文章编号:1673-1522(2017)02-0181-06 DOI: 10.7682/j.issn.1673-1522.2017.02.001

基于空间自适应剖分的Lightcuts多光源聚类算法

袁昱纬[1],刘传辉[1],全吉成[1,2],王宏伟[2],吴 晨[1]

(1.海军航空工程学院电子信息工程系,山东 烟台 264001;2.空军航空大学航空航天情报系,长春 130022)

摘 要:针对Lightcuts算法在面对大规模复杂光源时计算效率较低的问题,提出了一种基于空间自适应剖分的Lightcuts多光源聚类算法。该算法采用二叉树森林代替传统Lightcuts算法中的二叉树,并提出自适应的视景体划分方法对三维场景进行剖分,通过构建包含"簇-光源"对的列表,快速剔除与当前渲染点无关的光源,同时利用空间聚类的相似性减少"光源割"搜索过程中的重复计算。实验结果表明,与传统方法相比,文章提出的算法在"光源割"计算阶段能够将搜索步数平均减少30.71%~42.09%,绘制时间平均缩短28.35%~34.84%,有效地加快了Lightcuts算法的计算速度,提高了多光源三维场景的绘制效率。

关键词:多光源;Lightcuts;自适应空间剖分;二叉树森林;场景绘制

中图分类号:TP391.9 **文献标志码**:A

随着三维场景绘制技术应用的普及,人们对多光源环境下的绘制性能要求越来越高,如采用传统的光源累加计算方法,会导致光照计算时间与光源数量成正比,并且较大的时间开销成为制约算法效率提升的关键。

多光源环境下的场景绘制加速方法主要有:基于分类管理的方法[1]、基于空间层次结构的方法[2]、基于并行处理的方法[3-4]和基于空间聚类的方法[5]等,但都各有利弊。此外,Hollander等[6]研究了多光源环境下的实时全局光照,可以近似计算大量光源照射下的几何体伪影,但无法为少量光源照射的几何体产生精确的阴影。Sun等[7]基于稀疏体素八叉树提出混合实时GI算法来实现多光源全局光照计算,但多光源的组织管理没有进行优化。Olsson等[8]提出的平铺阴影技术支持包含大规模光源的三维场景绘制,该算法能够大幅度提高前向渲染和延迟渲染的效率。Olsson等[9]还提出了基于虚拟阴影图的算法来支撑大规模光源的绘制,能够快速剔除场景中与光照计算无关的几何体,但没有提高多光源本身的光照计算效率。

基于聚类的多光源场景绘制加速算法是通过对光源进行聚类,并采用代表光源表示光源集合,在保证绘制质量的前提下尽量减少参与计算的光源数量,降低光照计算的计算量,其中最典型的算法是Walter等[10]提出的Lightcuts算法。Lightcuts算法基于二叉树进行聚类,其关键在于绘制时如何高效地在二叉树中搜索"光源割"。Condon等[11]基于Lightcuts算法提出了预计算的方案,提高了"光源割"的搜索效率,但由于存储开销较大,不适合大规模场景的绘制。Walter等[12]和Laurent等[13]还分别提出了双向Lightcuts算法和前向Lightcuts算法。王光伟等[14]利用空间的连续性改进了Lightcuts算法,基于空间聚类的思想减少了"光源割"的搜索计算,但当光源和场景中的几何体分布极不均匀时,其加速效果并不明显。

本文基于Lightcuts算法,采用二叉树森林代替传统Lightcuts算法中的二叉树结构,并提出自适应的视景体划分方案对三维场景进行剖分,通过构建包含"簇-光源"对的列表,实现了一种基于空间自适应剖分的Lightcuts多光源聚类算法,不仅能够快速剔除与当前渲染点无关的光源,而且能够利用相邻渲染点的"光源割"搜索路径,有效降低多光源环境下的计算量并减少绘制时间,提高多光源三维场景的绘制效率。

1 改进的Lightcuts多光源聚类算法

1.1 Lightcut算法原理

Lightcuts算法由多光源聚类和计算"光源割"2部分组成。

在光源的聚类部分,按照光源依据权值最小原则聚集成二叉树。2光源间的权值与距离、亮度相关,权值的算法见文献[10],本文不再赘述。构建过程是每

收稿日期:2017-02-21; **修回日期**:2017-03-12

基金项目:国家自然科学基金资助项目(61301233);吉林省自然科学基金资助项目(20130101069JC)

