摘要

对于实时应用程序，评估像素着色器会消耗越来越多的计算预算。但是，可见表面区域，光照条件和相机位置的显着时间连贯性允许在帧之间重用计算密集型着色计算，从而在视觉质量几乎不降低的情况下实现显着的性能改进。本文研究了一种基于反向重投影的缓存方案，该方案允许像素着色器存储和重用在可见表面点执行的计算。我们提供了一些准则，以帮助程序员选择合适的值进行缓存，并提供几种策略来使缓存的条目保持最新。我们的结果证实，该方法可为许多常见的实时效果（包括预先计算的全局照明效果，立体渲染，运动模糊，景深和阴影贴图）提供可观的性能提升。

1 介绍

随着专用图形硬件的功能和灵活性的不断增长，实时渲染应用程序中的明显趋势是像素着色复杂度的稳定增长。如今，图形处理预算中有相当一部分用于评估像素着色器。 因此，最近的研究已经研究了用于优化这些计算的通用技术，方法是降低其复杂度[OKS03，Pel05]，或减少生成的片段数[DWS88，NBS06]。

在本文中，我们开发了一种缓存策略，该策略利用了实时阴影计算的固有时空一致性（图1）。 在非常高的帧速率下以及在连续的帧之间，相机和照明参数以及可见表面点集，它们的属性和最终外观通常几乎没有差异。因此，从头开始重新计算每个帧可能会造成浪费。可以利用这种一致性来降低使用缓存机制生成单个帧的平均成本，该缓存机制允许在连续帧之间的像素着色器中存储，跟踪和检索昂贵的计算结果。尽管已经在不同的渲染上下文中开发了许多缓存技术，但是我们的技术是针对运行在商品GPU上的交互式应用程序而独特设计的，这些应用程序对可分配给缓存维护的计算资源和带宽施加了严格的约束。

我们介绍了一种新的缓存策略，该策略设计用于基于反向重投影的实时应用程序。 生成每一帧时，我们将所需的数据存储在视口大小的屏幕外缓冲区的可见表面点上。在下一个帧中生成每个像素时，我们将其表面位置重新投影到最后一帧中，以确定它先前是否可见并因此存在于缓存中。如果可用，我们可以重用其缓存的值来代替执行冗余的和可能昂贵的计算，否则，我们将从头开始对其进行重新计算，并使其在缓存中可用于下一帧。我们的方法不需要复杂的数据结构或CPU和GPU之间的总线流量，提供了有效的缓存访问，并且易于实现。

通过展示如何将其用于加速许多常见的实时着色效果，我们演示了该方法的实用性。我们报告的场景结果包含了预先计算的全局照明效果，立体渲染，运动模糊，景深和阴影贴图。总而言之，本文做出了以下贡献:

1. 我们针对实时着色计算引入了一种基于反向重投影的新缓存方案。它提供了一种通用且有效的机制，在时间上用于存储，跟踪和共享表面信息（第3节）；
2. 我们制定了一套准则来选择要缓存的值以及在什么情况下（第4节）；
3. 我们设计和评估各种刷新策略，以使缓存项保持最新（第5节）；
4. 我们开发了一种理论，用于在多帧内分摊随机估计数量的成本（第6节）；
5. 我们提出了一个可行的原型系统，并评估了我们的缓存技术是否适用于各种常见的实时渲染应用程序（第7节）。

2 相关工作

在许多渲染上下文中，已经研究了在附近的视点或连续在动画序列中生成的帧之间重用昂贵的计算。我们方法的主要区别之一是我们不重用可见性信息：仅重用着色计算。此外，我们专注于利用针对商品图形硬件的方法来开发实时像素着色器中的相干性。在这种情况下，重要的是最大程度地减少缓存开销，并确保所有计算都在GPU上进行（因此限制了GPU和CPU之间的任何总线带宽）。另一方面，使用现代图形硬件来实现缓存机制有许多计算优势。我们的方法利用了硬件支持的Z缓冲和硬件纹理过滤。

