基于八叉树自适应体归并的光线跟踪加速结构

袁昱纬[1,2]，全吉成[1,2]，吴 晨[1]，刘 宇[2]，王宏伟[2]

[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，吉林 长春 130022

摘要 针对光线跟踪算法计算量大和运行效率低的问题，提出了一种采用八叉树自适应体归并(OAVM)的光线跟踪加速结构。该结构将八叉树模型的空节点自适应地聚集为包围体，尽可能地减小了光线与空白节点的求交次数。基于 OAVM 的一种多级八叉树结构的特点，提出了采用 Morton 码对各层级的所有节点分别进行编码的算法，该结构所采用的存储方式和邻域查询算法有效减小了指针数量，避免了递归搜索。同时，该算法可以有效处理大规模动态场景的局部更新问题。基于 Liang-Barsky 算法，光线相交测试的计算速度得到提升。实验结果表明，和传统结构算法相比，所提出算法的指针总数平均减少了 54.45%，光线相交测试时间平均缩短了 52.37%，大幅加快了相交测试速度，提升了场景的渲染效率。

关键词 光计算；光学数据处理；光线跟踪；八叉树；自适应体归并；相交测试；邻域查询

中图分类号 TN29 **文献标识码** A

doi：10.3788/AOS201737.0120001

Ray Tracing Acceleration Structure Based on Octree Adaptive Volume Merging

Yuan Yuwei[1,2], Quan Jicheng[1,2], Wu Chen[1], Liu Yu[2], Wang Hongwei[2]

[1] *Department of Electronic and Information Engineering, Naval Aeronautical and Astronautical University, Yantai, Shandong* 264001, *China*;

[2] *Department of Aeronautic and Astronautic Intelligence, Aviation University of Air Force, Changchun, Jilin* 130022, *China*

Abstract In order to overcome the problems of large amounts of calculation and low operating efficiency of ray tracing algorithm, a ray tracing acceleration structure based on the octree adaptive volume merging (OAVM) is proposed. Through gathering blank nodes of an octree model as a bounding volume adaptively, this structure can reduce the intersection number between ray and blank nodes as much as possible. Based on the characteristic that OAVM is a multi-level octree structure, an algorithm with the Morton code to encode all the nodes at different levels is proposed. The storage method and neighborhood search algorithm used in this structure can reduce the amount of pointers and avoid the recursive search effectively. In the meanwhile, the algorithm deals with the problem of partial update a in large scale dynamic scene effectively. Based on the idea of Liang-Barsky algorithm, the calculation speed of intersection test for rays is improved. The experiment results indicate that, compared with traditional algorithms, the proposed algorithm can reduce the total number of pointers by 54.45% averagely. The time of ray intersection test is reduced by 52.37% averagely. The ray intersection test time is decreased and the scene rendering efficiency is improved.

Key words optics in computing; optical data processing; ray tracing; octree; adaptive volume merging; intersection test; neighborhood search

OCIS codes 200.4560; 110.1758; 100.2000

收稿日期：2016-07-05；**收到修改稿日期**：2016-07-21

基金项目：吉林省自然科学基金(20130101069JC)、军内武器装备重点科研项目(KJ2012240)

作者简介：袁昱纬(1988—)，男，博士研究生，主要从事装备仿真与虚拟现实方面的研究。

E-mail：yyw57156@hotmail.com

导师简介：全吉成(1960—)，男，博士，教授，博士生导师，主要从事三维可视化方面的研究。

E-mail：jicheng_quan@126.com(通信联系人)

1 引　言

光线跟踪算法通过模拟光的传播过程，实现了对三维场景的渲染，成为生成真实感图形的主要算法之一，现已广泛应用于三维场景建模与仿真[1]、视觉光学[2]、三维测量[3]等领域。光线跟踪算法涉及大量光线与场景的求交计算，计算量较大。主要通过提高求交速度、减少求交次数、加大光线采样间隔、采用并行算法等方式提高光线跟踪算法的效率[4]。八叉树、层次包围盒(BVH)、KD 树、空间二分搜索树(BSP)和均匀网格等都是常用的光线跟踪加速结构。此外，Maria 等[5]提出了约束凸空间划分(CCSP)，该方法需要完整的拓扑描述来构建 CCSP。李静等[6]将均匀网格创建为空盒，代替连续的空白区域，但是空盒的生成过程较为复杂。Navrátil 等[7]和 Hu 等[8]提出了利用图形处理器(GPU)的分布式并行渲染管线动态调度来提升求交效率。Zhou 等[9]从单指令多数据流(SIMD)指令集出发，研究光线跟踪。Nery 等[10]从硬件层面研发了光线跟踪算法的数字信号处理芯片。

在八叉树结构的加速算法方面，Yoder 等[11]提出了通过八叉树邻域分析表迭代计算邻域的方法，但该方法受邻域分析表的限制。Tian 等[12]提出了基于八叉树分解的自适应编码方法，但该方法仅用于场景压缩，未考虑场景的渲染效率。Namdari 等[13]提出了基于父节点邻域分析进行八叉树广度优先搜索，提高了叶子节点的遍历速度。Liu 等[14]利用八叉树栅格化提高了 GPU 的体绘制速度，采用了跳跃空区域的方案，但跳跃需依次进行。王文玺等[15]将指针八叉树用于光线跟踪算法，但未对其结构进行优化。张文胜等[16]运用八叉树的邻域分析，减少了算法的递归次数，但是未减少光线与空节点的相交测试次数。傅欢等[17]采用局部凸性和八叉树对点云数据进行分割，但其邻域关系的表达较为复杂。颜健等[18]通过推导运动累加数学模型，将光线跟踪过程转换为旋转运动或平移运动，避免光线与曲面联立方程组。Hornung 等[19]研究了基于 Morton 的八叉树节点编码方法，该方法较实用，但规则的存储方式增大了加速结构的存储空间。