作者简介:袁昱纬(1988-),男,博士生。

次选取权值最小的2个光源，并选择其中1个作为该聚类的代表，然后将代表光源与其他剩下的光源一起继续搜索权值最小的光源，并加入到二叉树中，直到只有最后1个光源，完成聚类过程。该过程形成的层次化结构的二叉树称为光源树，聚类过程如图1所示。

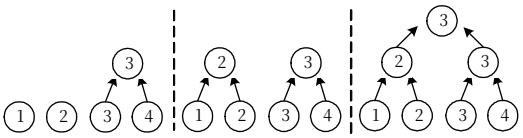


图1 Lightcuts算法的光源聚类过程示意图

Fig.1 Diagram of light source clustering process of lightcuts algorithm

在计算“光源割”部分，是在二叉树结构里找到场景绘制所需的代表光源。由于场景中某个几何体往往需要计算多个光源的光照，为避免遍历所有光源，Lightcuts算法根据韦伯－费德勒定理[15]，在阈值范围内，使用光源树中的一些光源聚类的代表光源，来减少参与计算的光源数目。在光源树的自顶向下搜索过程中，若某个节点的子节点的最大可能亮度在该节点的全部光源亮度中大于阈值范围的上限，则继续向该子节点搜索，直至找到小于阈值范围的子节点或到达叶子节点，这个过程叫作计算“光源割”。

计算“光源割”的过程中，需要先确定多光源的二叉树结构中某一节点 C 对渲染点 x 的亮度的贡献 $L_c(x)$，以及节点 C 的某个子节点光源理论上所能产生的亮度最大值 $E_{\text{upperbound}}$。

$$L_c(x,w)=\sum_{i\in c}M_i(x,w)G_i(x)V_i(x)I_i\approx M_j(x,w)G_j(x)V_j(x)\sum_{i\in c}I_i \text{。} \tag{1}$$

式(1)中：x 为渲染点；M 为材质因子；w 为视点方向；G 为几何因子；V 为可见性；I 为光源亮度；i 为实际光源；j 为代表光源；$\sum_{i\in c}I_i$ 为节点 C 包含光源的亮度之和。

$$E_{\text{upperbound}}=M_{\max}(x,w)G_{\max}(x)\sum_{i\in c}I_i \text{。} \tag{2}$$

式(2)中：$M_{\max}$ 为材质因子的上限；$G_{\max}$ 为几何因子的上限。

此外，由于 V 取上限值，则令 $V=1$，因此在式(2)中可以略去。

最后，根据韦伯－费德勒定理，渲染点 x 的亮度误差阈值 E_x 由下式确定：

$$E_x=\sum L_c(x,w)\times 0.01 \text{。} \tag{3}$$

式(3)中：$\sum L_c(x,w)$ 为节点 C 所含光源对渲染点的亮度之和；0.01为韦伯参数，这里也可以采用文献[12]为了更高效的光照计算而提出的0.02，具体可以由实际情况确定。

最后，通过判断 $E_{\text{upperbound}}$ 和 E_x 的大小关系来确定节点是否需要继续分裂。

1.2 采用二叉树森林取代单独的二叉树

在传统的基于Lightcuts算法的多光源聚类中，场景中的光源采用二叉树结构进行组织和聚类。然而在实际应用中发现，有些场景的光源数量虽然很多，但由于光源的影响范围有限，部分光源的影响范围之间并没有重叠，此时如果将它们依据权值最小原则聚类在二叉树结构中，虽然能够在后续计算“光源割”时再将它们分开，但是这需要额外的计算量。因此，为了减少二叉树结构的层级，同时适应后文提出的基于空间自适应剖分的多光源聚类算法，降低搜索时的计算量，本文采用二叉树森林代替单独二叉树对光源进行聚类：

1)按照传统Lightcuts算法找到权值最小的2个光源之后，不直接将其聚类组成二叉树，而是判断2个光源的权值是否大于某一阈值，该阈值根据场景实际情况，由两光源的影响范围恰好重叠时的权值进行确定；

