计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2019,28(6):178−182 [doi: 10.15888/j.cnki.csa.006962]

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

基于虚拟现实的渲染优化算法①

李媛媛, 罗　训

(天津理工大学 计算机科学与技术学院, 天津 300384)
通讯作者: 罗　训, E-mail: luo@tjut.edu.cn

摘　要: 随着虚拟现实技术的不断发展, 对虚拟场景的真实度要求也越来越提高. 然而在虚拟场景中, 复杂的地形、大量的植被和建筑使需要渲染的数据量大得惊人, 故渲染速度成为了虚拟现实技术的一大瓶颈. 现有的研究并不能很好的提升虚幻引擎中的渲染速度, 还会出现“突越”和对视野外模型剔除效果差的问题. 本文提出一种游戏线程与渲染线程并行和双层裁剪算法. 首先在虚幻引擎中将游戏线程与渲染线程并行以提升渲染速度, 然后使用淡入淡出细节层次算法进行第一层裁剪, 最后使用缓慢剔除算法进行第二层裁剪, 提升剔除效果. 实验证明, 该方法与串行线程相比渲染速度提升了 40%, 与传统单层裁剪算法相比, 帧率也达到了 55.

关键词: 虚拟现实; 淡入淡出层次细节算法; 缓慢剔除算法; 双层裁剪算法

引用格式: 李媛媛,罗训.基于虚拟现实的渲染优化算法.计算机系统应用,2019,28(6):178–182. http://www.c-s-a.org.cn/1003-3254/6962.html

Rendering Optimization Algorithm Based on Virtual Reality

LI Yuan-Yuan, LUO Xun

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

Abstract: With the continuous development of the virtual reality technology, the requirement for the authenticity of the virtual scene is getting greater and greater. However, there are a lot of data to be rendered in the virtual scene, which is caused by the complex terrain as well as a large number of vegetation and buildings. As a result, the rendering speed becomes a bottleneck of the virtual reality technology. In the existing research, it is impossible to improve the rendering speed of the illusion engine very well. Some problems can also appear about “breakthrough” and the poor effect of removing the invisible model. In this paper, a kind of parallel and double-layer cut algorithm for game threads and rendering threads is presented. Firstly, game threads and rendering threads are paralleled in the illusion engine to improve the rendering speed. Then, the fade in-and-out levels of detail algorithm are applied to cut the first level. Finally, the fade culling algorithm is adopted to cut the second level to improve the culling effect. The experiment shows that the rendering speed of the above method is improved by forty percent compared with the rendering speed of serial threads. Its frame rate is also improved by fifty-five percent compared with the traditional single-layer cut algorithm.

Key words: virtual reality; fade in-and-out levels of detail; fade culling algorithm; double-layer cut algorithm

① 基金项目: 国家重点研发计划 (2017YFB1002600); 天津市科委项目 (15ZXHLGX00240,16JCZDJC30500,16JCTPJC49700); 国家级、天津市级和天津理工大学校级大学生创新创业训练计划; 虚拟现实技术与系统国家重点实验室开放课题 (BUAA-VR-16KF-14); 北京市食品安全大数据技术重点实验室开放基金 (BKBD-2016KF03)

Foundation item: National Key Research and Development Program of China (2017YFB1002600); Tianjin Municipal Science and Technology Commission (15ZXHLGX00240, 16JCZDJC30500, 16JCTPJC49700); China Undergraduate Training Programs for Innovation and Entrepreneurship Grants at National, Tianjin Municipal, and Tianjin University of Technology Levels; State Key Laboratory of Virtual Reality Technology and Systems of China Fund (BUAA-VR-16KF-14); Beijing Key Laboratory of Big Data Technology for Food Safety Fund (BKBD-2016KF03)

收稿时间: 2018-12-13; 修改时间: 2019-01-08, 2019-01-22; 采用时间: 2019-01-28; csa 在线出版时间: 2019-05-25

随着虚拟现实的不断发展, 人们在使用虚拟现实系统的过程中, 对虚拟场景的真实性及实时渲染性的要求也逐步增高, 而随之虚拟场景也变得越来越复杂, 这就意味着所需要渲染的数据会大得惊人. 现如今随着游戏产业的蓬勃发展, 虚幻引擎的免费开放吸引了无数游戏开发人士以及学生的使用, 在虚幻引擎中创建一个具有真实感的场景势必包含地形、植被、建筑、氛围、特效等, 为了使场景的真实感更强, 不会出现场景卡顿的现象, 最重要的是需要在庞大复杂的虚拟场景中提高渲染效率.

