软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2011,22(10):2488−2496 [doi: 10.3724/SP.J.1001.2011.03927]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

线性复杂度的网格优化划分*

李 静+, 王文成

(中国科学院 软件研究所 计算机科学国家重点实验室,北京 100190)

Optimizing Grid Construction in Linear Complexity

LI Jing+, WANG Wen-Cheng

(State Key Laboratory of Computer Science, Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China)

+ Corresponding author: E-mail: lij@ios.ac.cn

Li J, Wang WC. Optimizing grid construction in linear complexity. *Journal of Software*, 2011,22(10): 2488−2496. http://www.jos.org.cn/1000-9825/3927.htm

Abstract: Uniform grid is one of the important spatial organization structures, which is widely used in many applications such as ray tracing, collision detection, path planning and so on. Its simplicity to compute makes it very suitable for processing dynamic scenes. Because its construction time, space requirement and application efficiency are closely related with the grid resolution, optimizing grid construction has always been an important topic in the world. Hense, a new optimization method for grid construction which ensures that both of the construction time and the space requirement are in the complexity $O(N)$, where N is the number of the primitives in the scene is proposed. In the related applications, such as ray tracing, it can achieve high acceleration efficiency, comparable with the best acceleration structures to date, such as the *kd*-tree, while reducing the construction time dramatically. These have been approved by the experimental results.

Key words: grid; ray tracing; dynamic scene; large scale scene; resolution

摘 要: 均匀网格划分是一种重要的场景空间组织结构,在光线跟踪绘制、碰撞检测、路径规划等方面有着广泛的应用.特别是由于其计算简单,很适合动态环境的处理.由于该结构的创建时间、空间需求和应用效率与网格分辨率密切相关,优化的网格划分一直是国际上探讨的重要问题.对此,提出一种新的优化划分方法,确保该结构的创建时间和空间需求都是 $O(N)$复杂度的.这里,N 是场景的面片数.同时,在相关的应用计算方面,比如光线跟踪,可与目前最好的加速计算结构相媲美.实验结果表明,该优化划分方法所产生的层次网格结构具有与当前主流的加速结构 kd 树相当的加速效率,且大幅降低了创建时间,优于已有的类似工作.

关键词: 网格;光线跟踪;动态场景;大规模场景;分辨率

中图法分类号: TP391 **文献标识码**: A

三维均匀网格是一种常用的场景空间组织结构.由于其创建速度快、计算简便且空间复杂度不高,因此与其他层次化组织结构(如八叉树[1]、*kd* 树[2,3])相比,在处理动态物体和大规模场景时具有更好的适应性.但是,均匀网格的应用计算效率在很大程度上取决于其划分的分辨率.合适的分辨率可以使均匀网格的效率优于其他

* 基金项目: 国家自然科学基金(60873182, 60773026, 60833007)
收稿时间: 2009-11-19; 修改时间: 2010-03-05; 定稿时间: 2010-08-13

空间组织结构,甚至是层次自适应结构[4];而不恰当的分辨率则会导致时间和空间开销的急剧上升,这些问题对于层次化的均匀网格结构显得尤为突出[1].不失一般性,下面我们主要围绕光线跟踪绘制来讨论网格的优化划分,因为其他方面的应用也主要是利用这样的结构进行类似的求交计算.

目前,优化网格划分分辨率的工作可分为两类:一类是基于代价预估模型的[5−8],另一类是基于经验公式的[9−13].前者的理论估算模型是在一些场景假设前提下得到的,使用时需要对场景类型做出预先分类,因而不便在实践中广泛使用;而后者根据统计经验得到,计算简单,是实践中比较常用的方法.根据文献[9]的研究,由于三角化后的物体表面面片分布均匀、大小相似,大多数面片朝向场景包围盒的最大面,因此可计算 3 个坐标轴方向上的网格分辨率 M_1,M_2,M_3 分别为 $M_3=\left[\sqrt[3]{\frac{NL_1^2}{L_2L_3}}\right],M_2=\left[\sqrt{\frac{NL_2}{M_3L_3}}\right],M_1=\left[\frac{N}{M_2M_3}\right]$,其中,方括号为取整运算,$L_i$ 为场景包围盒在第 i 个轴向上的长度,N 为场景中的面片数目.由于立方体具有最小的表面积与体积比,因此近立方体形网格单元能够使光线跟踪代价最小.所以,许多研究都将分辨率设定为

$$M_i=L_i\sqrt[3]{\frac{\lambda N}{V}},i\in\{x,y,z\} \tag{1}$$

这里,V 为一个网格占据空间的体积,λ为调整参数.对于不同的场景和光线跟踪计算的实现,λ的取值不尽相同.Shirley[10]认为,λ的值在 2~10 之间,其缺省设置为 8.Wald[11]得到的经验最优值为 5.Ize[7]取得的实验值为 8.Lagae[12]按照光线跟踪总时间的优化计算,将λ取值为 4.在最新的工作中,Kalojanov[13]也将λ设置为 5.

虽然基于经验的网格分辨率优化计算方法已取得了很好的效果,但由于实践中场景包围盒的长宽高尺寸的差别一般比较大,场景中面片尺寸的差异也会比较大,因此这样计算的分辨率会使网格划分的时间和空间复杂度超过 $O(N)$,从而降低了应用计算的效率.为此,我们在这类工作的基础上提出新的分辨率优化计算方法,以有效处理场景包围盒的长宽高尺寸差异较大及面片尺寸差异较大的问题,确保均匀网格划分的时间和空间复杂度都是 $O(N)$的,以更好地满足应用需求.进一步地,我们对层次化的网格划分进行相应的处理,自适应地约束过度的网格划分和细分层次深度.实验结果表明,基于我们工作所得的网格结构优于已有的类似工作,且其层次化的结构在加速光线跟踪计算方面与目前最有效的 *kd* 树结构相当,但大幅降低了创建时间.

本文第 1 节介绍线性复杂度的均匀网格优化划分方法.第 2 节详述层次网格的优化方法.第 3 节进行实验分析.第 4 节进行总结.

