基于层次细节模型的遮挡裁剪算法

许云杰 胡事民
(清华大学计算机科学和技术系，北京 100084)

摘 要 遮挡裁剪和应用层次细节模型是两种有效的三维复杂场景渲染加速算法. 为了快速地进行三维复杂场景的渲染，提出了一种结合层次细节模型与遮挡裁剪技术的算法框架，该算法首先在预处理阶段，将场景划分成不同空间层次结构；然后在运行时刻，对较高的空间层次，可应用遮挡裁剪技术判别场景的可见性，并裁剪掉不可见场景部分；而在局部的较低层次上，则应用网格简化方法来选择适当的模型层次细节. 实验结果显示，该算法取得了较好的加速性能.

关键词 遮挡裁剪 三维渲染 层次细节模型
中图法分类号：TP391.72 TP391.9 **文献标识码**：A **文章编号**：1006-8961(2002)09-0962-06

An Occlusion Grid Culling Algorithm Based on LOD Models

XU Yun-jie, HU Shi-min
(*Department of Computer Science and Technology, Tsinghua University, Beijing* 100084)

Abstract Occlusion culling and LOD(Level of Details) both are effective techniques in accelerating the rendering process of 3D large scene. An algorithm framework which integrate occlusion culling and LOD is proposed in this paper. It partitions the scene volume into a hierarchical structure during the preprocess stage, and in runtime, it uses occlusion culling in the high level to cull the invisible scene, while view-dependent simplification are used in the low level to show the detail. This algorithm is mainly based on Luebke's work of mesh simplification and Heinrich's work of lazy occlusion grid culling in the image space, it extends the data structure of 'vertex tree' to support occlusion culling and also uses an occlusion grid buffer to reduce the number of expensive occlusion queries at pixel-level. Experiment result shows that significant speedup is achieved through our algorithm.

Keywords Occlusion culling, 3D Rendering, Lervl of details models

0 引 言

在计算机三维渲染领域，比如游戏、漫游、虚拟现实等应用中，经常需要动态交互地实现一些复杂场景，虽然当前计算机的图形处理能力得到了很大的提高，但仍然不能满足实际应用的需要. 一般说来，这类图形处理的耗时正比于被送入渲染管道的面片数量. 由于可见性裁剪和模型的网格简化方法从软件算法的角度，减少了不必要的送入渲染管道的面片，从而获得了渲染过程的加速效果.

在构成复杂场景多边形网格模型的所有面片中，其中往往只有很小的一部分面片是可见的. 可见性裁剪算法(Visibility Culling Algorithms)就是依据当前的视点位置，将那些不可见的场景部分裁剪掉，而只将可见的部分送入渲染管道，根据场景中不可见面片的种类，可以将裁剪分为视野区域外的裁剪(View Frustum Culling)、背向视点面片的裁剪(Backface Culling)和视野内被其他面片所遮挡部分的裁剪(Occlusion Culling)等 3 种裁剪类型. 前两种裁剪比较容易实现，已有成熟的算法；而对后一种裁剪类型，还没有一般性的算法.

基金项目：国家自然科学基金项目(69902004)
收稿日期：2002-06-13；**改回日期**：2002-07-08

遮挡裁剪的概念最早是由 Jones 和 Clark 提出的[1,2]；之后，Airey 和 Teller 首先将它应用于建筑物场景可见性的预处理中[3,4]，在这类场景中，观察者的视线总是会被某些墙、门窗等遮挡，而局限在个别房间（区域）内，致使看不到其他的绝大部分区域；Hudson 的方法是首先从场景中选出能产生较好遮挡效果的面片集（Occluder）[5]，然后对这些面片作投影计算，其中位于各阴影区域内的场景就是被遮挡的部分. 该算法的缺点是只能利用单独的面片作为遮挡面，且遮挡效果只能由该单一的面片决定；Koltun 等人提出了一种可以将多个面片合并成一个遮挡面（通常称为 Occluder Fusion）来进行处理的方法[6]，但是由于这种方法的预处理计算复杂度太高，因此难以推广到三维情况. 上述方法都是直接对场景中的几何面片作处理，也称为物体空间（Object Space）的方法，同类的方法还有许多，比如 Coorg，Durand 等提出的方法[7,8]. 与物体空间相对应的是图象空间（Image Space）. 在图象空间，Greene 提出了一种分层 Z-Buffer 算法[9]，其不同于传统的 Z-buffer 算法，它由多个不同层次的 Z-Buffer组成，最底层就是传统的 Z-buffer，以后每上一层，分辨率都随着降低，该层中的每个像素保存的 z 值就是下一层 2×2 像素的最大深度值. 这样就可通过这种金字塔形的结构来加速深度值的比较. 类似的算法还有 Zhang 的 HOM（Hierarchical Occlusion Maps）算法等[10].

