实验一：直线生成算法

姓 名： 陈嘉乐

学 号： 2021218152

班 级： 计科21-3班

实验地点： 计算机楼

实验时间： 2024.3.19

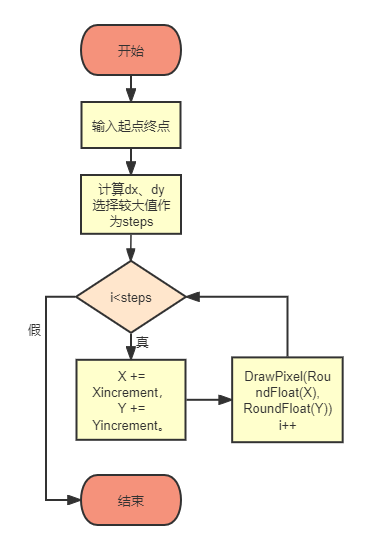
1. 实验目的和要求

理解直线生成的原理；掌握典型直线生成算法；掌握步处理、分析实验数据的能力；

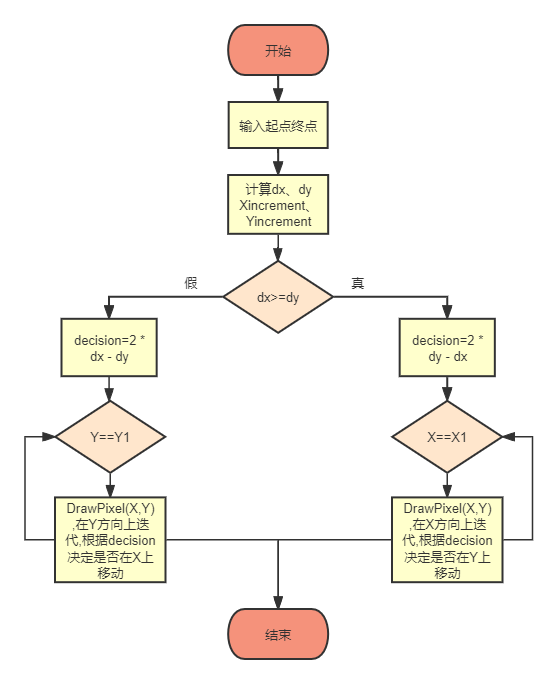
编程实现DDA算法、Bresenham中点算法；对于给定起点和终点的直线，分别调用DDA算法和Bresenham中点算法进行批量绘制，并记录两种算法的绘制时间；利用excel等数据分析软件，将试验结果编制成表格，并绘制折线图比较两种算法的性能。

1. 实验环境和工具
   * 开发环境:VisualC++6.0
   * 实验平台:ExperimentFrame\_0ne(自制平台)
2. 实验结果
   1. 程序流程图

DDA算法程序流程图：



Bresenham算法程序流程图：



* 1. 程序代码

void CExperiment\_Frame\_OneView::DDA(int X0, int Y0, int X1, int Y1)

{

//----------请实现DDA算法------------//

// Calculate dx and dy

int dx = X1 - X0;

int dy = Y1 - Y0;

// Determine the number of steps needed

int steps = abs(dx) > abs(dy) ? abs(dx) : abs(dy);

// Calculate increment in x and y for each step

float Xincrement = dx / static\_cast<float>(steps);

float Yincrement = dy / static\_cast<float>(steps);

// Initial point

float X = X0;

float Y = Y0;

// Plot the initial point

DrawPixel(RoundFloat(X), RoundFloat(Y));

// Iterate through each step

for (int i = 1; i <= steps; i++)

{

X += Xincrement;

Y += Yincrement;

// Plot the point

DrawPixel(RoundFloat(X), RoundFloat(Y));

}

}

void CExperiment\_Frame\_OneView::Mid\_Bresenham(int X0, int Y0, int X1, int Y1)

{

//-------请实现Mid\_Bresenham算法-------//

// Calculate dx and dy

int dx = X1 - X0;

int dy = Y1 - Y0;

// Determine the increments for x and y

int Xincrement = dx > 0 ? 1 : -1;

int Yincrement = dy > 0 ? 1 : -1;

