NVIDIA GameWorks中的高级光影与抗锯齿特效

曹家音 – 内容技术工程师
jecao@nvidia.com

NVIDIA

摘要

- VXAO
- HRTS
- TXAA 3.0
- ANSEL
- HDR Display

基于体素化的环境光遮蔽算法

- VXGI是NVIDIA最先进的实时全局光照解决方案
  - 每帧更新几何体的体素化信息
  - 计算高度近似、真实的漫反射和高光反射的全局光照
  - 对于主流的游戏开发而言，相对来说开销比较大
- VXAO是VXGI的一种特殊模式
  - 抛弃了所有与光照有关的信息，只保留遮蔽信息
  - 更高的渲染效率
  - 集成VXAO相对来说容易很多

基于屏幕空间AO算法的缺陷

- 视角相关
  - 被遮挡的物体不会对AO有任何贡献
  - 屏幕边缘会出现错误的AO结果
- 局限性
  - 只有近距离的遮挡物才对AO有贡献
- 模糊
  - 为每一个像素计算AO开销太大，一般都用半屏幕渲染（1/4计算量）

为什么VXAO要由于SSAO

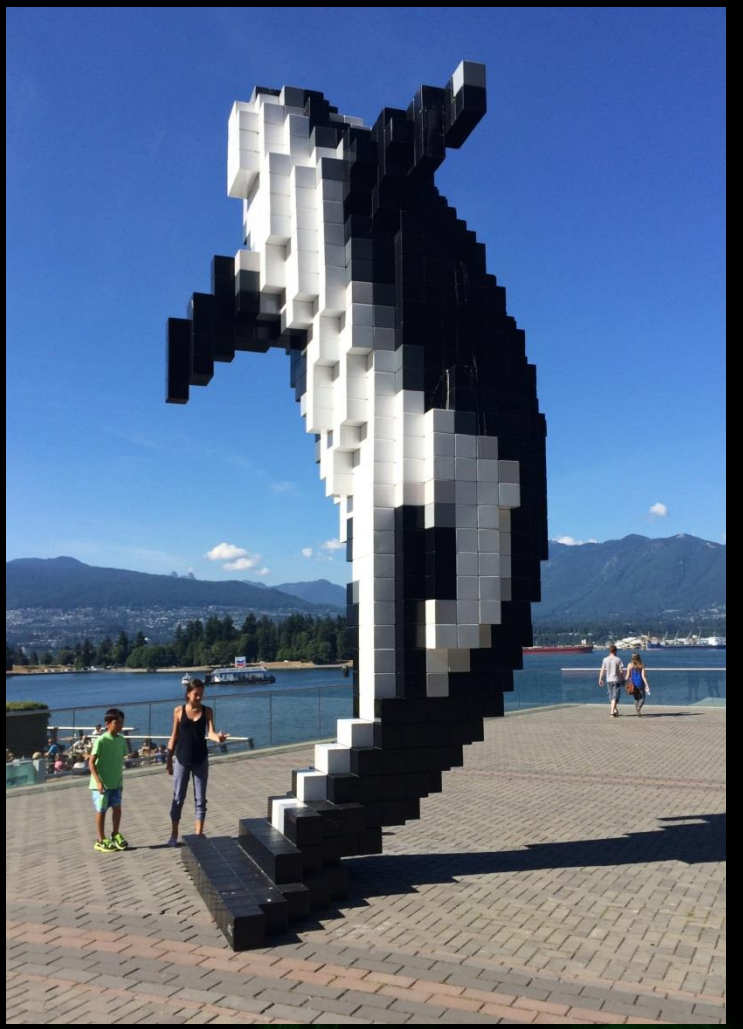
- 没有任何之前提到的问题
- 依赖于世界空间的解决方案
  - 被遮挡的物体同样对于AO有贡献
  - 即使在视角后面的物体也会有贡献（需要更宽松的裁剪算法）
- 使用基于体素化的跟踪方案
  - 投影物体可以与被投影物有一定距离
  - 相对远的信息会从较粗糙的体素信息中获取
  - 改变AO的影响距离对于算法的性能改变不大

