GPU在实时阴影绘制中的应用

刘 双，申闫春，狄翠萍
（北京信息科技大学 计算机学院，北京 100192）

摘 要：实时阴影在增强三维场景真实感方面起着非常重要的作用。阴影体算法是实时阴影绘制中效果非常理想的一种方法。但是随着场景复杂度的增加，该算法计算量比较大，将导致绘制效率的降低。另一方面，随着可编程GPU技术的发展，GPU的渲染速度远远大于CPU，为提高三维场景的渲染效率提供了更大的空间。在此基础上，介绍了一种在GPU上生成阴影体的方法，加速实时阴影绘制。利用图形硬件的图形处理单元(GPU)的运算能力和可编程性，将生成阴影体的大量计算从CPU转移到GPU，从而有效地提高实时阴影的绘制效率。

关键词：图形处理单元；阴影体；几何着色器
中图分类号：TP391.9 **文献标识码**：A **文章编号**：1673-629X(2009)11-0226-04

Real-Time Shadow Rendering Using GPU

LIU Shuang, SHEN Yan-chun, DI Cui-ping
(School of Computer, Beijing Information Science & Technology University, Beijing 100192, China)

Abstract: Real-time shadow is very crucial for improving 3D scene reality. Shadow Volume algorithm is one of the most effective algorithms for Real-Time shadow computation. However, as the complexity of the scenes increasing, this algorithm may slow down the render efficiency because it needs more computation. On the other hand, with the development of programmable GPU, the render speed of GPU is much faster than CPU, and it has provided more space for the render efficiency acceleration of the 3D scenes. Based on this, introduced a method, which generate shadow volume on GPU to faster Real-Time shadow rendering. With the use of computation capacity and programmability provided by graphics processing unit(GPU)of underlying graphics hardware, most of computation of generating shadow volume are transferred from CPU to GPU, and this method can effectively accelerate real-time shadow rendering.

Key words: GPU; shadow volume; geometry shader

0 引 言

阴影在虚拟现实中扮演了重要的角色，它提供光源对物体的照射信息，增加计算机虚拟场景的真实感。动态实时阴影技术是计算机图形学研究中一个基本问题，同时也是一个难点。阴影体算法是作为虚拟现实中常用的生成实时阴影的算法之一，其优点是可以在任何地方产生正确的清晰阴影。传统生成阴影体的过程全部由CPU完成，因而算法的性能受到约束。随着计算机图形硬件的快速发展，尤其是可编程图形硬件的出现，为三维场景中实时阴影体算法的研究提供了更加广阔的空间。

收稿日期：2009-02-28；**修回日期**：2009-05-21
基金项目：北京市教育科研计划项目(KM200811232006)；科技型创新基金(BT2008-13)
作者简介：刘 双(1985-)，女，硕士研究生，研究方向为虚拟现实及仿真技术、流媒体技术；申闫春，博士后，教授，研究方向为虚拟现实及仿真技术、流媒体技术。

1 阴影体算法介绍

阴影体算法是Franklin C. Crow在1977年写的一篇论文“SHADOW ALGORITHMS FOR COMPUTER GRAPHICS”里提出的。这是一种使用纯几何信息来生成阴影的方法。其原理是先从光源的方向去寻找物体的轮廓边，然后将轮廓边沿着光源方向延伸，所有延伸出来的面形成了一个筒形的区域。在筒的一端用面向光源的遮挡面封闭，而在延伸出去的无穷远处也用一个平面进行封闭，就得到一个完全密封的区域，称其为阴影体(shadow volume)[1]。该算法可以应用于一般的图形硬件上，唯一的要求就是模板缓冲。模板缓存类似深度缓存，通过对所要绘制的像素进行模板测试，判断像素是否写入模板缓冲，此外，模板缓冲还提供了递增和递减的操作，通过这些操作可以判断物体是否在阴影体内。

目前，阴影体主要有两种实现方式，z-pass方法和z-fail方法，以下分别介绍这两种方式。

z-pass方式：

z－pass 是阴影体最初的标准算法,用来确定某一个像素是否处于阴影当中。

其原理是:

