实验二：直线的生成

姓 名：

学 号：

班 级：

实验地点： 计算机楼

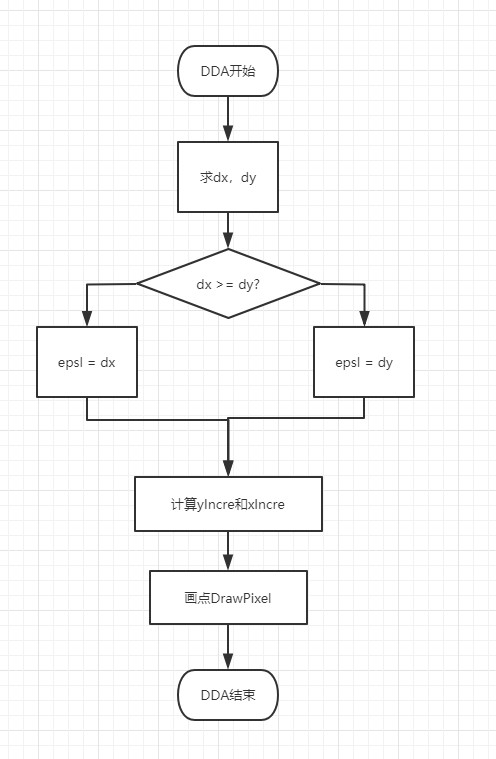
实验时间： 2021.04.20

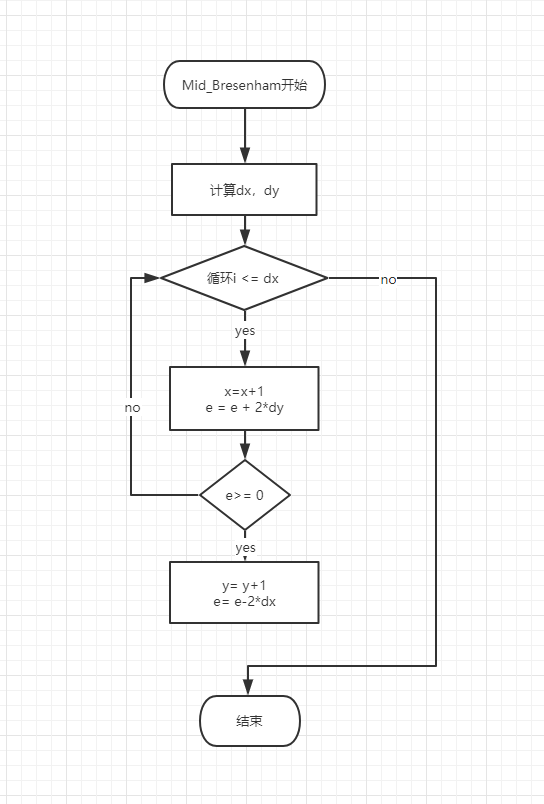
1. **实验目的和要求**

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）
2. **实验结果**
   1. **程序流程图**

