上海大学

Shanghai University

《计算机图形学》

上机实验报告

（电子版）

组长：　何琪辰（06122477）

组员：　叶剑峰（06122491）

黄亦琦（06122473）

学院：　计算机工程与科学学院

时间：　2008 学年冬季学期

指导老师：　丁友东

概述

完成实验的程序共有三个文件，1.exe ，FractalArt.exe，VRMLReader.exe

实验 1-3

编译环境：Qt4
运行环境：Qt4
操作系统：跨平台
程序主界面如图 1所示。

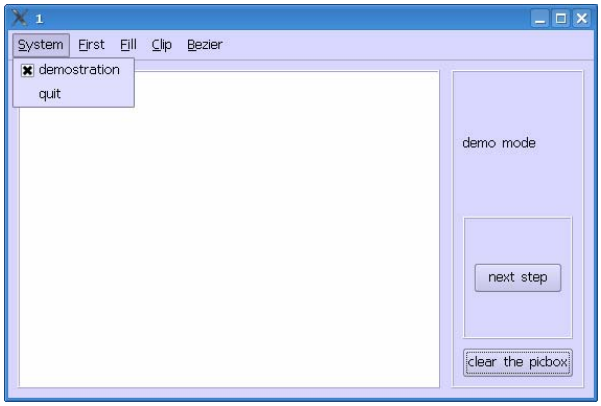

图 1 实验 1-3 程序主界面

实验 4-5

编译环境：Visual Studio 2005 C#
运行环境：.net Framework 2005
操作系统：Windows 系列
程序主界面如图 2所示。

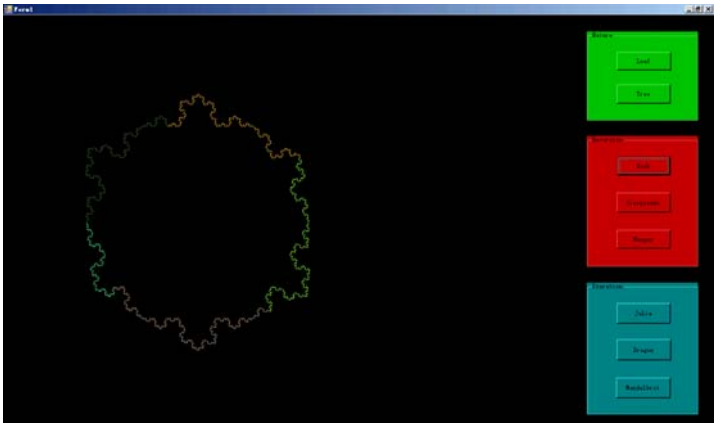

图 2 实验 4-5 程序主界面

实验一：二维基本图元的生成与填充

目的：

- 了解并掌握二维基本图元的生成算法与填充算法
- 参照 Windows 的画笔或 Office 中的绘图模块设计学会自己编程实现有关算法

内容

1. DDA 直线算法

算法描述如下(window_s.cpp):

```
int len;
double dx, dy;
double x, y;
int i;
if (abs(x2-x1) > abs(y2-y1))
{
    len = abs(x2-x1);
}
else
{
    len = abs(y2-y1);
}
dx = (static_cast<double>(x2)-static_cast<double>(x1))/static_cast<double>(len);
dy = (static_cast<double>(y2)-static_cast<double>(y1))/static_cast<double>(len);
x = x1 + 0.5;
y = y1 + 0.5;k
for (i=0; i<len; ++i)
{
    picbox->setPoint(static_cast<int>(x), static_cast<int>(y), r, g, b);
    x += dx;
    y += dy;
}
```

程序设计如下：

如图 3所示，选择DDA就可以进入DDA画线方式。

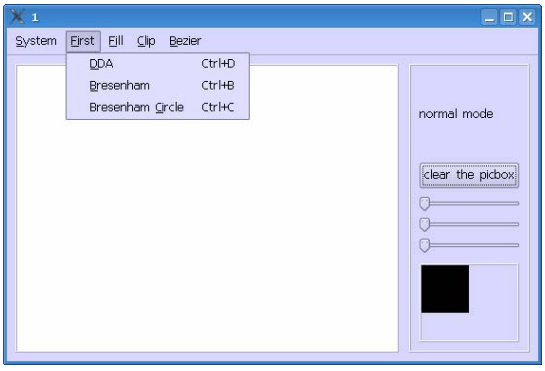

图 3 菜单选项

可以在右方的调色板上选择自己画线的颜色。

当鼠标点击白色绘图区就选定了直线的起点。此时移动鼠标便可看到直线的另一端跟随鼠标移动，此直线为DDA方式生成。单击鼠标，便确定了直线的终点，如图 4所示。

用此方法，可以在绘图区画多条直线，如图 5所示。

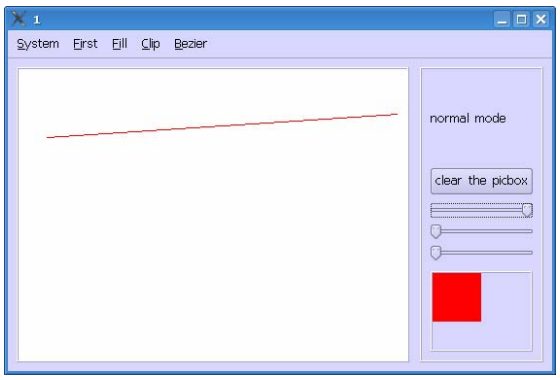

图 4 DDA 画线演示

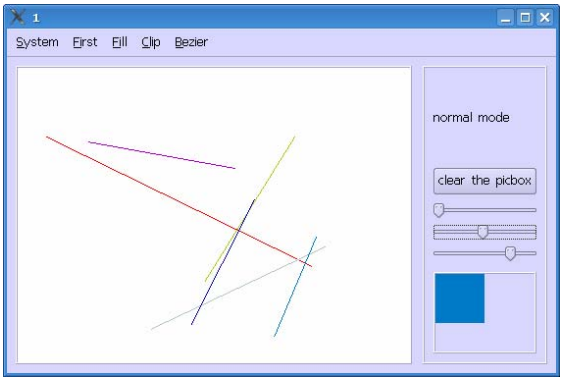

图 5 绘制多条 DDA 直线演示

2. Bresenham 直线

算法描述如下(window_s.cpp)：

```
if (x1 < x2)
{
    x = x1;
    y = y1;
    dx = x2 - x1;
    dy = y2 - y1;
    if (dy >= 0)
    {
        if (dy <= dx)
        {
            m = static_cast<double>(dy)/static_cast<double>(dx);
            e = m - 0.5;
            for (i=0; i<dx; ++i)
            {
                picbox->setPoint(x, y, r, g, b);
                while (e > 0)
                {
                    ++y;
                    e -= 1;
                }
                ++x;
                e += m;
            }
        }
    }
}
```

