Bresenham直线生成算法的改进

贾银亮 张焕春 经亚枝
(南京航空航天大学自动化学院, 南京 210016)

摘 要 直线是图形的基本元素,研究其生成算法具有重要意义。由于经典的 Bresenham直线生成算法一次计算只能生成一个像素点,效率较低。为了提高直线生成效率,通过对其进行改进,提出了一种利用直线前两行像素行的像素点数目来计算其余各像素行的像素点数目的算法。该算法在保持 Bresenham算法不使用取整和小数运算的优点下,还提高了直线生成效率,一次计算可以生成一个像素行。

关键词 计算机图形学 Bresenham算法 判定变量
中图法分类号: TP391.41 **文献标识码**: A **文章编号**: 1006-8961(2008)01-0158-04

A Modified Bresenham Algorithm of Line Drawing

JIA Yin-liang, ZHANG Huan-chun, JING Ya-zhi
(*Nanjing University of Aeronautics & Astronautics, Nanjing* 210016)

Abstract Bresenham algorithm is the most fundamental algorithm for drawing line segments in computer graphics. Canonical Bresenham algorithm can only generate one pixel of a line each time. We proposed a new method by improving it in a novel way, which can generate pixel of a line row by row according to the first and second row of a line and inherit the advantages of Bresenham algorithm without division and decimal fraction. Finally, its efficiency has been proved to be much better than those existing methods.

Keywords computer graphics; Bresenham algorithm; determinant variable

1 引 言

由于直线是生成各种图形的基本元素,且直线生成算法是其他各类图形算法的基础,因此,目前直线的生成算法已得到了人们广泛的研究,并已出现了许多有效的算法。其中最著名的是 Bresenham算法,其优点在于不需要进行小数和取整运算,只需要使用整数加法和乘法来计算即可。Bresenham算法的缺点在于效率不高,也没有充分利用像素之间的相关性,一次计算只能生成一个像素点[1]。

许多文献对 Bresenham算法进行了一定程度的改进[2~4],它们都是通过求直线斜率的方法来计算各像素行的像素点数目,以提高直线生成的效率。但这些算法改进幅度不大,主要表现在:改进算法往往引入了浮点运算,甚至需要除法或取整运算,这恰恰破坏了 Bresenham算法不进行小数和取整运算的特点。这些改进算法一次计算虽然可以画出多个像素点,但运算更加复杂。

本文在对 Bresenham算法和现有的改进算法进行深入研究后,充分认识到经典的 Bresenham算法基本思想的重要性,又考虑到 Bresenham算法本身的缺陷,提出了一种改进算法,从而克服了上述直线生成算法的弱点。该改进算法在保持 Bresenham算法只需要使用整数加法和乘法,不需要进行小数和取整运算的优点的基础上,通过一次计算生成一个像素行,其执行效率明显高于其他直线生成算法,并且算法实现简单、运算量少。

2 改进的 Bresenham算法

本文的算法和程序只对斜率在[0, 1]之间的直线进行讨论,并设直线的起点为 (x_0, y_0),终点为 (x_1, y_1),且 $x_1 \geqslant x_0$、$y_1 \geqslant y_0$,而对于一般的情况则利

收稿日期: 2006-07-11; 改回日期: 2006-09-12

第一作者简介: 贾银亮(1979~)男。2001年获南京航空航天大学测试计量技术及仪器专业硕士学位。现为南京航空航天大学自动化学院测试工程系讲师,在职博士研究生。主要从事计算机图形学和总线技术方面的研究。E-mail: flybeamuaa@163.com