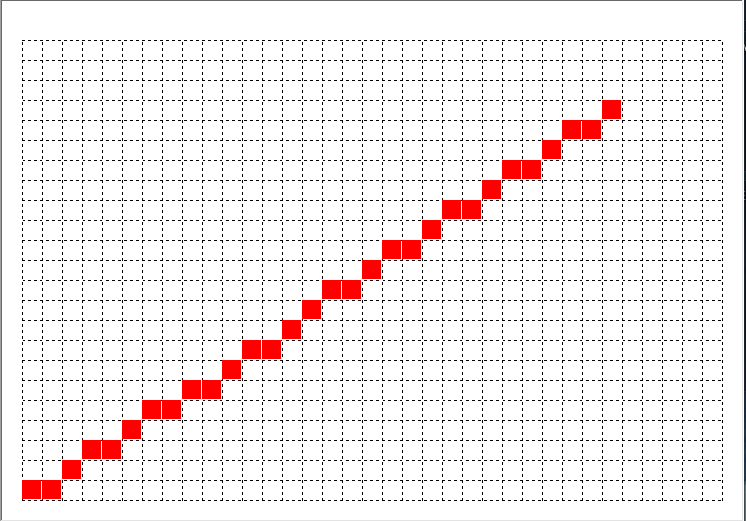




* 1. **程序代码**

void CExperiment\_Frame\_OneView::DDA(int X0, int Y0, int X1, int Y1)  
{  
//----------请实现DDA算法------------//  
int dx, dy, epsl, x, y, k;  
float xIncre, yIncre;  
dx = X1 - X0;  
dy = Y1 - Y0;  
x = X0;  
y = Y0;  
if(abs(dx) > abs(dy)){  
epsl = abs(dx);  
}  
else {  
epsl = abs(dy);  
}  
xIncre = (float)dx/(float)epsl;  
yIncre = (float)dy/(float)epsl;  
for(k = 0; k<=epsl; k++){  
DrawPixel(int(x+0.5), (int)(y+0.5));  
x += (int)xIncre;  
y += (int)yIncre;  
}  
  
}  
  
void CExperiment\_Frame\_OneView::Mid\_Bresenham(int X0, int Y0, int X1, int Y1)  
{  
//-------请实现Mid\_Bresenham算法-------//  
int x,y,dx,dy,e;  
dx=X1-X0;  
dy=Y1-Y0;  
e=-dx;扫描线  
x=X0;  
y=Y0;  
    for(int i=0;i<=dx;i++){  
 x=x+1;  
 e=e+2\*dy;  
 if(e>=0){  
    y=y+1;  
e=e-2\*dx;  
 }  
}  
}

* 1. **运行结果**



* 1. **运行结果分析**

实验与预期相同，符合实验结果

1. **实验心得**

在经过该实验后，已经较为熟练的掌握了DDA算法和中点Bresenham算法，并且成功跑通程序。

实验三：多边形扫描转换算法

姓 名：

学 号：

班 级：

实验地点： 计算机楼

实验时间： 2021.04.20

**1 实验目的和要求**

理解多边形扫描转换的原理；掌握典型多边形扫描转换算法；掌握步处理、分析实验数据的能力；

* 编程实现基本X-扫描线转换算法（必做）；
* 编程实现有效边表转换算法（选做）

**2 实验环境和工具**

本试验提供自带实验平台

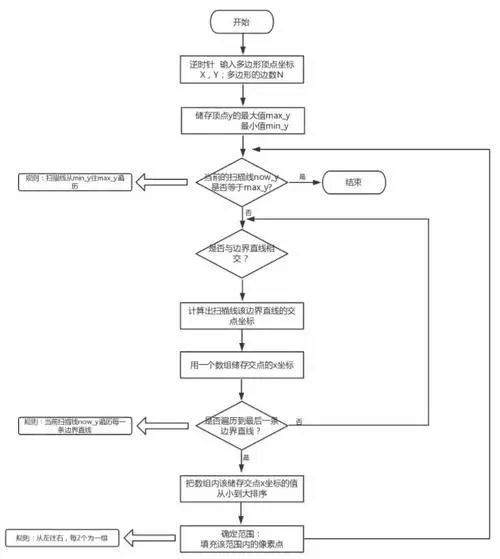
* 开发环境：Visual C++ 6.0
* 实验平台：Free\_Curve（自制平台）

本实验提供名为 Polygon\_Conversion的平台，该平台提供基本绘制、设置、输入功能，学生在此基础上实现X-扫描线算法和有效边表转换算法。

* 平台界面：如图 3‑1所示
* 多边形输入，界面如图 3‑2所示：
  + 用户按【功能】🡪【输入多边形……】菜单开始输入多边形；
  + 单击鼠标左键输入多边形顶点；
  + 点击鼠标右键结束多边形输入，并将最后一个顶点和第一个顶点进行连接；
* 参数设置：界面如图 3‑3所示
  + 用户按“【功能】🡪【设置……】”启动设置对话框
  + 设置内容：
    - 填充色
    - 是否填充多边形
    - 选择转换算法

**3 实验结果**

**3.1 程序流程图**



**3.2 程序代码**

void CPolygon\_ConversionView::X\_Scan\_Line\_Conersion(int Vertices[][2], int VertexNum)

{

int ymax, ymin, count;

float k;

ymax = Vertices[0][1];

ymin = Vertices[0][1];

for(int p = 0; p < VertexNum; p ++)

