文章编号:1671-6833(2010)05-0125-04

基于光栅显示器的反走样图元生成算法研究

李晓楠,王文义,王春霞

(中原工学院 并行处理技术研究所,河南 郑州 450007)

摘 要:从光栅设备产生图形走样的根本原因出发,在Bresenham算法基础上,提出了一种改进的直线和圆的反走样图元生成算法.算法首先根据与理想曲线相邻的两个像素点到曲线的距离确定其灰度值,使像素灰度值与其到理想曲线的距离成反比,并存储其左右共三列像素对的灰度值;然后使用高斯滤波掩模对像素点的灰度进行平滑处理,生成反走样图元.该算法已在Visual C++ 6.0环境下实现,具有较好的显示效果,相对于全屏图像处理具有较低的空间和时间复杂度.

关键词:光栅显示;计算机图形学;反走样;高斯滤波

中图分类号:TP391 **文献标识码**:A

0 引言

在光栅设备上绘制与显示曲线,是计算机图形学的基本操作之一.光栅显示器是由一系列离散的像素点组成,当用这些像素点来显示连续图形时,往往会产生锯齿状等失真现象,这种现象称为走样[1],用来消除或减少走样的技术称为反走样.常用的反走样技术有几种,如提高显示分辨率,全屏图像处理和图元生成时反走样[2]等.对于第一种,由于受到硬件和价格的限制,一般不宜采用[1];对全屏图像处理,因为需要先做边缘检测后才能做平滑处理,时空开销比较大[3];而采用图元生成时反走样技术较之前两种,能够以较小的代价,实现对图元的高效处理.

笔者在Bresenham算法基础上,结合两像素反走样思想[4]和高斯滤波法[5],针对直线和圆,提出了一种图元生成时的反走样算法,取得了较好的显示效果.

1 算法原理

1.1 Bresenham算法

Bresenham算法是被广泛应用于直线扫描的转换算法之一.该算法主要通过点亮距理想直线最近的像素点来绘制直线[6],如图1所示.

为不失一般性,考虑第一象限中斜率$0 \leqslant k \leqslant 1$的情况.如图1所示,根据Bresenham算法原理,选取x轴为主位移方向,理想直线段(图中斜线)按照光栅步长被采样后与上下相邻像素中心连线产生交点,图中实心点距离各自y方向扫描线上交点的距离比较近,因此实心像素点将作为交点被点亮.由于图中3个实心像素点既没有完全在理想直线上,也没有在同一条扫描线上,因此产生了锯齿.

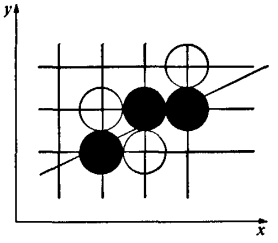

图1 Bresenham算法示意图

Fig. 1 Diagram of Bresenham algorithm

通常人眼对曲线的动力学特性比较敏感,而对于它在空间的绝对位置表现惰性.这是因为人眼很难观察到在巨大背景下物体相对微小的移动,却很容易捕捉到物体微小的变形,即令人不愉快的图形走样[4].目前普遍的处理方法是基于人眼对某一区域颜色的识别是取决于该区域颜色的整体效果而采用空间混色原理来对直线进行修正[7].

收稿日期:2010-03-10;修订日期:2010-05-30

基金项目:河南省基础与前沿技术研究项目(082300410300)

作者简介:李晓楠(1983-),女,河南淅川人,中原工学院助教,硕士,研究方向为图形图像的并行处理技术,E-mail:lxnbupt@gmail.com

1.2 两像素方案

由于理想直线上下相邻像素对直线都有贡献,因此可以同时点亮两个像素点,使其具有不同的灰度,用其混合效果来代表理想直线.现在的设备大都具有多个灰度级,所以实现起来较容易.

设直线为 $y=f(x)$,预期灰度为 G_0,扫描线 $x=i$ 时,理想直线上的采样点坐标为 $(i,f(i))$,若用 $G[i,f(i)]$ 代表其灰度,则 $G[i,f(i)]=G_0$.理想采样点 $(i,f(i))$ 的灰度可以通过设置上下相邻像素的灰度来仿真.

$$\begin{cases} G[i,\lceil f(i)\rceil]=G_0\times(f(i)-\lfloor f(i)\rfloor) \\ G[i,\lfloor f(i)\rfloor]=G_0\times(\lceil f(i)\rceil-f(i)) \end{cases} \tag{1}$$