基于CPU的加速动画序列离线渲染的方法[Bad88，AH95，BDT99]通常通过正向投影将着色信息从一帧散射到下一帧。这产生了必须明确修复的间隙和遮挡伪影，从而增加了这些技术的复杂性并降低了其效率。甚至在最近的GPU加速版本[DWWL05，ZWL05]中，由正向投影难题方法所引起的类似问题也试图使离线渲染器具有交互性[BFMZ94，WDP99]。

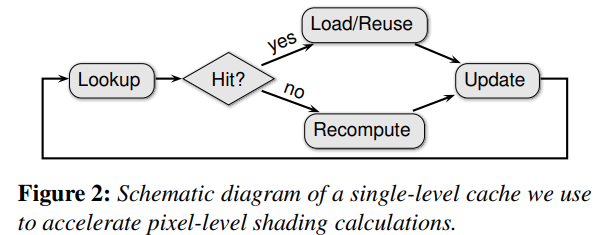
更好的选择是使用反向重投影，并利用硬件支持将样本收集到先前生成的当前帧中。如果GPU可以访问场景几何图形（即使是简化形式的图形），这也是可能的。 [CCC87]，[WS99]，[SS00]和[TPWG02]遵循此方法，并将样本存储在世界空间中。 不幸的是，这些技术需要在CPU上维护的复杂数据结构，并在CPU和GPU之间引入相当大的总线流量（特别是对于实时应用程序）。

除了将样本与几何直接关联之外，还可以将它们存储在映射到静态几何[SHSS00]的纹理中。 用更简单的基于图像的表示[RH94，MS95]代替几何图形以加快复杂环境的实时渲染的方法也采用了类似的想法。但是，维护和分配这些纹理也需要CPU干预。相比之下，我们的方法仅使用完全在GPU上维护的屏幕外缓冲区。解决缓存命中和未命中的问题由Z缓冲区本地执行，而检索则使用本地纹理过滤进行处理。与同样利用一致性的基于硬件的系统[RP94，TK96]不同，我们的缓存方案针对的是商品硬件，不需要程序员的明确控制。

最后一组相关方法利用空间相干性从一组图像有效地生成新颖的视图[CW93，MB95，MMB97]。 尽管我们的方法也可以解释为扭曲渲染的帧，但它被设计为支持动态生成的场景，例如游戏中的场景。

3 Reverse reprojection caching

图2中的示意图说明了我们用于改善像素着色器性能的单级缓存的类型。在生成每个像素时，着色器将测试高速缓存中是否有特定计算的结果。如果是这样，则着色器可以在最终像素颜色的计算中重用此值。否则，着色器将正常执行并存储可缓存的值，以在下一帧期间进行潜在的重用。请注意，存储在缓存中的值不必是最终的像素颜色，而可以是将从此类重用中受益的任何中间计算。



三个关键因素决定了使用这种类型的缓存修改的着色器是否优于其原始版本。首先，必须经常执行命中路径以证明其用途合理。其次，在发生缓存命中的情况下评估着色器的总成本必须小于未修改的着色器（这包括管理缓存的开销）。第三，存储在高速缓存中的值必须在连续帧之间保持相关性，以免在着色中引入明显的误差。

考虑到这一标准，我们提出了一个非常简单的缓存策略：将与可见表面点相关的值简单地存储在视口大小的屏幕外缓冲区中。这样可以实现高效的读写，并提供很高的缓存命中率（我们将在第4节中讨论在缓存中维护相关信息的第三个因素）。

图3显示了定量证据，表明此策略导致高速缓存命中率很高.它为图1中的动画序列绘制了在两个连续的帧中仍可见的像素比率.帕台农神庙序列使用静态几何图形和移动的摄像头来生成庙宇飞越。请注意，其缓慢的相机运动会导致非常高且均匀的命中率。女主人公序列显示了一个动画角色，带有经过加权的蒙皮顶点经过摄影机。在序列开始时连贯性的快速增加是由于她从右边进入场景。最后，忍者序列显示了一名动画战士在表演武术动作。他的快速脚步和动作导致相应地块的周期性下降。 对于我们用来测试方法的所有场景，我们观察到的命中率通常超过90％。