场景中的物体或者物体的不同部分，在不同的视点相对位置或不同的光照环境等条件下，表现出不同的清晰度，随着清晰度的降低，可以逐步简化物体原有的网格，即可用数目较少的多边形来表示原物体. 网格简化的思想最早由 Clark 提出[2]，随后的研究产生了很多网格简化方法. 如 Cohen、Rossignac 等提出的方法[11,12]. 此外，Hoppe 还提出了一种基于边压缩的渐进式简化方法[13]. 但由于这些方法都只能从总体上对物体作层次细节的选择，而不能决定物体的局部层次细节，因此被称为视点无关的网格简化. 更复杂的网格简化方法是视点相关的网格简化，它能够决定物体的局部层次细节如何表示，比如靠近视点或正对光源的物体局部，用较高的层次细节表示，这类方法的代表是文献[14]，[15]等提出的方法.

由上可知，可见性裁剪和网格简化方法，由于从两种不同角度减少了送入图形渲染管道的面片数量，从而加速了整个显示过程. 由于可见性裁剪算法虽裁剪了不可见的场景部分，但是对于场景中一些不必要的细节，仍然是原样送入渲染管道；而网格简化方法虽简化了场景中不必要的细节，但是对于不可见的部分，也仍然是先作简化，然后渲染，因此，结合这两方面特点，显然可以获得更好的加速性能. 但是，迄今为止，对这方面算法的深入研究还很少，仅见于文献[16]、[17]等为数不多的文章.

Luebke 提出了一种基于空间层次结构的网格简化方法[15]，即顶点树（Vertex Tree）网格简化（图 1）方法. 该方法是将整个场景空间以层次结构表示（通常是八叉树形式），简化时，首先将最外层的树节点包含整个场景，然后依次进行递归分割，将各面片顶点划分到相应的子节点，直到叶子节点为止. 在算法的运行时刻，某些节点在简化条件的控制下（比如投影面积）可以简化成一个点，且其内部所含的顶点都由该点所代替，这类节点称为边界节点，比这类节点层次更低或更高的节点，分别称为非活动节点和活动节点. 这样整个树结构就可以分成活动区、边界区和非活动区 3 个连续区域，而其能够最终显示的面片都是位于活动区和边界区节点内的面片. 在算法中，则将这些边都记录在一个面的链表结构中，称为活动面表. 在对这类场景作漫游时，由于图象相邻帧之间的视觉区别往往很小，其表现在节点树上就是边界区内节点向上几个层次的简化或者向下几个层次的扩展，因此图象相邻帧的活动面表变化较小.

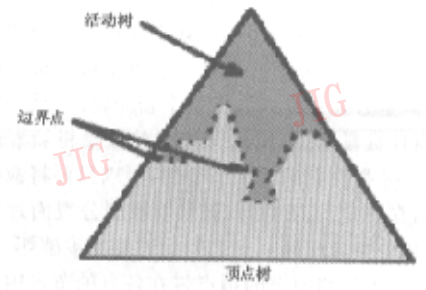

图 1　基于顶点树的网格简化方法（引自文献[15]）

1　算法思想

1.1　预处理——场景模型的空间分层

为了能够进行视野区域裁剪，Luebke 曾对算法作了改进，即由原来的采用一个活动面表改为采用多个活动面表，其中挂有活动面表的节点称为裁剪节

点,由于裁剪节点是包含所对应活动面表内每个面片的最小节点,故视野区域裁剪将针对每个裁剪节点进行,如果该节点的包围盒位于视野内部,或者与视野相交,则判定该节点是可见的,否则该节点就被裁剪,而视野区域裁剪就是通过这种粗略的测试来完成的,其粗略度由活动面表的个数决定,由于裁剪节点的递归深度越深,节点数就越多,其对应的活动面表个数也随之增加,因此对视野区域的判别也就越细致.

