DOI:10.19651/j.cnki.emt.1902682

一种基于改进 Bresenham 算法的三角形光栅化技术

张加林 阮成肖
(江苏自动化研究所 连云港 222061)

摘 要：三角形光栅化是图形处理器必不可少的一环，为了提高三角形光栅化的效率，降低 GPU 硬件设计成本，结合 Bresenham 算法和基于加法的除法器两者各自的特点，提出了一种改进的 Bresenham 算法，并将其应用到三角形光栅化中。该算法通过软件仿真验证，并在 FPGA 的硬件测试平台中进行测试对比验证。结果表明，改进 Bresenham 算法能够较好的实现三角形光栅化，并且比传统 Bresenham 算法占用更少的硬件资源，相应降低了 GPU 硬件设计单位成本，变相提高了 GPU 中三角形光栅化效率。

关键词：图形处理器；三角形；Bresenham；边函数；光栅化
中图分类号：TN47 **文献标识码**：A **国家标准学科分类代码**：510.40

A triangle rasterization based on improved Bresenham algorithm

Zhang Jialin Ruan Chengxiao
(Jiangsu Automation Research Institute, Lianyungang 222061, China)

Abstract: Triangular rasterization is an indispensable part of the graphics processor. In order to improve the efficiency of triangle rasterization and reduce the hardware design cost of GPU. This paper combines the characteristics of Bresenham algorithm and adder-based divider to propose an improved Bresenham algorithm. And the algorithm is applied to triangle rasterization. The algorithm is verified by software simulation, and tested in the hardware test platform of FPGA. The results show that the improved Bresenham algorithm can achieve triangle rasterization better and consumes less hardware resources than the traditional Bresenham algorithm, which reduces the unit cost of GPU hardware design and improves the efficiency of triangle rasterization in GPU.

Keywords: graphic processing unit; triangle; Bresenham; edge function; rasterization

0 引 言

图形处理器(graphic processing unit,GPU)从出现到现在，硬件架构已经有过几次的改革，但是图元的光栅化思想在图形管线中都是必不可少的一环[1]，光栅化是将基本图元(点、线段、三角形)转化成一个个像素的过程[2]，图元的光栅化的质量与效率直接影响整个 GPU 管线的性能。而在 GPU 的所有图元对象中，三角形图元是最基本最重要的图元，也是组成任何其它更为复杂二维或三维对象的基本图元。

三角形光栅化最重要的指标是效率[3-5]，即 GPU 在单位时间内能够处理多少个三角形图元。针对不断提升的对 GPU 处理性能的需求，高性能的 GPU 通常会集成几十至几百个并行的光栅化模块来提升性能。单纯依靠增加模块数量来提升光栅化效率的方式会增加芯片的规模和复杂度，增加设计成本，应该在扩展光栅化模块数量的基础上提升单个光栅化模块的效率。

早期的一种三角形光栅化方法是采用数字微分分析算法(digital differential analyzer,DDA)算法[6-7]，遍历三角形的各个边，从而将三角形分割成一个个水平跨度，但是需要用到浮点运算，效率较低。针对 DDA 算法的浮点运算的缺点，出现了 Bresenham[8-10]线段光栅化算法，算法最大的优点就是不需要进行浮点数运算，已经成为目前实际应用中硬件光栅化标准算法。

根据判断三角形像素点位置关系的方法，出现了边函数的算法[11-13]，通过直线方程判断点是否在三角形内部，从而得到整个三角形区域。该算法是一种有利于硬件实现的算法方式，按照不同的扫描方式，该算法又分为很多的种类。

本文经过对现有的光栅化算法的研究，提出一种基于改进 Bresenham 算法的三角形光栅化技术。算法中结合 Bresenham 算法对于线段光栅化的高效的特点，以及边函

收稿日期:2019-02

数对于三角形像素点位置判断的优势[14-15],快速高效地得到三角形的三条边的像素点的坐标,然后根据三角形内部点位于三角形同一水平跨度两点之间,得到所有三角形的内部坐标。

该算法既能够利用硬件以实现的 Bresenham 算法,又不需要遍历三角形内部所有的点,减少了计算的点的个数,从而大大提高了三角形光栅化的效率。

1 改进 Bresenham 算法

本文根据 Bresenham 算法整数运算的步进式特点,以及以牛顿迭代法为原理的基于加法的除法器的硬件易实现性特点,实现改进 Bresenham 算法,将除法器的整数运算结果运用到 Bresenham 算法的步进运算中,较大地提高了线段光栅化的效率和准确度。然后结合边方程判断三角形的边与像素点的位置关系,得到三角形内部所有的像素点。

1.1 Bresenham 算法

Bresenham 算法采用了整数运算,节约了大量硬件资源,有效地避免了浮点运算。算法分为 2 个步骤:1)根据直线的斜率选择线的光栅化基坐标轴(计算过程要求斜率小于 1),2)根据直线斜率逐一计算并选择需要填充的像素点。