为了满足提供有效高速缓存访问的第二个条件，我们注意到，仅针对可见表面点维护高速缓存项的策略提供了许多计算优势:

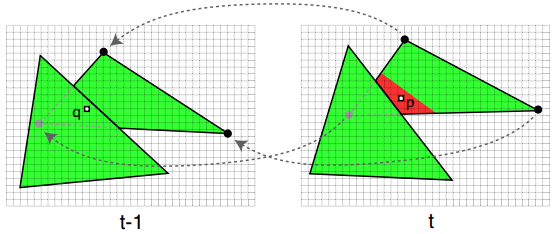
1. 每个像素仅使用一项，高速缓存存储器需求对输出敏感，因此与场景复杂性无关.
2. 高速缓存项与屏幕像素一一对应，因此在写入过程中不需要坐标转换.
3. 缓存查找所需的坐标转换可以在顶点着色器中高效执行（第3.1节）；
4. Z缓冲区中已经提供了查找过程中所需的每一项的深度（第3.2节）；
5. 对过滤后的纹理查找的硬件支持可以对缓存命中（第3.2节）和高质量的缓存读取（第3.3节）进行可靠的检测；
6. 数据永远不会离开GPU，从而消除了CPU的低效率总线通信.

接下来，我们详细介绍如何修改像素着色器以提供这种类型的缓存。

3.1 确定缓存坐标

我们面临的主要计算挑战是有效地计算像素的相应场景点在上一帧中的位置。我们利用对透视校正的插值[HM91]的硬件支持，并将此计算的大部分从像素级移到顶点级。

在时间,假定在每个像素处发生的计算结果已存储在屏幕空间缓冲区中(图4,左).在下一帧,在顶点着色器中计算时间处每个顶点的齐次投影空间坐标,应用程序向该物体提供了世界,相机和投影矩阵以及动画参数（例如补间因子和用于蒙皮的混合矩阵）.在我们的情况下,该应用程序还为顶点程序提供了处的转换参数,从而允许它计算前一帧的投影坐标.这些坐标成为每个变换顶点的属性(图4,右),这又导致硬件对其进行插值,自动为每个像素p访问投影空间坐标在时间产生表面点的.最终的缓存坐标是用像素着色器中的简单除以得到的.



**图4:左图**,帧着色计算和像素深度存储在屏幕空间缓冲区中.**右图**:在下一帧中,每个顶点也在时间处通过模型,相机和投影矩阵(以及任何动画参数)进行变换.这些值成为时间t的顶点属性,这些属性经过透视校正内插,使每个像素可以访问它在缓存中的位置.为了检测缓存未命中,我们将像素的重新投影深度与其在缓存中位置处存储的深度进行比较.

3.2 检测缓存命中

在时间的可见表面点在时间可能已经被不相关的点遮挡了(图4).除了几何相交的情况外,点和的深度不可能在时间匹配.因此,我们将在时间处的深度及其缓存坐标与在时间处深度缓冲区中的值进行比较(非常类似于阴影测试).如果缓存的深度在的预期深度的内,我们将报告缓存命中.否则,我们将报告缓存未命中.我们使用双线性插值来重建存储在前一帧的深度.跨越重大深度变化的值的加权总和将与从顶点着色器接收的重新投影深度不匹配,因此自动导致缓存未命中.结果,这大大减少了报告错误的缓存命中和在深度不连续附近错误地重建值的机会.为了进一步提高鲁棒性,我们将设置为最小的Z缓冲区增量（也可以使用[AS06]之类的技术来计算该值）.