此算法针对 0-45 度方向直线进行绘制，其余部分与此类似，不再赘述。

程序运行过程与DDA直线演示相类似。先在菜单栏选择Bresenham直线演示。然后点击画板确定直线的起点和终点即可完成直线的绘制，如图 6所示。

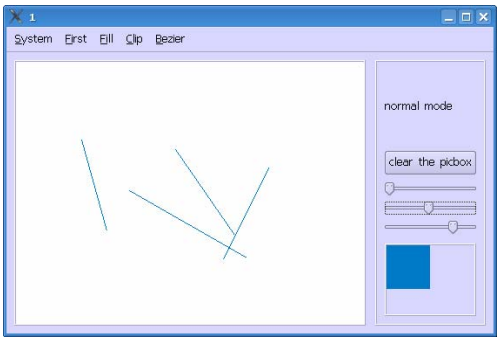

图 6 Bresenham 直线绘制演示

3. Bresenham 画圆法

算法描述如下(window_s.cpp)：

```
i = 2 * (1 - r);
while (y >= x)
{
    picbox->setPoint(cx+x, cy+y, m_r, m_g, m_b);
    picbox->setPoint(cx+x, cy-y, m_r, m_g, m_b);
    picbox->setPoint(cx-x, cy+y, m_r, m_g, m_b);
    picbox->setPoint(cx-x, cy-y, m_r, m_g, m_b);
    picbox->setPoint(cx+y, cy+x, m_r, m_g, m_b);
    picbox->setPoint(cx+y, cy-x, m_r, m_g, m_b);
    picbox->setPoint(cx-y, cy+x, m_r, m_g, m_b);
    picbox->setPoint(cx-y, cy-x, m_r, m_g, m_b);
    if (i < 0)
    {
        d = 2 * i + 2 * y - 1;
        if (d < 0)
        {
            ++x;
            i += (2 * x + 1);
        }
        else
        {
            ++x;
            --y;
            i += (2*x -2*y + 2);
        }
    }
    else
    {
        if (i > 0)
        {
            d = 2*i - 2*x - 1;
            if (d <= 0)
            {
                ++x;
                --y;
                i += (2*x -2*y + 2);
            }
            else
            {
                --y;
                i -= (2*y - 1);
            }
        }
        else
        {
            ++x;
            --y;
            i += (2*x -2*y + 2);
        }
    }
}
```

程序设计如下：

在绘图区域选择圆的圆心，此时会程序便会绘制出圆形，由一条绿色半径跟随鼠标，如图 7所示。

再点击一点确定圆形半径，即可完成圆形绘制，如图 8所示。

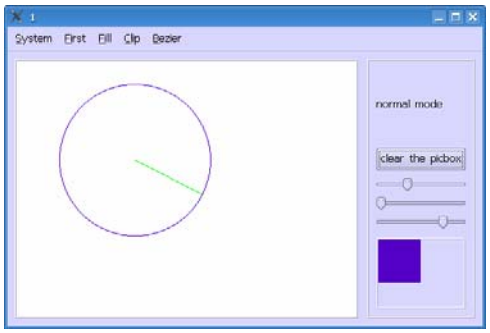

图 7 Bresenham 圆形绘制

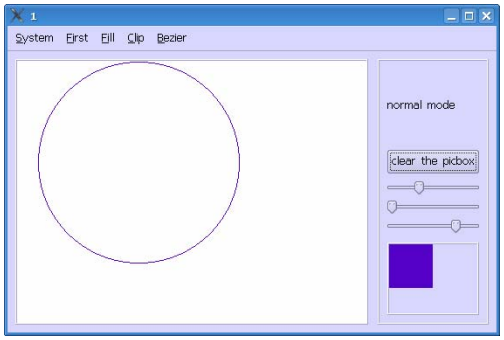

图 8 绘制 Bresenham 圆形结果

4. 演示方式 DDA 直线绘制

当当前状态为 DDA 直线绘制方式时，选择 system 菜单下 demostration 即可进入演示模式。

演示模式下，直线的点被放大 20 倍，且可以控制直线绘制进程。

当用户点击绘图板上的点时，确定了直线的起点，以红点描述，如图 9所示。

再次选择化图板上的象素点以确定其重点，以绿色描述，起点与终点之间由一条红色连线，如图 10所示。

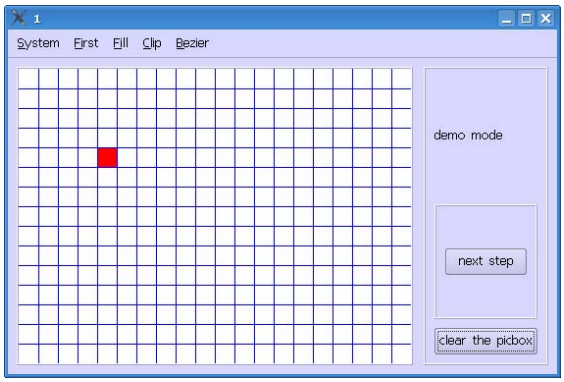

图 9 Bresenham 圆形演示确定圆心及圆上一点

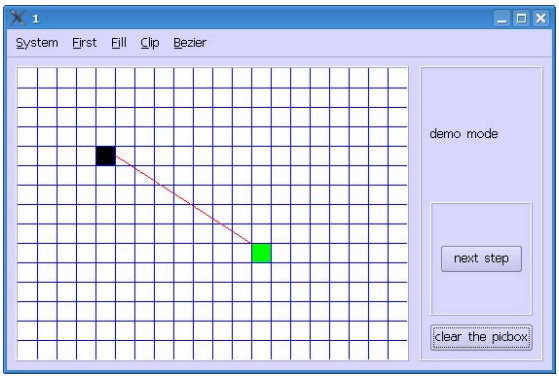

图 10 DDA 演示

单击“next step”按钮可以绘制一个象素点，如图 11所示。

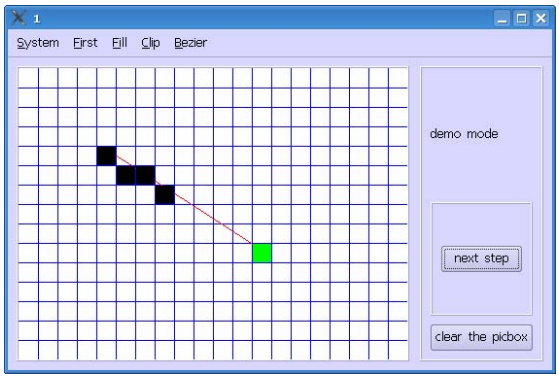

图 11 DDA 演示算法过程演示

以此方法，可以在绘图区绘制多条演示DDA直线，如图 12所示。

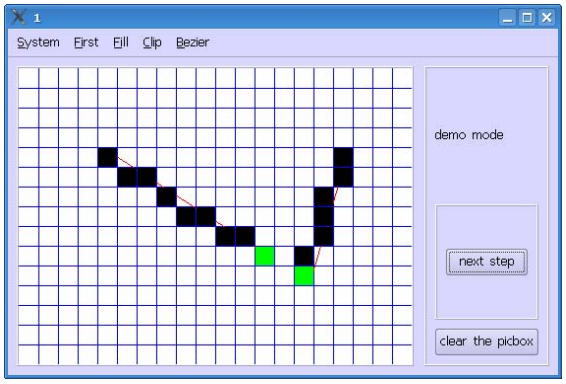

图 12 多条 DDA 演示直线

5. 演示方式 Bresenham 直线绘制

进入Bresenham直线演示模式与进入DDA直线演示模式相类似，进入Bresenham直线演示模式之后，界面与DDA直线类似，用鼠标选取直线的起点和终点之后，便可点击“next step”按钮进行单步绘制操作，如图 13所示。

