实验一：直线生成算法

姓 名： 侯腾跃

学 号： 2022217477

班 级： 计科22-4班

实验地点： 计算机中心楼三机房

实验时间： 2025.4.9

1. 实验目的和要求

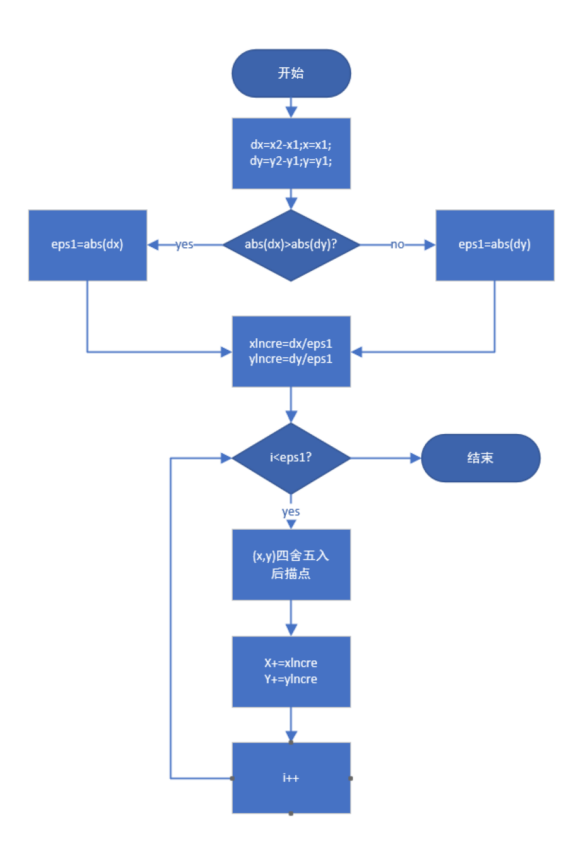
理解直线生成的原理；掌握典型直线生成算法；掌握步处理、分析 实验数据的能力； 编程实现 DDA 算法、Bresenham 中点算法；对于给定起点和终点的 直线，分别调用 DDA 算法和 Bresenham 中点算法进行批量绘制，并记 录两种算法的绘制时间；利用 excel 等数据分析软件，将试验结果编 制成表格，并绘制折线图比较两种算法的性能。