1 线性复杂度的网格优化划分

在实践中,很难保证场景包围盒的长宽高的尺寸差不多,也难以保证场景面片的尺寸差不多.这样,按照公式(1)计算的网格分辨率就会使网格划分的时间复杂度和空间复杂度超过 $O(N)$,因而降低应用计算的效率.下面,我们分别对这两种情况进行讨论且给出解决办法,并由此得到新的网格优化划分方法.实验结果表明,新方法对于大多数模型都是有效的.

1.1 根据包围盒尺寸自适应地调整网格分辨率

当根据公式(1)计算场景包围盒各个轴向上的网格分辨率时,可能会使某些轴向上的分辨率小于 1.由此,经过取整计算后,将会使网格单元数量急剧增加.不失一般性,设场景包围盒最短边所在轴向 i 上的分辨率为 M_i.当 $M_i<1$ 时,经过取整后,$M_i'=1$.这就相当于该方向上的单元数增加 $1/M_i$ 倍,从而使总网格单元数也增加 $1/M_i$ 倍,或许会极大地超过预期的λN 个.例如,设某场景 N=1000,λ=1,场景包围盒长宽高分别为 1 000,1 000,1.此时,若按照公式(1)计算,则得到 3 个轴向上的分辨率分别为 100,100 和 0.1.取整后分别为 100,100 和 1.这样,实际总网格单元数为 10 000,超出原计划的λN 达 10 倍.若考虑极端情况,即某轴向上的边长很接近 0,则情况会更糟.在实际应用中,可能出现这种情况的典型场景,如:一大片面积很大的草地;或者一块很大的地面上,许多小生物在上面行走等等.此外,在某些类型的层次网格(如 HUG[14])以及网格与层次包围盒的混合结构的形成过程中,即便是进一步均匀了网格划分的局部物体集合,也很可能具有极度不平衡的包围盒尺寸.

对此,我们相应地调整了其他轴向上的网格分辨率,以使得网格单元总数仍保持在λN 左右.即:当两个轴向上的分辨率均小于 1 时,则将这两个轴向上的分辨率均设置为 1,而另一轴向上的分辨率改为λN;当只有一个轴向上的分辨率小于 1 时,则将这个轴向上的分辨率设置为 1,而使另外两个轴向上的分辨率的乘积为λN.具体算法如下所述:

算法 1.

设场景包围盒 3 个轴向上的长度分别为 L_i,L_j 和 L_k,其中,$i\in\{x,y,z\}$,$j\in\{x,y,z\}$,$k\in\{x,y,z\}$,且 $i\neq j\neq k$.将它们由大到小地顺序排列,不妨设为 $L_i\geqslant L_j\geqslant L_k$,则,

(1) 若 L_i,L_j,L_k 均为 0,则 $M_i=M_j=M_k=1$;

(2) 若 $L_i>0$,但 $L_j=0$ 且 $L_k=0$,则 $M_i=\lambda N$,$M_j=M_k=1$;

(3) 若 $L_j>0$,但 $L_k=0$,则 $M_i=\left\lfloor\sqrt{\lambda N\frac{L_i}{L_j}}+0.5\right\rfloor$,$M_j=\left\lfloor\sqrt{\lambda N\frac{L_j}{L_i}}+0.5\right\rfloor$,$M_k=1$.此时,若 $M_j<1$,则按照步骤(2)重新计算 M_i,M_j,M_k 的值;

(4) 若 $L_k>0$,则 $M_i=\left\lfloor L_i\sqrt[3]{\frac{\lambda N}{V}}+0.5\right\rfloor$,$M_j=\left\lfloor L_j\sqrt[3]{\frac{\lambda N}{V}}+0.5\right\rfloor$,$M_k=\left\lfloor L_k\sqrt[3]{\frac{\lambda N}{V}}+0.5\right\rfloor$.此时,若 $M_k<1$,则按照步骤(3)重新计算 M_i,M_j,M_k 的值.

1.2 根据面片尺寸的差异调整网格分辨率

当将场景包围盒划分为均匀网格后,各个网格单元要存储所有经过它的面片的指针.当面片比较大,跨越多个网格单元时,这些网格单元均要有指针指向这个面片,这样就会增加空间开销,并会影响应用光线跟踪的效率.由于公式(1)是在假设面片大小差不多的情况下被提出来的,它可能会使网格中指针数目超过 $O(N)$的复杂度.例如,对一个含有 N 个面片的场景,设场景包围盒为 1×1×1 的立方体.那么按照公式(1),可以建立一个 $N^{1/3}\times N^{1/3}\times N^{1/3}$ 的均匀网格,每个网格单元大小皆为 $N^{-1/3}\times N^{-1/3}\times N^{-1/3}$.若这些面片皆与 xy 平面平行,且各面片的包围盒大小为 1×1×0,那么每个面片都会占据 $N^{1/3}\times N^{1/3}$ 个网格单元,由此指针总数将会达到 $O(N^{5/3})$.此类场景的典型例子,如:一本微微翻开的大小为 1×1×1 的厚书,其中每一页纸的大小都近似于 1×1;或者一丛茂密的窄叶草,每片草叶都近似为高度为 1 的小细条.在后面的实验中我们以随机生成的细长面片场景为例来模拟草地模型.

为降低指针数目,使其复杂度尽可能地接近 $O(N)$,我们进行如下的处理:考察每个面片的包围盒大小,并以其平均尺寸作为划分网格单元的大小.这将使得大多数面片只与有限几个网格单元相交,从而可以控制总的面片指针数目不是很高.具体算法如下所述:

算法 2.

