计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2020,29(6):235−240 [doi: 10.15888/j.cnki.csa.007404]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于虚拟现实的复杂场景加载优化算法①

叶　诚, 罗　训

(天津理工大学 计算机科学与工程学院, 天津 300384)
通讯作者: 叶　诚, E-mail: 1321034729@qq.com

摘　要: 目前大规模场景模型生成的需求量日益增加, 现提出了一种基于区域的自动 LOD (Levels Of Details) 构建算法. 该算法基于动态网格简化算法, 在游戏设计制作的过程中, 游戏开发人员会经常需要对美工部门人员提供的模型进行优化. 为了可以使模型的面数得到简化, 并且不改变模型的外观. 当今流行的 LOD 技术非常擅于处理这种情况, 判断模型与摄像机的距离如果超过一定范围之后, 自动调取不同层次的模型, 当模型距离摄像机很远的情况下使用面数低的模型替换高模, 这样可以提高帧率并且降低摄像机前的三角面以及顶点数从而减少渲染压力. 一般模型的简化分为静态和动态模型的简化.大部分情况下, 程序员会让美工部门提供几套不同面数的模型或者通过模型简化工具对高模进行减面并存成多个 Mesh, 并在程序运行的时候, 根据模型与摄像机的距离远近动态的替换 Mesh. 这是一种静态的方法.这里将尝试使用一种动态的网格简化和 LOD 技术相结合的方法. 这种新型算法的大大的简化了操作流程, 美术人员只需要提供一个模型, 程序员可以使用这种方法生成量级不同的低模, 根据摄像机与模型的远近自动的调取不同精度的模型.

关键词: 虚拟现实; 动态网格简化算法; LOD 技术; 自动 LOD 技术

引用格式: 叶诚,罗训.基于虚拟现实的复杂场景加载优化算法.计算机系统应用,2020,29(6):235–240. http://www.c-s-a.org.cn/1003-3254/7404.html

Rendering Optimization Algorithm for Large-Scale Scene Based on Virtual Reality

YE Cheng, LUO Xun

(School of Computer Science and Engineering, Tianjin University of Technology, Tianjin 300384, China)

Abstract: At present, the demand for large-scale scene model generation is increasing. A region-based automatic Levels Of Details (LOD) construction algorithm is proposed. The algorithm is based on the dynamic mesh simplification algorithm. In the process of game design and production, game developers often need to work on the art. The model provided by the department staff is optimized in order to simplify the number of faces of the model and not change the appearance of the model. Today's popular LOD technology is very good at handling this situation, judging the distance between the model and the camera. After a certain range is exceeded, the models of different levels are automatically retrieved. When the model is far away from the camera, the model with a low number of faces is replaced by the high-modulus. This can increase the frame rate and reduce the number of triangles and the number of vertices in front of the camera to reduce rendering stress. The simplification of the general model is divided into the simplification of static and dynamic models. In most cases, the programmer will let the art department provide several sets of different face models or reduce the face of the high model and save it into multiple mesh through the model simplification tool. When the program is running, according to the distance between the model and the camera, the mesh is replaced dynamically. This is a static method. Here we will try to use a combination of dynamic mesh simplification and LOD technology. This new algorithm

① 收稿时间: 2019-10-17; 修改时间: 2019-11-15; 采用时间: 2019-11-18; csa 在线出版时间: 2020-06-10

greatly simplifies the operation process, while the artist only needs to provide a model, then the programmer can use this method to generate low-modules of different magnitudes, and automatically pick up models of different precision according to the distance between the camera and the model.

Key words: virtual reality; dynamic mesh simplification algorithm; LOD technology; automatic LOD technology