第一步:开启深度缓存,渲染整个场景,以真实视点作为视点得到关于所有物体的深度图。这和阴影图(shadow maps)算法里面的深度图是有区别的,在阴影图算法中深度图是以光源为视点得到的。

第二步:关闭深度缓存,开启模板缓存,渲染所有的阴影体。对于阴影体面对视点的面,如果像素通过深度测试,则该像素对应的模板值加一;如果深度测试的结果失败,模板值不变。对于阴影体中背向视点的面,如果深度测试的结果失败,模板值减一,否则保持不变。

第三步:根据每个像素的模板值判断其是否处于阴影当中,如果模板值大于零,则该像素在阴影体内,否则在阴影体外,然后据此绘制阴影效果。

简单概括 z－pass 算法就是从视点向物体引一条视线,当这条射线进入阴影体的时候,模板值加一,而当这条射线离开阴影体的时候,模板值减一。如果最终模板值为零,则表示射线进入和离开阴影体的次数相等,物体不在阴影体内。图 1 演示了被渲染片元在阴影体中的情况。

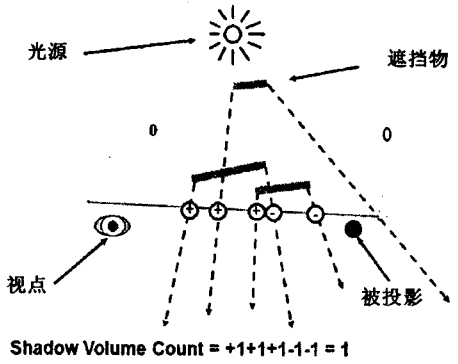

图 1　z－pass 方法

z－pass 算法有一定的缺陷,当视点进入阴影体后,该算法会产生错误的阴影。因此,John Carmack, Bill Bilodeau 和 Mike Songy 各自独立提出了另一种算法:z－fail 算法。该算法弥补了 z－pass 算法的不足,不管视点是否在阴影体内,都能产生正确的阴影。

z－fail 算法原理如下:

第一步:开启深度缓存,渲染整个场景,得到深度图。(这一步和 z－pass 的完全一样)

第二步:关闭深度缓存,开启模板缓存。渲染阴影体,对于背向视点的面,如果深度测试失败,模板值加一,如果深度测试通过,模板值不变。对于面向视点的面,如果深度测试失败,模板值减一,否则,模板值不变。

第三步:根据每个像素的模板值判断其是否处于阴影当中,如果模板值大于零,则这个像素在阴影体内,否则在阴影体外,然后绘制阴影效果。

z－fail 算法可以正确处理视点在阴影体内的情况,如图 2 所示。

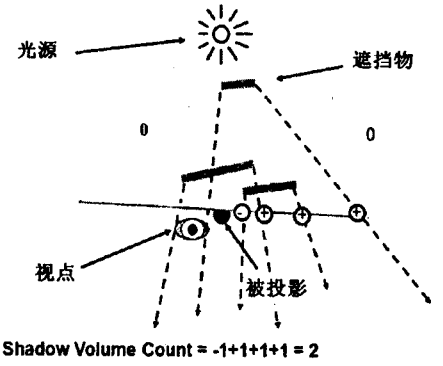

图 2　z－fail 方法

阴影体算法的优点在于它可以应用于一般的图形硬件上。其次,由于这种算法不是基于图像的,从而避免了采样问题,这样可以在任何地方产生正确的清晰阴影[2]。但是生成阴影体的计算量比较大,需要先判断每个物体的轮廓边,然后将所有的轮廓边延伸至无穷远,这一过程会迅速消耗填充率。尤其在绘制大场景时,需要极其巨大的显存带宽甚至是 CPU 占用率,从而影响场景的渲染速度。

