



# 深入RTX实时光线追踪技术：原理、接口、 算法与应用

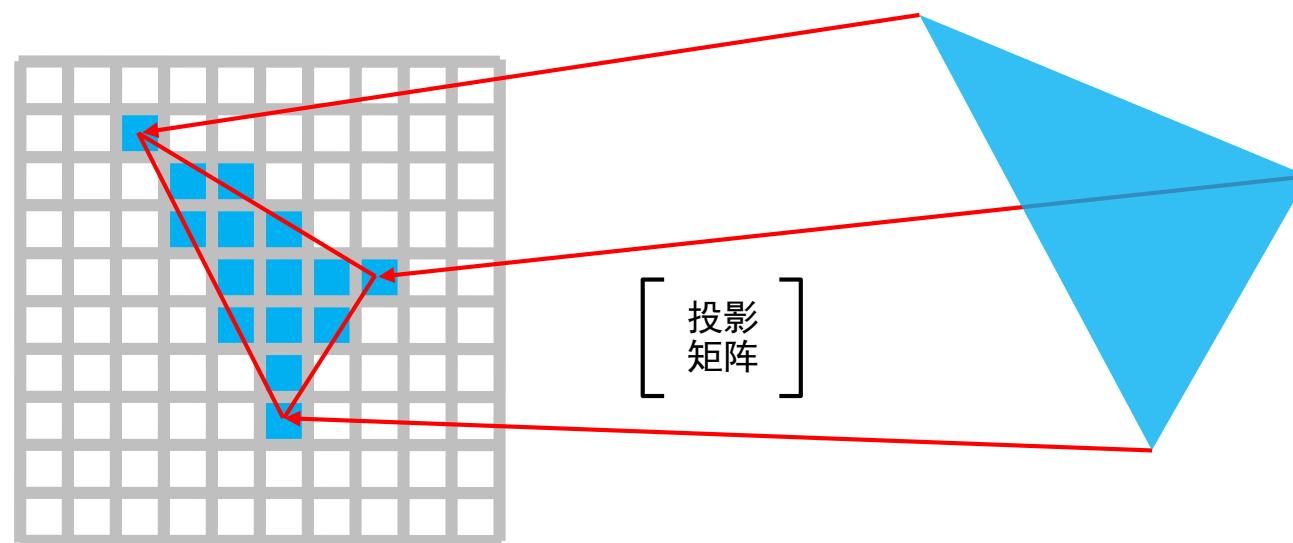
李元亨 内容技术开发工程师, 11/22/2018

# 概要

- ▶ 光线追踪与光栅化
- ▶ DirectX Raytracing
- ▶ NVIDIA RTX与实时光线追踪
- ▶ 最佳实践

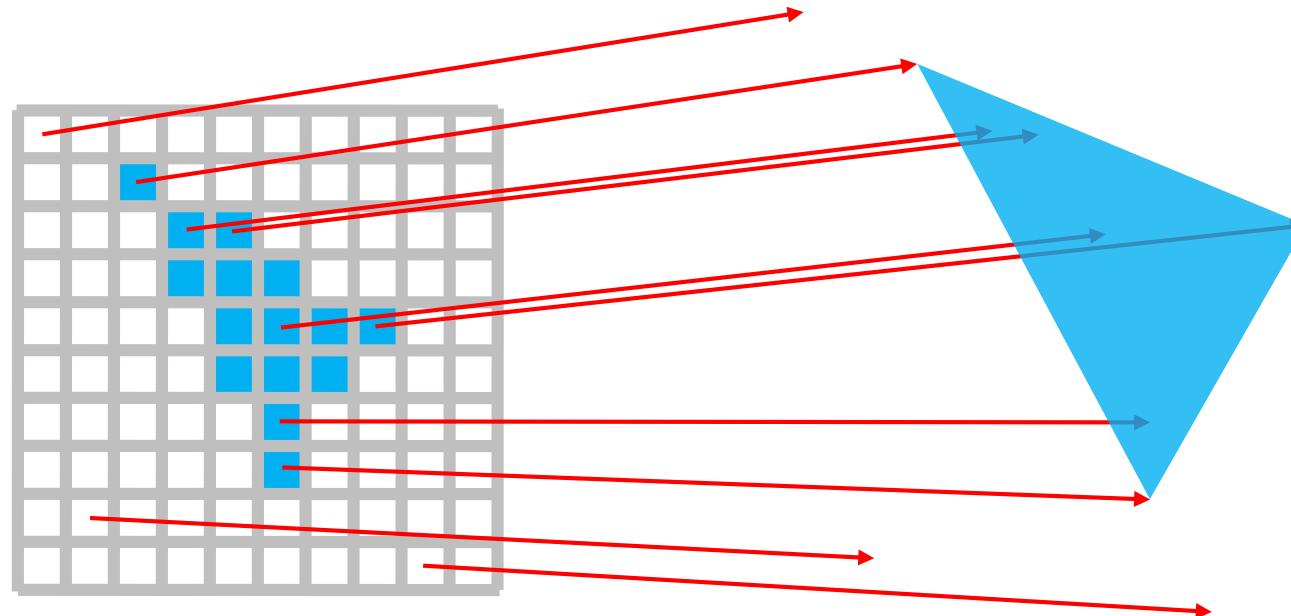
# 光栅化 (RASTERIZATION)

- ▶ 通过投影矩阵把物体投影到屏幕上



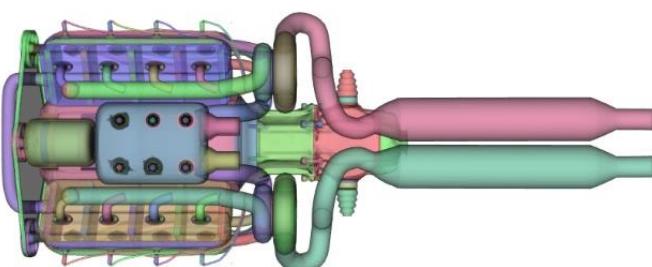
# 光线追踪 (RAYTRACING)

