对称扫描四步增量画线算法

柳士俊 邓北胜 徐怀刚
(中国气象局培训中心,北京 100081)

摘 要 直线的生成方法一直是计算机图形学中的基本问题,为此提出一种四步增量算法,该算法中只用到了整数加法运算和左移位运算,大大降低了硬件实现的复杂度,同时有效地提高了速度,易于硬件实现.由于每次可同时画出 4 个像素,故其效率大约是 Bresenham 算法的 3~4 倍.另外,还可根据直线的对称性进行对称扫描变换,以进一步提高效率.在算法中,根据直线的几何特征而采用的二叉树搜索法,使其平均每点判断次数几乎与 Bresenham 算法相同,而其平均每点迭代次数却远小于 Bresenham 算法.

关键词 增量算法 Bresenham 算法 四步增量算法 对称扫描
中图法分类号: TP391.41 **文章标识码**: A **文章编号**: 1006-8961(2002)10-1054-04

Four-step Scan-symmetrical Incremental Generation of Lines

LIU Shi-jun, DENG Bei-sheng, XUN Huai-gang
(*Chinese Meteorological Administration Training Centre, Beijing* 100081)

Abstract The generation of lines is elementary problem in computer graphics. A four-step incremental generation of lines is proposed. The algorithm firstly gives the choice standard of the pixel that the most closed to the straight line, and then converts it to integral variable form. So it develops and describes a new generation of lines, the four-step incremental generation of lines. The algorithm is easy to be implemented by hardware. It's only integer addition and left shift operation being used in the algorithm of incremental generation of lines. So the algorithm has lower complication in hardware and high speed. The four pixels being drew on the same time by using this algorithm. So the speed of the four-step incremental generation of lines is as three or four times as the speed of the Bresenham algorithm. Scan-symmetrical based on the symmetry of lines cause speed of the four-step incremental generation of lines increased as well. Binary search tree method based on geometrical characteristic of lines is used in the four-step algorithm. So almost the same times of decision per pixel between the four-step algorithm and the Bresenham algorithm, and the times of iterative per pixel of the four-step algorithm is much more less than the Bresenham algorithm.

Keywords Incremental generation of lines, Bresenham algorithm, Four-step incremental generation of lines, Scan-symmetrical

0 引 言

在计算机图形学的几何造型中,直线(线段)作为一种基本的几何元素,其生成方法一直是计算机图形学领域中令人们感兴趣的问题.目前广泛使用的直线扫描算法是 Bresenham 算法[1],由于采用增量算法,确定所求像素只需判断一下误差项的符号,故其速度较其他算法快.后来由 Wu 改进了这种算法[2],此算法每次决定两个像素的走法,因而称之为双步算法,其效率比 Bresenham 算法提高了大约一倍.而四步增量算法,每次可画出 4 个像素,效率大约是 Bresenham 算法的 3~4 倍.若进一步考虑到直线对于中心的对称性,采用从两个端点同时相向画线的办法,总的效率在此基础上仍可得到进一步提高.

收稿日期:2001-03-05;改回日期:2002-03-20

1　四步增量算法

如图 1 所示，设直线方程为 $y=kx+b$，$\Delta x>0$，$\Delta y\geq 0$，且 $\Delta x\geq\Delta y$，则 $0\leq k\leq 1$，由此可知

$$x_{inc}=1,y_{inc}=k \tag{1}$$

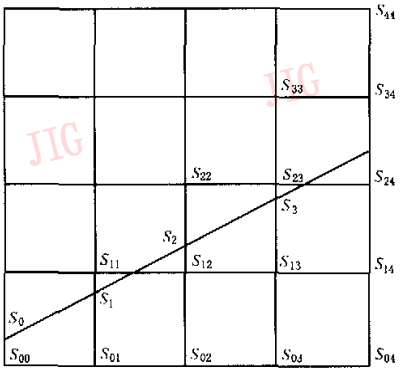


图 1

设直线与垂直网格线的交点为 $S_i,i=0,1,\cdots,4$，诸网格线交点为 $S_{ij},i,j=0,1,\cdots,4$，假设与当前起点 S_0 最接近的像素坐标为 $S_{00}(x_r,y_r)$. 为讨论方便，把网格纵坐标分为 5 个区间