2)若小于该阈值，则按照Lightcuts算法对两光源进行聚类，并指定代表光源；若大于该阈值，则不进行聚类，分别按照两棵二叉树进行组织；

3)重复上述聚类过程，直至所有代表光源之间的权值都大于阈值，则停止对光源二叉树的聚类，得到二叉树森林，如图2所示，二叉树森林中每棵二叉树的根节点都是该树各自的代表光源。

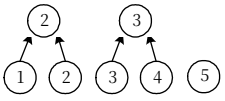


图2 采用二叉树森林的Lightcuts光源聚类

Fig.2 Lightcuts source clustering based on binary tree forest

1.3 采用视景体自适应空间剖分的多光源聚类

由于三维场景中可能存在的光源数量较多，且照射范围有限，经过基于Lightcuts的多光源聚类，可能得到包含较多棵二叉树的二叉树森林，也就是说聚类后仍然存在较多的光源或代表光源。为了在渲染时进一步快速剔除与渲染点无关的光源，同时利用场景的空间相关性，本文提出采用视景体自适应空间剖分的多光源聚类方法。

在绘制大规模三维场景时一般采用基于视点的

空间剖分方法，以快速剔除不必要的计算。但是如果采用均匀剖分的方法，会导致因为分辨率不能同时适应场景的所有区域，而产生较严重的欠采样或过采样现象。

本文借鉴文献[16]提出的场景剖分方法，进一步考虑到屏幕像素分辨率为 $r_{S1}\times r_{S2}$，且 r_{S1} 和 r_{S2} 可以不相等的情况，提出将视景体的自适应划分方法改进为：

$$\frac{C_{i+1}}{C_i}\leqslant kr_i+1,\ k=\frac{2\tan(\gamma/2)}{\max(r_{S1},r_{S2})}。\qquad(4)$$

式(4)中：C_i 为划分平面在深度方向 z 上的坐标；γ 为视景体的视角；第 i 个划分区域所对应光照计算的分辨率为 $r_i\times r_i$，r_{S1}、r_{S2}、r_i 和 γ 都是已知量。

特别地，视景体划分的初始点 $C_0=n$，划分的终点为场景的边界。

在确定每个划分区域对应的光照计算分辨率的情况下，可以基于式(4)的迭代关系，从视景体划分的起始点开始，自适应地计算出各个划分平面在视景体深度方向的位置坐标，且这种划分方式下的光照计算走样程度最小[16]，而不是根据经验来确定划分参数，如文献[17-18]。

基于视景体的场景分割示意图见图3。

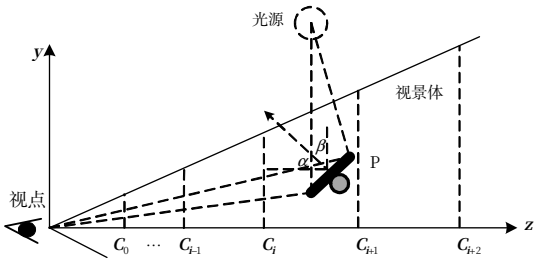


图3　基于视景体的场景分割示意图

Fig.3　Diagram of spatial subdivision based on view frustum

上述划分方法将视景体细分成不同深度的条带状空间，然后在与深度方向垂直的平面上进行规则二维网格分割，在空间中形成自相似的子视景体，完成对三维场景基于视景体的自适应空间剖分。这些剖分形成的子视景体在本文中称为簇，如图4所示，其中蓝色的簇表示其包含几何体，黄色范围表示光源的光照范围。

在完成上述视景体的自适应空间剖分后，本文将场景中的所有几何体按簇进行组织，确定视景体中哪些簇包含几何体。一旦某个簇确定被几何体占用，则给该簇分配光源或光源树。如果存在某个光源或光源树能够对该簇产生影响，即该簇位于光源的影响范围内，就产生一个“簇－光源”对，完成一次采用视景体自适应空间剖分的光源(树)聚类，如图4中与簇 C_0 关联的光源(树)包括 L_0 和 L_1。继续上述光源(树)的聚类过程，直至所有簇都完成与光源(树)的关联。每个簇包含几何体的数量和分辨率则可以通过视景体细分时簇的大小来调节。