基于体素化的锥形跟踪简介

- 把场景几何信息体素化，存储其遮蔽信息
  - 降低体素化信息的采样频率
  - 体素化的内容为我们提供了计算环境光遮蔽的必要信息
- 通过锥形跟踪来计算收集环境光遮蔽信息

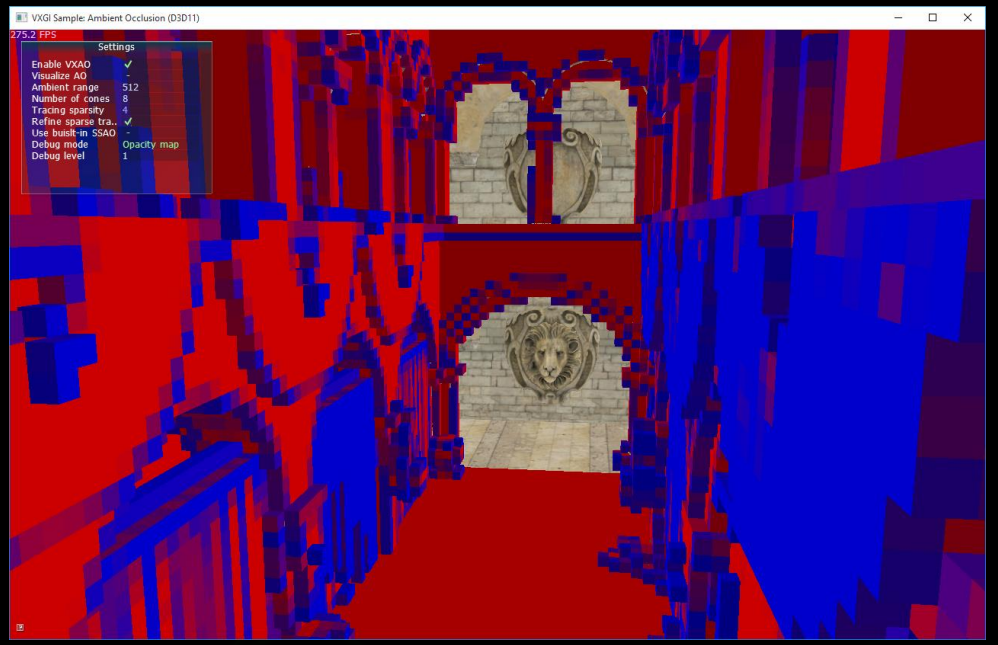
体素化

- 把几何体转换为体素的过程
- 可以理解为三维空间的‘光栅化’



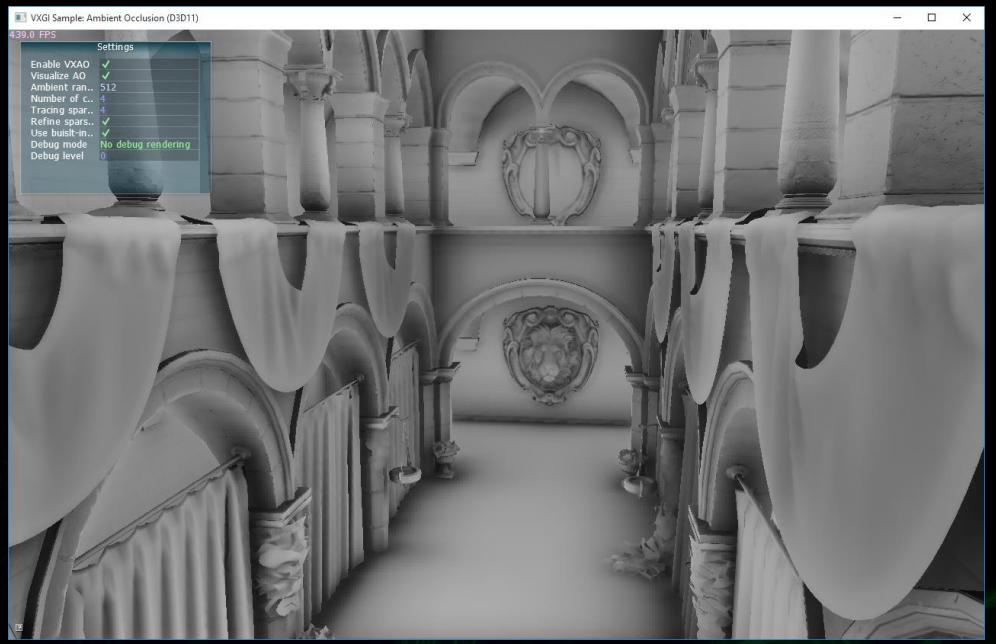
A binary voxel representation of an object with color information

