第4卷(A版)　第7期
1999年7月
中国图象图形学报
Journal of Image and Graphics
Vol.4(A),No.7
July 1999

改进的 Bresenham 直线生成算法

郑宏珍
(哈尔滨工业大学计算机系,哈尔滨　150001)
赵　辉
(哈尔滨工业大学数学系,哈尔滨　150001)

摘　要　提出了一种新的直线生成算法,该算法通过预知每个象素行要点亮的象素点,实现了在一个象素行上同时处理多个象素。在配有块写入图形存储器的系统中,该算法可以实现并行填充象素。对小斜度直线,该算法可以避免 Bresenham 算法中偏差计算浪费现象。

关键词　Bresenham 算法　并行处理　图形存储器

1　图形存储器与 Bresenham 算法的缺陷

早期的 VRAM 图形存储器芯片和现在流行的 SGRAM 图形存储器芯片[1]都支持一种块写入方式,在块写入方式下,可以并行写入一行上的八个或更多具有相同颜色的象素点。基于这种图形存储器芯片的图形处理算法要最大限度地考虑并行性。传统的 Bresenham 算法[2]一直被认为是一种简单、高效的直线生成算法,但是从现在的观点看它是一种逐个象素点计算的算法,并且自身存在明显的偏差计算浪费现象。Bresenham 算法的工作过程如下:

假设直线的斜率 $k<1$,点亮直线的左侧端点处象素,然后变量 x 每次增加一个象素,变量 y 根据直线偏差的计算结果每次增加0或1,即下一个点亮的象素点根据直线偏差结果选择 $(x+1,y)$ 或 $(x+1,y+1)$,直到直线的右侧端点处结束。

直线在左侧起始处的偏差 $e_0=0$,第 m 步的偏差 $e_m=k\times m-[k\times m]$,其中 $[k\times m]$ 表示不超过 $k\times m$ 的最大整数。Bresenham 算法规定当 $e_m\geq 0.5$ 时,点亮的直线上方象素点,$e_m<0.5$ 时,点亮的直线下方象素点。

当直线的斜率较小时,Bresenham 算法将出现偏差计算浪费现象。如图1所示,直线的斜率 $k=\frac{1}{6}$,直线的起点为(0,0),终点为(12,2)。从(0,0)点出发 x 每次增加一个象素,计算偏差。初始偏差 $e_0=0$,在 $x=1$ 处,$e_1=e_0+k=\frac{1}{6}<\frac{1}{2}$,点亮的直线下方象素点(1,0);在 $x=2$ 处,$e_2=e_1+k=\frac{1}{3}<\frac{1}{2}$,点亮的直线下方象素点(2,0);在 $x=3$ 处,$e_3=e_2+k=\frac{1}{2}$,点亮的直线上方象素点(3,1)。显然前面两次计算均是无用的,仅有第3次计算是有效的。继续后面的计算可知 e_4,e_5,e_6,e_7,e_8 的计算均是无用的,e_9 是有效的。由此可以看到 Bresenham 算法的偏差计算浪费非常严重。

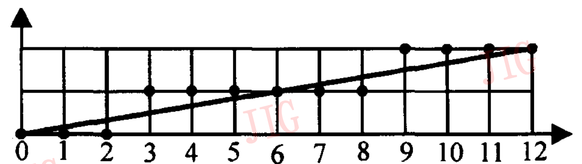

图1　Bresenham 算法的例子

收稿日期:1998-04-20;收到修改稿日期:1998-09-04

这个例子启发我们，能否仅在行或列增加的地方或附近进行计算，每次将一行或列上要点亮的连续象素点全部确定下来，这样既节省了计算，又为象素点的并行点亮（填充）提供可能性。

2　改进的 Bresenham 算法

以下的讨论中，我们假设直线的斜率 $k \le 1$。Bresenham 算法没有定义偏差为 0.5 时，应点亮直线上方还是下方的象素，我们定义同时点亮直线上方和下方的象素。

引理 1　直线与若干象素行相交，则这些象素行上按 Bresenham 算法被点亮的象素数目或者相等或者相差 1。

证明：按 Bresenham 算法定义，当偏差大于 0.5 时点亮直线上方的象素，当偏差小于 0.5 时点亮直线下方的象素，如图 2 所示，象素行上被点亮的象素一定位于一个连续的区域内，该区域的两个端点与直线沿垂直象素行方向上的距离为 0.5。

