

图1：与先前的工作相比，我们算法表现出最佳剔除性能和带宽。 与更复杂的反馈算法的理想版本相似，同时保持前向剔除方法的简单性。 通过压缩深度，与竞争算法相比，我们需要的带宽要少得多。

分层深度剔除是一项重要的优化，它存在于所有现代高性能图形处理器中。 我们提出了一种基于分层深度表示的新颖剔除算法，其中每个样本的掩码指示每个样本所属的层。 与以前的工作相比，我们的算法本质上是前馈，后者依赖于延迟的反馈回路。 与竞争算法相比，它易于实现且约束更少，这使得对硬件体系结构进行负载平衡变得更加容易。 与以前的工作相比，我们的算法性能非常好，通常可以达到最佳剔除效率的90％以上。此外，我们可以通过压缩分层深度缓冲区将带宽减少多达16％。

1 介绍

2014年，笔记本电脑和台式机细分市场销售了超过4亿个图形处理器。在每个GPU中，都有一个高度优化的固定功能分层深度剔除单元，该模块逐块使用遮挡剔除技术[Greene等。 1993年； Morein 2000]。 为了使到深度缓冲器的存储器通信量最小化，已经花费了大量的工程努力来微调这些单元，这反过来改善了性能和/或降低了功率。遮挡剔除透明地集成在GPU中，即大多数用户在不知道遮挡剔除的情况下会享受到它的好处。每年出货的单位数量众多，并且分层深度剔除带来的性能/功耗优势使其非常重要，这不仅对于提高效率，而且对于不同的用例而言，使实现变得更简单，更可靠。

我们提出了一种新颖的剔除算法，该算法使用分层深度表示以及将每个样本关联到一个图层的选择蒙版。在我们的算法中，剔除和更新表示非常便宜且简单，并且与以前的方法不同，我们无需花费昂贵的反馈回路即可计算出精确的深度范围[Hasselgren和Akenine-Moller 2006]。此外，由于我们不需要扫描后端中所有样本的深度值，因此我们在选择图块大小方面拥有更大的自由度。反过来，这使得更容易平衡图形管道。请参见图1，以了解我们算法的潜在剔除示例。

2 之前成果

格林等人[1993]提出了一个基于完整深度金字塔的剔除系统，每个级别都有保守的zmax值。但是，尽管影响深远，但始终保持更新整个深度金字塔是不切实际的。 Morein [2000]提出了一种更实用的方法，其中最大深度zmax被存储并针对每个图块进行计算。如果保守估计瓷砖内部三角形的最小深度大于瓷砖的zmax，则可以剔除与瓷砖重叠的三角形部分。此外，还可以存储最小图块深度zmin，该深度用于避免深度读取。如果保守估计三角形的最大深度小于zmin [Akenine-Moller and Strom 2003]，则三角形可以轻易覆盖图块（假设没有进行alpha /模板测试等），并且可以跳过读取操作。摘自文献[Hasselgren and Akenine-Moller 2006; [在2000年更上一层楼]，我们推论到zmax通常是根据图块中每个样本的深度计算得出的，必须使用反馈回路将其传递给分层深度测试。理想情况下，每次重写具有最大深度值的样本时，都应重新计算和更新图块的zmax值，但是通常为了减少计算而更新的频率较低。例如，当从深度缓存中逐出瓦片时，可以重新计算zmax。

遮挡查询会计算通过深度测试的碎片数量，并可用于剔除整个对象[Bittner等。 2004; Guthe等。 2006年； Mattausch等。 2008年； Staneker等。 [2003年]使用简单的代理几何图形，例如边界框。 Aila和Miettinen [2004]提出了一种用于动态遮挡剔除的系统，在游戏行业中，已证明将遮挡查询基于软件光栅化是有用的，以更好地平衡CPU和GPU的负载[Collin 2011]。 张等。 [1997]提出使用分层遮挡图进行遮挡查询。 它们不是将查询基于深度缓冲区，而是使用全分辨率，分层覆盖图，并将深度分别存储在低分辨率深度估计缓冲区中。