现阶段场景模型生成技术并不发达, 程序员在处理生成模型场景问题上还存在很多情况没有解决.本研究在游戏引擎的实验环境下, 讨论了大规模复杂场景生成阶段的一些重要问题, 优化了相关算法. 本文主要研究大规模复杂场景生成后场景模型的优化, 根据模型和摄像机的远近实时调用不同层级的模型方面进行研究. 大型复杂生成场景文件在游戏引擎中的漫游存在着模型渲染上和模型调取的问题, 该研究使用了模型加载及其渲染优化技术, 优化了场景漫游的工程简化了制作流程; 在模型场景渲染技术上, 通过使用一种基于 LOD 场景模型的新型算法, 解决了大规模复杂场景的实时渲染有许多技术上的难点, 主要包括海量地形数据的表示和组织、动态场景的管理以及场景的真实感表现等. 即按照生成模型的规模大小, 使用模型切割代码将模型切割成尺寸大小均等的不同尺寸, 满足工程对于场景漫游的要求; 相关论文研究了关于大规模地形的快速渲染方法, 其思想是将之前分割好的地块都先简化为一些不同细节层次的网格模型, 存储在外部存储介质上. 程序根据摄像机距离模型的远近调取合适的地形块导入内存, 再进行模型的渲染. 这种方法的缺陷是增加了很多的步骤, 在不同的模型切换时会产生视觉上的卡顿问题. 本文对于自动 LOD 技术进行探讨采用网格简化算法结合视点的动态 LOD 技术, 提高了帧率减少了当前占用 CPU 进行计算的时间, 并且减少了摄像机下的三角形和顶点数从而减少了渲染的工作量.

本文在游戏引擎中使用天津工业大学的生成模型(如图 1) 进行漫游, 采用不同方法对该工程进行优化处理, 提出将 LOD 技术和动态网格简化算法并行计算形成一种新型的自动 LOD 构建算法, 主要贡献可以概括为:

(1) 将游戏引擎中 LOD 技术与动态网格简化算法并行节省渲染时间.

(2) 提出了自动 LOD 构建方法只需要使用一套模型就可以应用 LOD 技术.

图 1 天津工业大学模型

1 相关技术介绍

在大型场景中经常使用的优化算法主要有: LOD 技术, 遮挡剔除算法, 动态加载, 本文主要对这 LOD 技术和网格简化算法进行了结合改进从而提出了自动 LOD 算法.

1.1 LOD 技术

LOD 技术, 也称为层次细节技术, 是在实时渲染显示系统中采取的细节省略技术, 此技术于 1976 年由 Clark 提出[1]. 作为目前主要的场景优化方法, 它根据物体的重要性、视点的相对速度或者位置等衡量标准, 动态改变场景中三维模型的复杂程度, 模型距离视点越近, 显示的模型细节越丰富, 从而降低了 CPU 和 GPU 需要处理的的顶点数量, 提高场景的渲染速度[2]. 在复杂模型的动态显示中, 当观察点距离某一物体很近时, 该物体的图像将在屏幕上占据较多的像素点; 而当观察点距离某一物体很远时, 该物体的图像只能在屏幕上占据很少的像素点. 在这种情况下, 用大量的面去精确表示该物体是不必要的[3]. 认为当物体覆盖屏幕较小区域时, 可以使用该物体描述较粗的模型, 并给出了一个用于可见面判定算法的几何层次模型, 以便对复杂场景进行快速绘制. LOD 技术在不影响画面视觉效果的条件下, 通过逐次简化景物的表面细节来减少场景的几何复杂性, 从而提高绘制算法的效率. 该技术通常对每一原始多面体模型建立几个不同精度的几何模型.与原模型相比, 每个模型均保留了一定层次的细节. 在绘制时, 根据不同的标准选择适当的层次模型来表示物体. LOD 技术具有广泛的应用领域. 目前在实时图

像通信、交互式可视化、虚拟现实、地形表示、飞行模拟、碰撞检测、限时图形绘制等领域都得到了应用, 已经成为一项要害技术. 很多造型软件和 VR 开发系统都开始支持 LOD 模型表示. LOD 生成的一种常用方法是网格简化. 目前常用的网格简化方式有边压缩、面收缩、点聚合和顶点删除等[4].