// Make dx and dy positive

dx = abs(dx);

dy = abs(dy);

// Variables for decision parameter and initial values

int decision, X, Y;

if (dx >= dy) // Slope <= 1

{

decision = 2 \* dy - dx;

Y = Y0;

// Iterate through each x

for (X = X0; X != X1; X += Xincrement)

{

// Plot the point

DrawPixel(X, Y);

// Adjust the decision parameter

if (decision > 0)

{

Y += Yincrement;

decision -= 2 \* dx;

}

decision += 2 \* dy;

}

DrawPixel(X, Y); // Plot the last point

}

else // Slope > 1

{

decision = 2 \* dx - dy;

X = X0;

// Iterate through each y

for (Y = Y0; Y != Y1; Y += Yincrement)

{

// Plot the point

DrawPixel(X, Y);

// Adjust the decision parameter

if (decision > 0)

{

X += Xincrement;

decision -= 2 \* dy;

}

decision += 2 \* dx;

}

DrawPixel(X, Y); // Plot the last point

}

}

// Simple round function for VC6.0

int RoundFloat(float x)

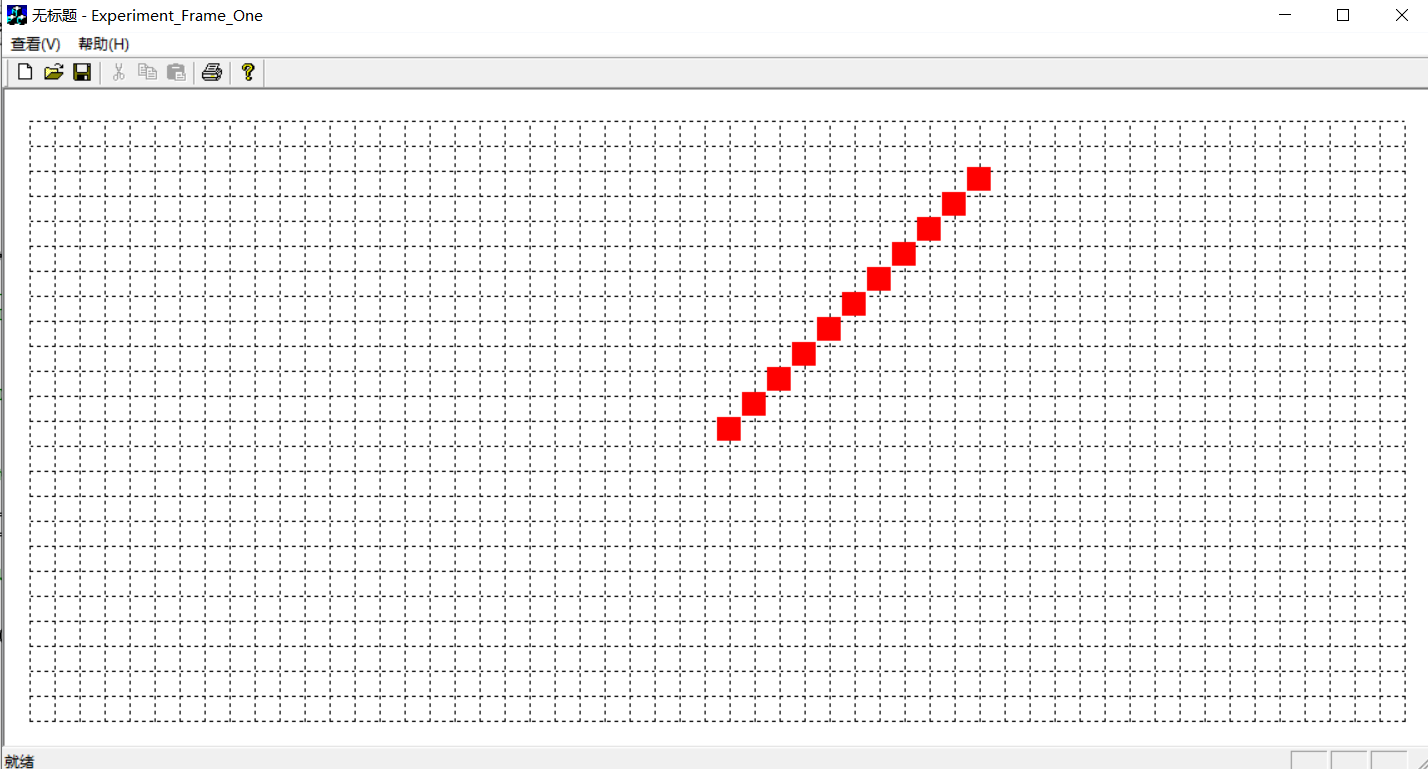
{

return static\_cast<int>(x + 0.5);

