案例 5　中点画线算法

知识点

- 直线中点算法。
- 定义颜色类。
- 定义直线类。
- 隐函数表示。

一、案例需求

1. 案例描述

假定直线的起点为红色，终点为蓝色。以窗口客户区中心为圆心，在自定义坐标系内沿圆周绘制间隔 30°的直线图。试基于中点算法设计直线类来实现。

2. 功能说明

(1) 自定义二维屏幕坐标系：原点位于客户区中心，x 轴水平向右为正，y 轴垂直向上为正。直线的起点坐标和终点坐标相对于窗口客户区中心定义。

(2) 设计 CRGB 类，其成员变量为 double 型的红绿蓝分量 red、green 和 blue，将 red、green 和 blue 分量分别规范到[0,1]区间。

(3) 设计 CLine 直线类，其成员变量为直线的起点坐标 P_0 和终点坐标 P_1，成员函数为 MoveTo()和 LineTo()函数。直线起点的颜色为红色，终点的颜色为蓝色。

3. 案例效果图

任意斜率由中点红色向两端蓝色渐变的直线绘制效果如图 5-1 所示。

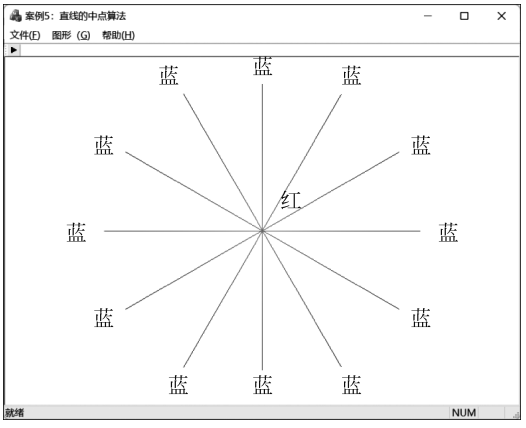

图 5-1　用中点算法绘制直线效果图

二、案例分析

中点算法是隐函数算法，隐函数容易判断点与图形的位置关系：点位于图形之内、之外、之上。

1. 整数中点算法

中点算法使用候选相邻像素的中点坐标判断哪个像素离直线更近，如图 5-2 所示。例如，如果中点位于直线的下方，则像素点 P_u 距离直线近；否则，像素点 P_d 距离直线近。

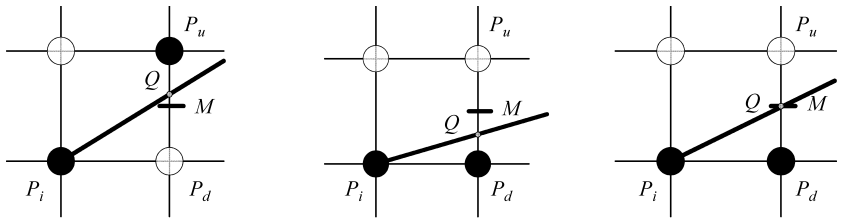

图 5-2　中点算法

本案例给出的是整数中点算法。直线的端点用整型类 CPoint2 定义。直线的颜色用 CRGB 类定义。

2. 线性插值公式

为了绘制颜色渐变直线，本案例对直线端点的颜色进行线性插值。

已经知道，线性插值公式的参数表达式为

$$P=(1-t)P_0+tP_1,\qquad 0\leqslant t\leqslant 1 \tag{5-1}$$

写成分量表示

$$x=(1-t)x_0+tx_1 \tag{5-2}$$

$$y=(1-t)y_0+ty_1 \tag{5-3}$$

$$c=(1-t)c_0+tc_1 \tag{5-4}$$

由式(5-2)有

$$t=\frac{x-x_0}{x_1-x_0} \tag{5-5}$$

代入式(5-4)，得到沿 x 方向的颜色线性插值公式：

$$c=\frac{x_1-x}{x_1-x_0}c_0+\frac{x-x_0}{x_1-x_0}c_1 \tag{5-6}$$

由式(5-3)有

$$t=\frac{y-y_0}{y_1-y_0} \tag{5-7}$$

代入式(5-4)，得到沿 y 方向的颜色线性插值公式：

$$c=\frac{y_1-y}{y_1-y_0}c_0+\frac{y-y_0}{y_1-y_0}c_1 \tag{5-8}$$

令 m 代表 x 或 y 方向，m_{start} 代表 x_0 或 y_0，m_{end} 代表 x_1 或 y_1，c_{start} 代表 c_0，c_{end} 代表 c_1。这样，沿 x 方向或者沿 y 方向的线性插值公式可以统一表示为

$$c=\frac{m_{\text{end}}-m}{m_{\text{end}}-m_{\text{start}}}c_{\text{start}}+\frac{m-m_{\text{start}}}{m_{\text{end}}-m_{\text{start}}}c_{\text{end}} \tag{5-9}$$