- ▶ 向场景中发射射线，找到射线与物体的交点



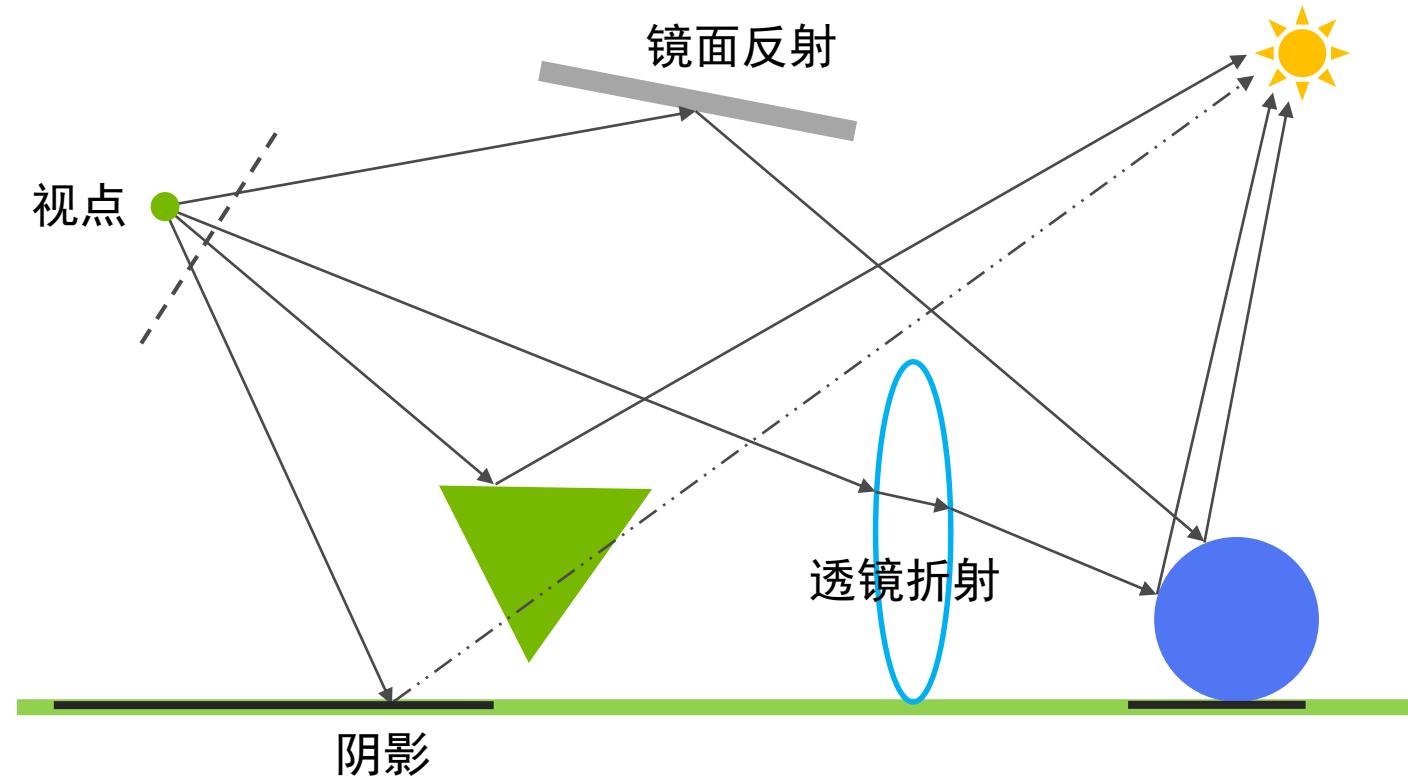
# 光栅化VS光线追踪

- ▶ 光栅化下很多效果实现非常困难



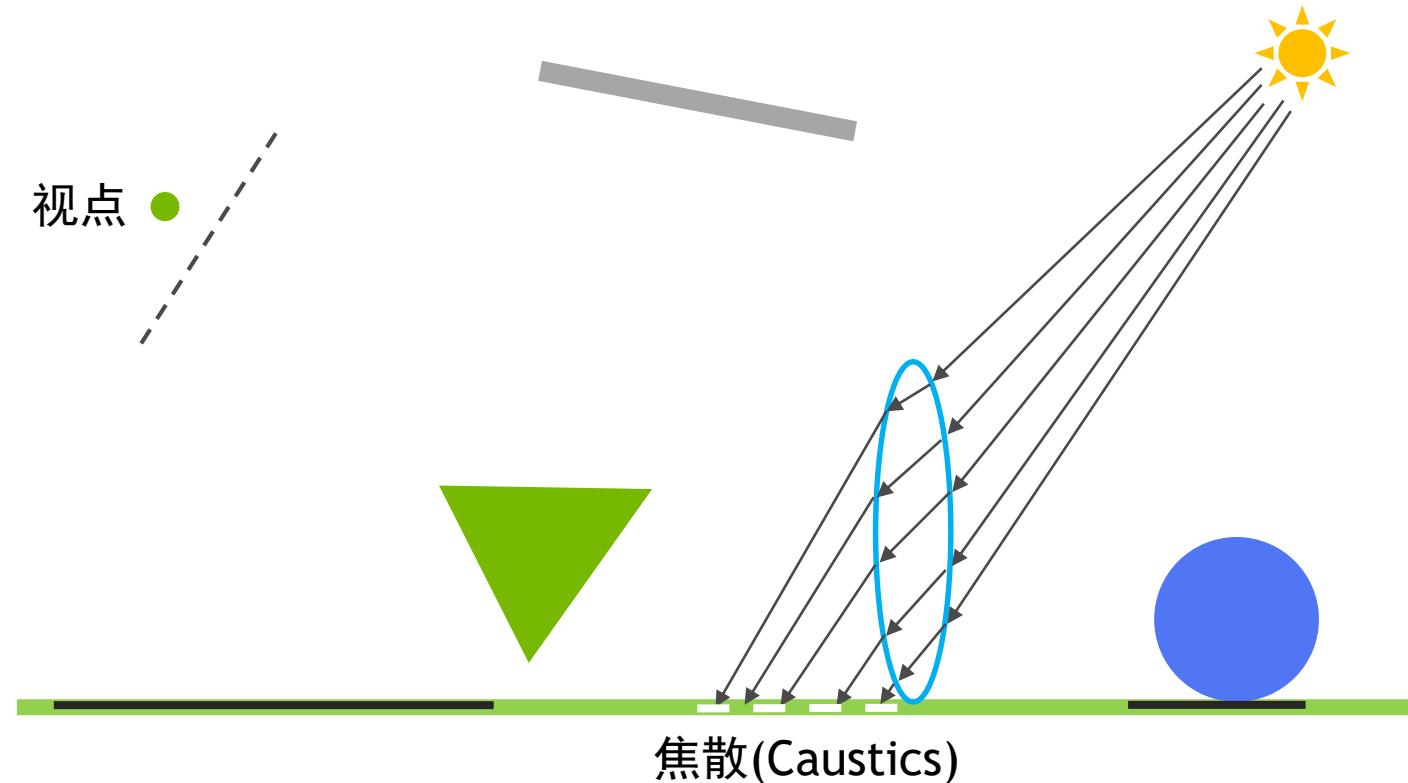
# 光栅化VS光线追踪

- ▶ 使用光线追迹能够从容应对

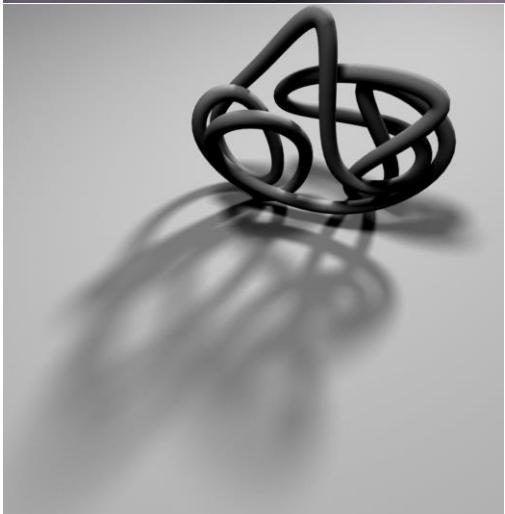
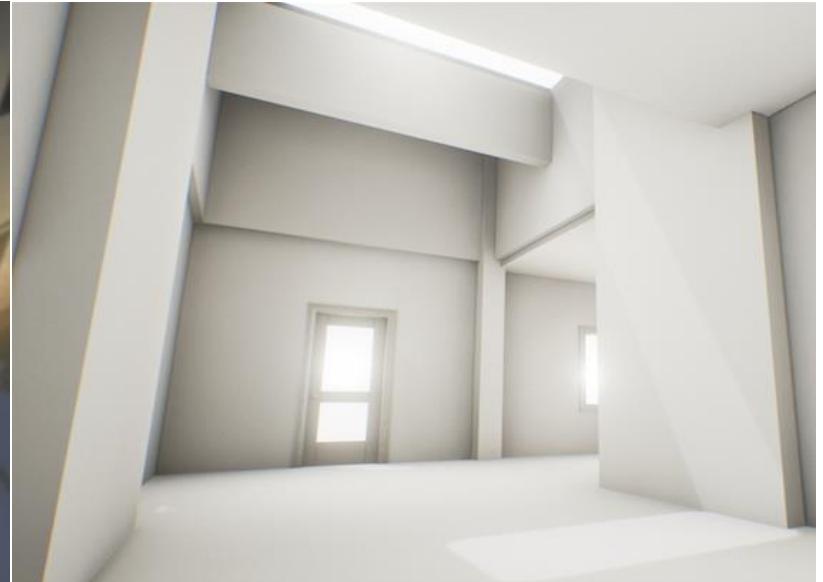