然而从该算法的数据结构中可以看到,由于某节点(图 2 中右上角的正方形包围盒)所含的面片 A 有一个顶点位于该视野区域内,所以即使该节点位于视野区域外,仍然需要对视野区域内的顶点作处理,也就是说,该节点不能简单地被裁剪(这种情况被称为节点的相关性).由于这种跨节点面片的普遍存在,因此将使裁剪性能明显降低.

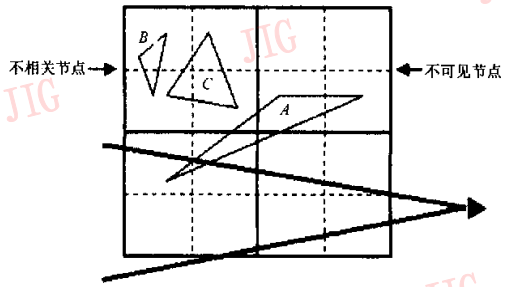

图 2 顶点树的不可见节点与不相关节点

为了解决节点相关性问题,本文采用了将这种跨节点面片分裂成几个非跨节点面片的方法.实践证明,经过这种分裂处理后,就不存在节点的相关性问题,经检测,在视野区域外的节点虽可以被简单地裁剪,但是分裂又导入如下两个新问题:一是分裂导致的面片数量增加,但由于增加的数量将随着递归的深入、分裂次数的增加而增加,所以只要将裁剪节点限制在一定的深度上,就可以控制分裂面片的数量;二是分裂后相邻节点之间的连接性被破坏了,其结果是位于分割面上的顶点将在各自的节点内进行合并、简化处理,就可能出现'裂缝'现象,对于这一问题,本文是采用对这种位于分割面上的顶点进行限制约束的方法来进行处理,即不允许对它做合并、简化操作.

本文在视野区域裁剪的基础上,加入了遮挡裁剪,最终使得整个算法思想形成这样一个框架,即在场景空间结构的较高层次(裁剪节点所在的层次)上,对场景物体进行可见性测试,以便将不可见部分裁剪掉;而在低层次的局部细节上,则应用网格简化方法来选择表现物体模型的适当细节层次.

1.2 运行时刻——遮挡裁剪与网格简化

1.2.1 对场景由前向后的遍历

由于产生遮挡关系的必要条件是前面的遮挡后面的,所以在遍历场景时,必须是以由前向后的方式进行,所幸的是,八叉树的空间分层结构自动地提供了这种遍历的特征.注意,这里的遍历只是在八叉树结构层次上的由前向后遍历的功能,而不是场景多边形面片层次上的由前向后的遍历.为简单起见,这里只考虑 2 维的情况,如图 3 所示.

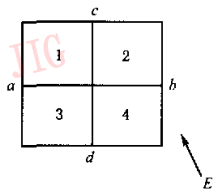

图 3 对场景由前向后的遍历

在图 3 中,由位于点 E 的视点对外部矩形包围盒内部的场景做由前向后的遍历.由图 3 可见,外部包围盒被细分为 4 个小的包围盒 1、2、3、4,其中,包围盒 3、4 与视点同在线段 ab 的一侧,包围盒 1、2 则刚好相反,由此可知,包围盒 3、4 位于包围盒 1、2 之前;同样,根据分割线段 cd 可以知道,因为包围盒 4 在包围盒 3 之前,包围盒 2 在包围盒 1 之前,所以最后的遍历顺序应该是包围盒 4→包围盒 3→包围盒 2→包围盒 1.当然,如果分割线段的顺序不同,那么遍历的次序也会调整,但是这种顺序同样保证了序号靠后的包围盒不会对序号靠前的包围盒形成遮挡关系.