本文基于八叉树节点编码的思路，结合空白节点的优化和跟踪光线的邻域查询方法，提出了改进的光线跟踪加速结构，即自适应体归并的多级八叉树结构(OAVM)。通过将八叉树的空节点自适应地聚集为包围体，跟踪光线可以更快地到达相交面片，减小了光线在空白区域求交的计算量。基于 Liang-Barsky 算法思想，提高了光线与包围体求交测试的效率。同时，改进的加速结构 OAVM 采用了多层结构，可较好地处理大规模动态场景的结构局部更新问题，从而减小了场景动态更新对绘制效率的影响，提高了场景绘制性能。

2 利用八叉树自适应体归并算法加速光线跟踪

2.1 自适应体归并的多级八叉树模型

将 OAVM 分为三个理论层次进行管理，分别是物理层、映射层和逻辑层。物理层即一维线性存储空间，可以为内存空间，也可以为物理磁盘。物理层的原理比较简单，可以利用操作系统进行管理。下面主要介绍映射层原理和逻辑层原理。

映射层采用 Morton 码对八叉树各层级的所有节点分别进行编码，节点的逻辑关系和节点存储位置之间的相对关系都隐含在节点的编码中。通过编码，可以加快节点查找的过程，提高光线跟踪效率，实现逻辑层与物理层之间的映射。同时，所有节点不再需要包含指向子节点和父节点的指针，每个叶子节点只需存储指向其包含面片的指针(对于实节点和灰节点)或指向该节点包围体的指针(对于空节点)。

逻辑层即 OAVM 的理论模型，采用分级管理策略构建 OAVM，并结合文献[20]提出的虚拟八叉树模型，按照 Z-Order 方法对八叉树每级的节点(包括空节点、灰节点和实节点)进行排序(即 Morton 码的大小顺序)和编码，并按这个顺序进行管理和存储，形成了一种类似于金字塔状的多分辨率结构，如图 1 所示，图中 L_i 表示八叉树的第 i 层。每一级对应一个存储空间，相对于下一级节点而言，上一级节点被称为节点块。利用八叉树自身的父子层级关系对节点进行归并，归并后节点的最小单元在逻辑层面上从某个节点提升为八叉树上一级的某个分块。

为了描述各个节点的逻辑位置与实际位置之间的关系，节点的编码可采用通用格网坐标(x，y，z) 的形式表示，并对不同层级的节点分别进行划分；节点的编码也可以采用节点层级加 Morton 码的形式(L，M)表示，其中 L 为节点所在的八叉树层级，M 为该节点在 L 层的 Morton 码。此外，这两种编码形式可以方便地进行相互转换[20]。

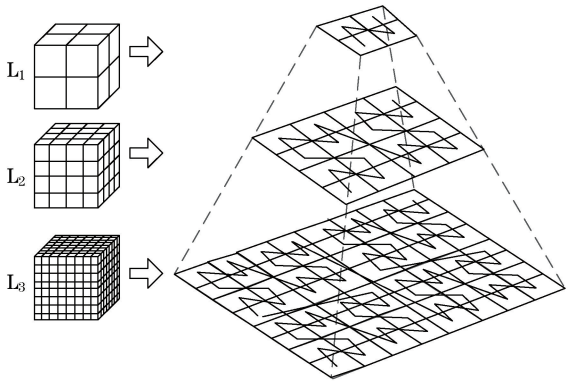

图 1　多级八叉树结构及各层 Morton 编码的二维示意图

Fig. 1　Two-dimensional diagram of multilayer octree structure and Morton encoding of each layer

进行归并操作后，空叶子节点不是指向每个空节点的边界，而是指向空节点归并后形成的包围体。包围体在八叉树归并过程中不断更新，且参加该包围体归并的节点都指向该包围体，不需要再重复存储，如图 2 所示。图 2 中第 2 层的方形节点 C 为叶子节点，第 1 层的方形节点 P 为第 2 层节点的父节点，箭头指向的圆表示该叶子节点的包围体或包含的模型面片，箭头表示指针。图 2(a)为归并前叶子节点的指针状态，图中空心圆为空叶子节点对应的空间边界，实心圆为叶子节点包含的面片。归并操作后，形成的有向无环图如图 2(b)所示，空心椭圆为归并后的包围体，图中的指针状态经过了调整。构建八叉树时，所有空叶子节点被初始化为八叉树规则剖分形成的空间边界，归并过程在下一节中讨论。