1. 实验环境和工具

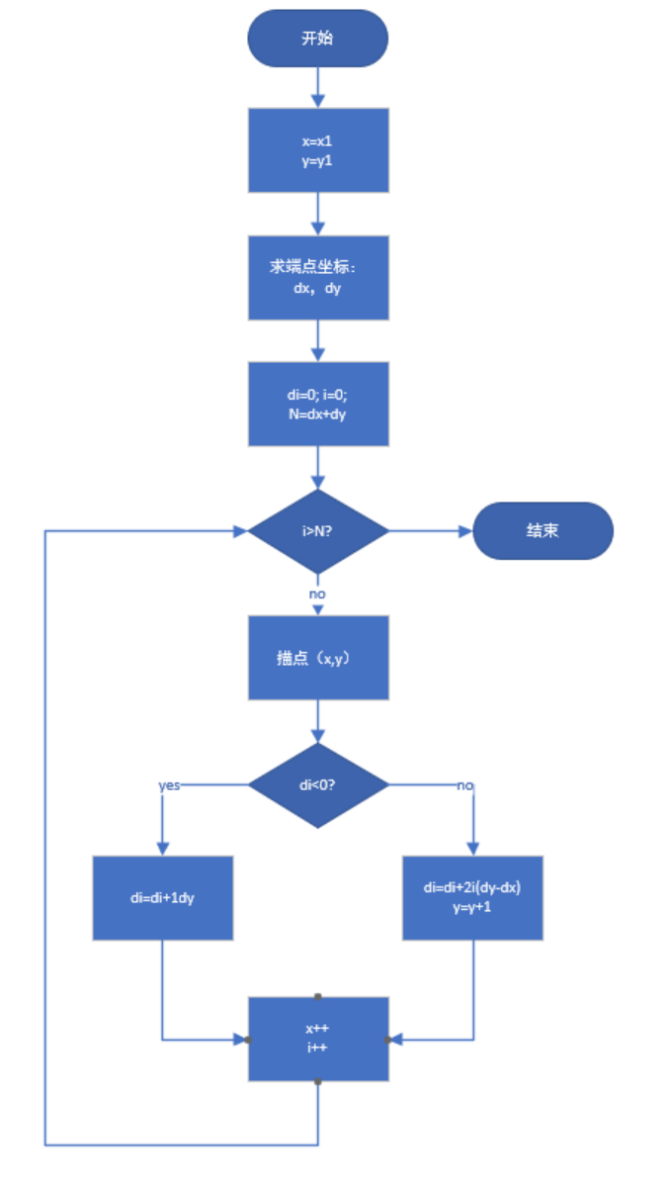
开发环境：Visual C++ 6.0

实验平台：Experiment\_Frame\_One（自制平台）

1. 实验结果
   1. 程序流程图

DDA算法流程图：  


Bresenham 中点算法流程图：



* 1. 程序代码

DDA算法：

float length, X, Y, incX, incY, i;

if(abs(X1 - X0) > abs(Y1 - Y0))

length = abs(X1 - X0);

else

length = abs(Y1 - Y0);

incX = (X1 - X0) / length;

incY = (Y1 - Y0) / length;

X = X0;

Y = Y0;

for(i = 1; i <= length; i++){

DrawPixel(int(X + 0.5), int(Y + 0.5));

X += incX;

Y += incY;

}

Bresenham 中点算法：

int dx, dy, d, UpIncre, DownIncre, X, Y;

if (X0 > X1){

X = X1; X1 = X0; X0 = X;

Y = Y1; Y1 = Y0; Y0 = Y;

}

X = X0;

Y = Y0;

dx = X1 - X0;

dy = Y1 - Y0;

d = dx - 2 \* dy;

UpIncre = 2 \* dx - 2 \* dy;

DownIncre = -2 \* dy;

while (X <= X1){

DrawPixel(X, Y);

X++;

if (d < 0){

Y++;

d += UpIncre;

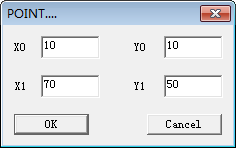
}else

d += DownIncre;

