光栅化时的线性纹理插值

韩红雷[1,2,3)]，王文成[1)*]

[1)](中国科学院软件研究所计算机科学国家重点实验室 北京 100190)

[2)](中国科学院研究生院 北京 100049)

[3)](中国传媒大学动画学院游戏设计系 北京 100024)

(whn@ios.ac.cn)

摘 要：针对扫描转换多边形时，逐像素计算准确的纹理坐标需要用到除法运算，效率较低的问题，提出一种使用线性插值代替除法运算的纹理坐标计算方法.该方法以与屏幕平行的平面切割多边形形成的线段来对多边形进行光栅化，可以线性插值的方式实现纹理坐标在透视投影下的正确计算，消除了一般光栅化计算中所需要的除法运算；对于离散化所引起的插值误差，通过增加误差修正操作进行改进.实验结果表明，采用文中方法能较好地提高计算效率，并得到计算精度很高的纹理坐标.

关键词：栅格化；扫描转换；纹理坐标插值；透视投影

中图法分类号：TP391

Linear Texture Coordinate Interpolation in Rasterization

Han Honglei[1,2,3)] and Wang Wencheng[1)*]

[1)](*State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing* 100190)

[2)](*Graduate University of Chinese Academy of Sciences, Beijing* 100049)

[3)](*Game Design Department, Animation School, Communication University of China, Beijing* 100024)

Abstract: Existing algorithms for pixel rasterization and texture coordinate interpolation always need per-pixel division operations, severely limiting the computation efficiency. This paper presents a new algorithm with only linear computation, and so achieves much acceleration. The new algorithm decomposes a 3D polygon into a set of line segments with the planes parallel to the viewing plane, and uses the obtained line segments for pixel rasterization and texture coordinate interpolation. As the line segments are parallel to the viewing plane, texture coordinates can be computed with linear interpolation, without the expensive division operations. For the errors from such a rasterization, simple supplementary measures are provided for correction to get high precise results. Experimental results have shown the effectiveness and efficiency of the new algorithm.

Key words: rasterization; scan conversion; texture coordinate interpolation; perspective projection

在传统的 3D 图形应用程序中，通常是将 3D 对象表示成世界坐标中的多边形或者三角形的集合(多边形可以最终分解为多个三角形)、多边形顶点上负载渲染所必需的信息(比如纹理坐标，法线和颜色等).在进行渲染时，对于多边形在屏幕上覆盖的每个像素，需要根据多边形顶点上的负载信息来得到相关的信息，以进行纹理色彩等计算.

在图形流水线中，渲染的任务包括将多边形顶点

收稿日期：2010-07-07；修回日期：2010-10-19.基金项目：国家自然科学基金(60773026)；中国传媒大学 211 三期工程项目(21103040204).
韩红雷(1980—)，男，博士研究生，讲师，主要研究方向为计算机图形学、计算机动画等；王文成(1967—)，男，博士，研究员，博士生导师，论文通讯作者，主要研究方向为计算机图形学、虚拟现实、科学计算可视化等.

坐标转换到屏幕空间,然后将多边形进行逐行扫描转换(一般是横向水平扫描线,每个扫描线中以多边形边作为边界).通常情况下,要达到透视正确的要求[1],每个像素信息计算要用到一系列数学操作,其中包括除法操作.在英特尔奔腾系列 CPU 中[2],浮点数除法的时钟周期为 33,而浮点数乘法和加法的时钟周期仅为 3,整型数加法为 1.这种情况下,如果能将扫描转换多边形过程中逐像素的除法运算转换为乘法或者加法,将可以极大地提高渲染效率.

线性插值是一种高效的插值手段,在扫描转换多边形时,若像素对应的纹理坐标之间是线性关系的话,就可以利用简单的加法或者乘法操作得到像素的纹理坐标信息.然而,在透视投影过程中,屏幕空间和纹理坐标空间之间是投影映射关系,不能满足线性插值的条件,故直接利用线性插值会出现明显的视觉错误[3].