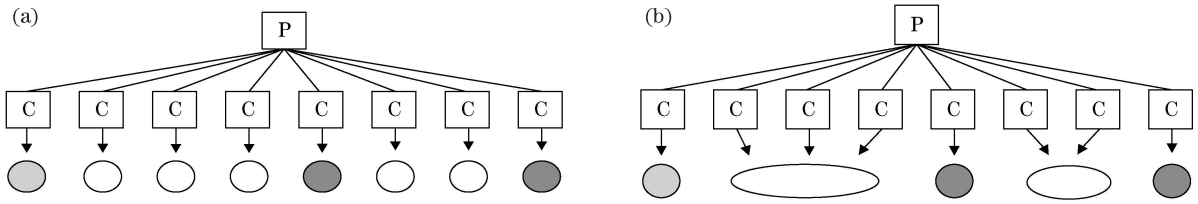

图 2　八叉树节点归并时叶子节点指针调整策略。(a) 归并前；(b) 归并后

Fig. 2　Pointer adjustment strategy for octree leaf node merging. (a) Before merging; (b) after merging

需要注意的是，所述的八叉树节点归并并不是节点本身的归并，而是节点指针所指向的包围体的归并。归并后指针指向新的包围体，丢弃每个节点原来的包围体或边界，八叉树的节点本身并不删除和归并。此操作虽然增加了一定的计算量，但是由于场景节点的归并可减少光线相交测试次数，因此有助于提升整个光线跟踪算法的性能，从而加快三维场景的生成速度。

2.2　场景节点的八叉树归并

由于三维场景中存在大量空节点，在进行光线与面片的相交测试前，需要先与若干空节点进行相交测试。为了减少光线与空节点的相交测试次数，在进行相交测试前，首先对空节点进行自适应体归并。

空节点即八叉树中不包含任何面片的叶子节点，对其进行归并时分为两种情况：1)同层空节点归并，即只对具有相同父节点的共面空节点(简称为同父空叶子节点)进行归并；2)邻层空节点归并，即将子节点全为空的节点朝父节点的方向进行归并。为方便描述，将归并后的节点分为 1 型节点(子节点全为空的节点)和 2 型节点(子节点不全为空的节点)。未进行归并操作的空叶子节点不属于这 2 种类型。

2.2.1　同父空叶子节点归并

同父空叶子节点归并过程的伪代码描述如下：

for（node i＝该父节点下所有叶子节点）
{if（node i＝空节点）&&（node i 节点未经过归并）
{num＝0；
for（node j＝node i 的兄弟节点中的所有共面节点）
{if（node j＝空节点）&&（node j 节点未经过归并）
{node i 节点与 node j 节点归并；
重新计算方向属性 D，按位进行与运算；
计算新的包围体，并更新指针；
num＋＋；}}}}
if（num＝8）归并后的节点类型为 1；
else 归并后的节点类型为 2。

2.2.2　邻层节点归并

在完成同父空叶子节点的归并后，将归并后的 1 型节点参照 2.2.1 节的方法向上一层级进行递归归并，以形成新的节点，直至出现 2 型节点。

在上述伪代码中，定义每个节点的初始方向属性 $D=D_1D_2D_3D_4D_5D_6$，其中 D_n（$n=1,2,\cdots,6$）分别为该节点的上、下、左、右、前、后方向同父的兄弟节点存在的情况。$D_n=0$，表示存在兄弟节点；$D_n=1$，表示不存在兄弟节点。例如图 1 中 L_1 层的右下角节点，其方向属性 $D=010110$。在归并过程中，对两节点的方向属性按位进行与运算，根据运算后结果不全为 0 来判定节点的共面性。同时，为避免产生过于狭长或过于复杂的包围体，对 2 型节点不再进行归并。

上述归并需要将两归并节点指向的包围盒进行合并，以形成新的包围体。新的包围体使用一系列平行于坐标轴的矩形面片进行表征，定义矩形面片为

$$\begin{cases} x=a \\ y_{\min}\leqslant y\leqslant y_{\max} \\ z_{\min}\leqslant z\leqslant z_{\max} \end{cases}, \tag{1}$$

$$\begin{cases} x_{\min}\leqslant x\leqslant x_{\max} \\ y=b \\ z_{\min}\leqslant z\leqslant z_{\max} \end{cases}, \tag{2}$$

$$\begin{cases} x_{\min}\leqslant x\leqslant x_{\max} \\ y_{\min}\leqslant y\leqslant y_{\max} \\ z=c \end{cases}, \tag{3}$$

式中 $x_{\min}$、$x_{\max}$为包围体所在区域在 x 方向的最小值和最大值；$y_{\min}$、$y_{\max}$为包围体所在区域在 y 方向的最小值和最大值；$z_{\min}$、$z_{\max}$为包围体所在区域在 z 方向的最小值和最大值；a、b、c 分别为矩形面片与 x、y、z 轴的交点坐标。

由这些矩形面片围成的区域即为新的包围体。在进行归并操作时，需将两归并节点中指向包围盒的指针指向新的包围体，不再存储原来的包围盒，但是并不删除八叉树节点本身，因此，在动态场景变化时八叉树的分辨率精度和整体结构不会受到影响。

在包围体归并的过程中，需要遍历所有的叶子节点。所有叶子节点数量为 $O(n)$，每个叶子节点至多访问 3 个相邻节点，因此包围体创建过程的时间复杂度为 $O(n)$。由于空叶子节点之间没有重叠区域，归并操作后的包围体之间也没有重叠区域，且归并后包围体个数一定小于归并前空叶子节点的个数，因此包围体创建过程的空间复杂度也为 $O(n)$。

2.3　跟踪光线与归并节点的相交测试

根据 2.1、2.2 节构造的八叉树归并结构 OAVM，在进行光线相交测试时，光线和归并后的包围体进行相交测试可一次跨过若干个空白节点，不需要与每个节点的边界进行相交测试，从而减少光线在空区域的相交