此模式下可进行多条Bresenham直线演示，如图 14所示。

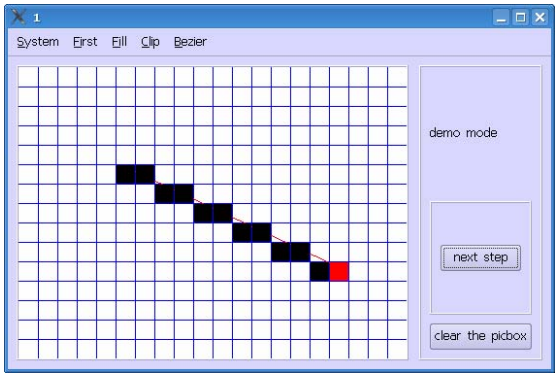

图 13 Bresenham 演示模式运行演示

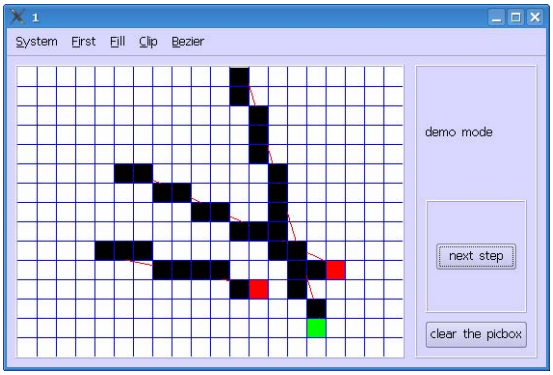

图 14 多条 Bresenham 演示直线

6. 演示方式 Bresenham 圆形绘制

进入Bresenham圆形演示绘制方式与之前演示方式相类似，在此不再赘述。鼠标选取圆形的圆心与圆上一点，圆心以红色描述，圆上一点以黄色描述，如图 15所示。

此时，单击“next step”便可开始单步Bresenham圆形绘制，如图 16所示。

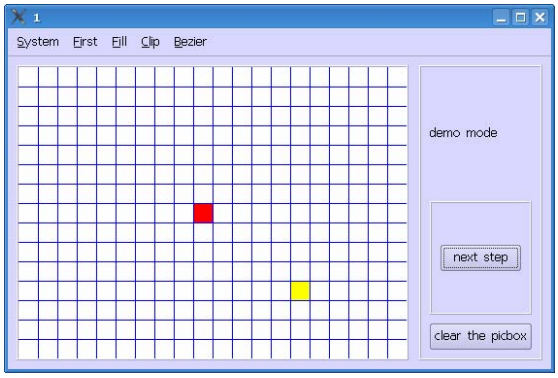

图 15 Bresenham 圆形演示模式

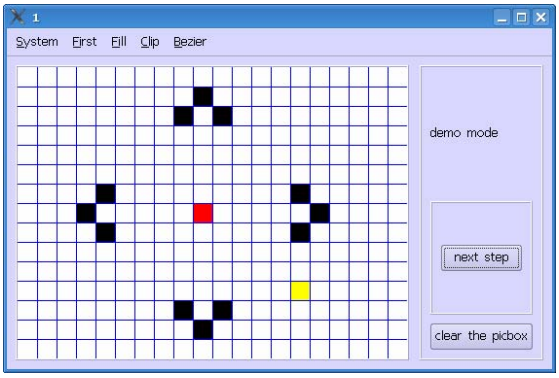

图 16 单步 Bresenham 圆形绘制过程

多次单击“next step”按钮后便完成Bresenham圆形的绘制，如图 17所示。

绘制多个Bresenham演示圆形如图 18所示。

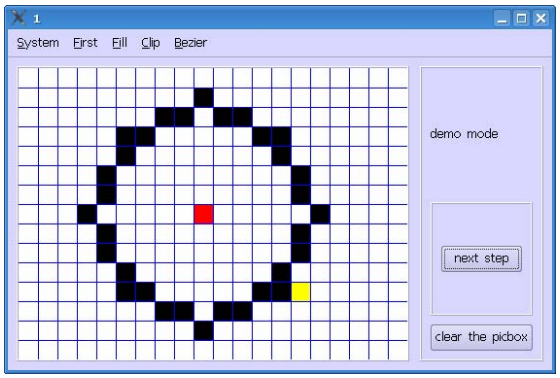

图 17 Bresenham 演示圆形绘制

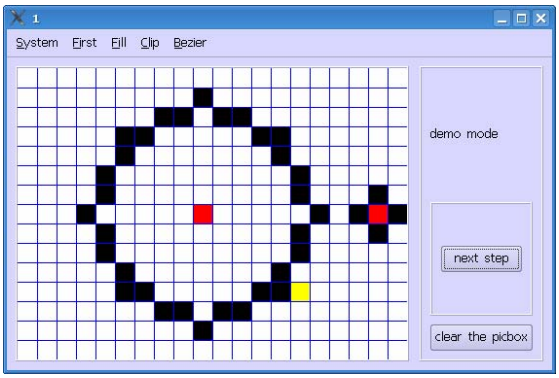

图 18 多个 Bresenham 演示圆形

7. 种子填充

算法描述如下(window_s.cpp):

```
QRgb c = picbox->getColor(x, y);
if (qRed(c)==255 && qGreen(c)==255 && qBlue(c)==255)
{
    picbox->setPoint(x, y, r, g, b);
    feedFill(picbox, x, y-1, r, g, b);
    feedFill(picbox, x, y+1, r, g, b);
    feedFill(picbox, x-1, y, r, g, b);
    feedFill(picbox, x+1, y, r, g, b);
}
```

程序设计如下：

首先进行多边形绘制操作，点击绘图区域，红点表示多边形的起点，绿线跟随鼠标，如图 19所示。当鼠标再次点击绘图区域时，确定了多边形上的边，当鼠标移至多边形起点时，会提示封闭多边形，如图 20所示。当点击后，多边形绘制完毕，如图 21所示。图 22所示为较为复杂的多边形。