实际Demo中的VXAO



- 体素化

- VXAO计算结果



Demo: HBAO+计算结果

Demo: VXAO计算结果

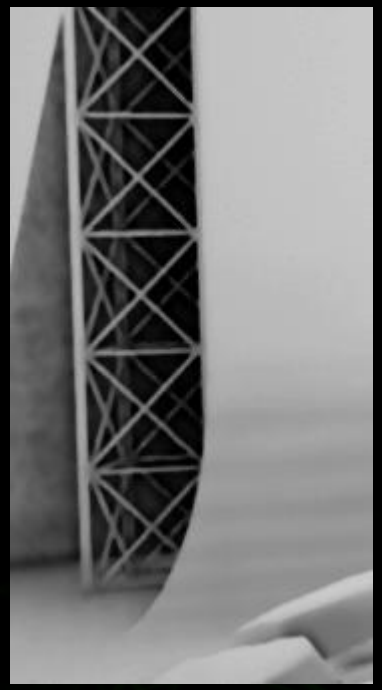
图像质量对比 #1

HBAO+

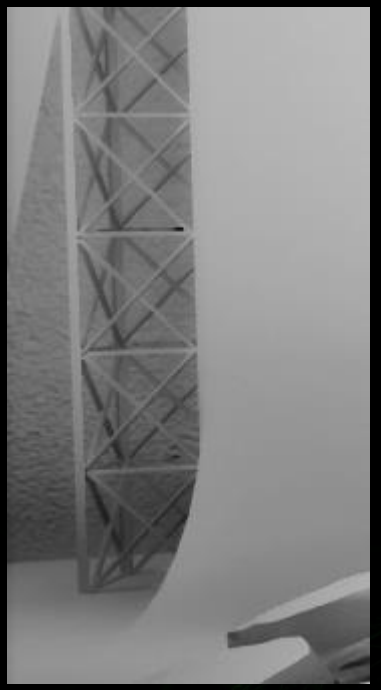
VXAO

- 坦克底下的地面
- 履带底部
- 过于模糊

图像质量对比 #2



HBAO+



VXAO

HBAO+

VXAO

HBAO+（有颜色）

VXAO（有颜色）

处理动态场景

- 体素化信息的创建和更新开销很大？
  - 错！
  - GTX 980在一个含有30万个三角形的场景中进行体素化，只需要1毫秒
- 大部分的体素化信息可以跨帧使用
  - 传入有更新的几何体的轴对齐包围盒
  - 只有这些几何体会被重新体素化，其它的信息不需要进行额外处理

VXAO性能（GTX980）

Pass	Time	Conditions
VXAO: 场景体素化	1.0 ms	30万三角形 128^3 clip-map, 5 LODs
VXAO: 体素化后处理	1.4 ms	
VXAO: 锥形追踪, 差值	1.6 ms	1920x1080
VXAO 总体	4.0 ms	
HBAO+ 总体	1.2 ms	1920x1080 模糊半径为4, 有法线通道

在Maxwell GPU平台上，HBAO+相对于VXAO快2-4倍, 但是质量差距很大

局部更新的性能改变

- 对于大多数的静态场景而言，体素化和后处理的开销会大幅度减少

体素化几何体比例	30%	40%	63%	83%	100%
体素化开销, ms	0.23	0.28	0.45	0.63	1.01
体素化后处理开销, ms	1.02	1.06	1.08	1.22	1.49
完整体素化的时间开销	50%	54%	61%	74%	100%