# 光栅化VS光线追踪

- ▶ 使用光线追踪能够从容应对

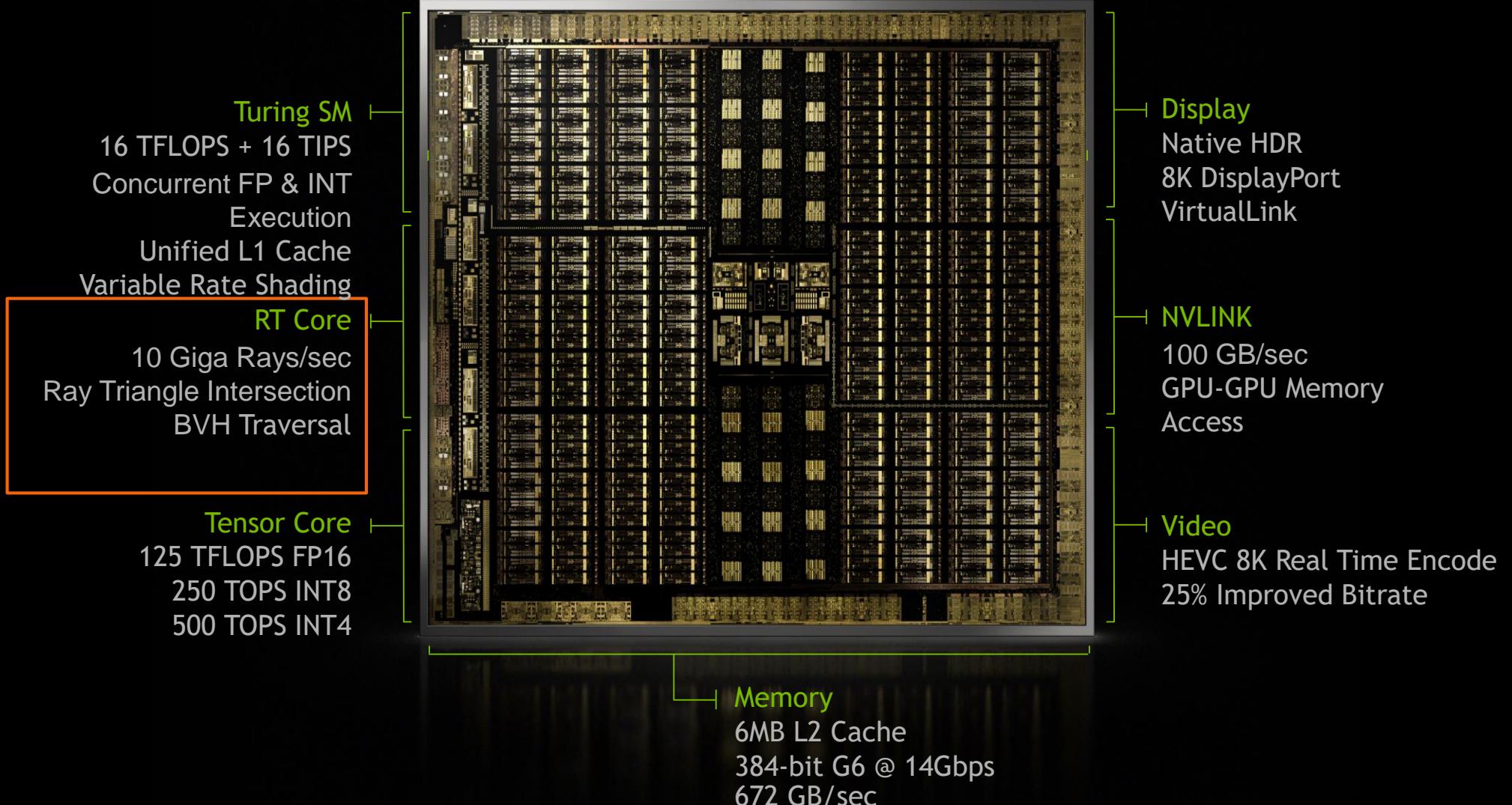


# 光线追踪



# TURING GPU

18.6 Billion xtors | 754 mm<sup>2</sup> | 48+48 GB 14GHz



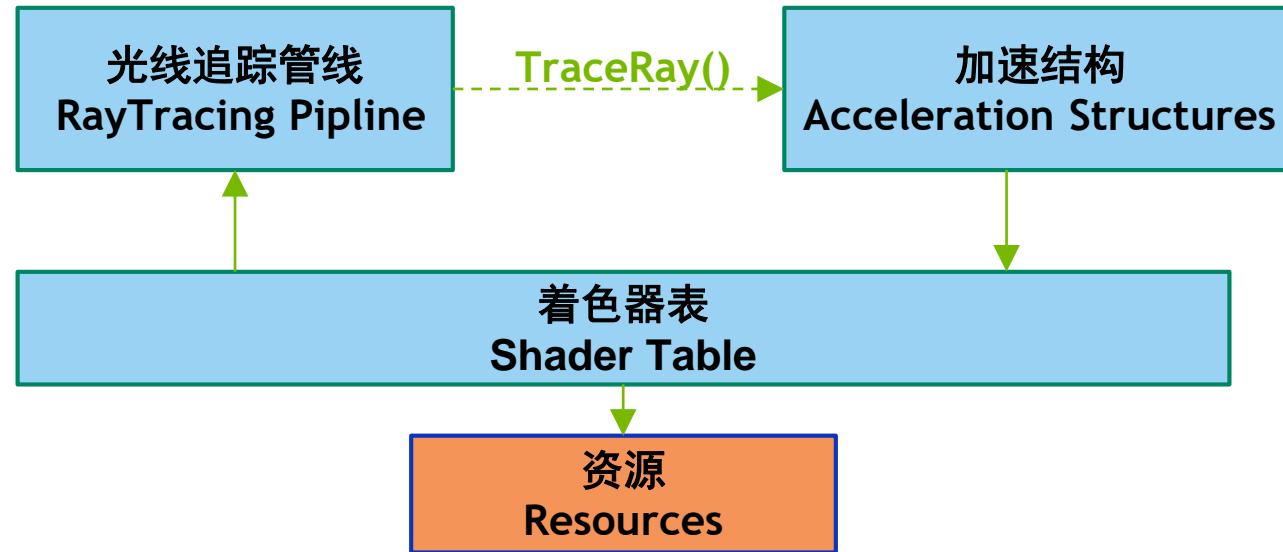
# 图灵中的光线追踪



# DIRECTX RAYTRACING

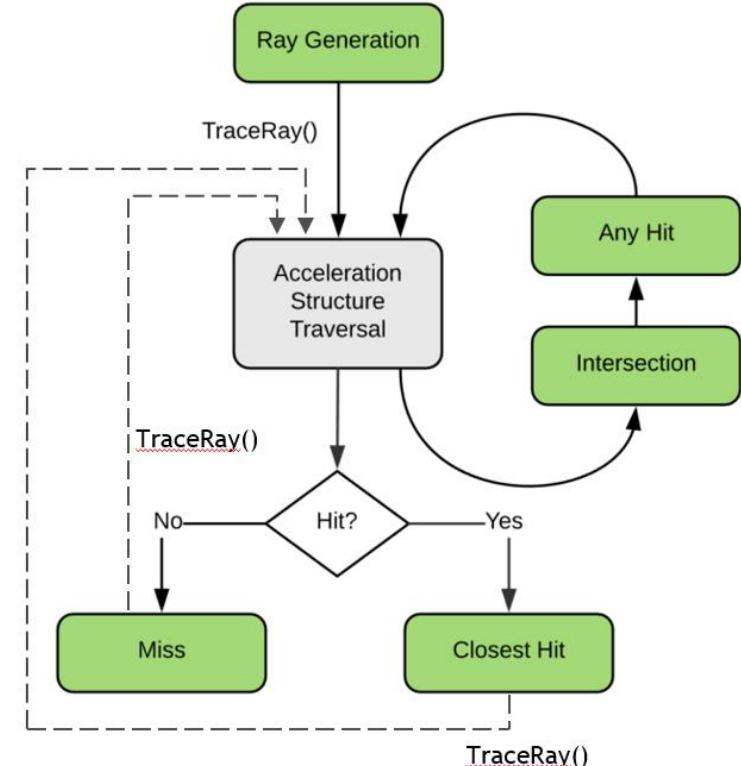
- ▶ DXR API为D3D12的直接扩展
  - 2018年10月正式版已发布（集成至Win10 RS5）
- ▶ 设计概要
  - 紧密结合光栅化与通用计算管线，易于实现混合渲染
  - 通过D3D12已有的command list和queue调用
  - 共享现有的buffer, constants和textures等资源，无需转换和映射
  - 光线追踪专用资源同其它D3D12资源，需应用自行管理