这是颜色的线性插值公式。计算机图形学中,线性插值公式应用广泛,可以对法向量、纹理坐标、深度指标等进行线性插值。

三、算法设计

对于 $0\leqslant k\leqslant 1$ 的颜色渐变直线,中点算法表述如下。

(1)使用鼠标选择起点坐标 $p_0(x_0,y_0,c_0)$和终点坐标 $p_1(x_1,y_1,c_1)$。要求起点的 x 坐标小于或等于终点的 x 坐标,如果不满足,则交换二者。

(2) 定义直线段当前点坐标 $p(x,y,c)$、中点误差项 e。

(3) 当前点取直线起点 $p=p_0$,计算 $e_0=\Delta x-2\Delta y$。

(4) 绘制点 p,判断 e 的符号。若 $e<0$,则(x,y)更新为$(x+1,y+1)$,e 更新为 $e_{i+1}=e_i+2\Delta x-2\Delta y$;否则$(x,y)$更新为$(x+1,y)$,$e$ 更新为 $e_{i+1}=e_i-2\Delta y$。 p 点的颜色取为 p_0与 p_1点颜色的线性插值结果。

(5) 如果当前点 x 小于 x_1,则重复步骤(4),否则结束。

四、案例实现

1. 设计 CRGB 类

为了规范颜色的处理,定义了 CRGB 颜色类,颜色分量有 red、green 和 blue,重载了“+”“-”“*”“/”“+=”“-=”“*=”“/=”运算符。“+”运算符用于计算两种颜色分量的和,“-”运算符用于计算两种颜色分量的差,“*”运算符用于计算数和颜色分量的左乘和右乘,“/”运算符用于计算颜色分量和数的商,复合运算符“+=”“-=”“*=”“/=”与此类似。成员函数 Normalize()将颜色分量规范到[0,1]闭区间内。

```
class CRGB                                                    //颜色类
{
public:
    CRGB(void);
    CRGB(double red, double green, double blue);
    virtual ~CRGB(void);
    friend CRGB operator+(const CRGB &c1, const CRGB &c2);    //运算符重载
    friend CRGB operator-(const CRGB &c1, const CRGB &c2);
    friend CRGB operator*(const CRGB &c1, const CRGB &c2);
    friend CRGB operator*(const CRGB &c1, double scalar);
    friend CRGB operator*(double scalar,const CRGB &c);
    friend CRGB operator/(const CRGB &c1, double scalar);
    friend CRGB operator+=(CRGB &c1, CRGB &c2);
    friend CRGB operator-=(CRGB &c1, CRGB &c2);
    friend CRGB operator*=(CRGB &c1, CRGB &c2);
    friend CRGB operator/=(CRGB &c1, double scalar);
    void Normalize(void);                                 //颜色分量规范化到[0,1]区间

public:
    double red;                                           //红色分量
```

```
    double green;                                          //绿色分量
    double blue;                                           //蓝色分量
};
CRGB::CRGB(void)
{
    red=1.0;
    green=1.0;
    blue=1.0;
}
CRGB::CRGB(double red, double green, double blue)      //重载构造函数
{
    this->red=red;
    this->green=green;
    this->blue=blue;
}
CRGB::~CRGB(void)
{
}
CRGB operator+(const CRGB &c1, const CRGB &c2)         //“+”运算符重载
{
    CRGB color;
    color.red=c1.red+c2.red;
    color.green=c1.green+c2.green;
    color.blue=c1.blue+c2.blue;
    return color;
}
CRGB operator -(const CRGB &c1, const CRGB &c2)        //“-”运算符重载
{
    CRGB color;
    color.red=c1.red-c2.red;
    color.green=c1.green-c2.green;
    color.blue=c1.blue-c2.blue;
    return color;
}
CRGB operator*(const CRGB &c1, const CRGB &c2)         //“*”运算符重载
{
    CRGB color;
    color.red=c1.red*c2.red;
    color.green=c1.green*c2.green;
    color.blue=c1.blue*c2.blue;
    return color;
}
CRGB operator*(const CRGB &c1,double scalar)           //“*”运算符重载
{
    CRGB color;
    color.red=scalar*c1.red;
    color.green=scalar*c1.green;
    color.blue=scalar*c1.blue;
    return color;
}
CRGB operator*(double scalar, const CRGB &c1)          //“*”运算符重载
```

```
{
    CRGB color;
    color.red=scalar*c1.red;
    color.green=scalar*c1.green;
    color.blue=scalar*c1.blue;
    return color;
}
CRGB operator /(const CRGB &c1, double scalar)          //“/”运算符重载
{
    CRGB color;
    color.red=c1.red/scalar;
    color.green=c1.green/scalar;
    color.blue=c1.blue/scalar;
    return color;
}
CRGB operator+=(CRGB &c1,CRGB &c2)                      //“+=”运算符重载
{
    c1.red+=c2.red;
    c1.green+=c2.green;
    c1.blue+=c2.blue;
    return c1;
}
CRGB operator-=(CRGB &c1,CRGB &c2)                      //“-=”运算符重载
{
    c1.red-=c2.red;
    c1.green-=c2.green;
    c1.blue-=c2.blue;
    return c1;
}
CRGB operator*=(CRGB &c1,CRGB &c2)                      //“*=”运算符重载
{
    c1.red*=c2.red;
    c1.green*=c2.green;
    c1.blue*=c2.blue;
    return c1;
}
CRGB operator /=(CRGB &c1,double scalar)                //“/=”运算符重载
{
    c1.red/=scalar;
    c1.green/=scalar;
    c1.blue/=scalar;
    return c1;
}
void CRGB::Normalize(void)                              //颜色规范化处理
{
    red=(red<0.0)?0.0 : ((red>1.0)?1.0 : red);
    green=(green<0.0)?0.0 : ((green>1.0)?1.0 : green);
    blue=(blue<0.0)?0.0 : ((blue>1.0)?1.0 : blue);
}
```