1.2 遮挡剔除算法

遮挡剔除算法利用遮挡剔除 (occlusion culling) 技术, 在游戏引擎中使用遮挡剔除时, 会在渲染对象被送进渲染管线之前, 将因为遮挡而不会被看到的隐藏对象或隐藏面进行剔除, 从而减少了每帧的渲染数据量, 提高了渲染性能[5]. 可见性剔除是提高大规模复杂场景的加速技术之一, 目的是减少送入图形管线的面片数量, 降低场景规模增长对图形系统的负担[6]. 对于给定的场景和观察视点, 通过判断场景中物体的可见性, 快速拒绝那些显然不可见的绘制元素, 从而减少送入图形绘制管线的几何复杂度[7]. 在遮挡密集的场景中, 性能提升会更加明显. 遮挡剔除是当一个物体被其他物体遮挡住而不在摄像机的可视范围内时, 不对其进行渲染. 在 3D 图形计算中并不是一个自动进行的过程, 因为在绝大多数情况下离相机最远的物体首先被渲染, 靠近摄像机的物体后渲染, 并覆盖先前渲染的物体 (这种重复渲染又叫做“OverDraw”), 它不同于视锥剪裁. 视锥剪裁只是不渲染摄像机视角范围外的物体, 而对于那些被其他物体遮挡, 但是依然在镜头范围内的物体, 则不会被视锥剔除. 当然当你使用遮挡剔除时, 视锥裁剪还是会生效的. 由于地形场景一般较为复杂, 对 CPU 开销很大. 为了降低地形的遮挡剔除过程中 CPU 的负荷, 提出了以 GPU 具有遮挡查询功能为基础的遮挡剔除算法[8].

2 流程优化

在游戏引擎中, 如果想使用传统的 LOD 技术 (如图 2) 就必须先将面数最大的模型分别简化为几种面数不同, 导出后再放入游戏引擎中使用 LOD 技术优化工程. 在这种新型的 LOD 技术中, 动态网格简化算法采用网格预先简化的方法事先生成 LOD 模型避免了直接使用网格简化算法所需完成的大量计算, 有效地降低了算法的时间开销. 在串行计算环境下, 网格简化算法使得一些较小规模 (包含10^5以下量级三角面单元) 的网格模型的在交互帧率下的动态 LOD 模型构建成为可能. 然而在面对大规模网格模型 (包含10^5及以上量级三角面单元) 时, 却仍无法满足交互帧率绘制需求.

在动态网格简化算法使用在大规模复杂模型的时候其时间性能需要有所提升, 该理论基于网格的动态 LOD 构建方法, 并且使用了三维模型的简化算法, 创造出一个新型高效率的自动 LOD 算法, 如图 3.

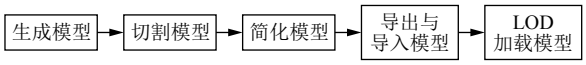

图 2 传统 LOD 加载算法处理步骤

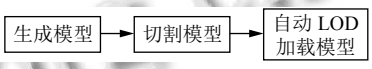

图 3 自动 LOD 加载算法处理步骤

算法主要具备如下优势:

(1) 适用于较大面数的模型, 具备良好的扩展性.

(2) 只需要一套模型.

(3) 可以自由缩减模型的面数.

3 自动 LOD 算法

3.1 自动 LOD 加载模型流程

自动 LOD 加载模型总体流程如图 4 所示.

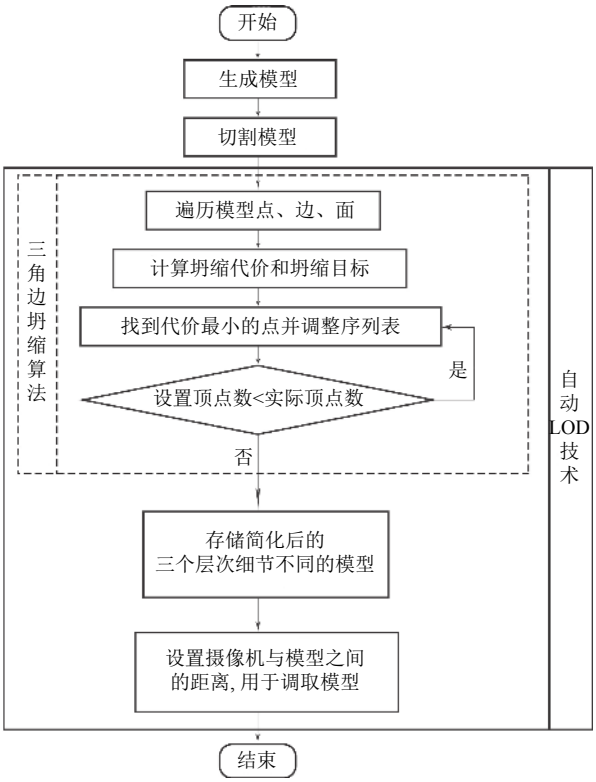

图 4 自动 LOD 加载模型流程

3.2 三角边坍缩

为了能够对模型使用网格简化选择三角边坍缩的办法实现, 可以让两个顶点并成一个顶点, 如图 5 所示.