# DXR高层结构



# DXR光线追踪管线

- ▶ 发射光线
  - 新D3D函数: `ID3D12CommandList::DispatchRays()`
  - 新Shader类型: Ray Generation Shader
  - 新HLSL函数: `TraceRay()`
- ▶ 光线的遍历与求交
  - 遍历加速结构(Acceleration Structures)
  - 新Shader类型: Intersection Shader
- ▶ 交点处理
  - 判断交点的有效性, 计算光照, 继续发射光线等
  - 新Shader类型: Closest Hit Shader, Any Hit Shader
- ▶ 无交点处理
  - 终止或继续发射光线
  - 新Shader类型: Miss Shader



# DXR光线追踪管线

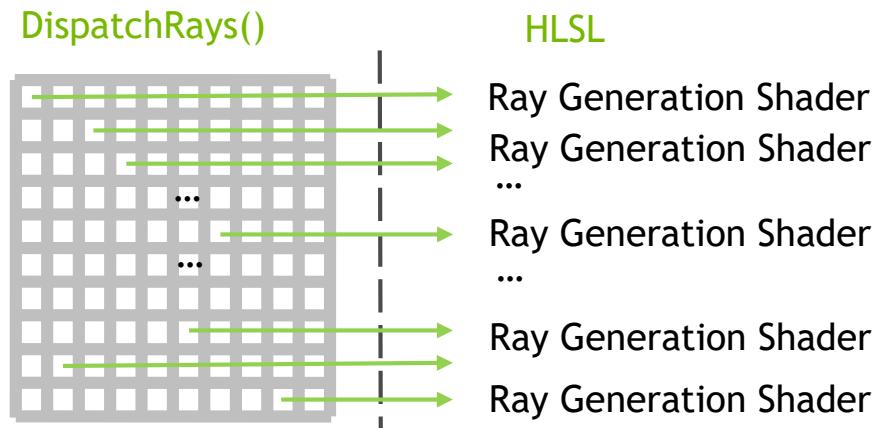
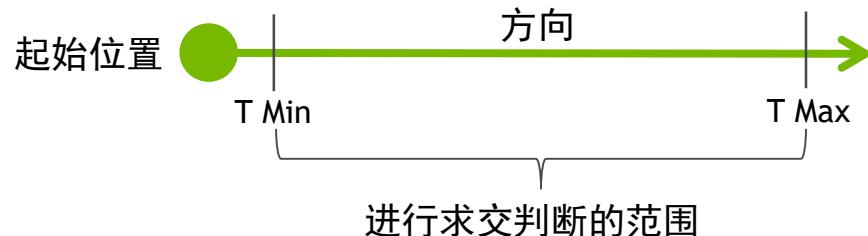
## 发射光线

### ▶ ID3D12CommandList::DispatchRays()

- 可以与Draw、Dispatch函数放在同一队列中
- 既可以放在Graphics Context中，也可以放在Compute Context中
- 声明一个线程阵列，调用ray generation shader

### ▶ HLSL函数TraceRay()

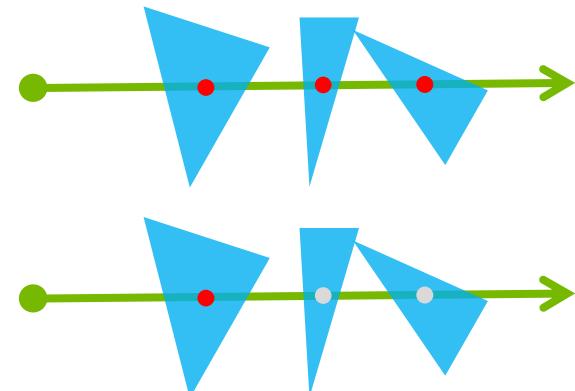
- 追踪一条射线
- 通过自定义Payload结构返回数据



# DXR光线追踪管线

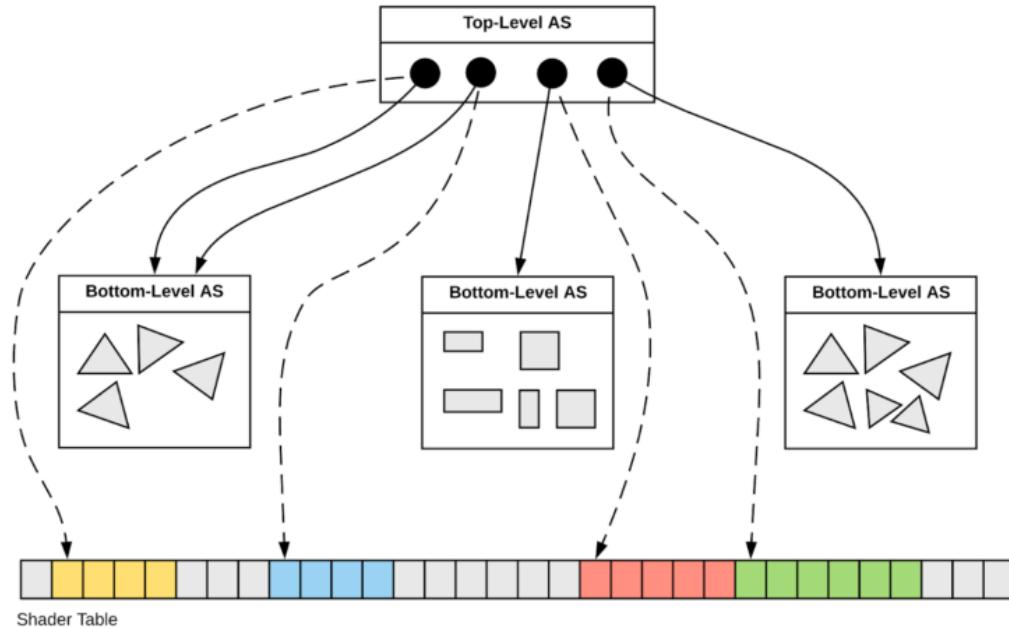
## 交点处理

- ▶ Hit Group: 由Closest Hit, Any Hit, Intersection三种Shader构成
  - 都不是必须的Shader, 可选择调用
- ▶ Intersection Shader
  - 用于自定义几何体与射线求交, 三角形与射线求交无需使用 (内置)
  - 通过调用ReportHit()报告交点, 同时传递Attributes给Hit Shaders
- ▶ Any Hit Shader
  - 射线有任一交点时被触发(不保证触发顺序)
- ▶ Closest Hit Shader
  - 针对最近的有效交点触发, 每条射线最多调用一次



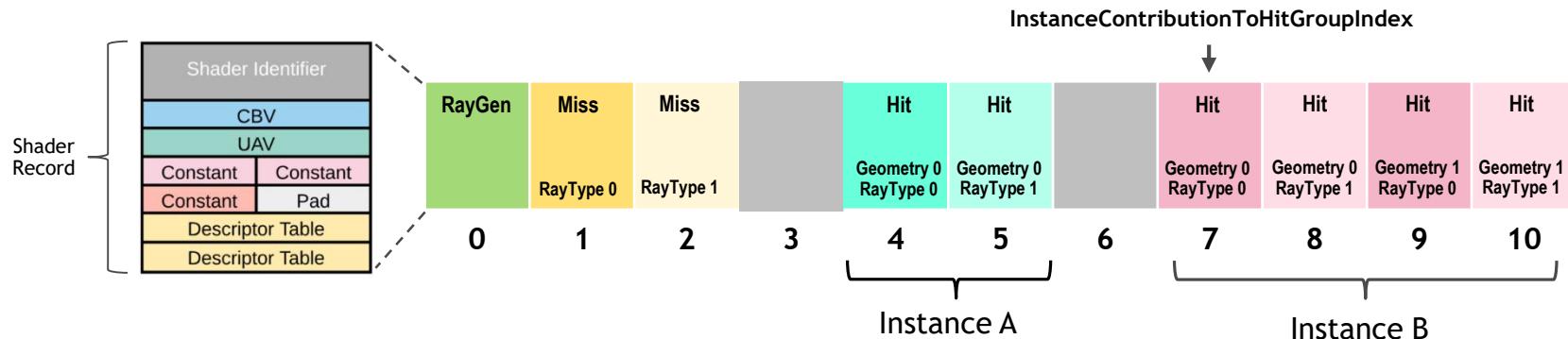
# 加速结构

- ▶ 两级层次结构
  - 底层(bottom-level):基本的几何体信息，包含三角形或包围盒数据
  - 顶层(top-level):指向底层几何体的实例、相应的资源索引和坐标变换数据
- ▶ 具体格式对应用不可见
- ▶ 由应用程序调用API显式构建/更新