随着计算机显卡处理能力的不断提高, 在游戏场景中, 越来越多的开发人员倾向于使用三维场景的光影、动画、粒子特效等, 使场景画面渲染的真实度越来越高, 让用户有身临其境的感觉[1]. 引擎中的优化占据着尤为重要的地位, 其可优化的覆盖范围非常广泛, 涉及到内存的分配、多线程加载、层次细节算法 (LOD)[2]等, 要想高效率解决引擎优化问题, 必须要找出实际瓶颈所在, 然后对症下药, 提高工程渲染运行效率.

本文在虚幻引擎中进行三维重建天津理工大学, 对整个工程进行渲染加速, 提出了将游戏线程与渲染线程并行计算和双层裁剪算法, 主要贡献可以概括为:

(1) 将虚幻引擎中游戏线程与渲染线程同步并行执行, 节省渲染时间;

(2) 提出了淡入淡出层次细节算法, 使用高度补偿的方法, 使每个层级之间过渡平滑, 从而解决传统的层次细节算法所出现的“突越”现象;

(3) 提出了缓慢剔除算法, 使用两个剔除距离, 利用不透明度来实现缓慢剔除, 从而解决传统的剔除算法所出现的“突越”现象;

(4) 提出了双层裁剪算法, 使用淡入淡出层次细节算法进行第一层裁剪, 再使用缓慢剔除算法进行第二层裁剪, 从而加快了渲染速度.

1 相关技术介绍

近年来在复杂的三维虚拟场景中最常用的优化算法主要有: 层次细节算法[3]和剔除算法[4], 这两种算法都是单层裁剪算法, 本文主要对这两种算法进行了改进, 并提出了双层裁剪算法.

1.1 层次细节算法

层次细节算法, 简称 LOD(Levels of Detail)[2], 是根据模型的节点在显示环境中所处的位置和重要度, 决定物体渲染的资源分配, 降低非重要物体的面数和细节度, 它的目的是通过保留重要视觉特征来生成简化的模型, 以降低场景复杂度, 实现场景实时绘制, 从而获得高效率的渲染运算[5].

在现实世界中, 人眼视角是有一定盲区的, 并不能看到 360 度的场景, 并且随着视线向远处延长, 看到的东西会越来越模糊. 因此在虚幻引擎中也会选择视野范围内的物体进行绘制, 将场景中的目标模型按照一定的区域划分法进行快速分割, 并将模型对象组织到树结构中, 然后从根节点开始进行视域剪裁测试, 当某些节点测试失败, 表示其位于视域体制外, 可在渲染之前将其剔除, 其子孙节点无需再进行测试, 可直接剔除[6].

传统的 LOD 算法将每个层级的三角面片数进行了大量的裁剪, 该方法虽然能提高渲染效率, 但在视觉上会出现“突越”现象, 即在层级之间切换不平滑. 故本文针对该问题提出了淡入淡出层次细节算法.

1.2 剔除算法

传统的剔除算法[7], 简称 CULL, 保证在远距离的时候对植被不进行渲染, 以及被遮挡的植被不进行渲染, 从而提高渲染效率. 传统剔除算法需要对每个单元进行两次可见性剔除判断, 从而避免了遮挡部分也进行渲染, 提高了渲染效率[8].

目前的遮挡剔除研究算法主要可以分为动态遮挡剔除算法和静态遮挡剔除算法[9]. 由于本文的实验场景是虚拟校园, 不考虑动态物体, 故采用了静态遮挡剔除算法. 静态遮挡剔除算法需要对场景进行预处理, Zaugg[10]等人提出了将基于区域的室内静态场景的方法扩展到了地形场景等. 将场景进行等单元划分, 依据视野向量与三角面片的平面法向量的夹角θ来判断是否剔除, 若$0 < \theta < 90$时, 可见; 若$90 < \theta < 180$时, 不可见, 进行剔除.

