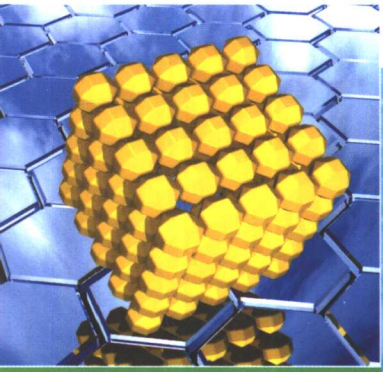
计算机图形学的基础算法



刘勇奎 著

科学出版社

计算机图形学的基础算法

刘勇奎 著

科学出版社

2002

内 容 简 介

本书是作者总结10多年来对计算机图形学基础算法研究成果的一部专著。书中内容的90%为作者已发表或尚未发表的研究成果。主要内容包括:图形的生成、裁剪、六角网格上的图形算法及与图形相关的图象处理与识别算法等。书中内容主要侧重于较新的象素级算法。

本书的读者对象包括计算机图形学的专业研究人员及大专院校师生。

图书在版编目(CIP)数据

计算机图形学的基础算法/刘勇奎著. -北京:科学出版社,2001

ISBN 7-03-007979-5

Ⅰ.计… Ⅱ.刘… Ⅲ.计算机图形学-算法 Ⅳ.TP391.41

中国版本图书馆CIP数据核字(2000)第61457号

科学出版社 出版

北京东黄城根北街16号

邮政编码:100717

http://www.sciencep.com

双青印刷厂 印刷

科学出版社发行 各地新华书店经销

*

2001年8月第 一 版 开本:850×1168 1/32

2002年1月第二次印刷 印张:7

印数:2 001—5 000 字数:188 000

定价:15.00元

(如有印装质量问题,我社负责调换<环伟>)

前　言

本书是作者多年来关于计算机图形学及相关的图象处理与模式识别研究成果的系统归纳和总结。书中包括了图形的生成、裁剪及图象显示与识别方面的多种算法。其中除了 Bresenham 算法等少数几个算法之外，均为作者提出的算法。这些算法有的已在国内外发表，有的还尚未发表。除此之外，还简要介绍了这些领域目前国际上最新的成果。

本书首先介绍了直线和曲线生成的几个算法。其中主要是象素级整数型生成算法。例如对于直线生成，目前国际上的研究热点是一步生成多点的算法。本书也讨论了这样的算法。另一个重点内容是介绍了一个可生成任意曲线的象素级整数算法。目前出现的象素级生成算法只有直线、圆及其它圆锥曲线；而该算法可生成任何常用曲线。本书的另一个内容是图形的裁剪算法，包括在其它文献中较难见到的凹多边形窗口的裁剪算法及曲线窗口和曲线裁剪算法等。本书还涉及到了与图形相关的图象处理及模式识别算法。最后介绍了六角网格的特点及在其上使用的图形及图象处理算法。

本书的研究得到了国家自然科学基金及辽宁省科技基金和省教委科技基金的资助。在本书出版之际，作者要感谢浙江大学石教英教授，他为作者提供了许多学术上的帮助。还要感谢我的研究生颜叶、周晓敏，他们对书中部分算法进行了实现和验证。

作　者

2001 年于沈阳

作者介绍

刘勇奎，男，1961 年 5 月生。1982 年毕业于吉林大学计算机系，获学士学位；1987 年毕业于沈阳工业大学计算机应用专业研究生，获硕士学位，之后留校任教。1989 年任讲师，1994 年晋升为副教授。1995 年入浙江大学 CAD 与计算机图形学国家重点实验室攻读博士学位，于 1999 年毕业获博士学位。同年晋升为沈阳工业大学信息科学与工程学院教授。之后去英国 De- Montfort 大学作为访问学者工作半年。现任大连民族学院计算机系教授（院特聘教授）、计算机科学研究所所长。

