自适应多步位移码直线绘制算法*

苗兰芳[1,2], 刘新国[1], 彭群生[1], 鲍虎军[1]

[1](浙江大学 CAD&CG 国家重点实验室,浙江 杭州 310027);
[2](浙江师范大学 计算机科学与工程学院,浙江 金华 321004)
E-mail: zjmlf@163.net
http://www.zju.edu.cn

摘要: 直线绘制是光栅图形学中一个最为基本的任务,加速传统直线绘制算法有着重要的实际意义.提出了一种新的直线绘制算法.与传统的直线绘制算法不同的是,该算法将直线直接表达成一串由 0 或 1 组成的位移码,并给出一个直线位移码的快速计算公式;在此基础上,通过对直线位移码周期性的分析,提出了一种新的自适应多步绘制算法.实验和理论分析表明,该算法能够大大减少生成直线的计算量,提高直线的绘制速度.

关 键 词: 图形系统;扫描转换;位移码;绘制;自适应算法
中图法分类号: TP391 文献标识码: A

直线是图形中最基本的元素之一,直线绘制算法的好坏对图形应用系统的效率和质量有着直接而又重要的关系.早在图形学诞生时,人们就开始研究基于随机扫描显示器的直线绘制算法.随着光栅扫描显示器的出现,出现了基于像素的逐点绘制方法[1].最著名的直线绘制方法是 20 世纪 60 年代中期出现的 Bresenham[2] 算法.在 Bresenham 算法中,所有的运算都是整数运算,绘制一个点的运算量是 1 次整数加法运算和 1 次符号判断.Bresenham 非常适合于硬件实现.为了进一步提高直线的绘制速度,后续的研究工作主要集中在如何 1 次生成位于直线上多个像素点[3~7].二步法[3~5]1 次判断生成两个点,该类算法首先将斜率小于 1 的直线分成两类,即小于 1/2 和大于 1/2 类.取二进制链码表示直线上当前采样点对前一采样点的位移,1 表示前一采样点在 X 方向和 Y 方向均前进一个单位(像素)步长,0 表示仅在 X 方向前进一个单位步长.对于斜率小于 1/2 的直线,位移链码中 1 之后必跟着 0,后一步不用判断即可确定,该算法对斜率接近 1/2 的直线效果较好,对趋于平坦的直线效果不够显著.四步法[6]将直线中可能会出现的所有四步行进码(18 种形式)事先作为一个矩阵存起来,再通过判断所画直线的形式来确定四步行进码,因此,该算法是以增加代码的复杂度和存储容量来提高速度的.多步法[7] 则提出了一个更加一般的 8,16,…,2^n 步算法(N-Step),但是该算法代码复杂度和存储容量更高.双向算法[8]则根据直线的对称性,从直线的两端同时生成两个点,从而提高绘制速度.上述多步法的共同特点是:每一次有效的特征判断都是生成固定数目的像素点.本文提出了一种自适应的多步位移直线算法,该算法首先将待绘制的直线直接表达成一串由 0,1 组成的位移码,根据该位移码的组成特点,自适应地确定一次生成的像素数目,从而获得更高的绘制效率,算法也十分简洁.

本文第 1 节给出直线位移码的定义和计算公式.第 2 节介绍一种基于直线位移码的单步绘制算法.第 3 节将进一步讨论直线位移码的一些有用的性质.第 4 节介绍本文提出的自适应的多步快速直线绘制算法.最后是结论和未来工作.

* 收稿日期: 2000-07-23; 修改日期: 2001-01-03
基金项目: 国家自然科学基金资助项目(69823003)

作者简介: 苗兰芳(1963－),女,浙江慈溪人,副教授,主要研究领域为计算机图形学,中文信息处理;刘新国(1972－),男,江西九江人,博士,主要研究领域为计算机图形学,虚拟现实;彭群生(1947－),男,湖南新化人,博士,教授,博士生导师,主要研究领域为真实感图形,虚拟现实,科学计算可视化,计算机辅助设计;鲍虎军(1966－),男,浙江温州人,博士,教授,博士生导师,主要研究领域为计算机动画,虚拟现实.