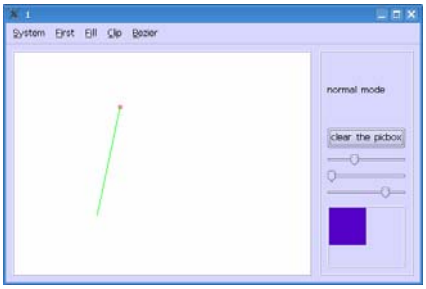

图 19 多边形绘制 1

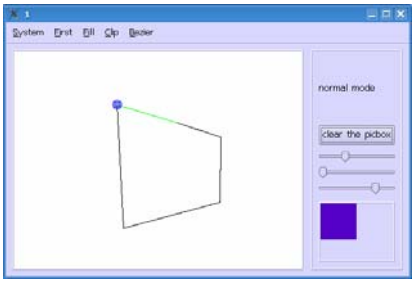

图 20 多边形绘制 2

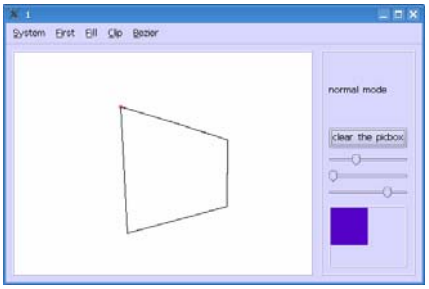

图 21 多边形绘制 3

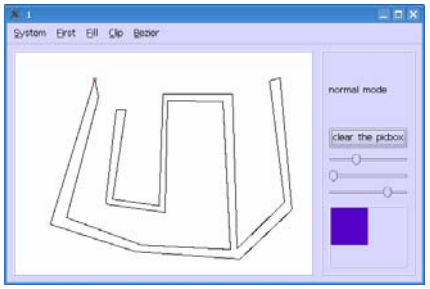

图 22 较为复杂的多边形

多边形绘制完成后，鼠标选取种子点，程序便自动开始填充，如图 23所示。当填充完毕后，如图 24所示。

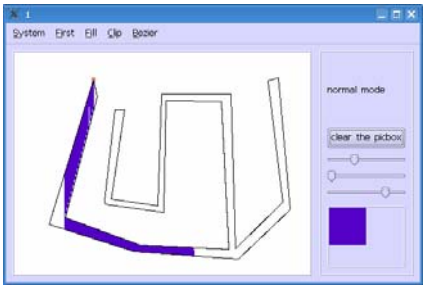

图 23 种子填充过程

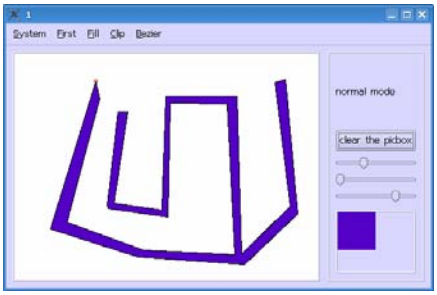

图 24 种子填充完成

同样，在绘图区域可以绘制多个区域进行填充，如图 25所示。

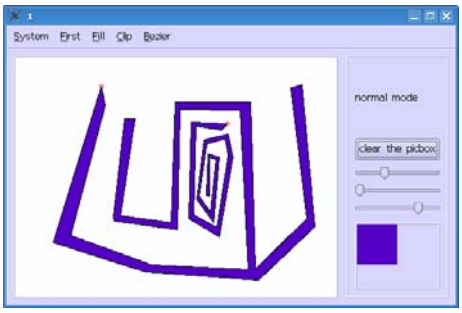

图 25 多个连通区域填充

8. 取反填充

选择菜单栏“Fill”项的“ScanFill”即可进入取反填充模式。

程序设计：

首先以同样方式绘制多边形，如图 26所示。

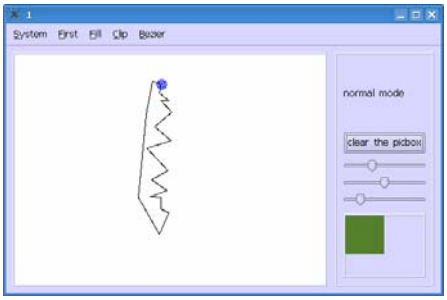

图 26 多边形绘制

绘制完成后，便开始填充操作，如图 27所示，完成后如图 28所示。

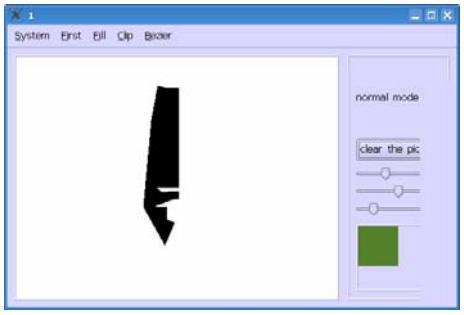

图 27 取反填充过程

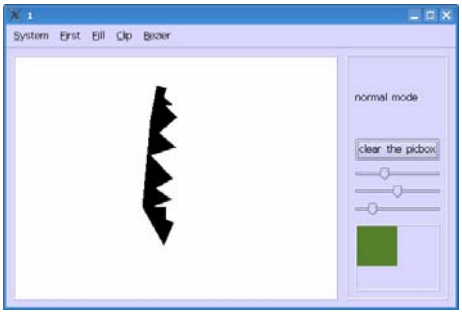

图 28 取反填充结果

实验二：二维图形的裁剪与变换

目的

- 了解并掌握二维基本图元的裁剪算法与变换算法
- 参照 Windows 的画笔或 Office 中的绘图模块设计学会自己编程实现有关算法

内容

1. Cohen-Sutherland 线段编码裁剪

算法描述如下(window_s.cpp):

```
void window::cohenClip(void)
{
    int i;
    for (i=0; i<m_poly1Cnt; i+=2)
    {
        cohenClipLine(m_poly1[i][0], m_poly1[i][1], m_poly1[i+1][0], m_poly1[i+1][1]);
    }
}

void window::cohenClipLine(int x1, int y1, int x2, int y2)
{
    int code1 = 0;
    int code2 = 0;
    int newX, newY;
    double m;
    if (x1 < m_clipMinX)
    {
        code1 += 1;
    }
    ……//省略其余点编码的判断
    if (0==code1 && 0==code2)
    {
        m_poly2[m_poly2Cnt][0] = x1;
        m_poly2[m_poly2Cnt][1] = y1;
        ++m_poly2Cnt;
        m_poly2[m_poly2Cnt][0] = x2;
        m_poly2[m_poly2Cnt][1] = y2;
        ++m_poly2Cnt;
        return ;
    }

    if ((code1 & code2) != 0)
    {
        return ;
    }

    if (x1< m_clipMinX && x2 >= m_clipMinX)
    {
        m = static_cast<double>(y2-y1) / static_cast<double>(x2-x1);
        newX = m_clipMinX;
        newY = static_cast<int>(y1 + m*(newX-x1));
        cohenClipLine(newX, newY, x2, y2);
        return;
    }
    ……//省略其他出界判断情况
}
```

程序设计如下：

首先绘制直线。方法与之前绘制直线方法相同。完成后点击鼠标右键，进入裁剪框绘制阶段，在绘图区点击鼠标左键，选取裁剪框，如所示，裁减框以红色描述。单击鼠标左键以确定裁减框，并进行裁减，裁减完成后效果如所示。