3.3 缓存重采样

与基于正向重投影的方法相比，我们的方法的主要优势在于，它将有问题的缓存样本散射转换为可管理的收集过程。但是，由于重投影像素通常不会映射到各个缓存的样本（图4），因此需要某种形式的重采样。缓存的统一结构和对纹理过滤的硬件支持大大简化了此任务。 实际上，除了深度不连续之外，缓存查找可以完全视为纹理查找。

最佳的纹理过滤方法取决于要缓存的数据及其在像素着色器中的功能。**最近邻居插值足以使平滑变化或由着色器进一步处理的缓存数据。另一方面，当缓存项中存在相当大的空间变化时，尤其是如果将显示该值时，双线性插值是合适的**。但是，不管采用哪种方法，跨多帧重复对缓存进行重采样都会最终使高频内容衰减。

尽管我们可以通过使用双线性插值法来重构深度来避免在遮挡边界附近对缓存进行重采样（第3.2节），但这不会阻止更宽泛的重构内核（例如三线性或各向异性滤波器）跨无关数据进行集成。但是，实际上，帧之间的场景变化很小，从而限制了重投影图中的失真量，并且无需使用更复杂的重建方法。

3.4 控制流策略

我们可以区分缓存命中和未命中这一事实，使我们能够编写像素着色器，以针对每种情况执行不同的路径。我们将所需的代码路径分别称为“命中”着色器和“未命中”着色器。

最直接的方法是根据深度测试在命中和未命中着色器之间进行分支。如果硬件支持动态流控制，则执行成本将取决于所采用的分支。但是，图形硬件的一个重要特征是，计算是按锁定步骤进行评估的：屏幕的整个块是并行执行的，每个块的速率与其最昂贵的像素成比例。因此，大像素块中的一个高速缓存未命中将惩罚整个区域的执行时间。幸运的是，在连续帧中可见像素的空间相干性（图1）大大减轻了这种影响，导致屏幕的大连续区域遵循相同的控制路径。

动态流控制的一种替代方法，是避免因执行锁定步骤而受到惩罚的方法，是依靠早期的Z剔除[SIM05]，该方法在评估关联的着色器之前测试像素的深度。在第一遍中，将执行高速缓存查找，如果命中着色器成功，则将执行命中着色器。如果未命中，则仅对像素进行深度移位以填充Z缓冲区。在第二遍中，早期的Z剔除可确保仅在那些像素上执行Miss Shader，而每个像素仅执行一次。不幸的是，在当前硬件中，深度移位操作无法在第一遍中使用早期的Z剔除。但是，由于热门着色器相对便宜，因此不会导致性能大幅下降。为了支持第5.2节中所述的细粒度，随机刷新策略，本文的大多数结果都是使用早期Z剔除方法生成的。

3.5 计算开销

图5显示了缓存查找过程的示意图。对顶点着色器的一种修改是，应用程序必须向其发送两组转换参数。由于许多实时应用程序经常会达到此类存储的硬件极限，因此尽管最近的硬件修订版提供了更大的极限以符合Direct3D R 10系统[Bly06]，但任何增加都可能会引起问题。

此外，我们的策略要求对几何图形进行两次转换：一次用于当前，另一次用于前一帧的查看参数。但是，如果渲染成本主要由像素处理决定，那么硬件肯定会欢迎在额外的顶点负载和像素负载的大幅减少之间进行权衡。此外，由于最新的GPU是统一架构，因此即使像素处理不是确定渲染成本的唯一因素，该系统也可以从像素处理的任何显着减少中受益。

在像素级别，我们的缓存策略需要额外一次除法和两次纹理查询。其中一次查询用于高速缓存的深度，另一个用于该像素的有效载荷信息。动态控制流机制也有开销（第3.4节）。自然，为了证明缓存的合理性，要替换的计算必须比增加的开销更为昂贵。

关于屏幕外缓冲区管理，在每个像素处缓存单个计算需要一个额外的深度和大小与视口相同的颜色缓冲区。可以通过将项存储在不同的颜色通道中或多个缓冲区（对于许多应用程序，单个未使用的alpha通道可能就足够）中来缓存多个值，对于这些缓冲区，可用的图形硬件最多支持八个并发渲染目标[Bly06]。