1.2.2 遮挡网格缓冲区(Occlusion Grid Buffer)

在显示阶段,算法对场景作由前向后(由近向远)的遍历.这样先遍历的场景物体将对后遍历的物体形成遮挡关系,且遮挡区域随着已遍历物体的增

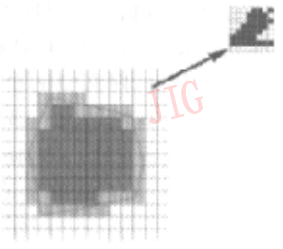

图 4 遮挡网格

多而扩大. 为了判断后遍历的物体是否被遮挡区域所遮挡，必须将该物体的屏幕投影区域与遮挡区域作比较，如果投影区域完全位于遮挡区域内部，则该物体必定被遮挡；反之，则未被遮挡. 虽然这种遮挡判别可以利用对屏幕像素作实际的硬件检测来实现，但不幸的是，这种像素检测会消耗一定的时间（在当前的硬件条件下，这种检测的消耗主要在于需要一个 setup-time），如果大量地使用这种检测操作（这在复杂场景的渲染过程中是必然的），那么将极大地降低算法的性能. 为了减少这种检测，算法采用 Heinrich 的方法[18]，将屏幕按水平和垂直方向划分成 $n \times m$ 的单元（cell），每个单元对应 16×16、32×32或其他数目的像素阵列. 由于每个单元的状态代表了它所对应的像素矩阵的遮挡情况（如图 4 中白、灰、黑方格即代表未被遮挡、部分被遮挡、完全被遮挡状态），因此遮挡网格缓冲区就代表了这些单元的状态，并随着场景的遍历而更新.

1.2.3　基于网格分割的遮挡裁剪

遮挡网格缓冲区单元的状态可分为下面几种情况：

（1）未被遮挡

（2）部分被遮挡

（3）完全被遮挡

（4）有待检验

在初始状态时，所有单元的状态位均设为未被遮挡. 但当某个单元位于包围盒的投影区域内部时，则这个单元称为非边界单元，而当单元与包围盒的投影区域相交时，则称为边界单元.

1.2.4　遮挡判别

为了检验某包围盒是否被遮挡，算法必须首先将该包围盒投影到投影面上，并求得在投影区域内部的非边界单元以及与投影边界相交的边界单元. 然后分下列情况讨论：

（1）如果某个非边界单元是未被遮挡或部分被遮挡的，那么单元里边那些未被遮挡的空白区域极有可能与包围盒内的面片相交，此时可认为包围盒未被完全遮挡.

（2）若某个边界单元状态未被遮挡，则包围盒也未被完全遮挡；

（3）若某个边界单元是部分被遮挡的，则进行硬件像素检测；

（4）若某个非边界单元是有待检验的，则做硬件检测：若检测结果是完全被遮挡，则修改单元的状态为完全被遮挡；反之，单元的状态改为部分被遮挡；

（5）若某个边界单元是有待检验的，则对重叠部分进行硬件检测：若检测结果是完全被遮挡，则单元的状态仍为有待检验；反之，单元的状态改为部分被遮挡.

1.2.5　场景遍历

算法在进行场景遍历时，将对场景的层次结构做递归检测，并依据包围盒的检测结果，而对相应八叉树节点分如下几种情况处理.

（1）若包围盒被完全遮挡，则其内部的面片也被完全遮挡，此时不用将它们送入渲染管道，而应结束对相应八叉树节点的遍历；

（2）若包围盒未被完全遮挡，且对应的八叉树节点所在层次高于裁剪节点的层次，则对该节点下一层的 8 个子节点进行递归检测；

（3）若包围盒未被完全遮挡，且对应的八叉树节点为裁剪节点（即存有活动面表的节点），则停止对后续子节点的遮挡检测，而代之以网格简化操作. 若继续利用相邻帧之间场景区别较少的特点，则只需对前一帧记录的边界节点作小范围的扩展或者压缩，并产生新的节点树和活动面表；同时将新的活动面片送入渲染管道，并调整包围盒投影区域内的单元（包括边界和非边界）的状态：如果相交单元原先的标志位是状态 1、2 或 4，则将该标志位改为有待检验.

从上面的算法中可以看到，单元的标志位若要变成完全被遮挡，则必须经过实际的硬件检测. 由此可见，在理论上，算法的结果必定是正确的，场景中可见的面片只是算法送入渲染管道的面片集的一个子集，也就是说，该算法是一种保守的算法.