(1) 按照算法 1 计算出场景包围盒 3 个轴向上的分辨率 M_i,$i\in\{x,y,z\}$;

(2) 根据各面片包围盒的平均尺寸,计算场景包围盒各轴向上的最大可允许的分辨率 $M_{\max\_i}$,$i\in\{x,y,z\}$:

(2.1) 计算面片包围盒的平均尺寸 LF_{avg_i},$i\in\{x,y,z\}$;

(2.2) 计算场景包围盒各轴向上的最大可允许的分辨率 $M_{\max\_i}=\alpha\times\frac{L_i}{LF_{avg\_i}}$.这里:$L_i$ 是场景包围盒各个轴向上的长度;α为一个控制常数,称为单元面片尺寸比;

(3) 若在步骤(1)中计算的 $M_i>M_{\max\_i}$,则置 $M_i=M_{\max\_i}$.

为了验证算法 2 的效率,我们进行了以下实验:随机生成 7 个场景,包含大小差别较大的细长三角面片,面片数从 1 000 逐步增长到 100 000.对这些场景分别根据公式(1)和算法 2 进行网格结构的建立,然后用光线跟踪进行绘制.相关实验结果在图 1 中给出,其中,图 1(a)是包含 1 000 个面片的场景(图 1(a)为包含 1 000 个细长随机三角面片场景的线框图,图 1(b)~图 1(d)分别给出两种创建算法对应的面片指针总数、创建时间和绘制时间随面片数变化的曲线).由实验结果可知:根据算法 2 的计算,所需的指针总数和创建时间与面片数基本呈线性关系,随着面片数的增长而缓慢增长.而根据公式(1)的计算,这两方面的曲线都呈超线性增长.在绘制效率方面,两者

的曲线几乎重合,效率相当.这表明,算法 2 相比于公式(1)能够有效降低网格创建时间和空间需求,但依然保持有相似的应用计算效率.

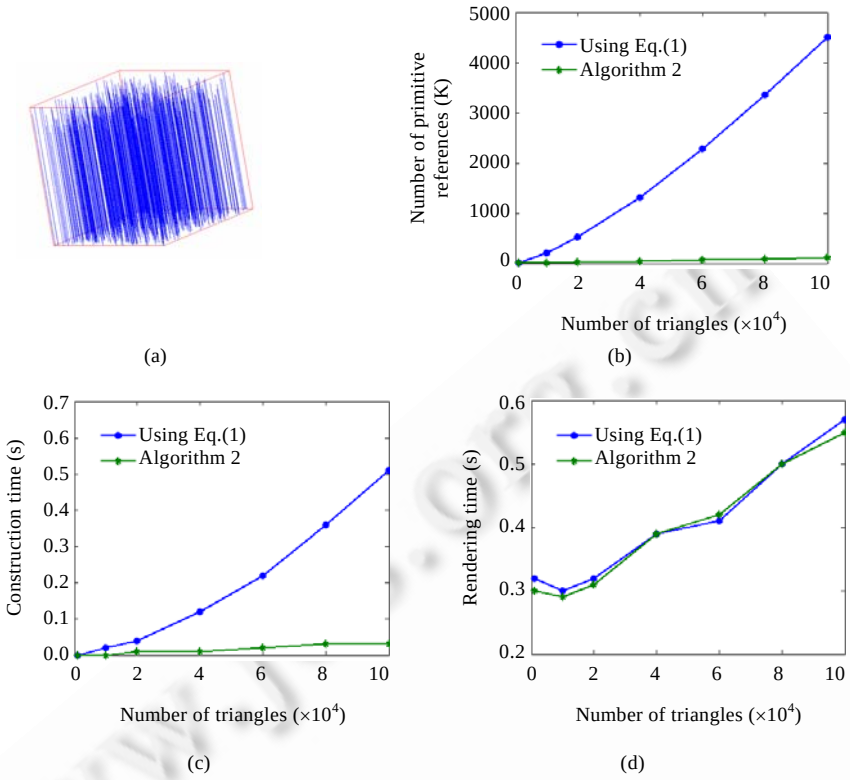

Fig.1 Performance of grids constructed by Eq.(1) and algorithm 2

图 1 根据公式(1)和算法 2 创建网格的性能对比

根据算法 2,通过使网格单元的尺寸不超过面片包围盒平均尺寸的α倍,一般可以使面片指针数目的复杂度为 $O(N)$.但对于小面片和大面片很多、而中间尺寸大小的面片很少的情况,这样处理依然有可能使指针数目超过 $O(N)$.仍以一个 1×1×1 的立方体场景为例进行说明.设场景包含 N 个面片,网格分辨率为 $M_x=M_y=M_z=N^{1/3}$,共 N 个网格单元.设面片包围盒在 3 个轴向上的平均长度皆为 $N^{-1/3}$,有 $N^{2/3}$ 个面片非常大,其各自包围盒的大小非常近似于 1×1×1;而另外 $N-N^{2/3}$ 个面片却非常小,其各自包围盒的大小接近于 0×0×0.这样,每个大面片占据的单元数都是 N,每个小面片占据的单元数是 1,最后的总面片指针数就是 $N\times N^{2/3}+(N-N^{2/3})=O(N^{5/3})$,超过了 $O(N)$.

对于这样面片大小分布极不均衡的情况,我们可先根据算法 2 计算出一个结果,再估算这样分辨率情况下所需的指针总数.即,根据每个面片的包围盒对照一个网格单元的尺寸以估算出该面片所需的指针数,再进行求和.若估算的指针数比较大,就调整网格单元的大小,再进行估算.一般尝试几次即可.由于一般不会遇到这种分布不均衡的情况,我们在实验中不再进行讨论.

2 层次网格的优化建立

在实际应用中,往往构建层次式的均匀网格结构,即相同大小的网格形成一个网格层次,而其中的每一个网格可包含更小尺寸的均匀网格.由此,可根据场景面片分布的特征自适应地调整划分结构,提高应用计算的效率.层次网格的建立,一般采用递归细分的方式[15].它首先为整个场景创建一层均匀网格,随后考察各个网格单元中包含的面片数,如所包含的面片数超过某设定的阈值 MAXP,则在该网格空间内再创建一层网格,该过程递归地进行,直到细分最深层次的网格单元所含面片数小于 MAXP 或者细分的层次深度达到一个设定的最大深

度 MAX*DEPTH* 为止.递归子网格的分辨率可以设定为固定分辨率,也可以根据各网格单元所含的面片数自适应地计算.一般地,固定分辨率的方式比较简单但会导致空间需求大幅度增加,而自适应计算的方式能够更好地适应面片分布不均匀和复杂的场景.分辨率的计算,通常是按照 $\sqrt[3]{\ }$ 准则[16,17].关于网格分辨率和层次网格的详细综述,参见文献[4,18,19].

