

中国地质大学（北京）

本科毕业设计外文资料翻译

学 院： 信息工程学院

专 业： 计算机科学与技术

姓 名： 陈雨航

学 号： 1004171217

指导教师： 季晓慧

外文出处： ACM Transactions on Graphics
(TOG), 2014, 33 (4) : 1-8.

附 件： 1.外文资料翻译译文; 2.外文原文。

完成日期： 2021 年 1 月 11 日

紧凑的预置体素化阴影

Erik Sintorn Viktor K  mpe Ola Olsson Ulf Assarsson
查尔姆斯理工大学

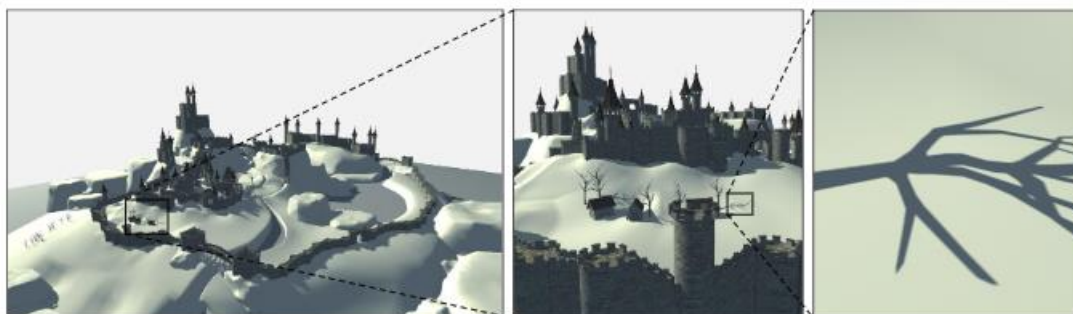


图 1: 一个使用我们的算法来评估不同尺度下观看场景时预计算出的太阳阴影的例子。我们紧凑的数据结构占用了 100MB 的图形内存, 相当于 256k*256k (即 2621442) 的阴影贴图。使用 9*9taps 的过滤器, 以 1080p 的分辨率评估阴影在少于 1ms 内完成。

摘要

对于游戏等实时应用程序来说, 在大场景中生成高质量的阴影是一个重要且具有挑战性的课题。我们提出了一种新的预计算阴影的数据结构, 使高质量的过滤阴影在场景中的任何点都能被重建。我们将高分辨率阴影转换为稀疏的体素八叉树, 并对每个节点对相应体素的光可见性进行编码, 且通过合并公共子树对树进行压缩。此方法得到的数据结构可以比相应的阴影贴图小多个数量级。我们也证明了它可以有效地实时评估大过滤内核。

CR 类别: I.3.7 【计算机图形学】: 三维图形与真实感——颜色、遮蔽、阴影和纹理;

关键词: 阴影、实时、预计算

1 简介

随着实时阴影算法的发展, 在大型开放场景中仍然很难生成高质量的阴影。当用户需要同时看到附近物体和远距离无别名阴影的情况下, 传统的阴影贴图算法【Williams 1978】就崩溃了。一个常见的补救方法是使用 Cascade Shadow Maps (CSMs), 该方法将视锥分区, 每个分区呈现一个阴影贴图【Engel 2006; Zhang et al. 2006; Lloyd et al. 2006】。这为所有视图样本提供了大致统一的阴影贴图分辨率。然而, 即使使用积极的剔除技术, 可能还是需要多次重新渲染场景的某些部分, 这将影响性能。此外, 远距离区域的分辨率, 即使与屏幕采样频率匹配良好, 也不足以捕捉复杂 shadow casters 而不产生锯齿。

实时应用程序中的大多数虚拟场景都包含很大一部分静态几何体, 并且通常也有一个或多个静态光源 (如太阳)。在这种情况下, 最好使用预计算的阴影, 这样既可以更快地进行评估, 又可以提供比完全动态技术更高质量的阴影。动态阴影技术, 如 CSMs, 可以用于动态几何体, 它只代表了三角形和填充率要求的一小部分。这通常会产生更高、更稳定的性能和更好的质量。

因此,这种情况在实践中非常普遍,并在某种形式上是得到了所有主要游戏引擎的支持。一种常见的方法是在纹理贴图(通常称为光线贴图)中存储预先计算的可见性信息,这些信息可以在阴影处理过程中立即查询。光线贴图可以几乎零成本的在阴影处提供光源可见性信息,但在某些方面非常有限。首先,它们只能用来评价静态几何表面的阴影,这是有问题的,因为动态几何体必须使用一些其他技术被静态环境遮蔽。其次,对于静态几何体,必须创建一个独特的参数化 UV,这将是困难和冗长的。第三,即使使用有损图像压缩技术,如果需要高分辨率,这些贴图仍然需要相当大的内存。

本文的目标是设计一个数据结构,提供静态几何体和静态光源的预计算阴影信息,他能保证在重建场景中的任意点重建高质量的过滤阴影。因此,静态几何体和动态几何体都可以接收来自静态环境的阴影,而单独的实时技术只需要支持来自动态几何体的阴影。我们通过将整个空间的阴影信息体素化生成一颗八叉树,然后通过合并公共子树压缩该树来实现这一目标。我们提供的数据结构非常紧凑,但仍然可以在着色时快速获得高质量的结果。我们的数据结构提供了相当于极高分辨率阴影图的信息,却只需要很小的内存成本。进一步来说,对于封闭几何体,或者当场景不使用动态对象时(例如建筑漫游),我们可以进一步压缩,使得场景和分辨率来说,需要比图 1 所示的 16 位阴影贴图少用了 3 个数量级的内存。

通过存储光的可见性,而不是深度值,可以非常有效地评估使用大过滤内核的高质量过滤,这使得阴影得到全高清分辨率的速度和质量远高于 CSM 方法。此外,我们还演示了通过两种实用方法来过滤远距离阴影,这两种方法都可以消除锯齿和提高性能。