1 直线位移码

所有的光栅图形输出设备都可以等价地看成是一个二维的像素网格,每一个像素都对应于一个网格点.绘制直线的任务就是确定位于给定的直线上的像素采样点.由于采样的离散性,这是一个逼近的过程.如图 1 所示,其中的直线 L 除了两个端点之外,直线并不经过像素网格点.一般常取位于直线附近的像素来逼近.例如,像素 P 来逼近直线 L 与网格的交点 C.下面,我们先简单叙述一下直线的逼近方法.不失一般性,假设直线的斜率在[0,1]之间.

如图 1 所示,直线 L 的两个端点 A,B 的坐标分别为 (x_a,y_a) , (x_a,y_a) .直线和网格线 $x=x_i=x_a+i$ ($i=0,1,2,...,x_b-x_a$)的交点的 Y 坐标值为

$$Y(x_i)=x_a+\frac{y_b-y_a}{x_b-x_a}(x_i-x_a).$$

像素网格上最靠近交点 $(x_i,Y(x_i))$ 的一个网格点是 (x_i,y_i) ,其中 $y_i=\lfloor Y(x_i)+0.5\rfloor$, $\lfloor R\rfloor$ 表示对 R 作取整运算.对于斜率 m 在[0,1]之间的直线,我们有:

$$y_{i+1}-y_i=\lfloor Y(x_{i+1})+0.5\rfloor-y_i=\lfloor Y(x_i)+m+0.5\rfloor-\lfloor Y(x_i)+0.5\rfloor.$$

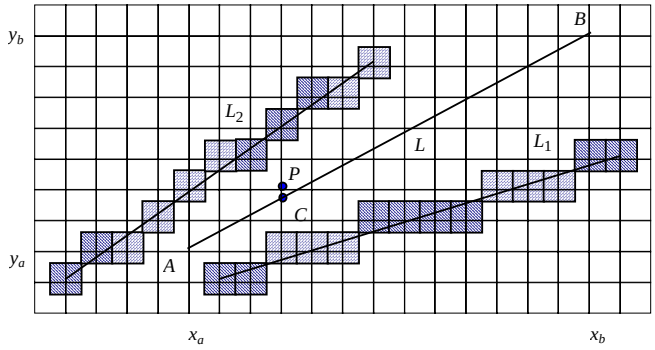

Fig.1 Approximation of lines by pixels (grid points)

图 1 直线的像素逼近(图中网格点为像素中心)

由于 $0\le m\le 1$,故 $0\le y_i-y_{i-1}\le 1$,因此 $y_{i+1}-y_i$ 只能是 0 或者 1.无论是 0 还是 1,直线均从 (x_i,y_i) 沿水平方向上前进一格,我们称 X 轴为驱动轴.当 $y_{i+1}-y_i$ 取值为 1 时,采样点 (x_i,y_i) 在沿水平方向前进一格的同时,还将沿 Y 轴正向前进一格.反过来,直线的离散逼近又可以由这些 $\{y_{i+1}-y_i\}$ 值惟一地确定.因此我们将这个 0,1 序列形象地定义为直线的位移码,称 1 的位置为它的跃阶位置,或跃阶点.

对于整数 $K>0$,很容易计算出直线 $Y=\frac{1}{K}X, X\ge 0$ 的位移码序列为

$$\underbrace{00...0}_{[(K-1)/2]}1\underbrace{00...00}_{K-1}1\underbrace{00...00}_{K-1}100... \quad (1)$$

其中第 1 个 1 前面的 0 的个数为[(K–1)／2],相邻两个 1 之间的 0 的个数为 K–1.显然,这是一个周期为 K 的周期序列,在一个周期中有且仅有一个跃阶点,且第 1 个跃阶位置为 $T=\lfloor (K-1)/2\rfloor$.

引理 **1**(位移码计算公式). 对任意的两个整数 $0<H<K$,记直线 $L_1:Y_1(X)=\frac{1}{K}X, X\ge 0$ 的位移码序列为 $A_1=\{A_1(i)\,|\,i=0,1,2,...\}$,直线 $L_H:Y_H(X)=\frac{H}{K}X, X\ge 0$ 的位移码序列为 $A_H=\{A_H(i)\,|\,i=0,1,2,...\}$,则:

$$A_H(i)=\sum_{s=0}^{H-1}A_1(i\cdot H+s). \quad (2)$$

证明:如图 2 所示, $x_i=i$.设直线 L_1 上第 i 个像素坐标为 $(x_i,y_{1,i})$,直线 L_H 上第 i 个像素坐标为 $(x_i,y_{H,i})$.因为 $Y_H(x_i)=Y_1(x_i\cdot H)$,所以,

$$y_{H,i}=\lfloor Y_H(x_i)+0.5\rfloor=\lfloor Y_1(x_i\cdot H)+0.5\rfloor=y_{1,i\cdot H}$$