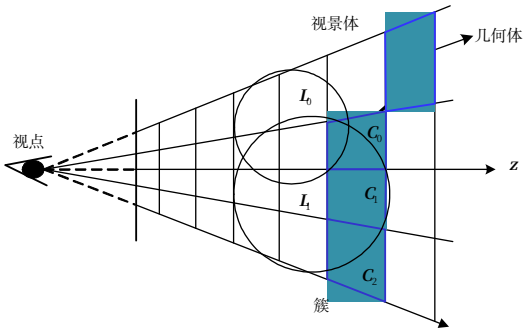


图4　视景体细分和光源分配示意图

Fig.4 Diagram of view frustum subdivision and light source distribution

为记录每个簇的“簇－光源”对，本文建立一个包含“簇－光源”对的列表，图5所示为图4的“簇－光源”对列表。应注意的是，某些簇可能被分配不止1个光源(树)，某些簇也可能由于位于光源范围之外，不存在“簇－光源”对，并且这些“簇－光源”对支持并行地生成。

C_0	L_0	L_1	C_1	L_1	C_2	L_1	……

图5　“簇－光源”对列表示意图

Fig.5　Diagram of “cluster-light” pair list

上述包含“簇－光源”对应关系的列表将每个簇与所有可能影响其包含几何体的光源联系在一起，同时为视景体区域的几何体提供一个尽可能小的、能够对其产生影响的光源集合。因此，在对场景中的几何体进行光照计算和渲染绘制时，通过列表查找簇对应的光源(树)，仅须采用相关的光源(树)对簇中的几何体进行光照和阴影计算，无关光源无须计算即可快速剔除，从而有效避免遍历无关的光源，提高场景绘制时的光照计算效率。

1.4　基于空间剖分的Lightcuts“光源割”计算

在通过“簇－光源”对列表排除无关光源后，需要针对各个簇对应的光源树采用Lightcuts算法搜索渲染点的“光源割”。由式(1)、(2)可以看出，渲染点的材质、可见性和几何特征等属性决定了其 $L_c(x,w)$ 和 $E_{\text{upperbound}}$，而绝大部分三维场景具有空间的相似性，也就是说空间区域相邻的渲染点具有相似的属性信息，

计算得到的“光源割”也是相似的。因此,本文利用空间相似性,在计算光源割时,采用共享相邻渲染点“光源割”路径的思路,提高“光源割”的计算速度。

在对三维场景进行分割时,本文采用第1.3节的视景体自适应剖分方案。对于剖分过程的某个格网,如果其中的所有渲染点在材质和几何特征等方面都很相似,则不继续剖分;如果其中的渲染点在材质和几何特征等方面有差异,则继续细分,直至达到分辨率要求。在对渲染点是否相似的判定上,本文约定如果2个渲染点具有相同的材质和可见性,且模型面片的法线方向变化量不超过10°,则认为它们是相似的。

由于场景进行了基于相似性的空间剖分,簇内的渲染点在计算“光源割”时可以利用簇内的相似性,减少公共“光源割”的计算,只计算2个光源割的差异部分,以此来减少“光源割”的计算量。具体方法如下:

1)在同一个簇内,计算渲染点的“光源割”时共享同一个存储堆栈;

2)如当前渲染点是该簇内的第一个渲染点,即初始点时,由式(1)、(2)计算相关渲染点的亮度上限值和阈值,得到“光源割”,计算方法与Lightcuts算法相同,并将“光源割”的搜索路径存放在堆栈中;

3)如果在当前点渲染时堆栈中已存在“光源割”,则以堆栈中上一次计算的“光源割”为初始“光源割”,采用回溯或继续搜索的方式计算当前渲染点的“光源割”,并将最新的“光源割”压入堆栈中存放;

4)循环3),直至该簇内所有渲染点计算完毕。簇内所有渲染点的“光源割”都可以在堆栈中获得。

因此,当第一个渲染点正常计算其“光源割”并压入堆栈后,该簇内接下来的渲染点可以利用上一次“光源割”的搜索路径继续进行阈值判断,确定是否需要继续向下分裂,并对“光源割”进行动态更新,而不需要每次都重新搜索整个二叉树。同时,由于簇内的空间相似性,大部分搜索路径是相同的,因而能够减少在计算“光源割”时的大量搜索运算。