2　实验结果及分析

算法实现的硬件环境是一台 CPU 为 P Ⅲ 450MHz、显卡 SiS6326、64M RAM 的 PC 机，编程环境是 MS Windows98、Visual C^{++} 6.0 和 OpenGL1.2. 像素检测是在 OpenGL 提供的模板缓冲区（Stencil Buffer）上进行的，对该缓冲区内像素的读取由函数 glReadPixel 完成. 为了方便起见，程序中，将屏幕输出范围限定在 640×480，按网格分割成 40×30 个单元，即每个单元对应 16×16 像素的阵列.

为了进行比较,本文分别用 3 种算法对一个面片总数约为 112 万的场景进行了渲染实验,图 5 和图 6 是实验结果.其中,图 5 是原始的场景图,即未采用层次细节模型和网格简化的场景图象,图 6 是采用了层次细节模型和网格简化后的场景图象.对图 5,又分别进行了不采用遮挡裁剪方法和采用遮挡裁剪方法的实验比较,由上面的分析可知,这种遮挡裁剪方法具有保守性,其场景中可见面片是经过裁剪后的面片集的真子集,由图 5、图 6 可见,这两种方法的显示效果是完全相同的.

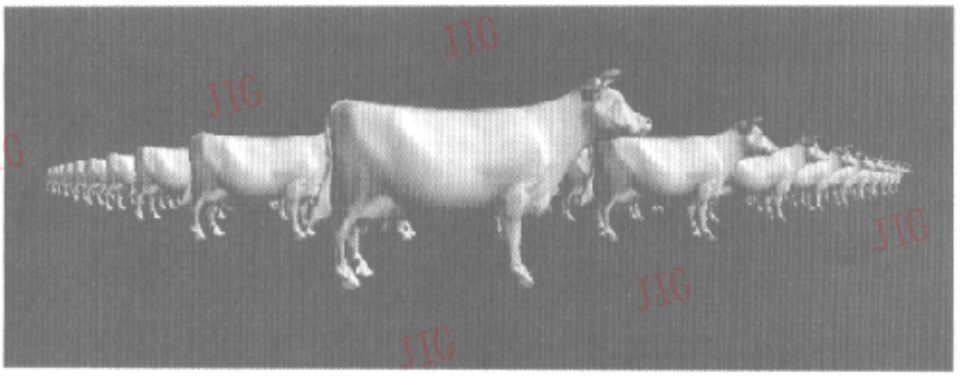

图 5 未采用遮挡裁剪与网格简化的原始场景图象

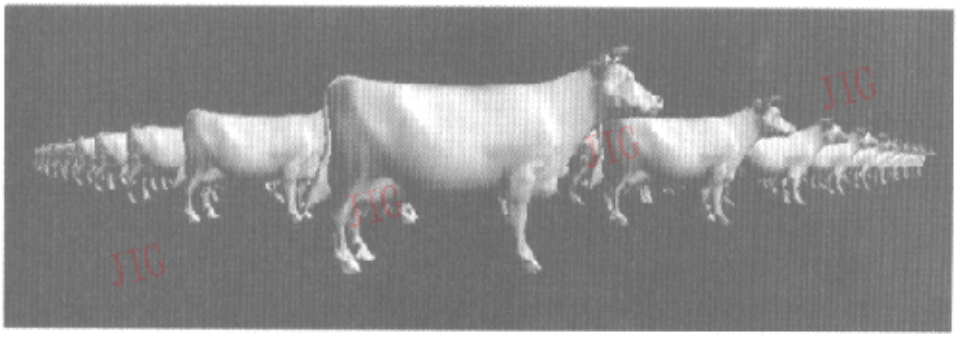

图 6 采用层次细节模型遮挡裁剪与网格简化后的场景图象

由表 1 可见,对于复杂的三维场景,尤其是具备高深度特征的,由于场景的大部分都被遮挡而不可见(比如本例),因此采用遮挡裁剪方法,可以大大减少实际渲染的面片数量;而采用层次细节模型,又更进一步加速了渲染过程,因为它可以减少不必要的细节显示,而同时又不会明显影响所生成的图象质量.