因此,

$$A_H(i)=y_{H,i+1}-y_{H,i}=y_{1,(i+1)\cdot H}-y_{1,i\cdot H}=\sum_{t=0}^{(i+1)\cdot H-1}A_1(t)-\sum_{t=0}^{i\cdot H-1}A_1(t)=\sum_{t=i\cdot H}^{(i+1)\cdot H-1}A_1(t)=\sum_{s=0}^{H-1}A_1(i\cdot H+s).$$ □

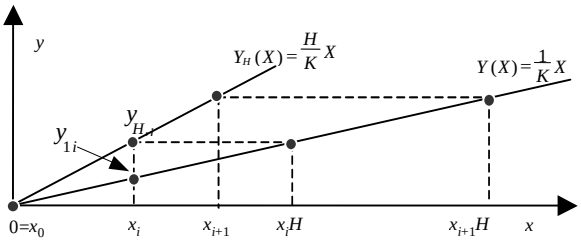

Fig.2 Displacement code calculation

图 2 位移码计算的几何说明

2 单步位移码直线绘制算法

从式(2)我们知道,计算直线 $Y_H(X)$ 位移码序列 A_H 就是对直线 $Y_1(X)$ 位移码序列 A_1 依次进行连续的 H 项求和过程:这 H 个元素中若有“1”,则 A_H 相应的位移码为 1,否则为 0.如果每次进行 H 个元素求和之后,都将 A_1 向左移 H 个位置,那么我们只需要判断移位后的序列 A_1 的第 1 组连续的 H 个元素中是否有 1.设初始时候 A_1 的第 1 个跃阶点为 T,则经过 i 次向左移出 H 个位置以后,第 1 个跃阶点为 $T+nK-iH$,n 为一个整数.因此:

$$A_H(i)=\begin{cases}1, & \text{当 } (T-i\cdot H)\bmod K<H,\\ 0, & \text{否则.}\end{cases} \tag{3}$$

根据这个思想,我们设计了如下的计算直线 $Y_H(X)$ 且 $0<H<K$ 的位移码序列的过程:

1. $T\leftarrow\lceil(K-1)/2\rceil-H$
2. 对 $i=0,1,2,\ldots$,
   2.1. 如果 $T<0$,则 $A_H(i)\leftarrow 1$, $T\leftarrow T+(K-H)$;
   2.2. 如果 $T\geq 0$,则 $A_H(i)=0$, $T\leftarrow T-H$.

计算出位移码以后,直线的绘制过程就变得非常简单,每一次都把 x 值加 1.同时,如果相应的位移码为 1,则将 y 值也加 1.完整的绘制算法如图 3 所示中的算法 1,其中算法输入为斜率在 0~1 之间的直线两端的整数坐标 (x_a,y_a) 和 (x_b,y_b) ,且 $x_a<x_b$.因为每一次我们都只得到一位位移码、绘制一个像素,所以称它为单步位移码直线绘制算法.

Algorithm 1. Single-Pixel line drawing algorithm①

```
K ← x_b − x_a
H ← y_b − y_a
T ← ⌈(K−1)/2⌉ − H
DrawPixel(x_a,y_a)
x ← x_a
y ← y_a
While x<=x_b Do
  If T<0 Then
    y ← y+1
    T ← T+(K−H)
  Else
    T ← T−H
  End If
  DrawPixel(x,y)
  x ← x+1
End While
```

Algorithm 2. Line periodically drawing algorithm②

```
……
If T=T_0 Then
  N ← x − x_a  N is periodical length
  While x<x_b
    For i=0 To N
      x ← x+1
      y ← y+A_H(i)
      DrawPixel(x,y)
    End For
  End While
End If
……
```

Algorithm 3. Adaptive multi-pixel line drawing algotithm③

```
K ← x_b − x_a
H ← y_b − y_a
T ← [(K−1)/2] − H
Divide(H,K–H,N,T_Change)
DrawPixel(x_a,y_a)
x ← x_a
y ← y_a
While x<=x_b Do
  If T<0 Then
    y ← y+1
    x_end ← x+N
    While x<x_end Do
      DrawPixel(x,y)
      x ← x+1
    End While
    T ← T+T_Change
  Else
    T ← T−H
    DrawPixel(x,y)
  End If
  x ← x+1
End While
```

①算法 1.单步位移码直线算法,②算法 2.直线的周期绘制算法,③算法 3.自适应多步位移码直线算法.

