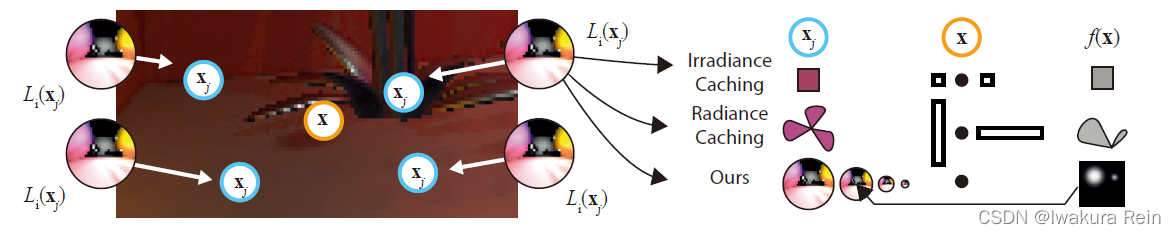
Real-time Neural Radiance Caching for Path Tracing

软件学院 郑沧平 22351280

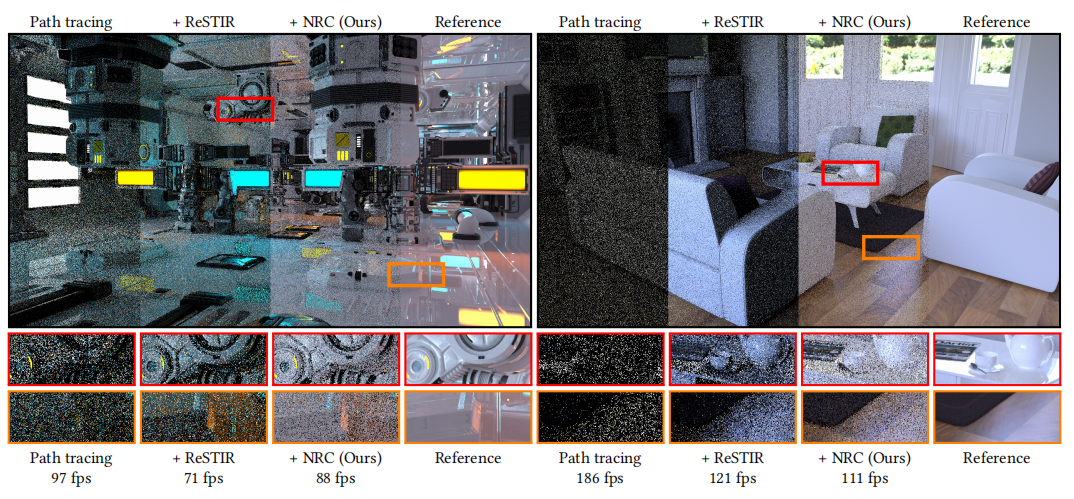
1. **前置知识**

**1.1 radiance cache**

我们知道Path tracing慢的一个原因，就是要递归求解渲染方程，这在实时渲染中是难以接受的。那么很自然有一个思路就是把场景的Radiance全cache起来，然后在合适的时机终止路径，直接用cache的结果。这是一个老算法，主要就是把一个点辐射出的Radiance 编码成球谐函数（Spherical Harmonics），然后在实时渲染中帮助做shading。Radiance caching可以作为预计算，也可以用来加速实时光追。简单来说，辐照度缓存将每个像素的入射辐照度投影到球谐函数（SH）基础上，将BRDF转换为SH，并对每个像素的两个密集向量执行点积运算。



而NVIDIA发表在Siggraph 2021的文章《Real-time Neural Radiance Caching for Path Tracing》提出了一种用神经网络实时拟合radiance cache的方法，称为Neural radiance caching（NRC）。具体来说，它把自训练（self-training）神经网络融入到了光追实时渲染中，实时训练一个8 层各64个神经元的全连接网络Neural Radiance Caching，并且做到了更新和缓存查询仅产生轻微的开销——在full HD分辨率（1920x1080）下时间代价约为2.6ms，以此在引入小偏差的情况下降噪、在模拟有限次弹射的情况下用神经网络预测无限次弹射的间接光照。