本文首先分析了目前计算透视正确纹理坐标的几种有代表性的方法及其各自的特点与不足;接着提出本文方法,将像素按照深度相同的原则组织成扫描线(通常是斜向的),以保证扫描线上的像素对应的纹理坐标满足线性关系,进而利用线性插值计算每个像素透视正确的纹理坐标;然后分析了这种方法带来的误差的特点,提出了修正误差的方法;最后给出了详细的实验数据及结论.

实际上,尽管本文分析过程着眼于计算像素的纹理坐标,但该方法也可以被应用于透视投影条件下扫描转换过程中其他数据的计算[4],如光照计算(Gouraud 或者 Phong 着色)、像素颜色计算(利用多边形顶点的颜色信息得到多边形内部像素的颜色)等.另外,本文的目的是计算高精度的纹理坐标,得到的纹理坐标还需要结合类似于文献[4]中的反走样技术来生成高精度的像素颜色,进而提高渲染质量.

1 透视正确的纹理坐标计算

文献[4]总结了纹理映射的 3 种方式:屏幕空间扫描、纹理空间扫描和 2 个空间同时扫描.其中屏幕空间扫描的伪代码如下:

```
for y
    for x
      compute u(x,y) and v(x,y)
      copy TEX[u,v] to SCR[x,y]
```

屏幕空间扫描是最常用的方法,该方法也叫做逆映射.映射的过程需要找到一种参数化表示,以便从屏幕空间映射到纹理坐标空间.纹理映射需要 2 步:纹理空间和齐次物体空间之间的映射,齐次物体空间和 2D 屏幕空间的映射.

对于参数化方式表示的曲面,这个过程易于表示,但是对于多边形网格组成的模型来说却不那么直接.对于多边形网格来说,在多边形顶点处定义了其对应的纹理坐标(u,v),这样就定义了一个从纹理空间到 3D 物体空间的仿射变换.屏幕坐标(x,y)和物体坐标(x',y',w')以及纹理坐标(u,v)之间的关系为

$$(x,y)=\left(\frac{x'}{w'},\frac{y'}{w'}\right) \tag{1}$$

$$\begin{bmatrix}u\\v\\1\end{bmatrix}=\begin{bmatrix}K & L & M\\N & P & Q\\R & S & T\end{bmatrix}\begin{bmatrix}x'\\y'\\w'\end{bmatrix} \tag{2}$$

通过式(1)(2)可以建立起屏幕坐标和纹理坐标之间的关系

$$(u,v)=\left(\frac{Kx+Ly+M}{Rx+Sy+T},\frac{Nx+Py+Q}{Rx+Sy+T}\right) \tag{3}$$

通过式(3),可以在屏幕空间扫描转换多边形的过程中计算得到每个像素对应的纹理坐标.但从式(3)可以看出,屏幕坐标和其对应的纹理坐标之间是双曲线关系,计算一个像素的准确纹理坐标至少需要 2 个除法操作或者一个求倒数操作和 2 个乘法操作.

目前有 2 种方法可避免逐像素除法这样的费时操作:将双曲线用直线或者二次曲线近似[5-6]和利用 Midpoint 算法[7].下面分别对这 2 种方法进行分析.

1) 使用直线或二次曲线近似双曲线的方法

这类方法利用了基于误差控制的技术,按照误差准则对多边形进行子分,子分多边形内部像素坐标和纹理坐标之间的双曲线关系用线性或者二次曲线逼近.文献[8]发现,误差的最大值发生在三角形的边上,故而可以通过控制三角形边上的误差来对原始三角形进行子分,每个子三角形采用传统方式进行扫描转换,并且采用线性插值计算像素纹理坐标,得到的纹理坐标误差在可控范围内.但是,该方法都是在取一种效率和误差的折中,所得到的纹理坐标误差较大,并且在某些情况下有明显的视觉错误,如渲染大面积多边形时.