多年来一直从事计算机图形学及图象处理方面的研究工作。曾主持完成国家自然科学基金及省、部级科技基金项目 7 项，还参加过国家重大攻关项目及横向课题多项。这些完成的项目中，有些还获得省市级成果奖。在此基础上，作为第一作者在《CVGIP: Graphical Models and Image Processing》等国际重要学术期刊上发表论文 6 篇，在《计算机学报》等国内期刊上发表论文 40 多篇。其中的 5 篇论文曾 8 次被著名的国际四大检索系统中的《EI》（工程索引）和《SCI》（科学引文索引）收录。另外还作为主编、副主编或参编出版教材书四部。名字被收录在国际著名期刊《IEEE Computer Graphics and Applications》的审稿人数据库中，还多次为国内《计算机学报》、《计算机辅助设计与图形学学报》及《中国图象图形学报》等三种期刊及 CAD Graphics 2001 等国际会议审稿。被评为辽宁省高校骨干教师并入选辽宁省百千万人才培养计划。

目　录

第一章　直线与曲线的生成 …… (1)
1.1　圆及椭圆的多边形逼近及线式生成 …… (1)
1.2　直线的象素级生成算法 …… (7)
1.2.1　Bresenham 直线生成算法 …… (8)
1.2.2　单点直线生成算法已无优化的余地 …… (11)
1.2.3　一个双点 Bresenham 直线生成算法 …… (12)
1.2.4　直线的对称生成 …… (16)
1.2.5　多点直线生成算法是当前的研究方向 …… (19)
1.2.6　多点直线生成算法所存在的问题 …… (22)
1.2.7　三维直线算法——体素的直线遍历 …… (23)
1.2.8　多灰度级直线 …… (31)
1.3　圆的象素级生成算法 …… (34)
1.3.1　圆的象素级生成算法概述 …… (34)
1.3.2　最快的象素级圆生成单点算法 …… (36)
1.3.3　圆的双步(双点)生成算法 …… (39)
1.3.4　圆生成算法的比较 …… (41)
1.4　抛物线的象素级生成算法 …… (44)
1.5　一个通用的象素级曲线生成算法 …… (50)
1.6　等值线的抽取与绘制 …… (53)

第二章　图形裁剪 …… (59)
2.1　矩形窗口的裁剪算法 …… (59)
2.1.1　矩形窗口的直线裁剪 …… (59)
2.1.2　矩形窗口的圆及椭圆裁剪 …… (62)
2.1.3　参数曲线裁剪 …… (71)

2.2 一般多边形窗口的直线裁剪算法 …… (74)
2.2.1 算法概述 …… (75)
2.2.2 交点计算 …… (76)
2.2.3 直线通过多边形的一个顶点或与其一边重合情况的处理 …… (78)
2.2.4 算法实现 …… (80)
2.3 圆形和椭圆形窗口裁剪算法 …… (82)
2.3.1 圆形窗口的线裁剪 …… (82)
2.3.2 椭圆形裁剪窗口 …… (85)
2.4 多边形窗口的多边形裁剪算法 …… (85)
2.4.1 基本概念与定义 …… (86)
2.4.2 新算法的数据结构 …… (88)
2.4.3 新算法 …… (90)
2.4.4 交点的判断与计算 …… (96)
2.4.5 两多边形的边重合或者两多边形在顶点处相交的特殊情况的处理 …… (100)
2.4.6 算法比较 …… (101)
2.4.7 小结 …… (104)

第三章 参数曲线的逐点生成 …… (105)
3.1 将参数表达式转换成非参数表达式后生成 …… (105)
3.1.1 二次 Bezier 曲线的生成 …… (105)
3.1.2 三次 Bezier 曲线的生成 …… (110)
3.1.3 算法的进一步优化 …… (113)
3.1.4 二次和三次 B 样条曲线的生成 …… (116)
3.2 参数曲线的直接生成 …… (119)
3.2.1 现有算法介绍 …… (119)
3.2.2 最佳的 n 值 …… (122)
3.2.3 双步逐点曲线生成算法 …… (124)

第四章　有关图象显示与识别的几个问题…… (135)
4.1　图象与图形的树表示及搜索 …… (135)
4.2　多面体的隐藏线消除 …… (138)
4.2.1　解决问题的方法 …… (141)
4.2.2　求凸多面体的一个可见面 …… (142)
4.2.3　消隐线算法 …… (144)
4.3　反走样技术 …… (146)
4.3.1　反走样直线算法 …… (146)
4.3.2　反走样圆算法 …… (151)
4.4　多灰度级图象的二值显示问题 …… (154)
4.4.1　误差分散方法及分析 …… (155)
4.4.2　误差分散方法的改进 …… (157)
4.5　噪声的模拟产生方法 …… (158)
4.6　借助曲线生成方法进行曲线识别 …… (159)
4.6.1　直线的识别 …… (159)
4.6.2　圆及椭圆链码的识别 …… (162)
4.7　边界曲线的特征点抽取 …… (167)
4.7.1　综合方法的基本原理 …… (167)
4.7.2　算法实现 …… (171)
4.8　一种有效的压缩链码 …… (173)
4.8.1　压缩链码 …… (175)
4.8.2　压缩链码与 Freeman 链码之间的转换 …… (177)
4.8.3　压缩链码与其它链码的比较 …… (179)