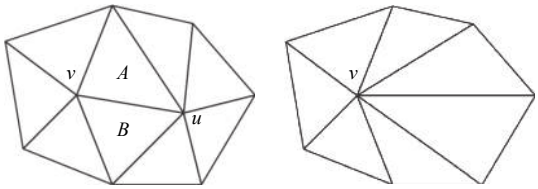

图 5 三角边坍缩

要实现边 uv 的坍缩需要删除两侧的面 A 和 B, 使用点 v 来代替点 u, 需要进行点的连接操作, 目标是连接点 v 和点 u 的邻点还要除去点 u, 点 v 为点 u 的坍缩目标.

对于 1 个实体模型 (具有封闭的边界, 根据边界可以将空间分为模型内部和模型外部 2 部分), 1 次坍缩, 可以移除 2 个三角面, 3 条边和 1 个顶点. 通过反复的迭代, 最终就会使模型简化到预期的面数.

如果想要删除掉某个点还要尽量保持原始模型外观这就需要考虑到坍缩的代价公式如下:

$$cost(u,v) = \|u-v\| \times \max_{f\in T_u}\{\min_{n\in T_{uv}}\{(1-f.normal\cdot n.normal)\div 2\}\} \tag{1}$$

在式 (1) 中顶点 u 的三角形被包含于集合 T_u 里, 顶点 u 和顶点 v 的三角形被包含于集合 T_{uv} 里.

式 (1) 中需要考虑两个顶点点 u 坍缩到点 v 并且需要将顶点 u 删除这一系列操作所花费的代价. 首先需要处理的是边, 在网格简化的流程中, 无关紧要的部分是首要剔除的部分. 其次应该要注意顶点 u 附近的曲率, 因为只有当点周围的曲率越平坦, 那么这个点也就越不重要. 值得关注的是, 连个顶点从点 u 坍缩到点 v 和从点 v 坍缩到点 u 的代价有所不同.

根据上述公式, 依赖于针对点与其周围邻点坍缩的代价进行计算, 再决定以代价最小的邻点为目标进行坍缩运算.

3.3 实现细节

根据以上的描述, 可以将实现分为以下步骤:

(1) 遍历模型上点、边和面的关系.

(2) 计算坍缩代价和坍缩目标, 并排序.

(3) 通过公式找到坍缩代价最小的顶点, 通过衡量相邻顶点的的坍缩代价和坍缩目标, 实时调整有序列表.

(4) 再次遍历顶点, 计算设置顶点数量是否小于现顶点数量, 如果是则重复第 (3) 步.

因为在程序运行的时候不能够实时进行上述步骤, 特别是第 (2) 步, 因为全部的顶点被遍历了很多次, 不必要的工作量过于繁重. 因此, 可以考虑将工作分成离线烘培和运行时这两个步骤, 具体实现步骤如图 6 所示.

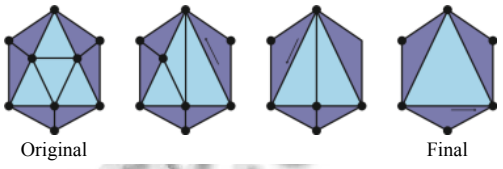

图 6 实现步骤

3.4 离线烘焙

离线烘焙会输出两个 int 数组: *permutation* 和 *vertex_map*.

步骤:

(1) 收集顶点信息: 主要是顶点位置和 *index*, 另外需要初始化两个列表: 包含顶点的三角面列表和顶点的邻居列表.

(2) 收集三角面信息: 需要得到模型上所有三角面里所含有的顶点和计算法线 (目的是计算曲率). 并且不断更新顶点的三角面列表将新的三角形填入列表, 还需要使每个顶点加入另外两个顶点的邻居列表中.

(3) 通过不断分析点与该点邻点的坍缩代价, 决定坍缩代价最小的邻点为最终的目标.

(4) 依据坍缩代价对所有的顶点进行排序.