$$I_m=\left[\frac{2m-1}{2},\frac{2m+1}{2}\right),m=0,1,\cdots,4 \tag{2}$$

很显然，由已知条件 $S_0\in I_0$，则与当前点 S_0 最接近的像素坐标为 $S_{00}(x_r,y_r)$，简记为 $S_0\to S_{00}$. 对于其他点，一般的选择标准为：当 $S_l\in I_m$ 时，与当前点 S_l 最接近的像素坐标为 $S_{ml}(x_r+l,y_r+m)$，简记为

$$S_l\in I_m\Leftrightarrow S_l\to S_{ml} \tag{3}$$

由简单的几何关系，S_l 的纵坐标 y 为

$$y=\left(1-\frac{l}{4}\right)y_1+\frac{l}{4}y_2 \tag{4}$$

式中 y_1,y_2 分别为当前始点 S_0 与末端点 S_4 的纵坐标. 令判别式

$$e_r=d_r-0.5=kx_r+b-y_r-0.5 \tag{5}$$

其中，d_r 为 S_l 点与 $y=y_r$ 行的距离.

式(5)是用来判别 S_l 是接近上面的网格点还是接近下面的网格点. 当 $e_r\geq 0$，选上面的网格点，$e_r<0$，选下面的网格点.

根据末端点 S_4 的位置，存在下面 4 种情况：

(1)当 $S_0\in I_0,S_4\in I_0$ 时，由式(3)得 $S_0\to S_{00}$，$S_4\to S_{04}$，即 $x_{r+4}=x_r+4,y_{r+4}=y_r$，代入式(5)得 $e_{r+4}=e_r+4k$. 又由式(4)得 $S_1\in I_0,S_2\in I_0,S_3\in I_0$，故由式(3)得 $S_1\to S_{01},S_2\to S_{02},S_3\to S_{03}$. 于是得到模式 0，简记为 $P_0=(S_{01}S_{02}S_{03}S_{04})$.

(2)当 $S_0\in I_0,S_4\in I_1$ 时，由式(3)得 $S_0\to S_{00}$，$S_4\to S_{14}$，即 $x_{r+4}=x_r+4,y_{r+4}=y_r+1$，代入式(5)得 $e_{r+4}=e_r+4k-1$. 又由式(4)得 $S_1\in I_0\cup I_1,S_2\in I_0\cup I_1,S_3\in I_0\cup I_1$，由于式(1)的限制，故当 $S_1\in I_1$ 时，只有 $S_2\in I_1,S_3\in I_1$，故由式(3)得 $S_1\to S_{11},S_2\to S_{12},S_3\to S_{13}$. 于是得到模式 1：$P_1=(S_{11}S_{12}S_{13}S_{14})$. 类似地可得到模式 2：$P_2=(S_{01}S_{12}S_{13}S_{14})$，模式 3：$P_3=(S_{01}S_{02}S_{13}S_{14})$，模式 4：$P_4=(S_{01}S_{02}S_{03}S_{14})$.

(3)当 $S_0\in I_0,S_4\in I_2$ 时，由式(3)得 $S_0\to S_{00}$，$S_4\to S_{24}$，即 $x_{r+4}=x_r+4,y_{r+4}=y_r+2$，代入式(5)得 $e_{r+4}=e_r+4k-2$. 又由式(4)得 $S_2\in I_1$，故由式(3)得 $S_2\to S_{12}$. 另外由式(4)还可得 $S_1\in I_0\cup I_1,S_3\in I_1\cup I_2$，故当 $S_1\in I_0,S_3\in I_1$ 时，由式(3)得 $S_1\to S_{01},S_2\to S_{12},S_3\to S_{13}$，于是得到模式 5：$P_5=(S_{01}S_{12}S_{13}S_{14})$. 对于其他情况类似地可得到模式 6：$P_6=(S_{01}S_{12}S_{23}S_{24})$，模式 7：$P_7=(S_{11}S_{12}S_{13}S_{14})$，模式 8：$P_8=(S_{11}S_{12}S_{23}S_{24})$.