2 过去的工作

实时阴影。有大量工作是关于实时阴影的生成的。关于详细调查,我们建议读者参考两本最近出版的著作: Eisemann 等【2011】与 Woo 和 Poulin【2012】。目前大多数需要从点光源投射阴影到大型虚拟世界的实时应用程序都使用了级联阴影贴图(CSM)技术的变体【Engel 2006; Lloyd 等 2006; Zhang 等 2006】。其思想是对视锥进行分区,并为每个分区分别渲染一个阴影贴图【Williams 1978】。这减少了因使用单个阴影贴图而引起的欠采样和过采样问题。然而为了完全隐藏由分区之间的锯齿和可见性过渡多引起的瑕疵,仍然需要大量的高分辨率阴影贴图。一个更普遍的实现的足够的阴影贴图分辨率的方法,是在第一次拍摄时估计阴影贴图分片所需的分辨率,然后以不同分辨率渲染不同分片的阴影贴图【Lefohn 等 2007; Giegl 和 Wimmer 2007】。而对于大多数实时应用来说,这仍然过于昂贵。迄今为止,这些所提及的技术都没有解决远距离 shadow casters 过于复杂而无法用低分辨率阴影贴图精确表示时出现的锯齿问题。

无论使用哪种离散阴影贴图方法,都必须在采样点使用某种形式的过滤进行重构。由于存储在阴影贴图深度值不能直接预过滤,这可能会变得非常昂贵。为了提供更大的过滤核,已经有几种替换方案,如方差阴影贴图(VSMs)【Donnelly 和 Lauritzen 2006】、卷积阴影贴图【Annen 等 2007】和指数阴影贴图(ESMs)【Annen 等 2008】。这些方法存储可见性函数的近似值,而不是单个深度,并且可以预先滤。由于它们的近似性,这些算法都存在失败案例。

还有一些基于阴影体积【Crow 1977; Heidmann 1991; Sintorn 等 2011】或基于不规则栅格化【Johnson 等 2005; Aila 和 Laine 2004; Sintorn 等 2008】的实时无别名算法,但这些方法目前还不够快,无法应用于实时的复杂场景中。

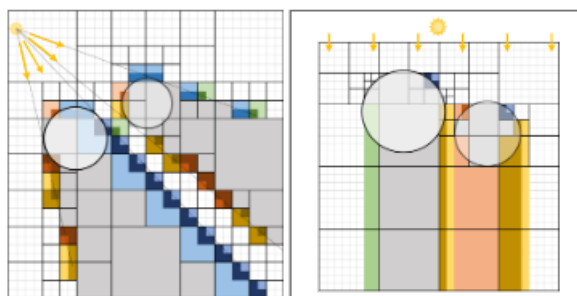


图 2：我们压缩方法的概述。左图：阴影空间的体素化，相同（彩色）节点被合并。

右图：光空间中的体素化提供了更好的压缩效果

预计算阴影。为了提高渲染的质量和性能，许多实时应用程序采用了某种形式的预计算辐射传输技术。我们建议读者参考 Ramamoorthi【2009】的调查，以了解这些方法的概述。预计算的点光源可见性可以存储在光线贴图中，用于快速、预过滤的查找。Rasmusson 等【2010】提出了一种专门针对光线贴图的硬件加速压缩方案，并提供了标准技术的概述。Lefebvre 和 Hoppe【2007】建议使用空间相干数据（如光线贴图）的层次表示来提高压缩率代价是查找变得更昂贵了。然而即使使用有损数据压缩方案，捕获高频率的可见性变化仍然需要过多内存以使光线贴图在大场景中可用。表面上的二进制可见度也可以通过距离场进行编码。正如 Green【2007】所建议的那样，这样的数据通常可以从低分辨率贴图中很好地重建，但是这种方法无法捕捉到瘦或复杂的阴影。此外，这些技术都需要对所有几何形状进行明确的 UV 参数化，并且只能在所指定的（静态）表面准确地存储可见性信息。

阴影贴图压缩。正如 Hasselgren 和 Akenine-Moller【2006】所讨论的，许多算法都试图实时压缩和解压深度缓冲区。这些方法降低了带宽需求，但是没有达到足够的压缩比来用于存储极高分辨率的阴影贴图。而 Arvo 和 Hirvikorpi【2005】的方法达到了较高的压缩比（例如，与简单场景的 16 位阴影贴图相比，压缩比为 50:1）。在他们的方法中，每条扫描线被压缩成一组线段，这些线段位于从光线中看到的前两个最接近的表面之间，它们可以分层存储，而着色器可以在查找过程中执行二进制搜索。然而他们的方法并不直接适用于动态阴影接收器，对于薄、非平坦的阴影发射器表现得很差。他们所建议的方法有助于提高 PCF 查找的性能和质量，因为他们可以在水平方向使用解析过滤，但是对于 $n \times n$ 过滤器，他们需要在贴图中执行 n 层搜索。

3 概述

我们的目标是设计一个数据结构，它能在合理的内存占用范围内表示与极高分辨率阴影贴图相同的信息。如果要对实时应用程序具有足够吸引力，那么在任何时候评估可见性都不应比使用阴影贴图更昂贵，这一点很重要。受到 Kampe 等【2013】工作中体素化几何体获得高压缩率的启发，我们采用了一种新颖的或许有点不直观的方法来压缩原始阴影贴图。

我们算法的基本思想是创建体素化的阴影空间并存储为八叉树，然后通过合并公共子树来压缩它，创建有向无环图（DAG）。如图 2，很明显，八叉树只需要沿着包围阴影空间的表面（我们称之为阴影边界）存储叶子节点，因此它的大小在很大程度上取决于该表面的面积。

为了提高公共子树合并的有效性，我们在投影光空间中进行体素化（参见图 2）。通过体素化该空间，除非阴影投射表面与该节点相交，一个坐标为 (x, y, z) 的 level 级节点将

与坐标为 $(x, y, z+1)$ 的节点相同。这个性质对未压缩的八叉树的大小没有影响，但是它意味着我们将突然在所有级别上拥有大量相同的节点，公共子树合并将会更有效。事实上，压缩的八叉树的大小在最坏的情况与（最近的）阴影贴图表面积成正比（见图 2）。此外，这些表面的公共子树将被合并，与存储八叉树本身和第 7 节所示的原始阴影贴图相比，这将导致极高的压缩率。