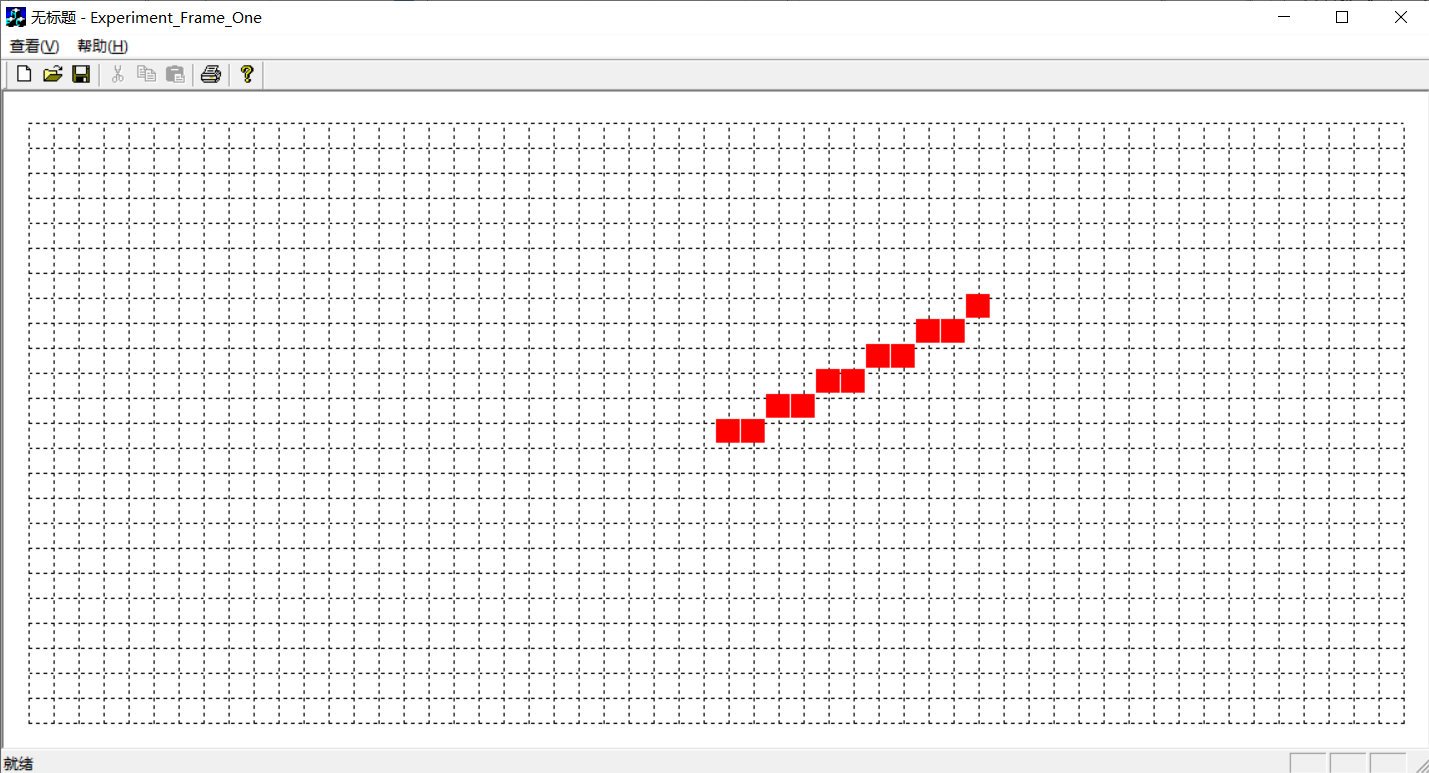
}

* 1. 运行结果

DDA算法：



Bresenham中点算法：



* 1. 运行结果分析

实验与预期相同，符合实验结果

1. 思考题（可选）

4.1水平、垂直直线的生成方法

在DDA算法中：

对于水平直线（即dy为0），Yincrement会是0，因为在y方向上没有变化。在循环中，X会按照Xincrement逐步增加，而Y保持不变，从而绘制出一条水平线。

对于垂直直线（即dx为0），Xincrement会是0，因为在x方向上没有变化。在循环中，Y会按照Yincrement逐步增加或减少（取决于dy的正负），而X保持不变，从而绘制出一条垂直线。

在Bresenham算法中：

由于Bresenham算法是根据直线的斜率和端点来逐步绘制直线上的点的，对于水平和垂直直线，它也能够正确工作。