(5) 替换坍缩代价最小的顶点: 根据坍缩公式设法得到代价最小的点 u 和它的邻边目标点 v, 读取包含顶点 u 的三角面列表, 只要哪个三角面里面含有顶点 v 就删除该三角面, 如果不能遍历到就将顶点 u 与顶点 v 进行交换. 再次分析顶点 u 邻点的坍缩代价以及坍缩目标, 根据最新的信息重新编排列表.

(6) 令 *permutation*[*u.index*]=当前顶点数量, 令 *vertex_map*[*u.index*]=*v.index* (如果没有坍缩目标, 则赋值为–1).

(7) 目前的点数大于 0, 如果是将返回第 (5) 步.

permutation 保存了每个顶点被移除的倒数次序 (1 是最后被移除的, 最大的是第一个被移除的), *vertex_map* 保存了每个顶点的坍缩目标的位置.

3.5 运行时

输入需要绘制的最大顶点数量 n.

步骤:

(1) 遍历三角面.

① 获得当前三角面的三个顶点的 *index*, 即 *idx0*、*idx1* 和 *idx2*.

② 如果 *permutation*[*idx0*]=*n* 则 *idx0*=*vertex_map*[*idx0*], 否则执行第⑤步.

③ 如果 *idx*0==–1 or *idx*1==*idx*0 or *idx*2==*idx*0, 该三角面不参与绘制. 同理映射并判断 *idx*1 和 *idx*2.

④ 返回第①步.

⑤ 当前三角面加入绘制列表.

(2) 根据三角面的数据, 整理顶点属性.

值得注意的是, 员网格信息将不会被删除, 再程序运行的时候, 还能够再任意的层次细节模型上切换.

3.6 细节优化

如果想要有出色的游戏性能以及更佳的画面效果, 该研究还可继续改进上述步骤:

(1) 最小堆排序: 在离线烘培的第 (4) 步中, 应该考虑需删除点的邻点的坍缩代价, 同时坍缩列表也要随之更新. 因此可以选择最小优先队列 (最小堆) 记住顶点和坍缩代价, 根据新的信息重新完善顶点的顺序.

(2) 删除不用顶点: 需要提高帧率并通过减少摄像机前的三角面和顶点降低渲染压力, 在第 (2) 步中, 需要依靠三角面的数据对 Mesh 的顶点数组 (还有 *uv*、*uv2*、*colors*、*normals*、*tangents*、*boneWeights* 等) 和三角面数组进行重新排列.

(3) 边界点处理: 在处理过程中有两个问题比较棘手, 一种是不闭合的三角面, 例如飘带、披风等. 另外一种是两个垫高点虽然有相同的坐标, 但是不同的 *uv* 或 *normal*, 例如 Unity3D 的 Sphere 是有一条接缝的. 假如这两种情况不进行参考, 在坍缩的过程中, 将会出现没能指出对的坍缩目标来代替原顶点, 会发生镂空和破损的情况. 根据上面说的问题, 我们在考虑坍缩消耗的问题上, 会将这些边缘点的曲率设为 2 (可以在编辑器里调整).

(4) 内存优化: 如果不想多次重新建立 Mesh 的 *vertices* 等数组, 可以选择 unsafe 的方式来改变数组的大小.

(5) JobSystem: 离线烘焙中使用了 JobSystem 来加速烘焙.

4 实验结果与分析

4.1 实验设计

本论文选取天津工业大学的生成模型场景为例子, 验证过程为自动 LOD 算法与遮挡删除技术, 动态加载, 普通漫游分别对同一场景随即视点位置进行渲染, 对比各项渲染数据.

4.2 实验结果分析

FPS (Time per frame and FPS): Frames Per Seconds 表示引擎处理和渲染一个游戏帧所花费的时间, 该数字主要受到场景中渲染物体数量和 GPU 性能的影响, FPS 数值越高, 游戏场景的漫游体验会更加平稳和舒适.

CPU: 获取到当前占用 CPU 进行计算的时间绝对值或时间点, 如果 Unity 主进程处于挂断或休眠状态时, CPU time 将会保持不变.

Tris & Verts: 摄像机下的所有三角形和顶点这也是主要瓶颈.