{

//求y的最高点ymax和最低点ymin

if(Vertices[p][1] > ymax)

{

ymax = Vertices[p][1];

}

if(Vertices[p][1] < ymin)

{

ymin = Vertices[p][1];

}

}

int Xn[100], i, ymax2, ymin2, temp;

for(i = ymin; i <= ymax;i ++)

{

// 求交点

count = 0;

for (int j = 0;j < VertexNum; j++)

{

if (j == (VertexNum - 1))

{

// 最后一个点和第一个点的线段

if(Vertices[j][1] >= Vertices[0][1])

{

// 线段y的最大值和最小值

ymax2 = Vertices[j][1];

ymin2 = Vertices[0][1];

}

else

{

ymax2 = Vertices[0][1];

ymin2 = Vertices[j][1];

}

if ( (i < ymax2) && (i > ymin2) )

{

// 判断交点是否在线上

k = (float)( (float)(Vertices[j][0] - Vertices[0][0]) / (float)(Vertices[j][1] - Vertices[0][1]) );

Xn[count] = (int)(k \* (float)i - k \* (float)Vertices[0][1] + (float)Vertices[0][0]);

count++;

}

else if(i == Vertices[j][1] )

{

// 扫描线经过线段顶点，只算前一个顶点

if( (Vertices[j-1][1] > i) && (Vertices[0][1] > i) )

{

Xn[count] = Vertices[j][0];

count++;

Xn[count] = Vertices[j][0];

count++;

}

else if ( (Vertices[j-1][1] < i) && (Vertices[0][1] > i) || (Vertices[j-1][1] > i) && (Vertices[0][1] < i) )

{

Xn[count] = Vertices[j][0];

count++;

}

}

}

else

{

// 其他线段的交点

if ( Vertices[j][1] > Vertices[j+1][1] )

{

ymax2 = Vertices[j][1];

ymin2 = Vertices[j+1][1];

}

else

{

ymax2 = Vertices[j+1][1];

ymin2 = Vertices[j][1];

}

if ( (i < ymax2) && (i > ymin2) )

{

// 判断交点是否在线上

k = (float)( (float)(Vertices[j][0] - Vertices[j+1][0]) / (float)(Vertices[j][1] - Vertices[j+1][1]) );

Xn[count] = (int)(k \* (float)i - k \* (float)Vertices[j+1][1] + (float)Vertices[j+1][0]);

count++;

}

else if(i == Vertices[j][1] )

{

// 扫描线经过线段顶点，只算前一个顶点

if(j != 0)

{

if( (Vertices[j-1][1] > i) && (Vertices[j+1][1] > i) )

{

Xn[count] = Vertices[j][0];

count++;

Xn[count] = Vertices[j][0];

count++;

}

else if ( (Vertices[j-1][1] < i) && (Vertices[j+1][1] > i) || (Vertices[j-1][1] > i) && (Vertices[j+1][1] < i) )

{

Xn[count] = Vertices[j][0];

count++;

}

else{}

}

else

{

//交点为第一个点

if ( (Vertices[VertexNum-1][1] > i) && (Vertices[j+1][1] > i) )

{

Xn[count] = Vertices[j][0];

count++;

Xn[count] = Vertices[j][0];

count++;

}

else if ((Vertices[VertexNum-1][1] < i) && (Vertices[j+1][1] > i) || (Vertices[VertexNum-1][1] > i) && (Vertices[j+1][1] < i) )

{

Xn[count] = Vertices[j][0];

count++;

}

else{}

}

}

}

}

}

for (int m = 0; m < count - 1; m++)

{

for(int n = m+1; n < count; n++)

{

if(Xn[m] > Xn[n])

{

temp = Xn[m];

Xn[m] = Xn[n];

Xn[n] = temp;

}

}

}

for(int a = 0;a < count; a += 2)

{

for(int b = Xn[a]; b < Xn[a+1]; b++)

{

DrawPixel(b,i);

}

}

return;

}

**3.3 运行结果**



**3.4 运行结果分析**

基本符合实际情况，程序运行良好。

4 **实验心得**

在经过该实验后，已经较为熟练的掌握了X扫描线算法，并且成功跑通程序。