基于主要遮挡物的动态可见性算法

石振锋 赵 辉
(哈尔滨工业大学数学系, 哈尔滨 150001)

摘 要 对景物密集的复杂场景,提出了基于主要遮挡物的动态可见性算法.该算法通过场景中预先定义的主要遮挡物,动态地形成一个遮挡树,位于遮挡树遮挡区域中的景物将被剔除.当场景按照 BSP 树组织,并按从前向后的顺序绘制场景时,算法具有高效率.对主要遮挡物采用简化的遮挡物代理,对盒子类型的遮挡物提出了一种有效的简化算法.该算法已经被"RTG 三维图形开发工具包"采用,经实际验证,对复杂场景,该算法可以明显地提高绘制速度.

关键词 计算机图形学(520·6030) 可见性 PVS Beam 树 BSP 遮挡树 包围盒
中图法分类号: TP391.41 **文献标识码**: A **文章编号**: 1006-8961(2003)02-0219-06

A Run-time Visibility Algorithm Based on Major Occluder

SHI Zhen-feng, ZHAO hui
(*Mathematics Dept. of Harbin Institute of Technology, Harbin* 150001)

Abstract A run time visibility algorithm based on major occluder for complicated scene with high density is presented. First, some classic visibility algorithm such as Beam Tree, Potential Visible Set(PVS) technique and other ideas based on shading objects are summarized in this paper. Then an improved algorithm based on real-time application is presented. By creating an occluding tree dynamically through primary shading objects predefined in the scene user designed, the new algorithm can eliminate all objects and scenes in the shading area of the shading tree. The algorithm performs very high efficiency when the scene user designed is organized according to Binary Space Partitioning(BSP) tree and rendered from front to back. The algorithm apply simplified shading proxy different form shading objects to primary shading objects, and present an efficient and practical simplified method for shading objects with box type. The new improved algorithm has been applied to RTG 3D Real-time Graphics Toolkits and the ability of obviously increasing rendering speed for complicated scene has been verified by many practical instances in developing procedure.

Keywords Computer graphics, Visibility, Potential Visible Set(PVS), Beam tree, Binary space partitioning, Occluding tree, Bounding box

0 引 言

在计算机三维图形中,可见面的检测算法对于三维场景的绘制效果和速度非常关键,尤其是在对复杂场景进行实时绘制时,可见性的检测显得更为重要.目前大部分三维图形加速卡均使用 z-buffer 算法来解决面的遮挡问题.但是人们在绘制复杂场景时发现,使用 z buffer 算法很容易产生 10 次以上的重复绘制[1].例如在虚拟城市中漫游时,由于高大建筑物的遮挡,许多位于视锥内的景物实际上是看不见的,如何快速剔除这些被完全遮挡的景物是避免重复绘制的主要方法.这类算法被称为可见性(Visibility)处理算法.进入 20 世纪 90 年代后,人们对可见性算法给予了广泛研究,比较有代表性的可见性处理算法有 Beam 树算法[1],PVS 算法[2],大遮挡物算法[3]以及层次 z-buffer 算法[4].

1 典型的可见性算法

1.1 Beam 树算法

为了避免重复绘制,Beam 树算法按从前向后的次序绘制场景,即当一个面绘制后,该面就占据了屏幕上的一个区域,后绘制的面中位于此占据区域

收稿日期:2002-03-20;**改回日期**:2002-09-20

内的部分将不再绘制，而只绘制不位于此占据区域内的部分，并使占据区域扩大. 一个面在屏幕上形成的占据区域，是指从视点处发出的光线因被多边形面遮挡而在屏幕上形成的阴影区域，阴影区域和视点形成三维空间的一个锥体，称为阴影锥. 随着绘制的进行，占据区域将逐渐扩大，阴影锥的数目逐渐增加. 为了快速检索，用一株二元树来组织阴影锥，称为 Beam 树.

Beam 树是 BSP(Binary Space Partitioning)树，其结点记录阴影锥的侧面，面的一侧为阴影锥的内部，另一侧为阴影锥的外部.

Beam 树算法处理一个面的基本方法为：搜索 Beam 树，如果面完全位于阴影区域外，则绘制该面，并将该面对应的二元树插入到 Beam 树中；如果面完全位于阴影区域内，则不绘制该面；如果面与 Beam 树的结点平面相交，则用结点平面将该面切分成两个面，对每个面及其对应的 Beam 子树，算法递归执行.