4 确定缓存数据

尽管重新使用昂贵的着色计算可以减少生成单个帧的延迟时间，但这会给着色带来与计算之间的变化率成比例的误差。例如，当相机和照明发生变化时，缓存高度抛光的对象的最终颜色将无法捕获变化的镜面高光。

当选择要缓存的值时，程序员应设法使其相关的计算量（例如，机器指令的数量和纹理获取）相对于其在帧之间的导数的大小的比率最大化。尽管我们将缓存内容的最终决定权留给了程序员，但我们已经确定了满足以下条件的几种着色计算类别：

1. 着色模型结合了昂贵的计算，显示出较弱的方向依赖性（例如，程序生成的漫反射反照率）;
2. 多遍渲染效果，将附近视图中的几幅图像组合在一起（例如，运动模糊和景深）；
3. 需要采样的效果在每个像素处的变化缓慢（例如抖动的超级采样阴影图）。

包含间接（或全局）照明效果的交互式着色技术通常属于第一类。确实，场景的全局着色分量通常表现出低频方向依赖性[NKGR06]，但是模拟这些效果可能需要大量计算。最简单的示例是局部照明模型中的漫射组件，该组件完全独立于视角和入射照明的方向（忽略余弦衰减）。如果将此模型建模为一个复杂的过程[Per85]，则在多个帧之间重用此值将改善渲染性能，而不会影响着色的准确性（图6）。其他示例包括预先计算场景中光线的全局传递以进行交互式显示的方法[SKS02]。这些要求昂贵的局部着色计算，通常导致值具有低频方向依赖性。在第7节中，我们应用我们的技术来加速一种在复杂照明下渲染半透明对象的单散射分量的方法[WTL05]。

通过缓存在第一遍过程中执行的昂贵计算，然后在后续遍中重用这些计算，还可以加快组合从附近视图渲染的几幅图像的多遍效果。由于摄像机在多个Pass之间的运动是已知的并且是固定的，因此我们的方法引入的误差是有限的。此外，组合视图的接近度放松了对缓存的计算表现出有限的方向依赖性的要求。我们将在第7节中介绍运动模糊,景深和立体效果的结果.

我们的缓存基础架构还支持分摊在多个帧上随机采样函数的成本（第6节）。这最适合在每个像素处固定（或缓慢变化）的采样功能。在7.3节中，我们描述了如何改善从阴影贴图渲染抗锯齿阴影边缘的流行技术的性能。

5 刷新缓存值

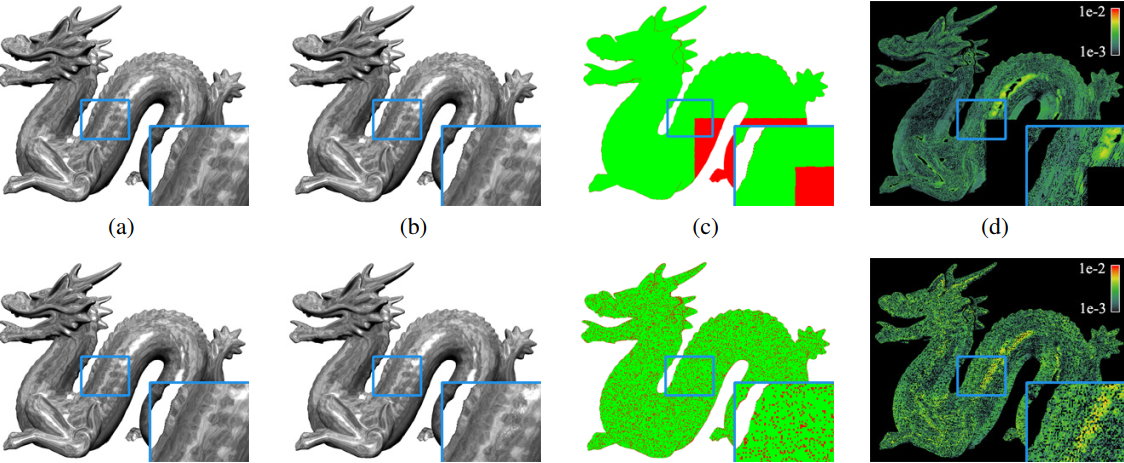
场景运动，变化的表面参数和重复的重采样最终会降低缓存项的准确性，因此必须定期刷新它们。如果我们将刷新率设置为与帧之间的变化率成比例，则可以通过重用计算来控制着色误差。 当然，并非总是可以事先预测其变化，因为由于未知的用户输入，它可能取决于场景运动。相反，我们依靠程序员根据第4节中的准则选择值，然后手动设置适当的刷新率。