第五章　六角网格及其图形算法…… (182)
5.1　六角网格及其特点 …… (182)
5.2　六角网格上的直线生成算法 …… (184)
5.3　六角网格上的椭圆生成算法 …… (192)
5.4　六角网格上的圆弧生成算法 …… (196)
5.5　六角网格上的裁剪算法 …… (198)

5.6 六角网格上的图象处理 …… (202)
5.6.1 六角网格上的数字化 …… (202)
5.6.2 几何失真校正算法 …… (204)
5.6.3 轮廓跟踪算法 …… (208)

参考文献…… (210)

第一章　直线与曲线的生成

直线与曲线是组成图形的基本元素。其生成(或称绘制)算法的优劣对整个图形系统的效率是至关重要的,因为在产生一幅图形的过程中要反复多次调用这些算法。曲线的生成算法可分为线生成和象素级(或称点)生成两类。所谓线生成就是以小的直线段来逼近和代替实际曲线而显示之;而象素级生成则是逐个象素地选择距离实际曲线最近的点而显示之。前者是随着随机扫描显示器的产生而出现的;后者则是为目前普遍使用的光栅扫描显示器而设计的,具有精度高、计算量小等特点。

1.1　圆及椭圆的多边形逼近及线式生成[1,2]

当圆的内接多边形边数足够多时,该多边形可以和圆接近到任意程度。因此在允许的误差范围内(可用圆周与多边形之间的最大距离来度量),可以用显示多边形代替显示圆。

设要绘制的圆的圆心为 $P_c(x_c, y_c)$,半径为 R。又设内接正多边形的一个顶点为 $P_i(x_i, y_i)$,内接正多边形两个顶点至圆心连线的夹角为 α,如图 1-1 所示,则

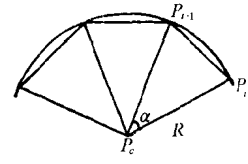


图 1-1　用内接正多边形逼近圆弧

$$x_i = x_c + R\cos i\alpha$$

$$y_i = y_c + R\sin i\alpha$$

可用递推关系得出下一个顶点 P_{i+1} 的坐标为

$$x_{i+1} = x_c + R\cos(i\alpha + \alpha) = x_c + (x_i - x_c)\cos\alpha - (y_i - y_c)\sin\alpha$$

$$y_{i+1} = y_c + R\sin(i\alpha + \alpha) = y_c + (x_i - x_c)\sin\alpha + (y_i - y_c)\cos\alpha$$

可用矩阵形式表示为

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} x_i - x_c \\ y_i - y_c \end{bmatrix} + \begin{bmatrix} x_c \\ y_c \end{bmatrix}$$

上式又可进一步简化成

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = 2\cos\alpha \begin{bmatrix} x_i \\ y_i \end{bmatrix} - \begin{bmatrix} x_{i-1} \\ y_{i-1} \end{bmatrix}$$

该式就是计算圆的内接多边形的各顶点的递推公式。因为 α 是常数，$\cos\alpha$ 只需在开始时算一次。这样，算一个顶点只需作两次乘法。

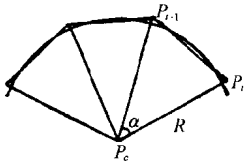


图 1-2　用与圆相交正多边形逼近圆弧

以上是众所周知的结果。这里要指出的是：如果用圆的相交正多边形代替内接正多边形逼近圆，如图 1-2 所示，会产生更好的效果。

比较图 1-1 和图 1-2 可以看出：在多边形的顶点数相同的前提下，用相交多边形逼近圆比用内接多边形逼近圆的误差更小，或者说逼近程度更高。而在算法具体实现时，只需将圆的半径 R 增加一个小的增量 x，其它没有任何变化。这相当于对半径为 $R + x$ 的圆执行内接多边形算法。下面我们将看到，恰当地选择这个增量可使多边形最佳逼近圆弧。