2　可编程图形硬件技术

人们对三维图形的实时性和真实感要求越来越高,仅仅在软件方面对算法进行改进具有一定的局限性。随着计算机硬件技术的发展,可编程图形硬件随之产生。可编程图形硬件不仅可以完成固定功能模块的工作,还可以通过使用着色语言编写自己的着色器程序取代原有 CPU 上的某些功能模块,为编程人员提供了一个非常灵活的操作空间。GLSL(OpenGL shading language)是继 HLSL(high－level shader language), Cg(C for graphics)之后的又一种高级着色语言,它是 OpenGL 2.0 规范的一部分,应用于 OpenGL 平台上。

早期 GPU 提供的顶点着色器和片断着色器能替代固定的顶点处理过程和像素处理过程,但只能对 CPU 中已有的数据进行处理,不能在 GPU 上生成新数据。如果要动态创建物体,只能在 CPU 执行。目前,NVIDIA 公司的 Geforce8 系列图形处理器 GPU 已经加入了 OpenGL 几何着色器,并将其作为 OpenGL2.0 标准的扩展[3]。几何着色器工作在顶点与像素着色器之间,专门用来处理场景中的几何图形。显卡中的顶点着色单元生成顶点信息之后,就会将这些结果交

给几何着色器，几何着色器可以根据顶点的信息来批量处理几何图形，还可以在原有顶点信息的基础上动态生成新的顶点，构成更为复杂的几何图形，处理结果能够快速地传递给其他着色器或显存，这个过程无需 CPU 参与，由 GPU 独立完成[3]。

3 GPU 在阴影体计算中的应用

文中利用了图形处理器的可编程能力，用 GLSL 实现轮廓边的判断以及阴影体的生成，可以有效地提高实时阴影的绘制速度。

3.1 轮廓边的定义

轮廓边表示从光源的角度看物体所得到的轮廓线。确定轮廓边的方法是找出那些朝向相反（一个面向光源，另一个背向光源）的两个三角形所共享的边，这些边最终形成轮廓边，其他的边都在阴影的内部。将轮廓边沿着光照的方向延伸到一定距离以外或者无穷远处得到阴影体。其中，计算三角形向光性的方法如图 3 所示，对于每个三角形计算 dot(L, Ni)，如果 (dot>0)，表示该三角形面向光源，否则，背向光源[4]。

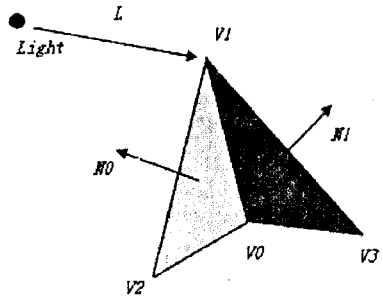

图 3 判断轮廓边

3.2 几何着色器生成阴影体

几何着色器处于顶点着色器和片元着色器之间，接收顶点着色器处理过的顶点，然后以图元为单位对顶点再进行一次处理，并重组输出一个以上的图元。

OpenGL 为几何着色器定义了新的可调整图元类型：GL_TRIANGLES_ADJACENCY_EXT，如图 4 所示。

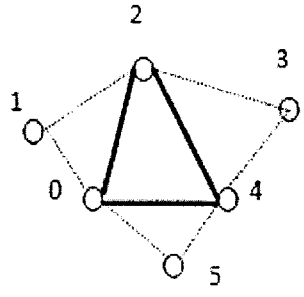

图 4 GL_TRIANGLES_ADJACENCY_EXT 类型

该类型图元由 6 N 个顶点组成，N 指的是该图元的数目。如图 4 所示，顶点 0、2、4 组成主三角形，而顶点 1、3、5 分别与主三角形的两个顶点组成修正三角形。通过 GL_TRIANGLES_ADJACENCY_EXT 模式传递主三角形的邻接信息到几何着色器，然后逐一判断主三角形与修正三角形相对光源的朝向是否相同，从而判断主三角形的三条边是否为轮廓边。将轮廓边的判断和生成阴影体的工作由 CPU 转移到 GPU，大大降低 CPU 的负担，从而有效提高渲染效率。