通过在每帧更新大小为的不同区域，我们可以确保整个缓存至少每帧刷新一次。这样可以在时间上平均分配计算开销，从而获得更流畅的动画。我们比较了将屏幕划分为刷新区域的两种策略。

* 1. 图块刷新策略

我们将屏幕划分为个不重叠的图块的网格，并保持在每个帧处递增的全局时钟，并将其作为统一属性传递给像素着色器。生成每个像素时，像素着色器将其图块索引添加到时钟，在以下情况下触发刷新（即，即使在缓存命中时也执行未命中着色器）：

这具有依次刷新每个图块的效果.图6分析了此刷新策略对将Perlin噪声扩散层与BlinnPhong镜面反射层组合在一起的简单着色器的影响.当用户交互式地调整相机和点光源时,我们缓存最终颜色并每四帧刷新一次其值.因为访问缓存比执行此计算要便宜得多,所以我们在错误可忽略的情况下将性能提高了一倍.



**图6**：刷新策略比较。（a和e）用户调整相机和点光源的位置时,龙模型的交互序列中的一帧被Perlin噪声扩散层和镜面反射层着色化（复制相同图像以并排显示）方面的比较。（b）使用四个平铺的刷新区域缓存最终像素颜色的结果。（c）相干图，其中显示重新计算了哪些像素（红色）以及从缓存中检索了哪些像素（绿色）。（d）每个像素着色误差的假彩色可视化。请注意，在要刷新的图块内部的像素处的五擦汗为零。（f，g，h）对同一场景使用随机分布的刷新区域的结果以及相关联的相干图，该图显示了缓存命中的分布和着色错误。两种策略每四个帧刷新一次项，并提供近100％的性能提升（即，常规渲染为35fps，而基于图块的和随机的分别为60fps和67fps）。

* 1. 随机分布区域刷新

我们还尝试了在整个屏幕上形成了随机分布的图案（请参见图6）的刷新。我们已经发现这些模式产生的感觉不那么令人反感的伪像，将图块边界处的尖锐不连续性交换为均匀分布在整个图像上的高频噪声。但是，如果在现代图形硬件中天真地实施以锁定步长执行相邻像素的方法，此策略可能会降低性能（第3.4节）。在这些情况下，重要的是要使用早期的Z消隐技术来提供流控制，以允许更新小至2×2像素的随机分布区域。

这些刷新模式可以通过对每个像素预先计算并存储随机分布的积分偏移来实现。我们生成这些，以便2×2区域具有相同的偏移量（参见图6）。在渲染期间，像素着色器根据其屏幕位置访问其偏移d并将此值添加到全局时钟t，并在以下条件下刷新：

* 1. 隐式刷新

许多受益于我们技术的着色器不需要显式刷新缓存。通过多次重用仅在单个帧内的遍之间的值，可以在多个渲染遍中计算出的效果可以利用缓存的好处。因此，缓存将在下一个帧的第一遍中完全刷新，从而避免了上述策略所需的任何开销。 我们的摊销抽样方法也是如此（第6节）。 在这种情况下，值会随着新样本的积累而迅速而平滑地衰减。