对于水平直线（即dy为0的情况），由于dx一定大于dy，算法会进入if (dx >= dy)的代码块。在这个块内，由于dy为0，decision参数在计算时也会为0（因为2 \* dy - dx为0），并且后续更新decision时由于2 \* dy为0，decision不会改变。因此，循环将只在x方向上迭代，y坐标保持不变，从而绘制出一条水平线。

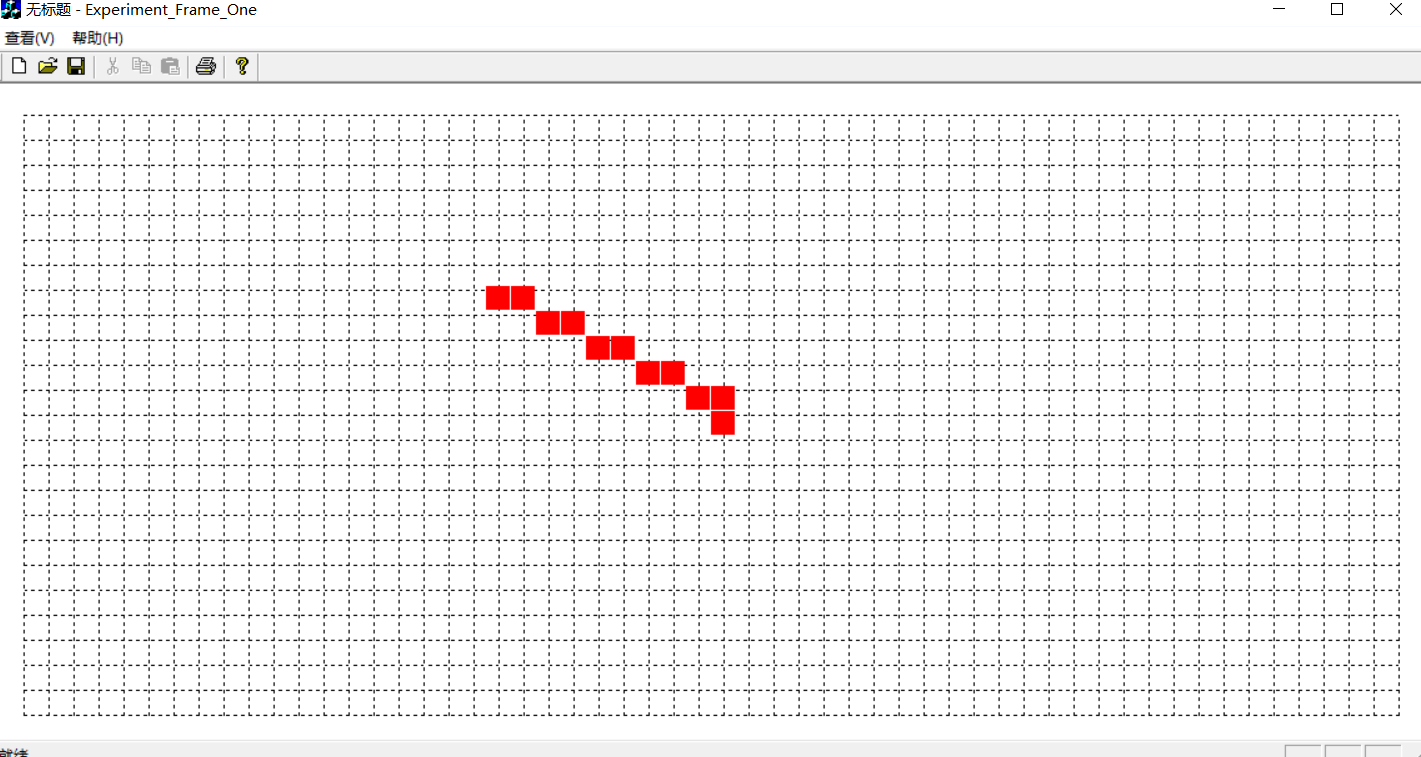
对于垂直直线（即dx为0的情况），else代码块将被执行，因为dx小于dy。然而，在这个块内，由于dx为0，decision参数在计算时同样为0（因为2 \* dx - dy为0），并且后续更新decision时由于2 \* dx为0，decision同样不会改变。循环将只在y方向上迭代，x坐标保持不变，从而绘制出一条垂直线。