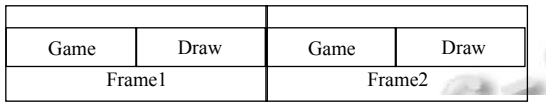
虽然传统的剔除算法能提高渲染效率, 但其在计算过程中将考虑到的向量为三维向量, 其计算量比二维向量增加很多; 且其只设置一个剔除距离, 若视野从一个可见去区域移动到一个不可见区域, 就会出现“突越”现象, 这样显得就很突兀, 故本文提出了缓慢剔除算法.

2 线程优化

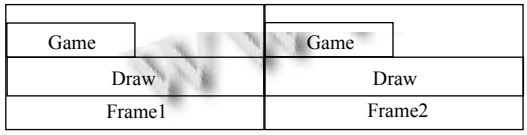
在虚幻引擎中, Frame 时间是一帧所花费的总时间, Game 是游戏线程, 它不与渲染直接关联, 如若游戏线程很慢, 则通常是工程内程序不够优化, 占据内存过多; Draw 是渲染线程, 包括图形应用程序编写接口 API (Application Programming Interface) 以及绘图调用

的设置, 若渲染线程很慢, 则说明工程中需渲染的三角面片过多. 在虚幻引擎中使用的是单线程, 游戏线程和渲染是串行的, 由于渲染中的数据通常依赖于游戏逻辑线程, 所以通常会先进行游戏逻辑的任务, 然后再进行渲染, 如图 1(a) 所示, 此时一帧所占的时间是 Game 和 Draw 两个线程执行帧数的总和, 所以场景的渲染效率相对较慢.

本文在虚幻引擎中将游戏线程和渲染分离成两个单线程, 如图 1(b) 所示, 并行起来执行, 此时一帧的时间就是 Game 和 Draw 各自执行时两个中耗时最长的帧数, 这样整个工程的运行效率就会得到显著的提升.



(a) 游戏线程与渲染线程串行



(b) 游戏线程与渲染线程并行

图 1 线程分布

将游戏线程与渲染线程并行之后还需要将其进行同步处理, C++中常用锁或临界区的方式来解决同步问题. 本文采用的是栅栏的技术即 Render Command Fence 机制. 所谓的栅栏技术就是指在下一次进行读取操作前, 所有的写操作都被写入了内存, 因而每次的读操作都是更新后的内存, 从而用来防止游戏线程和渲染线程运行相差过大, 导致整个工程运行错乱.

3 双层裁剪算法

3.1 第一层裁剪—淡入淡出层次细节算法

传统的层次细节算法在随着不同 LOD 等级之间进行切换时, 会出现顶点高度值的突然变化, 从而导致视觉上出现“突越”现象, 本文提出了淡入淡出层次细节算法, 对顶点的高度值进行缓慢变化处理即高度补偿来解决该现象.

现假设由一层切换到另一层时, 某一顶点由 A 切换为A', 其高度差为Δh, 总过渡时间为 t, 在切换过程中, 本文先对高度进行补偿, 然后再进行等级之间的切换. 如式 (1) 所示, 其中Δt为当前已过渡的时间.

$$h_{A'} = h_A + \frac{\Delta t}{t} \cdot \Delta t \tag{1}$$

在大规模的虚拟场景中使用层次细节算法可以有效地裁剪掉所设置的视野范围外的模型, 同时根据模型在视野范围内的比例可以选择精裁剪或是粗裁剪, 故本文采用淡入淡出层次细节算法进行第一层裁剪.

3.2 第二层裁剪—缓慢剔除算法

传统的剔除算法主要是进行三维向量的计算, 且在剔除不可见面时会出现“突越”现象, 本文提出了缓慢剔除算法, 将传统的剔除算法中三维向量的计算优化为二维向量的计算, 同时设置了两个剔除距离, 利用不透明度来解决“突越”现象.