式中:$f(i)-\lfloor f(i)\rfloor$ 为下邻像素点到理想采样点 $(i,f(i))$ 的距离;$\lceil f(i)\rceil-f(i)$ 为上邻像素点到理想采样点 $(i,f(i))$ 的距离,亮度与其到理想采样点的距离成反比.因为 $f(i)-\lfloor f(i)\rfloor+\lceil f(i)\rceil-f(i)=1$,所以 $G[i,\lceil f(i)\rceil]+G[i,\lfloor f(i)\rfloor]=G_0$.因此,式(1)的整体效果是一个亮度为 G_0 的区域,这个区域的焦点坐标为 $(i,f(i))$,即理想直线 $y=f(x)$ 上的点.相当于把每个像素对都绘制在理想直线 $y=f(x)$ 的两侧,并且这些点的亮度都与它们到该直线的距离成反比.像素点离直线越近就越亮,那么当眼睛综合了这些像素点的效果之后,其整体的效果就等同于理想直线.

现在的问题是,由于在某些位置灰度被均匀的分布到两个像素点上,而那些位于理想直线上的点,或者离理想直线很近的点就获得了很高的灰度,甚至是全部的灰度,从而导致亮度跳变,致使走样依然很明显.

1.3 低通滤波

低通滤波通常被用来消除噪声,也就是突变的高频成分.因此,可以用低通滤波去除直线中的亮度跳变,以平滑直线.

高斯滤波是一类根据高斯函数的形状选择滤波掩模的平滑滤波方法,通过滤波掩模实现邻域运算.即用一个有奇数点的滑动窗口在图像上滑动,将窗口中心点对应的图像像素点的灰度值用窗口内的各个点的灰度值的加权平均值代替[1].

$$h(i,j)=\frac{1}{\sqrt{2\pi\sigma}}e^{\frac{i^2+j^2}{2\sigma^2}} \tag{2}$$

由式(2)可知,二维高斯函数具有旋转对称性,即滤波器在各个方向上的平滑程度是相同的.另外高斯函数又是单值函数,这表明高斯滤波器用像素邻域灰度的加权值来代替该点的灰度值时,每一邻域像素点的权值对该点与中心点的距离是单调增减的.空域高斯平滑滤波可以表示为:

$$g(i,j)=\sum_{m=-K}^{K}\sum_{n=-L}^{L}W(m,n)f(i+m,j+n) \tag{3}$$

其中,$g(i,j)$ 为滤波后的灰度;f 为滤波前的灰度;$W(m,n)$ 为高斯滤波器掩模,窗口大小为 $(2K+1)\times(2L+1)$.考虑到实时性,选取较小的滤波窗口,即 $K=1$、$L=1$ 时如图2所示的八邻域高斯滤波掩模[8],可以看出,距离中心越近的点,加权系数越大.

$$\frac{1}{16}\times\begin{array}{|c|c|c|}\hline 1&2&1\\\hline 2&4&2\\\hline 1&2&1\\\hline\end{array}$$

图2 3×3高斯滤波掩模

Fig.2 3×3 Gaussian filter mask

2 改进反走样算法描述

图元生成过程是:在Bresenham算法的基础上,用两像素算法计算出某一列两个像素的灰度值之后,利用其左右两列两对像素的灰度值,通过高斯滤波掩模进行低通滤波,滤波后将产生该列4个像素的亮度值,并绘制该列像素点;依次进行下去,直到整条直线绘制完毕.

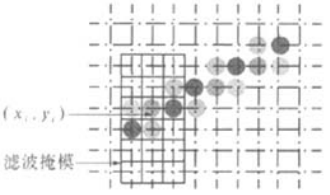

图3 改进的反走样算法示意图

Fig.3 Diagram of the new anti-aliasing algorithm

改进的反走样算法如图3所示,以反走样直线的生成为例,仅考虑第一象限斜率 $k\leqslant 1$ 的情况(其他位置的直线经过转换后同样适用该算法).已知起点 (x_0,y_0) 和终点 (x_e,y_e),$0\leqslant k=\frac{y_e-y_0}{x_e-x_0}\leqslant 1$,用一个6行3列的二维数组 Gray[0:5][0:2]存储高斯滤波前用两像素反走样计算的3对像素点的灰度值.用8位存储像素的灰度值,则有256个灰度级,从0(黑)到255(白).算法过程描述如下.

Step1:用255初始化 Gray 数组.