4.2 用DDA算法实现斜率为负的直线的生成方法

斜率的正负会影响dx和dy的符号，而dx和dy的符号会进一步影响Xincrement和Yincrement的符号。当斜率为负时，dx和dy的符号会相反，导致在每一步中，x坐标和y坐标的增减方向也会相反。

例如，如果起点(X0, Y0)在终点(X1, Y1)的右上方（即斜率为负），那么dx会是一个正数，而dy会是一个负数。这样，Xincrement会是正数（因为x坐标向右增加），而Yincrement会是负数（因为y坐标向下减少）。在循环中，x坐标会逐渐增加，而y坐标会逐渐减少，从而绘制出一条向下倾斜的直线。

它并不直接考虑斜率的正负，而是根据dx和dy的值来计算增量，并据此更新坐标。因此，无论是正斜率、负斜率，还是接近水平或垂直的直线，DDA算法都能够正确地绘制出来。



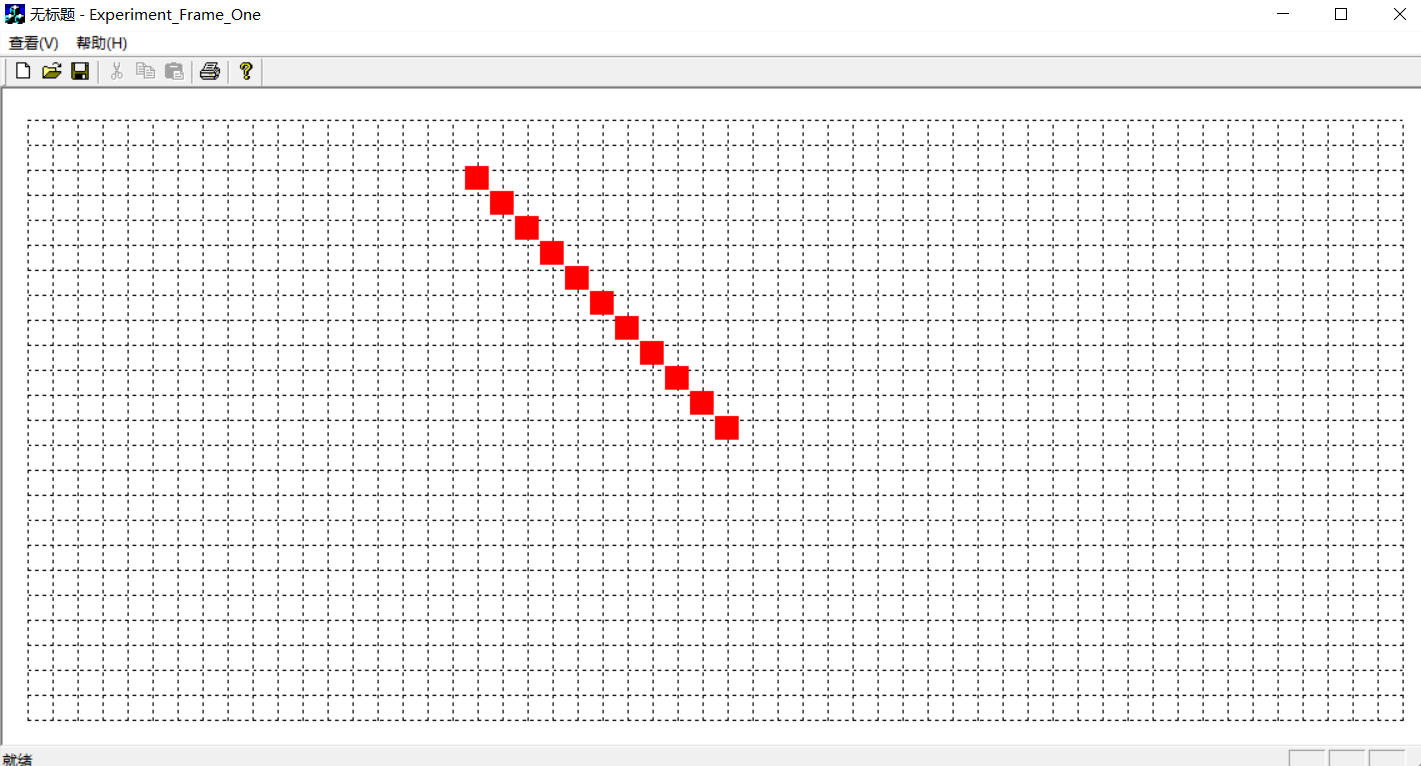
4.3用中点Bresenham算法实现斜率为负的直线的生成方法

通过dx和dy的正负来处理。算法并不直接处理斜率的正负，而是处理直线端点之间的差值dx和dy，并根据这些差值来确定每一步应该向哪个方向移动。

具体来说，当dx或dy为负数时，Xincrement或Yincrement会被设置为-1，这表示在迭代过程中，x或y的坐标将减小而不是增加。这是通过条件语句dx > 0 ? 1 : -1和dy > 0 ? 1 : -1来实现的。

例如，假设直线的起点在终点的右上方（即X0 > X1且Y0 > Y1），那么dx和dy都将是负数。在这种情况下，Xincrement和Yincrement都将被设置为-1，这意味着在循环中，x和y的坐标都将逐步减小，直到达到终点的坐标。

决策参数decision根据dx和dy的绝对值来计算，而不是它们的实际值。因此，无论dx和dy是正还是负，决策逻辑都是一样的。算法关注的是直线端点之间的相对位置，而不是它们在坐标系中的绝对位置或方向。



4.4测试比较算法的性能

* 提示1：因为绘制1条直线时间很短，所以需要绘制大量直线才能比较它们之间的性能；
* 提示2: DrawPixel需要耗费时间，但它的时间性能和直线绘制算法无关，因此在比较不同算法性能时，应该屏蔽它的影响，如何屏蔽？

// 性能测试函数

void TestPerformance()

{

const int NUM\_LINES = 100000; // 测试的直线数量

clock\_t start, end;

double ddaTotalTime = 0, bresenhamTotalTime = 0;

// 测试 DDA 算法

start = clock();

for (int i = 0; i < NUM\_LINES; i++)

{

// 生成随机坐标

int X0 = rand()%100;

int Y0 = rand()%100;

int X1 = rand()%100;

int Y1 = rand()%100;

// 调用 DDA 函数

DDA(X0, Y0, X1, Y1);

}

end = clock();

ddaTotalTime = static\_cast<double>(end - start) / CLOCKS\_PER\_SEC;

// 测试 Mid\_Bresenham 算法

start = clock();

for (int i = 0; i < NUM\_LINES; i++)

{

// 生成随机坐标

int X0 = rand()%100;

int Y0 = rand()%100;

int X1 = rand()%100;

int Y1 = rand()%100;

