实验内容：

已实现一个在VC++环境中采用MFC建立的工程，此工程实现了所有的用户接口功能，需填写其中的两个函数：CCGPainterView::DrawLine()和CCGPainterView::DrawCircle()：

1. 编程实现Bresenham直线生成算法或其它直线生成算法；
2. 编程实现Bresenham圆弧生成算法或其它圆弧生成算法。

实验目的：

1. 熟悉画直线和画圆这两种最基本的图形算法；
2. 体会计算机中图形“光栅化”的特点；
3. 练习算法的编码以及调试的整个流程。

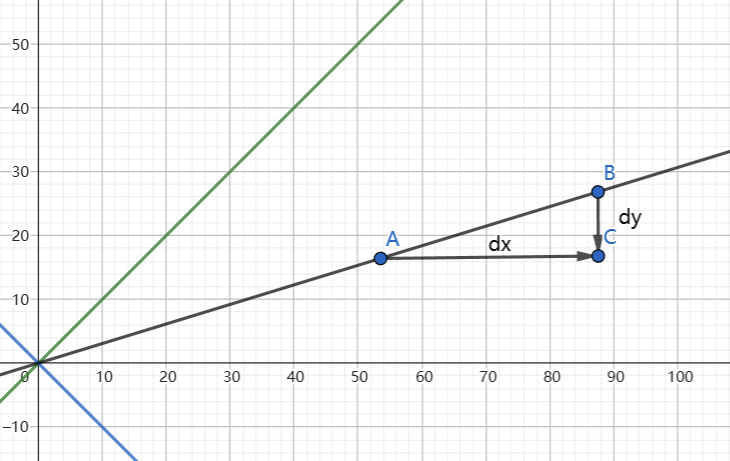
实现方法

算法流程图，说明算法各函数的功能；

1. Bresenham画直线算法

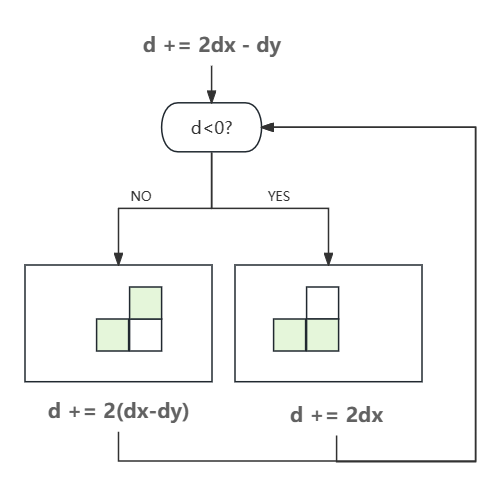
BresenhamDrawLine是实现Bresenham画直线算法的函数，以下为算法实现的详细解释。

1. 对于第一象限的斜率小于1的直线来说，实现bresenham绘制直线的流程如下：



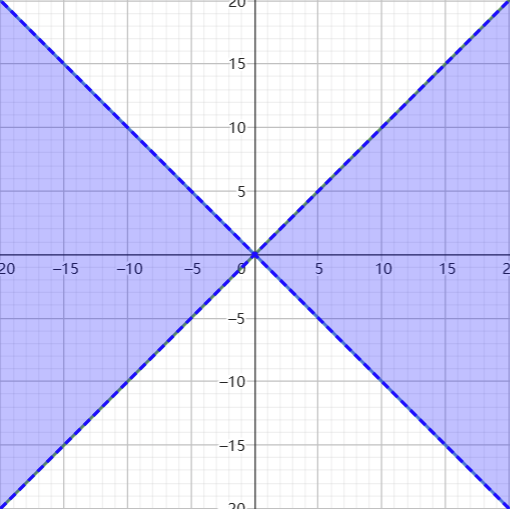
初始化起点和终点坐标，计算dx和dy（直线线段在水平方向和垂直方向的跨度）。

绘制过程中以x轴（水平方向）为增量，使用一个额外的变量d来跟踪当前位置离理想直线的偏离。根据d的值，决定下一个像素点的位置。如果d小于0，y值保持不变，即直线向右绘制；如果d大于等于0，y值加一，即直线向右上方绘制。d的初始值为2dx-y, 每次绘制时都更新d值（如下图所示）。



持续重复以上步骤直到达到终点坐标为止。

1. 推广到如下图所示蓝色区域（斜率绝对值小于1的区域），只需要改变直线绘制的方向即可。举例来说，如果获取到的dx和dy都小于0，意味着是从右向左，从下到上绘制的，我们在进行循环迭代步骤时可以从最右端的x开始向左遍历水平轴上的坐标，垂直轴上可以绘制的方向选择为向左和向左上。代码中定义了xstep和ystep，将这些变量赋值为+1或者-1来决定步向。