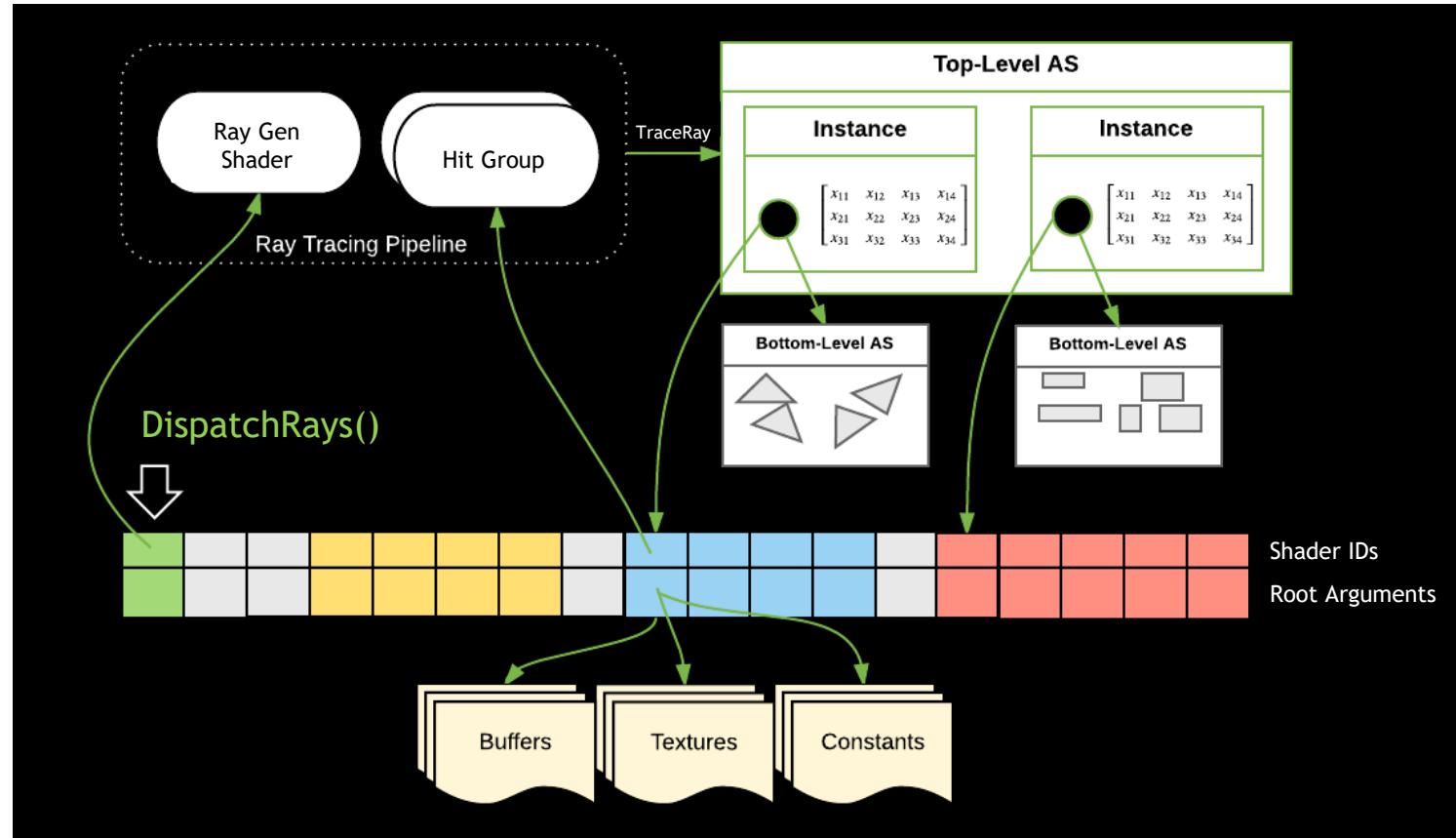
`ID3D12CommandList::BuildRaytracingAccelerationStructure(...)`

# 着色器表(Shader Table)

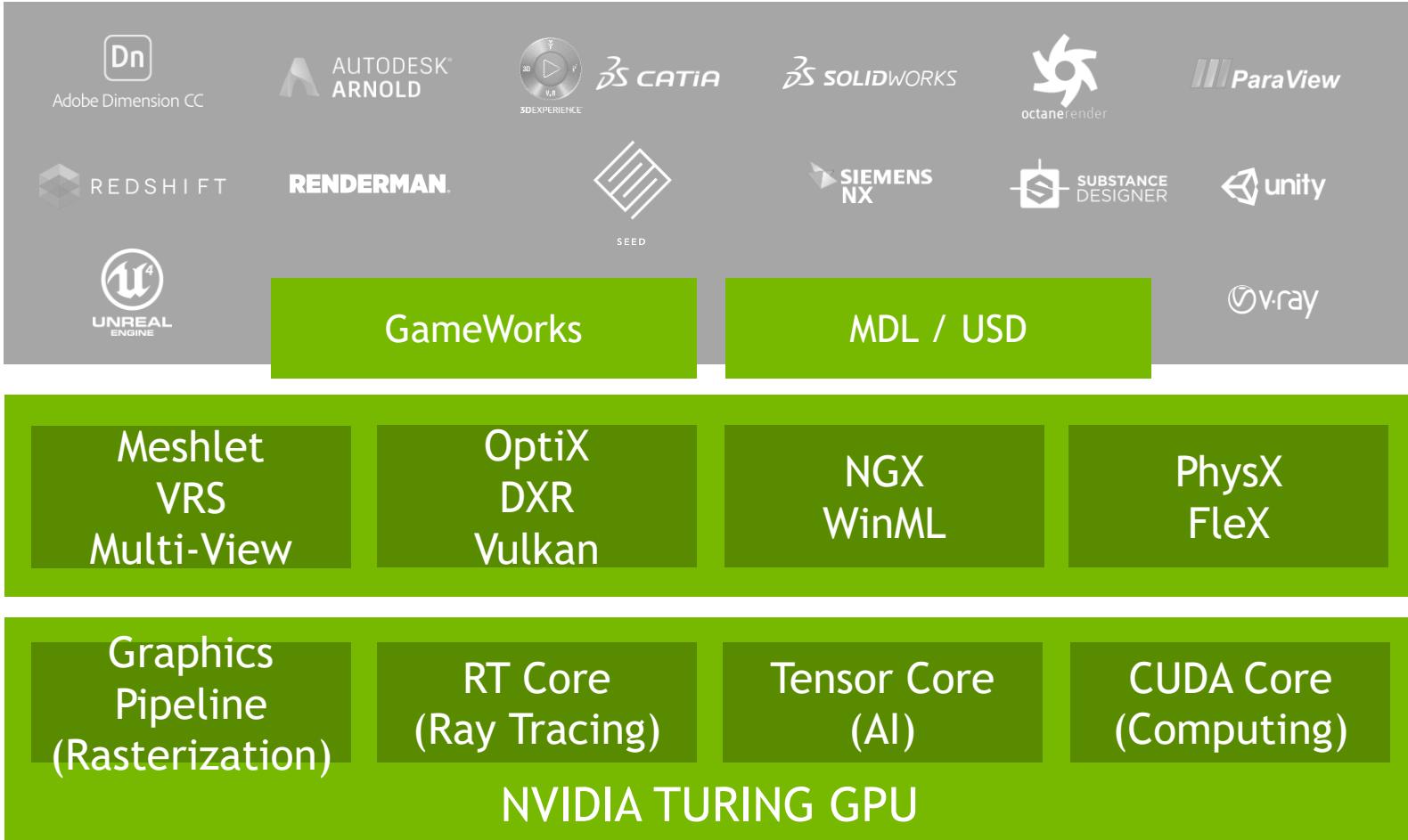


- ▶ 着色器表 (Shader Table)包含了一次光线追踪中可能用到的所有资源
- ▶ 由多条等长的记录构成，每条记录可包含ShaderID, CBV, UAV, SRV等
- ▶ 顶层加速结构中的节点指向着色器表中的对应记录