但是,这样建立层次网格的方法会出现两个问题[14]:(1) 当场景中有很多大的面片与大多数网格相交时,递归无法终止,因为各个网格中的面片数总是大于 MAX*P*;(2) 指向物体的指针会很多,并可能随着层次数的增加而呈指数级的增长.例如:设第 1 层的网格 G_1 共生成了 λN 个网格单元以及 βN 个面片指针,且 G_1 内的所有非空单元都满足细分条件,那么第 2 层的所有网格所处理的面片总数就是 βN,生成的网格单元总数是 $\beta\lambda N$.若细分条件总是能得到同样的满足,则依此类推可知第 k 层网格生成的单元总数将是 $\beta^{k-1}\lambda N$,生成的面片指针数为 $\beta^k N$.显然,这样生成的层次网格需要很多的创建时间和空间,不利于应用计算效率的提升.

为了避免冗余的中间层次和过多的物体指针的出现,文献[14]提出:对具有近似大小且位置临近的面片进行聚簇,然后对各个簇包围盒进行网格生成,再将这样的网格当作独立的物体进行处理以形成整个场景的层次网格结构.这样虽然能够有效改善层次网格的质量,但聚簇计算不容易进行.对此,我们提出简便的方式,以有效减少场景的过度划分和递归深度:其一是限制每层网格的单元数目;其二是根据被细分单元所包含的子网格的单元个数来决定细分深度.

递归地创建层次网格的基本过程如下:

(1) 对于一个拥有 N 个面片的场景,首先建立一个单层均匀网格,记为 G_1;

(2) 遍历网格中所有单元,若发现某单元 i 可继续细分,则将该单元视为一个包含 N_i 个面片,且以该单元所占空间为包围盒的场景,对此子场景建立子网格 G_{2i};

(3) 遍历第 2 层所有子网格的所有单元,若发现某单元 j 仍可继续细分,则可继续建立子网格 G_{3j};

(4) 如此类推,直到所有单元都不满足细分条件为止.

当对一个层次的网格进行下一层次的子网格划分时,我们通过乘以一个系数来降低其分辨率,不妨设为 $1/\beta$.这样,在对第 2 层的一个包含 N_i 个面片的网格单元细分时,细分后的网格单元数将由 λN_i 降低为 $(1/\beta)\times\lambda N_i$.由此,第 2 层网格生成的细分网格单元总数将从 $\beta\lambda N$ 降低为 λN,生成的面片指针总数由原来的 $\beta^2 N$ 减少为 βN,与第 1 层网格生成的网格单元数以及面片指针数相同.以此类推,每一层次的网格单元总数都近似为 λN,面片指针总数近似为 βN.因此,当网格层次数增长时,网格单元数和面片指针的数量仅以线性增长,可显著降低空间需求和创建时间.实际工作中,β的取值不尽相同,可设为“单元内面片指针数/单元内子网格中所有面片指针数”(其中,单元是指将要被细分的一个单元,单元内子网格是指该单元被细分后形成的子网格),称为细分指针比,以自适应地反映面片分布状况与网格划分的关联性.实验结果表明,这样的设置能够取得不错的效果.

递归细分停止的条件,一般是设为单元内的面片个数低于某个阈值 MAX*P*.如前所述,这样可能导致无限细分.一种解决办法是设定最大层次深度 MAX*DEPTH*.当层次深度大于它时,停止划分.这种方法虽然能够适时地截止划分,但不能及时制止无效划分.对此,我们以单元细分的分辨率作为终止条件.换言之,当判断一个单元是否需要细分时,先计算如果其细分,将以何种分辨率创建其子网格,如果在该分辨率下的单元总个数小于一个设定的常量 γ,称为细分截止阈值,则它不进行细分.根据我们的细分分辨率计算方式,随着细分层次的增加,子网格的分辨率是逐步变少的,因此,我们的递归终止条件总是可以得到满足,并能对各个局部自适应地处理.

虽然我们的层次网格建立方法可以保证创建时间和空间都是 $O(N)$,但并不能保证在所有极端情况下,网格层次数都能保持在常数范围内.例如,设有一个场景,其包围盒的两个角点分别为[0,0,0]和[1,1,1].所有的面片都集中在两个对角[0,0,0]和[1,1,1]周边的极其小、体积无限接近 0 的区域内.这样,每一层次都只有两个单元包含面片,而且面片指针数不变,因此总能以相同的分辨率继续细分.尽管这种情况在实际应用中几乎不会出现,但为了保证我们算法的完备性,还是要追加一个细分终止条件,即网格总层次数不能超过某个阈值 MAX*DEPTH*.从大量实验的结果来看,一般场景的层次数不会超过 4,因此我们可以保守地将该阈值设为 4×2=8.

为了验证本文方法的有效性,我们进行了如下实验:设置 7 个不同精细程度的手骨(hand)模型,面片数从 10

000 变化到 600 000.图 2(a)展示了包含 10 000 个面片的 hand 模型场景.实验中的参数设置为λ=1,α=1,γ=8.图 2(b)~图 2(e)分别展示了单元总数、指针总数、创建时间和绘制时间随面片数变化的情况(图 2(a)为包含 10 000 个面片的 hand 模型.图 2(b)~图 2(e)分别为网格单元总数、面片指针总数、创建时间和绘制时间随面片数增长而变化的统计曲线).由图 2 可知,随着面片数的增加,我们的方法创建的网格单元总数、面片指针总数和创建时间均呈线性增长,且绝对数值比原始递归网格算法少很多.场景规模越大,差距越明显.而在绘制效率方面,我们的方法与原始算法的基本一致(大多数情况下略少).这表明,我们的层次网格创建方法不但可以大幅降低创建时间和空间消耗,而且还保持了很高的绘制效率.