对于完全不局部更新的情况来说，一些较大的墙体和地面会有很大开销
实际的数据往往要更依赖于具体的场景信息（这个测试场景是Sponza Atrium）

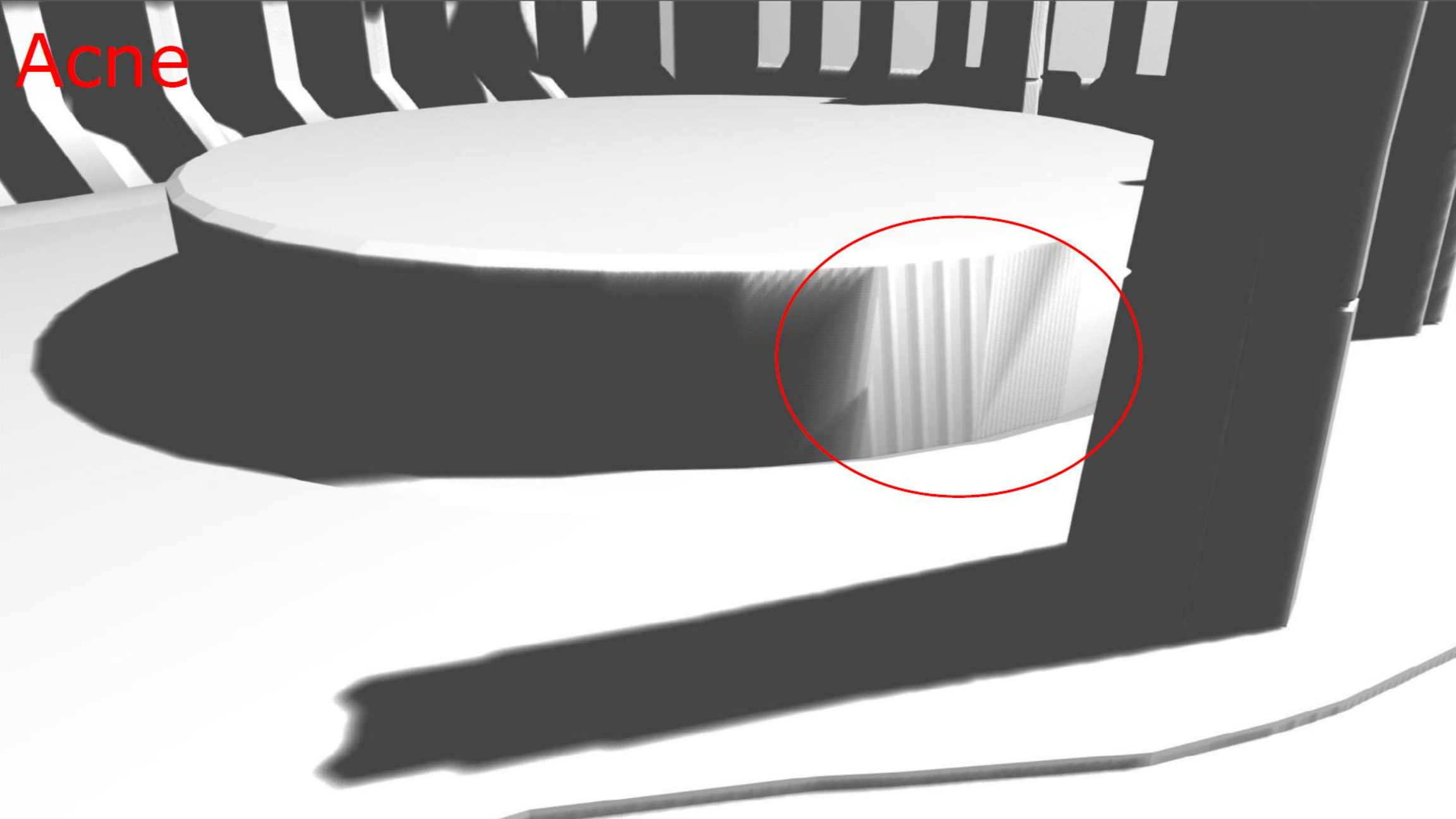