4 数据结构与评估

正如 Kampe 等【2013】论文所述，八叉树通过一个 DAG 表示，每个节点将由一个子掩码和一组指针组成。只需指向相同的节点，父节点就能简单的共享相同的子树。Kampe 等【2013】将表面体素化，节点与表面相交或者不相交。因此，子掩码每个子节点只需要一个位。然而，我们的数据结构表示一个体素化的体积，因此我们的子掩码必须能够识别一个节点：

- 1) 与阴影边界相交的节点；
- 2) 完全位于阴影空间之外的节点；
- 3) 完全位于阴影空间之内的节点。

因此，我们的子掩码存储为 16 位字，而不是 8 位（即每个孩子 2 位）。

DAG 的最后三层（代表 $4*4*4$ 体素的子体积）将不能从公共子树合并中受益。最小的子树可能由一个具有 16 位子掩码和 32 位指针的父节点以及一个具有 8 位子掩码的子节点组成。因此，合并三个最低级别并将我们的叶子存储为 $4*4*4$ 位网格（64 位叶子掩码）是有效的。

为了判断一个点 p 是否处于阴影中，我们获取根节点的子掩码并检查包含 p 的子节点在子掩码中的状态。如果这个节点被标记为完全在阴影区域之内或之外，我们就完成了。如果它与阴影边界相交，将获取相应的指针，然后对子节点重复该过程。如果遍历到一个叶子节点，在叶子掩码中执行直接查找。

因此，对于分辨率为 $2^n * 2^n * 2^n$ 的 DAG，我们需要对任何需要遍历到叶节点的样本执行 $2(n-2)+1$ 次获取操作。然而，许多样本可以提前终止遍历，对于那些剩下的样本，缓存命中率非常高。尽管如此，我们的数据结构中的查找肯定比贴图查找更昂贵。然而，我们将证明过滤查找可以非常有效，甚至超过传统的阴影贴图。

闭合对象优化。如果我们知道部分或全部阴影投射对象是封闭的，并且用户永远不会从这些对象的内部查看场景，我们可以优化我们的数据结构，将压缩率提高一个数量级。位于一个或多个这样的封闭空间中区域的对象被认为是未定义的（参见图 3a），完全位于未定义空间中的节点可以设置为可见或不可见，这取决于哪种选择提供较小的 DAG（参见图 3b）。这种优化的效果是相当显著的，因为它意味着大多数以前必须详细定义的阴影投射表面，现在只需要在其轮廓处有精细的细节（见图 3c）。在 Arvo 和 Hirvikorpi【2005】的文章中，两个最接近的阴影投射表面之间的区域被认为是不确定的，我们的方法与之相似，但在动态物体上能更好的压缩和进行阴影投射。

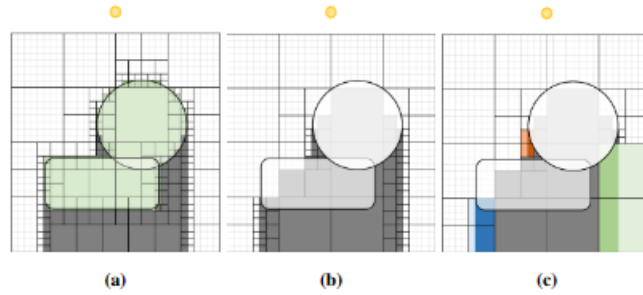


图 3: 对于封闭对象, a) 对象内的体素被认为是未定义的 (绿色), b) 树被简化, c) 公共子树被合并 (相同的节点以对应颜色表示)。

5 过滤

阴影贴图是一个离散的深度图像, 因此很容易因几种不同的方式引入锯齿和其他误差。在这一节中, 我们将使用 Eisemann 等【2011】提出的术语讨论我们的算法如何处理这些错误。

初始采样错误在创建阴影贴图时引入。当阴影贴图以低于最终屏幕采样频率的频率渲染时, 就会发生欠采样, 从而导致离散阴影贴图的可见放大。这种错误可以用 CSM【Engel 2006】技术缓解。当阴影映射以相对于几何体不足的分辨率渲染时, 引入初始样本锯齿。通过对阴影图进行超采样, 并对结果进行预过滤, ESM 等技术【Annenl 等 2008】可以减少这些问题。通过我们的解决方案, 初始采样错误可以避免, 因为我们压缩的数据结构可以容纳非常高的分辨率。如果分辨率足以满足近距离阴影的要求, 那么它也足以满足远距离物体的需求。

重采样错误发生在阴影贴图信号在查找过程中不正确重构时。如果一个视图样本被认为是一个小表面补丁, 可见性查找应该返回基于该补丁在阴影地图上的投影经过滤的可见性。由于阴影贴图不能预过滤, 通常的解决方案是使用 Percentage Closer Filtering (PCF)【Bunell 和 Pellacini 2004】来模糊阴影边缘, 这在许多样本中会变得非常昂贵。允许预过滤的替代方法 (如 VSMs【Donnelly 和 Lauritzen 2006】) 对于大的过滤器内核来说会更快。

对于我们的数据结构, PCF 将是一个非常吸引人的选择, 因为我们存储所有体素的二进制可见性, 每个体素最多只有一个比特。因此当我们获取包含我们体素可见性的叶子掩码时, 我们手边已经有了大量近距离可见性的样本。因为存储的可见性 (不是深度), 确定可见样本的百分比成为一个简单的位掩码操作 (见第 6.2 节)。

为了进一步利用这一点, 我们注意到, 如果所有 PCF 样本都在同一深度采集, 我们可以通过重新排列我们层次结构的最底层来确保在单个叶掩码中获取更多有效的可见性样本, 从而使叶掩码编码为 $8 \times 8 \times 1$ 而不是 $4 \times 4 \times 4$ 个体素。叶子节点上面的层次将编码 $1 \times 1 \times 8$ 个节点, 更高的层次将拥有与之前相同的节点布局。我们将在第 6.2 节中展示这种替换后的节点布局将保证的 PCF 查找变得十分高效。

为了消除前景中的阴影边界锯齿, 常见的做法 (通常已经足够了) 是对所有查找使用一个固定大小的过滤器内核。然而, 如果一个大场景使用单个的阴影贴图, 则远距离的视图样本实际上可能需要一个极大的过滤器内核来避免锯齿。我们建议采用两种不同的方法来处理这种情况。两种方法都将当前考虑的视图样本的像素转换为投影光空间, 以确定体素的近似面积。如果面积大于 PCF 过滤器的大小, 仅是使用该过滤器是不够的, 我们将计算树中的

