次数为 2500 次生成的蕨类植物

图二十七 迭代次数为 10000 次生成的蕨类植物

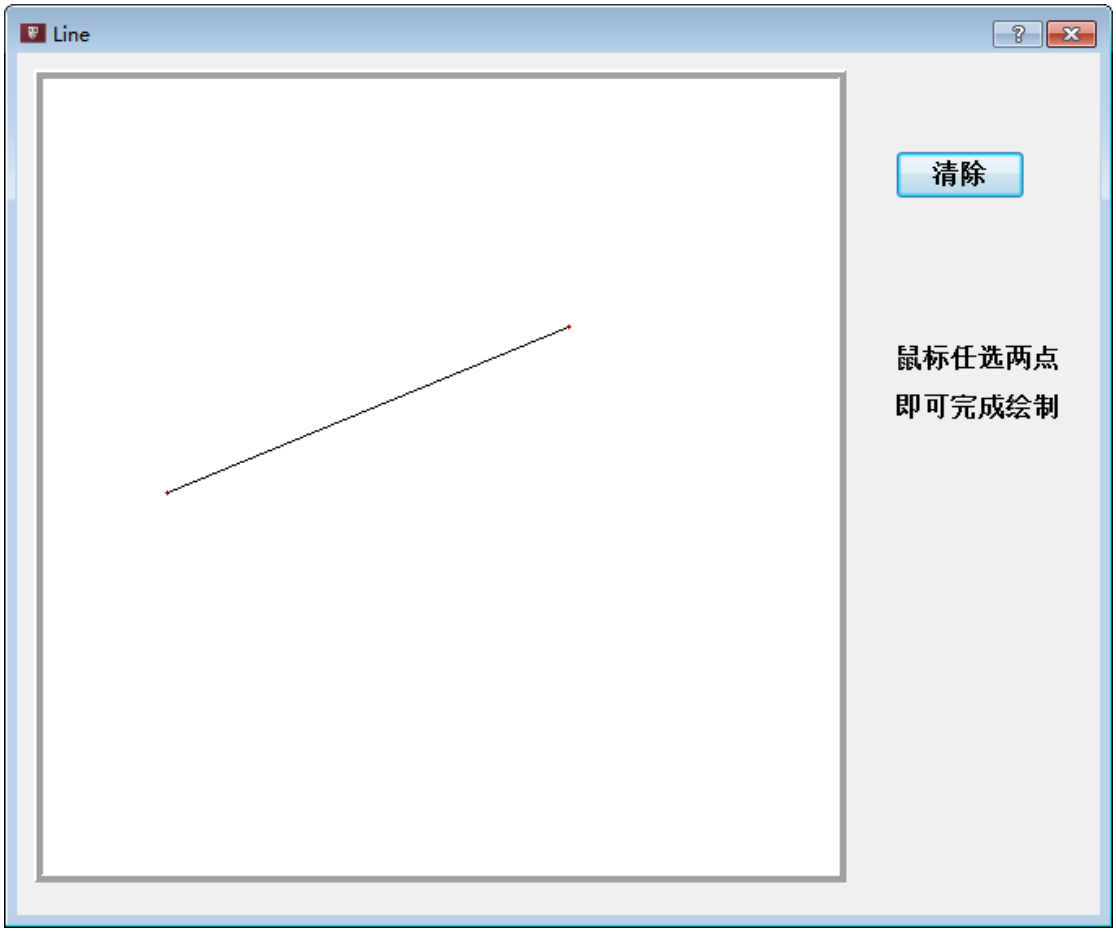
图二十八 迭代次数为 100000 次生成的蕨类植物

图二十九 迭代次数为 400000 次生成的蕨类植物

由以上实验可见，迭代次数越多，生成点越多，蕨类植物就越逼真。

四、软件使用说明

4.1 直线



对于直线，只需要任选两点，即完成直线的绘制，可重复绘制，按清除按钮清空画板。

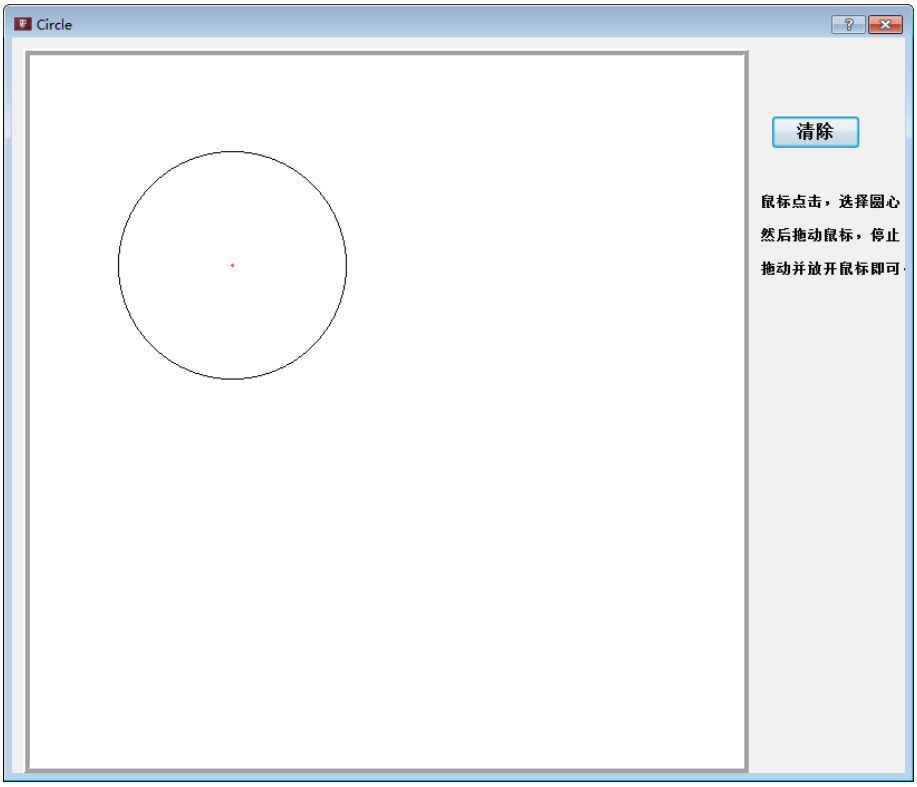
4.2 圆

对于圆的生成，分为两步：

（1） 先点击鼠标左键，选择圆心，并按住鼠标左键不放。