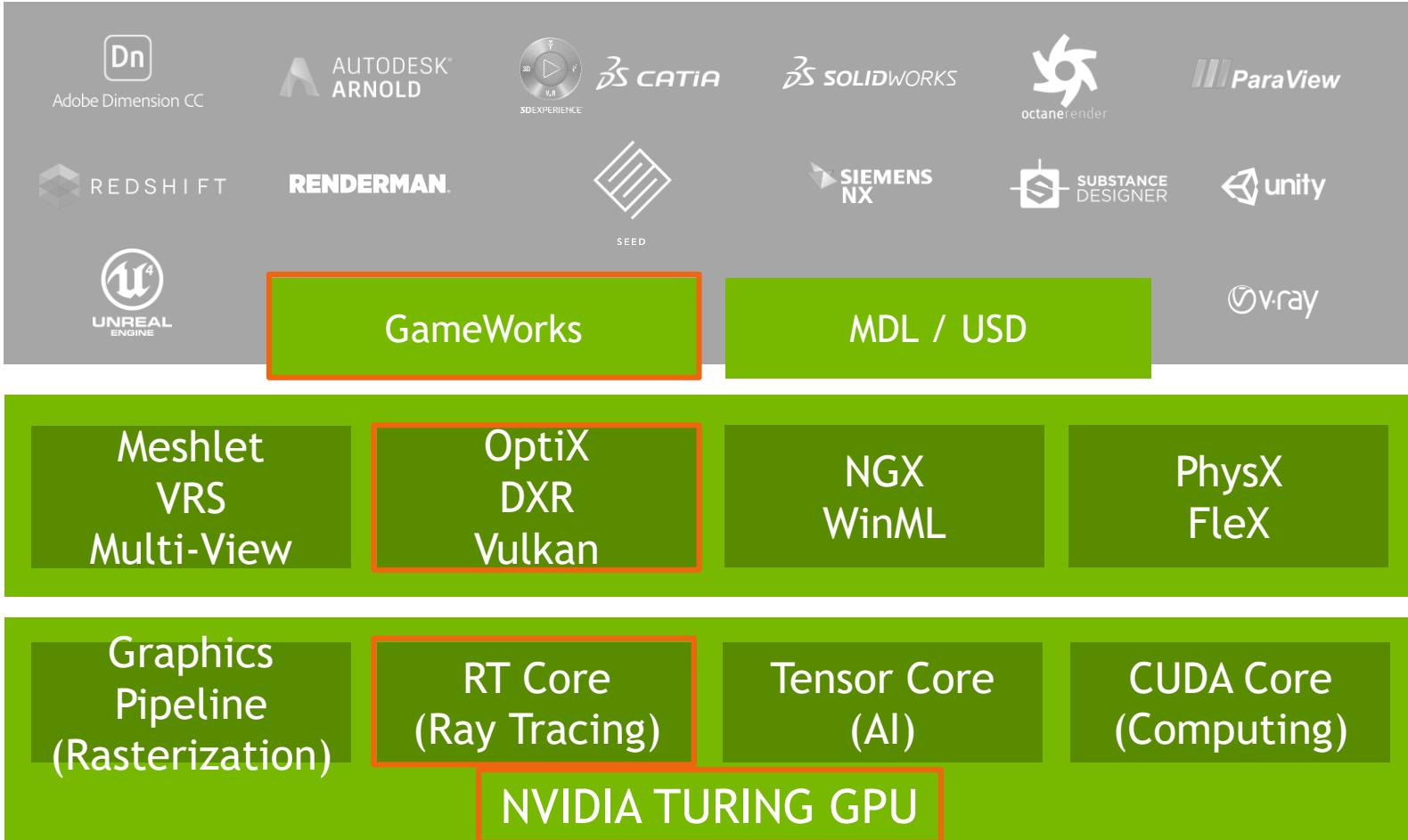
# DXR 总述



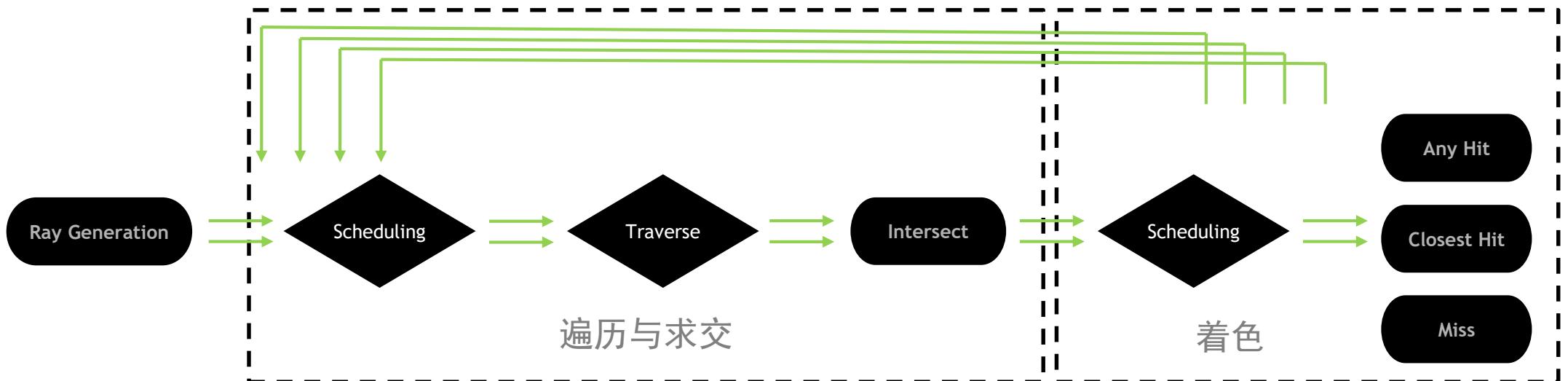
# NVIDIA RTX



# NVIDIA RTX



# RTX光线追踪构架

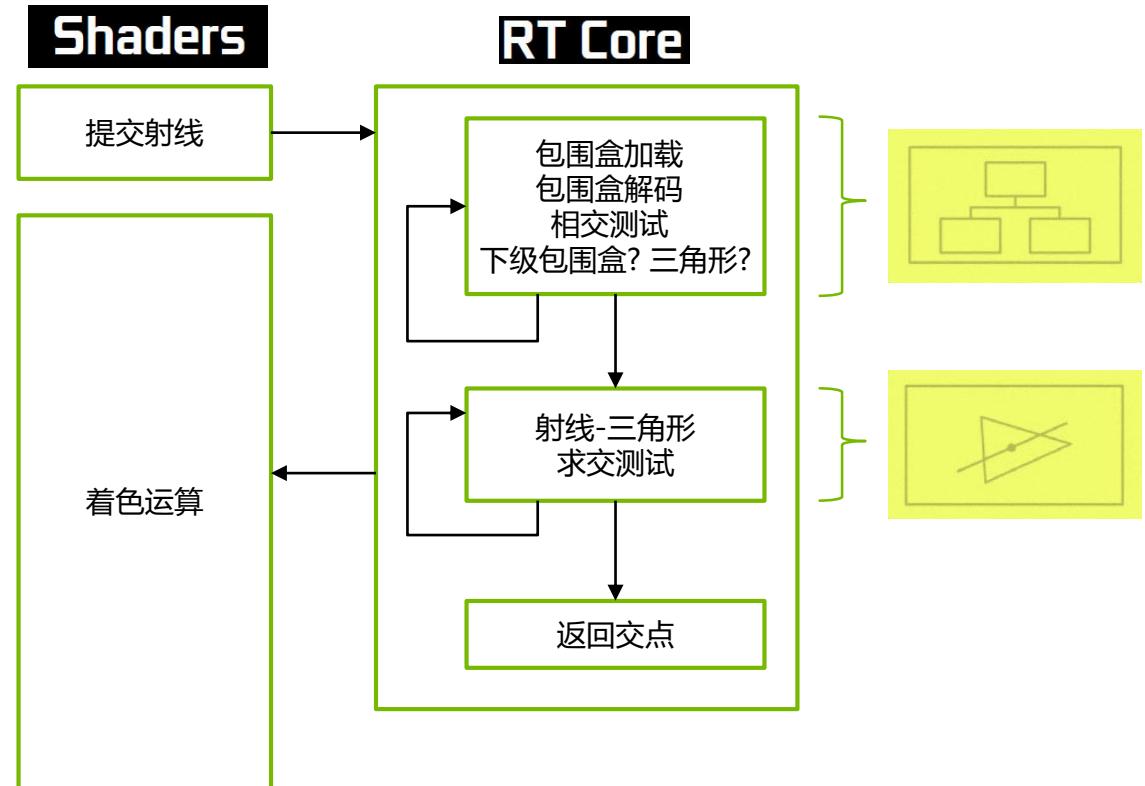


- ▶ 混合执行固定功能单元与可编程单元
- ▶ 固定功能硬件单元RT Cores: 调度 (Scheduling), 遍历 (Traverse), 光线-三角形求交
- ▶ 可编程单元Shaders: Ray Generation, Intersect, Any Hit, Closest Hit, Miss