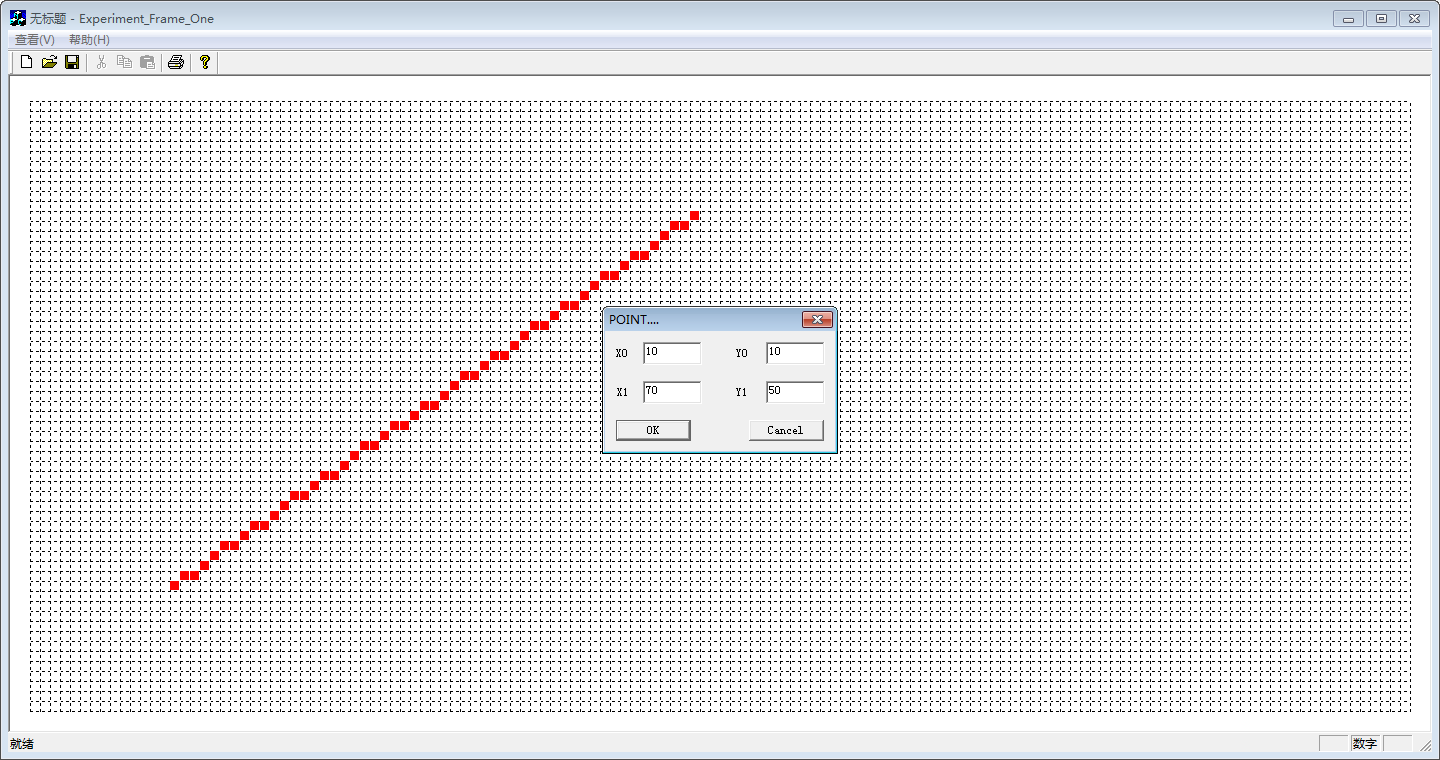
}

* 1. 运行结果

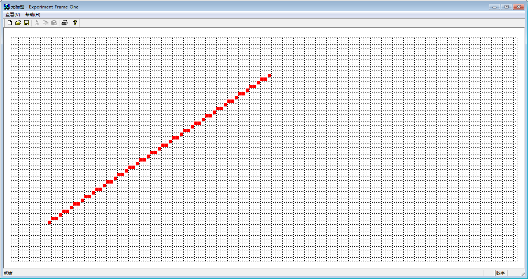
斜率大于0小于1：



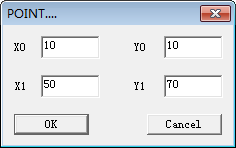
DDA：



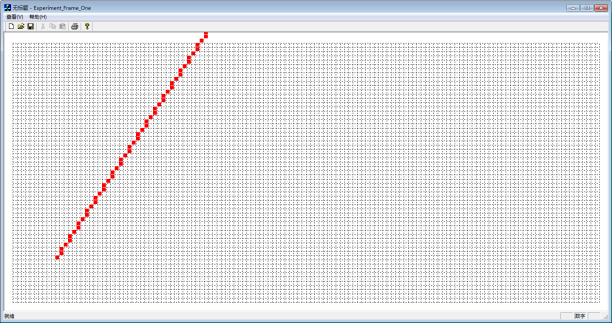
Bresenham：



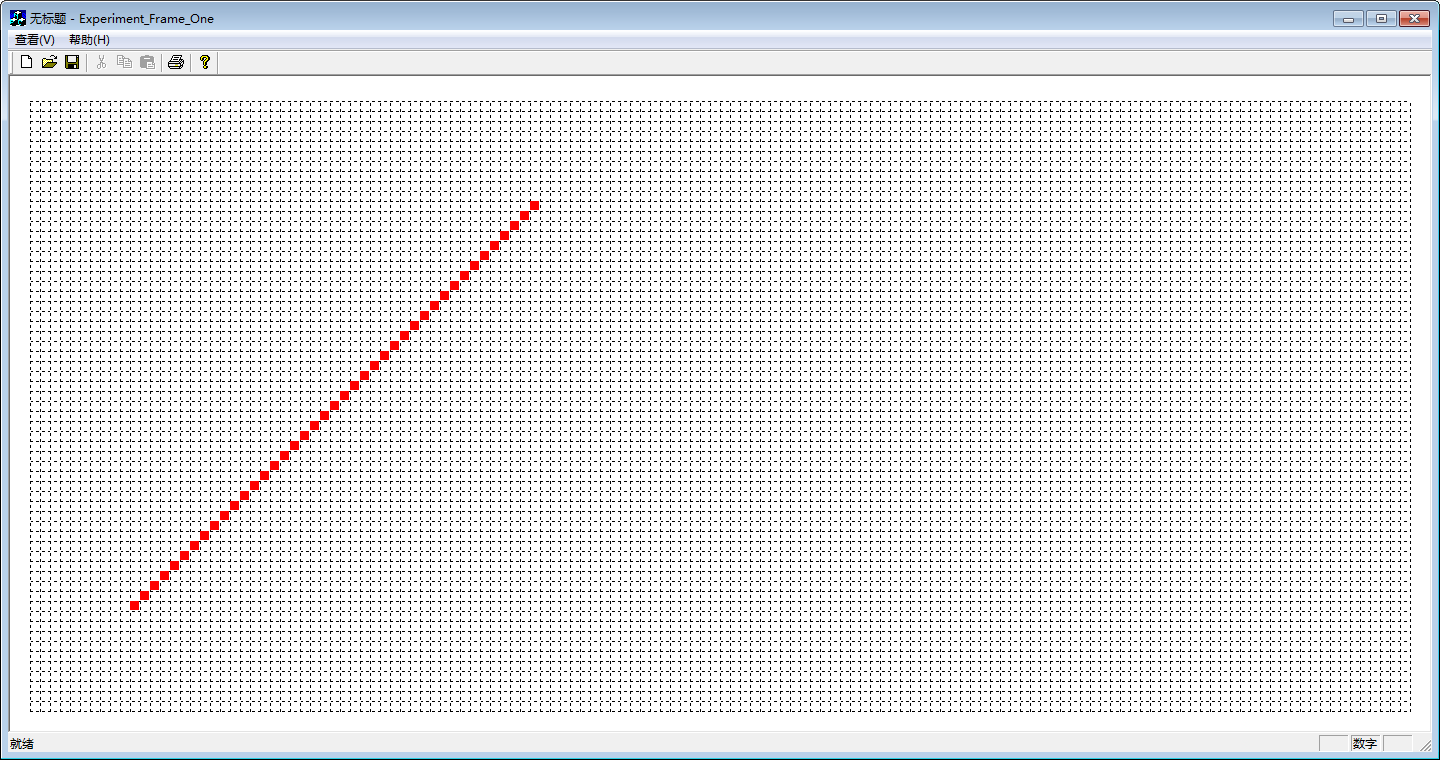
斜率大于1：



DDA：



Bresenham：



* 1. 运行结果分析

当斜率大于0小于时，DDA算法与Bresenham中点算法均可画出正确的直线，当斜率大于1时，Bresenham中点算法无法画出正确直线，需要改进。

1. 思考题（可选）

4.1水平、垂直直线的生成方法

**水平直线的生成方法**

输入参数：起点 (x1,y) 和终点 (x2,y)。

计算范围：确定 x 的范围，即从 x1到 x2 （假设 x1≤x2）。

绘制像素：对于范围内的每个 x 值，绘制像素点 (x,y)。

**垂直直线的生成方法**

输入参数：起点 (x,y 1) 和终点 (x,y2)。

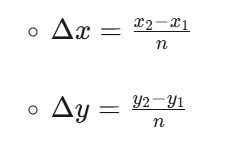
计算范围：确定 y 的范围，即从 y1到 y2（假设 y 1≤y 2）。

绘制像素：对于范围内的每个 y 值，绘制像素点 (x,y)。

4.2 用DDA算法实现斜率为负的直线的生成方法

1.计算步数：根据 x 或 y 的变化范围确定步数 n。如果 ∣x2 −x1∣>∣y2 −y1∣，则 n=∣x2 − x1∣；否则 n=∣y2 − y1∣。