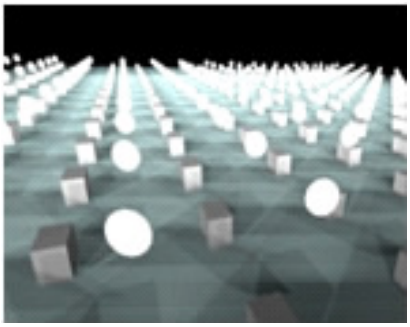
但是,在利用上一次计算的“光源割”时,会发现上一次计算的“光源割”可能已经满足阈值要求,没有必要继续分裂,甚至当前节点的父节点或更高层级的节点也都已满足阈值要求,上面若干层节点都没有必要分裂。在这种情况下,虽然不会降低渲染的质量,但是会使参与光照计算的光源数量增加,增大渲染时的计算开销。为了尽量避免这种情况的发生,本文提出在计算当前渲染点的“光源割”时,首先判断堆栈中“光源割”的分割情况。如果欠分割,则继续对其进行分割,考虑其子节点所代表的光源;如果过分割,则进行回溯,回溯到的节点层级需要满足:① 该层级节点的父节点大于式(3)所述的误差阈值;② 该层级节点恰好能满足阈值要求。

2 实验结果与分析

本文实验平台的软硬件环境为:CPU Intel Core i7-6700 3.4 GHz,RAM 32 GB,NVIDIA GTX 1070,Windows 7 ultimate x64操作系统。实验素材包括3个多光源场景:Corridor1、Corridor2和Cubes场景。Corridor1场景包含30个静态光源,Corridor2场景包括30个静态光源和2个在空间中匀速运动的动态光源,Cubes场景包含256个光源,光源均匀分布在场景上方,且照射范围相同。上述实验场景渲染后的效果如图6所示。

a)Corridor1场景
a)Corridor1 scene

b)Corridor2场景
b)Corridor2



c)Cubes场景
c)Cubes scene

图6 实验场景
Fig.6 Test scene

将本文提出的基于空间自适应剖分的Lightcuts算法与传统Lightcuts算法对比，分别构建二叉树森林和二叉树结构。在计算“光源割”的过程中，实验根据韦伯－费德勒定理，亮度的误差阈值的系数选取为0.02。分别计算渲染点在对应“光源割”中的平均节点数量、平均搜索步数和绘制所需时间，如表1所示。渲染时，每个场景在屏幕空间的分辨率设定为$1K\times1K$，不使用GPU的并行计算。

表1　本文提出的改进方法与传统方法的对比

Tab.1　Performance comparison between traditional lightcuts algorithm and our method

场景	算法	光源割平均节点数	平均搜索节点数	绘制时间/ms
Cubes	Lightcuts算法	6.32	12.64	930
	本文改进算法	6.35	7.32	606
Corridor1	Lightcuts算法	10.03	20.06	589
	本文改进算法	10.11	13.90	422
Corridor2	Lightcuts算法	15.26	30.52	313
	本文改进算法	15.32	20.29	215

从表1中可以看出，虽然光源割的平均节点数变化不大，但是在平均搜索步数方面，本文提出的改进的Lightcuts算法与传统Lightcuts算法相比，在3个实验场景中分别减少了42.09%、30.71%和33.52%，平均绘制时间分别减少了34.84%、28.35%和31.31%。这是因为本文算法能够利用空间自适应剖分在簇内形成的相关性，避免对每个渲染点都进行完整的“光源割”计算，减少了相同路径的重复搜索，降低了计算开销，具有很好的加速效果，同时“簇－光源”对列表能够有效剔除无关光源对绘制速度的影响，缩短绘制的时间。

Cubes场景中的单个光源影响范围较小，光源之间重叠的数量也不多，在对该场景进行本文改进的Lightcuts聚类时，二叉树森林中二叉树的数量较多，每个光源树的层级较少，因此“光源割”的节点数和搜索节点数较少，过分割的节点数也相对较少，效率提高最明显，但由于Cubes场景中光源总数最多，场景规模最大，其绘制消耗的总时间最多。而另外2个场景Corridor1和Corridor2中光源相互重叠较多，二叉树森林中二叉树的数量较少，在每棵二叉树中需要搜索的节点数量就相对较多，本文改进算法对绘制效率的提高幅度相对较小。