# RTX光线追踪构架

## RT Cores

- ▶ RT Cores提供遍历和求交功能
  - 遍历加速结构：自顶向下搜索BVH
  - 射线-三角形求交：返回重心坐标和交点距离
- ▶ 调用RT Core
  - 光追管线中的Shader提交射线至RT Core输入队列
  - Shader可继续执行其它操作
  - RT Core返回结果，Shader执行进一步运算



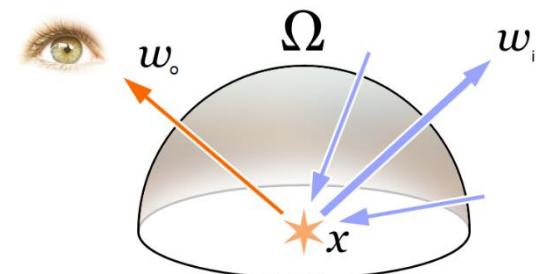


\*图片来自互联网

# 实时光线追踪

## 为什么会有噪点？

- ▶ 在求解渲染方程时，需使用蒙特卡洛采样：
  - $L(\omega_o) = \int_{\Omega} L(\omega_i)f(\omega_o, \omega_i)|\omega_i \cdot n|d\omega_i = \frac{1}{N} \sum_{i=0}^N L(\omega_i)f(\omega_o, \omega_i)|\omega_i \cdot n|/p(\omega_i)$
- ▶ 上式Estimator中的每一项都是半球域上的复杂函数
  - 入射辐亮度(Incoming radiance)  $L$ , 可见性  $V$ , BRDF  $f$ , 采样密度分布函数  $p$
- ▶ 当前硬件条件下，每像素至多几十个采样(spp/samples per pixel)
- ▶ 低采样率导致Estimator的高方差，表现为噪点



# GameWorks Ray Tracing

## ► Ray Tracing Denoiser

- 复杂的反射(RT Reflections)
- 面光源软阴影(RT Shadow)
- 环境遮挡(RT AO)

# 光线追踪的反射

## 与SSR对比

- ▶ 屏幕空间反射(SSR) 的问题
  - 无法反射到屏幕空间以外的几何体
  - 反射的高光不正确
    - 高光依赖于视角，反射光线的视角已经不同于主视角，但依然采样了主视角下的高光颜色

# 光线追踪的反射

与其他反射技术对比

- ▶ 预集成的光源探针 (light probes) / 环境贴图
  - 只适合静态场景
  - 位置放置不恰当容易产生错误
  - 粗糙度被离散化到有限的几个级别
- ▶ 平面反射
  - 仅能应用于平面
  - 需要多渲染一遍场景
  - 较难实现光泽反射(glossy)



屏幕空间反射 + 反射环境贴图



1SPP光线追踪反射（降噪后）



参照基准 (GROUND TRUTH)



1SPP光线追踪阴影（降噪前）

# 光线追踪的反射 降噪

- ▶ 仅对入射辐亮度 (Incoming Radiance) 进行降噪
  - $L(\omega_o) = \int_{\Omega} L(\omega_i) f(\omega_o, \omega_i) |\omega_i \cdot n| d\omega_i \approx \underbrace{\int_{\Omega} L(\omega_i) d\omega_i}_{\text{降噪!}} \underbrace{\int_{\Omega} f(\omega_o, \omega_i) |\omega_i \cdot n| d\omega_i}_{\text{预积分}}$
- ▶ BRDF 及高光被预集成，不参与降噪，所以不会糊掉
- ▶ 各项异性的交叉双边过滤器
- ▶ 使用历史帧数据改善质量及稳定性（类似 TAA）

# 光线追踪的面光源软阴影

为什么要光线追踪？

▶ 高质量的软阴影效果

- 大面积复杂光源产生的Contact Hardening阴影
- 相比角色胶囊阴影，几何细节更加精确
- 相比距离场阴影，支持非刚体运动（如蒙皮）
- 兼容其他解析面积光照算法

▶ 消除阴影贴图技术带来的瑕疵：

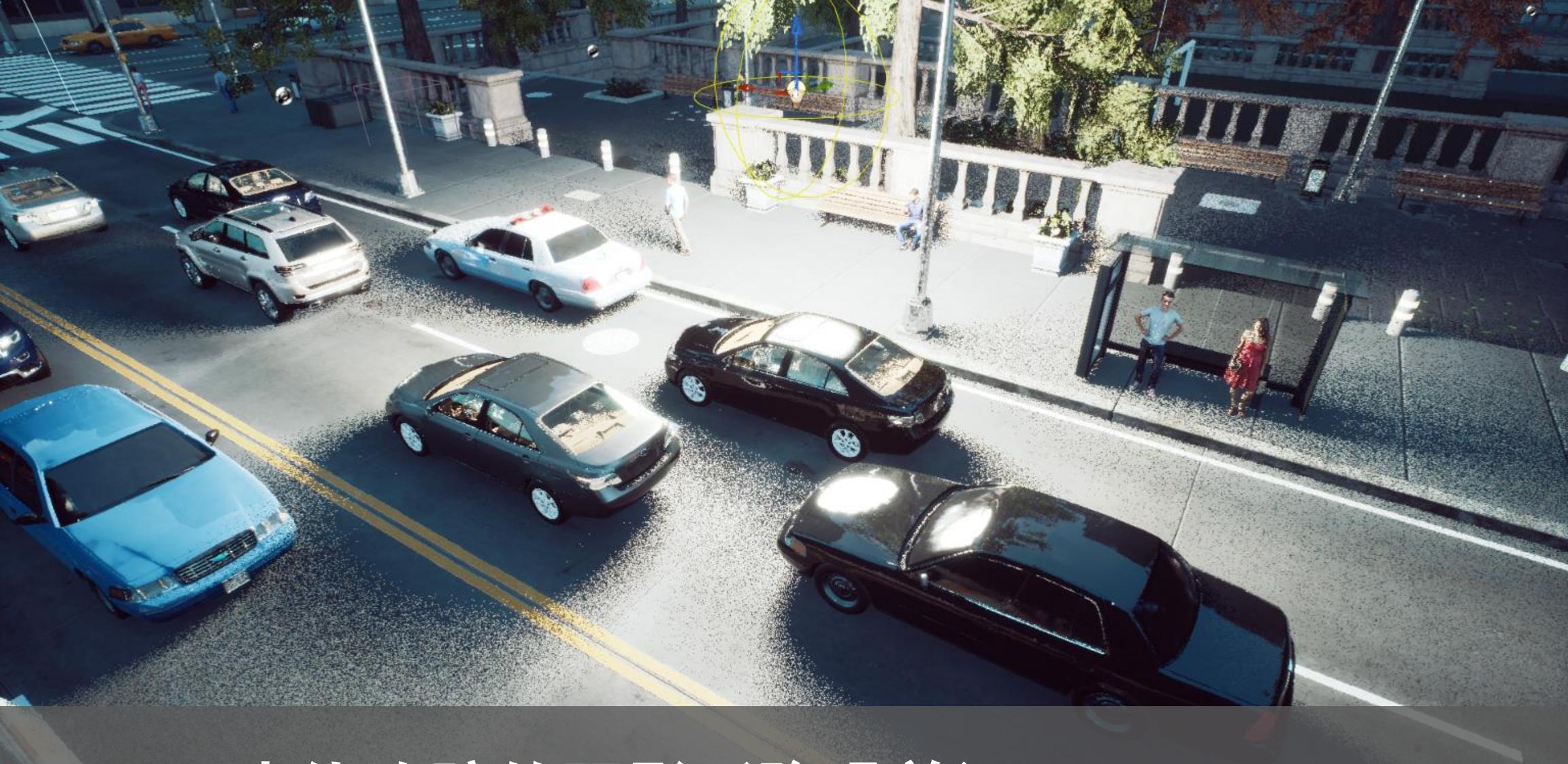
- 阴影锯齿，粉刺 (Acne)，浮空 (Peter Panning)，CSM接缝等