测试次数，提高跟踪算法的速度。图 3 为光线 P 穿过 OAVM 包围体的示意图，其中点 A、B、C、D、E 分别为光线与不同包围体的交点。

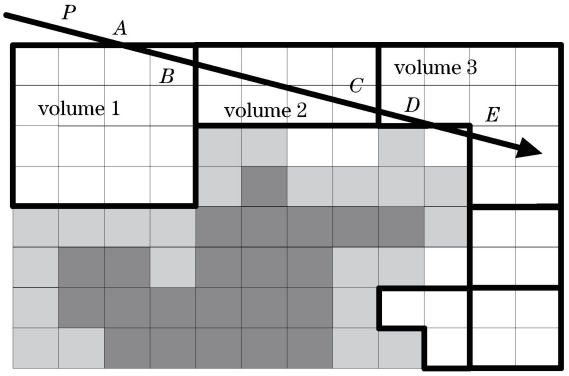

图 3　跟踪光线穿过归并后包围体的二维示意图

Fig. 3　Two-dimensional diagram of tracing rays passing through merged bounding volume

在进行光线与包围体相交测试时，采用先将光线与包围体各个面片分别求交再进行判断的方法比较简洁、易于实现，但是效率偏低。为了提高 OAVM 的加速性能，基于 Liang-Barsky 算法的思想，将三维相交测试计算转化为一维计算。光线 P 的参数方程形式为

$$\begin{cases} x = x_0 + iu \\ y = y_0 + ju \\ z = z_0 + ku \end{cases}, \tag{4}$$

式中(x_0, y_0, z_0)为光线源点的坐标，u 的范围为$[0,1]$，(i, j, k)为光线 P 的方向，i、j、k 分别为光线在 x、y、z 方向的照射距离。应保证 i、j、k 足够大，以使光线可以到达场景边缘。

根据包围体面片的表达形式，若光线上点(x, y, z)在包围体$(x_{\min}, x_{\max})$、$(y_{\min}, y_{\max})$、$(z_{\min}, z_{\max})$定义的区域内，则满足

$$\begin{cases} x_{\min} \leqslant x_0 + iu \leqslant x_{\max} \\ y_{\min} \leqslant y_0 + ju \leqslant y_{\max} \\ z_{\min} \leqslant z_0 + ku \leqslant z_{\max} \end{cases}。 \tag{5}$$

为方便表述，(5)式也可表示为

$$up_k \leqslant q_k, \tag{6}$$

式中 $k=1,2,\cdots,6$；$p_1=-i$，$p_2=i$，$p_3=-j$，$p_4=j$，$p_5=-k$，$p_6=k$；$q_1=x_0-x_{\min}$，$q_2=x_{\max}-x_0$，$q_3=y_0-y_{\min}$，$q_4=y_{\max}-y_0$，$q_5=z_0-z_{\min}$，$q_6=z_{\max}-z_0$。

按照光线与包围体的相对位置，可分为如下几种情况。

1) 当 $p_k=0$ 时，光线平行于 k 对应的包围体边界，其中 k 分别对应包围体的前、后、左、右、上、下边界。如果 k 同时满足 $q_k<0$，则光线完全在包围体之外，且与包围体无交点。

2) 当所有 $p_k\neq0$ 时，可通过计算参数 u(u_1 和 u_2)来判断光线与包围体的关系，$u=q_k/p_k$。u_1 取所有 q_k/p_k 中的最大值，即 $u_1=\max(q_k/p_k)$，其中 $p_k<0$。u_2 取所有 q_k/p_k 中的最小值，即 $u_2=\min(q_k/p_k)$，其中 $p_k>0$。当 $u_2>u_1$ 时，光线与包围体有交点，交点可以通过 u_1 和 u_2 计算，反之则没有交点。

基于 Liang-Barsky 算法的思想进行光线求交，计算量较小，可快速排除光线与包围体无交点的情况，无需依次遍历包围体的每个面。当 u_1 和 u_2 确定后，仅需一次计算即可得到光线与包围体的交点。将 Liang-Barsky 算法推广为 Cyrus-Beck 算法，可用于包围体多数为凸多面体的极复杂三维场景。由于 Cyrus-Beck 算法与 Liang-Barsky 算法类似，不再赘述。

2.4　光线跟踪与八叉树的邻域查询

求得跟踪光线与包围体的出射点后，该出射点即为下一个包围体的入射点。为了减少递归次数，采用改进的邻域分析方法。由于在进行八叉树归并的同时并未删除八叉树的节点，只是更改了结构中指针指向的

包围体，未破坏八叉树结构。因此，利用 2.1 节中的多级八叉树模型，在场景空间和八叉树节点编码之间建立空间映射关系，可以根据当前节点编码 N 和光线跟踪方向 $D(\alpha,\beta,\gamma)$ 等信息进行父子查询和邻域查询，其中 α、β 和 γ 为 -1、0 或 1。查询过程分为三步：1）根据光线出射点坐标，计算出射节点的编码（即 Morton 码）；2）根据光线的参数方程及其与当前包围体的关系，得到光线跟踪方向 D；3）基于提出的 OAVM 的邻域查询算法，查询光线进入的下一节点编码。由于八叉树节点本身并未被归并或删除，因此邻域查询结果指向的包围体即为下一个包围体，从而可直接进行下一包围体的相交测试。