为验证本文提出的基于空间自适应剖分的Lightcuts算法在利用空间相关性方面的优势，除传统Lightcuts算法，本文还与同样基于空间相关性的Lightcuts改进算法进行了比较，包括文献[14]提出的基于空间聚类增强的Lightcuts算法（S.C.Lightcuts）和文献[19]提出的Coherence Lightcuts算法（C.Lightcuts）。在实验过程中，以Cubes场景为例，视点飞过场景上方，分别统计3种Lightcuts改进算法的平均搜索节点数和绘制时间，并计算它们相对于传统Lightcuts算法的减少率，如图7、8所示，图中的横坐标为在场景中连续变化的视角对应的每一帧，纵坐标分别为搜索节点数量的减少率和绘制时间的减少率。减少率的计算式为：

$$R=\left(N_{\mathrm{b}}-N_{\mathrm{a}}\right)/N_{\mathrm{b}}\ 。\tag{5}$$

式(5)中：N_{a}为对比算法的相关指标（如搜索节点数量和绘制时间）；N_{b}为传统Lightcuts算法的相关指标；R为减少率，R越大说明相关指标减小越多，性能提高越大。

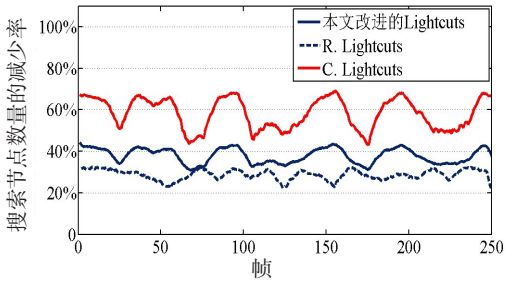


图7　搜索节点数量减少率随视点变化的情况

Fig.7　Reduction ratios of the number of nodes in search with the change of viewpoint

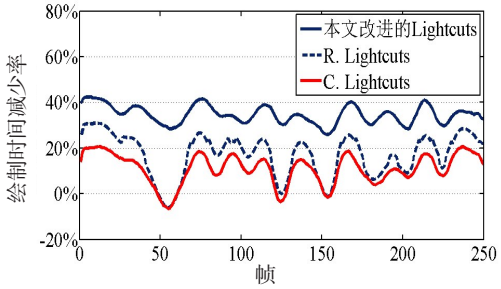


图8　绘制时间减少率随视点变化的情况

Fig.8　Reduction ratios of rendering time with the change of viewpoint

与S.C.Lightcuts算法相比，本文提出的改进算法在绘制时间和搜索节点数量方面都明显更优。这是因为该文献[14]的方法将屏幕空间分为若干像素块，每个像素块内部的渲染点的光照采用插值算法来快速求得，可以节省一定的光照计算时间，但是每个像素块需要分别建立局部“光源树”，并且需要在每个像素块的4个角分别计算“光源割”，因而文献方法在搜索节点数量上的减少并不明显。而本文算法不仅可以利用每个剖分区域内上一个渲染点的光源割，避免相同遍历路径的重复搜索，而且还优化了亮度贡献的计算，因而本文算法可以获得更高的搜索节点数量减少率和绘制时间减少率。

与C.Lightcuts算法相比，该方法在搜索节点数量方面明显优于本文算法，这是由于文献[19]对算法搜索过程进行了改进，通过对屏幕空间的剖分，将每个剖分单元看作一个整体计算“光源割”，剖分单元内部的所有渲染点仅需进行一次“光源割”的搜索计算，因而可以获得更高的搜索节点减少率，但该算法仅能减少搜索的计算，对于Lightcuts算法中的亮度贡献计算并没有进行改进，因而其绘制时间减少率仍然低于本文提出的改进Lightcuts算法。