(4)当 $S_0\in I_0,S_4\in I_3$ 时，由式(3)得 $S_0\to S_{00}$，$S_4\to S_{34}$，即 $x_{r+4}=x_r+4,y_{r+4}=y_r+3$，代入式(5)得 $e_{r+4}=e_r+4k-3$. 又由式(4)得 $S_1\in I_0\cup I_1,S_2\in I_1\cup I_2,S_3\in I_2\cup I_3$，故当 $S_1\in I_0$ 时，考虑到式(1)的限制，只有 $S_2\in I_1,S_3\in I_2$，故由式(3)得 $S_1\to S_{01},S_2\to S_{12},S_3\to S_{23}$. 于是得到模式 9：$P_9=(S_{01}S_{12}S_{23}S_{24})$. 对于其他情况类似地可得到模式 10：$P_{10}=(S_{11}S_{12}S_{23}S_{34})$，模式 11：$P_{11}=(S_{11}S_{22}S_{23}S_{24})$，模式 12：$P_{12}=(S_{11}S_{22}S_{33}S_{34})$.

(5)当 $S_0\in I_0,S_4\in I_4$ 时，由式(3)得 $S_0\to S_{00}$，$S_4\to S_{44}$，即 $x_{r+4}=x_r+4,y_{r+4}=y_r+4$，代入式(5)得 $e_{r+4}=e_r+4k-4$. 又由式(4)得 $S_1\in I_1,S_2\in I_2,S_3\in I_3$，故由式(3)得 $S_1\to S_{11},S_2\to S_{22},S_3\to S_{33}$. 于是得到模式 13：$P_{13}=(S_{11}S_{22}S_{33}S_{44})$.

根据上述分析，给出如下 4 步增量算法：

算法 1

```
初始化
线段起点(x1,y1)，线段终点(x2,y2)，画点起始坐标(x,y)
变量进行整数化处理
若 x<x2 则进行以下处理，否则结束
  if  e+4k<2
      if  e+4k<1
          if  e+4k<0 /*S4∈I0*/
              draw_pixel(x,y,P0),e=e+4k
```

```
        else  /* 0<=e+4k<1,S4∈I1 */
            if  e+2k<0
                if  e+3k<0 draw_pixel(x,y,P4)
                else  draw_pixel(x,y,P3)
            else
                if  e+k<0 draw_pixel(x,y,P2)
                else  draw_pixel(x,y,P2)
            e=e+4k  1, y=y+1
    else  /* 1<=e+4k<2,S1∈I2 */
        if  e+k<0
            if  e+3k<1 draw_pixel(x,y,P1)
            else  draw_pixel(x,y,P6)
        else
            if  e+3k<1 draw_pixel(x,y,P5)
            else  draw_pixel(x,y,P8)
        e=e+4k-2, y=y+2
else
    if  2<=e-4k<3, /* S4∈I3 */
        if  e+2k<1
            if  e+k<0 draw_pixel(x,y,P5)
            else  draw_pixel(x,y,P10)
        else
            if  e+3k<2 draw_pixel(x,y,P11)
            else draw_pixel(x,y,P12)
        e=e-4k-3,y=y+3
    else /* 3<=e+4k,S1∈I1 */
        draw_pixel(x,y,P13),e=e+4k-4,y=y+4
    x=x+4
end
```

为了使算法适宜于用硬件的快速实现，要求算法无乘除法运算，只用到加法运算及左移位(乘 2)运算，且为整数运算，为了做到这一点，引入变换 $f=2\times\Delta x\times e$，上述算法中的各个判断都可用以上变换进行整数化处理. 由于 f 与 e 符号相同，因此可用 f 代替 e 作判别式，使算法中只有整数运算. 由此，选择标准式(3)并变为

$$e+lk\in[m-1,m)\Leftrightarrow S_l\rightarrow S_{ml} \qquad (6)$$

过程 draw_pixel(x,y,P_i)按照指定模式 P_i 从位置(x,y)开始写 4 个像素，采用并行写帧缓存的技术，可使之与写单个像素的速度一样[2].