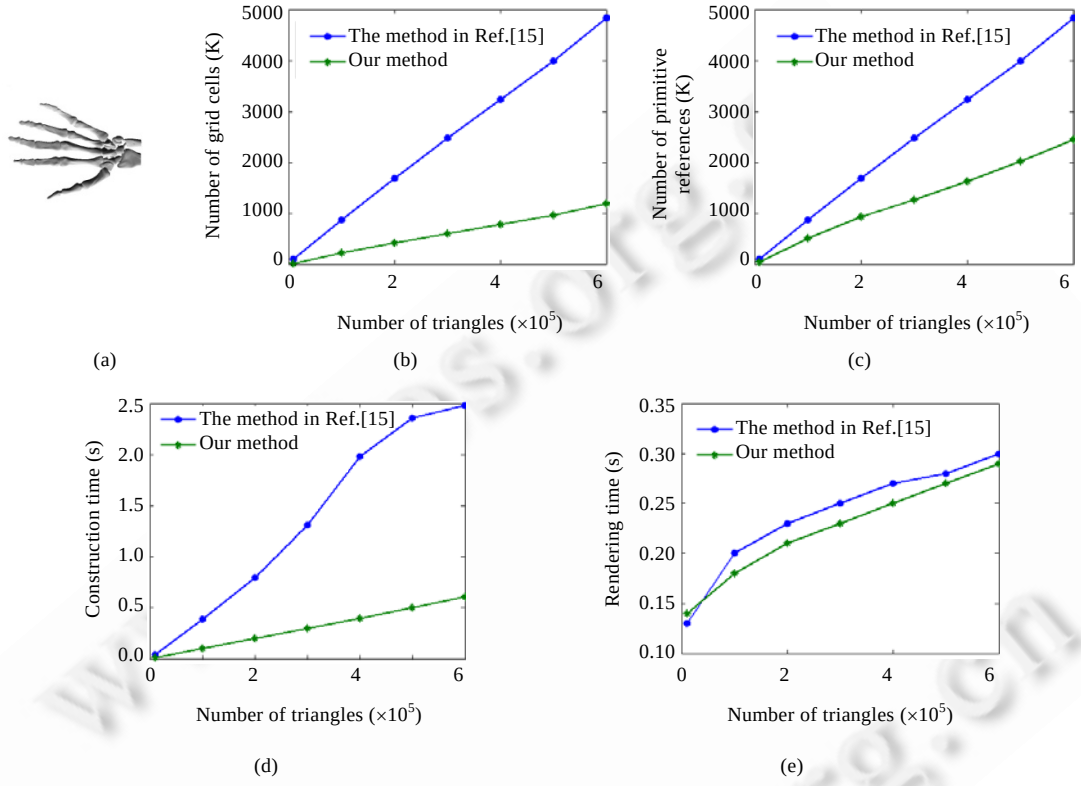

Fig.2 Performance of the algorithm[15] and our method for constructing hierarchical grids

图 2 通常的递归网格创建算法[15]和我们的优化创建算法的性能比较

3 实验与讨论

为检验本文工作的效率,我们将本文所提方法与一些常用的结构进行对比,并采用类似配置的机器、相同的场景和算法流程进行测试.为检验新方法的实用性,我们采用了当前国际上相关研究所常用的场景模型进行检测,其中, Bunny,Armadillo,Dragon,Buddha,Blade 都来自斯坦福大学网站[20],Robots,Museum,Kitchen 场景都来自标准动态光线跟踪测试场景 BART[21].所用计算机的配置为 AMD Athlon(tm) 3200+ CPU,1GB 内存(文献[21]中使用的测试机器为 AMD Athlon(tm) 64 X2 Dual Core Processor 3800+,只使用单核单线程,与我们的配置基本相同).绘制算法是单线程、非 SIMD 实现的逐根光线跟踪.测试场景、视点位置、图像分辨率与文献[22]保持一致,见表 1.其中:N_c,N_p 和 $N_{c\&p}$ 分别为网格单元(或结点)总数、面片指针总数以及单元指针之和,单位均为 K,r_c=N_c/N,r_p=N_p/N,$r_{c\&p}$=各结构的 $N_{c\&p}$/ORG 的 $N_{c\&p}$;T_c,T_r 和 $T_{c\&r}$ 分别为创建时间、绘制时间和总时间($T_{c\&r}$=T_c+T_r),单位均为秒(s),s_c=各结构的 T_c/ORG 的 T_c,s_r=各结构的 T_r/ORG 的 T_r,$s_{c\&r}$=各结构的 $T_{c\&r}$/ORG 的 $T_{c\&r}$.进行对比实验的结构包括:用本文方法创建的优化递归网格结构(ORG)、目前加速性能最好的 *kd* 树(*kd*-tree)[22]、一般

均匀网格(UG)[22]、递归网格(RG)[15]、两层递归网格(RG-2)[7].在此,*kd*-tree 和 UG 结构相关的数据直接引自文献[22],而我们实现 RG,RG-2 和 ORG 结构的程序,除了它们各自的划分方法不同之外,其他方面均一样.