// 调用 Mid\_Bresenham 函数

Mid\_Bresenham(X0, Y0, X1, Y1);

}

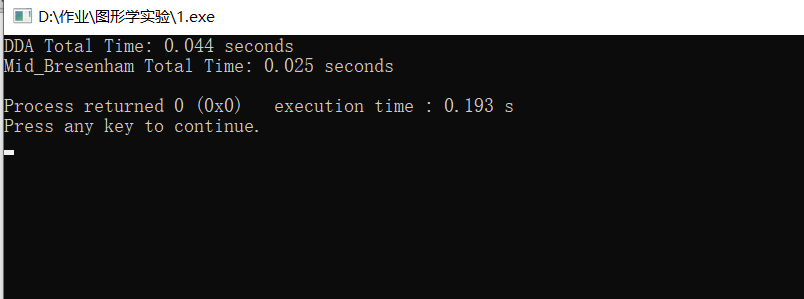
end = clock();

bresenhamTotalTime = static\_cast<double>(end - start) / CLOCKS\_PER\_SEC;

cout << "DDA Total Time: " << ddaTotalTime << " seconds" << endl;

cout << "Mid\_Bresenham Total Time: " << bresenhamTotalTime << " seconds" << endl;

}



4.5 把坐标系转换为具备四象限模式的坐标系

在DrawPixel函数里将起始坐标设置为画框的中央即可实现具备四象限模式的坐标系。代码如下：

void CExperiment\_Frame\_OneView::DrawPixel(int X, int Y)

{

CDC \*pDC = GetDC();

CBrush \*pNewBrush = new CBrush;

pNewBrush->CreateSolidBrush(m\_PointColor);

CBrush \*pOldBrush = pDC->SelectObject(pNewBrush);

CPen \*pNewPen = new CPen;

pNewPen->CreatePen(PS\_NULL, 1, RGB(0, 0, 0));

CPen \*pOldPen = pDC->SelectObject(pNewPen);

CRect ClientRect;

GetClientRect(&ClientRect);

// 计算中心点坐标

int centerX = (ClientRect.left + ClientRect.right) / 2;

int centerY = (ClientRect.top + ClientRect.bottom) / 2;

CRect Point;

Point.left = centerX + X \* m\_iPointSize;

Point.right = Point.left + m\_iPointSize;

Point.top = centerY - Y \* m\_iPointSize;

Point.bottom = Point.top + m\_iPointSize;

pDC->Rectangle(&Point);

pDC->SelectObject(pOldBrush);