所提出的具有层次性、方向性的快速八叉树邻域查询算法的步骤如下。

1）父节点查询。将当前节点编码 N 按二进制位截尾 3 位，即可得到当前节点的父节点编码 N'。

2）子节点查询。将当前节点编码 N 按二进制位左移 3 位，然后按 Z-Order 顺序与子节点在父节点的区域码 F 相加，即 $N'=8N+F$，$F=0,1,\cdots,7$，得到当前节点的子节点编码 N'。

3）邻域查询。对当前节点编码 N 和搜索方向 $D(\alpha,\beta,\gamma)$ 进行计算，得到当前节点在搜索方向 D 上的邻域节点 N'，计算式为

$$N'=N+4\alpha+2\beta+\gamma, \tag{7}$$

式中 α、β 和 γ 为 -1、0 或 1。

3　动态场景的加速结构局部更新

面对大规模动态场景时，加速结构更新与重建效率成为影响光线跟踪算法性能的关键因素。由于大规模场景的动态变化一般只发生在部分区域，而且变化的节点仅占整个八叉树结构的一部分，因此要求改进的加速结构针对局部区域的动态更新效率远高于整个加速结构的更新效率。

由于在 2.1 节中采用了多级八叉树管理策略，同时八叉树的节点归并操作并未删除节点本身，因此场景节点可方便地以上层的节点块为单位进行组织与索引。结合文献[21]中面向动态场景提出的双层结构，只需对存在变化的节点进行重新组织和构建，而不是整个八叉树结构，以减小动态场景更新的运算量和运算时间。这里只讨论空节点与实（灰）节点转换（即节点性质发生变化）时的加速结构更新方法。当节点性质未改变时，只需更新节点指向的内容，无需调整八叉树的归并结构。更新流程如下。

1）找到场景中节点性质发生变化的八叉树节点（如图 4 中的红色部分），如果是实节点转换为空节点，则进入步骤 2），反之进入步骤 3）。

2）当实节点变为空节点时，将空节点视为空叶子节点。若存在共面包围体，则进行归并操作。若归并后该节点的同父兄弟节点都为空节点，即产生 1 型节点，则继续向上一级归并，直至出现 2 型节点。

3）当空节点变为实节点时，将该节点从原来的包围体中删除，重新计算新的包围体，并更新原来的节点类型；还需将该层级从上一级中脱离出来，并更新上一级的节点类型，直至遇到 2 型节点，完成对原节点的更新。

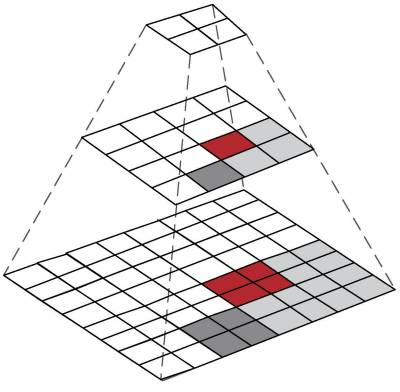

图 4　八叉树模型局部更新的二维示意图

Fig. 4　Two-dimensional diagram of partial update of octree model

4　实验结果与分析

实验平台的软硬件环境为：CPU Intel Xeon E5620 2.4 GHz * 2，RAM 48 GB，NVIDIA GTX580，Windows 2008 Server R2 64 位操作系统。实验样本包括两类：一类为斯坦福大学提供的 Bunny、Buddha 和 Dragon 场景；另一类为查尔姆斯理工大学提供的动画光线跟踪测试标准（BART）中的实验场景 Kitchen、Robot 和 Museum 3。第 1 类场景中只有一个较为规则的对象，第 2 类场景中均为非规则分布场景。对这些实验场景进行渲染，结果如图 5 所示，场景的基本信息如表 1 所示。

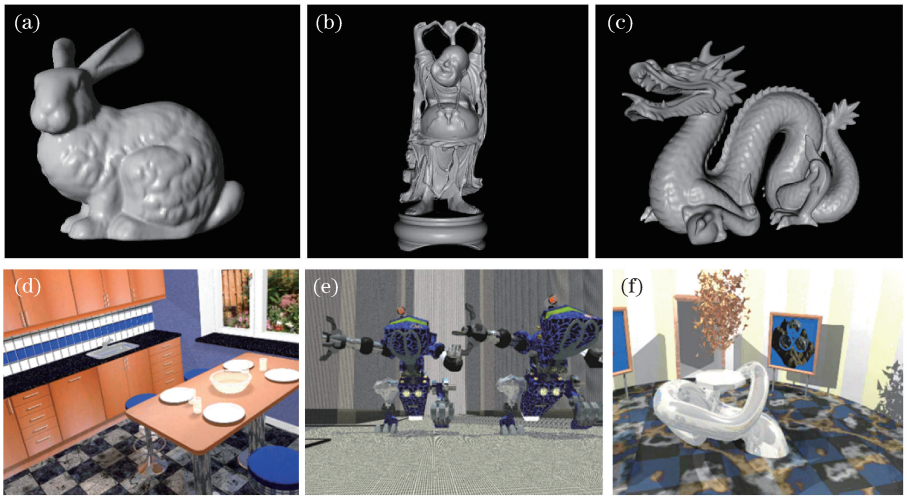

图 5　实验场景。(a) Bunny 场景；(b) Buddha 场景；(c) Dragon 场景；(d) Kitchen 场景；(e) Robot 场景；(f) Museum 3 场景