二维向量的计算主要是通过投影变换将视野向量坐标归一化为$\vec{v} = (0,0,1)$, 假设某个三角面片为ΔABC, 其中点 A、B、C 的坐标分别为$A(x_1,y_1,z_1)$, $B(x_2,y_2,z_2)$, $C(x_3,y_3,z_3)$, 则ΔABC的法向量$\vec{n}$的计算如式 (2) 所示:

$$\vec{n} = \overrightarrow{AB} \times \overrightarrow{BC} = \begin{vmatrix} \vec{x} & \vec{y} & \vec{z} \\ x_2 - x_1 & y_2 - y_1 & z_2 - z_1 \\ x_3 - x_1 & y_3 - y_1 & z_3 - z_1 \end{vmatrix} \tag{2}$$

将$\vec{n}$与视野向量$\vec{v} = (0,0,1)$点乘如式 (3) 所示:

$$\vec{n} \cdot \vec{v} = (x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1) \tag{3}$$

由式 (3) 可以看出此时的点乘结果与 Z 轴坐标无关, 将其转换成了二维向量坐标的计算, 此时由公式 (4) 计算角度θ:

$$\vec{n} \cdot \vec{v} = |\vec{n}| \times |\vec{v}| \times \cos\theta \tag{4}$$

根据θ来进行判断哪些面可见, 哪些面需要剔除, 在剔除时, 为了避免视觉上产生“突越”现象, 同时缓慢剔除算法将剔除算法设置两个剔除距离: 起始剔除距离 S 和终止剔除距离 E, 在材质中将植被蒙版和 Vertex Color 的 alpha 通道进行相乘, 赋值给不透明度或蒙版值. 当模型在视野范围内后, 会对模型距离摄像机的距离进行计算, 设模型距离摄像机的距离为 d, 然后与起始距离和终止距离进行比较, 若$d \leqslant S$, 则模型不进行剔除, 即进行细渲染; 若$S \leqslant d \leqslant E$, 则模型会根据距离值的增大而之间调节不透明度; 若$d \geqslant E$, 则此时的模型完全在视野范围外, 只需要将其及之后的所有模型完全剔除即可. 使用改进的剔除算法会实现一种随着距离淡入淡出的效果, 使效果更佳真实.

由于在虚拟场景中会出现大量的遮挡现象, 而剔除算法会根据每个三角面片进行可见性判断再进行剔除, 故本文采用缓慢剔除算法进行第二层裁剪, 对遮挡部分及模型背面进行二次裁剪, 从而提高渲染效率.

3.3 总结

假设淡入淡出层次细节算法的层级为 3 层, 其对应模型所需占屏幕尺寸 (Screen Size) 分别为 M1、M2、M3, 根据模型所占屏幕尺寸进行相应比例裁剪, 且在每个层级之间切换时使用高度补偿的方法解决“突越”现象, 缓慢剔除剔除算法的视野向量与三角面片的平面法向量的夹角θ, 剔除的起始距离为 S, 终止距离为 E, 整个双层裁剪算法的总流程图如图 2 所示.