Fig.3 Pseudo code algorithms

图 3 伪代码算法描述

单步位移码直线绘制的算法分析:(1) 所有的运算都是整数的运算,包括加法、减法、除 2(可以用向右的一次移位操作代替)以及整数的符号判别;(2) 每绘制一个像素的运算量为一次整数加/减法和一次符号位判断,与 Bresenham 直线算法具有相同的运算量.不例外地,本文也只详细地讨论了斜率在 0~1 之间的直线,即:端点坐标满足 $0 < x_b - x_a \le y_b - y_a$ 的直线.对其他的各种情况我们可以作类似的分析,或根据对称性进行变换处理,这里不再详述.

3 位移码的周期性研究

上面曾经提过位移码 A_1 是一个周期为 K 周期序列,所以位移码 A_H 也是一个周期序列.根据抽象代数中的循环群理论,我们可以知道位移码 A_H 的周期为

$$\frac{K}{(H,K)},$$

(H,K) 表示 H 和 K 的最大公约数.

利用周期性我们知道,在图 3 的算法 1 中当且仅当在绘制完成第 1 个周期的像素后,T 的值又回到最开始的初值.因此在图 3 的算法 1 中,若保存 T 的初值,记录所得到的位移码,并且在 DrawPixel(x,y)的后面增加如图 3 所示的算法 2 中的一段周期绘制代码,那么我们在完成一个周期后就能够绘制整条直线.

上面周期绘制算法的效率与位移码的周期长度、周期次数有关.如果周期长度越短,那么周期次数就越多,算法的效率也越高.反之周期长度越长,那么周期次数就越少,算法的效率也就越低.最坏的情况是,当 $H = y_b - y_a$ 和 $K = x_b - x_a$ 互质的时候,这时,周期长度为 $x_b - x_a$,计算量反而比算法 1 要大.

然而通过观察我们知道,对于所有的直线段,其离散的像素逼近都呈现一定的"准周期性".例如图 1 中的直线 L_1,此时 $H = y_b - y_a = 4$, $K = x_b - x_a = 13$,整个直线段可以划分为 5 份水平的直线段,而且中间 3 段的长度都至少为 3 个像素.如果把直线继续延长,可以发现中间部分的每一段的长度都至少为 3,且至多为 4.再考察图 1 中的直线 L_2,此时 $H = y_b - y_a = 7$, $K = x_b - x_a = 10$,整个直线段可以划分为 4 份倾角为 45° 的直线段,而且中间部分的每一段的长度都至少为 3.上面的观察结果提示我们,可以以一小段一小段的方式对直线进行绘制,从而获得比逐个像素方式更快的绘制速度.

考察直线 L_1 和直线 L_2 的位移码:

直线 L_1: 0 1 0 0 1 0 0 0 1 0 0 1 0

直线 L_2:1 0 1 1 1 0 1 1 0 1

如图 1 所示.我们发现直线 L_1 中间那些水平的直线段对应着位移码中 1 及其后面一串的 0;直线 L_2 中间那些 45° 倾角的直线段对应着位移码中 0 及其后面一串的 1.其实这是一个普遍的规律:

引理 **2**. 设有从 (x_a, y_a) 到 (x_b, y_b) 的直线, $0 < H = y_b - y_a < K = x_b - x_a$.那么在该直线的位移码中,相邻两个 1 之间连续 0 的数目 w 为 $\left\lfloor \frac{K-H}{H} \right\rfloor$,或 $\lceil K/H \rceil - 1$.

证明:设 A_H 中 $A_H(i)$ 和 $A_H(i+n)$ 是两个相邻的 1,则 $w = n - 1$.设对 A_1 进行 i 次向左移 H 位后第 1 跃阶点为 T,则 $0 \le T < H$,且第 2 跃阶点为 $T + K$.所以对 A_1 再进行 n 次向左移 H 位后,原来 $T + K$ 位置上的 1 的新位置 $T + K - n \cdot H$ 将在 $[0, H)$ 之间.因此,

$$0 \le T + K - n \cdot H < H, \text{即 } T + K - H < n \cdot H \le T + K.$$

又因为 $0 \le T < H$,所以, $\lceil (K-H)/H \rceil < n \le \lceil (K+H)/H \rceil$.这两个相邻的 1 中间 0 的个数为 $w = n - 1$,因此 $\lceil (K-H)/H \rceil \le w \le \lceil K/H \rceil = \lceil (K-H)/H \rceil + 1$.故 w 为 $\lceil (K-H)/H \rceil$,或 $\lceil K/H \rceil - 1$. □

同理可证下面的引理.