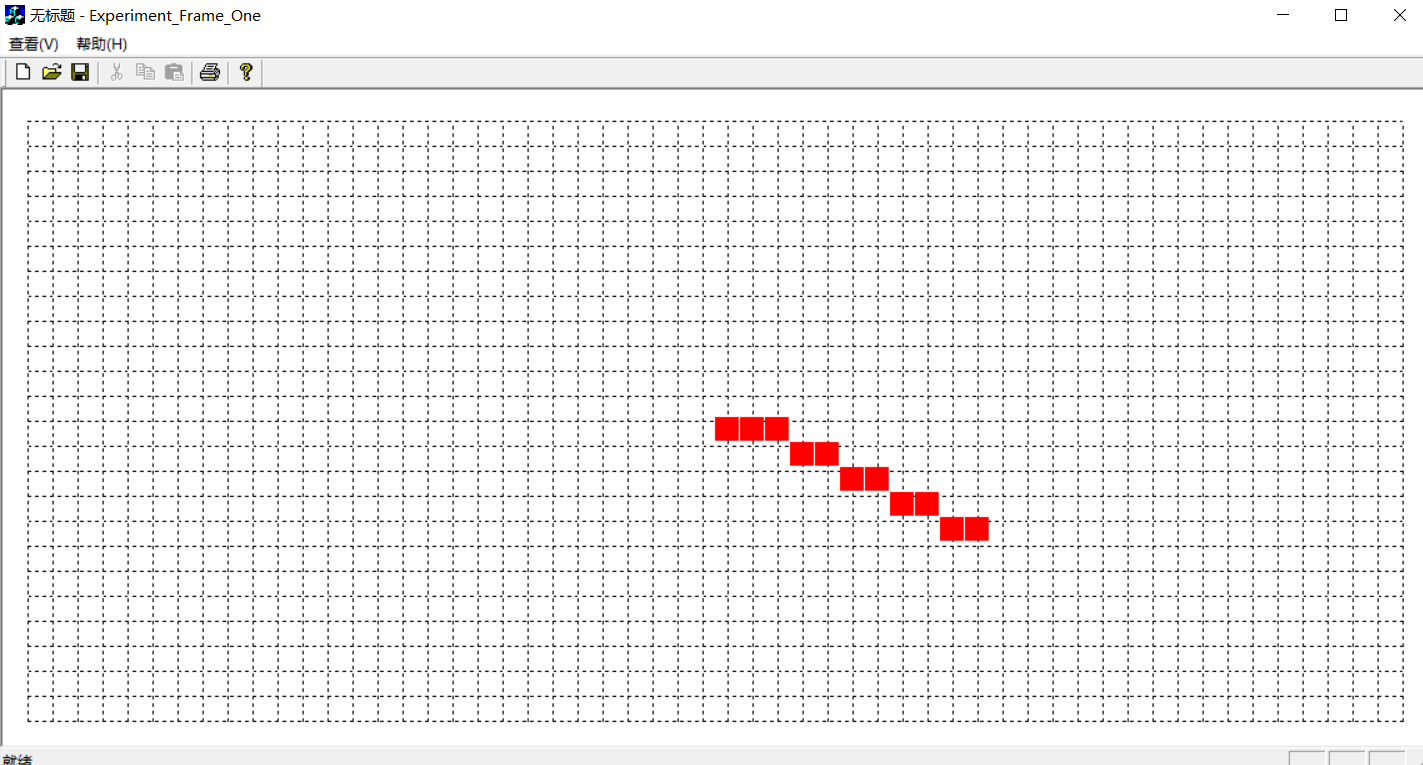
pNewBrush->DeleteObject();

pDC->SelectObject(pOldPen);

pOldPen->DeleteObject();

}

效果：



1. 实验心得

在本次实验中，我深入学习和实践了DDA算法和Bresenham算法，这两种算法都是计算机图形学中用于绘制直线的重要算法。通过编写代码并实际运行，我对这两种算法有了更加直观和深刻的理解。

DDA算法通过计算直线上每一点的坐标来绘制直线，它依赖于增量的概念，通过逐步累加增量来得到新的坐标点。算法的优点在于其简单直观，能够处理任意斜率的直线，包括水平、垂直以及斜率为负的情况。然而，由于DDA算法涉及到浮点数的运算，这可能会导致精度问题，尤其是在一些对精度要求较高的场合。Bresenham算法是一种基于决策参数的直线绘制算法，它通过比较决策参数与阈值的大小来决定每一步的移动方向。与DDA算法不同，Bresenham算法完全使用整数运算，因此避免了浮点数的精度问题。在编程实现过程中，我发现Bresenham算法的代码相对复杂一些，但它的效率更高，尤其是在处理大量像素点的绘制时。此外，Bresenham算法还能够处理斜率为负的情况，只需在计算增量时考虑正负即可。

**表. 实验成绩评定表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 评价内容 | | 权重 | 得分 |
| **验收** | 实验原理是否理解；程序能否运行；实验结果是否正确；任务是否全部完成。 | | 0.5 |  |
| **实验报告** | 1 | 报告格式是否规范，语言使用是否规范，行文是否流畅，是否图文并茂； | 0.2 |  |
| 2 | 实验原理、实验步骤描述是否正确、详实；  程序流程图是否规范，代码实现是否正确；  实验数据记录是否完整，实验结果是否正确；  实验结果的分析、对比是否充分； | 0.2 |  |
| 3 | 实验体会是否正确，是否提出了自己独到见解。 | 0.1 |  |
| 合计 |  | | | |
| 指导教师（签章）： 年 月 日 | | | | |