2. 设计二维点类

定义二维点类 CP2,将颜色信息绑定到二维点上。CP2 类中成员变量 x 和 y 为浮点型,在输出时需要取整。

```
#include "RGB.h"
class CP2
{
public:
    CP2 (void);
    virtual ~CPoint2(void);
    CP2 (double x, double y);
    CP2 (double x, double y, CRGB c);
public:
    double x;
    double y;
    CRGB c;
};
CP2::CP2 (void)
{
    x=0.0;
    y=0.0;
}
CP2::CP2 (double x, double y)
{
    this->x=x;
    this->y=y;
    this->c=CRGB(0.0, 0.0, 0.0);
}
CP2::CP2 (double x, double y, CRGB c)
{
    this->x=x;
    this->y=y;
    this->c=c;
}
CPoint2::~CP2(void)
{
}
```

3. 设计 CLine 直线类

定义直线类绘制任意斜率的直线,其主要成员函数为 MoveTo()和 LineTo()。

```
#include "P2.h"                                    //带颜色的浮点数二维点类
#include "Point2.h"                                //带颜色的整数二维点类
class CLine
{
public:
    CLine(void);
    virtual ~CLine(void);
    void MoveTo(CDC * pDC, CP2 p0);                //移动到指定位置
    void MoveTo(CDC * pDC, double x0, double y0, CRGB c0);
    void LineTo(CDC * pDC, CP2 p1);                //绘制直线,不含终点
```

```
    void LineTo(CDC * pDC, double x1, double y1, CRGB c1);
    CRGB LinearInterp(double m, double mStart, double mEnd, CRGB cStart, CRGB cEnd);
private:
    CPoint2 P0;                                  //起点
    CPoint2 P1;                                  //终点
};CLine::CLine(void)
{
}
CLine::~CLine(void)
{
}
void CLine::MoveTo(CDC * pDC, CP2 p0)            //绘制直线起点函数
{
    P0.x=ROUND(p0.x);
    P0.y=ROUND(p0.y);
    P0.c=p0.c;
}
void CLine::MoveTo(CDC * pDC, double x0, double y0, CRGB c0)     //重载函数
{
    MoveTo(pDC, CP2(x0, y0, c0));
}
void CLine::LineTo(CDC * pDC, CP2 p1)
{
    P1.x=ROUND(p1.x), P1.y=ROUND(p1.y), P1.c=p1.c;
    int dx=abs(P1.x-P0.x);
    int dy=abs(P1.y-P0.y);
    BOOL bInterchange=FALSE;
    int signX, signY;
    signX=(P1.x>P0.x)?1 : ((P1.x<P0.x)?-1 : 0);
    signY=(P1.y>P0.y)?1 : ((P1.y<P0.y)?-1 : 0);
    if (dy>dx)
    {
        int temp=dy;
        dy=dx;
        dx=temp;
        bInterchange=TRUE;
    }
    int e=dx-2 * dy;
    CPoint2 p=P0;                                //从起点开始绘制直线
    for (int i=1; i<=dx; i++)
    {
        pDC->SetPixelV(p.x, p.y, CRGBtoRGB(p.c));
        if (bInterchange)
        {
            p.y+=signY;
            p.c=LinearInterp(p.y, P0.y, P1.y, P0.c, P1.c);
        }
        else
        {
            p.x+=signX;
            p.c=LinearInterp(p.x, P0.x, P1.x, P0.c, P1.c);
```

```
        }
        if (e<0)
        {
            if (bInterchange)
                p.x+=signX;
            else
                p.y+=signY;
            e+=2*dx-2*dy;
        }
        else
            e-=2*dy;
    }
    P0=P1;
}
void CLine::LineTo(CDC*pDC, double x1, double y1, CRGB c1)   //重载函数
{
    LineTo(pDC, CP2(x1, y1, c1));
}
CRGB CLine::LinearInterp(double m, double mStart, double mEnd, CRGB cStart, CRGB
    cEnd) {
    CRGB color;
    color=(mEnd-m)/(mEnd-mStart)*cStart+(m-mStart)/(mEnd-mStart)*cEnd;
    return color;
}
```

