**直线段的扫描转换**

1. **请用任意一种编程语言实现三种直线段扫描转换算法；**

①数值微分(DDA)法：

void DDALine(int x0,int y0,int x1,int y1,int color)

{

int x；

float dx, dy, y, k;

dx= x1-x0, dy=y1-y0;

k=dy/dx, y=y0;

for (x=x0; x<=x1, x++)

{

drawpixel (x, int(y+0.5), color);

y=y+k;

}

}

②中点画线法

void DDALine(int x0,int y0,int x1,int y1,int color)

{

int x；

float dx, dy, y, k;

dx= x1-x0, dy=y1-y0;

k=dy/dx, y=y0;

for (x=x0; x<=x1, x++)

{

drawpixel (x, int(y+0.5), color);

y=y+k;

}

}

③Bresenham算法

void BresenhamLine (int x0, int y0, int x1, int y1, int color) {

int x, y, dx, dy,i;

float k, e;

dx = x1 - x0, dy = y1 - y0, k = dy / dx;

e = -0.5, x = x0, y = y0;

for (i = 0; i <= dx; i++) {

drawpixel (x, y, color);

x = x + 1，e = e + k;

if (e >= 0) {

y++, e = e - 1;

}

}

}

1. **比较三种算法的运行效率，并给出分析结论。**

效率比较：

①数值微分(DDA)法：

对于大多数直线，DDA算法的运行效率适中，但由于其使用浮点运算，当处理大量数据时，速度可能不如Bresenham算法。

②中点画线法：

由于可能涉及大量的浮点运算，并且需要处理每个x值，因此其运行效率通常最低。

③Bresenham算法：

由于其只使用整数运算，通常情况下，Bresenham算法的运行效率是最高的，特别是在绘制较长的直线时。

分析结论：

速度：Bresenham算法通常是最快的，因为它避免了昂贵的浮点运算，并且算法设计上考虑了计算机的整数运算特点。

准确性：对于大多数直线，Bresenham算法和DDA算法都能提供足够的准确性。但对于非常斜的直线，DDA算法可能会出现较大的误差。

通用性：直线方程算法在理论上适用于所有直线，但在实际应用中，其效率通常不如其他两种算法。

实现难度：DDA算法最简单，Bresenham算法相对复杂，但考虑到效率问题，Bresenham算法通常是首选。