用变换不难求得。

引理 1　用 Bresenham 算法生成的直线在除起始和终止两像素行外, 其他各像素行的像素点个数 e满足:

$$Q \leqslant e \leqslant Q + 1$$

式中, Q 为 (dx/dy)的下取整 ("/"表示除法)、$dx = x_1 - x_0$、$dy = y_1 - y_0$。

例如, 一条从 (0, 0)到 (100, 16)的直线, 其 $Q = 6$, 则该直线除起始和终止两像素行外, 其他各像素行的像素点个数为 6或 7个。

证明　Bresenham 算法利用当前被选中的像素来选择下一个像素。设坐标为 (x_p, y_p)的像素点 $\boldsymbol{p}$ 已被选中, 则该直线的下一个像素点应为点 $\boldsymbol{E}$ 或点 $\boldsymbol{G}$ (如图 1所示)。考察点 $\boldsymbol{E}$ 和点 $\boldsymbol{G}$ 的中点 $\boldsymbol{M}$, 设 $F(x, y)$为待生成直线, 若 $F(\boldsymbol{M}) > 0$, 则点 $\boldsymbol{M}$ 在直线下方, 应选点 $\boldsymbol{G}$, 若 $F(\boldsymbol{M}) \leqslant 0$, 则点 $\boldsymbol{M}$ 不在直线下方, 应选点 $\boldsymbol{E}$。

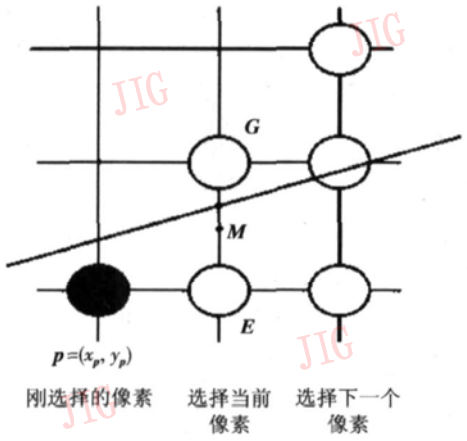

图 1　Bresenham 算法的像素栅格

Fig 1　Pel grid of Bresenham

Bresenham 算法首先定义直线的一个像素点的判定变量 $d = F(x_p + 1, y_p + 1/2)$, 式中 (x_p, y_p)是上个像素点的坐标; 然后生成直线时, 就可根据各个像素点的判定变量 d 来选择: 若 $d > 0$, 则在下一个像素行生成一个像素点, 即选点 $\boldsymbol{G}$; 若 $d \leqslant 0$, 则在原像素行再生成一个像素点, 即选点 $\boldsymbol{E}$。在选点的同时计算下一个像素点的判定变量, 当选点 $\boldsymbol{G}$ 时, $d = d + 2(dy - dx)$; 当选点 $\boldsymbol{E}$ 时, $d = d + 2dy$。

现假设某像素行有 e个像素点 $(e = 1, 2\cdots\cdots)$, 而将该像素行的各像素点的判定变量设为 $d_1 \sim d_e$, 将上一像素行最后一像素点的判定变量设为 d_k, 将下一像素行第 1个像素点的判定变量设为 d_i (如图 2所示)。

若上一像素行只有一个像素点, 则由双步直线

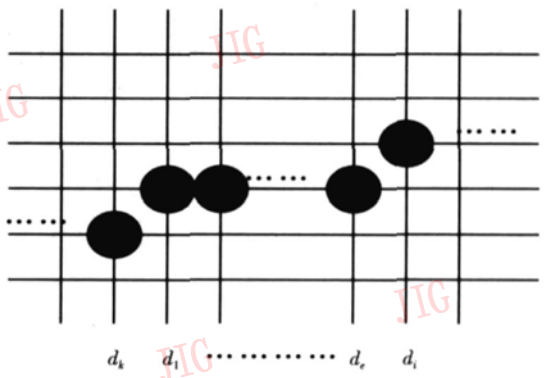

图 2　有 e个像素点的像素行

Fig 2　A pel line

生成算法可知, $1 \leqslant dx/dy < 2$, 且该直线每个像素行第最多有两个像素点[1], 显然此时引理 1成立。

若上一像素行不止一个像素点, 则由 Bresenham 算法知

$$\begin{cases} d_1 = d_k + 2dy & (1) \\ d_e = d_1 + 2(e-1)dy - 2dx & (2) \\ d_i = d_e + 2dy & (3) \end{cases}$$