Step2:特殊处理端点 (x_0,y_0),根据两像素反走样思想计算 $(x_1,\lfloor y_1\rfloor)$,$(x_1,\lceil y_1\rceil)$,根据其与 (x_0,y_0) 的上下相对位置将其存入 Gray 数组第2列的相应位置;分别以 Gray 数组第1列的第1至4行的像素为中心点,利用图2所示的高斯滤波

掩模计算其灰度值,并绘制 x_0 列像素点.

Step3:将 Gray 数组右移一列,判断 x_i 是否为终端,若是,则转向 Step4 执行,否则根据两像素反走样思想计算出 $(x_{i+1}, \lfloor y_{i+1} \rfloor)$,$(x_{i+1}, \lceil y_{i+1} \rceil)$,根据其与 (x_i, y_i) 的上下相对位置将其存入 Gray 数组第 2 列的相应位置;分别以 Gray 数组第 1 列的第 1 至 4 行的像素为中心点,利用图 2 计算其灰度值,并绘制第 x_i 列像素点.

Step4:特殊处理端点 (x_e, y_e),此时 $(x_e, \lfloor y_e \rfloor)$,$(x_e, \lceil y_e \rceil)$ 被存在 Gray 数组第 1 列,即滤波掩模的中心点,将 Gray 数组第 2 列置为 255,同样用高斯滤波掩模计算 Gray 数组第 1 列的第 1 至 4 行的灰度值,并绘制 x_e 像素点.

3　程序实例与实验结果分析

3.1　程序实例

算法的 VC ++ 实现:

```
Anti_Line(CDC * pDC, double x1, double y1,
double x2, double y2)
{ flag=0; // 掩模上下移动的标记
  Gray[i][j]=255; //i 从 0 到 5,j 从 0 到 2
  计算斜率 grad=(x2-x1)/(y2-y1);
  yf=y1+grad;
  tempy1=int(yf); //取 yf 的整数部分
  //计算理想直线上下相邻像素点灰度值
  gray1=invfrac(yf); // 1-frac(yf)
  gray2=frac(yf); //取 yf 的小数部分
  c1=byte(gray1*255);
  c2=byte(gray2*255);
  Gray[2][2]=c2;
  Gray[3][2]=c1;
  for(i=1;i<5;i++) //计算并绘制起始点
  { //用高斯掩模对像素点灰度进行平滑
    int temp=(Gray[i-1][0]+(Gray[i-1]
    [1]<<1)+Gray[i-1][2]+(Gray[i][0]
    <<1)+(Gray[i][1]<<2)+(Gray[i][2]
    <<1)+Gray[i+1][0]+(Gray[i+1][1]
    <<1)+Gray[i+1][2])>>4;pDC->Set-
    Pixel(x1,int(yf)+i-2,RGB(temp,temp,
    temp));
  }
  for(x=x1+2;x<=x2;x++)
  { yf=yf+grad; //计算下一个标准的 y 值
    if(flag==1) //上移掩模,第 0 行置为白色
    { for(i=5;i>0;i--)
      {
        Gray[i][0]=Gray[i-1][0];
        Gray[i][1]=Gray[i-1][1];
        Gray[i][2]=Gray[i-1][2];
      }
      for(i=0;i<3;i++) Gray[0][i]=255;
    }
    else if(flag==-1)
    { //下移掩模,第 5 行置为白色 }
    右移掩模;
    if(x==ix2) //若是终结点
    {c1=255; c2=255; }
    else
    {//计算理想直线上下相邻像素点灰度值}
    if(int(yf)<tempy1) //给第 2 列赋值
    { //第 2 列其他行为 255
      Gray[1][2]=c2;Gray[2][2]=c1;flag=1;
    }
    else if(int(yf)==tempy1) //给第 2 列赋值
    { //将 2,3 行分别赋 c2,c1,其它行置 255
      flag=0; }
    else //给第 2 列赋值
    { //将 3,4 行分别赋 c2,c1,其它行置 255
      flag=-1; }
    for(i=1;i<5;i++)//画出中间一列 4 个点
    { //用高斯掩模对像素点灰度进行平滑
      //绘制像素点 }
    tempy1=int(yf);
  }
}
```

3.2　实验结果分析

图 4 是分别用 4 种方法绘制的 4 条直线.图中,a 线条是使用 Bresenham 算法绘制的直线,没有反走样处理,直线的锯齿形走样现象很明显;b 线条是用两像素反走样算法绘制的直线,保持了直线的动力学特性,但是亮度跳变明显;c 线条是采用高斯滤波处理 Bresenham 算法绘制的直线,各像素的亮度基本一致,但走样现象依然很明显;d 线条是用笔者提出的反走样算法绘制的直线,直线的形状及走向保持一致,像素亮度跳变减小,边界平滑,基本消除锯齿现象,达到较好的效果.