设图 1-2 的圆相交多边形中各直线段端点（即多边形的顶点）距圆弧的法向距离为 x。下面计算 x 为多大时才能使多边形最大限度地逼近圆弧，即最佳逼近。一般来说，最佳逼近有两种衡量方法。一种是使多边形与圆弧所夹的面积最小；另一种是使多边形与圆弧之间的法向最远点的距离最近。下面先计算在面积逼近意义下最佳的 x 值。

由于一个 α 角扇区是对称的，所以只需计算 $\frac{\alpha}{2}$ 角扇区部分的

多边形与圆弧所围成的面积。为了能得出 x 的直观的、简洁的表达式，将这一段圆弧用直线代替，如图 1-3 所示。图中还为各线段设了标记，以便计算。

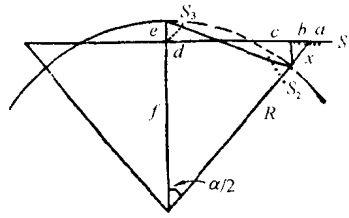


图 1-3　最佳 x 值的计算

从图 1-3 可知

$$f=(R+x)\cos\frac{\alpha}{2}$$

$$e=R-f=R-(R+x)\cos\frac{\alpha}{2}$$

$$d=e\cdot tg\frac{\pi-\frac{\alpha}{2}}{2}=\left[R-(R+x)\cos\frac{\alpha}{2}\right]\operatorname{ctg}\frac{\alpha}{4}$$

$$a=x\cdot\sin\frac{\alpha}{2}$$

$$b=x\cdot\cos\frac{\alpha}{2}$$

$$c=(R+x)\sin\frac{\alpha}{2}-a-d$$

$$=R\sin\frac{\alpha}{2}-\left[R-(x+R)\cos\frac{\alpha}{2}\right]\operatorname{ctg}\frac{\alpha}{4}$$

多边形与圆弧所夹面积为

$$S=S_1+S_2+S_3=\frac{1}{2}ab+\frac{1}{2}bc+\frac{1}{2}de=\frac{x^2}{4}\sin\alpha$$

$$+x^2\cos^2\frac{\alpha}{2}\operatorname{ctg}\frac{\alpha}{4}+\frac{R}{4}x\sin\alpha-\frac{3R}{2}x\cos\frac{\alpha}{2}\operatorname{ctg}\frac{\alpha}{4}$$

$$+\frac{3R}{2}x\cos^2\frac{\alpha}{2}\operatorname{ctg}\frac{\alpha}{4}+\frac{1}{2}R^2\operatorname{ctg}\frac{\alpha}{4}-R^2\cos\frac{\alpha}{2}$$

$$\times \operatorname{ctg}\frac{\alpha}{4}+\frac{R^2}{2}\cos^2\frac{\alpha}{2}\operatorname{ctg}\frac{\alpha}{4}$$

将上式对 x 求导

$$\frac{dS}{dx}=\frac{1}{2}x\sin\alpha+2x\cos^2\frac{\alpha}{2}\operatorname{ctg}\frac{\alpha}{4}+\frac{R}{4}\sin\alpha$$

$$-\frac{3R}{2}\cos\frac{\alpha}{2}\operatorname{ctg}\frac{\alpha}{4}+\frac{3R}{2}\cos^2\frac{\alpha}{2}\operatorname{ctg}\frac{\alpha}{4}$$

令上式为零,求出 x 为

$$x=-\frac{\frac{R}{4}\sin\alpha-\frac{3R}{2}\cos\frac{\alpha}{2}\operatorname{ctg}\frac{\alpha}{4}+\frac{3R}{2}\cos^2\frac{\alpha}{2}\operatorname{ctg}\frac{\alpha}{4}}{\frac{1}{2}\sin\alpha+2\cos^2\frac{\alpha}{2}\operatorname{ctg}\frac{\alpha}{4}}$$

$$=\frac{1-\cos\frac{\alpha}{2}}{1+\cos\frac{\alpha}{2}}R$$

x 的这个值使多边形与圆弧所夹面积最小。

下面计算使算法在点逼近意义下最佳的 x 值。

由于圆相交多边形与圆弧之间的法向最远点距离为 x 和 e 的最大者,而从图 1-3 可见,x 与 e 互补,即如果 x 增大,则 e 必然减小,反之亦然。这样只有当 x 等于 e 时,$\max(x,e)$ 才会最小。因此,令 $x=e=R-(R+x)\cos\frac{\alpha}{2}$,求出的 x 值就是使算法在点逼近意义下最佳的 x 值,即