对模式的判断，程序中采用了二叉树搜索法，模式 0 为 3 次判断，模式 1，2，3，4 为 5 次判断，模式 5～12为 4 次判断，模式 13 为 2 次判断，平均每画一点的判断次数为 $\frac{3\times1+5\times4+4\times8+2\times1}{14\times4}=1.02$，几乎与 Bresenham 算法(一次)一样.

在迭代的次数上，模式 0 为 2 次迭代，其余 13 种为 3 次迭代，故平均每画一点的迭代次数为 $\frac{2\times1+3\times13}{14\times4}=0.73$，而 Bresenham 算法每画一点的迭代次数为 $\frac{2+3}{2}=2.5$，这样，该算法相对于 Bresenham 算法的倍率为 $\frac{2.5}{0.73}=3.42$.

综上所述，该算法的效率大约是 Bresenham 算法的 3～4 倍. 另外，当算法运算到线段的末端时，像素可能不够 4 个，此时应在程序中加一个判断，然后转入 Bresenham 算法.

2 对称扫描四步增量画线算法

考虑到线段对中心点的旋转对称性，可以从其两端同时进行扫描转换，可在一步迭代中同时画 8 个像素点. 对称扫描四步增量算法如下：

算法 2

初始化

线段起点(x_1,y_1)，线段终点(x_2,y_2)，画点起始坐标$(x,y;u,v)$，其中，$u=x_2-4$，$v=y_2-4$

变量进行整数化处理

若 $x<x_2$ 则进行以下处理，否则结束

```
if  e+4k<2
    if  e+4k<1
        if  e+4k<0 /* S4∈I2 */
            draw_sy(x,y,u,v,P6,P65),e=e-4k
        else  /* 0<=e-4k<1,S4∈I1 */
            if  e+2k<0
                if  e+3k<0 draw_sy(x,y,u,v,P4,P45)
                else  draw_sy(x,y,u,v,P3,P38)
            else
                if  e+k<0 draw_sy(x,y,u,v,P1,P18)
                else draw_sy(x,y,u,v,P2,P28)
            e=e-4k-1,y=y+1,v=v-1
    else  /* 1<=e+4k<2, S4∈I4 */
        if  e+k<0
            if  e+3k<1 draw_sy(x,y,u,v,P5,P58)
            else draw_sy(x,y,u,v,P6,P68)
        else
            if  e+3k<1 draw_sy(x,y,u,v,P7,P75)
            else  draw_sy(x,y,u,v,P8,P88)
        e=e+4k-2, y=y+2,v=v-2
else
    if  e+4k<3, /* S4∈I3 */
```

```
        if  e+2k<1
            if  e+k<0 draw  sy(x,y,u,v,P9,P9S)
            else  draw_sy(x,y,u,v,P10,P10S)
        else
            if  e+3k<2 draw  sy(x,y,u,v,P11,P11S)
            else  draw_sy(x,y,u,v,P12,P12S)
        e=e+4k-3, y=y+3, v=v-3
    else  /* 3<=e+4k, S4∈I4 */
        draw_sy(x,y,u,v,P13,P13S),
        e=e+4k-4, y=y+4, v=v-4
    x=x+4, u=u-4
end
```

上述算法中，过程 draw_sy(x,y,u,v,P_i,P_{iS}) 按照指定模式 P_i 从位置(x,y)开始写 4 个像素，同时按照指定模式 P_{iS}从位置(u,v)开始写 4 个像素，故同时可画 8 个像素点，模式 P_{iS}是模式 P_i 的旋转对称模式，即原 5×5 点阵模式绕其中心旋转而成，设 $ROT(\theta)(P_i)$表示对模式 P_i 旋转 θ 角的旋转变换群，则 $P_{iS}=ROT(\pi)(P_i)$. 此时，并行写帧缓存的带宽增加一倍，目前的技术完全可以满足.