笔者根据上述思想还实现了反走样圆的生成算法,显示效果如图 5(b)表盘所示,5(a)表盘使用 Bresenham 算法绘制.

在全屏图像处理中做反走样,通常需要存储

整屏图像的灰度值,还需要逐个计算相邻像素的灰度值之差,然后才能对差值达到阈值的像素进行低通滤波[9].我们可以用空间复杂度(存储占有量)和时间复杂度(计算量)两个指标来衡量新老算法的效果:对于常用的屏幕显示像素数1 024×768,老算法需要存储1 024×768个像素的位置和灰度值,并进行1 024×767次减法运算;而笔者提出的反走样图元生成算法仅需要存储图元上相邻的3×4=12个像素的位置并计算其灰度值即可,所以新算法的时空复杂度都被大大降低,从而明显的提高了效率.

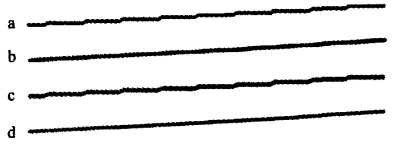

图4 直线显示效果

Fig.4 Display results of lines

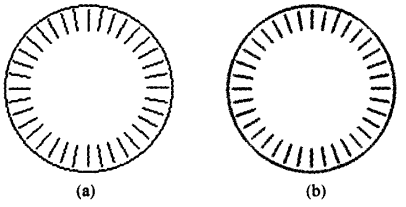

图5 表盘显示效果

Fig.5 Display results of dials

4 结语

笔者提出的改进反走样算法,利用两像素思想保持了曲线的动力学特性,使曲线亮度的整体效果等价于真实曲线;使用高斯滤波实现边界亮度平滑,同时避免了对整幅图形的搜索;采用增量计算和对称性加快了图形生成的速度.对算法进行了改进,用位移运算替代乘法和除法运算,拆除循环以充分利用GPU的流处理器运算能力,使该算法很容易被改编成能够在基于CPU+GPU的CUDA架构下运行的并行程序.

参考文献

[1] WALLER M D, EWINS J P, WHITE M et al. Efficient coverage mask generation for antialiasing[J]. IEEE Computer Graphics and Applications, 2000, 20(6): 86-93.

[2] 李震霄,何援军.任意宽度直线的绘制与反走样[J].武汉大学学报:工学版,2006,36(4):130-133.

[3] 李代强,李成贵.基于FPGA的座舱仪表图形反走样技术研究[J].计算机工程与应用,2006,42(2):93-95.

[4] WU X. An efficient antialiasing technique [J]. Computer Graphics,1991,25(4):143-152.

[5] LIU N, JIN H Z, ROCKWOOD A P. Antialiasing by gaussian integration[J]. IEEE Computer Graphics and Applications, 1996,16(3):58-63.

[6] 姜丹丹,李成贵.全罗盘画面中的刻度线反走样技术[J].液晶与显示,2009,24(1):130-134.

[7] 孔令德.计算机图形学基础教程(Visual C++版)[M].北京:清华大学出版社,2008:56-57.

[8] GONZALEZ R C, WOODS R E.数字图像处理[M].2版.阮秋琦,译.北京:电子工业出版社,2007:93-96.

[9] 陈玉梅,余洪山,贺攀峰.一种分层并行迭代式链码跟踪直线提取算法[J].郑州大学学报:工学版,2006,27(2):94-97.

Anti-aliasing Algorithm Study of Graphics Primitives based on Raster Display

LI Xiao-nan, WANG Wen-yi, WANG Chun-xia

(Institute of Parallel Processing Technology, Zhongyuan Institute of Technology, Zhengzhou 450007, China)

Abstract: According to the study about primary reason of aliasing, this paper presents a renewed anti-aliasing algorithm of lines and circles based on the Bresenham algorithm. The algorithm computes grays of the two pixels nearby the perfect curve, making its gray inversely proportional to its distance between these pixels to the curve, stores the adjacent three columns of pixels, and then makes low-pass filtering on the primitives with Gaussian filter mask. This algorithm has low space and time complexity compared with full-screen image processing, and has been implemented in Visual C++ 6.0 with better show effect.

Key words: raster display; computer graphics; anti-aliasing; Gaussian filtering