我们的算法使用了受Jouppi和Chang [1999]启发的封堵器合并。他们通过存储低精度的深度平面方程，提出了一种低成本的抗锯齿和透明度算法。每个像素存储固定数量的平面，并使用合并试探法处理溢出。同样，Beaudoin和Poulin [2004]扩展了MSAA [Akeley 1993]，以使用分层索引结构来引用每个图块的一小组颜色和深度值。为了处理层溢出，他们选择降低采样率，而不是使用有损合并启发式方法。

Greene和Kass扩展了他们的早期工作[Greene等。 （1993年）包括使用着色算法的间隔算法和具有可见性的四叉树细分的带错误边界的抗锯齿功能（1994年）。 此外，格林等。 [1996]使用BSP树使用覆盖蒙版的金字塔分层遍历场景和屏幕空间，以实现有效的抗锯齿。 为了节省小三角形的像素着色工作，Fatahalian等人。 [2010]使用聚合覆盖蒙版从相邻三角形收集并合并四片段。 他们的蒙版的目的与我们的蒙版的目的不同，他们使用它来避免将阴影合并到几何不连续点上，而我们的蒙版表示是深度缓冲区的有损但保守的近似值。

3 图形架构概览

为了使我们的算法更加具体化，我们首先描述了GPU深度管道的典型实现，如先前的工作所介绍的那样[Hasselgren and Akenine-Moller 2006]。正如大多数现代图形API所指定的那样，图2的第一行从功能的角度描述了渲染管线。随着一系列三角形穿过渲染管线，深度和颜色缓冲区将逐渐更新。对于每个样本，距离最近的三角形的深度及其颜色将被存储。 由于性能的原因，实际的硬件管道通常与API规范不同，可以在图2的中间行看到一个常见的实现。在下面，为了简化说明，我们将讨论限制在小于深度的函数范围内，但是 这些技术可以推广到流行API（例如DirectX，OpenGL）以及即将到来的Vulkan API中使用的所有类型的深度函数。

栅格化 我们跳过了图形处理器的几何处理部分，并在光栅化器单元开始讨论。光栅化器负责确定哪些样本与特定三角形重叠。作为一种优化，现代光栅化器通常在图块上工作，图块是个样本的组，其中是每个像素的样本数。对每个图块执行保守测试，以确定它是否被完全覆盖，完全位于三角形之外或部分重叠。仅对于部分与三角形重叠的图块，才需要进行按样本覆盖率测试。计算样本覆盖率后，将使用片段着色器对每个片段进行着色，然后进行深度测试，以确定可见性。从图2的中间行可以看出，常见的优化方法是将每个样本的覆盖率测试放在z或HiZ层次单元之后。理由是，HiZ可能会在每次样本覆盖率测试发生之前移除或剔除图块，从而提高了该单元的性能。

HiZ单元 深度测试可能会消耗大量的内存带宽和计算能力[Aila等。 2003]。 因此，硬件流水线通常具有HiZ单元，目的是在可以针对整个样本组明确确定深度测试的结果时，使用粗略的深度测试来快速丢弃（剔除）或接受图块。为此，HiZ单元维护深度缓冲区的保守版本,称为粗略深度缓冲区，其中每个图块包含深度边界.