与算法 1 类似，在对模式的判断上，P_0，P_{0S}为 3 次判断，$P_1\sim P_4$，$P_{1S}\sim P_{4S}$为 5 次判断，$P_5\sim P_{12}$，$P_{5S}\sim P_{12S}$为 4 次判断，P_{13}，P_{13S}为 2 次判断，平均每画一点的判断次数为$\frac{3\times2+5\times8+4\times16+2\times2}{28\times8}=0.51$，几乎是 Bresenham 算法的一半.

在迭代的次数上，P_0，P_{0S}为 3 次迭代，其余为 5 次迭代，故平均每画一点的迭代次数为$\frac{3\times2+5\times26}{28\times8}=0.607$，算法相对于 Bresenham 算法的倍率为$\frac{2.5}{0.607}=4.12$.

3　讨论与结论

从表 1 可以看出，随着算法中每次迭代画点数的增加，平均每画一点的判断次数和平均迭代次数都在减小，故综合来看，对称扫描四步增量算法是最优的. 同时我们也看到，这种效率的提高并非是线性的，随着点数的增加，图 1 右下脚三角形内的候选点的可选择样本是按阶乘增长的，如 Bresenham 算法只有两种选择. 双点算法[2]有 6 种选择，根据直线的几何特征，化简为 4 种模式，压缩率为 67%，本文四点算法总候选样本有 120 种，根据直线的几何特征化简为 14 种模式，压缩率为 12%，这为有效地降低平均判断次数和平均迭代次数提供了可能. 随着算法中每次迭代时所画的点数的增加，算法的复杂性也随之提高，由于进行了整数化处理，使该算法只用到了整数加法运算和左移位运算，从而大大降低了硬件实现的复杂度，同时也有效地提高了速度.

表 1　各算法平均每画一点的判断次数与迭代次数比较

算法	Bresenham 算法	双点算法	四步增量算法	对称扫描四步增量算法
判断次数	1	1.13	1.02	0.51
迭代次数	2.5	1.38	0.73	0.607

由此可见，一次并行地画多个点的算法从原理到实践上都是可行的，仿此方法可推广至 5 点、6 点等等，进一步地，可以考虑研究是否存在一次画 n 个点的通用算法. 本文在理论上指出了并行多点画法的基本方法：(1)根据直线的几何特征寻找最接近直线上点的像素点的选择标准，进而构造模式集；(2)算法上采用二叉树搜索，减少了判断时延；(3)根据直线的对称性进行对称扫描变换.

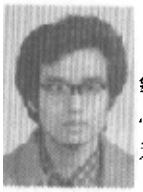
在实用领域，本算法可构成各类图象图形卡的底层基本算法，由于直线生成是许多其他图形算法的基元，故可以大大提高图象图形卡的整体性能指标.

参 考 文 献

1　孙家广，杨长贵. 计算机图形学(新版)[M]. 北京：清华大学出版社，1995：165～170.

2　Wu Xiaolin, Rokne J G. Double-step incremental generation of lines and circles [J]. Computer Vision, Graphics, and Image Processing, 1987, 37(Mar): 331～344.

柳士俊　1964 年生，1986 年获青岛海洋大学物理系理学学士学位，现为中国气象局培训中心科技培训部讲师. 主要研究领域为计算机系统结构、算法优化、流体力学、高速处理器结构设计.



邓北胜　1962 年生，1984 年获南京气象学院学士学位，现为中国气象局培训中心科技培训部主任，副教授. 主要研究领域为大气科学、计算机科学.

徐怀刚　1963 年生，1984 年获南京大学学士学位，现为中国气象局培训中心科技培训部讲师. 主要研究领域为大气科学、计算机科学.