$$x=\frac{1-\cos\frac{\alpha}{2}}{1+\cos\frac{\alpha}{2}}R$$

幸运的是,该值刚好与面积最佳逼近意义下求出的 x 值完全相同,这更证实了它是使多边形最佳逼近圆弧的唯一的取值。

综上所述,在用线生成算法产生半径为 R 的圆弧时,将 R 增大到

$$R+x=R+\frac{1-\cos\frac{\alpha}{2}}{1+\cos\frac{\alpha}{2}}R=\frac{2}{1+\cos\frac{\alpha}{2}}R$$

然后用传统的内接多边形算法生成的圆弧将比不增大 R 的内接多边形算法更逼近参照圆弧，而且是最佳逼近。由此得出本算法的执行过程如图 1-4 所示。

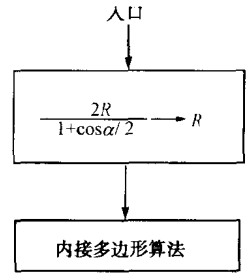


图 1-4　相交多边形逼近算法的执行过程

下面我们来分析算法的逼近误差，在内接多边形算法中，参照圆弧与生成的多边形之间的最大误差（即最近点距离）是

$$e'=\left(1-\cos\frac{\alpha}{2}\right)R$$

而由上面的计算可知，新算法的最大误差是

$$e=\frac{1-\cos\frac{\alpha}{2}}{1+\cos\frac{\alpha}{2}}R$$

二者之比为

$$\frac{e}{e'}=\frac{1}{1+\cos\frac{\alpha}{2}}$$

由于 α 值在实用中是很小的，因此，这个比值约为 $\frac{1}{2}$。这表明相交多边形逼近算法的最大误差比内接多边形算法几乎缩小了一半。

相交多边形方法同样适用于对椭圆弧的逼近。设椭圆弧的中心为 $P_c(x_c,y_c)$，长半轴为 A，短半轴为 B，长轴与 x 轴的夹角为 φ，则作变换

$$\overline{x}-x_c=(x-x_c)\cos\varphi+(y+y_c)\sin\varphi$$

$$\bar{y} - y_c = -(x - x_c)\sin\varphi + (y - y_c)\cos\varphi$$

椭圆方程便表示为

$$\frac{(\bar{x} - x_c)^2}{A^2} + \frac{(\bar{y} - y_c)^2}{B^2} = 1$$

再作变换

$$x' - x_c = \sqrt{\frac{B}{A}}(\bar{x} - x_c)$$

$$y' - y_c = \sqrt{\frac{A}{B}}(\bar{y} - y_c)$$

则椭圆弧变成圆弧

$$(x' - x_c)^2 + (y' - y_c)^2 = AB$$

由于这两个变换式的系数行列式的值均为 1，因此能保持区域的面积不变。这样在面积最佳逼近的意义下，对变成的这个圆弧的最佳逼近就是对原椭圆弧的最佳逼近。

现在，圆弧的半径是$\sqrt{AB}$，根据前述的圆弧逼近方法，要对圆弧进行最佳逼近，应将这个半径增大到

$$\frac{2\sqrt{AB}}{1 + \cos\frac{\alpha}{2}}$$

上式可变换成

$$\sqrt{\frac{4AB}{\left(1 + \cos\frac{\alpha}{2}\right)^2}} = \sqrt{\frac{2A}{1 + \cos\frac{\alpha}{2}} \cdot \frac{2B}{1 + \cos\frac{\alpha}{2}}}$$

可见，对椭圆弧进行最佳逼近，应先将其长、短半径 A 和 B 分别增大到$\frac{2A}{1 + \cos\frac{\alpha}{2}}$和$\frac{2B}{1 + \cos\frac{\alpha}{2}}$，然后用传统的内接多边形算法可生成最佳逼近的椭圆弧。此过程与圆弧逼近非常统一。