哪个级别（L 级）分辨率更合适。

多分辨率反走样（MRAA）。第一种解决方案是简单地生成几个数据结构，每个数据结构对应两个分辨率的幂，并将它们并排存储。对于每个查找，我们决定最合适（基于 L）的分辨率，然后遍历相应的数据结构。这种方法允许对任意大小的过滤器内核进行非常高质量的反走样处理，但与仅存储完整分辨率的 DAG 相比，仍然需要两倍的内存空间。

体素占用反走样（VOAA）。如果无法接受额外的内存成本，另一种方法是为每个内部节点存储包含在阴影中的体素的比例（占用）。这可以在压缩之后完成，并且信息存储在包含子掩码的单词的未使用的上半部分中，因此这种方法不会产生额外的内存开销。当遍历数据结构时，我们现在可以在到达 L 级节点时停止遍历，并将该节点的占用率作为可见性值返回。请注意，这是一个近似值，并不等同于标准的过滤查找，因为在这种情况下，过滤器表示体积，而不是面积。为了避免动画中可见的细节级别转换，可以在层次 L 和层次 L-1 之间对占用值做线性插值。

6 实现

在本节中，我们将讨论我们所建议的数据结构的构建和渲染的实现细节。然后我们将讨论一些遍历算法的优化。

6.1 构建数据结构

我们从阴影贴图中生成数据结构，而这会使用简单的自顶向下算法，我们可以使用任何可以确定叶子体素可见性的方法。我们从一个与所需体素化的（x，y）分辨率匹配的阴影贴图开始。在我们的实现中，我们的数据结构的 z（深度）分辨率被设置为等于 x 和 y，但是布局可以任意选择。然后我们每一步将分辨率减半，为这个阴影贴图建立一个最小最大层次结构。为了构建数据结构，我们从根节点（包含整个光锥体）开始，并根据最小最大层次结构的第二级（分辨率 2*2）测试它的所有子节点。如果轴向包围框（AABB）完全位于阴影贴图的最大深度之外，整个节点处于阴影中，根节点的子掩码被更新以反映这一点。相反，如果子节点的整个 AABB 比层次结构中的最小深度更靠近光线，则节点被标记为可见。如果这两个条件都不成立，则将该节点标记为相交，并对该子节点递归重复该过程，直到达到数据结构的最高级。在最后一级，根据原始阴影贴图测试体素，并在叶掩码中记录可见性。我们现在有了一个稀疏的体素八叉树（SVO），我们继续按照 Kampe 等人【2013】的描述将其简化为最优 DAG。在高分辨率下，我们生成并压缩分片中的阴影贴图，从每个分片中生成一组子 DAG，然后将它们组合并压缩成最终的、最优的 DAG。

闭合对象优化。使用稍微修改过的算法将封闭对象插入到树中。正如第 4 节所解释的那样，任何完全位于一个或多个封闭对象内部的节点都可以被认为是未定义的。任何完全位于所有封闭对象之外的节点必须具有正确的可见性。在光线照射下，任何与多边形的交叉点都意味着进入或退出一个封闭的物体。在我们的实现中，我们将第一个交叉点和光线再次进入空间的交叉点之间的所有节点定义为未定义节点。同时识别后续的未定义区域（因为这些区域被阴影空间包围）几乎没有什么好处，这会使算法更加复杂。

为了实现这一点，我们生成（除了阴影贴图之外）一个包含第一个未定义区域结束深度的贴图。为了处理交叉的封闭对象，我们通过深度剥离【Everitt 2001】来提取场景的层次，并保持一个逐像素计数器（正面增加，背面减少），以找到计数器返回零的第一个深度。

该贴图还建立了最小最大层次结构，并且改变了初始 SVO 的生成，使得一个既不与映射相交但位于未定义区域内的节点被标记为未定义节点。一个与两个映射中的任意一个相交的节点将被进一步遍历。每当一个节点的所有子节点都被评估，父节点将决定其未定义子节点的最终状态。如果一个节点只包含阴影和未定义的子节点，整个节点被设置为阴影。如果一个节点只包含可见的和未定义的子节点，它被设置为可见。如果节点包含所有三种类型的子节点（阴影、可见和未定义），则将未定义的节点设置为阴影，并将节点标记为相交。最后一条规则是必须的，因为最终的数据结构不能包含未定义的节点。选择未定义的节点，使整个节点与已经存在的节点匹配，可能会有潜在的好处，但是我们还没有找到执行这种搜索的有效方法。在这一步之后，我们得到了一个 SVO，它被压缩得和之前完全一致。

如果数据结构不打算与动态几何体结合使用，同样的优化可以用于非封闭对象。Arvo 和 Hirvikorpi【2005】指出，在这种情况下，第一个和第个二阴影投射层之间的空间可以被认为是未定义的（因为它不会被查询）。

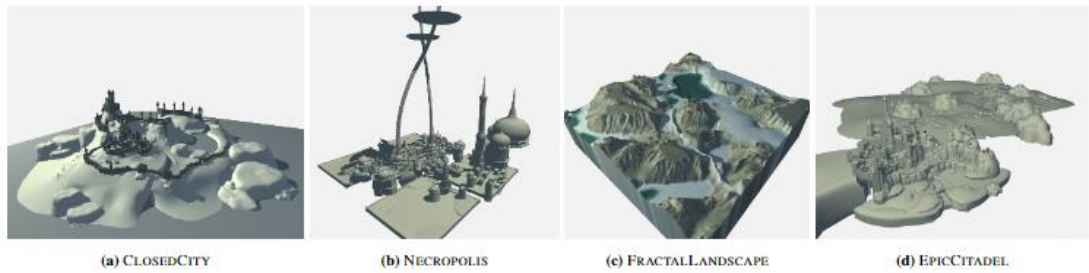


图 4：用于测试数据结构的压缩和性能的场景

6.2 遍历数据结构

我们已经在延迟着色上下文中评估了我们的数据结构。第一次通过的相机，位置，法线和材料属性存储在一个 G 缓冲区。我们的阴影评估代码是在 CUDA 中作为一个单独的通道实现的，每个视图样本都会启动一个线程。内核评估视图样本的光可见性，并最终将结果存储在一个用于着色的辅助缓冲区中。请注意，遍历代码中都能在片段着色器中实现，因此我们的数据结构也可以在前向着色上下文中工作的很好。