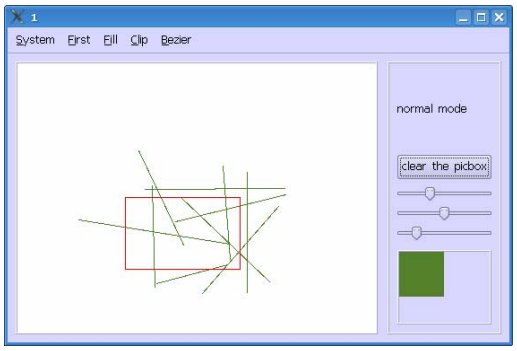

图 29 裁减框绘制

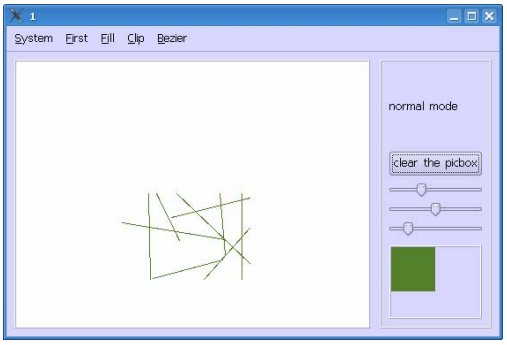

图 30 Cohen-Sutherlan 裁减结果

2. Sutherland-Hodgman 多边形裁剪法

以相同方法进入 Sutherlan-Hodgman 多边形裁减模式

左边裁减算法如下(window_s.cpp)：

```
//le
ft
m_poly2Cnt = 0;
for (i=0; i<m_poly1Cnt-1; ++i)
{
    m = static_cast<double>(m_poly1[i+1][1]-m_poly1[i][1]) / static_cast<double>(m_poly1[i+1][0] - m_poly1[i][0]);
    if (m_poly1[i][0] < m_clipMinX)
    {
        if (m_poly1[i+1][0] < m_clipMinX)
        {
            //nothing to do
        }
        else
        {
            //go in
            newX = m_clipMinX;
            newY = static_cast<int>(m_poly1[i][1]+m*(newX-m_poly1[i][0]));
            m_poly2[m_poly2Cnt][0] = newX;
            m_poly2[m_poly2Cnt][1] = newY;
            ++m_poly2Cnt;
            m_poly2[m_poly2Cnt][0] = m_poly1[i+1][0];
            m_poly2[m_poly2Cnt][1] = m_poly1[i+1][1];
            ++m_poly2Cnt;
        }
    }
    else
    {
        if (m_poly1[i+1][0] < m_clipMinX)
        {
            //go out
            newX = m_clipMinX;
            newY = static_cast<int>(m_poly1[i][1]+m*(newX-m_poly1[i][0]));
            m_poly2[m_poly2Cnt][0] = newX;
            m_poly2[m_poly2Cnt][1] = newY;
            ++m_poly2Cnt;
        }
```

```
            else
            {
                m_poly2[m_poly2Cnt][0]      =
m_poly1[i+1][0];
                m_poly2[m_poly2Cnt][1]      =
m_poly1[i+1][1];
                ++m_poly2Cnt;
            }
        }
}
if (m_poly2Cnt > 0)
{
    m_poly2[m_poly2Cnt][0]                  =
m_poly2[0][0];
    m_poly2[m_poly2Cnt][1]                  =
m_poly2[0][1];
    ++m_poly2Cnt;
}

for (i=0; i<m_poly2Cnt; ++i)
{
    m_poly1[i][0] = m_poly2[i][0];
    m_poly1[i][1] = m_poly2[i][1];
}
m_poly1Cnt = m_poly2Cnt;
//end of left
```

程序设计如下：

进入Sutherland-Hodgman多边形裁减模式，此时，在绘图区域绘制多边形，如图 31所示。绘制完成后选取裁减框，方法如Cohen-Sutherland中绘制裁减框，如图 32所示。

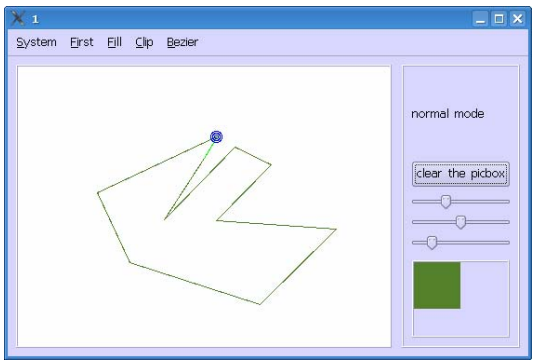

图 31 多边形绘制

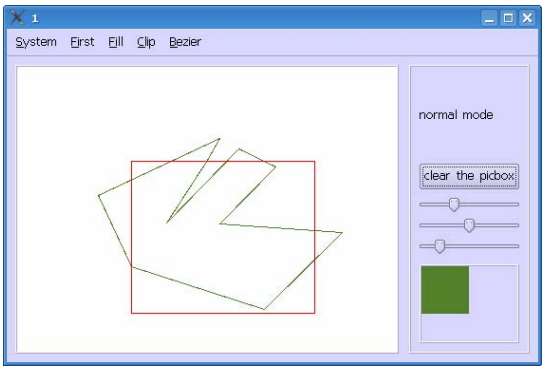

图 32 绘制裁减框

裁减完成后如图 33所示。

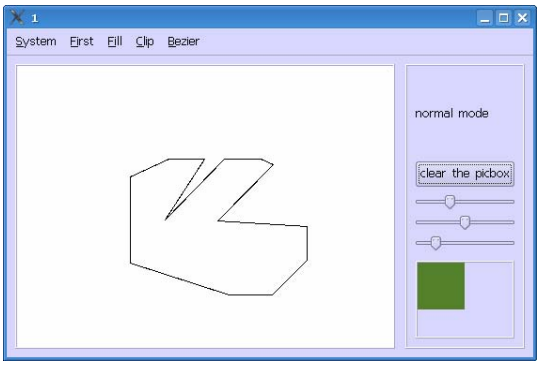

图 33 多边形裁剪结果

*图像平移缩放请见实验五

实验三：Bezier 曲线

目的

- 理解并会自己编程实现二维 Bezier 曲线的画图

内容

1. Bezier 曲线绘制

算法描述如下：

```
for (i=0; i<m_poly2Cnt; ++i)
{
    x = 0;
    dis = 0;
    for (j=m_poly2[i][0]; j<m_poly2[i][1]-1; ++j, ++x)
    {
        vp[x][0] = m_poly1[j][0];
        vp[x][1] = m_poly1[j][1];
        dis += static_cast<int>(sqrt((m_poly1[j][0]-m_poly1[j+1][0])*(m_poly1[j][0]-m_poly1[j+1][0])
                + (m_poly1[j][1]-m_poly1[j+1][1])*(m_poly1[j][1]-m_poly1[j+1][1])));
    }
    n = m_poly2[i][1] - m_poly2[i][0] ;
    vp[n-1][0] = m_poly1[m_poly2[i][1]-1][0];
    vp[n-1][1] = m_poly1[m_poly2[i][1]-1][1];
    lp[0] = -1;
    lp[1] = -1;
    step = 15.0 / dis;
    for (t=0.0; t<=1.0; t+=step)
    {
        ap[0] = 0;
        ap[1] = 0;
        for (k=0; k<=n; ++k)
        {
            if (0 == k)
            {
                factor = 1;
                a = 1.0;
                for (x=0; x<n-1; ++x)
                {
                    a *= (1-t);
                }
                b = 1.0;
            }
            else
            {
                factor *= (n-k);
                factor /= k;
                a /= (1-t);
                b *= t;
            }
            ap[0] += static_cast<int>(a*b*factor*vp[k][0] +0.5);
            ap[1] += static_cast<int>(a*b*factor*vp[k][1]+0.5);
        }
        if (lp[0]!=-1 && lp[1] !=-1)
        {
            picbox->drawLine(lp[0], lp[1], ap[0], ap[1], 0, 0, 255, 1);
```

```
            }
            lp[0] = ap[0];
            lp[1] = ap[1];
        }
    }
}
```