使用 Beam 树算法不会出现重复绘制，但它引入的开销非常巨大. 随着绘制的进程，Beam 树越来越深，绘制一个面时，搜索和裁剪的工作越来越多. 在 Quake 游戏中，有人曾经尝试使用 Beam 树算法，但由于对复杂的场景，搜索和裁剪的开销太大，不能保证图形的实时绘制，结果放弃了该算法.

1.2 PVS 技术

PVS(Potential Visible Set)算法是对视点所在区域，预先计算出当视点在区域内漫游时，场景中各景物的可见性，表示这个可见性的集合称之为 PVS. 在显示时，根据视点所在区域的 PVS 决定景物是否需要进行绘制. 预先计算 PVS 的好处在于：它能大大提高对视锥外景物的剔除速度，并且对位于视锥内的景物只有少量的重复绘制(通常会出现 50% 的重复绘制)；它导致性能更加平均. 由于不必再对复杂场景进行附加的可见性测试，使最差情况与最好情况下的性能差距变小了. 但预计算 PVS 的过程比较复杂，计算量很大，另外，PVS 的大小也是一个要考虑的问题. 对于有上万个多边形的场景来说，一个未被压缩的 PVS 大小为几兆到几十兆字节，所以必须对 PVS 进行合理的数据压缩.

1.3 遮挡物思想

考虑到场景中的遮挡主要是由一些大遮挡物产生的，文献[3]提出了一种基于大遮挡物的绘制算法. 该算法根据视点的位置动态地在场景中选取一小部分遮挡物，以备遮挡时选用，由于在许多情况下，场景中的大部分遮挡是由视点附近的一些多边形引起的，因此选取视点附近的多边形作为遮挡物.

2 评述与改进

评述一个算法的优劣不是一件容易的事，因为每个算法都有其特定的应用环境，所以选择算法时应该依据具体的应用环境而定.

现在 3D 加速卡已经是 PC 机上的标准配置，底层支撑软件有 OpenGL 和 Direct3D. 用户只须把将要绘制的面或面组，通过接口程序传送给加速卡，加速卡就能够自动完成面或面组的绘制. 值得注意的是，组成景物的面最好成组地传送至加速卡，一个面一个面的传送方式将显著地降低绘制速度.

2.1 Beam 树算法的特点

Beam 树算法是完整、简明的算法，它对景物之间相互遮挡的问题给出了最直接的解决方案. 但是，Beam 树算法在使用中存在以下缺点：

(1)计算量太大，难以实用；

(2)要求景物按从前向后的顺序绘制，场景建模复杂，不能处理运动景物；

(3)必须一个面一个面地处理，不利于将面成组传送至加速卡.

2.2 PVS 算法的特点

PVS 算法是一种使用静态手段解决动态问题的算法，在处理景物可见性问题上，代表了一个独特的发展方向. PVS 算法有如下特点：

(1)可见性判断的开销极小，与其他传统算法比较，其绘制速度最快；

(2)可见性检查不完整；

(3)空间开销巨大；

(4)预处理时间太长(大于几小时)；

(5)不能处理运动景物；

(6)必须一个面一个面地处理，不利于将面成组传送至加速卡.

其中最后一点可以通过改进 PVS 算法，使其可见性检查不深入到面一级，这样既有利于面的成组传送，又可以减小空间开销.

2.3 大遮挡物算法的特点

大遮挡物算法[3]是对 Beam 树算法的一种改进，它没有将所有先绘制的面作为遮挡物，从而简化了遮挡物集合，明显地提高绘制速度. 该算法在绘制阶段，动态选取遮挡物，没有在场景数据库中引入额

外数据，从而具有广泛的适用性. 然而该算法的不足之处也正在于此，算法不能保证应该作为遮挡物的面全部被选中，在避免重复绘制上，效率不高；另一方面，在由一级上筛选作为遮挡物的面，计算量大，限制了筛选范围.