图 7 中描述了各个算法之间在同一场景下, 摄像机前三角面和顶点的数量, 三角面与顶点的数量呈反比, 数量越大计算机所需要做的工作就越多.

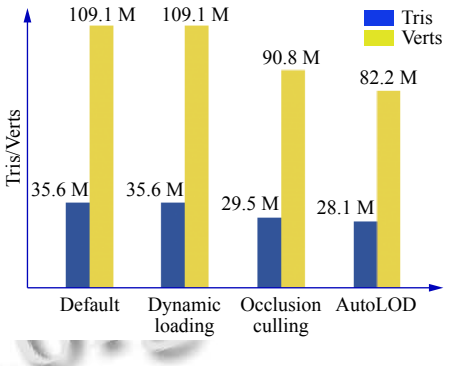

图 7 实验数据对比

图 8 中描述了各个算法之间在同一场景下, 帧率和 CPU 进行计算的时间, 因为 FPS 和 CPU 的数值与三角面和顶点数量有直接关系, 所以提出自动 LOD 算法.

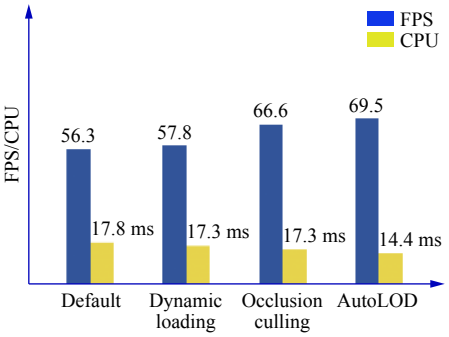

图 8 实验数据对比

文中使用的方法自动 LOD 考虑到了之前几种优化方法的缺点 (算法优缺点对比如表 1 所示), 通过使用三角边坍缩算法, 不仅节省了模型导出和导入步骤还节省了大量的内存空间.

表 1 算法优缺点对比

算法名称	简称	优点	缺点
动态加载	Dynamic loding	帧率高, 运行速度快	大规模地形不能全面观察
遮挡删除	Occlusion culling	帧率高, 运行速度快	运行时需要占用大量内存
LOD		帧率高, 运行速度快	需要占用额外的存储空间
自动 LOD	AutoLOD	帧率高, 运行速度快	

5 总结

针对大型场景漫游提出了新的自动 LOD 技术, 实验解决了只使用一套生成模型就可以应用 LOD 技术的方法, 减少了工作的步骤, 对工程中大型生成场景模型的漫游实施了优化处理,不仅提高游戏帧率, 而且减少当前占用 CPU 进行计算的时间, 还节省了不必要的存储空间, 具有一定的参考价值和实际意义.

参考文献

1 张绍江. LOD 技术在 Unity 场景优化中的应用. 中国科技信息, 2015, (13): 86–87. [doi: 10.3969/j.issn.l1001-8972.2015.13.029]

2 吴宪君. 高斯模糊算法的改进及图像处理应用. 计算机光盘软件与应用, 2013, 16(19): 129, 131.

3 吴有用, 万旺根, 金龙存, 等. 一种新的连续性 LOD 实现算法. 微电子学与计算机, 2010, 27(6): 185–187.

4 费红辉, 王毅刚. 大规模场景分割及 LOD 结构生成算法研究. 计算机应用与软件, 2012, 29(7): 227–230. [doi: 10.3969/j.issn.1000-386X.2012.07.066]

5 张绍江. 遮挡剔除技术在 Unity 场景优化中的应用. 安徽电子信息职业技术学院学报, 2015, 14(4): 22–24. [doi: 10.3969/j.issn.1671-802X.2015.04.006]

6 聂俊岚, 郑鹏. 一种基于 GPU 的遮挡剔除算法改进研究. 微计算机信息, 2009, 25(27): 29–31.

7 高宇, 邓宝松, 吴玲达, 等. 一种复杂场景遮挡剔除的优化算法. 计算机辅助设计与图形学学报, 2007, 19(5): 583–588. [doi: 10.3321/j.issn:1003-9775.2007.05.007]

8 冉光灿, 谢晓尧, 景凤宣. 复杂 3D 地形的遮挡剔除算法研究. 福建电脑, 2012, 28(8): 4–7. [doi: 10.3969/j.issn.1673-2782.2012.08.003]