6 平摊采样

图形中的许多量是由随机过程产生的，该过程结合了函数[DW85，Coo86]的许多随机选择的样本。交互式应用程序受其计算预算可提供的最大样本数限制.我们的缓存基础结构允许分摊多帧函数采样成本，从而以可比的帧速率提高这些估计的质量.如第4节所述，此方法最适合在固定或缓慢变化的每个像素处对函数进行采样。

我们的目标是计算

其中是感兴趣的函数,而是积分域(例如,如7.3节所述,单个像素内的着色覆盖率).蒙特卡洛技术[RC04]将方程3近似为根据概率密度选择的个样本的加权和:

衡量其质量的近似值的方差与样本数量成反比,与分布的质量成正比:

在每一帧,我们都可以将缓存项替换为其当前值和一个新样本的加权组合,并按其概率加权：

可以很容易地证明是方程3的无偏估计,其方差为:

注意,任何样本对当前估计值的相对贡献均呈指数下降,时间常数等于.因此,的值控制了估计器的方差和对场景变化(即,对的变化)的响应度之间的权衡.图7量化了这种关系,其中总衰减为以帧为单位测量的时间,直到样本按1/256缩放(即以8位精度完全丢失)为止,例如,选择的值可将方差减小至原始值的1/4（公式7）,并有效地合并前10帧的样本（另请参见图11b和11d）.相反,将方差减小1/8倍需要设置,并将总衰减量增加到22帧.实际上,我们凭经验确定.

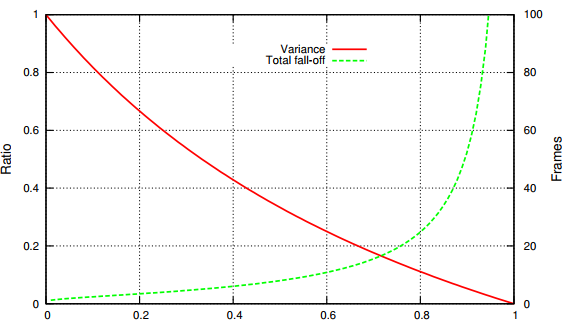


图7：当使用递归滤波器执行摊销的超级采样时,在减少方差的量（方差曲线）和有助于当前估计的帧数（总下降）之间要权衡 -断曲线.这种权衡由参数λ控制.

7 结果

我们已经使用缓存策略来加速几种常见的交互式渲染技术。这些是根据第4节中的指南进行选择的，包括基于预先计算的光传输的半透明对象的着色模型，在多个渲染过程中计算出的几种常见效果以及渲染抗锯齿阴影贴图边界的技术。

我们与常规渲染方法的比较着眼于质量和性能之间的权衡。对于像素着色和几何处理复杂性之间具有可比平衡的应用，我们报告的趋势将相似。我们的结果是使用带有ATI X800显卡的P4 3.2GHz生成的。

7.1 半透明材质的着色模型

我们使用我们的方法来加速基于预计算光传输[SKS02]的技术，以在复杂的全频远距离照明[WTL05]下交互式渲染半透明物体。 他们的模型考虑了[JMLH01]提出的完整的双向表面散射反射率分布函数（BSSRDF），其中包括单次散射和漫射多次散射。多重散射项与视图方向无关，因此可以使用一组按顶点传递向量进行捕获，这些向量使用非线性小波逼近[NRH03]进行了预先计算和压缩。这些矢量的点积与以相同小波表示的环境照明在CPU上进行计算，成为顶点属性.

他们使用单个散射项[JB02]的近似值,该近似值允许将任意相位函数分解为个项的总和，这两个函数的每一个分别仅取决于局部光和视角方向.他们预先计算了个额外的每个顶点颜色,这些颜色捕获环境照明和这些与光有关的项的乘积.视相关项存储在纹理贴图中,并在渲染期间在局部折射视图方向进行评估,从而得到完整的着色模型:

对于光密材料,输出辐射呈现出非常低的频率方向依赖性.但是,评估方程式8需要获取个纹理，并且对于精确分解而言可能会很昂贵。因此，在多个帧之间重用此计算仅以边际着色误差为代价而换取生成单个帧的成本.

