**基于光线跟踪渲染框架的粒子数据**

**表面动态重构**

赵海洋

（清华大学 航天航空学院）

**摘要**

粒子表示的具有明确表面的液体或固体需要进行表面重构后才可以渲染。现有的表面重构算法中基于各向异性核的标量场等值面提取法可以取得较好的效果，但是其缺点是计算比较耗时且消耗较大内存。本文基于该算法，提出了一种使用扩展型八叉树结构的算法改进，即结合光线跟踪算法，通过只对可见表面附近计算并存储粒子密度场值，实现了降低计算量及减小内存消耗的效果，且同时完成了对构造出的三角面片的空间划分，提高光线跟踪过程求交的效率。

**关键词：**表面重构，粒子数据，光线跟踪，八叉树

**1 概述**

近年来，无网格法发展迅速。由于其在大变形问题上具有的优势，各类无网格法被广泛应用于自由表面流体、流固耦合、冲击爆炸等问题的仿真中，使用比较广泛的无网格法主要有光滑质点流体动力学法（SPH）、质点网格法（PIC）及在其基础上发展而来的FLIP法及物质点法。这些方法的共同点是都以大量离散的粒子来表示模型实体，而粒子表示的实体无法直接用于渲染，需要对其进行表面提取工作。考虑到提取表面的质量和效率，这并不是一项很容易的工作。目前普遍采用的表面提取方法是隐式表面法，包括Hoppe等人[1]提出的符号距离函数法和MÜLLER等人[2]使用的用标量场的等值面提取算法。这类方法通常使用Marching Cubes算法提取等值面并生成三角面片继而进行渲染。该类方法易于实现，但是其效果受限于粒子形状和位置，需要进行很多如表面平滑等提高效果的工作。

本文的工作主要针对Yu的工作[3] 进行改进。该算法使用各向异性核来获得更加平滑的表面，但代价是需要进行大量计算。考虑到大部分模型表面层只占整体的很小一部分体积，于是可以无需对模型内部粒子进行计算；同时采用光线跟踪渲染框架时，光线只会和可见表面进行求交运算，不可见表面无需构造，如此可以提升效率并降低内存消耗。本文提出了一种采用动态增长的八叉树结构来构造和存储表面的算法来实现该功能。

**2 现有表面重构算法**

**2.1 相关工作**

从粒子数据中提取表面是一件并不容易的工作。早先隐式表面重构主要应用在基于欧拉网格的仿真中，如Sethian等[4]提出的level-set法，该算法通过速度场构造符号距离函数实现。

针对粒子数据的表面重构算法最早有Blinn等[5]提出的Classic Blobby Spheres算法，该算法在每个粒子的中心位置构造一个径向基函数，并将所有粒子的径向基函数累加得到标量场进而进行等值面提取。该方法的缺点是生成的表面不够平整。

Zhu和Bridson[6]针对Blinn的算法提出了一种改进算法，该算法使用一点周围若干粒子的加权平均位置作为径向基函数的核心位置，而不再使用每个粒子的中心位置。该方法使用新的径向基函数进行累加得到标量场，并使用空间规则网格在网格结点上采样，在进行光滑操作后，在网格单元内提取等值面。

Yu和Turk[3]在这类方法的基础上进一步改进，使用椭球形的各向异性核代替球形的各向同性核。该算法通过搜索邻近粒子，并利用其位置信息构造一个协方差矩阵，并通过求解该矩阵特征值的方法来构造表征各向异性核的矩阵。随后通过邻近粒子位置的加权平均来对每个核的位置进行平滑操作。最后使用和Zhu等相同的算法，即使用空间规则网格采样标量场，再在网格单元内使用Marching Cubes算法[7]生成三角面片。该算法的主要效率瓶颈为各向异性核的构造和标量场的计算，其算法要为模型中的每个粒子构造协方差矩阵并计算其特征对，且为每个空间网格结点计算标量场值，这消耗了大量时间，同时存储所有网格结点也会耗费较大内存。

**2.2 标量场的构造及等值面提取**

对各向同性核，标量场可以表示为：

 (1)

其中j为所有影响到位置 的粒子的编号，W即为每个粒子处的各向同性核，形式如下：

 (2)

其中是一个缩放值，*d*为仿真的维数，*h*为核心的影响半径，一般由仿真计算阶段提供，为某点到核心的位移矢量 ，*P*为一个对称的衰减函数，用来描述场变量值在核内的分布。

对各向异性核，文献[7]中使用了一个的正实数矩阵来代替，这使得核心从球体变成了椭球体（椭圆），的各阶特征值的倒数代表该椭球体的各主轴长度。此时核心表示为

 (3)

于是场变量表示为

 (4)