Bresenham 算法采用单位步进,意即:当起始像素点为(x_i,y_i)时,下一点的横坐标为 x_i+1,决定下一点的 y 坐标值到底是 y_i+1,还是保持不变为 y_i,该算法是通过两个待取 y 值分别到线段的距离大小来判定的。如图 1 所示,假设线段上方的像素到线段的距离是 d_1,下方到线段的距离是 d_2,则根据这两者的差值 d_2-d_1 的符号来决定下一像素的 y 值。如果差值大于 0,说明线段上方的像素点更加靠近直线段,下一点的 y 值为 y_i+1;若差值小于 0,说明线段下方的像素点更加靠近像素点,这时的 y 值保持不变。

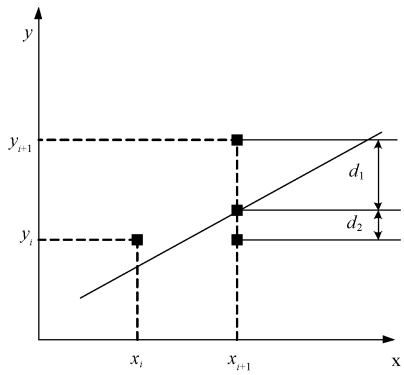

图 1 Bresenham 算法的判定变量

两像素位点到直线的距离分别为:

$$d_1=(y_i+1)-y=y_i+1-k(x_i+1)-b \tag{1}$$

$$d_2=y-y_i=k(x_i+1)+b-y_i \tag{2}$$

要比较这两个距离的大小,从而可以确定最佳的 y 值,距离差为:

$$d_2-d_1=2k(x_i+1)+2b-2y_i-1 \tag{3}$$

引入决定参数 p_i,定义为:

$$p_i=\Delta x(d_2-d_1)=2\Delta y\times x_i-2\Delta x\times y_i+2\Delta y+2\Delta x\times b-\Delta x \tag{4}$$

式中:Δx 与 Δy 是线段两端点的水平和垂直的偏移量。因为 Δx 是大于 0 的,所以 p_i 的符号也就与距离差 d_2-d_1 的符号保持一致,又由于 $2\Delta y+2\Delta x\times b-\Delta x$ 为常量,并且与像素的位置无关,在计算 p_i 会被抵消,所以令它为 c。

利用递推的性质,可以得到下一个决定参数值。

$$p_{i+1}=2\Delta y\times x_{i+1}-2\Delta x\times y_{i+1}+c \tag{5}$$

$$p_{i+1}-p_i=2\Delta y(x_{i+1}-x_i)-2\Delta x(y_{i+1}-y_i) \tag{6}$$

因为是单位步进,所以 $x_{i+1}-x_i=1$,而 $y_{i+1}-y_i$ 是取值为 1 还是为 0,取决于 p_i 的符号。若 p_i 为正,那么说明上一个像素点的距离差为正,上面的像素值靠近绘制直线,$y_{i+1}=y_i+1$,即 $y_{i+1}-y_i$ 取值为 1;若 p_i 为负,说明下面的像素点靠近待绘制直线,$y_{i+1}=y_i$,即 $y_{i+1}-y_i$ 取值为 0。

1.2 基于加法的除法器

本文设计的基于加法的除法器是依据三角形光栅化的实际应用环境。假设需要得到 b/a 的值,其中 $b\geqslant 0$,$b\leqslant 2^m$,$a>0$(对于 $a=0$ 的特殊情况,单独计算),那么,令 $b=a\times q+r$,其中 a,b,r,q 均属于整型,q 就是我们需要得到的商,那么因为 a 是整型,且 $b\leqslant 2^m$,那么 $q\leqslant 2^m$,r 是余数。

所以,设定 q 为 $m+1$ 位的二进制的整型,即 $k_mk_{m-1}\cdots k_1k_0$,那么

$$q=k_0+k_1\times 2+k_2\times 2^2+\cdots+k_m\times 2^m=k_0+2(k_1+2(k_2+2(\cdots+2k_m)\cdots)) \tag{7}$$

如果知道每一个 k 的值,也就得到 q 的值。在这里我们进行 $m+1$ 次的迭代运算,将每一个 k 的值计算出来,就可以直到 q 和 r 的值。

本文的基于加法的除法器,在三角形光栅化设计中主要完成对直线段斜率和线段起始点的计算,并将其应用到 Bresenham 算法的步进遍历中去。

1.3 改进 Bresenham 算法描述

对于 Bresenham 算法来说,该算法首先限定了直线的斜率是小于 1 的,这样在根据 x 方向进行步进时,y 方向的取值要么是 0,要么是 1,但是不会大于 1,这样就不会由于斜率大于 1 而导致直线断裂。但是,对于三角形来说,需要得到的是三角形三条边及其内部的像素点,如果在 y 方向是步进的,而在 x 方向即使在直线上是断裂的,但是形成的三角形仍然是内部已填充的,而不会出现断裂现象。三角形与直线的这种区别就可以允许在斜率大于 1 的时候,仍然使用 Bresenham 算法进行迭代计算。