**1.2 自训练算法**

自训练（Self-Training）是一种半监督学习算法，用于在标签稀缺但未标注数据丰富的场景下提升模型性能。其基本原理是，首先在有标签的数据集上训练模型，然后使用这个模型来预测未标注样本的标签。接着，将预测置信度较高的样本及其预测的伪标签加入训练集，重新训练新的模型。这个过程不断迭代，通过不断地使用模型自身的预测结果来扩展训练集，从而提高模型在未标注数据上的表现。

自训练常用于半监督学习场景，特别适用于那些标签稀缺但存在大量未标注数据的情况。通过这种方式，模型可以充分利用未标注数据中的信息，从而提升其性能。同时，自训练也可以作为迁移学习的一种策略，将模型从一个任务迁移到另一个任务时，通过在新数据上进行自训练来适应新的任务。

**2 Neural Radiance Caching**

由Kajiya提出的渲染方程（式1），可知即为该着色点的辐射度（radiance）：



本文所提出的Neural Radiance Caching（简称NRC）的目的，就是估算，估算值记作。首先，对算法进行介绍。

**2.1 算法概述**

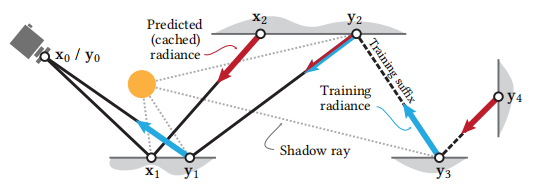
渲染每帧包括两个阶段：

（1）计算着色

对每个像素仅发射一条光线，如图中（x0、x1、x2）和（y0、y1、y2）。对它们正常地做实时光线追踪，并在终点处（即x2和y2）采用NRC的输出。文中提到所使用的的光线追踪算法如下：在初级顶点使用screen-space ReSTIR计算直接光照，在后续顶点使用多重重要性采样（MIS），采样策略为LightBVH和BSDF。

1. 训练模型

对于一小部分（通常少于3%）路径（如图中的y0、y1、y2）延长几个顶点，形成一个训练后缀（y2、y3、y4）。对于终点（即y4）仍然采用NRC的输出，但是要把这个 Radiance 值沿着光路反向传递回去，按照path tracing的算法计算出之前的弹射点处（即y1、y2、y3）的能量值，以此作为输入NRC的训练数据。



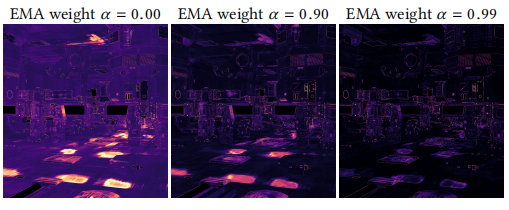
图中的红色箭头表示该点的radiance采用NRC的输出，蓝色箭头表示该点的radiance由Path tracing计算得到，作为NRC的输入。NRC的输出通过一定的计算步骤又变化为NRC的输入，这就是自训练。

**2.2 支持动态场景**

在渲染动态内容时，例如改变相机位置或动画几何形状，NRC需要不断地进行适应。因此，需要使用较高的学习率（learning rate），同时使用多重梯度下降（[multiple gradient descent](https://www.zhihu.com/search?q=multiple gradient descent&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"article","sourceId":"684918141"}" \t "https://www.zhihu.com/_blank)）算法。然而，这会导致渲染帧之间出现时间伪影，如闪烁和振荡，因此文中提出对神经网络的权重做了emporal filtering，采用指数滑动平均（Exponential Moving Average, [EMA](https://www.zhihu.com/search?q=EMA&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"article","sourceId":"684918141"}" \t "https://www.zhihu.com/_blank)）的方法：