Fig. 5　Test scenes. (a) Bunny scene; (b) Buddha scene; (c) Dragon scene; (d) Kitchen scene; (e) Robot scene; (f) Museum 3 scene

表 1　实验场景的统计数据

Table 1　Statistical data of test scenes

Scene	Bunny	Buddha	Dragon	Kitchen	Robot	Museum 3
Number of faces	69451	1087716	871414	110561	71708	10143
Number of objects	1	1	1	393	1848	208

为测试改进的光线跟踪加速结构（即 OAVM）的性能，分别采用均匀空间格网、改进的 KD 树、经典的八叉树和 OAVM 4 种算法对以上场景进行实验，针对场景创建、光线求交和整体绘制三个方面进行性能测试。图 6 为 Bunny 场景和 Buddha 场景建立加速结构后的效果图。蓝色线框表示空叶子节点归并后形成的包围体，这些包围体与场景无交点；红色线框表示与场景有交点的叶子节点（实节点或灰节点），这些叶子节点没有进行归并操作。

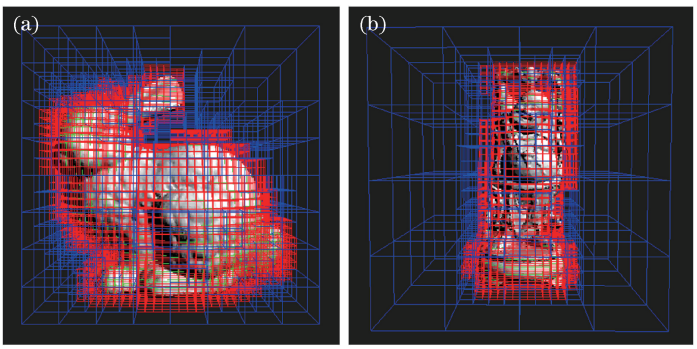

图 6　(a)Bunny 场景和(b)Buddha 场景中建立的 OAVM

Fig. 6　OAVM established in (a) Bunny scene and (b) Buddha scene

不同算法在创建场景方面的性能对比如表 2 所示，可以发现：在创建时间方面，本文算法优于 KD 树算法，且与空间格网算法和经典的八叉树算法相近；但是由于节点的归并操作，本文算法耗时较经典的八叉树

算法略长，节点总数未减少，指针总数大幅减少了 54.45%。这是由于在加速结构节点的归并操作后未删除节点，同时采用了 Z-Order 编码对八叉树进行管理。

表 2　实验场景的创建性能

Table 2　Creation performance of test scenes

Scene	Index	Grids	KD-tree	Octree	OAVM
Bunny	Creation time /ms	821	1652	989	1014
	Total number of nodes	1.33×10^6	2.25×10^6	1.76×10^6	1.76×10^6
	Total number of pointers	3.90×10^5	7.90×10^5	7.10×10^5	3.30×10^5
Buddha	Creation time /ms	5747	16581	6083	6427
	Total number of nodes	2.62×10^7	3.33×10^7	3.36×10^7	3.36×10^7
	Total number of pointers	5.85×10^6	7.59×10^6	8.29×10^6	3.79×10^6
Dragon	Creation time /ms	7355	12806	8817	9765
	Total number of nodes	2.40×10^7	2.75×10^7	3.12×10^7	3.12×10^7
	Total number of pointers	4.98×10^6	6.58×10^6	6.90×10^6	3.12×10^6
Kitchen	Creation time /ms	1574	2274	1715	1941
	Total number of nodes	3.45×10^6	6.61×10^6	5.48×10^6	5.48×10^6
	Total number of pointers	9.80×10^5	1.39×10^6	1.75×10^6	8.10×10^5
Robot	Creation time /ms	958	2351	1059	1175
	Total number of nodes	2.09×10^6	3.50×10^6	2.72×10^6	2.72×10^6
	Total number of pointers	5.10×10^5	1.03×10^6	1.25×10^6	5.70×10^5
Museum 3	Creation time /ms	330	532	412	441
	Total number of nodes	9.20×10^5	1.67×10^6	1.19×10^6	1.19×10^6
	Total number of pointers	1.80×10^5	2.20×10^5	2.50×10^5	1.10×10^5

构建不同的加速结构对 6 个实验场景分别进行光线与场景的相交测试实验，并对单根光线访问的平均节点数和相交测试所需时间这 2 个参数进行测试，测试结果如图 7、8 所示。从图 7、8 可以看出，对单根光线进行相交测试时，本文算法所需访问的节点数明显小于其他几种方法，且光线到达模型面片的相交测试次数较少。随着场景规模的扩大，访问的节点数并未同比例增加，而是保持相对稳定的状态，所需的相交测试时间与均匀空间网格、KD 树和八叉树算法相比分别平均提高了 50.79%、45.24%、52.37%，大幅加快了光线的相交测试速度。