2) Midpoint 算法

该算法很好地分析了双曲线公式的特点,巧妙地利用了相邻像素间的相关性,并利用迭代加法结

合条件判断来计算像素的纹理坐标，避免了除法运算[1,7,9]. 但是利用该算法得到纹理坐标误差绝对值最大是 0.5，在需要精度较高的场合难以适用.

Kim 等[10]通过深入挖掘 Midpoint 算法的优势，提出了能够得到任意精度纹理坐标的方法. 该方法首先利用类似 Midpoint 的纹理坐标计算方式得到当前像素纹理坐标的整数部分，接着使用有限次的加法迭代以及条件判断得到小数部分（下文中将该算法称为 Kim 算法）. Kim 算法中设计了特殊的硬件来支持这种算法，原因是计算纹理坐标小数部分时像素之间是独立的，硬件可以实现并行计算. 但这种算法存在一些缺陷：1）效率的提升有赖于文中提出的硬件结构；2）计算精度的提高会造成栅格化效率的急剧下降.

本文提出一种精确计算像素纹理坐标的方法，使用线性插值作为计算纹理坐标的手段，每个像素计算只需要少数浮点数乘法及加法操作，其误差修正过程使用了直线逼近双曲线的策略.

2　线性插值的条件

由于屏幕坐标和其对应的纹理坐标之间是双曲线关系，对于每条扫描线，如果直接采用双线性插值算法计算像素纹理坐标，则会造成类似图 1 a 所示的透视错误，图 1 b 所示为透视正确的渲染图像[3].

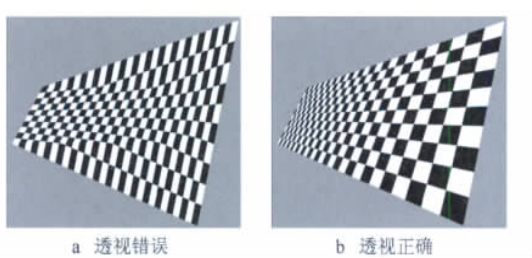

图 1　采用不同方法对四边形扫描转换的结果

实际上，水平扫描线上相邻的像素投影到物体表面后，它们之间的距离不再相等，而是呈现非线性的变化，如图 2 所示；由于物体空间和纹理空间之间的变换是仿射变换，所以最终映射到纹理空间后也是非线性变化的，这将导致无法直接使用线性插值计算像素纹理坐标.

通过观察这个过程我们发现，扫描线上像素点的深度不一致是造成这个问题的原因所在（如图 2）. 那么相应的，若扫描线上的点深度一致，则扫描线上的线性变化映射到物体空间后仍然是线性的，也就可以使用线性插值方式来得到纹理坐标. 在

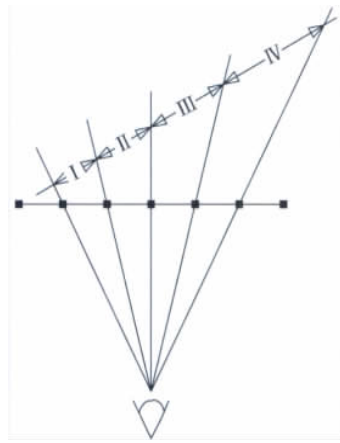

图 2　屏幕空间中的线性变化在物体空间中是非线性的

图 3 中，以平行屏幕的平面切割空间中待栅格化的多边形形成线段 L，在屏幕空间中很容易证明和 L 平行的每条直线 L' 上的点深度相同，本文将这些直线称为固定深度扫描线. 和传统扫描线不同，这种类型的扫描线不是水平方向的，而可能是屏幕空间的任意角度.