程序设计如下：

进入Bezier曲线绘制模式。在绘图框进行折线绘制，如图 34所示。点击鼠标右键结束绘制，此时Bezier曲线会自动在绘图况中绘制出来，如图 35所示。

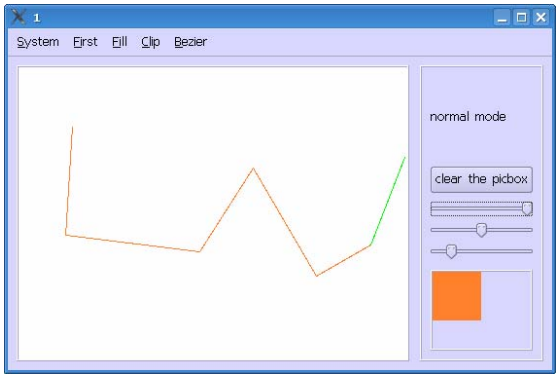

图 34 折线绘制

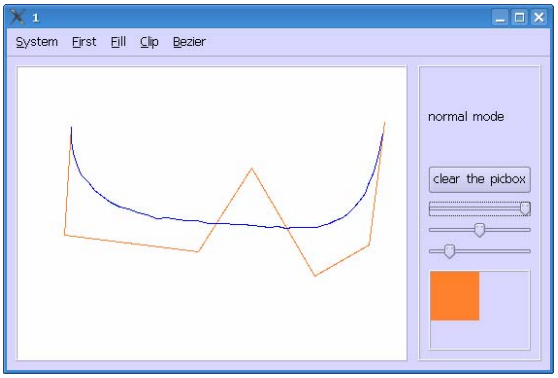

图 35 Bezier 曲线绘制

可以在绘图框中绘制多条 Bezier 曲线。

需要修改顶点坐标时，鼠标移动到该点位置，点击左键“拾起”该点，此时，顶点会变为红色，如图 36所示。鼠标移动到相应位置后，点击鼠标左键“放下”该点，此时根据新坐标点的Bezier曲线会自动绘制完成，如图 37所示。

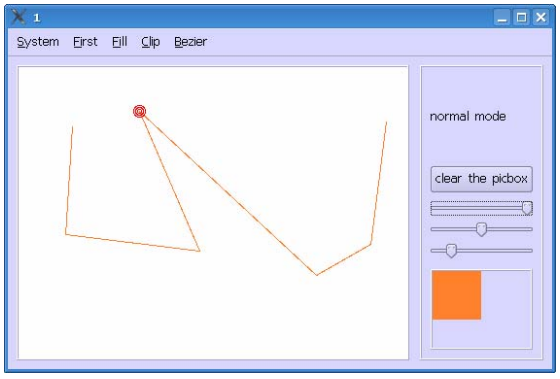

图 36 移动顶点

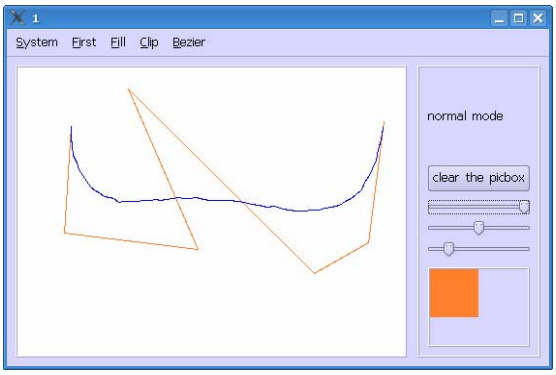

图 37 新坐标点 Bezier 曲线绘制

实验四：分形图形

目的

- 理解并会自己编程实现 简单的分形几何造型算法；
- 实现一类函数递归图形；
- 实现一类基于生成元的分形图形；
- 实现一类递归图形；
- 实现一类基于迭代函数系统 IFS 的分形图形。

内容

1. 分形模拟自然界图形
   Leaf 分形树叶
   Tree 分形二叉树
2. 函数递归分形图形
   Koch 基于生成元经典分形图形
   Sierpinski 函数递归分形三角形
   Menger 函数递归分形正方形
3. 迭代数系统的分形图形
   Julia Julia 域
   Dragon‘龙’域
   Mandelbrot Mandelbrot 域

典型算法举例

1. Leaf 分形树叶

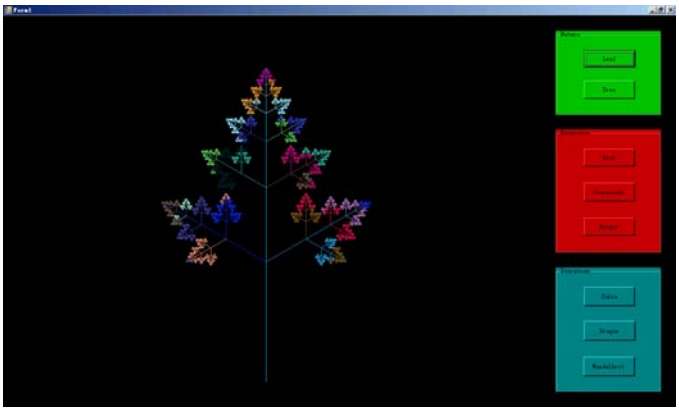

图 38 Leaf 分形图形

算法描述：

```
if (len < 1)       //如果树长度小于，递归结束
    return;

Point endP = new Point();
double sin = Math.Sin(angle);
double cos = Math.Cos(angle);
int x = startP.X;
int y = startP.Y;

endP.X = (int)(x + len * cos+0.5);          //计算终点位置
endP.Y = (int)(y - len * sin+0.5);

Random rm = new Random();            //设置随机颜色，画出五彩的叶子
Pen myPen = new Pen(new SolidBrush(Color.FromArgb(rm.Next(255), rm.Next(255),
rm.Next(255))));

g.DrawLine(myPen, startP, endP);             //画出主叶茎

Point nextP = new Point();
nextP.X = x;
nextP.Y = y;
for (int i = 0; i < 10; i++)             //每次取当前叶干的（-0.618）黄金分割点处
{                                                    //作为支叶茎的起点与长度

    nextP.X = (int)(nextP.X + len * (1 - 0.618) * cos+0.5);
    nextP.Y = (int)(nextP.Y - len * (1 - 0.618) * sin+0.5);

    Leaf(nextP, len * (1 - 0.618), angle - Math.PI / 3, g);
    Leaf(nextP, len * (1 - 0.618), angle + Math.PI / 3, g);
    len *= 0.618;
}
```