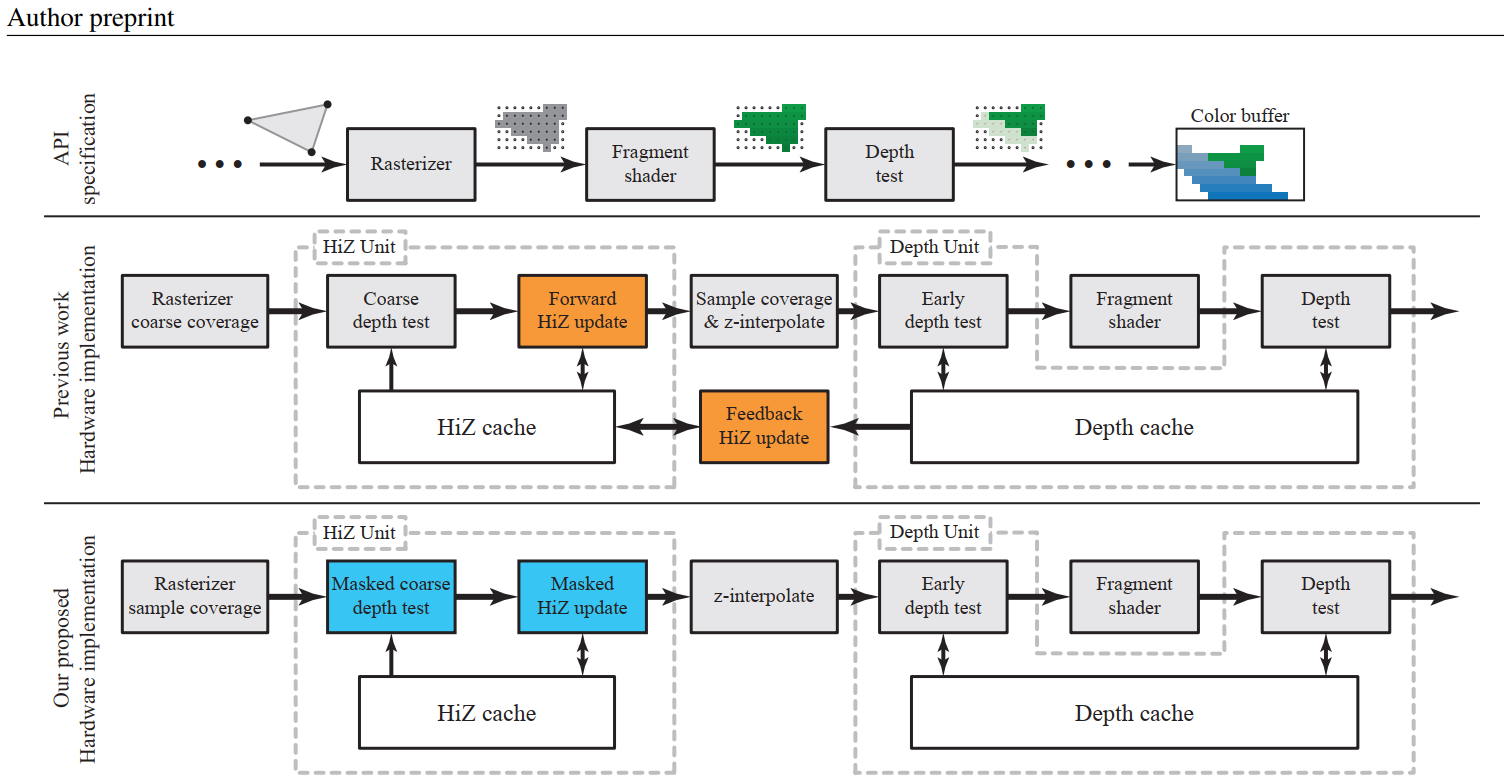


图2. 顶部：根据OpenGL和DirectX API规范的栅格化器，片段着色器和深度单位的简化概述。中：根据以前的工作，其硬件架构具有HiZ剔除单元和早期的深度测试。这些附加单元的目的是通过提前遮挡剔除来改善性能。它们对程序员是透明的，可以在不更改API的情况下实现。下：我们建议的体系结构。我们新颖的HiZ剔除算法通过使用带有每个样本选择掩码的分层深度表示，完全消除了对反馈HiZ更新的需要，这对于剔除非常有效且易于更新。

粗糙深度测试 当图块达到HiZ单位时，第一步是计算保守范围[zmin tri;瓷砖内传入三角形的深度的最大深度[zmax tri] [Akenine-Moller等。 2008]。然后使用区间重叠测试针对粗略深度缓冲区测试这些界限。例如，对于小于深度函数，我们可以得出结论，如果zmin tri≥zmax tile，则所有样本的按样本深度测试将失败，而如果zmax tri <zmin tile则通过。这就留下了一个模棱两可的深度范围，无法确定每个样本的深度测试结果。因此，粗略深度测试具有三种结果之一，即失败，通过或模糊。失败（即被剔除），立即将瓷砖丢掉，无需进一步处理。传递的和不明确的图块沿管道发送以进行进一步处理，主要区别在于，不明确的图块必须在深度单位中进行全面的深度测试，而琐碎通过的图块可以简单地覆盖深度缓冲区的内容。这是一个很小的差异，但是根据体系结构，仅写操作可能会导致带宽低于执行全深度测试所需的读-修改-写操作[Akenine-Moller和Str om 2003]。