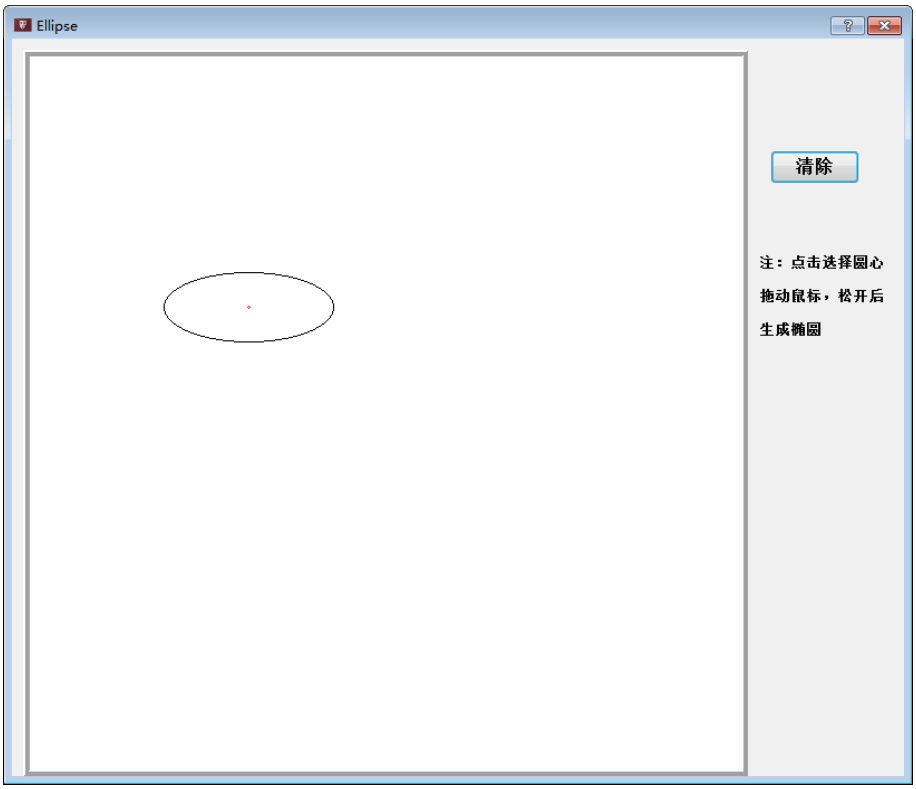
（2） 拖动鼠标，在合适的位置松开，松开后自动完成圆的绘制。

注意：由于画板尺寸有限，不要选择太大的半径。



4.3 椭圆

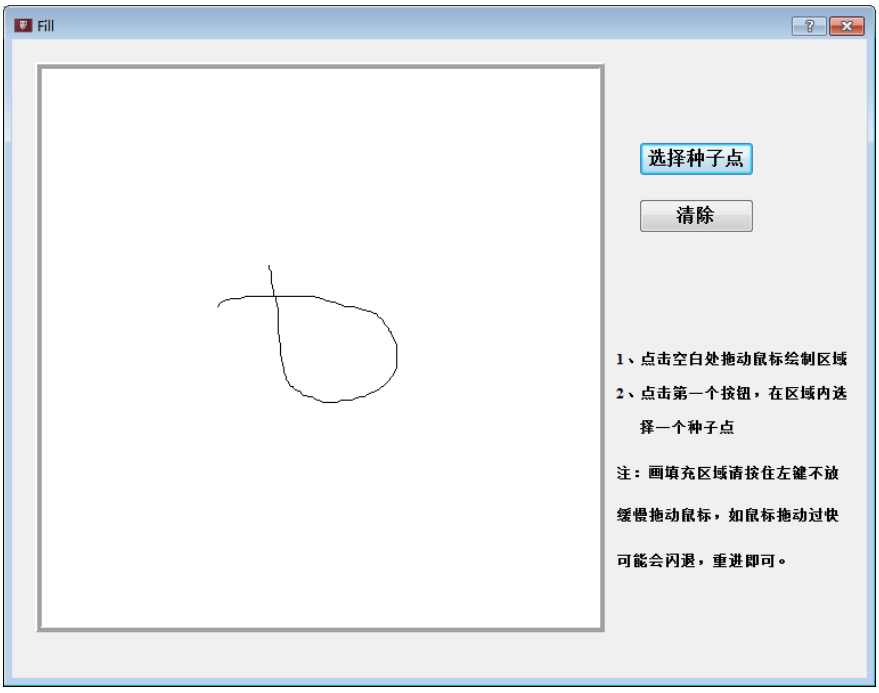
椭圆的生成和圆的生成类似，此处不在赘述。



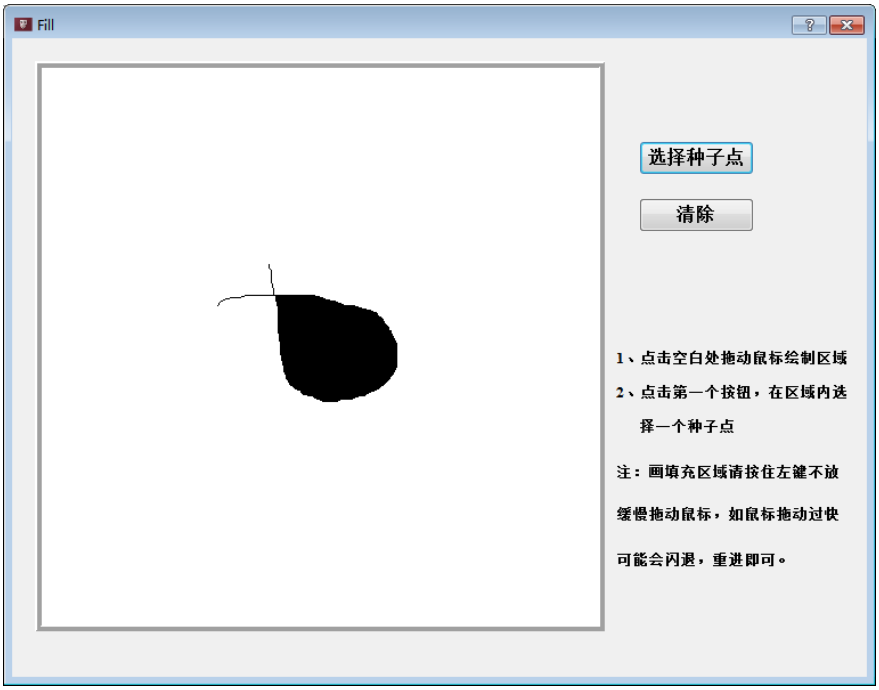
4.4 区域填充

区域填充步骤如下：

（1） 在空白画板处绘制填充区域。



（2） 点击“选择种子点”按钮，此时鼠标变为十字形，然后在区域内点击即可。