其中是影响域内第*j*个粒子为了平滑表面而被调整后的位置。其中的构造需要先构造一个协方差矩阵

 (5)

并通过计算其特征对来构造

由(4)表示的标量场需要离散到空间网格结点上，然后再对所有网格单元使用Marching Cubes算法生成三角面片。出于效率考虑，在对一个各向异性核采样前先求其AABB包围盒，并计算哪些网格结点在包围盒内，然后再去在这些结点位置采样标量场值，如图1所示。

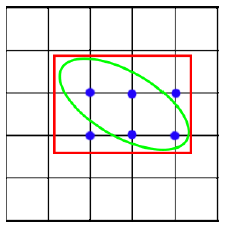


图1 各向异性核的AABB包围盒采样示意

1. **基于动态八叉树网格结点的采样和存储**
   1. **数据结构**

传统的八叉树结构是一种区域划分结构，它将容纳物体的最大区域按照每层八个卦限划分下去，直到某一层的一个区域内不再包含物体或已经划分到底层体素。

区域划分出的八个卦限按照x方向相差1，y方向相差2，z方向相差4的方式编号，这样对于空间任一点，可以由坐标直接计算出其所在卦限的子节点索引，无需将点坐标和区域边界进行比较。该结构插入和查找的时间复杂度均为，可以大大提高效率。

以上这种传统八叉树是在确定了物体最大范围情况下进行逐级细分，而有些情况下确定物体最大范围并不是一件容易的事，或者这个范围是在动态变化的，这时候这种数据结构实用起来就显得较为不方便。于是本文对八叉树的生成方式进行了改进，使它能够从最底层的一个体素开始“增长”。图2.3展示了增长式生成的过程

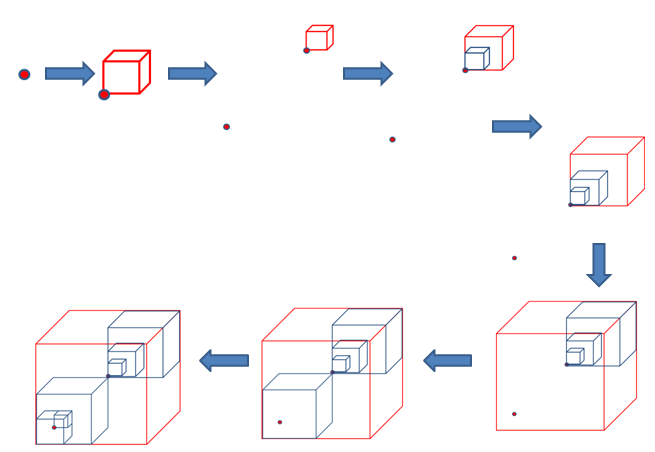


图2 动态扩展型八叉树插入过程

在插入每一个结点时，都按最小体素长度对结点位置进行对齐，然后再进行插入操作；

1. 插入第一个结点时，以它为零点构造第一个最小的体素，并使之成为根结点，记高度为0；

2. 插入第二个结点时，如果它在根结点区域外，则构造大一层的立方体区域，使之成为新的根节点，记高度为1，原有根节点成为它的子节点；

3. 如果新插入结点和跟结点的距离大于根节点区域的边长，那么继续构造大一层的区域，更新根结点，高度增加1，重复2直到该距离小于根节点区域边长

4. 判断新插入结点所在的卦限，由此调整原有根节点在新根节点中的卦限

5. 按照传统插入算法完成新结点的插入过程

* 1. **算法原理和框架**

本文的重构算法和已有工作最大的区别在于，已有工作是将所有表面重构完成后再统一渲染，然而对物体的内部区域和不可见的表面所做的运算是一种巨大的浪费；而由于本文采用光线跟踪的渲染框架，追踪光线并不会到达物体内部（不透明物体）和不可见表面，所以可以只对光线跟踪点进行表面重构，即实现边渲染边重构的动态结构，省去了不必要的计算量，同时也节省了内存消耗。而且，当变换渲染视角后，该算法也无需重新构造全部表面，只需要查询之前构造好的表面，并继续重构变换视角后新看到的一部分表面即可。

这里使用上一节提到的动态增长型八叉树存储标量场值及构造出的三角面片，在八叉树叶子节点对应的网格结点处对标量场进行采样，并在该网格单元内使用Marching Cubes算法构造三角面片并存储在该叶子节点中。使用该结构无需将全空间划分为规则网格单元，而是只在光线经过的区域动态生成层次结构网格，可以极大地减少网格数量，并且使用它存储的标量场值和构造的三角面片可以以的时间复杂度被快速查找。这里需要指出的是，虽然使用体素的哈希算法[9]可以实现更高效的标量场值插入和查找，但是由于本文使用光线跟踪的渲染框架，即当使用哈希算法快速构造出三角面片以后，还是需要诸如八叉树这样的空间划分数据结构来组织这些三角面片来实现求交运算的加速，所以还不如从一开始就使用八叉树结构，这样重构好的表面就已经自动被组织起来，无需花费额外的计算成本，在追踪其它光线时便可实现快速求交。