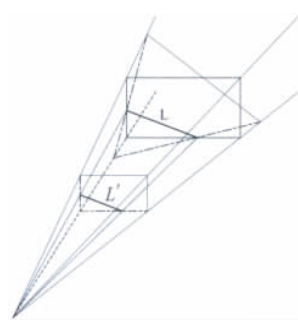

图 3　线性插值的条件

下面证明，如果沿着固定深度扫描线来扫描转换多边形，则像素纹理坐标的变化是线性的.

通过分析式(1)(2)可以得到（下文以计算纹理坐标 u 为例，v 的计算类似）

$$u=K\times x\times w'+L\times y\times w'+M\times w' \tag{4}$$

如果扫描线上深度一致，w' 在这条扫描线上是常数，则式(4)可以表示为

$$u=A\times x+B\times y+C,\ A,B \text{ 和 } C \text{ 为常数} \tag{5}$$

由于屏幕空间上的扫描线可以表示为 $y=gx+b$ 的形式（其中 g 为固定深度扫描线的斜率. 为了便于讨论，下文中假设固定深度扫描线的斜率 g 的绝对值小于 1. 如果 g 的绝对值大于 1，将 x 和 y 调换即可），则式(5)最终可以表示为 $u=Step\times x+D$，$Step$ 和 D 为常数，故

$$u(x)-u(x-1)=Step \tag{6}$$

式(6)表明,若沿着固定深度扫描线来扫描转换多边形,则像素纹理坐标的变化是线性的. 可以采用直线的扫描转换算法对每条固定深度扫描线进行扫描转换(如 DDA 算法),然后利用递加 $Step$ 的方式计算其经过像素的纹理坐标.

3 算法改进

第 2 节中给出的固定深度扫描线算法虽然可以使用递加的方式来得到扫描线经过像素的纹理坐标值,但是此时的扫描线一般是倾斜的,失去了传统的水平或者垂直方向扫描线方式的优势,比如难以使用多边形边界剪切扫描线等.

本文需要对算法进行修改,以便能按照传统扫描转换方式来遍历多边形所覆盖的所有像素. 图 4 所示为对一个多边形进行固定深度扫描线转换的过程. 图 4 中,上半部分表示固定深度扫描线的离散化模式;下半部分是利用这个扫描线对多边形进行扫描转换的结果. 由于扫描线按照从下到上的顺序间隔 1 个像素递进,而每个像素和扫描线在 y 方向的距离不超过 0.5,所以屏幕上的每个像素都可以唯一地被分配到离它最近的扫描线中. 更进一步地,在 y 方向上相邻的 2 个像素,其所属的扫描线也相邻. 这就意味着,在屏幕上的每列像素中只要索引到一个像素的扫描线,其他像素的扫描线也随之确定.

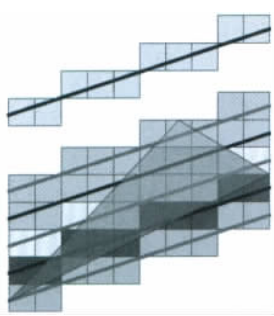

图 4 屏幕像素和固定深度扫描线的对应关系

这种利用索引方式找到像素所属扫描线的方法需要按照垂直扫描线的方式扫描转换多边形(如果扫描线的斜率 $|g|>1$,采用水平扫描线),且需要为每个扫描线计算起始点的纹理坐标以及纹理插值步长信息并保存(假设分别保存于 $TexCoord$ 和 $Step$ 数组中). 则第 2 节中的固定深度扫描线算法可以修改如下.

