1. 画线算法

场景中的直线段由其两端点的坐标位置来定义。要在光栅显示器上显示一条线段，图形系统必须将两端点投影到整数屏幕坐标，并确定离两端点的直线路径最近的像素位置。接下来将颜色值装入帧缓存相应的像素坐标处。视频控制器从帧缓存读出写入的颜色并绘制屏幕像素。这一过程将一条线段数字化为一组离散的整数位置。一般而言，这些位置是实际线段路径的近似。例如，计算出的线段上的位置（10.48,20.51 ）转换为像素位置（10,21）。坐标值舍人到整数，引起除水平和垂直以外所有线段的阶梯效应（锯齿形），如图1



光栅线段特有的阶梯现象在低分辨率系统中特别容易看出来，而在高分辨率系统中可以得到改善。更有效的平滑光栅线段的技术是基于调整线段路径上的像素强度（细节参照15节）。

直线方程

根据直线的几何特征可确定直线路径的像素位置。直线的笛卡儿斜率截距方程为



其中，m为直线的斜率，b为y轴截距。给定图所示线段的两个端点(, )和(,), 可以计算斜率m和y轴截距b：



显示直线段的算法以直线方程1及式2，3中计算方法为基础。

对于任何沿直线给定的x增量δx，可以从式2中计算出对应的 y 增量δy。



同样，可以得出对应于指定的x增增量δx



这些方程形成了模拟设备（如向量扫描系统）中确定偏转电压的基础，其中有可能造成微小的偏转电压变化。对于具有斜率绝对值|m|<1的直线，可以设置一个较小的水平偏转电压δx，对应的垂直偏转电压则可以使用式4计算出来的δy来设定；而对于斜率值|m|>1的直线，则设置一个比较小的垂直偏转电压δy，对应的水平偏转电压则由式5中计算出来的δx 来设定；对于斜率 m=1的直线，δx=δy，因此水平偏转和垂直偏转电压相等。在每一种情况下，都可以在指定的端点之间生成一条斜率为 m 的平滑直线段。

在光栅系统中，通过像素绘制线段，水平和垂直方向的步长受到像素的间距的限制。也就是必须在离散位置上对线段进行取样，并且在每个取样位置上确定距离线段最近的像素。图3给出了线段的扫描转换过程，及相对x 轴的离散取样点位置。

数字微分分析仪(digital differential analyzer，DDA)方法是一种线段扫描转换算法，基于便用式4或式5来计算δx或δy，在一个坐标轴上以单位间隔对线段取样，从而确定另一个坐标轴上最靠近线路径的对应整数值。

首先考虑如图2所示的具有正斜率的线段。例如，如果斜率小于等于1，则以单位x间隔 (δx=1)取样，并逐个计算每一个y值，



下标k取整数值，从第一个点1开始递增直至最后的端点。由于m可以是0与1之间的任意实数，所以计算出的y值必须取整。

对于具有大于1的正斜率的线段，则交换x和y的位置。也就是以单位y间隔(δy =1)取样顺序计算每个x值



此时，每一个计算出的x值要沿y扫描线舍人到最近的像素位置。

式6和式7基于从左端点到右端点处理线段的假设（参见图2 ）。假如这个过程中的处理方向相反，即起始端点在右侧，那么δx= -1,并且



或者（当斜率大于1时）是δy = -1，并且



式6到式9也可以用来计算具有负斜率的线段的像素位置。假如斜率的绝对值小于1，并且起始端点在左侧，可设置δx=1并用式6计算y值。当起始端点在右侧（具有相同的斜率）时，我们可设置δx= -1并且由式8得到y的位置。同样，负斜率的绝对值大于1时可以使用δy=-1和式9或者δy=1和式7进行计算。

这个算法可以概括为下面的过程：输人线段两个端点的像素位置。端点位置间水平和垂直的差值赋给参数dx和dy。绝对值大的参数确定参数steps的值。该值也是在即将画出的这条线段上的像素数目; 按照这个数值，沿线段路径计算每一步的下一个像素位置。先绘制起始位置 (, ) 的像素，然后调整每一步的x和y，获得并逐一绘制余下的像素。假如dx的绝对值大于dy的绝对值，且x0小于xEnd，那么x和y方向的增量值分别为1和m。假如x方向的变化较大，但x0大于xEnd，那么就采用减量-1和-m来生成线段上的每个点。在其他情况下，y方向使用单位增量（或减量），x方向使用1/m的增量(或减量)。

#include <stdlib.h>

#include <math.h>

inline int round (const float a)

{

return int (a + 0.5);

}

void lineDDA (int x0, int y0, int xEnd, int yEnd)

{

int dx=xEnd-x0, dy=yEnd-y0, steps, k;

float xIncrement, yIncrement, x = x0, y = y0;

if (fabs (dx) > fabs (dy))

steps = fabs (dx);

else

steps = fabs (dy);

xIncrement = float (dx) / float (steps);

yIncrement = float (dy) / float (steps);

setPixel (round (x), round (y));

for (k = 0; k < steps; k++)

{

x += xIncrement;

y += yIncrement;

setPixel (round (x), round (y));

}

}

DDA方法计算像素位置要比直接使用直线方程1计算的速度更快。它利用光撖特性消 除了直线方程1中的乘法，而在x或y方向使用合适的增量，从而沿线路径逐步得到各像素的位置。但在浮点增量的连续叠加中，取整误差的积累使得对于较长线段所计算的像素位置偏离实际线段。而且该过程中的取整操作和浮点运算仍然十分耗时。我们可以通过将增量m和1/m分离成整数和小数部分，从而使所有的计算都简化为整数操作来改善DDA算法的性能。在10节将讨论用整数步长计算1/m增量的方法。在下一节中,我们考虑既能用于直线又能用于曲线的 更通用的扫描线程序。