且 $d_k \leqslant 0$、$d_1 > 0$、$d_e \leqslant 0$、$d_i > 0$

将式 (1)、式 (2)代入式 (3), 则

$$\begin{aligned} d_i &= d_e + 2dy = d_1 + 2(e-1)dy - 2dx + 2dy \\ &= d_k + 2dy + 2(e-1)dy + 2(dy - dx) \\ &= d_k + 2(e+1)dy - 2dx \end{aligned}$$

即 $d_k + 2(e+1)dy - 2dx > 0$, 而 $d_k \leqslant 0$, 则 $2(e+1)dy - 2dx > 0$, 可得 $e > dx/dy - 1$, 又因 e为正整数, 所以 $e \geqslant Q$。

若因为 $d_e = d_1 + 2(e-1)dy - 2dx \leqslant 0$, 且 $d_1 > 0$, 所以 $2(e-1)dy - 2dx < 0$, 则 $e < dx/dy + 1$, 又因 e为正整数, 所以 $e \leqslant Q + 1$。

则 $Q \leqslant e \leqslant Q + 1$, 得证。

由引理 1可知, 除起始和终止两像素行外, 其他各像素行有 Q 或 $Q + 1$个像素点, 设某像素行第 1个像素点判定变量为 d_0, 那么只要确定与本像素行第 1个像素点之后第 $k(k = Q)$个像素点对应的判定变量 (设为 d_k)的符号就可以知道该像素行应有几个像素点。而 $d_k = d_1 + 2kdy - 2dx$, 若 $d_k \leqslant 0$, 则该像素行有 $Q + 1$个像素点, 同时把 d 值加上 $2dy$ 即得到下一像素行第 1个像素点的判定变量; 否则该像素行有 Q 个像素点, 此时 d_k 即为下一像素行第 1个像素点的判定变量。这样只需要计算一次 d 值即可画出一个像素行的全部像素点, 因而效率较高。

如何计算 Q 是本文的核心内容, 若直接用 (dx/dy)下取整来求 Q, 会破坏 Bresenham 算法只需要使用整数加法和乘法, 而不需要进行小数和取整运算的优点。为了提高效率又不做取整运算, 本文提出了一种用一条直线前两个像素行的像素点个数来求 Q 的算法。下面先考虑直线的第 1个像素行的像素点个数与 Q 的关系。

引理 2 若 Bresenham 算法生成的直线的第 1个像素行 (即 $y=y_0$ 的像素行)有 n 个像素点, 则 $2n-2\leqslant dx/dy<2n$。

证明 若第 1个像素行有 n 个像素点, 则该直线第 n 个像素点的 $d\leqslant 0$而第 $n+1$个像素点的 $d>0$, 即

$$\begin{cases} d_0+(n-2)2dy\leqslant 0 \\ d_0+(n-1)2dy>0 \end{cases}$$

式中, $d_0=2dy-dx$ 为 d 的初值, 即与该直线第 2个像素点对应的 d 值, 将 d_0 代入上式可得

$$\begin{cases} 2dy-dx+(n-2)2dy\leqslant 0 \\ 2dy-dx+(n-1)2dy>0 \end{cases}$$

则 $2n-2\leqslant dx/dy<2n$, 得证。

由引理 2可知, Q 为 $2n-1$或 $2n-2$, 为进一步确定 Q, 再考虑直线第 2个像素行的像素点个数与 Q 的关系。

引理 3 若 Bresenham 算法生成的直线的第 2个像素行 (即 $y=y_0+1$的像素行)有 m 个像素点, 则 $m-1\leqslant dx/dy<m+1$。

证明 由引理 1知, $m=Q$ 或 $m=Q+1$。若 $m=Q$, 则 $m+1>dx/dy\geqslant m$, 若 $m=Q+1$, 则 $m>dx/dy\geqslant m-1$, 所以 $m-1\leqslant dx/dy<m+1$, 得证。