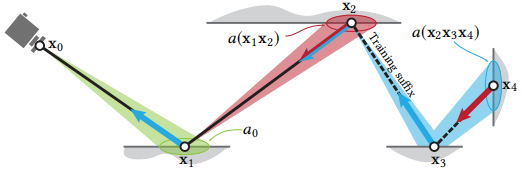


其中越大，代表惯性越大，即使用更多的过去信息，论文使用。本方法大约能在8帧左右（约70ms）收敛到理想值。下图展示了不同参数下的图像稳定性，颜色越深表示帧间变化越小。

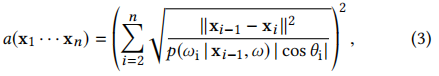


**2.3 路径终止算法**

本文采用了一种非常有趣的路径终止算法。使用area-spread去衡量光追路径顶点所覆盖的区域，当区域的面积足够大时终止路径。



一条路径所对应的面积可以被估计为：

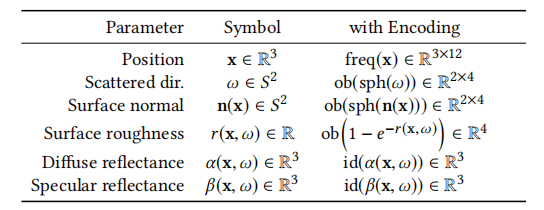


其中p是BSDF采样PDF，θ是光线与弹射点的法线的夹角。设置一个超参数c，当时，终止路径。

**2.4 网络输入**

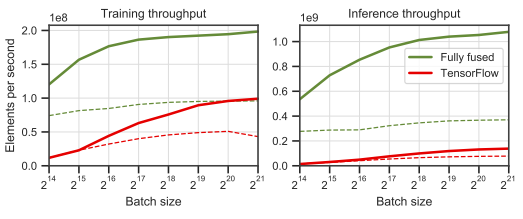
最自然的想法就是用时空坐标（x，ω）作为神经网络的输入，然而这并不能很好地表示辐照度。因此，输入通过与其他与散射辐照度相关的参数进行增强：表面法线n、表面粗糙度r、漫反射率α和镜面反射率β。通过能够利用这些相关性，神经网络将变得更加准确。当这些相关性是（近似）线性的时，网络更容易识别它们。对于漫反射和镜面反射率来说，情况已经如此；因此，我们直接将它们作为输入提供给网络。然而，x、ω、n和r这些量与散射辐照度之间存在高度非线性关系。对于这些量，选择一个合适的编码方式将它们映射到更高维度的空间，可以使关系更加线性化，从而提高神经近似的准确性。

因此，我们使用[Neural Importance Sampling（ACM ToG2021）](https://link.zhihu.com/?target=https://dl.acm.org/doi/10.1145/3341156" \t "https://www.zhihu.com/_blank)中提出的one-blob encoding，当参数的微小变化不会引起radiance（即我们想要的网络输出）的剧烈变化时，这是一个很不错的编码方式。网络输入的参数如下：



**2.5 基于Tensor Core的实现**

为了充分利用GPU的内存层次结构，论文作者从零开始使用GPU编程语言实现了神经网络，而没有使用TensorFlow等现成AI框架。其效率比TensorFlow提高了5-10倍。



这是因为神经网络的计算成本与其宽度成二次方关系，而其内存流量则呈线性关系。然而，现代GPU的计算吞吐量远大于其内存带宽。这意味着对于像这样的窄神经网络，线性内存流量成为了瓶颈。因此，提高性能的关键在于尽量减少对慢速“全局”内存（VRAM和高级缓存）的访问，并充分利用快速片上内存（低级缓存、“共享”内存和寄存器）。

**3 总结**

Neural radiance caching的效果非常惊人，它能够实时地拟合场景中的radiance信息，对渲染效果有显著提高。这项技术为实时渲染领域带来了新的突破和可能性。