2.计算增量：



3.初始化：设置初始点 (x,y)=(x1,y1)。

4.绘制像素：在每一步中，更新坐标 (x,y)，并绘制像素点 (round(x),round(y))。

4.3用中点Bresenham算法实现斜率为负的直线的生成方法

1.计算初始参数：

dx=x2 − x1

dy=y2 − y1

初始决策参数 p=2⋅dy+dx

2.初始化：设置初始点 (x,y)=(x1,y1)。

3.绘制像素：

如果 p<0，选择东方向的像素点，更新 p=p+2⋅dy。

如果 p≥0，选择东南方向的像素点，更新 p=p+2⋅dy+2⋅dx。

更新 x=x+1，如果 y 增加，则 y=y−1（因为斜率为负）。

4.重复：重复上述步骤，直到 x=x2 。

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？

1. 生成大量随机直线：确保直线的起点和终点随机分布，覆盖不同的斜率和长度。
2. 替换DrawPixel为统计函数：将DrawPixel替换为统计调用次数的函数，或者直接使用空函数。
3. 分别运行不同算法：对每种直线绘制算法，使用相同的随机直线集合进行测试。
4. 记录时间或迭代次数：记录每种算法的运行时间或DrawPixel的调用次数。
5. 比较结果：比较不同算法的运行时间或迭代次数，得出性能差异。

4.5 把坐标系转换为具备四象限模式的坐标系

1.确定屏幕中心点：屏幕中心点通常作为四象限坐标系的原点。假设屏幕的宽度为 W，高度为 H，则屏幕中心点的坐标为 (W/2,H/2)。

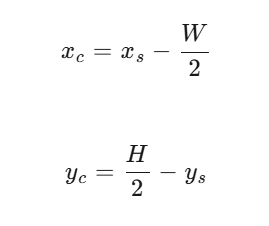