为了评估视图样本的可见性，我们首先计算它所在的体素的离散坐标 $(0, 0, 0) \leq P < (S, S, S)$ （其中 S 是数据结构的解决方案）。我们通过简单地缩放样本的标准化坐标来获得 P 。因此， P 的分量将是 $L-1$ 位宽的整数，其中 L 是数据结构中的层数。为了找出在某个级别上要遍历到哪个子节点，我们提取每个组件的第 $(L-2-i)$ 位，并将它们连接到标识子节点的 3 位子索引（参见图 5）。

接下来，我们读取根节点的子掩码（在 level $i=0$ ），并查看与此子索引对应的位。如果节点被标记为完全阴影或可见，我们分别返回可见性 0 或 1。如果节点被标记为相交，我们继续递归，直到我们达到 $L-3$ 级，即叶子节点上面的级别。在这里，我们使用 z 的最后三位作为子索引（由于上面解释的扁平叶节点布局）。在叶子节点级别， x 和 y 的最后三位被连接起来，形成叶掩码正确位的索引。

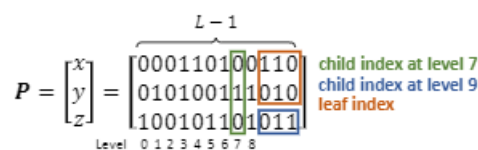


图 5：离散位置定义了遍历顺序。每个位为相应的轴向和层级定义一个半空间。将每

个组件的位 k 组装起来形成的整数是要在相应层级遍历的子索引。最后三个级别的编码不同，以便快速查找 PCF

过滤查找。如第 5 节所述，最后三个级别被存储为 $8*8$ 位包含相同深度的体素的可见性的网格。因此，当过滤器大小为 $9*9$ 或更小时，我们最多需要访问 4 个这样的叶节点，并且我们事先知道这些节点是什么。为了得到过滤后的阴影值，我们需要对于每个节点计算包含的体素在过滤器内部和在阴影中的比例。对于每个叶子节点，我们生成一个位掩码来清除过滤器外的所有位（参见图 6）。在使用这个位掩码对子掩码进行“与操作”之后，我们计算剩余的偏移位，所有四个节点的偏移位之和除以过滤器中的体素总数（本例中为 $9*9$ ），以获得过滤后的可见性。

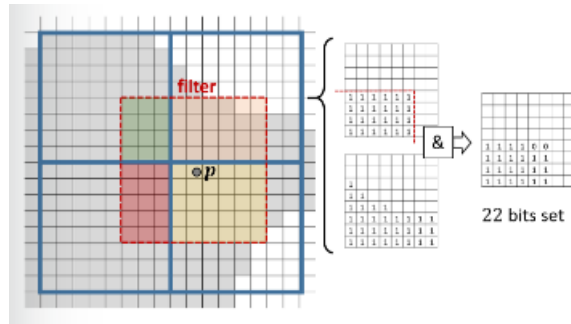


图 6：为了获得一个过滤后的阴影值，我们为每个节点计算一个位掩码，它对应于过滤器覆盖的体素。这个掩码是和叶掩码一起使用。然后我们计算结果中的偏移位数。

遍历顺序。通过过滤查找，我们将访问几个叶节点，并且在开始遍历之前我们知道每个叶节点的路径。通常情况下，这些路径的起点是相同的，我们可以很容易地确定它们在哪个层次上出现分叉。因此，我们可以通过比较两个连续叶节点的路径和只回溯到路径分叉的层级来显著减少遍历所花费的时间。

如果我们这样做，访问叶节点的顺序也很重要。我们可以通过选择总是继续到需要回溯最小级别数的叶节点来减少遍历操作中的迭代次数。如图 7 所示，我们可以看到从一个叶节点移动到另一个叶节点需要回溯到两个不同级别中的一个，L1 或 L2（使用 $9*9$ 或更小的过滤器）。无论我们选择哪个顺序，我们都会回溯到其中一个层次一次，再回溯到另一个层次两次。因此，在一个固定的起点上，我们将简单地选择两个预定的遍历顺序中的一个，这样我们只需要回溯到更高的层次一次。选择哪个遍历顺序只取决于 L1 和 L2 中哪个更高。

对于较大的过滤器内核，可能的顺序有更多的排列。我们优化的遍历实现目前最大支持 $17*17$ 大小的过滤器，最多需要访问 9 个叶子节点。在这些节点中，有些节点有相同的父节点，并且应该被连续访问（顺序不重要）。图 7c 显示了我们遇到这些节点的四种不同方式和八种不同的遍历顺序，其中一种是最优的。对于这种大小的过滤器，我们使用一个预先计算好的查找表来选择正确的遍历顺序。我们用于遍历数据结构的优化 CUDA 内核可用作补充材料。

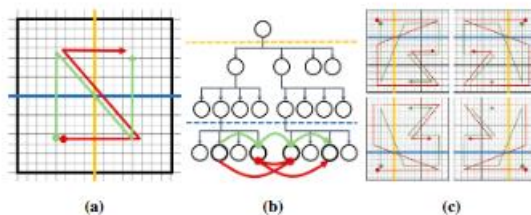


图 7：a) $9*9$ 过滤器的两个可能遍历顺序，其中一种是最优的。b) 它们如何访问层次

结构的示例。在这种情况下，绿色遍历顺序更有效。c) 我们为 17*17 过滤器选择的八个遍历顺序

优化。我们不需要一个完整的堆栈来执行这个回溯。因为我们知道我们只需要返回两个可能级别中的一个，所以这些级别是存储状态的唯一级别。遍历代码的最终优化是简单地将 DAG 的顶部存储在网格结构中，这样我们就可以立即获取需要遍历的子 DAG 的偏移量。在我们的实现中，我们始终使用大小为 128*128*128 的网格，有效地将 DAG 的前 8 个级别替换为 8mb 的网格。