由引理 2、3就可以得到以下定理:

定理 若 Bresenham 算法生成的直线的前两个像素行分别有 n和 m 个像素点, 则

$$Q=\begin{cases} 2n-2 & 若\ 2n>m+1 \\ 2n-1 & 若\ 2n=m+1 \\ m-1 & 若\ 2n<m+1 \end{cases}$$

证明 由引理 2、3知, 若 $2n>m+1$, 而且 $2n$、$m+1$都是正整数, 则 $2n-1\geqslant m+1$, 即因

$$2n-1\geqslant m+1>dx/dy\geqslant 2n-2$$

故

$$Q=2n-2 \qquad (4)$$

若 $2n<m+1$, 则 $m\geqslant 2n>dx/dy\geqslant m-1$所以

$$Q=m-1 \qquad (5)$$

若 $2n=m+1$, 则考虑直线第 $n+1$个像素点 (即第 2像素行第 1个像素点)的判定变量 $d_{n+1}=d_0+2(n-1)dy>0$, 而第 $m+n$个像素点 (第 2像素行最后 1个像素点)的判定变量 $d_{m+n}\leqslant 0$, 且第 $m+n+1$个像素点的 $d_{m+n+1}>0$, 即

$$\begin{cases} d_{n+1}+2(m-1)dy-2dx\leqslant 0 \\ d_{n+1}+2mdy-2dx>0 \end{cases}$$

若将 $d_{n+1}=d_0+2(n-1)dy$、$d_0=2dy-dx$、$2n=m+1$代入, 则得 $2n-2/3>dx/dy\geqslant 2n-4/3$, 即

$$Q=2n-1 \qquad (6)$$

由式 (4) ~ (6)结果, 得证。

由该定理可见, 只要比较 $2n$与 $m+1$的大小就可以确定 Q, 这样就避免了取整运算。

3 算法实现

从以上的描述就可以实现优于 Bresenham 的直线生成算法, 其 C语言程序如下。程序中, 变量 *valuel*, *valueB* 分别表示直线颜色和背景色; 函数 *Linel* (int *x*, int *y*, int *n*, int *value*)表示以颜色 *value* 点亮 (x, y)、$(x+1, y)$、$(x+2, y)$、…、$(x+n, y)$, 共 n个像素点, 该函数执行完成后, $x=x+n$; 变量 $k=Q$, $k_1=Q+1$; 变量 *cre* 为像素行第 1个像素点的 d到该点后面第 k个像素点 d值的增量; 若用数组$N[\,]$记录第 2、3像素行第 1个像素点的 x坐标, 则直线的第 1个像素行包含像素点的个数 $n=N[0]-x_0$; 第 2个像素行包含像素点的个数 $m=N[1]-N[0]$, 而且只需两次减法就可求得 n和 m。程序用 Bresenham 算法生成第 1、2和最后一个像素行, 而其他像素行则由新算法生成。

```
lineB(x0, y0, x1, y1, valuel, valueB)
{ int dx, dy, incrE, incrG, d, x, y;
/* Bresenham 算法所定义的变量* /
  int n, m, k, k1, i= 0, cre;
  intN[2];
/* 改进算法增加的变量* /
  dx= x1- x0;
  dy= y1- y0;
  d= 2* dy- dx;
  incrE= 2* dy;
  incrG= 2* (dy- dx);
  x= x0;
  y= y0;
  Linel(x, y, 1, valuel);
  while((y≤y0+ 1)&& (x≤x1))
  { if(d≤0)
```

```
    {d+ = incrE; x+ +; }
  else
    {d+ = incrG; x+ +; N[i+ +] = x; y+ +; }
 Linel(x, y, l, valuel);
  }
/* 用 Bresenham 算法生成第 1、2像素行* /
 d- = incrG;
/* d对应第 3像素行第 1个像素点* /
 Linel(x, y, l, valueB);
/* 防止第 3像素行第 1个像素点被点亮两次* /
 n=N[0]-x0; m=N[1]-N[0];
 n= 2* n; m+ +;
 if(n>m)
  {k= n- 2; k1=m; }
 else
  { if(n= =m)
      {k= n- 1; k1= n; }
    else{k=m- 2; k1= n; }
    }
 cre= incrG+ (k- 1)* incrE;
while(y< y1)
 {d+ = cre;
  if(d≤0)
  {Linel(x, y, k1, valuel);
   d+ = incrE;
  }
  else
   {Linel(x, y, k, valuel);
    }
    y+ +;
   }
   Linel(x, y, x1- x+ 1, valuel);
/* 生成最后一个像素行* /
}
```