2. Koch 雪花图形

生成元：直线三分之一段用正三角形代替

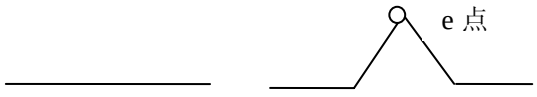

图 39 Koch 生成元

如果 e 点随即选取，将得到分子布朗运动的模型；

由一个正六边形最终得到一片雪花，如图 40 Koch雪花所示。

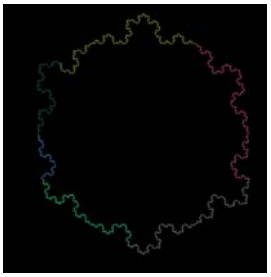

图 40 Koch 雪花

算法描述如下：

```
if (n > 0)
{
    double r;
    if ((a.X - b.X) == 0)
        r = Math.PI / 2;
    else
        r = Math.Atan(Math.Abs(a.Y - b.Y) / Math.Abs(a.X - b.X)); //角度
    double v = 0;
    //生成各点坐标
    Point c = new Point((int)(a.X + (b.X - a.X) / 3+0.5), (int)(a.Y + (b.Y - a.Y) / 3+0.5));
    Point d = new Point((int)(a.X + 2 * (b.X - a.X) / 3), (int)(a.Y + 2 * (b.Y - a.Y) / 3));
    double l = Math.Sqrt((c.X - d.X) * (c.X - d.X) + (c.Y - d.Y) * (c.Y - d.Y));
    Point e;
    //尖角点 e 位置要分四象限讨论
    if (b.Y - a.Y >= 0 && b.X - a.X > 0)
    {
        v = r;
    }
    else if (b.Y - a.Y > 0 && b.X - a.X <= 0)
    {
        v = Math.PI - r;
    }
    else if (b.Y - a.Y <= 0 && b.X - a.X < 0)
    {
        v = Math.PI + r;
    }
    else if (b.Y - a.Y < 0 && b.X - a.X >= 0)
    {
        v = 2 * Math.PI - r;
    }



    e = new Point((int)(l * Math.Cos(v + Math.PI / 3) + c.X + 0.5), (int)(c.Y + l * Math.Sin(v +
Math.PI / 3) + 0.5));
```

```
        Koch(a, c, n - 1, g);
        Koch(c, e, n - 1, g);
        Koch(e, d, n - 1, g);
        Koch(d, b, n - 1, g);
    }
    else
    {                   //递归到最后一层，开始画线

        Random rnd = new Random();
        Color c = Color.FromArgb(255, (byte)rnd.Next(255), (byte)rnd.Next(255), (byte)rnd.Next());

        Pen p = new Pen(Brushes.AliceBlue);
        p.Color = c;                //赋予雪花随机颜色
        g.DrawLine(p, new Point(a.X, a.Y), new Point(b.X, b.Y));
    }
```

3. Julia 域

对于复数 z0=x+iy，取不同的 x 值和 y 值，函数迭代的结果不一样：对于有些 z0，函数值约束在某一范围内；而对于另一些 z0，函数值则发散到无穷。由 于复数对应平面上的点，因此我们可以用一个平面图形来表示，对于哪些 z0 函数值最终趋于无穷，对于哪些 z0 函数值最终不会趋于无穷。我们用蓝色的深浅来区别不同的发散速度。由于当某个时候|z|>2 时，函数值一定发散，因此这里定义发散 速度为：使|z|大于 2 的迭代次数越少，则发散速度越快。

算法描述如下：

```
public struct Complex           //实现复数结构
{
    public double real;
    public double imaginary;

    public Complex(double real, double imaginary){    //constructor
        this.real = real;
        this.imaginary = imaginary;
    }
//定义运算符重载
    public static Complex operator +(Complex c1, Complex c2){
        return new Complex(c1.real + c2.real, c1.imaginary + c2.imaginary);
    }
    public static Complex operator *(Complex c1, Complex c2){
        return new Complex(c1.real * c2.real-c1.imaginary * c2.imaginary,
            c1.imaginary*c2.real+c1.real*c2.imaginary);
    }
```

```
}
public void Julia(Graphics g, double cRe, double cIm)
{
    Complex c = new Complex(cRe, cIm);
    Complex z = new Complex(0, 0);
    for (double i = -300; i <= 300; i++)
        for (double j = -200; j <= 200; j++)
        {
            z.real = i / 200;
            z.imaginary = j /200;
            int n;
            for (n = 0; n < 255; n++)
            {
                if (Math.Sqrt(z.real * z.real + z.imaginary * z.imaginary) > 2) break;
                else z = (z * z) + c;
            }
            Color nc = new Color();
                                    n *= 10;
            if (n > 255) n = 255;
            nc = Color.FromArgb(0, 0, n);      //不同发散速度给与不同颜色
            g.FillRectangle(new SolidBrush(nc), (int)(i + 300), (int)(j + 200), 1, 1);
        }
}
private void button6_Click(object sender, EventArgs e)
{
    Graphics g = CreateGraphics();
    g.Clear(this.BackColor);
    Julia(g, -0.75, 0);
    g.Dispose();
}
```

美丽的图 41表现的就是f(z)=z^2-0.75 时的Julia集。考虑复数函数f(z)=z^2+c，不同的复数c对应着不同的Julia集。 也就是说，每取一个不同的c你都能得到一个不同的Julia集分形图形，并且令人吃惊的是每一个分形图形都是那么美丽。

