基于八叉树的体渲染优化

Dezeming Family

2023 年 4 月 11 日

DezemingFamily 系列文章和电子书**全部都有免费公开的电子版**，可以很方便地进行修改和重新发布。如果您获得了 DezemingFamily 的系列电子书，可以从我们的网站 [https://dezeming.top/] 找到最新的版本。对文章的内容建议和出现的错误也欢迎在网站留言。

目录

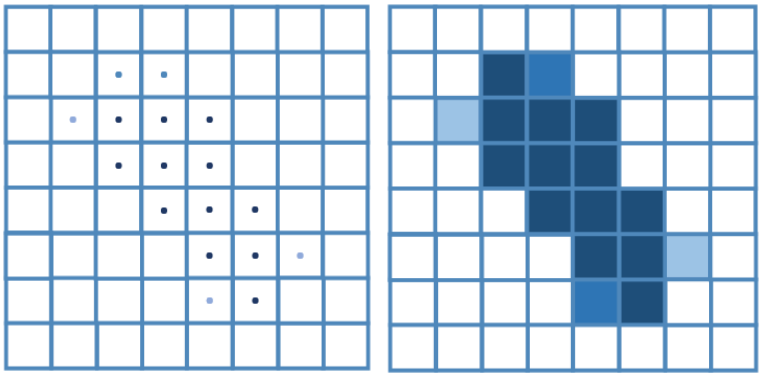
一　预备知识

预备知识：体渲染、3D 纹理、体素、树数据结构。

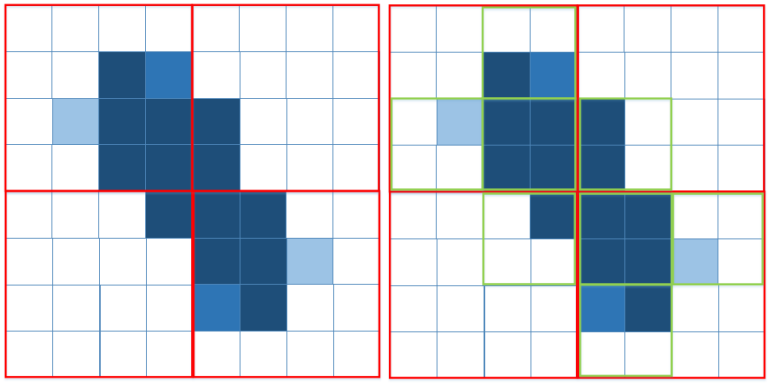
二　最简单的八叉树结构

最简单的八叉树方法其实就是将体素简化为小立方体，然后我们需要 ray-Box 求交算法来与包围盒求交。

具体步骤如下，为了方便描述，我们用二维平面来表示（所以是用四叉树来组织的），假设空间中体素是如下左图的情景，然后我们将体素方格化，其中有颜色的区域表示体素有值，白色就是无值：

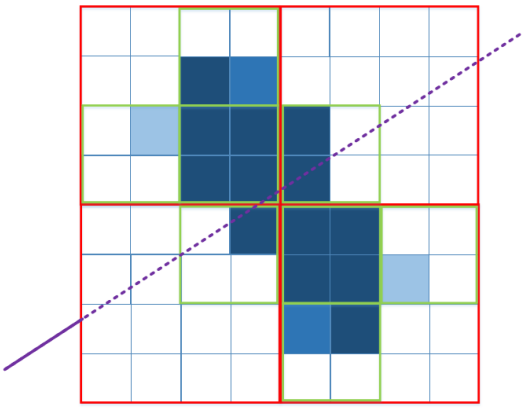


构建八叉树的根节点，然后一级节点划分为 4 个子区域，见下图红框；每个子区域又下分为四个子区域，但是只需要囊括有值的体素子区域，见下图右图的绿框：

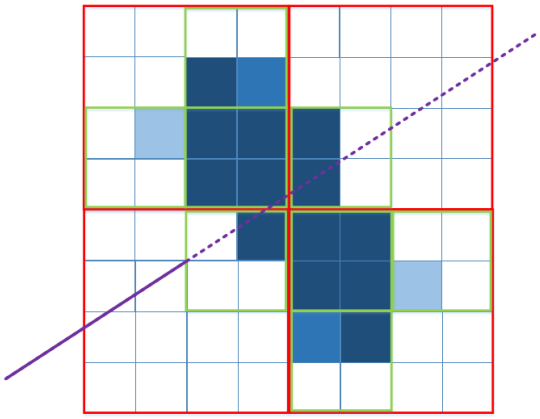


构建体素包围盒时是从底向上构建的，进行区间包围盒向上合并，直到构建完整个八叉树。

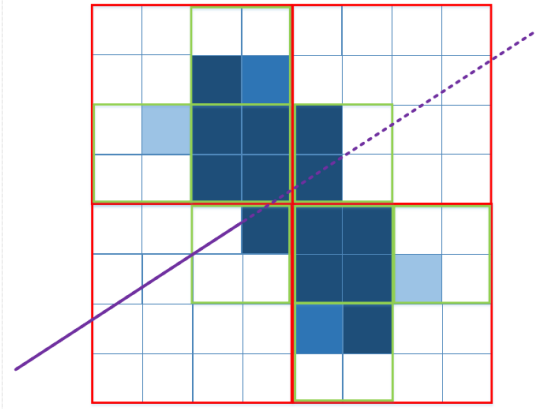
在进行最近邻体素搜索时，会经历如下几个步骤。首先跟包围盒求交，找到交点。下图中紫色表示投射的相机射线，实线表示已经前进的距离：



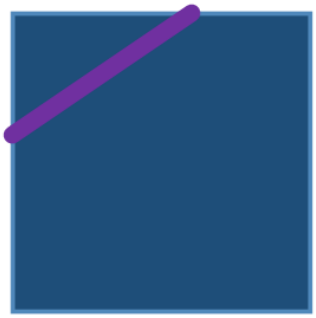
与左下角红框求交后，红框进栈，然后与红框下的框再进行求交，因为红框下只有一个子节点，而且射线也确实能跟该子节点相交，所以前进到绿框：



然后绿框进栈，绿框下面仍然只有一个节点，其他体素都没有值，而且射线能与该节点相交：



然后计算在该体素内的这一小段的着色值（对于传统 Ray-Casting 来说，是吸收发射后的值）：



从该体素出来之后，就达到了绿框的边界（到达边界需要前进的距离是早已算出的），因此绿框弹栈。然后也达到了红框的边界，红框弹栈。之后，射线又会与左上角的红框相交，因此左上角的红框进栈。其他过程不再赘述。

我们可以看到，一个小方框内所有位置的值都是相同的，对于八叉树组织的结构来说，要插值并不容易，需要能够再去访问到周边其他体素，因此效率不是很高。

三　其他体素八叉树

对于八叉树我了解并不是很多，其他的优化和加速技术在国内也没见到有相关的论文和资料介绍。在 [3, 4] 中有比较详细的八叉树方案，但自从 VDB 变得大火以后，一般的八叉树方法已经不再那么重要和热门。

其他内容以后如果有需要，或者研究中遇到再进行补充。

参考文献

[1] Laine S, Karras T. Efficient sparse voxel octrees–analysis, extensions, and implementation[J]. NVIDIA Corporation, 2010, 2(6).

[2] https://zhuanlan.zhihu.com/p/195730581

[3] https://anteru.net/blog/2008/voxels-sparse-octrees-virtualization/

[4] https://code.google.com/archive/p/efficient-sparse-voxel-octrees/