算法 1. 使用固定深度扫描线进行线性插值得到像素纹理坐标的算法

① 将多边形转换到观察坐标系当中;

② 求得该多边形所在平面和投影平面相交直线的斜率 g;

③ 找到处于多边形下方的第一条固定深度扫描线 L',设其和多边形包围盒左边相交点坐标向下取整后为 (x_s, y_s);

④ while 固定深度扫描线 L' 和投影到屏幕中的多边形相交;

⑤ 计算 $Step_u$, $Step_v$ 并保存于 $Step$ 数组中;

⑥ 计算 L' 起点像素 (x'_s, y'_s) 的纹理坐标 (u, v) 并保存于 $TexCoord$ 中;

⑦ L' 向上移动一个像素, $y'_s += 1$;

⑧ endwhile

⑨ while 垂直扫描线和投影到屏幕中的多边形相交

⑩ 找到垂直扫描线中的多边形边界像素 $p(x, y)$;

⑪ $offset = x - x_s$;

⑫ $index = y - \mathrm{int}(y_s + g * offset + 0.5)$;

⑬ while 像素 p 不是垂直扫描线中多边形的另一个边界像素

⑭ $u = TexCoord[index].u + offset * Step[index].u$;

⑮ $v = TexCoord[index].v + offset * Step[index].v$;

⑯ 通过纹理坐标 (u, v) 得到当前的像素颜色;

⑰ $y = y + 1$;

⑱ $index = index + 1$;

⑲ endwhile

⑳ 下一条垂直扫描线;

㉑ endwhile

4 算法复杂度分析

假设固定深度扫描线数目为 n. 下面从时间复杂度和空间复杂度进行分析.

1) 时间复杂度. 从算法 1 的第⑬~⑲行可以看出,在最内层循环中要用到 2 个浮点数乘法、2 个浮点数加法和 2 个整数加法. 从第⑨~⑫行可以看出,在每个垂直扫描线中需要用到 1 个浮点数乘法、2 个浮点数加法和 2 个整数加法.

2) 空间复杂度. 从算法 1 的第⑤,⑥行可以看出,该算法需要用到 $4n$ 个浮点数来在扫描转换过程中保存计算所需的数据.

5 误差修正

由于屏幕像素是整数坐标,而固定深度的扫描线的斜率是任意的,所以第 3 节中给出的算法会产

生误差. 如图 5 所示，利用算法 1 计算出的纹理坐标值实际上是固定深度扫描线上的纹理坐标，而非像素点的纹理坐标. 这种误差会造成渲染图像出现“锯齿效果”的走样现象.

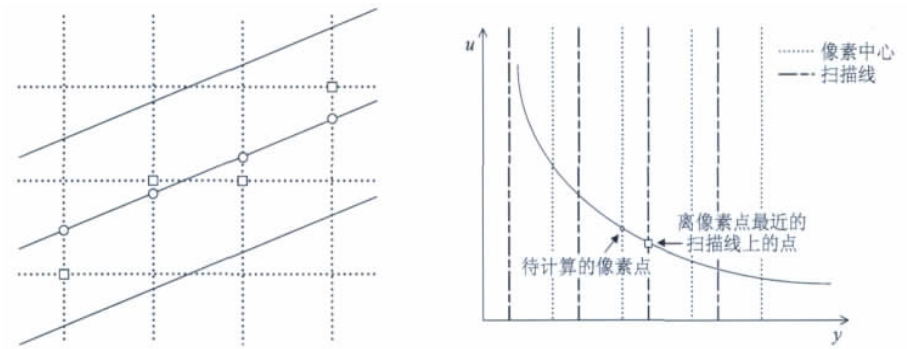

图 5　固定深度扫描线上的点和像素中心之间有一定的误差

解决误差问题的办法就是减少离散点和斜向扫描线之间的距离. 可采用如下 3 种方式：

1）减少固定深度扫描线的间隔进行误差修正（下文中称为加密扫描线修正误差）. 从图 6 a 可以看出，利用这种方式可以减少像素和固定深度扫描线之间的距离，进而提高纹理坐标精度. 但是这种方式会成倍增加空间复杂度，也会增加索引固定深度扫描线时的复杂度.