取c= -0.835, -0.2321，得到‘龙’形， 如图 42所示。

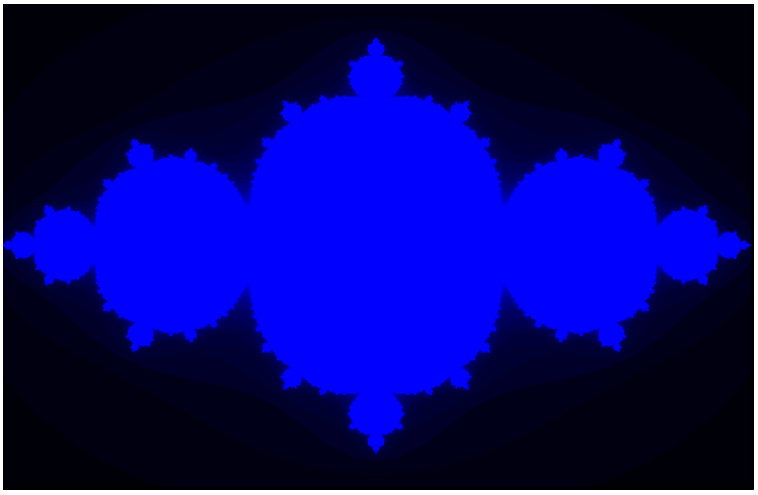

图 41 f(z)=z^2-0.75 时的 Julia 集

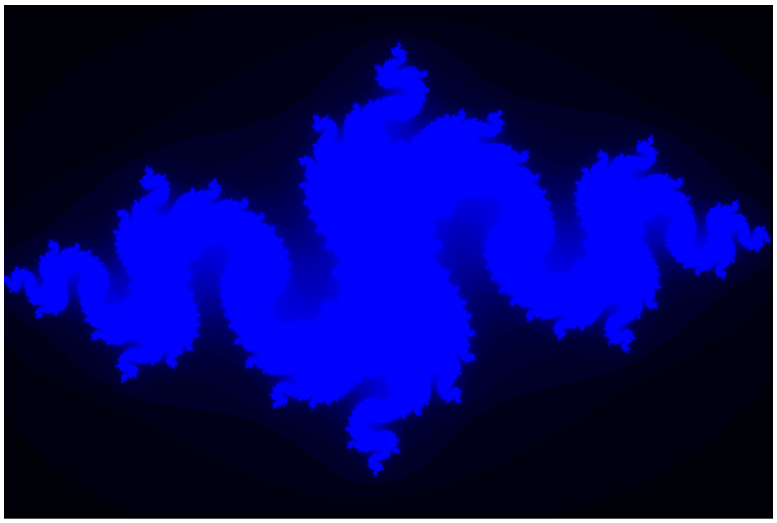

图 42 c= -0.835, -0.2321 ‘龙’形

若取不同颜色代表不同的发散速度，得到彩色图像，如图 43所示。

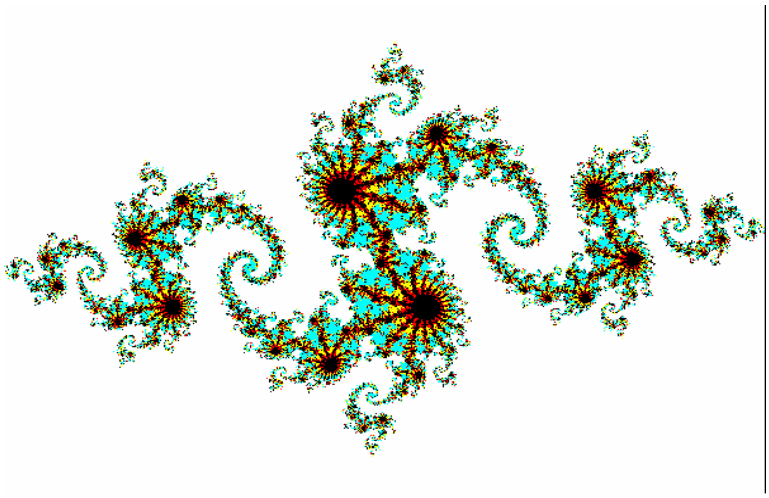

图 43 颜色代表发散速度的 Julia 集

如果我们固定z0=0，那么对于不同的复数c，函数的迭代结果也不同。由于复数c对应平面上的点，因此我们可以用一个平面图形来表示，对于某个复数c，函数 f(z)=z^2+c从z0=0开始迭代是否会发散到无穷。我们同样用不同蓝色来表示不同的发散速度，最后得出的就是

Mandelbrot集分形图形，如图 44所示。

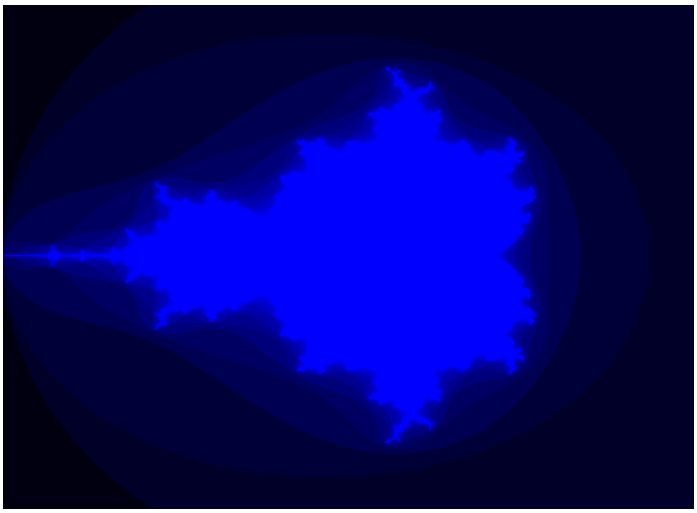

图 44 Mandelbrot

实验五：自选题目

目的

- 理解并会自己编程实现一些常用的分形曲线造型算法

内容

1. VRML 文件读取器

程序界面如图 45所示。

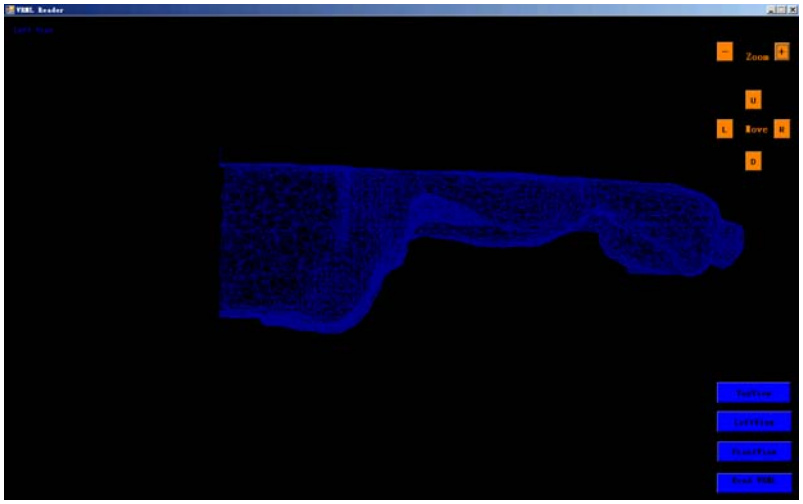

图 45 VRML 文件读取器

功能：

能够读取简单的基于三角面片的 VRML 三维模型，以三视图显示，并能够放缩和平移。

操作说明：

如图 46所示，点击“Read VRML”，在弹出的文件选择菜单中选择电子版中的一个.wrl 文件打开，菜单左上角显示“Successively done”表示文件已成功读取完毕。

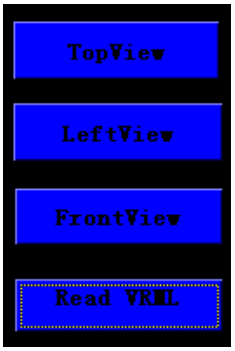

图 46 VRML 读取器操作菜单

点击 TopView,LeftView,FrontView 显示模型的三视图。
用控制盘（图 47）缩放或平移图形，以获得最佳观测效果。

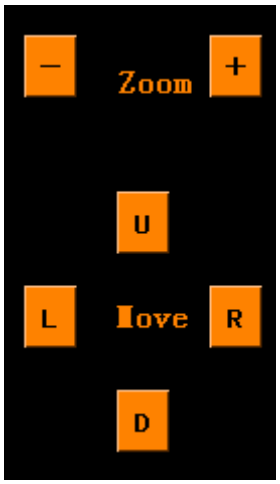

图 47 VRML 读取器控制盘

总结

1. 小组分工

何琪辰：实验 1-3 编写及相关部分报告的撰写。
叶剑峰：实验 4、5 编写及相关部分报告的撰写。
黄亦琦：实验测试，实验报告撰写。

2. 实验体会

通过 5 个实验的上机编写制作，我们可以更加清楚的掌握并且了解了计算机图形学的各个章节的知识，并且通过动手实践，也锻炼了我们的编程能力，和团队合作能力。特别在分形图形程序开发的过程中，我们在互联网上查找许多有关资料，对分形图形逐渐了解。对它在数学上，美学上能够达到的美丽而感到惊叹。一方面是我们对图形学更加深了了解，另一方面也让我们体会到图形学的内涵，让我们十分感兴趣。

在完成 5 个实验的过程中，我们收获非常大，何琪辰同学他自己用 Qt 开发了一个画图板控件，采用双缓冲技术，使图形不会因窗口的遮挡而消失。美中不足的就是，设计之初该程序时可以跨平台运行的，但由于 Qt 自身的原因，Windows 下内存消耗特别大而无法运行，致使无法将 5 个程序集合在一个程序平台上运行。

叶剑峰同学在开发分形图形时完成了复数类的编写，并精心设计了每个分形图形，还为每个图形取名，对分形图形的美找了迷，可以说，在我们实验报告上的一些分形图形是世界上独一无二的，只有在我们的报告上会出现。另外他用 C#编程实现了 VRML 文件的读取，整个 VRML 操作(读取，显示)封装在类中，可以被其他程序方便的使用，虽然只能实现三角面片模型的读取，但是深入了解了 VRML 文件的结构，很有收获。

黄亦琦也在这个实验中收获颇丰，虽然没有开发程序，但是何琪辰同学和叶剑峰同学开发完成的程序都必须在他仔细严格的测试下没有任何问题才能算完成。同时也完成了部分实验报告的撰写工作。

最后我们要感谢丁友东老师对我们的指点，同时也让我们对图形学产生浓厚的兴趣，谢谢！