引理 **3**. 设有从 (x_a, y_a) 到 (x_b, y_b) 的直线, $0 < H = y_b - y_a < K = x_b - x_a$.那么在该直线的位移码中,相邻两个 0 之间连续 1 的数目 w 为 $\lceil H/(K-H) \rceil$ 或 $\lceil K/(K-H) \rceil - 1$.

4 自适应的多步位移码直线绘制算法

因为单步的直线绘制算法在计算量几乎已经达到了最省的程度,如 Bresenham 算法和第 2 节中的单步位移码直线绘制算法.要获得更高的绘制效率,我们必须朝着一次判断生成多个像素的方向努力.利用上节中的两个引理我们可以很方便地把第 2 节中的单步位移码直线绘制算法改造为一个多步绘制算法.而且这个多步算法的步长可以自动地进行调整以适应不同斜率的直线.

考虑一条从 (x_a,y_a) 到 (x_b,y_b) 的直线,令 $H=y_b-y_a, K=x_b-x_a$, 且 $0<H<K$.如果 $H<K-H$,即直线的斜率小于 0.5,根据引理 2 我们知道,直线位移码中相邻的两个 1 之间 0 的个数至少有 $\lfloor (K-H)/H \rfloor > 1$ 个.以下用 M 表示 $\lfloor (K-H)/H \rfloor$.也就是说,每一次当我们得到 1 个等于 1 的位移码的时候,我们接下来所求出来的 M 个位移码肯定是 0.对算法 1 稍加修改:即在求得一个位移码 1 之后便处理 M 个位移码 0,于是实现了一个自适应的多步算法,如图 3 所示中的算法 3,其中子程序 Divide(H,K–H,N,T_{Change})是求 H 除 $(K-H)$ 的商和余数分别赋给 N 和 T_{Change}, (x_a,y_a) 以及 (x_b,y_b) 为一斜率在(0,0.5)之间的一条直线的两端点整数坐标,且 $x_a<x_b$.

在算法 3 中,(1) 所有的计算都是整数的运算;(2) 多步法比单步法所节省的计算量在于位移码中的 0 进行成批处理.在单步法中,一个 0 生成一个像素所消耗的计算量为 1 次整数加/减法和 1 次符号位判断,而在多步法中,在每处理完一个位移码 1 以后,可连续生成 M 个像素,而消耗的计算量仅为 1 次整数加/减法.故减少的计算量为 $\left\lfloor \frac{K-H}{H} \right\rfloor \cdot H - H = (K-H)-(K-H)\%H - H$ 整数加/减法次以及 $M \cdot H = (K-H)-(K-H)\%H$ 次符号位判断.单步法中一个循环的计算量为 K 次整数加/减法和 K 次符号位判断,而多步法同样生成 K 个像素所需计算量为 $2H+(K-H)\%H = 2H+K\%H$ 次整数加减法和 $H+(K-H)\%H = H+K\%H$ 次符号位判断.因为 $H<K/2$,所以多步法的效率比单步法的效率要高,而且 K/H 的值越大,多步算法效率越高.

如果 $H>K-H$,即直线的斜率大于 0.5,根据引理 3 我们知道,直线位移码中相邻的两个 0 之间 1 的个数至少有 $\lfloor H/(K-H) \rfloor > 1$ 个.以下用 N 表示 $\lfloor H/(K-H) \rfloor$.也就是说,每一次当我们得到一个等于 0 的位移码的时候,我们接下来所求出来的 N 个位移码肯定是 1.与算法 3 类似,我们对算法 1 稍加修改,即在当我们计算出一个位移码 0 以后,接着绘制 N 个 1 所代表的像素,于是实现了处理斜率在(0.5,1)之间的直线的一个自适应的多步算法,这里不再详述具体代码.同样,我们可以得到这段多步法代码的计算量为 $2(K-H)+K\%(K-H)$ 次整数加减法和 $K-H+K\%(K-H)$ 次符号位判断.因为 $(K-H)<K/2$,此时多步法的效率比单步法的效率要高,而且 $K/(K-H)$ 的值越大,多步算法效率越高.对于其他斜率的直线可以进行类似的处理.