Table 1 Performance comparisons of our optimized recursive grids and other structures

表 1 用本文方法创建的优化网格结构与其他加速结构的性能对比

		N_c[K] (r_c)	N_p[K] (r_p)	$N_{c\&p}$[K] ($r_{c\&p}$)	T_c[s] (s_c)	T_r[s] (s_r)	$T_{c\&r}$[s] ($s_{c\&r}$)
Static scenes (500×500)							
Bunny (69,451) 	*kd*-tree	663.8 (9.6)	408.4 (5.9)	1 072.2 (3.8)	0.76 (15.2)	0.35 (0.8)	1.11 (2.4)
	UG	663.8 (9.6)	245.1 (3.5)	908.9 (3.2)	0.26 (5.2)	0.38 (0.9)	0.64 (1.4)
	RG	200.6 (2.9)	438.8 (6.3)	639.3 (2.3)	0.1 (2.0)	0.33 (0.8)	0.43 (0.9)
	RG-2	739.8 (10.7)	640.6 (9.2)	1 380.4 (4.9)	0.1 (2.0)	0.3 (0.7)	0.4 (0.9)
	ORG	85.0 (1.2)	198.1 (2.9)	283.3 (1.0)	0.05 (1.0)	0.42 (1.0)	0.47 (1.0)
Armadillo (345,944) 	*kd*-tree	2 891.1 (8.4)	837.4 (2.4)	3 728.5 (1.9)	3.73 (9.3)	0.30 (0.9)	4.03 (5.4)
	UG	3 447.8 (10.0)	833.5 (2.4)	4 281.3 (2.2)	1.51 (3.8)	0.47 (1.3)	1.98 (2.6)
	RG	1 006.8 (2.9)	1 908.9 (5.5)	2 915.7 (1.5)	0.5 (1.3)	0.33 (0.9)	0.83 (1.1)
	RG-2	3 240.7 (9.4)	2 229.7 (6. 5)	5 470.4 (2.8)	0.5 (1.3)	0.26 (0.7)	0.76 (1.0)
	ORG	628.7 (1.8)	1 334.4 (3.9)	1 963.0 (1.0)	0.4 (1.0)	0.35 (1.0)	0.75 (1.0)
Dragon (844,037) 	*kd*-tree	4 426.7 (5.2)	3 827.2 (4.5)	8 253.9 (1.6)	9.65 (9.9)	0.63 (0.9)	10.28 (6.1)
	UG	4 260.3 (5.1)	3 834.4 (4.5)	8 094.7 (1.6)	3.50 (3.6)	0.94 (1.3)	4.44 (2.6)
	RG	2 726.0 (3.2)	6 071.8 (7.2)	8 797.8 (1.7)	1.35 (1.4)	0.65 (0.9)	2 (1.2)
	RG-2	8 237.4 (9.8)	6 185.2 (7.3)	14 422.6 (2.8)	1.23 (1.3)	0.52 (0.7)	1.75 (1.0)
	ORG	1 569.9 (1.9)	3 639.5 (4.3)	5 209.4 (1.0)	0.98 (1.0)	0.7 (1.0)	1.68 (1.0)
Happy buddha (1,051,739) 	*kd*-tree	2 675.6 (2.5)	8 114.4 (7.7)	10 790.0 (1.7)	14.73 (12.9)	0.72 (2.1)	15.45 (10.5)
	UG	5 306.6 (5.1)	10 613.1 (10.1)	15 919.7 (2.5)	5.79 (5.1)	0.79 (2.3)	6.58 (4.5)
	RG	2 986.0 (2.8)	6 341.7 (6.0)	9 327.8 (1.5)	1.42 (1.3)	0.31 (0.9)	1.73 (1.2)
	RG-2	10 309.8 (9.8)	8 108.6 (7.7)	18 418.4 (2.9)	1.59 (1.4)	0.32 (0.9)	1.9 (1.3)
	ORG	1 874.8 (1.8)	4 397.3 (4.2)	6 272.1 (1.0)	1.14 (1.0)	0.34 (1.0)	1.47 (1.0)
Blade (1,765,388) 	*kd*-tree	14 984.3 (8.5)	3 388.4 (1.9)	18 372.7 (2.0)	16.45 (9.1)	0.34 (0.6)	16.80 (7.1)
	UG	8 862.8 (5.0)	5 369.6 (3.0)	14 232.4 (1.6)	6.85 (3.8)	0.64 (1.1)	7.50 (3.2)
	RG	5 062.1 (2.8)	10 197.5 (5. 8)	15 259.6 (1.7)	3.56 (2.0)	0.51 (0.9)	4.07 (1.7)
	RG-2	16 778.8 (9.5)	12 527.3 (7.1)	29 306.1 (3.2)	2.52 (1.4)	0.39 (0.7)	2.91 (1.2)
	ORG	2 770.8 (1.6)	6 320.1 (3.6)	9 091.0 (1.0)	1.81 (1.0)	0.57 (1.0)	2.38 (1.0)
Dynamic scenes (400×300)							
Robots (71,580)	*kd*-tree	506.3 (7.1)	356.3 (5.0)	862.6 (1.9)	0.75 (10.7)	0.24 (0.6)	0.99 (2.2)
	UG	358.9 (5.0)	229.4 (3.2)	588.3 (1.3)	0.73 (10.4)	5.30 (13.6)	6.03 (13.1)
	RG	612.0 (8.6)	33 72.8 (47.1)	39 84.8 (8.6)	0.33 (4.7)	0.30 (0.8)	0.62 (1.4)
	RG-2	688.7 (9.6)	415.3 (5.8)	1104.0 (2.4)	0.10 (1.4)	0.37 (1.0)	0.47 (1.0)
	ORG	125.4 (1.8)	336.3 (4.7)	461.7 (1.0)	0.07 (1.0)	0.39 (1.0)	0.46 (1.0)
Museum (75,687)	*kd*-tree	313.6 (4.1)	1 782.3 (23.6)	2 095.9 (2.2)	2.57 (21.4)	0.16 (0.6)	2.74 (6.9)
	UG	378.0 (5.0)	271.6 (3.6)	649.6 (0.7)	0.79 (6.6)	0.20 (0.7)	0.99 (2.5)
	RG	195.0 (2.6)	4 606.4 (60.9)	4 801.4 (4.9)	0.35 (2.9)	0.27 (1.0)	0.62 (1.6)
	RG-2	778.5 (10.3)	6 417.4 (84.8)	7 195.9 (7.4)	0.55 (4.6)	0.28 (1.0)	0.83 (2.1)
	ORG	125.8 (1.7)	850.2 (11.2)	976.0 (1.0)	0.12 (1.0)	0.28 (1.0)	0.40 (1.0)
Kitchen (110,540)	*kd*-tree	335.6 (3.0)	353.7 (3.2)	689.3 (1.0)	1.03 (10.3)	0.22 (0.7)	1.25 (3.0)
	UG	555.1 (5.0)	233.7 (2.1)	788.8 (1.2)	0.79 (7.9)	1.36 (4.3)	2.15 (5.1)
	RG	1 707.2 (15.4)	37 360.3 (338.0)	39 067.5 (58.3)	2.95 (29.5)	0.40 (1.3)	3.35 (8.0)
	RG-2	1 012.2 (9.2)	639.1 (5.8)	1651.3 (2.5)	0.15 (1.5)	0.33 (1.0)	0.47 (1.1)
	ORG	209.6 (1.9)	461.0 (4.2)	670.6 (1.0)	0.1 (1.0)	0.32 (1.0)	0.42 (1.0)