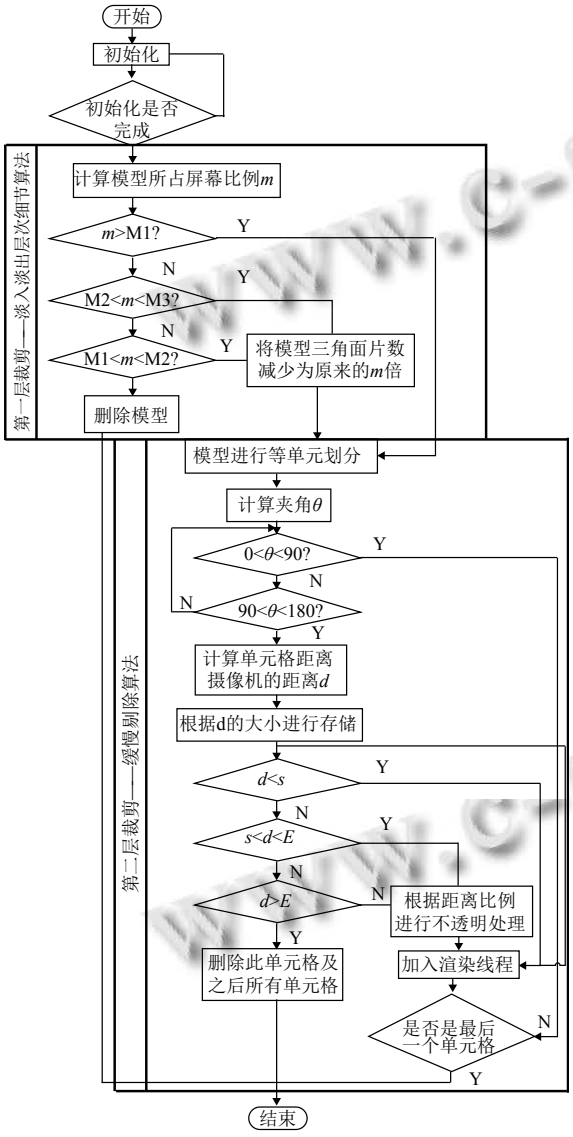


图 2 双层裁剪算法总流程图

现将淡入淡出层次细节算法、缓慢剔除算法以及双层裁剪算法进行简称、特点的总结, 方便实验时进行简单表述, 总结如表 1 所示.

4 实验结果与分析

4.1 线程串行、并行对比实验

本文将大量的场景组件的位置计算放到异步线程中, 利用动态组件处理将其转移到异步纹理流任务, 使它们能够与其他游戏线程任务得到并行的处理, 从而提高渲染效率, 本文对游戏线程与渲染线程的串行、并行做了多组实验, 用 Frame 数据来表示对比结果, 对比实验结果如图 3 所示.

表 1 算法总结

算法名称	简称	特点
淡入淡出层次细节算法	FLOD	单层裁剪
缓慢剔除算法	FCULL	单层裁剪
双层裁剪	DCut	淡入淡出层次算法进行第一层裁剪 缓慢剔除算法进行第二层裁剪

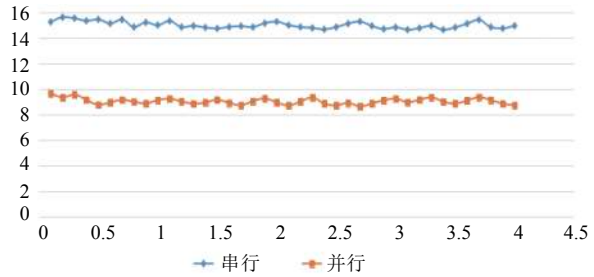


图 3 串行、并行实验 Frame 对比

由图 3 可以看出游戏线程与渲染线程串行时的 Frame 在 15 上下浮动, 而游戏线程与渲染线程并行时的 Frame 在 9 上下浮动, 一帧所花费的时间减少了 40% 左右. 但是仅依靠线程并行, 大规模的虚拟场景的渲染并不能有很大的改善.

4.2 LOD 和 FLOD 实验对比

本文以天津理工大学为整个实验场景, 设置三个 LOD 等级, 为了对比明显, 将分别对建筑物和植被的 Screen Size 参数设置一样, 如表 2 所示.

表 2 LOD 等级及 Screen Size 设置

LOD 等级	0	1	2
Screen Size	0.6	0.3	0.1

在每个层级之间进行切换, 看是否会出现“突越”现象, 实验结果如表 3 所示.

表 3 LOD 算法与 FLOD 算法实验对比表

算法名称	层级之间切换是否会发生“突越”			
	0→1	1→0	1→2	2→1
LOD 算法	是	是	是	是
FLOD 算法	否	否	否	否

由表 3 可以看出在进行相邻等级之间切换时, 本文提出的淡入淡出层次细节算法很好的解决了“突越”现象.

4.3 CULL 和 FCULL 对比实验

由于本文是利用缓慢剔除算法进行第二层剔除, 故同样需要保证其不会发生“突越”现象, 本文将传统的剔除算法的剔除距离设置为 400 cm, 将缓慢剔除算法的剔除的起始距离 S 设置为 400 cm, 剔除终止距离 E 设置为 600 cm, 来进行实验对比, 实验结果如表 4.

表 4 CULL 算法与 FCULL 算法实验对比表

算法名称	是否发生“突越”
CULL 算法	是
FCULL 算法	否

由表 4 可看出, 采用缓慢剔除算法设置两个剔除距离, 利用不透明度可以实现模型的缓慢剔除, 避免了“突越”现象.