改进的 Bresenham 算法,不因为直线斜率的大小而改变步进的方式,单独按照一个方向进行步进计算。改进的

算法首先利用基于加法的除法器计算边函数 $f(x,y)=Ax+By+C=0$ 的边的斜率 Δx 和边的初始有效坐标 x_0，计算的斜率和初始坐标的值都会存在一个余数，然后按照 Bresenham 算法步进的方式每次一个单位的步进，在另一个方向上增加一个斜率值和余数的累积，如果余数累积大于 1，则在斜率值的基础上加 1，否则就保持斜率值。

假设在 y 方向对直线进行步进，那么 y 相对 x 的斜率设为 Δx，设初始坐标为 (x_0,y_0)，在利用基于加法的除法器进行斜率和初始坐标计算时，会产生两个余数 r 和 e，那么在 y 方向步进 1 时，$x_1=x_0+r+\Delta x+r=x_0+\Delta x+(e+r)$，则只需要知道 $e+r$ 是否大于 1(在硬件计算中是整型分子大于整型分母)，就能判断 $x_1=x_0+\Delta x+1$ 还是 $x_1=x_0+\Delta x+1$，如果加 1，就会 $e=e+r-1$，否则令 $e=e+r$，然后将 e 代入下一次迭代中。

2　三角形光栅化

三角形光栅化的功能结构流程如图 2 所示，一个三角形的三个顶点参数和三条边函数被接口模块接收，并根据顶点信息得到包围三角形最小的矩形边界框；然后根据边函数判断使 $f(x,y)=Ax+By+C\geqslant 0$ 的像素点为三角形内部，三条边分别被三个边函数归一化模块接收，将不同类型的直线方程(参数 A、B 的正负)统一转换为统一类型的边函数方程 $f(x,y)=Ax+By+C\geqslant 0,(A<0,B\geqslant 0)$，并对齐到三角形的边界框的相对坐标，如图 3 所示，相对坐标规定边界框左上角为(0,0)，X_{max}、Y_{max} 代表包围三角形的边界框的大小。

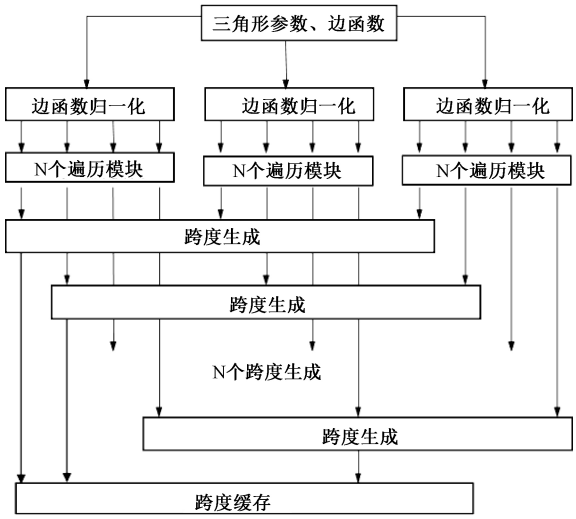

图 2　三角形光栅化功能结构流程

遍历模块是整个三角形光栅化最重要的部分，每个遍历模块一个时钟周期生成一个端点，每条边一个时钟周期生成 N 个端点。在这一过程中共经历 3 次处理，首先，遍历模块会沿着边界框的 $x=0$ 进行向下步进，也就是边界遍历，找到使 $f(0,y)\geqslant 0$ 的 y 的取值。然后利用基于加法的除法器算出当前位置 x_0 和 Δx 的值，同时得到的还有初始 x_0 的余数 e_0 和 Δx 的余数 r_0。最后，根据 4 个初始值，按照改进 Bresenham 算法沿着 y 方向进行遍历，直到 x 或 y 的值超过边界框为止，就得到每一个有效的 y 坐标对应的 x 坐标，也就是得到每个跨度的端点坐标，如图 3 所示，为沿着 y 方向进行遍历得到的 x 的坐标；该坐标经过归一化还原为其实际坐标。

