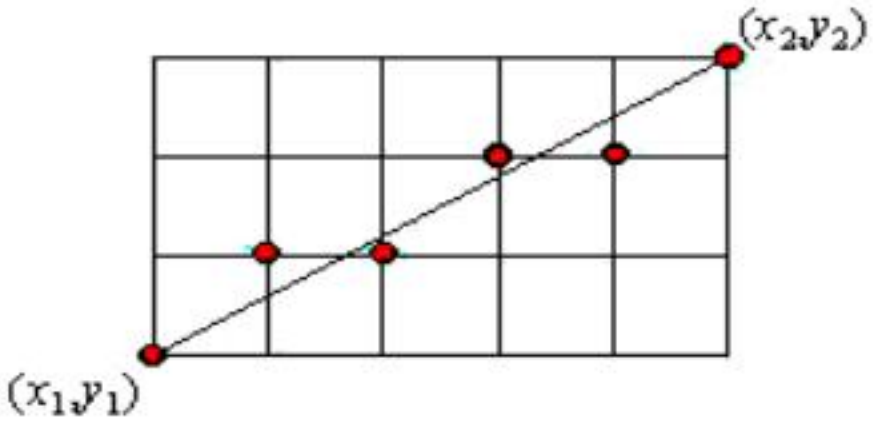
一、DDA 直线扫描转换算法原理和实践

1、原理

（1）首先考虑当直线的斜率的绝对值 **|k|≤1** 的情况：



设待画线段两端点的坐标值 (x1, y1) 和 (x2, y2)，假定 x1 < x2
则 **dy = (y2 – y1) / (x2 – x1)，|dy|≤1**
此时，x 坐标值每增加 1，y 坐标值增加 dy
也就是说点 (x1, y1) 的下一个点为 (x1 + 1, y1 + dy)
由于在绘制像素点时 x 和 y 坐标值必须为整型，所以需要将坐标值加上 0.5 再强转为 int（这样就能选出离当前坐标值最近的像素点）。
所以当点 **(int(xi+ 0.5), int(yi + 0.5))** 绘制完成后，下一个应该绘制的点为 **(int(xi + 1 + 0.5), int(yi + dy + 0.5))**
（2）当 **|k|≥1** 时，则应该计算 dx 的值，然后每次让 y 坐标值加 1，x 坐标值加 dx。

2、实践

1）题目要求

（1）在 MFC 单文档应用程序中，可利用鼠标左键拖拽在视图区绘制矩形；
（2）在绘制图形时实现橡皮线功能，即可实时看到绘制效果；
（3）实现视图重画功能，即在窗口最小化或被其它窗口遮挡后，恢复窗口时视图区中绘制的图形仍然应该存在；
（4）使用双缓冲技术，防止窗口重绘时出现闪屏现象。

2）分析

（1）首先在 view 窗口中重写鼠标左键按下、弹起以及鼠标移动函数。记录鼠标左键按下的点作为矩形的左上角点，鼠标移动时的点作为矩形的右下角点，利用这两个点确定矩形的四条边，然后使用 DDA 直线扫描转换算法完成直线的绘制。在鼠标左键弹起时，将此时记录的两个点（也就是矩形的左上角点和右下角点）存储到一个点集合中，该点集合存储了所有矩形的左上角点和右下角点，用于窗口重绘，实现视图重画功能。
（2）重画时加入双缓冲技术，并且重写 WM_ERASEBKGND 的消息处理函数，直接返回 true，禁止重绘时对窗口进行擦除，防止出现闪屏现象。
（3）为了实现橡皮线功能，应该选用 R2_NOTXORPEN 绘图模式：
pDC->SetROP2(R2_NOTXORPEN)
该模式在绘图时采用的实际颜色为：NOT(屏幕颜色 XOR 绘图颜色)
也就是说，在使用 pDC->SetPixel(x, y, color) 绘制像素点时，首先获取该像素点现在的屏幕颜色，然后与 color 取异或再取反。
举个例子：当前屏幕 (x, y) 点的颜色为白色，想要绘制成红色，此时先将白色和红色取异或也就是红色的反色，然后再取反又变成了红色；而如果当前屏幕 (x, y) 点的颜色为红色，想要绘制成红色，此时先将红色和红色取异或也就是黑色，然后再取反就变成了白色，也就实现了擦除的功能。
这样每次鼠标移动时，首先将上次所画矩形重画一遍（也就是先擦除掉），再绘制现在的矩形即可。

3）代码实现

（1）首先在 CTestView 类中添加四个成员变量：

```
CPoint m_startPoint, m_endPoint;   //起始点和终止点
bool m_LButtonDown;   //绘制矩形时检测鼠标左键啊是否按下，初始为 false
vector<CPoint> m_rectPoints;   //绘制矩形所存储的点集合，用于实现视图重画功能
```