程序分析：分别为 MoveTo()函数和 LineTo()函数定义了重载函数。LineTo()函数中的 LinearInterp()函数用于对直线两端点的颜色进行线性插值。程序中，点的计算全部采用浮点数运算。本段代码中的颜色转换的宏定义如下：

```
#define CRGBtoRGB(c) RGB(c.red*255, c.green*255, c.blue*255)
```

4. 设计 CTestView 的 OnDraw 函数

```
void CTestView::OnDraw(CDC*pDC)
{
    CTestDoc*pDoc=GetDocument();
    ASSERT_VALID(pDoc);
    if (!pDoc)
        return;
    // TODO: 在此处为本机数据添加绘制代码
    CRect rect;
    GetClientRect(&rect);
    pDC->SetMapMode(MM_ANISOTROPIC);              //自定义二维坐标系
    pDC->SetWindowExt(rect.Width(), rect.Height());
    pDC->SetViewportExt(rect.Width(), -rect.Height());
    pDC->SetViewportOrg(rect.Width()/2, rect.Height()/2);
    rect.OffsetRect(-rect.Width()/2, -rect.Height()/2);
    CP2 p0(0, 0, CRGB(1, 0, 0));                  //直线起点
    CP2 p1(0, 0, CRGB(0, 0, 1));                  //直线终点
    double R=300;                                 //圆的半径
    CLine*pLine=new CLine;
```

```
    for (int alpha=0; alpha<360; alpha+=30)
    {
        p1.x=R*cos(alpha*PI/180);
        p1.y=R*sin(alpha*PI/180);
        pLine->MoveTo(pDC, p0);
        pLine->LineTo(pDC, p1);
    }
    delete pLine;
}
```

程序说明：将圆周按 30°的等分角计算等分点。以自定义二维坐标系原点为起点(红色端点)，以圆周等分点为终点(蓝色端点)，绘制颜色渐变直线段。

五、案例小结

(1) 定义了浮点型的二维点类 CP2 和 CRGB 颜色类。

(2) 隐函数容易判断点与图元的相对位置，中点算法是一种隐函数判断算法。

(3) 中点算法使用网格中点判断上下像素或左右像素中哪个像素离理想直线更近。

(4) 本案例使用的是整型直线端点，算法中使用的是完全整数中点算法，避开了计算直线段的斜率。

六、案例拓展

在窗口客户区内按下鼠标左键绘制 3 个点。设定第 1 个顶点颜色为红色、第 2 个顶点颜色为绿色，第 3 个顶点颜色为蓝色。使用鼠标绘制边界光滑着色的三角形，鼠标移动到第一个顶点时闭合三角形，如图 5-3 所示。这里三角形的顶点在设备坐标系内定义。

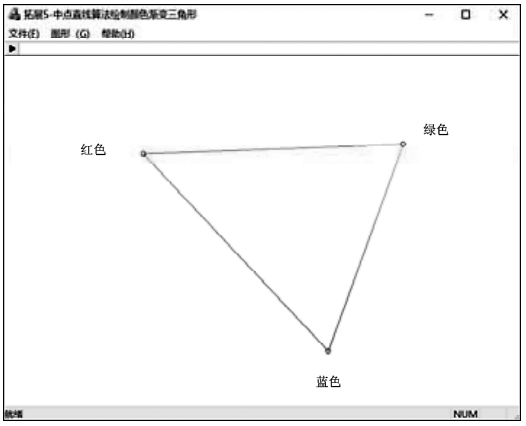

图 5-3　交互式绘制颜色渐变三角形