参照基准 (GROUND TRUTH)



1SPP光线追踪的阴影（降噪后）



1SPP光线追踪的阴影（降噪前）



阴影贴图 (SHADOW MAPPING)

# 光线追踪的面光源软阴影

## 降噪

- ▶ 仅对可见性项  $V(\omega_i)$  降噪

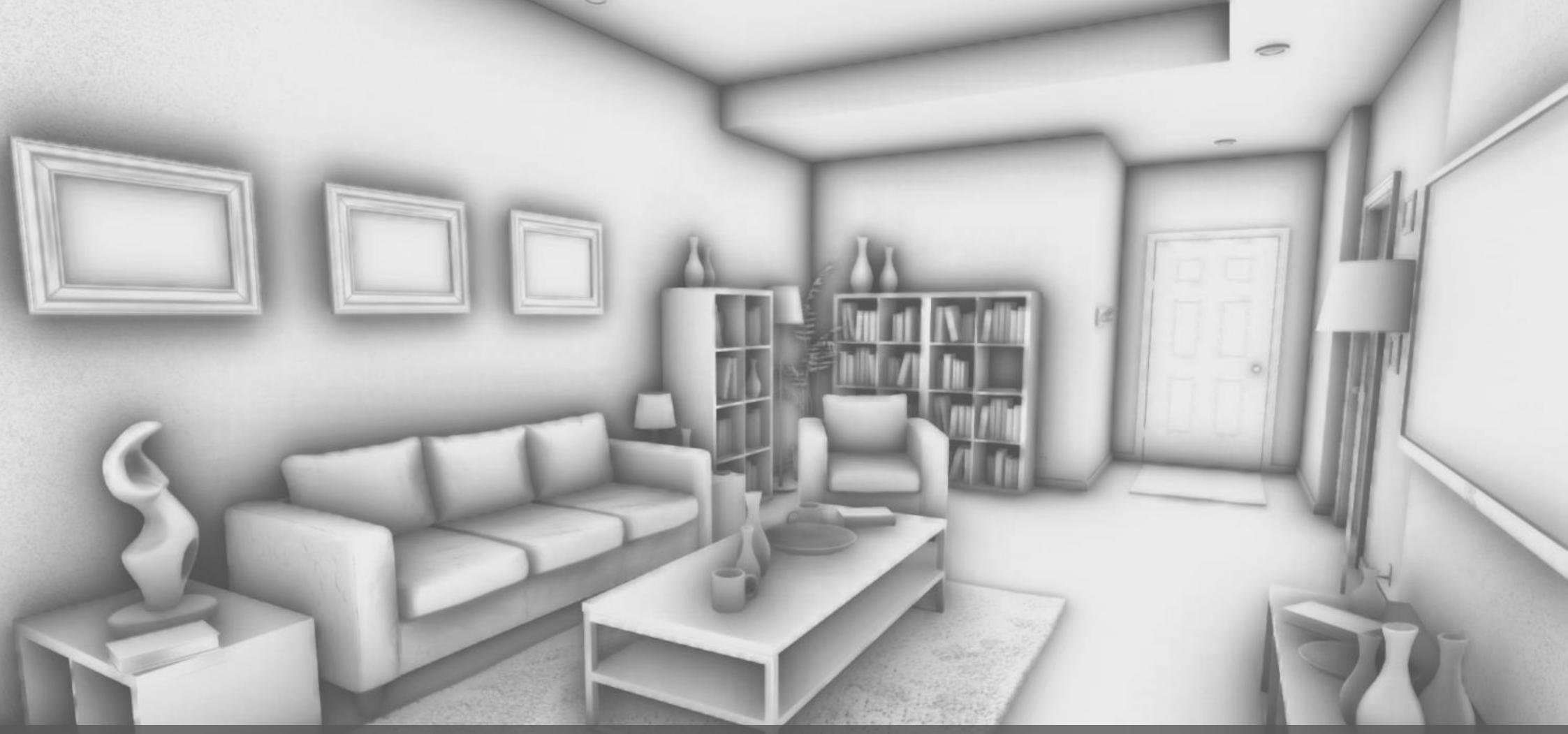
$$\circ \quad L(\omega_o) = \int_{\Omega} f(\omega_o, \omega_i) L_d(\omega_i) V(\omega_i) |\omega_i \cdot n| d\omega_i \approx \int_{\Omega} \underbrace{V(\omega_i) d\omega_i}_{\text{降噪!}} \underbrace{\int_{\Omega} f(\omega_o, \omega_i) L_d(\omega_i) |\omega_i \cdot n| d\omega_i}_{\text{解析逼近}}$$

- ▶ 可分离交叉双边过滤器 (Separated cross bilateral filter), 半径及权重可变
- ▶ 辅助信息: 交点距离, 场景深度、法线、光源大小及方向
- ▶ 使用历史帧数据改善质量及稳定性 (类似TAA)
- ▶ 不同的光源类型使用不同的降噪方案
- ▶ 每个光源单独降噪

# 光线追踪的环境遮挡

为什么要光线追踪？

- ▶ SSAO的问题：
  - 在深度不连续的地方产生“暗光晕”
  - 屏幕空间以外的物体无法产生遮挡信息
- ▶ RTAO质量更高，更加真实



屏幕空间环境遮挡(SSAO)



参照基准 (GROUND TRUTH)



1SPP光线追踪的环境遮挡（降噪后）



1SPP光线追踪环境遮挡（降噪前）

# 光线追踪的环境遮挡

## 降噪

- ▶ 可分离交叉双边过滤器
  - 根据交点距离(Hit T)来估算卷积核尺寸
- ▶ 开放区域使用较大的卷积核, 接触区域 (Contact Region) 使用较小的尺寸
  - 开放区域中可见性变化较慢, 共享范围可以更广
  - 保留接触区域的暗影(Contact Darkening)
- ▶ 只需2-4 spp, 即可保留接触区域的遮挡细节

# 内容制作

- ▶ 引擎内置的路径追踪及AI降噪
  - 结果可以作为实时渲染的参照基准(Ground Truth)
    - ✓ 材质及灯光的调校、场景亮度控制等
    - ✓ 渲染算法的正确性验证
  - 可直接用于光照贴图的烘焙
    - ✓ 不用导出场景到外部烘焙工具
    - ✓ 即时预览模式或批量生产模式
  - 在DXR上很容易实现
  - RTX提供了高效的运行基础

**Lightmap Baking Preview  
- Directional Light**

# 最佳实践

## ▶ 加速结构

- 静态物体合并构建，且只构建一次
- 动态物体能更新的就不要重建
- 控制每帧构建的数量，减少不必要的构建
- 多线程构建
- BuildAS调用之间不要加Barrier，放到后面批量处理
- 使用Instance
- 合理使用标记

# 最佳实践

- ▶ 减少Trace光线的数量，同时也要减少Shading的复杂度
- ▶ 尽量使用三角形图元
- ▶ 注意递归深度
- ▶ 尽量减小Payload及Attributes的尺寸
- ▶ 尽量使用Opaque标记
- ▶ 尽量使用 ACCEPT\_FIRST\_HIT\_AND\_END\_SEARCH 标记
- ▶ 尽量不要使用 CULL\_BACK\_FACING\_TRIANGLES 标记
- ▶ 贴图使用MipLevel 0

# 目前的资源

- ▶ GameWorks Ray Tracing:  
<https://developer.nvidia.com/gameworks-ray-tracing>
- ▶ NVIDIA RTX: <https://developer.nvidia.com/rtx>
- ▶ DXR SDK:  
<http://forums.directxtech.com/index.php?topic=5860.0>
- ▶ 教程: <https://github.com/NVIDIAGameWorks/DxrTutorials>
- ▶ Falcor: <https://github.com/NVIDIAGameWorks/Falcor>



Thank you!