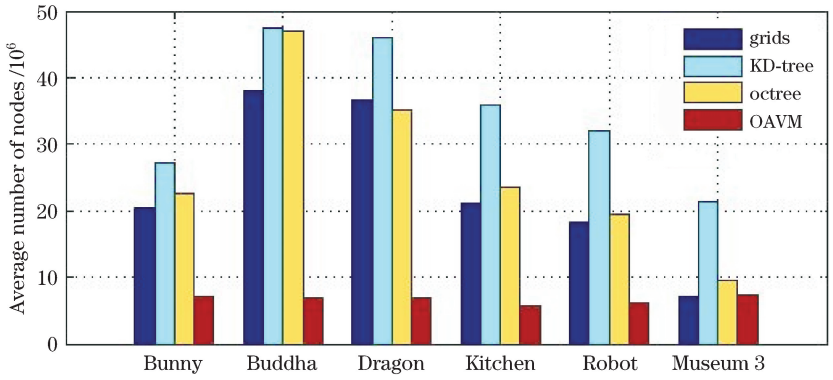

图 7　不同实验场景单根光线访问的平均节点个数

Fig. 7　Average node number of a single ray passing through different scenes

为检测不同动态场景下本文算法在不同分辨率时的处理效率，对 BART 中的 3 个实验场景分别以不同的分辨率进行解析和渲染，同时加入了 Rain 场景。通过在场景中均匀随机产生大规模下落的小球对 Rain 场景进行模拟(目标数为 15 M)。帧频 1、帧频 2、帧频 3 分别对应绘制分辨率 1024×768、800×600、400×300，帧频单位为 frame/s。在解析和渲染的过程中，均未使用 GPU 计算，绘制时考虑纹理、阴影和二次衍生光线。相关统计数据如表 3 所示。

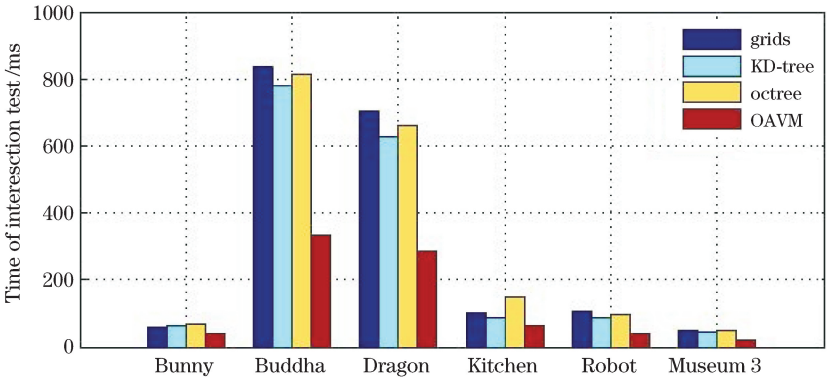

图 8　不同实验场景相交测试所需时间

Fig. 8　Time of intersection test in different scenes

表 3　实验场景的绘制性能

Table 3　Rendering performance of test scenes

Scene	Frame rate /(frame/s)		
	1	2	3
Kitchen	3.1	4.6	11.8
Robot	3.9	5.3	16.6
Museum 3	5.6	6.8	20.6
Rain	1.5	2.3	5.8

从以上数据可以看出，本文算法在未使用 GPU 优化的条件下，依然可以有效地绘制出动态场景。绘制效率随着场景分辨率的增加而降低。但是，面对大规模随机运动的场景，如 Rain 场景，本文算法绘制效率不高，这是由于场景中全部目标都随机运动时，无论哪种算法都要重构几乎整个加速结构，因此本文算法优势并不明显，但是，通过对场景中的目标进行运动估计，可以提高本文算法的效率。本文所提出的方法减小了光线相交测试时访问空节点的计算量，同时减少了动态场景加速结构的更新时间，在动态场景的绘制中具有较好的光线跟踪加速作用。

5　结　　论

提出了一种采用八叉树自适应体归并的光线跟踪加速结构 OAVM，通过构建多级八叉树模型，并通过自适应地聚集空节点、基于 Liang-Barsky 算法的相交测试和改进的邻域查询等过程，减少了跟踪光线到达相交面片的步数，加快了相交测试的运算速度，同时避免了递归搜索。与现有方法相比，本文算法的指针总数平均减少了 54.45%，光线相交测试时间平均缩短了 52.37%，提高了光线跟踪算法的性能。八叉树的多层结构考虑了动态场景的局部更新问题，这种类似于金字塔状的多分辨率结构减小了场景更新对整体结构的影响，在面对包含 110 个面片的动态场景时，仍可以获得较高的帧频，满足动态场景的实时绘制要求。将提出的光线跟踪加速结构应用于视觉光学、三维测量等领域，同样可以获得较高的计算效率。随着可编程 GPU 技术的发展，研究者们越来越重视并行计算在光线跟踪领域的应用。因此下一步的工作应当考虑以并行处理的方式进一步改进光线跟踪算法及其加速结构，使其在 GPU 上达到最优配置。

参　考　文　献

[1] Cai Xun, Zeng Liang, Liu Guangguo. Survey of ray tracing in volume rendering[J]. Computer Engineering and Design, 2009, 30(21): 4956-4959.
蔡　勋, 曾　亮, 刘光国. 光线跟踪方法在体绘制中的应用与发展[J]. 计算机工程与设计, 2009, 30(21): 4956-4959.

[2] Luo Han, Yuan Changying. Retroreflective performance analysis of cube corner membrane structure[J]. Acta Optica Sinica, 2015, 35(3): 0323001.
罗　汉, 袁长迎. 立方角锥型膜结构的逆反射特性计算[J]. 光学学报, 2015, 35(3): 0323001.