使用几何着色器之前，需要指出输入和输出图元的类型。输入类型为 GL_TRIANGLES_ADJACENCY_EXT，因为输出的图元既包括三个顶点的遮挡面和远平面，还有四个顶点的阴影体侧面，所以选择 GL_TRIANGLES_STRIPS 类型，同时指定允许输出最大顶点数为 18(遮挡面 3 个顶点 + 远平面 3 个顶点 +3 * 侧面 4 个顶点)[5]。

为了使用高级绘制语言 GLSL，几何着色程序需要用到两个扩展库的支持：GL_EXT_geometry_shader4 和 GL_EXT_gpu_shader4。着色运算时，几何着色器需要获取顶点着色器处理过的图元的顶点坐标 gl_PositionIn[5]，提取设置光源位置的 uniform 变量：vec4 light_pos。

在几何着色器中进行阴影体计算大致分为 3 步：

第一步：计算主三角形的相对光源的朝向，如果主三角形背向光源，则说明该三角形不是遮挡面，程序返回，否则，执行后面的操作。

以下提供了在 Geometry shader 中计算三角形朝向的关键代码：

```
uniform vec4 light_pos; //取得光源的位置
vec3 ns[3]; // 法线
// 计算主三角形每个顶点的法线[6]
ns[0] = cross(gl_PositionIn[2].xyz - gl_PositionIn[0].xyz,
gl_PositionIn[4].xyz - gl_PositionIn[0].xyz );
ns[1] = cross(gl_PositionIn[4].xyz - gl_PositionIn[2].xyz,
gl_PositionIn[0].xyz - gl_PositionIn[2].xyz );
ns[2] = cross(gl_PositionIn[0].xyz - gl_PositionIn[4].xyz,
gl_PositionIn[2].xyz - gl_PositionIn[4].xyz );[3]
// 计算每个顶点的相对光源的方向
d[0] = light_pos.xyz - light_pos.w * gl_PositionIn[0].xyz;
d[1] = light_pos.xyz - light_pos.w * gl_PositionIn[2].xyz;
d[2] = light_pos.xyz - light_pos.w * gl_PositionIn[4].xyz;
// 判断主三角形的相对光源的朝向
if ( ! (dot(ns[0],d[0])>0||dot(ns[1],d[1])>0 ||dot(ns[2],
d[2])>0) ) {
// 如果背向光源则返回
if ( robust = =0 ) return;
.....
```

第二步:渲染主三角形,形成阴影体的遮挡面(front cap)和远平面(back cap),然后输出图元。

其中,遮挡面就是主三角形本身,远平面由主三角形顶点延伸至无穷远处形成,实现顶点延伸的代码如下:

```
v[0] = vec4(light_pos.w * gl_PositionIn[0].xyz - light_pos.xyz, 0);
v[1] = vec4(light_pos.w * gl_PositionIn[2].xyz - light_pos.xyz, 0);
v[2] = vec4(light_pos.w * gl_PositionIn[1].xyz - light_pos.xyz, 0);
```

生成的顶点经过投影变换之后,调用 EmitVertex()输出,最后调用 EmitPrimitive()输出新生成的三角形图元。

第三步:对于主三角形的每一个修正三角形,判断相对光源的朝向,如果背向光源,说明它与主三角形的公共边为轮廓边,将轮廓边的两个顶点延伸至无穷远,与原来的顶点形成一个四边形,将该四边形以三角形带的模式输出[5]。图 5 为生成阴影体侧面。