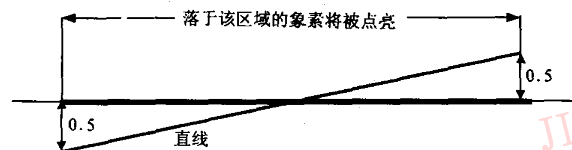

图 2　象素行上被点亮的象素区域

显然，对每个象素行来讲，这个区域的长度都相等，并且为 $\frac{1}{k}$。

现在设 a,b 为任意两个与直线相交的象素行，a 行被点亮 m 个象素，则 a 行被点亮的象素区域长度小于 $m+1$；若 b 行上被点亮的象素数目至少为 $m+2$ 的话，则 b 行上被点亮的象素区域长度大于等于 $m+1$，二者长度不能相等，证毕。

定理 1　设直线的斜率 k 满足 $\frac{1}{2(n+1)} < k \le \frac{1}{2n}$，$n$ 为正整数，则：

① 第 1 行除端点外点亮 n 个象素；

② 第 2 行点亮的象素数目为 $2n, 2n+1$ 二者之一。

证明：

① 设第 1 行除端点外点亮 i 个象素，因第 i 个象素被点亮，我们有：$i \times k \le 0.5$。

故 $i \le \frac{0.5}{k} < 0.5 \times 2(n+1) = n+1$。

又因第 $i+1$ 个象素没有被点亮，我们有：$(i+1) \times k > 0.5$。

故 $i > \frac{0.5}{k} - 1 \ge 0.5 \times 2(n-1) = n-1$。

因此 $i = n$。

② 设前两行除第 1 行端点外点亮的象素数目之和为 j，则 j 应满足：

$$j \times k \le 1 + \frac{1}{2} < (j+1) \times k$$

由此推出：$3n - 1 < j \le 3n + \frac{3}{2}$

故 j 的可能取值为：$3n, 3n+1$。由于第 1 行除端点外点亮 n 个象素，因此第二行点亮的象素数目为 $2n, 2n+1$ 二者之一。证毕。

如果将第 1 行看作是中间行，即认为直线通过端点继续向外延伸，则由对称性可知在定理 1 的条件下第 1 行点亮 $2n+1$ 个象素，结合引理 1，立得：

定理 2　设直线的斜率 k 满足 $\frac{1}{2(n+1)} < k \le \frac{1}{2n}$，$n$ 为正整数，则任意中间行上点亮的象素数目为 $2n, 2n+1$ 二者之一。

下面我们将根据上面的结论给出新的直线生成算法。我们已经知道一行上要点亮的象素数目为 $2n, 2n+1$ 二者之一，我们仅需在每行上被点亮的象素区域的右端点附近确定到底是 $2n$ 还是 $2n+1$。我们可以先点亮 $2n$ 个象素，然后在第 $2n$ 个象素处计算偏差，如果偏差大于等于 $\frac{1}{2}$，则结束该行；否则在第 $2n+1$ 个象素处再计算一次偏差，如果偏差大于 $\frac{1}{2}$，则结束该行。这样我们最多在每行计算 2 个点的偏差。

上述按 Bresenham 算法的偏差概念计算偏差存在的问题是：我们有时以 $2n$ 作为步长，有时以 1 作为步长计算偏差，增加了计算复杂性。为了避免这种计算复杂性，我们引入一种新的偏差概念。

如图 3，我们考虑每行上被点亮的象素区域，它的长度为 $\frac{1}{k}$，点亮 $2n$ 个象素后，剩余长度为 $\frac{1}{k} - (2n-1)$，如果该长度大于 1，则至少还要点亮一个象素。基于这种思想我们考虑将每行点亮若干象素