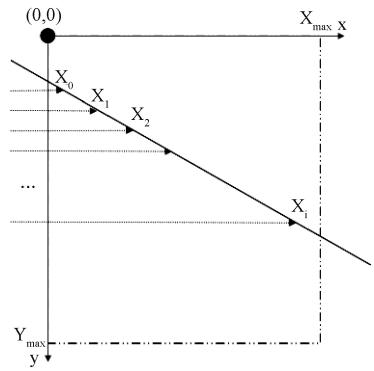

图 3　Bresenham 遍历直线

跨度生成模块将同一 y 坐标对应的三条边的 x 坐标进行比较，找到一条跨度的左右两个端点，即生成了一条跨度，并将其送到跨度缓存中。这样一个三角形的光栅化即完成，得到每个跨度端点以及端点之间每一个点即三角形内部的所有的点。

3　仿真验证

文中利用改进的算法进行 SystemC 建模仿真，验证算法的正确性。建模仿真进行算法的验证结果如图 4 所示，图中左侧为利用该算法进行仿真的结果，右侧为参考图形。可以看出，左侧算法完全可以实现对三角形的光栅化，且效果与右侧一致。

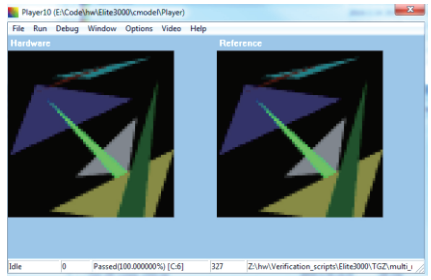

图 4　建模仿真验证结果

本文通过 FPGA 验证测试平台对算法进行硬件设计验证。与传统的基于 Bresenham 算法的三角形光栅化进行对比，如表 1 所示，可以看出改进算法使用的硬件资源更少。而且通过对大量不同种类数量三角形进行光栅化的速

度进行统计对比,改进的算法在总体速度上面提高了约 32%。

表 1 两种算法的 FPGA 实现硬件资源占用情况对比

算法对比	硬件资源		
	寄存器	查找表	LUT
传统 Bresenham 算法的光栅化技术	3728	6936	1027
改进 Bresenham 算法的光栅化技术	3120	4653	875

4 结 论

本文提出的一种改进 Bresenham 算法的三角形光栅化技术,极大地减少了三角形内部像素点的计算量,相应提高了三角形边的遍历速度,且算法占用较少的硬件资源,节省了 GPU 的硬件资源,降低了硬件的单位成本。结合三角形光栅化的并行设计思想,在充分合理利用硬件资源的基础上,提高光栅化过程的速度和效率。目前该算法的实现仍停留在 FPGA 验证阶段,后续会在 GPU 设计中进行工程验证。

参考文献

[1] 田泽,刘天江,张骏等.基于扫描线填充的三角形图元双向光栅化技术[J].小型微型计算机系统,2015,36(6):1398-1402.

[2] 符鹤,谢永芳.TBR 架构 GPU 中三角形高效光栅化[J].中国图象图形学报,2015,20(4):527-532.

[3] 黄锐.实时嵌入式图形系统中的三角形光栅化研究与设计[D].上海:上海交通大学,2008.

[4] 张文杰,张宝军,贺琼.三角形光栅化的硬件设计与验证[J].信息技术,2018(3):135-139.

[5] 聂嬰,田泽,马城城.一种并行扫描的三角形光栅化算法设计与实现[J].信息通信,2016(3):67-68.

[6] 刘青楠,曾泽仓,杜慧敏,等.一种易于硬件实现的嵌入式 GPU 三角形光栅化算法[J].微电子学与计算机,2018,35(9):26-31.

[7] 唐煜程,张明君,王浩宇,等.基于 GPU 的三维人脸数据动态线性快速修复[J].电子测量与仪器学报,2016,30(6),959-967.

[8] 王文栋,李开宇,杨盛亚.基于 FPGA 的反走样直线生成算法研究[J].电子测量技术,2015,38(4):83-87.

[9] 于平.基于 GPU 加速的辐射度光照算法的研究及应用[J].国外电子测量技术,2016,35(11),46-52,57.

[10] 王睿,陈春晓,刘高,等.基于自适应包围盒划分的体绘制加速方法研究[J].仪器仪表学报,2014,35(11):2560-2566.

[11] 王佳.Tile-based 嵌入式图形处理器的研究与设计[D].济南:山东大学,2014.

[12] 陈训.基于 OpenGL 的嵌入式图像系统的研究[D].西安:西安工程大学,2012.

[13] 刘钊.基于 Tile 的三角形光栅化算法研究与实现[D].西安:西安电子科技大学,2018.

[14] 詹鹏. 面向移动设备的 3D 图形光栅化处理单元的设计与实现[D]. 西安:西安科技大学,2012.

[15] 谭显强.基于 FPGA 的 3D 图形处理器 IP 核的设计与实现[D].南京:南京航空航天大学,2010.

作者简介

张加林,高级工程师,主要研究方向为潜艇指控、嵌入式系统开发、图形显示等。

阮成肖,工程师,主要研究方向为图形显示。

E-mail:ruanchengxiao@163.com