2.4　改进的算法

综合 Beam 树算法和大遮挡物算法，给出一种实用的可见性处理算法. 具体如下：

(1) 缩小 Beam 树算法中占据区域集合的规模

借助大遮挡物算法的思想，在场景中定义一些主要遮挡物，占据区域集合由主要遮挡物生成. 主要遮挡物可以是景物本身或其他简化的替代物.

(2) 可见性检查不深入到面一级

对景物的可见性检查，仅使用景物的包围盒，不涉及景物内部的面. 这样做的好处是可以减小可见性检查的工作量，另一方面也有利于实现面的成组传送.

(3) 使用 BSP 技术

按从前向后的顺序绘制场景，可见性处理效率比较高. 用 BSP 树结构来组织场景中的静态景物或至少使用 BSP 树结构来组织场景中的主要遮挡物，这样可以快速获得景物从前向后的绘制次序.

(4) 实现运动景物的可见性处理

在 Beam 树的结点平面中引入遮挡面自身，当景物无序绘制时，仍可进行可见性处理. 运动景物将在场景中最后绘制，从而提高了可见性检查的成功率.

3　基于主要遮挡物的可见性算法

改进算法所选取的主要遮挡物由一组面或一组长方体组成，主要遮挡物在建模阶段由设计者指定.

3.1　遮挡树

给定遮挡物上的一个面(以下称为遮挡面)，遮挡面的遮挡区域是视点与面各边形成的锥体被遮挡面自身截去头部的一个截头锥，位于截头锥内的景物将被遮挡面所遮挡. 如图 1 所示，遮挡面所形成的遮挡区域可用一株二元树来表示，称为遮挡树.

遮挡树的每个结点上记录一个平面，该平面是由视点与遮挡面的边定义的平面(称为边界面)，或是遮挡面自身. 每个结点上的平面将空间分为两个部分，标记为内侧和外侧. 对边界面，内侧是指遮挡面所在的一侧；对遮挡面，内侧是指远离视点的一侧.

遮挡树是在 Beam 树的基础上引入遮挡面自身产生的，引入遮挡面自身后，景物可以不按从前向后

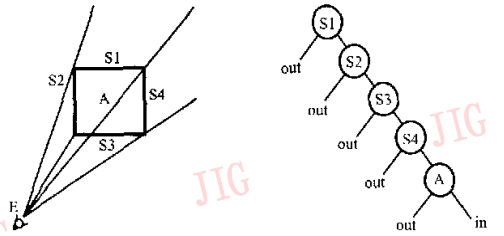

图 1　遮挡区域的二元数描述

的顺序显示，因为位于遮挡面前面的景物不被遮挡面所遮挡，这样就可以处理运动景物的可见性.

遮挡树结点的数据结构定义如下：

```
Struct OTreeNode;
{
    PLANE plane; //面方程
    OtreeNode * pIn; //内侧子树指针
    OtreeNode * pOut; //外侧子树指针
};
```

其中，*plane* 由(N,d)组成，N 为平面的法向量，d 为平面方程中的常数. 给定三维空间一点 P，如果

$$N \cdot P + d < 0$$

则 P 位于平面的内侧，否则 P 位于平面的外侧. 特别地，对于边界面 $d=0$.

给定一个遮挡面，遮挡树的产生过程为：取遮挡面的第 1 条边对应的边界面为树的根结点，置 *pOut*=NULL；后继边对应的树结点挂在 *pIn* 上，依此类推；边处理完毕后，将遮挡面自身结点挂在最后，置 *pIn* = NULL，*pOut* = NULL. 函数 OtreeNode * CreateOTree(POL *pol*) 对给定的多边形面 *pol*，创建一个遮挡树，返回遮挡树根结点指针.

3.2　遮挡树生长

通过向一个遮挡面的遮挡树插入另一个遮挡面的遮挡树，使遮挡树生长，产生多个遮挡面的遮挡树.

```
void InsertOTree( OtreeNode * pt, POL pol)
{
  if (! pt ) {
    pt=CreateOTree(pol);
    return;
  }
  if (pol 与 pt→plane 相交) {
    将 pol 切分为两个遮挡面 polIn,polOut;
    InsertOTree(pt→pIn, polIn);
    InsertOTree(pt→pOut, polOut);
  }
```

```
else if (pol 在 pt→plane 外侧)
    InsertOTree(pt→pOut, pol);
else {
    if (! (pt→pIn)) //pol 被遮挡
    return;
  else
    InsertOTree(pt→pIn, pol);
}
}
```

设 $pOTree$ 为指向场景遮挡树的全局变量，初始时 $pOTree$ = NULL，pol 为遮挡面. 如果 $pOTree$ ≠ NULL，$pOTree$ = CreateOTree(pol)，否则 InsertOTree($pOTree$, pol).