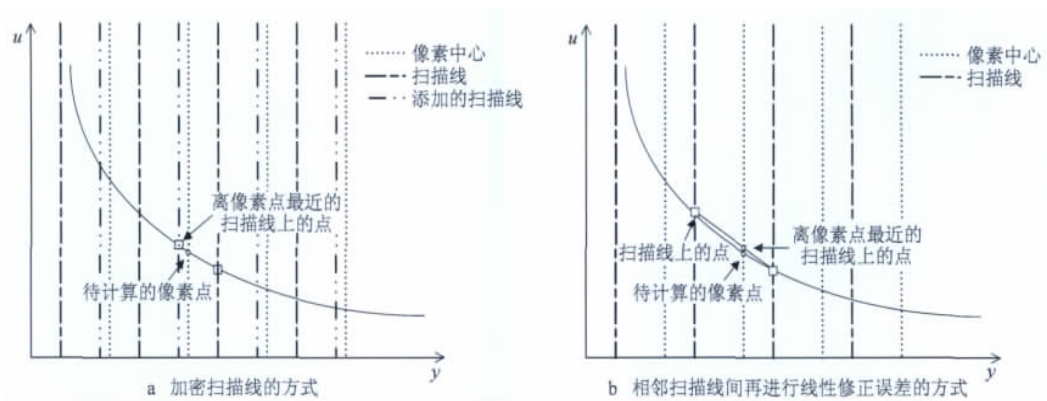

图 6　误差修正

2）找到离像素点最近的 2 条扫描线，将纹理坐标和屏幕坐标的双曲线关系近似看作线性关系，以像素点和扫描线之间的距离作为参数再进行一次线性插值（下文中称为线性修正误差）. 该方式和算法 1 在空间复杂度上一样，只在第⑨行后的代码中有所不同，并且在最内层循环中该方式只增加了 2 个浮点数乘法和 4 个浮点数加法. 但从图 6 b 可以看出，该方式的误差修正效果明显.

3）上面 2 种方式的结合. 这种方式指的是既增加固定深度扫描线的密度，又使用线性修正误差. 很明显，这种修正方式的代价也是最大的.

6　比较及结论

图 7 所示为采用不同的纹理坐标计算方法计算得到像素纹理坐标，并采用双线性插值的方式从纹理中得到像素颜色. 图 7 a，7 b 所示为采用 Kim 算法计算得到的渲染图像，纹理坐标平均误差分别为 0.25 和 0.125；图 7 c 所示为采用本文提出的固定深度扫描线算法；图 7 d 所示为采用本文提出的固定深度扫描线算法并进行线性修正误差算法；图 7 e 所示为采用式(3)计算准确的纹理坐标得到的渲染图像. 可以看出，Kim 算法可以较好地控制渲染图像的质量；直接使用固定深度扫描线算法容易造成渲染图像的“锯齿”效果，误差较大；如果对固定深度扫描线算法进行线性修正误差，将获得接近“精确”的渲染结果.

图 8 所示的 2 个统计表是利用本文方法对 14 个不同场景进行纹理坐标计算所使用的时间比较（将 Kim 算法达到相同平均误差所需时间归一化为 1），图 8 b 所示为在计算纹理坐标时进行了线性修正

误差，而图 8 a 没有. 可以看出，经过误差修正后的准确率有极大提高. 要达到和本文方法接近的精确度，Kim 算法要花费更多的时间(每种算法都采用软件方式实现).