[3] Wu Shuangqing, Zhang Yin, Zhang Sanyuan, *et al*. Analysis of three-dimensional measurement system and the coordinates calibration in Fourier transform profilometry[J]. Acta Optica Sinica, 2009, 29(10): 2780-2785.
吴双卿, 张　引, 张三元, 等. 傅里叶变换轮廓术物体三维形貌测量的系统分析及其坐标校准方法[J]. 光学学报, 2009, 29(10): 2780-2785.

[4] Walter B, Drettakis G, Greenberg D P. Enhancing and optimizing the render cache[C]. Proceedings of the 13th Eurographics Workshop on Rendering, 2002: 37-42.

[5] Maria M, Horna S, Aveneau L. Constrained convex space partition for ray tracing in architectural environments[J]. Computer Graphics Forum, 2016, DOI: 10.1111/cgf.12801.

[6] Li Jing, Wang Wencheng, Wu Enhua. Ray tracing of dynamic scenes by managing empty regions in adaptive boxes[J]. Chinese Journal of Computers, 2009, 32(6): 1172-1182.
李　静, 王文成, 吴恩华. 基于空盒自适应生成的动态场景光线跟踪计算[J]. 计算机学报, 2009, 32(6): 1172-1182.

[7] Navrátil P A, Fussell D S, Lin C, *et al*. Dynamic scheduling for large-scale distributed-memory ray tracing[C]. Eurographics Symposium on Parallel Graphics and Visualization, 2012: 61-70.

[8] Hu W, Huang Y, Zhang F, *et al*. Ray tracing via GPU rasterization[J]. Visual Computer, 2014, 30(6-8): 697-706.

[9] Zhou P, Meng X. SIMD friendly ray tracing on GPU[C]. International Conference on Computer-Aided Design and Computer Graphics, 2011: 87-92.

[10] Nery A S, Nedjah N, França F M G. Efficient hardware implementation of ray tracing based on an embedded software for intersection computation[J]. Journal of Systems Architecture, 2013, 59(3): 176-185.

[11] Yoder R, Bloniarz P A. A practical algorithm for computing neighbors in quadtrees, octrees, and hyperoctrees[C]. Proceedings of the 2006 International Conference on Modeling, Simulation & Visualization Methods, 2006: 249-255.

[12] Tian J, Jiang W F, Luo T, *et al*. Adaptive coding of generic 3D triangular meshes based on octree decomposition[J]. The Visual Computer, 2012, 28(6): 819-827.

[13] Namdari M H, Hejazi S R, Palhang M. MCPN, octree neighbor finding during tree model construction using parental neighboring rule[J]. 3D Research, 2015, 6: 29.

[14] Liu B Q, Clapworthy G J, Dong F, *et al*. Octree rasterization: Accelerating high-quality out-of-core GPU volume rendering[J]. IEEE Transactions on Visualization & Computer Graphics, 2013, 19(10): 1732-1745.

[15] Wang Wenxi, Xiao Shide, Meng Wen, *et al*. Ray tracing algorithm based on octree space partition method[J]. Journal of Computer Applications, 2008, 28(3): 656-658.
王文玺, 肖世德, 孟　文, 等. 一种基于八叉树空间剖分技术的光线跟踪算法[J]. 计算机应用, 2008, 28(3): 656-658.

[16] Zhang Wensheng, Xie Qian, Zhong Jin, *et al*. Acceleration algorithm in ray tracing by the octree neighbor finding[J]. Journal of Graphics, 2015, 36(3): 339-344.
张文胜, 解　骞, 钟　瑾, 等. 基于八叉树邻域分析的光线跟踪加速算法[J]. 图学学报, 2015, 36(3): 339-344.

[17] Fu Huan, Liang Li, Wang Fei, *et al*. A point cloud segmentation algorithm using local convexity and octree[J]. Journal of Xi'an Jiaotong University, 2012, 46(10): 60-65.
傅　欢, 梁　力, 王　飞, 等. 采用局部凸性和八叉树的点云分割算法[J]. 西安交通大学学报, 2012, 46(10): 60-65.

[18] Yan Jian, Peng Youduo, Cheng Ziran, *et al*. Moving accumulative computation method for flux distribution of heat absorber in symmetry concentrating solar collector system[J]. Acta Optica Sinica, 2016, 36(5): 0508001.
颜　健, 彭佑多, 程自然, 等. 对称型太阳能聚光集热系统吸热器能流分布的运动累加计算方法[J]. 光学学报, 2016, 36(5): 0508001.

[19] Hornung A, Wurm K M, Bennewitz M, *et al*. OctoMap: An efficient probabilistic 3D mapping framework based on octrees[J]. Autonomous Robot, 2013, 34(3): 189-206.

[20] Lü Guangxian, Pan Mao, Wu Huanping, *et al*. Research on large virtual octree model for true three dimensional geoscience modeling[J]. Acta Scientiarum Naturalium Universitatis Pekinensis, 2007, 43(4): 496-501.
吕广宪, 潘　懋, 吴焕萍, 等. 面向真三维地学建模的海量虚拟八叉树模型研究[J]. 北京大学学报(自然科学版), 2007, 43(4): 496-501.

[21] Hapala M, Havran V. Review: Kd-tree traversal algorithms for ray tracing[J]. Computer Graphics Forum, 2011, 30(1): 199-213.