1. 再推广到剩下的斜率绝对值大于一的区域：

上面的直线我们都是用水平增量来绘制的，当斜率绝对值大于一，可以看作直线与斜率绝对值小于一的直线关于y = x、y = -x对称。因此我们用对称的思想，只需要把增量改为垂直方向的y，并把上文中误差项d的计算公式的x和y对调即可。

1. 在屏幕坐标系上的绘制逻辑与普通坐标系有所区别。

屏幕坐标系的y轴正方向为垂直向下，如dy小于0是从上往下绘制直线而不是从下往上了，向右上方绘制是y--不是y++，将类似的代码中改为相反即可。

核心代码片段：

while (x != x2) {

x += xStep;

if (d < 0) {

d += increE;

}

else {

d += increNE;

y += yStep;

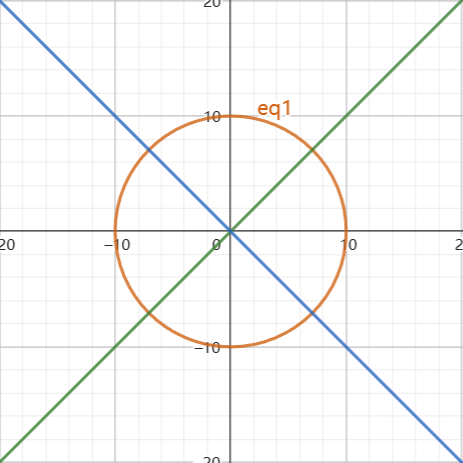
}

pDC->SetPixelV(x, y, cLineColor);

}

1. Bresenham画圆算法

由于圆的对称性，只对八分之一圆进行绘制，再根据对称拓展到另外七个区域。代码中绘制部分为eq1所对应圆弧。



与上面的画直线方法类似，画圆算法也只截取八分之一的区域来入手。但是与上面的直线绘制不同，由于圆没有方向性，我们只绘制八分之一的圆弧就可以通过对称画出整个圆。因为屏幕坐标系上不是与上图一致的原点坐标系，我们定义了一个paint8Pixel函数用于在屏幕上绘制圆的八个对称像素点，这个函数可以完成屏幕坐标系对上述坐标系的映射。给定圆心坐标ptOrigin，以及相对于圆心的偏移量x和y，函数在八个对称位置上设置像素的颜色。

void paint8Pixel(CDC\* pDC, CPoint ptOrigin, int x, int y, COLORREF cLineColor)

{

pDC->SetPixelV(ptOrigin.x + x, ptOrigin.y - y, cLineColor);

pDC->SetPixelV(ptOrigin.x + y, ptOrigin.y - x, cLineColor);

pDC->SetPixelV(ptOrigin.x + x, ptOrigin.y + y, cLineColor);

pDC->SetPixelV(ptOrigin.x + y, ptOrigin.y + x, cLineColor);

pDC->SetPixelV(ptOrigin.x - x, ptOrigin.y + y, cLineColor);

pDC->SetPixelV(ptOrigin.x - y, ptOrigin.y + x, cLineColor);

pDC->SetPixelV(ptOrigin.x - x, ptOrigin.y - y, cLineColor);

pDC->SetPixelV(ptOrigin.x - y, ptOrigin.y - x, cLineColor);

}

BresenhamDrawCircle函数是用来绘制圆的主要函数。在函数开始时，初始化变量x为0，y为半径iRadius。

设置变量d来记录损失值，初始值为1 - iRadius。随后，通过循环迭代来更新d的值。在每次迭代中，检查d的值。如果d小于0，表示下一个点位于圆的内部，然后d的值更新为2 \* x + 1。如果d大于等于0，表示下一个点位于圆的边界或外部，然后d的值更新为2 \* (x - y) + 1，同时y减少1。

在每个迭代中，调用paint8Pixel函数来绘制新的八个对称像素点。直到x >y时停止循环。

注：代码中的y--代表y向下绘制，不是屏幕坐标系中的向上绘制，这里的代码还是对应y轴垂直向上的坐标系，在paint8Pixel中才转换为屏幕坐标系。

核心代码：

while (++x && x <= y)

{

if (d < 0)

{

d += 2 \* x + 1;

}

else

{

y--;

d += 2 \* (x - y) + 1;

}

paint8Pixel(pDC, ptOrigin, x, y, cLineColor);

}

两个函数实现后，在CCGPainterView::DrawLine()和CCGPainterView::DrawCircle()中分别调用BresenhamDrawLine函数和BresenhamDrawCircle函数。

实验结果

如图所示，程序可正常完成直线和圆的绘制功能。