5 结论和未来工作

基于直线的位移码,我们已经介绍了一种新的快速直线绘制算法.该算法根据直线的斜率自适应地选择一个步长,以一次判断生成多个采样点的方式对直线进行绘制.表 1 是对 Bresenham 算法和本文中单步及多步法计算量的理论分析结果,表明该算法能够极大地减少计算量.注意到,在自适应的直线绘制算法中有一次除法运算,对于很短的线段可能得不偿失.在实际应用中可加以预先判断:如果线段太短,则使用单步法进行绘制.

我们看到,直线的位移码在本文算法的推导和实现中起了极其重要的作用.目前只有少数几条简单性质被挖掘和利用,未来工作的一个方向是继续研究直线位移码的其他一些性质,例如,如何精确确定相邻 1 之间 0 的个数以及相邻 0 之间 1 的个数等等,从而实现更加快速的绘制算法.

Table 1 Comparison of the computation of line drawing algorithms
表 1 直线绘制算法运算量比较分析

		Addition/subtraction method①	Multiplication/division method②	Sign bit judgement③
Bresenham		K	0	K
Single-pixel displacement code④		K	0	K
Multi-Pixel displacement code⑤	$H/K<0.5$	$2H+K\%H$	1	$H+K\%H$
	$H/K>0.5$	$2(K-H)+K\%(K-H)$	1	$(K-H)+K\%(K-H)$

①加减法,②乘除法,③符号位判断,④单步位移码,⑤多步位移码.

References:

[1] Tang, Rong-xi, Wang, Jia-ye, Peng, Qun-sheng. Computer Graphics Course. Beijing: Science Press. 1990. 60~65 (in Chinese).

[2] Bresenham, J.E. Algorithm for computer control of a digital plotter. IBM Systems Journal, 1965,4(1):25~30.

[3] Rokne, J., Rao, Y. Double-Step incremental linear interpolation. ACM Transactions on Graphics, 1992,11(2):183~192.

[4] Liu Yong-kui. Generating and clipping of line and curve [Ph.D. Thesis]. Hangzhou: Zhejiang University, 1997 (in Chinese).

[5] Graham, P., Sitharama, Lyengar, S. Double-And triple-step incremental linear interpolation. IEEE Computer Graphics and Applications, 1994,20(5):49~53.

[6] Bao, P., Rokne, J. Quadruple-Step line generation. Computers & Graphics, 1989,13(4):461~469.

[7] Graeme, W. Australia, G.L. *N*-Step incremental straight-line algorithms. IEEE Computer Graphics and Applications, 1994,20(5):66~72.

[8] Yao, C., Rokne, J.G. Bi-Directional incremental linear interpolation. Computer & Graphics, 1996,20(2):295~305.

附中文参考文献:

[1] 唐荣锡,汪嘉业,彭群生.计算机图形学教程.北京:科学出版社,1990.60~65.

[4] 刘勇奎.直线和曲线的生成与裁剪[博士学位论文].杭州:浙江大学,1997.

An Adaptive Multi-Pixel Line Drawing Algorithm Based on Displacement Code*

MIAO Lan-fang[1,2], LIU Xin-guo[1], PENG Qun-sheng[1], BAO Hu-jun[1]

[1](*State Key Laboratory of CAD&CG, Zhejiang University, Hangzhou* 310027, *China*);

[2](*College of Computer Science and Engineering, Zhejiang Normal University, Jinhua* 321004, *China*)

E-mail: zjmlf@163.net

http://www.zju.edu.cn

Abstract: Line drawing is a fundamental task for raster graphics. Any acceleration of the traditional line drawing process is of great significance. In this paper, a new line drawing algorithm is presented. The algorithm differs from the traditional line drawing algorithm in that a line is directly represented as a series of displacement codes consisting of 0's and 1's, which can be easily determined from a formula. Based upon periodic characteristics of the codes, a new adaptive multi-pixel line drawing algorithm is put forward. Both experimental results and analysis show that this algorithm greatly reduces the computation of line drawing and accelerates the line drawing process.

Key words: graphics system; scan conversion; displacement code; rendering; adaptive algorithm

* Received July 23, 2000; accepted January 3, 2001
Supported by the National Natural Science Foundation of China under Grant No.69823003