在 ORG 的实现中需要确定 3 个参数,即网格密度系数λ、单元面片尺寸比α和细分截止阈值γ.对此,我们进行了一些实验,并从空间消耗、网格创建时间、绘制时间和总时间等方面考察不同参数组合所生成网格的性能,最后以"λ=1,α=2,γ=16"作为我们实验中的参数设置.在下面的论述中,如不特别声明,参数(λ,α,γ)都是以上设置.

在 RG 的实现中,若单元所含面片数大于某个阈值β,则继续细分,每次划分λN 个网格单元,这里的 N 为对应网格内的面片数.其中,λ和β的取值与 ORG 结构的λ和γ取值一样,分别为 1,16.由于很多场景如果不限制层次深度都会导致过度细分乃至使内存超限,因此对于 Dragon,Happy,Blade 和 Museum 场景,我们强制限制其最大层次数为 2,Robot 场景和 Kitchen 场景强制限制其最大层次数为 4.

Ize 等人[7]提出了根据场景不同的分布情况计算均匀网格以及递归网格最优分辨率的方法.该方法并不能自适应细分,只能预先指定层次数.因此,我们测试了其两层网格结构(RG-2)的实现.在我们的实现过程中,双层

网格的首层网格分辨率 $M_1=(8N)^{0.6}$,第 2 层网格分辨率 $M_2=8N_2$.这里,N_2 为子网格中的面片数.

表 1 列出了对比实验的数据.下面,我们将从空间、时间两个方面分析 ORG 相对于其他结构的性能.

3.1 空 间

单元总数(N_c)和面片指针总数(N_p)是衡量加速结构空间需求的主要参数,与实现无关.为了直观地表示它们与场景规模的关系,表 1 中加入了它们与场景面片总数的比值,即 $r_c=N_c/N$ 和 $r_p=N_p/N$.数据显示,对各种场景,ORG 的 r_c 均介于 1 和 2 之间;而 r_p 则大多介于 3 和 5 之间.取值比较稳定,并没有随着场景规模的增加而增加,这表明,ORG 的空间复杂度是 $O(N)$的.在总的空间需求方面,可以合理地用单元和指针总数($N_{c\&p}$)作为衡量参数,并以其他结构的 $N_{c\&p}$ 相比于 ORG 的 $N_{c\&p}$ 的比值 $r_{c\&p}$ 进行考察.从表 1 中的数据可知,除了 Museum 模型的 UG 结构外,其他结构都比 ORG 结构所需的空间要多.尤其是在 Kitchen 场景中,ORG 大幅降低了一般递归网格(RG)的空间开销(RG 的 $r_{c\&p}$ 值达到了 58.3).

3.2 时 间

表 1 中列出了各结构的创建时间(T_c)、绘制时间(T_r)和总时间 $T_{c\&r}=T_c+T_r$.为了便于比较,还加入了其他结构相对于 ORG 的时间比值,即创建时间比 s_c,绘制时间比 s_r 以及总时间比 $s_{c\&r}$.

对于创建时间,ORG 在各个场景中都是最少的.其中,*kd* 树的创建时间平均是 ORG 的 12.3 倍,对于 Museum 场景更是达到了 21.4 倍.根据文献[22]中的介绍,其中的 *kd* 树实现经过了高度优化,已经比文献[3]的实现快两倍.也就是说,如果 ORG 对比文献[3]中的 *kd* 树实现,上述时间比还应当翻番.虽然最近有些工作通过并行计算或利用硬件大幅降低了 *kd* 树的创建时间[23,24],但并没有改善其复杂度.我们的工作同样可以并行化处理,但从算法的时间复杂度上来看,ORG 在创建时间上大大少于 *kd* 树的创建时间.

对于绘制时间,ORG 并不总是最优的,但与其他结构中最快者相差不大.特别是与目前公认的最快加速结构 *kd* 树相比,其比值 s_r 在 0.6~2.1 之间.这说明,ORG 能够保持很好的加速性能,可与其他结构相匹敌.

对于动态场景的绘制,总时间 $T_{c\&r}$ 是关键的性能衡量指标.由于 ORG 结构大幅降低了创建时间,同时又保持了加速性能,因此其总时间也明显减少.由表 1 可见,各结构 $r_{c\&r}$ 的值大多介于 1 和 13 之间,平均为 3.3.

综上,ORG 在大幅降低存储需求和创建时间的同时,保证了很好的光线行进求交加速性能.这得益于新方法能够根据场景情况自适应地选择恰当的网格分辨率和递归层次.与其他用于对比的结构相比,ORG 结构的光线跟踪效率虽然并不总是最优(平均提高 0.5 倍),但其创建效率却能大幅提升(平均提高 5.4 倍,最多 28.5 倍),由此使得其创建与绘制的总效率最优(平均提高 2.3 倍).而且,新方法的空间效率是最好的.这些都表明,新方法很适合处理动态场景和大规模复杂场景.