算法流程如图2.4所示。

Cube与表面不相交

从光线与包围盒交点开始以较大cube进行MarchingCubes 算法

否

否

是

是

Cube与

表面相交

继续步进

在八叉树中查找是否有该体素

对体素内的三角形面片进行传统求交运算

将该体素插入八叉树（成为根节点）构造三角面片并存入该节点

Cube边长减半，继续步进，直到cube大小等于最小体素

使用增长型八叉树存储cube顶点

场变量值

继续后续追踪

判断光线是否与物体包围盒相交

发射追踪光线

图3 表面重构算法流程

* 1. **算法实现**

由于重构算法中最耗时的部分为标量场值的计算（包括计算每个粒子的G矩阵，累加式(4)中的所有等），所以改进的一切出发点就是尽可能地减少计算的次数。一般来说，在载入模型时，都要对模型求一次AABB包围盒，这样光线如果不与包围盒相交，就无需进行表面重构操作；进一步的，我们还可以用k-d树结构实现的最近邻搜索算法进行一次最近的粒子搜索，若该距离大于视点到光线与包围盒交点距离，则还可以将光线前端进一步前移，进一步减少计算次数，如图2.5所示。

图4 表面重构自适应网格原理

当从视点（Eye）发出追踪光线后（红色光线），首先搜索最近粒子，并找到了区域，得到了，于是可以直接从A点开始执行Marching Cubes算法，并首先以一个实现设定好的较大边长的cube步进，如果cube不与物体表面相交，则将光线穿出该cube，并向前移动一个小量，由此计算坐标可以找到下一个cube，如图中的1和2；当cube与物体表面相交时，如3，则二分cube的边长按上面的步骤继续步进，直到cube已经减小到最小体素，这时当找到第一个与物体相交的最小体素时，按照Marching Cubes算法构造三角面片，并将其存储在体素（八叉树根节点）中，然后计算光线与三角形面片的交点；如果不存在交点，则以最小cube继续步进，直到找到交点为止。图中色圆点是需要计算的位置，计算一点的时，为每个被第一次搜索到的粒子计算其G矩阵。这种策略导致当模型物体体积较大时，完全无需计算物体内部大量网格结点的，且当另外一条光线再次经过已经构造好的表面时，只需要去计算光线和三角形面片交点即可，如图中Eye2的光线。以上情况针对于反射表面，如果是投射，当光线找到入射表面的交点射入物体内部时，可以再次增大步进cube的边长，这样物体内部也只需计算少量