根据遮挡树的生成算法可知，不存在重合及相交的遮挡区域.

3.3 长方体遮挡物

现实世界中许多景物，特别是建筑物，可以简化为一个长方体或几个长方体的组合，因而引入长方体作为遮挡物比较具有代表性.

如果将长方体作为由 6 个遮挡面组成的遮挡物来处理，必定会在遮挡树中引入一些无效的结点，这些无效结点在遮挡树生长和景物可见性检查时，会导致无效的切割和判别，影响处理速度. 而将长方体简化为由其轮廓边组成的遮挡面，则可减少遮挡树的结点，避免遮挡区域过于“破碎”，提高可见性检查的处理速度.

如图 2 所示，当眼睛在长方体的左前上方观察时，可以看到 9 条边. 其中边 1-2-3-4-5-6 构成了长方体的外轮廓边；边 7,8,9 为内边，内边的公共顶点称为内点，内边和内点应该剔除掉.

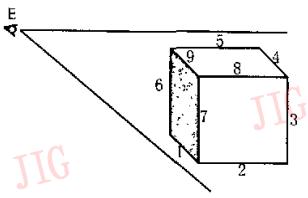

图 2 长方体的轮廓边

注意到长方体的 6 个面将空间分为 27 个区域，剔除掉长方体内部区域，视点所在区域有 26 种可能性供选择. 通过二分的办法，最多检测 6 个面就可决定视点所在的区域. 可以事先将视点所在区域与外轮廓边组成的对应关系做好，然后只要检查到视点所在区域，就可立即得到长方体的外轮廓边.

使用面代替长方体时，还要正确决定遮挡面自身平面的位置. 可以考虑让该面通过长方体中心，并垂直于视点和长方体中心的连线. 因为遮挡面自身平面将后于边界面检查，故这种作法不会出现错误.

3.4 景物可见性判别

判别一个景物是否可见，就是检查景物是否完全位于遮挡树的遮挡区域中. 在检查景物与遮挡树的位置关系时，不涉及到景物的面，仅使用景物的包围盒(通常是长方体)进行检查. 尽管景物包围盒的可见性与景物的可见性并不完全等价，但这种作法可以减少可见性检查的工作量.

遮挡树的遮挡区域是每个遮挡面遮挡区域的并集. 一个包围盒可能被多个物体的遮挡面的并集所遮挡，但不被任何一个独立的遮挡面所遮挡. 如图 3 所示，包围盒 T 被遮挡面 A 和 B 的并集完全遮挡，但是 A 和 B 均不能独立完全遮挡 T.

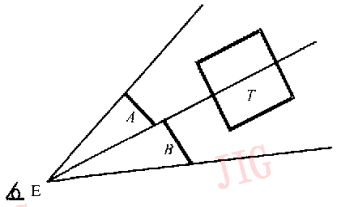

图 3 T 被 A、B 的并集遮挡

遮挡面的遮挡区域不能通过简单的处理合并在一起，尤其是引入遮挡面自身平面后，合并处理几乎是不可能的. 解决方案是用遮挡面的遮挡区域边界面将包围盒切开，对切开的部分独立进行检查.

3.4.1 包围盒可见性检查

用来进行可见性检查的包围盒 $\boldsymbol{T}$ 可以表示为 $\boldsymbol{T}=(\boldsymbol{P},\boldsymbol{E})$，其中，$\boldsymbol{P}$ 为可见顶点的集合，$\boldsymbol{E}$ 为可见棱边的集合. 根据视点的位置，最多时 $\boldsymbol{P}$ 包含 7 个顶点，最少时包含 4 个顶点，最多时 $\boldsymbol{E}$ 包含 9 个棱边，最少时包含 4 个棱边.

检查包围盒是否位于遮挡区域中，可以归结为两种操作：检查包围盒位于平面的哪一侧及用平面将包围盒切开.