4 结 语

对于均匀网格结构的优化划分,本文以其常用的分辨率优化计算公式为基础,深入探讨场景包围盒的尺寸和面片大小的差异对其的影响,提出了新的处理算法,以更好地优化网格划分.同时,进一步探讨层次化网格生成时各子网格分辨率的优化计算及层次深度的控制,使其创建时间复杂度和空间复杂度都是 $O(N)$的.实验结果表明,相比于常用的加速结构,新方法在创建时间和空间需求方面有大幅度的性能提升.而在应用计算方面,比如光线跟踪绘制,具有与其他结构相当的效率,即使与当前最快的主流加速结构 *kd* 树相比也不逊色.因此,新方法很适合处理动态场景和大规模场景,更有利于实践应用的效率提升.

References:

[1] Chang AY. Theoretical and experimental aspects of ray shooting [Ph.D. Thesis]. New York: Polytechnic University, 2004.

[2] Reshetov A, Soupikov A, Hurley J. Multi-Level ray tracing algorithm. ACM Trans. on Graphics, 2005,24(3):1176−1185. [doi: 10.1145/1186822.1073329]

[3] Wald I, Havran V. On building fast *kd*-trees for ray tracing, and on doing that in *O*(*N*log*N*). In: Proc. of the IEEE Symp. on Interactive Ray Tracing 2006. 2006. 61−69. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4061547&tag=1 [doi: 10.1109/RT.2006.280216]

[4] Havran V, Prikryl J, Purgathofer W. Statistical comparison of ray-shooting efficiency schemes. Technical Report, TR-186-2-00-14, Czech Republic: Czech Technical University, 2000.

[5] Cleary JG, Wyvill G. Analysis of an algorithm for fast ray tracing using uniform space subdivision. The Visual Computer, 1988, 4(2):65−83. [doi: 10.1007/BF01905559]

[6] Aronov B, Brönnimann H, Chang AY, Chiang YJ. Cost prediction for ray shooting in octrees. Computational Geometry: Theory and Applications, 2006,34(3):159−181. [doi: 10.1016/j.comgeo.2005.09.002]

[7] Ize T, Shirley P, Parker SG. Grid creation strategies for efficient ray tracing. In: Proc. of the IEEE/EG Symp. on Interactive Ray Tracing 2007. Ulm: IEEE, 2007. 27−32. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4342587 [doi: 10.1109/RT.2007.4342587]

[8] Li J, Wang WC, Wu EH. Optimizing grid resolutions for ray tracing. Journal of Computer-Aided Design & Computer Graphics, 2008,20(8):968−977 (in Chinese with English abstract).

[9] Klimaszewski K, Sederberg TW. Faster ray tracing using adaptive grids. IEEE Computer Graphics and Applications, 1997,17(1):42−51. [doi: 10.1109/38.576857]

[10] Shirley P. Objects per grid cell. Ray Tracing News, 2002,15(1). http://tog.acm.org/resources/RTNews/html/rtnv15n1.html

[11] Wald I, Ize T, Kensler A, Knoll A, Parker SG. Ray tracing animated scenes using coherent grid traversal. ACM Trans. on Graphics, 2006,25(3):485−493. [doi: 10.1145/1179352.1141913]

[12] Lagae A, Dutré P. Compact, fast and robust grids for ray tracing. Computer Graphics Forum, 2008,27(4):1235−1244. [doi: 10.1111/j.1467-8659.2008.01262.x]

[13] Kalojanov J, Slusallek P. A parallel algorithm for construction of uniform grids. In: Proc. of the High Performance Graphics 2009. New Orleans, 2009. 23−28. http://dl.acm.org/citation.cfm?id=1572773 [doi: 10.1145/1572769.1572773]

[14] Cazals F, Drettakis G, Puech C. Filtering, clustering and hierarchy construction: A new solution for ray-tracing complex scenes. Computer Graphics Forum, 1995,14(3):371−382. [doi: 10.1111/j.1467-8659.1995.cgf143_0371.x]

[15] Jevans D, Wyvill B. Adaptive voxel subdivision for ray tracing. In: Proc. of the Graphic Interface 1989. 1989. 164−172.

[16] Jansen E, Leeuw W. Recursive ray traversal. Ray Tracing News, 1992,5(1). http://tog.acm.org/resources/RTNews/html/rtnv5n1.html

[17] Jansen, E. Comparison of ray traversal methods. Ray Tracing News, 1994,7(2). http://tog.acm.org/resources/RTNews/html/rtnv7n2.html

[18] Havran V. Heuristic ray shooting algorithms [Ph.D. Thesis]. Prague: Czech Technical University, 2000.

[19] Cosenza B. A survey on exploiting grids for ray tracing. In: Proc. of the 6th Eurographics Italian Chapter. 2008. 89−96. http://www.bibsonomy.org/bibtex/2876c162ea5d7469db06d3ecd4a030188/dblp

[20] The Stanford 3D Scanning Repository. http://graphics.stanford.edu/data/3Dscanrep/

[21] BART: A Benchmark for Animated Ray Tracing. http://www.ce.chalmers.se/research/group/graphics/BART/

[22] Havran V, Herzog R, Seidel HP. On the fast construction of spatial hierarchies for ray tracing. In: Proc. of the IEEE/EG Synp. on Interactive Ray Tracing 2006. 2006. 71−80. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4061548&tag=1 [doi: 10.1109/RT.2006.280217]

[23] Shevtsov M, Soupikov A, Kapustin A. Highly parallel fast *kd*-tree construction for interactive ray tracing of dynamic scenes. Computer Graphics Forum, 2007,26(3):395−404. [doi: 10.1111/j.1467-8659.2007.01062.x]

[24] Kun Z, Hou QM, Wang RN, Guo BN. Real-Time *kd*-tree construction on graphics hardware. ACM Trans. on Graphics, 2008,27(5):126:1−126:11. [doi: 10.1145/1409060.1409079]

附中文参考文献:

[8] 李静,王文成,吴恩华.加快光线跟踪计算的网格优化划分.计算机辅助设计与图形学学报,2008,20(8):968−977.

李静(1972－),女,北京人,博士,助理研究员,主要研究领域为真实感绘制,计算机图形学,计算几何.

王文成(1967－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为科学计算可视化,虚拟现实,计算机图形学.