程序只需两次减法就可求得 n 和 m, 再通过比较运算即可求得 Q, 这样不需要取整或除法运算, 也没有浮点运算, 而且求得 Q 后只需一次计算即可生成一个像素行, 其效率比 Bresenham 算法明显提高。

为进一步提高算法效率, 可以用线段本身的对称性, 利用新算法产生起点一侧的半条线段, 至于终点一侧的半条线段, 则可以看成是以终点为起点线段的生成。起点一侧的线段生在 x 或 y 方向每前进一个坐标单位, 终点一侧的线段生成就在 x 或 y 方向后退一个坐标单位。这种方法在许多文献中都有描述[3,5], 本文就不再详述了。

以例 1中的线段生成为例, Bresenham 算法需要加法和比较各 100次, 而新算法则只需各 27次。若再利用线段对称性, 则只各需要 16次。

4 结　论

该算法已在计算机上得到了实现, 经验证, 其执行效率明显优于传统的 Bresenham 算法。表 1是新算法与 Bresenham 算法分别编程实现后的效率比较, 程序在 Intel Pentium Ⅲ 866MHz, 256MB 内存的计算机上用 VC++ 6.0运行, 设直线的起点都是 (0, 0), 表中所列时间为坐标计算时间, 不包括将像素点显示在屏幕上所需要的时间。

表 1　新算法与 Bresenham 算法的效率比较

Tab. 1　Compare of new algorithm and Bresenham algorithm

终点坐标	运行时间 (μs)	
	Bresenham 算法	新算法
(1000, 1)	15	16
(1000, 10)	15	4
(1000, 100)	16	4
(1000, 1000)	15	14
(10000, 1)	129	139
(10000, 10)	129	20
(10000, 100)	130	6
(10000, 1000)	133	5
(10000, 10000)	128	120

由表 1可见, 若直线的斜率为 1或像素行不超过 3行, 则该算法与 Bresenham 算法效率基本一致, 其他情况下该算法效率都优于 Bresenham 算法, 而且待生成线段包含的像素点越多, 节省的计算量越大。

参考文献 (References)

1　James D Foley. Introduction to Computer Graphics[M]. Beijing: China Machine Press, 2004: 48~56. [James D Foley. 计算机图形学导论 [M]. 北京: 机械工业出版社, 2004: 48~56.]

2　Zheng Hong-zhen. The improvement of Bresenham algorithm [J]. Journal of Image and Graphics, 1999, 4(7): 606~608. [郑宏珍. 改进的 Bresenham 直线生成算法 [J]. 中国图象图形学报, 1999, 4(7): 606~608.]

3　Sun Yan. The parallel algorithm of Bresenham [J]. Computer Engineering and Applications, 2001, 37(21): 136~137. [孙岩. 并行的 Bresenham 直线生成算法 [J]. 计算机工程与应用, 2001, 37(21): 136~137.]

4　Boyer V. Auto-adaptive step straight-line algorithm [J]. Computer Graphics and Applications, 2000, 20(5): 67~69.

5　Lin Li. Bresenham based 4-point line drawing algorithms [J]. Journal of Jinan University (Natural Science), 2003, 24(5): 19~22. [林笠. 基于 Bresenham 算法的四步画直线算法 [J]. 暨南大学学报 (自然科学版), 2003, 24(5): 19~22.]