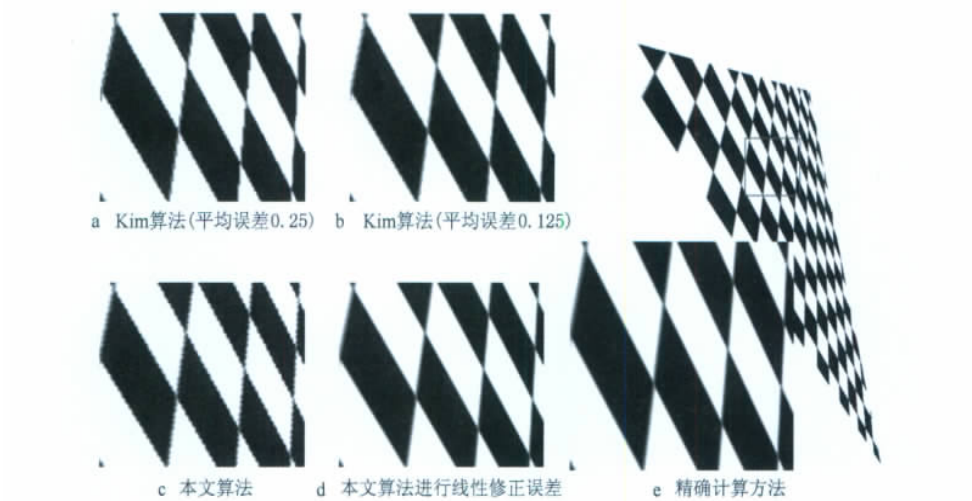

图 7 Kim 算法以及本文算法渲染效果比较

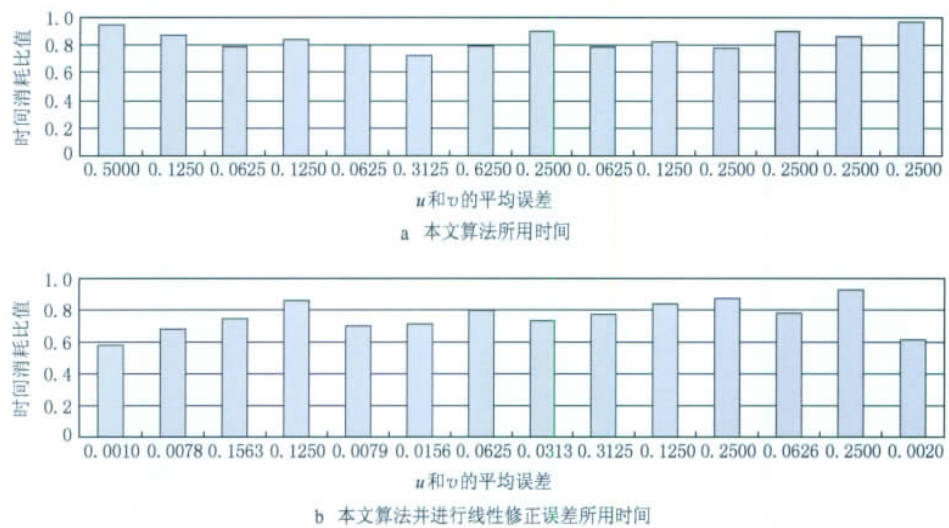

图 8 本文算法效率

图 9 所示为本文算法的误差统计表，其中对比了是否使用误差修正的效果. 可以看出，同样的场景进行线性修正误差操作后，绝大多数情况下可以极大地提高精度，甚至将误差减小到接近于 0.

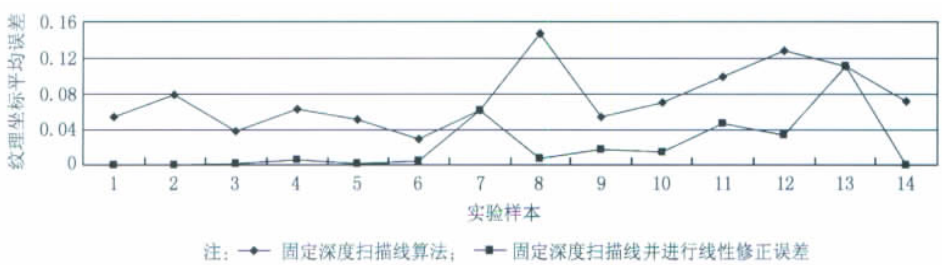

图 9 本文算法精度统计

图 10 中对比了文中提出的不同误差修正方式的效果及时间效率，可以发现，单纯使用加密固定深度扫描线进行修正误差的误差修正效果较好，但远达不到进行线性误差修正的水平，并且随着扫描线密度的增加，其效率下降比较严重. 使用线性修正误差可以极大提高精确度，但这种情况下加密扫描线

并不能显著地提高误差修正的效果，反而使得计算效率降低.