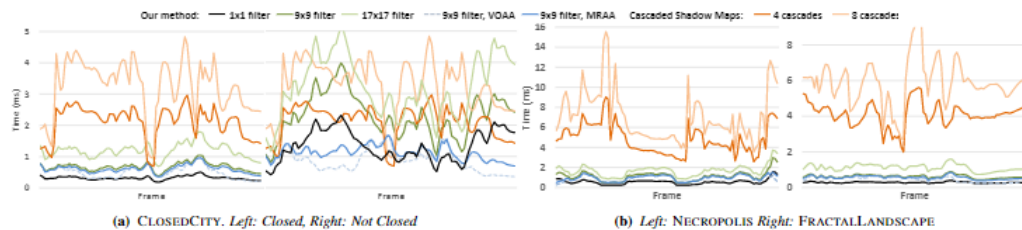


图 8：我们的方法在不同大小的过滤器和不同的抗锯齿技术下以及在不同层数的 CSM 的性能。CSM 的查找开销在 0.2ms 到 0.4ms 之间，视口剔除开销在 0.1ms 到 0.35ms 之间

7 结果

在本节中，我们将报告使用我们的数据结构来压缩阴影贴图的结果，并使用它们来实时评估阴影。所有的性能结果都是在配备了 Intel Core i7-3930K CPU 和 Nvidia TITAN GPU 的 PC 上测试的。所有渲染图像的原始大小是 1920*1080。

场景。在本节中，我们使用了四个不同的场景进行比较（见图 4）。墓地（2.1M 的三角面）和 EPIC 城堡（373K 的三角面）场景来自 Epic UDK。封闭城市场景（613K 三角面）是一个由我们制作的模拟游戏的场景，所有的物体都是封闭的。分形景观场景（2.1M 三角面）是一个随机生成的分形景观，完全由封闭几何构成。我们已经为其中的三个场景创建了飞越路径，用于测量这些场景不同视角下的性能。对于分形景观场景，我们添加了一个动态对象，它接收来自静态几何体的阴影。场景和飞行路径在附带的视频中展示。

压缩。各种场景的压缩数据的最终大小如图 9 所示。封闭城市和分形景观场景都只包含封闭的几何图形，我们用封闭的几何图形优化和不用封闭的几何图形优化分别构建了它们。与 256k*256k 的 16 位阴影贴图相比，使用这种优化方法，两个场景的压缩因子都超过了三个数量级。如果没有优化，压缩比仍然是非常好的。

墓地和 EPIC 城堡的场景不是封闭的，所以我们不能使用我们的封闭对象优化来进行这些构建。尽管如此，两个场景压缩到相同分辨率（128k*128k）的 16 位阴影图的 2% 左右，在合理的内存预算下做出如此重大的优化成为可能。墓地如 6.1 节所述，还利用了静态几何体的优化，然后压缩到约 0.5%。

构建时间。目前，建立数据结构是一个相当耗时的过程。我们的实现 CPU 上允许，只执行非常简单的工作负载并行化，所以运行时间肯定会得到改善。从下表可以看出，构建数据结构所花费的时间与构建数据结构的阴影贴图的分辨率大致成比例。

FractalLandscape, closed.							
	1K ³	4K ³	16K ³	32K ³	64K ³	128K ³	256K ³
	0.5s	2s	18s	1m8s	4m16s	17m35s	1h32m

Resolution:	1024 ³	4096 ³	16K ³	32K ³	128K ³	256K ³
16-bit SM	2 MB	34 MB	537 MB	2 GB	34 GB	137 GB
Closed	202 KB	1 MB	6 MB	13 MB	50 MB	100 MB
Compression	9.61%	3.53%	1.16%	0.60%	0.14%	0.07%
Not Closed	489 KB	4 MB	36 MB	96 MB	639 MB	1638 MB
Compression	23.32%	12.41%	6.69%	4.48%	1.86%	1.19%
FractalL	2 MB	34 MB	537 MB	2 GB	34 GB	137 GB
Closed	164 KB	759 KB	3 MB	7 MB	29 MB	61 MB
Compression	7.81%	2.26%	0.62%	0.33%	0.09%	0.04%
Not Closed	490 KB	5 MB	50 MB	146 MB	1155 MB	-
Compression	23.37%	14.34%	9.29%	6.81%	3.36%	-
Neopolls	2 MB	34 MB	537 MB	2 GB	34 GB	-
Not closed	539 KB	5 MB	37 MB	100 MB	657 MB	-
Compression	25.72%	13.66%	6.94%	4.66%	1.91%	-
Static geometry	444 KB	3 MB	14 MB	31 MB	144 MB	-
Compression	21.19%	8.43%	2.69%	1.46%	0.42%	-
Citadel	2 MB	34 MB	537 MB	2 GB	34 GB	-
Not closed	585 KB	6 MB	44 MB	115 MB	727 MB	-
Compression	27.90%	16.76%	8.19%	5.37%	2.11%	-

图 9：压缩的结果

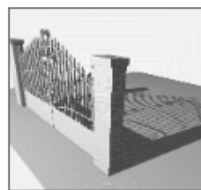
级联阴影贴图。为了比较，我们基于 Zhang 等人【2005】的描述实现了 CSM 算法。视锥分割距离是线性和对数的混合，并针对每个场景进行了调整。为场景创建了一个边界体积层次结构，并为每个分剔除几何图形。剔除使用 CUDA 在 GPU 上实现，并直接在 OpenGL 缓冲区生成绘图命令。为了将批绘制到级联中，我们使用分层渲染和单个绘制调用（glMultiDrawElementsIndirect），利用几何体着色器将每个批路由到适当的级联层。为了提高性能，我们通过在剔除 shadow casters 之前执行额外的剔除过程检测空级联，即那些不包含任何阴影接收器的级联。只有硬件支持的 2*2taps PCF 被用来过滤阴影。每个级联层的分辨率为 2048*2048，深度精度为 16 位。如图 8 所示，执行剔除和着色所花费的时间很少，大部分使用 CSMs 的时间都花在了三角形的光栅化上。这表明，将 CSMs 用于动态几何，同时使用静态几何的解决方案，可以通过减少光栅化的三角形数量来提供高性能的解决方案。