检查包围盒位于平面的哪一侧可归结为检查 $\boldsymbol{P}$ 位于平面的哪一侧. 在检查包围盒与边界面的位置关系时，仅使用组成轮廓边的顶点就够了，而不必考虑内点，这样对边界面来说，最多检查 6 个顶点，最少检查 4 个顶点.

当包围盒与某个边界面相交时，要考虑 $\boldsymbol{E}$ 中哪

些棱边与边界面相交，相交的棱边要被边界面切开. 在最坏的情况下，可见棱边中有 4 个与边界面相交. T 被边界面切开后变成 $T_1=(P_1,E_1)$ 及 $T_2=(P_2,E_2)$. 其中，P_1 为 P 中位于边界面内侧的顶点及棱边与边界面的交点 C 组成的集合，E_1 为 E 中位于边界面内侧的棱边和那些因边界面切开棱边而产生的新棱边组成的集合；P_2、E_2 与 P_1、E_1 类似，只不过取边界面外侧的顶点和棱边. 如果 T 与遮挡面自身平面相交，则 T 可见.

包围盒的类型记为 BOX，使用函数 IsOccluded(OtreeNode * p, BOX T) 检查一个包围盒 T 是否被遮挡，如果 T 被遮，则挡返回 TRUE，否则返回 FALSE.

算法 1　检查一个包围盒 T 是否被遮挡

```
//根据视点位置取包围盒 T;
//指针变量 p 指向遮挡树的结点;
BOOL IsOccluded(OtreeNode * p, BOX T)
  {
  if (! p) return FALSE;
  if (T 与 p 指向处的 plane 相交) {
     if(! (p→pIn)) //plane 为遮挡面自身结点
       return FALSE;
     if (! (p→pOut)) //T 位于遮挡区域外
       return FALSE;
     plane 将 T 切分为 T1 和 T2;
  if (! (IsOcclude(p→pOut, T2)))
       return FALSE; //外部不被遮挡
     else
       return (IsOccluded(p→pIn,T1));
  }
  if (T 位于 plane 的外部 ) {
    if (! (p→pOut) )
      return FALSE;
    return (IsOcclude(p→pOut, T));
  }
  else if (! (p→pIn)){
    return TRUE;
  else
    return (IsOcclude(p→pIn, T));
  }
}
```

3.4.2　景物可见性检查算法

由于景物的层次结构，包围盒存在着复杂的嵌套，故检测时从最外层开始，使用递归算法逐层处理.

景物一般是由多个子景物通过层次树组织起来的. 在 RTG 三维图形开发工具包中，景物通过层次二元树组织，称之为景物树. 景物是否被遮挡的检查与绘制景物是一起进行的.

算法 2　具有可见性检查的景物绘制算法

```
void RenderScene(STreeNode * p1, OtreeNode * p2)
{
  if (! p1) return;
  if (景物树是最底层) {
    绘制 p1 指向的景物;
    return;
  }
    if (! IsOccluded(p2, p1→pb) {
    RenderScene(p1→pLeft, p2);
    RenderScene(p1→pRight, p2);
  }
}
```

算法 2 并不是最快速的算法，它存在两个问题：①当上层包围盒完全落于遮挡区之外时，不必对下层包围盒进行遮挡检查，直接绘制下层景物. 实现这点需要在算法 1 中引入景物完全落于景物遮挡区域之外的信息. 为简洁起见，该算法略去. ②当上层包围盒部分被遮挡时，要记录遮挡树当前结点位置. 当下层包围盒检查时，直接从该结点处开始，而不必从遮挡树根结点开始. 这种利用空间相关性的处理比较繁琐，该算法略去.

3.4.3　两级结构的遮挡树

根据遮挡物出现的顺序来动态创建的遮挡树不是一株平衡的二元树，在一些特殊情况下，遮挡树严重不平衡，导致遮挡树非常深，遮挡树生长和景物可见性判别的效率非常低. 为了避免这种情况，可以考虑使用具有两级结构的遮挡树.