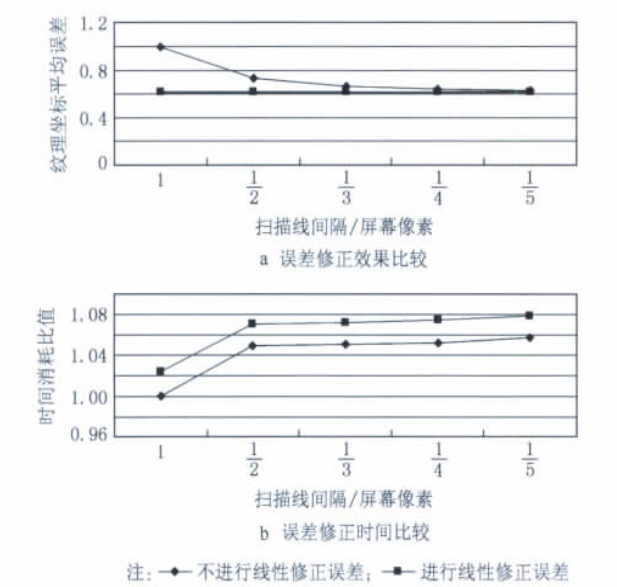

图 10　进行误差修正对纹理坐标误差及效率的影响

综上，对于纹理坐标精度要求较低的场合，可以直接使用固定深度扫描线算法；对于精度要求较高的场合，应使用固定深度扫描线算法并进行线性修正误差. 本文方法较好地达到了精确度和计算效率的平衡，通过占用一定的内存空间，解决了直接使用固定深度扫描线算法所引起的无法使用传统扫描转换方式的不足；通过线性修正误差，又可以在不过多增加时间的情况下很好地提高纹理坐标计算精度. 该方法计算简便，很容易在硬件上实现，可适用于各种计算平台，具有很好的应用前景.

致谢　感谢中国传媒大学动画学院费广正教授在本文写作过程中提出的宝贵意见！

参考文献(References)：

[1] Blinn J. Hyperbolic interpolation [J]. IEEE Computer Graphics and Applications, 1992, 12(4): 89-94

[2] Hecker C. Let's get to the (floating) point [OL]. [2010-07-07]. http://chrishecker.com/images/f/fb/Gdmfp.pdf

[3] Heckbert P S, Moreton H P. Interpolation for polygon texture mapping and shading [M] // Rogers D E, Earnshaw R A. State of the Art in Computer Graphics: Visualization and Modeling. New York: Springer, 1991: 101-111

[4] Heckbert P S. Survey of texture mapping [J]. IEEE Computer Graphics and Applications, 1986, 6(11): 56-67

[5] Demirer M, Grimdale R L. Approximation techniques for high performance texture mapping [J]. Computer & Graphics, 1996, 20(4): 483-490

[6] Hecker C. Perspective texture mapping, part 4: approximations [OL]. [2010-07-07]. http://chrishecker.com/images/3/37/Gdmtex4.pdf

[7] Pitteway M L V. Algorithms for drawing ellipses or hyperbolae with a digital plotter [J]. The Computer Journal, 1967, 10(3): 282-289

[8] Kamen Y, Shirman L. Triangle rendering using adaptive subdivision [J]. IEEE Computer Graphics and Applications, 1998, 18(2): 95-103

[9] Barenbrug B, Peters F J, van Overveld C W A M. Algorithms for division free perspective correct rendering [C] //Proceedings of ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware. New York: ACM Press, 2000: 7-13

[10] Kim D, Kim L S. Area-efficient pixel rasterization and texture coordinate interpolation [J]. Computers & Graphics, 2008, 32(6): 669-681