表 1 3 种方法的结果比较

	场景	是否遮挡裁剪	是否层次模型	渲染面片数(k 片)	渲染时间(s)
一	1	否	否	1120	17.2
二	1	是	否	174	4.5
三	2	是	是	44	1.5

3 结 论

本文算法结合了层次细节模型和遮挡裁剪技术两者优点,从而有效地加速了复杂场景的渲染,其不仅实现复杂度较小,并且加速效果明显.该方法首先在预处理阶段,将场景划分成不同空间层次结构,然后在运行时刻,对较高的空间层次应用遮挡裁剪技术判别场景可见性,裁剪掉不可见部分;而在局部的较低空间层次上,则应用网格简化方法来选择表现物体模型的适当细节层次.

本算法还存在许多有待改进的地方,比如对于网格简化,人们现在采用的方法多是简单的顶点树

节点内相邻顶点的合并，它虽然实现简单，并具有拓扑无关性，但是也存在简化比较粗糙的缺点，下一步的工作可以考虑采用更精细的网格简化方法，以进一步提高算法的效率.

参 考 文 献

1 Jones C B. A new approach to the 'hidden line' problem[J]. The Computer Journal, 1971,14(3):232～237.

2 Clark J H. Hierarchical geometric models for visible surface algorithms[J]. Communications of the ACM, 1976,19(10):547～554.

3 Airey J, Rohlf J, Brooks F. Towards image realism with interactive update rates in complex virtual building environments [J]. Computer Graphics, 1990,24(2):41～50.

4 Teller S. Visibility computations in densely occluded polyhedral environments[D]. PhD thesis, University of California Berkley, 1992.

5 Hudson T, Manocha D, Cohen J *et al*. Accelerated occlusion culling using shadow frusta[A]. In:Proceedings of 13th ACM Sympos. on Computational Geometry[C], New York, 1997:1～10.

6 Koltun V, Chrysanthou Y. Virtual occluders: An efficient intermediate PVS representation [A]. In: Proceedings of SIGGRAPH'2000 [C], InterLaken Switzerland, 2000:156～168.

7 Coorg S, Teller S. Real-time occlusion culling for models with large occluders[A]. In:Proceedings of 12th ACM Symposium on Computational Geometry[C], Philadelphia USA, 1996:78～87.

8 Durand F, Drettakis F, Thollot J. Conservative visibility preprocessing using extended projections[A]. In:Proceedings of SIGGRAPH'2000[C], New Orleans. USA,2000:168～178.

9 Greene N, Kass M, Miller G. Hierarchical Z-buffer visibility [J]. Computer Graphics,1993,27(10):231～238.

10 Zhang Hansong, Manocha D, Hudson T *et al*. Visibility culling using hierarchical occlusion maps [A]. In: Proceedings of SIGGRAPH'97[C], Los Angeles, 1997:77～88.

11 Cohen J, Varshney A, Manocha D *et al*. Simplification envelopes[J]. Computer gRAPHICS, 1996,30(8):199～208.

12 Rossignac J, Borrel P. Multi-resolution 3D approximations for rendering complex scenes[A]. In: Proceedings of Conference Modeling in Computer. Graphics[C], Berlin, German. 1993:455～465.

13 Hoppe H. Progressive meshes[J]. Computer Graphics, 1996,30(8):99～108.

14 Hoppe H. View-dependent refinement of progressive meshes [A]. In: Proceedings of SIGGRAPH'97[C], Los Angeles, 1997:189～197.

15 Luebke D, Erikson C. View-dependent simplification of arbitrary polygonal environments [A]. In: Proceedings of SIGGRAPH'97[C], Los Angeles, 1997:198～208.

16 Andujar C, Saona-Vazquez C, Navazo I *et al*. Integrating occlusion culling and levels of detail through hardly-visible sets [J]. Computer Graphics Forum, 2000,19(3):499～506.

17 El-Sana J, Sokolovsky N. Integrating occlusion culling with view-dependent rendering [A]. In: Proceedings of IEEE Visualizaition'01[C], San Diego. California, 2001:371～378.

18 Heinrich H, Tobert R F, Purgathofer W. Real-time occlusion culling with a lazy occlusion grid [A]. In: Proceedings of EUROGRAPHICS'01[C], Manchester UK,2001:215～223.

许云杰　1976 年生，清华大学计算机科学与技术系硕士研究生.研究领域为计算机辅助几何设计、计算机图形学.

胡事民　1968 年生，博士，清华大学计算机科学与技术系副教授、系副主任.研究领域为计算机辅助几何设计、计算机图形学、人机交互.