评估性能。图 8a 和图 8b 给出了评估飞行中每帧的阴影所花费的时间，以及我们使用四级和八级级联实现 CSMs 所花费的时间。CSM 测量的时间是渲染阴影贴图所花费的时间和执行查找所花费的时间的总和，因为每个阴影贴图必须新的视角下重新渲染。

在大多数场景中，我们的算法所有变体的表现都远远超过 CSMs，同时也提供了更高的视觉质量。封闭城市的封闭版本符合这一点，但在非封闭版本中性能更接近。原因在于视点离场景相当远，这意味着叶子附近的内存访问模式是不连贯的。第 5 节中提出的反走样方案通过在树的更高处终止来改善这种情况。这种行为在使用闭合几何优化构建的场景中几乎不存在，因为无阴影表面的遍历通常可以在树中很高的位置结束，这表明这不仅是一种存储优化，而且大大提高了运行时的性能。

结果表明，我们的算法对大尺寸的过滤器处理效果很好，在最坏的情况下，从单个样本到 9*9taps 的着色时间增加了 2.5 倍，17*17taps 的开销是单个 tap 的 4.5 倍。

我们通过比较固定顺序的遍历所花费的时间（仍然只是回溯到叶子节点之间最近的公共层次）来衡量使用遍历顺序优化所获得的性能改进。在封闭城市场景（使用封闭对象优化构建）中，对于 9*9 过滤器和 17*17 过滤器，最坏情况下的运行时间分别减少了 7.5%和 10.8%。在墓地中，相应的数字是 14.2%和 26.6%。



Filter Size 9x9	Blur	Evaluate	Sum
Shadow Maps	-	0.52ms	0.52ms
VSM	0.62ms	0.19ms	0.81ms
Ours	-	0.43ms	0.43ms
Filter Size 17x17	Blur	Evaluate	Sum
Shadow Maps	-	1.9ms	1.9ms
VSM	0.82ms	0.19ms	1.01ms
Ours	-	0.76ms	0.76ms

图 10：使用大过滤器的查找性能

过滤性能。我们针对大过滤器阴影查找所花费的时间对我们的算法、标准阴影贴图和 VSMs 进行了比较。阴影数据结构的分辨率在所有情况下都是 2048*2048。结果与场景和视图一起显示在图 10 中。

对于阴影贴图算法，我们使用硬件加速的 PCF 过滤器在每个纹理查找中取 2 个样本。我们使用了 16 位阴影贴图，所以内存消耗为 8MB。对于 VSMs，我们渲染了一个标准的阴影贴图，然后用一个可分离的盒子过滤器进行两次模糊处理。VSM 存储在一个双通道的 32 位缓冲区中，因此内存成本为 32MB，这使得我们的方法即使对于小的静态映射（981Kb，对于这个场景，当不使用任何节省空间的优化时）也很有趣。虽然在 VSM 中查找速度非常快，但如果在 CSM 设置中使用，贴图也必须对每帧进行模糊处理，而且模糊阶段的成本相当高，且具有较高分辨率的贴图缩放效果很差。

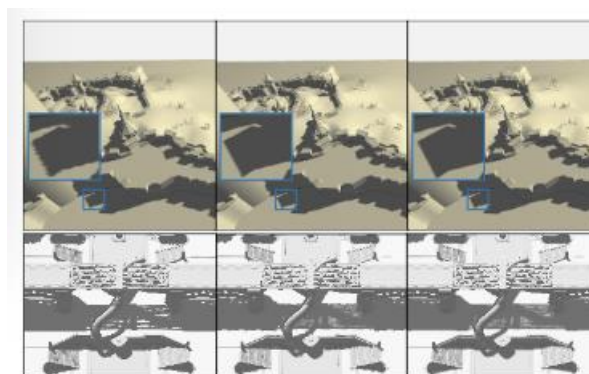


图 11：顶部：过滤质量。从左到右依次是 CSM（4 层，2*2，1.8ms）、CSM（8 层，2*2，2.4ms）和我们的（9*9 过滤器，0.47ms）。底部：远距离阴影抗锯齿。看一个小区域的墓地图像（另见补充视频）。从左到右为：9*9 过滤器，VOAA，MRAA

图像质量。图 11 的最上面一行显示了我们的基本 9*9 过滤器与硬件支持的快速 2*2 阴影贴图过滤器的对比。使用我们的方法可以获得更高的阴影分辨率和更大的过滤尺寸，从而得到更高质量的阴影边界。下面一行显示了墓地场景中一个复杂结构的 shadow cast。相机离得很远，图中只显示了图像的一小部分区域。在这里，9*9 过滤器显然是不够的。VOAA 方法可以很好的消除投射阴影的锯齿，但是在动画中过渡非常明显。MRAA 方法工作得非常好，几乎没有明显的变化。

8 局限性和未来的工作

如果 z(深度)的分辨率是足够的，我们的技术存储可见性信息的质量等同于阴影贴图。因此，可以使用与阴影贴图类似的偏置技术来避免硬阴影的自阴影伪影效应。然而，为了使

用更大尺寸的过滤器进行有效的查找，我们的方法根据相同的深度对所有样本进行测试，这比使用单独的阴影贴图 `taps` 更具限制性。我们的解决方案是始终将阴影查找定位在表面法线方向的过滤器大小的一半，这可能导致非常大的过滤器阴影的可见偏移。更复杂的偏置方法，以及开发一种阻断搜索方法来估计半影宽度，都是未来研究的有趣领域。

由于我们的数据结构在光空间的分辨率是一致的，透视光在世界空间的分辨率是非一致的。因此，靠近光源的可见度可能会以不必要的高分辨率存储。未来的改进是允许在距离光线越远的地方增加 (x, y) 分辨率。

我们的反锯齿方法需要一个很大的偏置（在世界空间中），对于非常远的阴影，可能会导致非常薄的 `shadow casters` 的漏光，虽然我们没有观察到任何这一现象。这个问题与 `CSMs` 发生的情况非常相似。使用 `VOAA` 方法，当我们远离阴影时，也可以看到从一个细节层次到另一个细节层次的转变，这可以通过在各层次之间进行线性插值来避免。