综上所述，本文提出的基于空间自适应剖分的Lightcuts算法能够有效提高多光源环境下的三维场景绘制速度，且性能比较稳定。

3 结论

本文采用二叉树森林代替传统Lightcuts算法中的二叉树，并提出自适应的视景体划分方案对三维场景进行空间剖分，通过构建包含“簇－光源”对的列表，实现了一种基于空间自适应剖分的Lightcuts多光源聚类算法，不仅能快速剔除与当前渲染点无关的光源，且能有效利用空间聚类的相似性减少搜索“光源割”的计算量。

与传统方法比，本文的算法在计算“光源割”阶段能将搜索步数平均减少 30.71%~42.09% ，绘制时间平均缩短 28.35%~34.84% ，提高了三维场景渲染效率。

参考文献：

[1] SHIRLEY P, WANG C, ZIMMERMAN K. Monte Carlo techniques for direct lighting calculations[J]. ACM Transactions on Graphics, 1996, 15(1): 1-36.

[2] PAQUETTE E, POULIN P, DRETTAKIS G. A light hierarchy for fast rendering of scenes with many lights[J]. Computer Graphics Forum, 1998, 17(3): 63-74.

[3] 闫志. 面向可编程着色器的多光源加速算法研究与实现[D]. 济南：山东大学，2014.
YAN ZHI. Research and implementation of multi-light source accelerating algorithm for programmable shaders [D]. Jinan: Shangdong University, 2014. (in Chinese)

[4] 张薇薇，杨怿菲. 多光源并行化算法的实现[J]. 火力与指挥控制，2016，41(3)：111-115.
ZHANG WEIWEI, YANG YIFEI. Study and implementation of lighting parallelization algorithm[J]. Fire Control & Command Control, 2016, 41(3): 111-115. (in Chinese)

[5] HUO Y, WANG R, JIN S, et al. A matrix sampling-and-recovery approach for many-lights rendering[J]. ACM Transactions on Graphics, 2015, 34(6): 1-12.

[6] HOLLANDER M, RITSCHEL T, EISEMANN E, et al. ManyLoDs: parallel many-view level-of-detail selection for real-time global illumination[J]. Computer Graphics Forum, 2011, 30(4): 1233-1240.

[7] SUN C, AGU E. Many-lights real time global illumination using sparse voxel octree[C]//International Symposium on Visual Computing. Germany: Springer International Publishing, 2015: 150-159.

[8] OLSSON O, ASSARSSON U. Tiled shading[J]. Journal of Graphics Tools, 2011, 15(4): 235-251.

[9] OLSSON O, BILLETER M, SINTORN E, et al. More efficient virtual shadow maps for many lights[J]. IEEE Transactions on Visualization and Computer Graphics, 2015, 21(6): 701-713.

[10] WALTER B, FERNANDEZ S, ARBREE A, et al. Lightcuts: a scalable approach to illumination[J]. ACM Transactions on Graphics, 2005, 24(3): 1098-1107.

[11] CONDON T, WALTER B, BALA K. Precomputed & hybrid variants of lightcuts[R]. New York: Cornell University, 2010.

[12] WALTER B, KHUNGURN P, BALA K. Bidirectional lightcuts[J]. Acm Transactions on Graphics, 2012, 31(4): 1-11.

[13] LAURENT G, DELALANDRE C, DE LA RIVIERE G, et al. Forward light cuts: a scalable approach to real-time global illumination[J]. Computer Graphics Forum, 2016, 35(4): 79-88.

[14] 王光伟，谢国富，王文成. 基于空间聚类增强Lightcuts的光照计算[J]. 计算机学报，2013，36(11)：2364-2370.
WANG GUANGWEI, XIE GUOFU, WANG WENCHENG. Enhancing illumination computation of lightcuts with spatial coherence[J]. Chinese Journal of Computers, 2013, 36(11): 2364-2370. (in Chinese)

[15] BLACKWELL H R. Luminance difference thresholds [M]. Berlin: Springer International Publishing AG, 1972: 78-101.

[16] 陈国栋，叶东文，党琪琪. 混合场景中阴影绘制方法研究[J]. 计算机工程，2016，42(2)：242-248.
CHEN GUODONG, YE DONGWEN, DANG QIQI. Research on shadow rendering method in hybrid scene[J]. Computer Engineering, 2016, 42(2): 242-248.

（下转第198页）