我们知道，对曲线进行逼近时，逼近误差与算法的执行速度是

一对矛盾的因素。也就是说，要使逼近误差小一些，就要增加取点的频度，即多取点。这无疑会因计算较多的点而增加计算量，从而降低算法的执行速度。而上述的相交多边形逼近算法在没有增加取点频度的前提下却减少了近一半的逼近误差。那么，换言之，用该方法生成的圆和椭圆弧与用传统的内接多边形算法生成的圆弧或椭圆弧逼近误差相同时，该算法会因计算的点少而比内接多边形算法的执行速度快。

从理论上来讲，用这种与曲线相交的直线段来逼近曲线的思想同样也可应用于其它曲线的线生成及逼近。这里关键是如何确定增量 x，以使直线段与曲线距离最近或所夹面积最小。

对于一般曲线的线生成，已经有一些较有效的算法，如我国学者提出的正负法[3]和 T-N 方法[4]等。最简单的一种生成曲线的方法就是对某一个变量（x，y 坐标或参数 t）均匀地取一些点而计算其它变量的值以求出曲线上的点，然后将这些点连接就是曲线的线生成。一种可供选择的取点方法是：在曲线弯曲程度大的地方取点的密度也大；而在弯曲程度小的部分则可使取点间隔大一些。具体实现时是使取点的间隔与曲线的曲率成反比来计算。

1.2 直线的象素级生成算法

我们目前普遍使用的显示器是光栅扫描显示器。其显示屏是由许多被称为象素的点组成的。这些象素是以水平方向和垂直方向成直线排列的。或者说显示屏是由一些水平线和垂直线构成的网格形成的，每个网格点（垂直线与水平线的交点）就是一个象素。在显示器上所显示的曲线图形或图象就是通过这些象素的“亮”与“不亮”（或不同的颜色）的各种组合而形成的。基于光栅显示的这一特点，我们要在其上绘制曲线（或图形）就要与象素打交道。例如我们要生成一条曲线，最好是逐点地选择那些距离曲线最近的象素并将其“点亮”，这就是象素级（或称点）的生成方法。这种方法的优点之一是生成的曲线误差小，精度高。所显示的象素与实

际曲线之间的距离一般不大于二分之一个象素单位。另一个优点就是下面我们将看到的，其算法一般只使用整数运算（至少在主循环内只使用整数运算），因而执行速度很快。

1.2.1 Bresenham 直线生成算法

说到直线的生成算法，最著名的就是 Bresenham 算法[5]。设要生成的直线的两个端点坐标为 $P_1(x_1, y_1)$ 和 $P_2(x_2, y_2)$，并令 $dx = x_2 - x_1$；$dy = y_2 - y_1$。

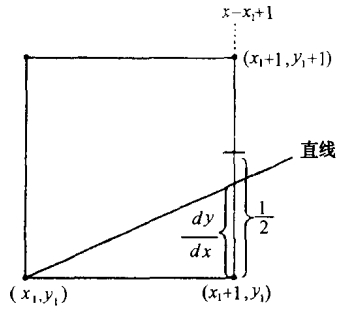


图 1-5 Bresenham 算法

现以 $0 \leqslant \frac{dy}{dx} \leqslant 1$ 的情况为例简要说明 Bresenham 算法的基本原理。对于斜率在这个范围内的直线，x 坐标每一步增加 1，即向右或右上方走；而 y 坐标就应增加 $\frac{dy}{dx}$，如图 1-5 所示。至于是向右走到象素 (x_1+1, y_1) 还是向右上走到象素 (x_1+1, y_1+1) 则取决于这两个象素中哪一个距离实际直线近。我们以这两点连线的中点 $(x_1+1, y_1+\frac{1}{2})$ 为判断标准。如果实际直线与垂直网格线 $x = x_1 + 1$ 的交点 $(x_1+1, y_1+\frac{dy}{dx})$ 在上述中点之下，即 $\frac{dy}{dx} < \frac{1}{2}$，则下一步应走到右邻象素上，并且下一步加 $\frac{dy}{dx}$（即 $2\frac{dy}{dx}$）并与 $\frac{1}{2}$ 进行比较以决定再下一步的走向；否则如果 $\frac{dy}{dx} \geqslant \frac{1}{2}$，就走到右上邻象素上，而下一步就应该用 $2\frac{dy}{dx}$ 与 $1+\frac{1}{2}$ 进行比较以决定再下一步的方向。这相当于用 $2\frac{dy}{dx}-1$ 与 $\frac{1}{2}$ 进行比较。现在我们可以总结出直线生成