4.4 双层裁剪算法与单层裁剪算法对比实验

本文提出的双层裁剪算法, 首先对整个场景进行淡入淡出层次细节算法裁剪, 对不在设置视野范围内的远处物体进行剔除, 仅仅加载渲染视野范围内的物体; 然后对视野范围内的物体进行单元分割进行第二次可见性判断, 将在视野盲区进行二次裁剪, 如建筑物的背面等.

在游戏引擎中最能体现渲染效率的就是帧率 (Frame Per Second, FPS) 了, FPS 越大表示其渲染效率越高, 反之亦然, 故本文将仅使用 FLOD 进行单层裁剪、仅使用 FCULL 进行单层裁剪和本文提出的双层裁剪进行 FPS 的实验对比. 现设置三层 LOD 等级, Screen Size 的参数设置如表 3, 缓慢剔除算法的起始距离为 S 设置为 400 cm, 剔除终止距离 E 设置为 600 cm. 由于每个 LOD 等级的 FPS 不同, 故本文对三个等级的 FPS 做了均值处理. 实验结果如图 4 所示.

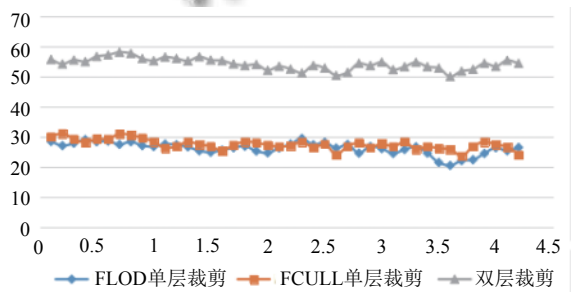


图 4 单层裁剪和双层裁剪 FPS 对比

由图 4 可以看出仅进行 FLOD 单层裁剪的 FPS 在 30 左右, 仅进行 FCULL 单层裁剪的 FPS 也在 30 左右, 但本文提出的双层裁剪 FPS 达到了 55 左右, 很大程度上提高了渲染效率.

5 总结

本文对虚幻引擎中单线程的游戏线程和渲染线程进行了同步并行化处理, 加快了一帧所需要的时间; 针对传统的层次细节算法提出了淡入淡出层次细节算法, 实验证明解决了层级切换所出现的“突越”现象; 针对传统的剔除算法提出了缓慢剔除算法, 实验证明解决了在超过剔除距离后出现的“突越”现象; 针对单层裁剪提出了双层裁剪, 首先使用淡入淡出层次细节算法进行第一层的裁剪, 裁剪掉在视野范围外的模型, 再使用缓慢剔除算法进行第二层裁剪, 裁剪掉被遮挡的模型部分, 从而很大程度上减少了所需渲染的三角面片数, 提高了渲染效率.

参考文献

1 符清芳, 张茹. 基于虚幻 4 的自然场景制作. 电脑知识与技术, 2016, 12(31): 188–189.
2 Hoppe H. Smooth view-dependent level-of-detail control and its application to terrain rendering. Proceedings Visualization ’98. Research Triangle Park, NC, USA, USA, 1998: 35–42.
3 刘晓, 刘镇, 梅向东. 基于实时 LOD 简化绘制的渲染优化方法. 湖南科技大学学报 (自然科学版), 2015, 30(4): 92–96.
4 刘宗香. 图形生成过程中遮挡问题的解决方法. 计算机时代, 1997, (10): 16.
5 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]
6 袁畅. GPU 加速的可见性剔除方法研究[硕士学位论文]. 长沙: 中南大学, 2013.
7 Hillesland K, Salomon B, Lastra A, *et al*. Fast and simple occlusion culling using hardware-based depth queries. Technical Report UNC-CH-TR02-039. Chapel Hill, NC, USA: University of North Carolina at Chapel Hill, 2002.
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]
9 额尔敦达来. 基于 GPU 的大规模地形场景绘制关键技术研究[硕士学位论文]. 长沙: 国防科学技术大学, 2005.
10 Zaugg B, Egber P. Voxel column culling: Occlusion culling for large terrain models. Proceedings of The 3rd Joint EUROGRAPHICS-IEEE TCVG Conference on Visualization. New York, NY, USA. 2001. 85–93.