1. **结果与讨论**

这里为了检验表面重构效果和扩展型八叉树算法对加速和降低内存使用量所起到的作用，使用了如图5中的模型

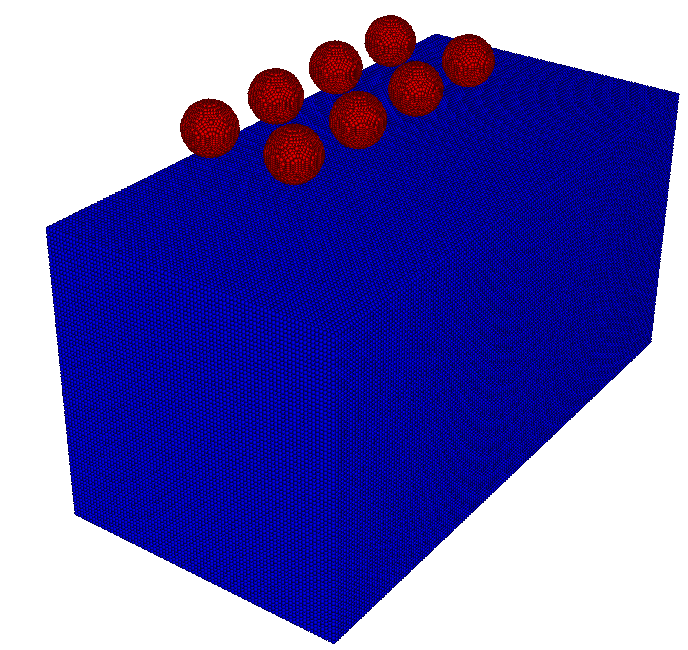


图5 模型粒子数据

该模型总粒子数为109.9万个，全模型空间若以最小体素尺寸离散为规则网格，总数为1117.8万个，渲染结果如图6和表1所示。

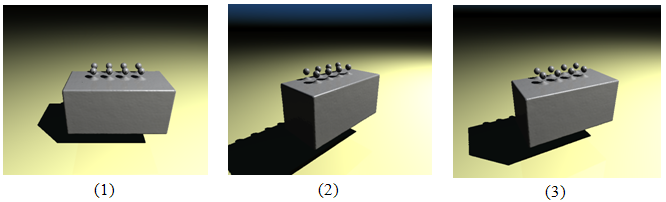


图6 模型的渲染结果图

表1 模型的渲染数据

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 帧数 | 预处理 | 渲染 | 使用粒子 | 生成八叉树 | 新生成三角 |
| 耗时(秒) | 耗时(秒) | 数(万个) | 网格(万个) | 面片(万个) |
| 1 | 18 | 488.9 | 31.4 | 92.9 | 24.1 |
| 2 | 0 | 126.5 | 34.4 | 109.5 | 7.37 |
| 3 | 0 | 44.15 | 34.4 | 111.9 | 2.53 |
| 1(第二次) | 0 | 6.61 | 34.4 | 111.9 | 0 |

由表1可以看出，使用扩展型八叉树结构可以极大地减小使用的网格数量，基本减小一个数量级，这可以大大降低内存消耗。由使用到的粒子数看，第一帧图像渲染过程中只用到了粒子总数的三分之一不到，也就是说构造G矩阵等最为耗时的操作仅在三分之一的粒子上进行，于是重构速度上也实现了提高。

在场景不变的情况下，我们变换不同的视角渲染三帧图像。对比第一帧和第二帧可以发现，使用到的粒子数和生成的八叉树网格只有小幅度增加，而新生成的三角面片数也大幅降低，这是因为被光源照亮的两个表面已经在第一帧的渲染过程中重构出了表面，在第二帧里只需要进行查询，于是渲染用的时间也大幅降低。而第二帧中新看到了一个背光面的表面，新生成的重构三角面片也大多来自于这个表面。于是我们继续转换一个视角，此视角可以看到已经在前两帧被重构过的三个表面，于是该帧使用的粒子数没有增加，而有少量新的三角面片被重构出来，这部分三角面片是因为三角面片比较小，前两帧的跟踪光线密度不够高，可能存在没有和光线相交的表面，而在更换视角后会相交。最后，我们对已经渲染过的第一帧再次渲染，此时由于完全没有新表面出现，渲染速度非常快，约比第一帧快两个数量级。

1. **总结和展望**

本文根据光线跟踪渲染框架的特点，从节省内存和提高渲染效率两个角度出发，提出了一种基于扩展型八叉树结构的优化算法。该算法将前人工作中基于空间规则网格的重构算法改为使用空间扩展型八叉树结构，只在可见表面上生成网格用于构造表面片，于是显著降低了内存消耗，并在非透明体表面的渲染中较大地提高了渲染效率。然而，由于本文的改进算法使用了较为粗大的步进cube作为初始cube，这导致了尺寸过小的细节，如固体的飞溅小碎片，液体的小液滴等，在有些情况下是无法被捕捉到的，此问题有待今后的工作来解决。另外，本文的光线跟踪框架是在CPU中实现的，这使得其效率仍然较低，今后的工作可以考虑对其进行GPU并行化。

**参考文献**

1. Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, Werner Stuetzle. Surface reconstruction from unorganized points. Computer Graphics (Proceeding of ACM SIGGRAPH 92),26(2):71-78,1992
2. Müller M, Charypar D, Gross M. Particle-Based Fluid Simulation For Interactive Applications[J]. Sca Proceedings of the Acm Siggraph/eurographics Symposium on Computer Animation, 2003:154-159.
3. Yu J, Turk G. Reconstructing Surfaces of Particle-Based Fluids Using Anisotropic Kernels[J]. Proc.of the Acm Siggraph/eurographics Symp.on Comp.anim.eurographics Association, 2010, 32(1):217-225.
4. Sethian J A. A fast marching level set method for monotonically advancing fronts[J]. Proceedings of the National Academy of Sciences, 1996, 93(4): 1591-1595.
5. Blinn J F. A generalization of algebraic surface drawing[J]. ACM Transactions on Graphics (TOG), 1982, 1(3): 235-256.
6. Zhu Y, Bridson R. Animating sand as a fluid[J]. ACM Transactions on Graphics (TOG), 2005, 24(3): 965-972.
7. In. Marching cubes: A high resolution 3D surface construction algorithm. SIGGRAPH '87[J]. Proceedings of Annual Conference on Computer Graphics & Interactive Techniques, 1987:163-169.