2.坐标转换公式：

从屏幕坐标系到四象限坐标系：

设屏幕坐标为 (xs,ys)。

四象限坐标为 (xc,yc)。

转换公式为：



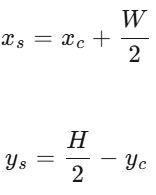
这里 yc的计算公式是因为屏幕坐标系中 y 轴向下为正，而四象限坐标系中 y 轴向上为正。

从四象限坐标系到屏幕坐标系：

设四象限坐标为 (xc,y c)。

屏幕坐标为 (xs,ys)。

转换公式为：

​

1. 实验心得

通过直线生成实验以及对坐标系转换的思考，我深刻体会到了计算机图形学中算法优化和坐标处理的重要性。在直线生成实验中，我们对比了DDA算法和中点Bresenham算法。DDA算法虽然简单直观，但由于涉及浮点运算，效率较低；而中点Bresenham算法通过整数运算，避免了浮点计算的复杂性，显著提高了绘制速度。这让我认识到，优化算法的核心在于减少不必要的计算和数据类型转换，从而提升性能。

在坐标系转换的思考中，我了解到将屏幕坐标系转换为四象限模式的必要性。这种转换不仅使图形处理更符合数学直觉，还简化了许多几何计算。例如，在四象限坐标系中，绘制函数图像和处理几何变换变得更加自然和高效。这让我意识到，选择合适的坐标系可以极大地简化问题，提高编程效率。

实验还让我明白，测试算法性能时需要排除无关因素的干扰。通过将`DrawPixel`函数替换为空函数或统计调用次数，我们能够更准确地比较不同算法的性能，而不是被像素绘制时间所误导。这种科学的测试方法为算法优化提供了可靠的依据。

**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 | | | | |