我们希望探索将我们的数据结构用于体积阴影单散射的光线追踪和透明阴影投射器的阴影压缩。我们也希望能够优化我们的构造方法。

鸣谢

EPIC 城堡和墓地场景都是通过 Epic Games 的非真实开发工具包分发的。分形地形场景是使用 `World Machine` 软件生成的。这项研究得到了瑞典战略研究基金会 RIT10-0033 的支持。用于这项研究的 TITAN GPU 是由 NVIDIA 公司捐赠的。

参考文献

- 【1】 AILA, T., AND LAINE, S. 2004. Alias-free shadow maps. In *Proc. EG Symposium on Rendering 2004, EGSR'04*, 161–166.
- 【2】 ANNEN, T., MERTENS, T., BEKAERT, P., SEIDEL, H.-P., AND KAUTZ, J. 2007. Convolution shadow maps. In *Proc. EG Symposium on Rendering 2007, EGSR'07*, 51–60.
- 【3】 ANNEN, T., MERTENS, T., SEIDEL, H.-P., FLERACKERS, E., AND KAUTZ, J. 2008. Exponential shadow maps. In *Proc. of Graph. Interface 2008, Canadian Information Proc. Soc., GI'08*, 155–161.
- 【4】 ARVO, J., AND HIRVIKORPI, M. 2005. Compressed shadow maps. *Vis. Comput.* 21, 3 (Apr.), 125–138.
- 【5】 BUNNEL, M., AND PELLACINI, F. 2004. Shadow map antialiasing. In *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*, R. Fernando, Ed. Pearson Higher Education.
- 【6】 CROW, F. C. 1977. Shadow algorithms for computer graphics. *SIGGRAPH Comput. Graph.* 11(July), 242–248.
- 【7】 DONNELLY, W., AND LAURITZEN, A. 2006. Variance shadowmaps. In *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games, ACM, I3D '06*, 161–165.
- 【8】 EISEMANN, E., SCHWARZ, M., ASSARSSON, U., AND WIMMER, M. 2011. *Real-Time Shadows*. A.K. Peters.

- 【9】 ENGEL, W. 2006. Cascaded shadow maps. In *ShaderX5: Advanced Rendering Techniques*, T. Forsyth, Ed., Shaderx series. Charles River Media, Inc.
- 【10】 EVERITT, C., 2001. Interactive order-independent transparency. Published online at http://www.nvidia.com/object/Interactive_Order_Transparency.html.
- 【11】 GIEGL, M., AND WIMMER, M. 2007. Fitted virtual shadow maps. In *Proceedings of Graphics Interface 2007*, ACM, GI '07, 159–168.
- 【12】 GREEN, C. 2007. Improved alpha-tested magnification for vector textures and special effects. In *ACM SIGGRAPH 2007 Courses*, ACM, SIGGRAPH '07, 9–18.
- 【13】 HASSELGREN, J., AND AKENINE-MÖLLER, T. 2006. Efficient depth buffer compression. In *Proc. 21st ACM SIGGRAPH/EGSymp. on Graphics Hardware*, ACM, GH '06, 103–110.
- 【14】 HEIDMANN, T. 1991. Real shadows, real time. *Iris Universe* 18, 28–31. Silicon Graphics, Inc.
- 【15】 JOHNSON, G. S., LEE, J., BURNS, C. A., AND MARK, W. R. 2005. The irregular z-buffer: Hardware acceleration for irregular data structures. *ACM Trans. Graph.* 24, 4 (Oct.), 1462–1482.
- 【16】 KÄMPE, V., SINTORN, E., AND ASSARSSON, U. 2013. High resolution sparse voxel dags. *ACM Trans. Graph.* 32, 4 (July), 101:1–101:13.
- 【17】 LEFEBVRE, S., AND HOPPE, H. 2007. Compressed random-access trees for spatially coherent data. In *Proceedings of the 18th Eurographics Conference on Rendering Techniques*, Eurographics Association, EGSR'07, 339–349.
- 【18】 LEFÖHN, A. E., SENGUPTA, S., AND OWENS, J. D. 2007. Resolution-matched shadow maps. *ACM Trans. Graph.* 26, 4 (Oct.).
- 【19】 LLOYD, D. B., TUFT, D., YOON, S.-E., AND MANOCHA, D. 2006. Warping and partitioning for low error shadow maps. In *Proceedings of the 17th Eurographics Conference on Rendering Techniques*, Eurographics Association, EGSR'06, 215–226.
- 【20】 RAMAMOORTHY, R. 2009. Precomputation-based rendering. *Found. Trends. Comput. Graph. Vis.* 3, 4 (Apr.), 281–369.
- 【21】 RASMUSSEN, J., STRÖM, J., WENNERSTEN, P., DOGGETT, M., AND AKENINE-MÖLLER, T. 2010. Texture compression of light maps using smooth profile functions. In *Proceedings of the Conference on High Performance Graphics*, HPG '10, 143–152.
- 【22】 SINTORN, E., EISEMANN, E., AND ASSARSSON, U. 2008. Sample-based visibility for soft shadows using alias-free shadow maps. In *Proc. of 19th EG Conf. on Rendering*, Eurographics Association, EGSR'08, 1285–1292.
- 【23】 SINTORN, E., OLSSON, O., AND ASSARSSON, U. 2011. An efficient alias-free shadow algorithm for opaque and transparent objects using per-triangle shadow volumes. *ACM Trans. Graph.* 30, 6 (Dec.), 153:1–153:10.
- 【24】 WILLIAMS, L. 1978. Casting curved shadows on curved surfaces. *SIGGRAPH Comput. Graph.* 12 (August), 270–274.
- 【25】 WOO, A., AND POULIN, P. 2012. *Shadow Algorithms Data Miner*. Taylor & Francis.
- 【26】 ZHANG, F., SUN, H., AND NYMAN, O. 2005. Parallel-split shadow maps on programmable GPUs. In *GPU Gems 3*, Addison-Wesley, H. Nguyen, Ed.
- 【27】 ZHANG, F., SUN, H., XU, L., AND LUN, L. K. 2006. Parallel-split shadow maps for large-scale virtual environments. In *Proc. Virtual Reality Continuum and Its Applications*,

ACM, VRCIA'06, 311–318.150:8