（2）DDA 直线扫描转换算法实现：

```
void CTestView::DDALine(CDC * pDC, int x1, int y1, int x2, int y2, COLORREF color)
{
    pDC->SetROP2(R2_NOTXORPEN); //设置绘图模式

    double dx, dy, e, x, y;
    dx = x2 - x1;
    dy = y2 - y1;
    e = (fabs(dx) > fabs(dy)) ? fabs(dx) : fabs(dy);
    dx /= e;
    dy /= e;
    x = x1;
    y = y1;
    for (int i = 1; i <= e; i++)
    {
        pDC->SetPixel((int)(x + 0.5), (int)(y + 0.5), color);
```

```
            x += dx;
            y += dy;
        }
}
```

（3）添加鼠标响应事件：

```
void CTestView::OnLButtonDown(UINT nFlags, CPoint point)
{
    this->SetCapture();//捕捉鼠标
    m_startPoint = point;
    m_endPoint = point;
    m_LButtonDown = true;

    CView::OnLButtonDown(nFlags, point);
}

void CTestView::OnMouseMove(UINT nFlags, CPoint point)
{
    if (m_LButtonDown) //绘制矩形和圆功能
    {
        CDC* pDC = this->GetDC();

        //绘制矩形，首先擦除上一次所画矩形，采用 R2_NOTXORPEN 绘图模式
        DDALine(pDC, m_startPoint.x, m_startPoint.y, m_endPoint.x, m_startPoint.y,
RGB(255, 0, 0));
        DDALine(pDC, m_startPoint.x, m_startPoint.y, m_startPoint.x, m_endPoint.y,
RGB(255, 0, 0));
        DDALine(pDC, m_endPoint.x, m_endPoint.y, m_endPoint.x, m_startPoint.y, RGB(255,
0, 0));
        DDALine(pDC, m_endPoint.x, m_endPoint.y, m_startPoint.x, m_endPoint.y, RGB(255,
0, 0));

        DDALine(pDC, m_startPoint.x, m_startPoint.y, point.x, m_startPoint.y, RGB(255, 0,
0));
        DDALine(pDC, m_startPoint.x, m_startPoint.y, m_startPoint.x, point.y, RGB(255, 0,
0));
        DDALine(pDC, point.x, point.y, point.x, m_startPoint.y, RGB(255, 0, 0));
        DDALine(pDC, point.x, point.y, m_startPoint.x, point.y, RGB(255, 0, 0));


        //保存新的直线段终点
        m_endPoint = point;
    }

    CView::OnMouseMove(nFlags, point);
```

```
}

void CTestView::OnLButtonUp(UINT nFlags, CPoint point)
{
    ReleaseCapture();//释放鼠标
    m_LButtonDown = false;

    //同时存储起始点和终止点，用于重绘
    m_rectPoints.push_back(m_startPoint);
    m_rectPoints.push_back(m_endPoint);

    CView::OnLButtonUp(nFlags, point);
}
```

（4）实现重绘功能：

① 重写 WM_ERASEBKGND 的消息处理函数

```
BOOL CTestView::OnEraseBkgnd(CDC* pDC)
{
    return true;
    //return CView::OnEraseBkgnd(pDC);
}
```

② 实现 OnDraw 函数：

```
void CTestView::OnDraw(CDC* pDC)
{
    CTestDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    if (!pDoc)
        return;

    //重绘时使用双缓冲技术
    CRect rect;
    GetClientRect(&rect); //得到客户端矩形区域
    CDC dcMem; //用于缓冲作图的内存 DC
    CBitmap bmp;//内存中承载临时图象的位图
    dcMem.CreateCompatibleDC(pDC);//依附窗口 DC 创建兼容内存 DC
    //创建兼容位图，使用 pDC 而不是 dcMem，因为如果使用 dcMem 的话位图只能是黑白的
    bmp.CreateCompatibleBitmap(pDC, rect.Width(), rect.Height());
    dcMem.SelectObject(&bmp);//将位图选择进内存 DC
    dcMem.FillSolidRect(rect, RGB(255, 255, 255));//按原来背景填充客户区，不然会是黑色

    //重绘矩形
    for (int i = 0; i < m_rectPoints.size(); i += 2)
    {
        CPoint startPoint = m_rectPoints.at(i);
```

```
            CPoint endPoint = m_rectPoints.at(i + 1);
            DDALine(&dcMem, startPoint.x, startPoint.y, endPoint.x, startPoint.y, RGB(255, 0, 0));
            DDALine(&dcMem, startPoint.x, startPoint.y, startPoint.x, endPoint.y, RGB(255, 0, 0));
            DDALine(&dcMem, endPoint.x, endPoint.y, endPoint.x, startPoint.y, RGB(255, 0, 0));
            DDALine(&dcMem, endPoint.x, endPoint.y, startPoint.x, endPoint.y, RGB(255, 0, 0));
        }

        //将内存 DC 上的图象拷贝到前台
        pDC->BitBlt(0, 0, rect.Width(), rect.Height(), &dcMem, 0, 0, SRCCOPY);
        dcMem.DeleteDC();//删除 DC
        bmp.DeleteObject();//删除位图
}
```

4）效果