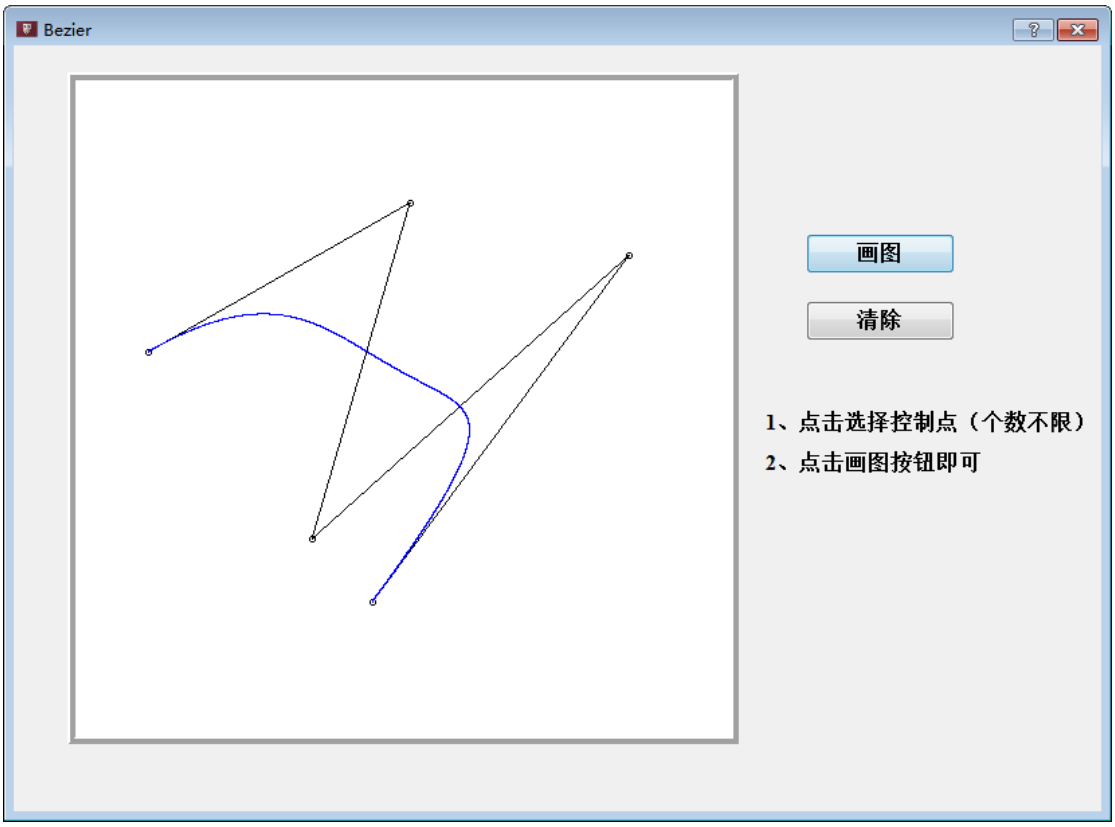


4.5 Bezier 曲线

Bezier 曲线使用说明如下：

（1） 选取若干个控制点。

（2） 点击“画图”按钮即可。



4.6 B 样条曲线

B 样条曲线和 Bezier 曲线使用方法相同，此处不再赘述。

4.7 Koch 曲线

Koch 曲线使用方法如下：

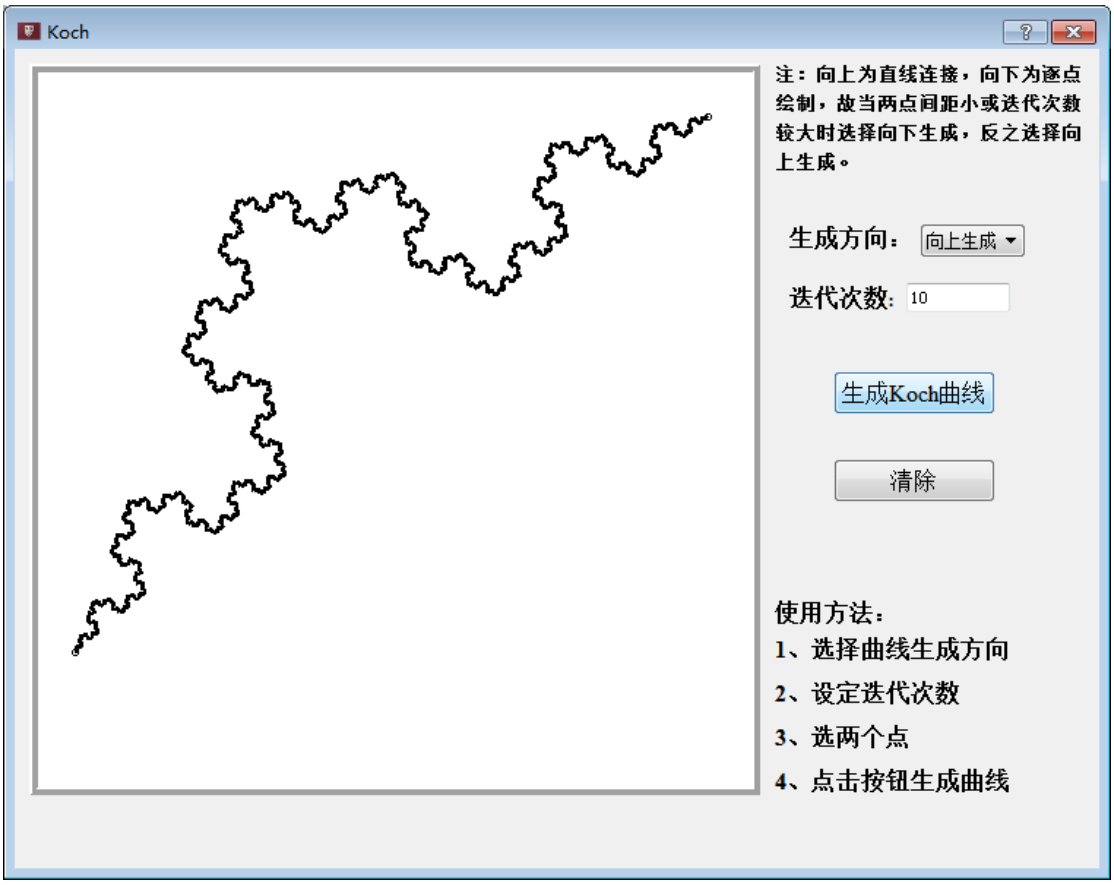
（1）选择生成方向（上或者下）

（2）设定迭代次数。

（3）选取两个点。

（4）点击“生成 Koch 曲线”按钮即可生成。