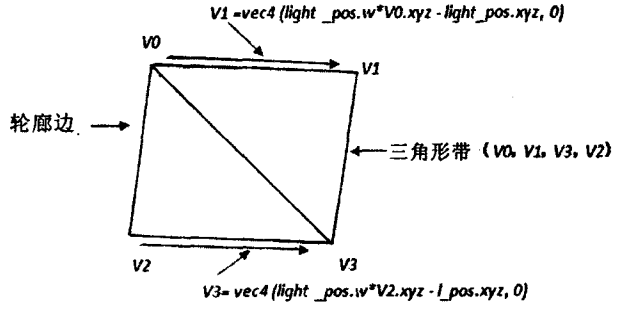

图 5　生成阴影体侧面

至此,GPU 完成阴影体的生成,CPU 将对生成的阴影体进行后续处理,通过模板缓存绘制阴影。

通过使用 OpenGL 的扩展 GL_ARB_geometry_shader_ext,整个程序在 CPU 为 PentiumD(双核) 2.80 Ghz; 内存为 1.0 G; GPU 为: Nvidia GeForce 8800 GTS(显存 640 M)的平台运行通过,图 6 显示了最终效果。

4　结束语

阴影体算法可以提供高质量的阴影效果,已经被广泛应用于虚拟现实技术。随着可编程图形硬件的应用,将阴影体生成算法映射到 GPU 来执行,从而加速复杂三维场景中实时阴影的绘制,在增强场景真实感同时,保证了绘制速度,带来逼真的视觉享受。还可以从其他方面进一步提高阴影的渲染速度,如采用剪切测试结合深度测试去掉场景中不需要渲染的物体等等[7]。

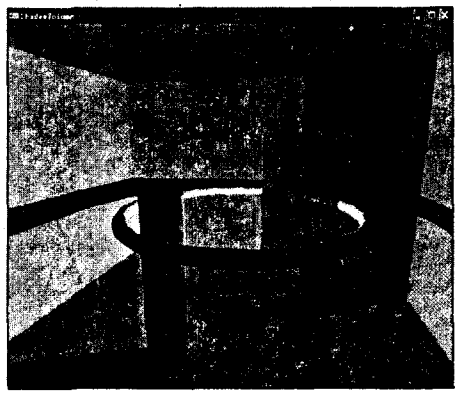

图 6　绘制阴影

参考文献:

[1] 李从信,吴秀芹,吴秀英,等. 基于虚拟现实技术的阴影算法理论研究[J]. 佳木斯大学学报,2005,25(3):291 - 293.

[2] 孙漠舟,费耀平. 一种新的实时阴影渲染算法[J]. 现代电子技术,2007(12):119 - 121.

[3] 顾　伟. OpenGL 几何着色器研究[J]. 中国科技信息,2008,20:113 - 115.

[4] Shi Yazheng. Performance comparison of CPU and GPU silhouette extraction in a shadow volume algorithm[D]. Sweden: UMEA University,2006.

[5] Nguyen H. GPU Gems 3[M]. Boston: Addison Wesley Professional,2007.

[6] 陈金水,颜伟琼. 基于 OpenGL 的三维建模在水利行业中的应用[J]. 计算机技术与发展,2006,16(3):197 - 199.

[7] Kilgard M J. Robust Stencil Shadow Volumes[EB/OL]. 2001. http://developer.nvidia.com/object/cedec_stencil.

(上接第 225 页)

[6] 李木金,王光兴. 一种被用于网络性能管理的模型及实现[J]. 计算机学报,1999(11):1196 - 1203.

[7] 徐　罡,黄　涛,刘绍华,等. 分布应用集成核心技术研究综述[J]. 计算机学报,2005(4):433 - 444.

[8] 马晓星,余　萍,陶先平,等. 一种面向服务的动态协同架构及其支撑平台[J]. 计算机学报,2005(4):467 - 477.

[9] 朱　磊,周明辉,刘天成,等. 一种面向服务的权限管理模型[J]. 计算机学报,2005(4):677 - 685.

[10] 许　峰,赖海光,黄　皓,等. 面向服务的角色访问控制技术研究[J]. 计算机学报,2005(4):686 - 693.

[11] 李长云,李　莹,吴　健,等. 一个面向服务的支持动态演化的软件模型[J]. 计算机学报,2006(7):1020 - 1028.

[12] 吴步丹,金　芝,赵　彬. 面向服务的建模:一种全过程复用的方法[J]. 计算机学报,2008(8):1293 - 1308.