引入一组边界面将场景图象逐次二分成一些均匀的区域，这组边界面构成了遮挡树的顶级，此时遮挡树是平衡的. 添加的边界面对应的树结点上，要设置一个标志位，以表明该边界面不是由遮挡物形成的. 遮挡树的顶级可以在场景图形数据加载阶段生成. 在遮挡树生长和景物可见性检查时，如果遮挡面和景物的包围盒与添加边界面相交，则要将遮挡面或景物的包围盒切开，因此添加边界面后会引入一些额外的切割操作，添加的边界面数目不宜过多，要根据场景的复杂程度适当选择.

对两级结构的遮挡树，遮挡树生成算法和包围盒可见性检查算法经简单修改，即可用于形成两级结构的遮挡树.

4 实验结果

本文的算法和建模工具已经在"RTG 三维图形开发工具包"软件上实现,实际结果表明,此算法在合理指定遮挡物的情况下,可以明显提高场景的绘制速度,对复杂场景的运行帧率提高了近一倍.以下是对 City(城市)、Office(办公室)场景的测试结果(如表 1 所示),并剪切了一些漫游时的图片(如图 4 所示).

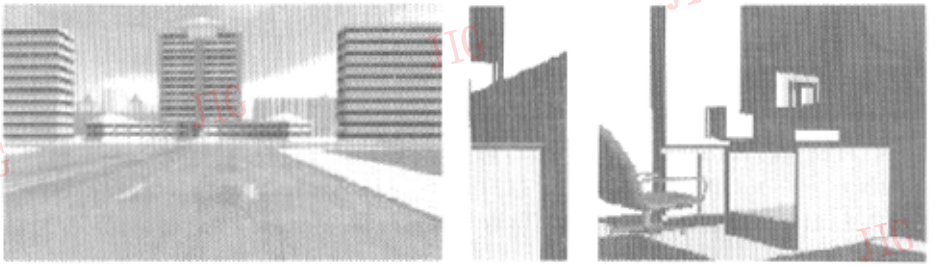

图 4 采用本算法后 City、Office 场景实时运行图

表 1 采用本算法后场景的运行数据

场景名称	多边形总数(个)	视锥剪切后(%)	遮挡处理后(%)
City	7952	30.2	7.6
Office	30343	25.7	6.57

视锥剪切后一栏数据是经过视锥剪切后所剩的可见多边形与多边形总数之比.遮挡处理后一栏数据是在分化视锥裁剪的基础上,进行遮挡判断后,剩下的可见多边形与多边形总数之比.

5 结 论

本文的工作主要具有如下 3 个特点:①用 BSP 方法组织场景,按从前向后的次序绘制,景物剔除效率高;②景物剔除和绘制同时进行,避免了因遮挡树规模过大而导致的搜索速度下降;③简化的遮挡物代理,显著降低了遮挡树的复杂性.经实例验证,采用本文算法,场景的绘制速度整体上有了很大程度的提高,尤其对存在大量运动物体的场景来说,效果更加明显.另外,场景的显示速度得到了平均化,帧速在场景的不同部分变化不大.

参 考 文 献

1 Michael Abrash. 图形程序开发人员指南[M]. 前导工作室译,北京:科学出版社,1998.3.

2 Teller S, Sequin C. Visibility computations in polyhedral three-dimensional environments [R]. U. C. Berkeley Report No. UCB/CSD 92/680, April 1992.

3 Satyan Coorg, Seth Teller. Real Time Occlusion Culling for Models with Large Occluders [A]. In: Proc. 1997 ACM Symposium on Interactive 3D Graphics [C], Los Angeles, California, USA, 1997:83~90.

4 Naylor B. Binary Space Partitioning trees as an alternative representation of polytopes[J]. Computer aided design, 1990, 22(4):223~234.

5 Norman Chin, Steven Feiner. Near Real-Time Shadow Generation Using BSP Trees [J]. Computer Graphics, 1989, 23(3):70~88.

6 Seth Teller. Visibility Preprocessing for Interactive Walk-throughs[J]. Computer Graphics, 1991, 25(4):61~69.

石振锋 1976 年生,2000 年获哈尔滨工业大学学士学位,现为哈尔滨工业大学数学系硕士研究生.主要研究方向为计算机三维图形实时生成技术、虚拟现实、计算机动画、计算机辅助几何设计.

赵 辉 1963 年生,副教授,1988 年获哈尔滨工业大学数学系硕士学位.主要研究方向为计算机图形学、虚拟现实软硬件技术.