后的剩余长度作为偏差。设第 m 步计算偏差在第 m 条中间行被点亮的象素区域的右端点处进行,我们定义第 m 步的偏差 e_m 为按步长 $2n$ 点亮了 $2n$ 个象素后,最右端被点亮的象素距该行上应该被点亮的象素区域的右端点的距离。因为每行上被点亮的象素区域恰好是首尾相接的,所以第 m 行被点亮的象素区域的左端点距离该行上应该被点亮的象素区域的左端点的距离为 $1-e_{m-1}$。我们有:

$$e_m=\frac{1}{k}-(2n-1)-(1-e_{m-1})=\frac{1}{k}-2n+e_{m-1}$$

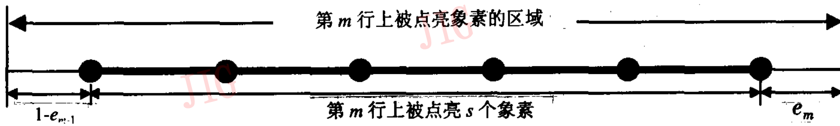

图3 偏差的定义

按上述偏差的定义,进行如下处理:若 $e_m\geq1$,在右端继续点亮一个象素,并置 $e_m=e_m-1$。由于起始行被点亮的象素区域的长度可看作$\frac{1}{2k}$,该行上点亮了 $n+1$ 个象素,故有:$e_0=\frac{1}{2k}-n$。当$\frac{1}{2}<k\leq1$ 时,可看作第1行除端点外点亮 $n=0$ 个象素,此时第二行点亮的象素数目为 $2n$, $2n+1$ 二者之一的结论仍成立。中间行被点亮象素区域的长度$\frac{1}{k}\leq1$,其中至少包含一个要点亮的象素,因此我们的算法仍成立。

当 $k>1$ 时,交换 x,y 变成 $k<1$ 的情况处理。

3 算法实现

以下我们简单描述算法产生直线的过程和直线的初始输入数据,对直线的类型判别和异常处理部分略去。算法可以经适当处理变成整数算法。

直线的原始数据为:

(x_0,y_0):直线的起点(在屏幕左边),直线的终点(在屏幕右边)。

算法输入数据为:$(x_0,y_0),y_1,n,s,e$,

其中:记 $k=\frac{y_1-y_0}{x_1-x_0}\leq1$ 为直线的斜率。$n=\left[\frac{1}{2k}\right]$;如果 $n=0, s=\frac{1}{k}$,否则 $s=\frac{1}{k}-(2n-1)$;$e=\frac{1}{2k}-n$。

函数 *FillPixels*(x,y,c)表示填充象素,其中(x,y)表示行填充起点,c 为连续填充的象素数。C语言格式的算法描述为:

```
Line(x0, y0, y1, n, s, e)
  int   x0, y0, y1, n;
  float  s, e;
  {
     FillPixels(x0, y0, n+1);
     for (x = x0 + n + 1, y = y0 + 1; y < y1; y++){
        e = s - 1 + e;
        if (e > 1){
           FillPixels(x, y, 2 * n + 1);
           x = x + 2 * n + 1;
           e = e - 1;
        }
        else{
           FillPixels(x, y, 2 * n);
           x = x + 2 * n;
        }
     }
     FillPixels(x0, y0, n + 1)
```

4 结 论

该算法为象素的并行填充提供了可行性,已经在哈尔滨工业大学设计的三维图形加速卡中用硬件实现,该图形卡采用 SGRAM,在每个象素行上可并行填充16个象素。就软件算法而言,在小斜度直线的情况下,其处理效率明显高于传统的 Bresenham 算法。

参 考 文 献

1 Graphics Memory Data Book. SAMSUNG Eleltronics, 1997.

2 Rogers D F. Procedural Element for Computer Graphics. McGraw-Hill, Inc, 1985.

郑宏珍　1967 年生,哈尔滨工业大学计算机系讲师。主要研究方向为神经网络、计算机图形学。

赵　辉　1963 年生,哈尔滨工业大学数学系副教授。主要研究方向为计算机图形学、计算机图象处理。

The Improvement of Bresenment Algorithm

Zheng Hongzhen

(*Department of Computer Science*, *Harbin Institute of Technology*, *Harbin* 150001)

Zhao Hui

(*Department of Mathematics*, *Harbin Institude of Technology*, *Harbin* 150001)

Abstract　In this paper, we present a new method for generating a straight line in raster devices. The algorithm foresees the pixels which will be activated in a raster line and implements parallel filling pixels. It will play an important role in computer system with graphics memory. For small slope line, the new algorithm can avoid the calculating of errors in Bresenham algorithm.

Keywords　Bresenham algorithm, Parallel process, Graphics memory