注：在 Koch 曲线中，迭代次数不宜过大，当向上生成时，若间距不大，不宜超过 10 次，在向下生成时，实验发现当迭代次数超过 12 次时软件会报错。



4.8 Julia 集

Julia 集使用说明如下：

（1）设定迭代次数（默认为 256 次）。

（2）设定 c 的实部和虚部。

（3）点击“生成 Julia 集”按钮即可。

4.9 Mandebrot 集

Mandelbrot 集使用说明如下：

（1）设定迭代次数（默认为 256 次）。

（2）点击“生成 Mandelbrot 集”按钮即可。

4.10 蕨类植物

蕨类植物使用说明如下：

（1）设定迭代次数（默认为 40 万次）

（2）点击 “生成蕨类植物”按钮即可。

五、感言

本次课程设计前前后后花了半个月左右，几乎每天都在做，由于没有接触过Qt，且 c++水平很 Low,期间遇到很多问题，通过百度、查书、问人等方法一个个解决了，代码也修改了很多次，虽然说用了很长时间，但是很享受这个过程，当看到一个个图形、图案被自己画出来的时候，那种欣慰和满足还是很令人兴奋的。

六、参考文献

[1]. 计算机图形学（第二版）/（美）赫恩(Hearn, D.)等著；蔡士杰等译. —2 版。—北京：电子工业出版社，2002.5

[2]. Qt5 开发及实例/陆文周主编/—2 版.—北京：电子工业出版社，2015.5

[3]. 计算机图形学实践教程：Visual C++版/孔令德著.—2 版.—北京：清华大学出版社，2013.3