图8将原始着色器的性能‡与应用我们的缓存策略以在连续帧的每个像素处重用公式8的结果进行了比较。我们没有明确刷新缓存，而仅在缓存未命中时重新计算（请参见图8（a）中的切口）。最后，我们将用两个替换4个纹理获取（一个用于解决缓存命中，一个用于获取实际有效载荷）以及维护缓存的计算开销。 对于此场景，我们观察到帧速率提高了30％。对于更复杂的预计算辐射传递技术，我们期望获得更好的结果。

7.2 多遍渲染效果

浮雕立体图像在单独的颜色通道中对场景的双目视图进行编码，并且可以在两个渲染过程中生成（请参阅[Dub01]以得到很好的评论）。这些视图的接近度使我们可以在对立视图的相互可见点重用为一个视图计算的着色信息。尽管先前的工作已经使用重投影来加速此技术，但它已应用于光线跟踪[AH93]和头部跟踪显示器[MB95]的背景中。

图9a演示了重新使用具有Perlin噪声函数的着色模型的最终颜色的效果，该函数需要每个像素512条指令（昂贵，但并非不合理）。如图9b所示，与地面真实情况的比较揭示了镜面高光周围明显的阴影错误。在图9c中，我们仅缓存和重用昂贵的噪声计算，并在每次通过时重新计算镜面反射贡献。此示例强调了选择值以缓存在帧之间逐渐变化的值的重要性。

蛮力立体渲染可在我们的系统上实现28fps。 仅缓存漫反射组件会将帧速率提高到39fps，而将最终颜色缓存则达到44fps。我们的方法可将帧率提高57％，视觉质量损失可忽略不计。

通过组合在稍微不同的时间点或从附近的视角渲染的场景的几张图像，可以模拟运动模糊和景深效果[HA90]。它们强大的时空连贯性使得可以在随后的遍历中重复使用在第一遍中计算出的昂贵的阴影计算-[CW93]和[HDMS03]分别在基于图像的渲染和光线跟踪动画的背景下探索了这一思想。此外，将多个图像平均在一起往往会模糊阴影误差，并将我们的技术使用范围扩展到具有更强视相关效果的值。

图10将用于渲染运动模糊和景深效果的蛮力技术与通过缓存获得的结果进行了比较。所示模型具有2.5k三角形和与图9相同的阴影。我们的技术允许以相同的质量渲染此场景大约两倍的速度，或者相反，以相等的帧速率组合两倍数量的样本。

7.3 抗锯齿阴影贴图边界

阴影图[Wil78]已成为以交互速率显示阴影的必不可少的工具.首先从每个光源的投影中心渲染场景，然后将Z缓冲区的内容存储在称为阴影贴图的纹理中。 从观察者的角度渲染场景时，每个像素都会测试其在每个地图中的位置，并累积可见光源的贡献。

由于摄像机处像素的采样模式与光源不同，因此这种简单的技术通常会受到混叠问题的困扰（图11a）。一种解决方案是提高阴影图[FFBG01，SD02]的有效分辨率。另外，里夫斯等人[RSC87]引入了百分比近似滤波（PCF），以通过用每个像素内的多个随机样本的平均值近似部分阴影覆盖率来减少这些伪影的方法（请参见图11b）。在我们的实验中，PCF通常需要少至16个样本来解析可接受的阴影边界（图11c）。

第6节所述的摊销抽样方法非常适合优化PCF。图11d显示了通过随机旋转固定采样模式并使用λ= 3/5的加权因子在缓存中递归累积这些采样在每个帧上生成4个采样的结果。这将我们的估算器的方差减小了1/4（公式7），从而提供了与每帧使用16个样本的原始方法可比的质量的图像（比较图11c和11d）。