混合式光线跟踪阴影

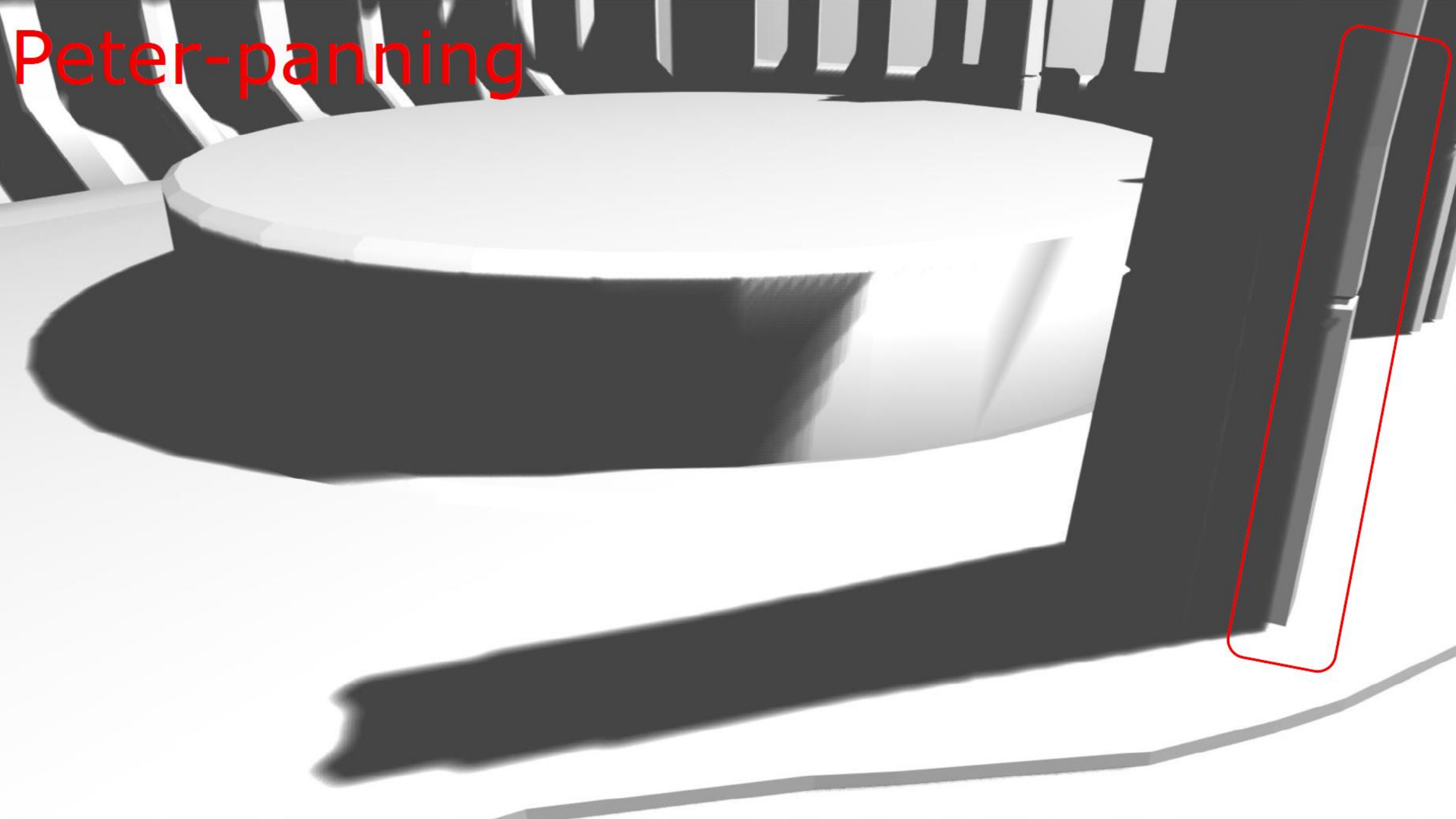
- 结合了光线跟踪的高级Shadow Map算法
- 优势
  - 精确到像素级别的阴影效果
  - 软阴影
  - HRTS会在两者之间自动混合

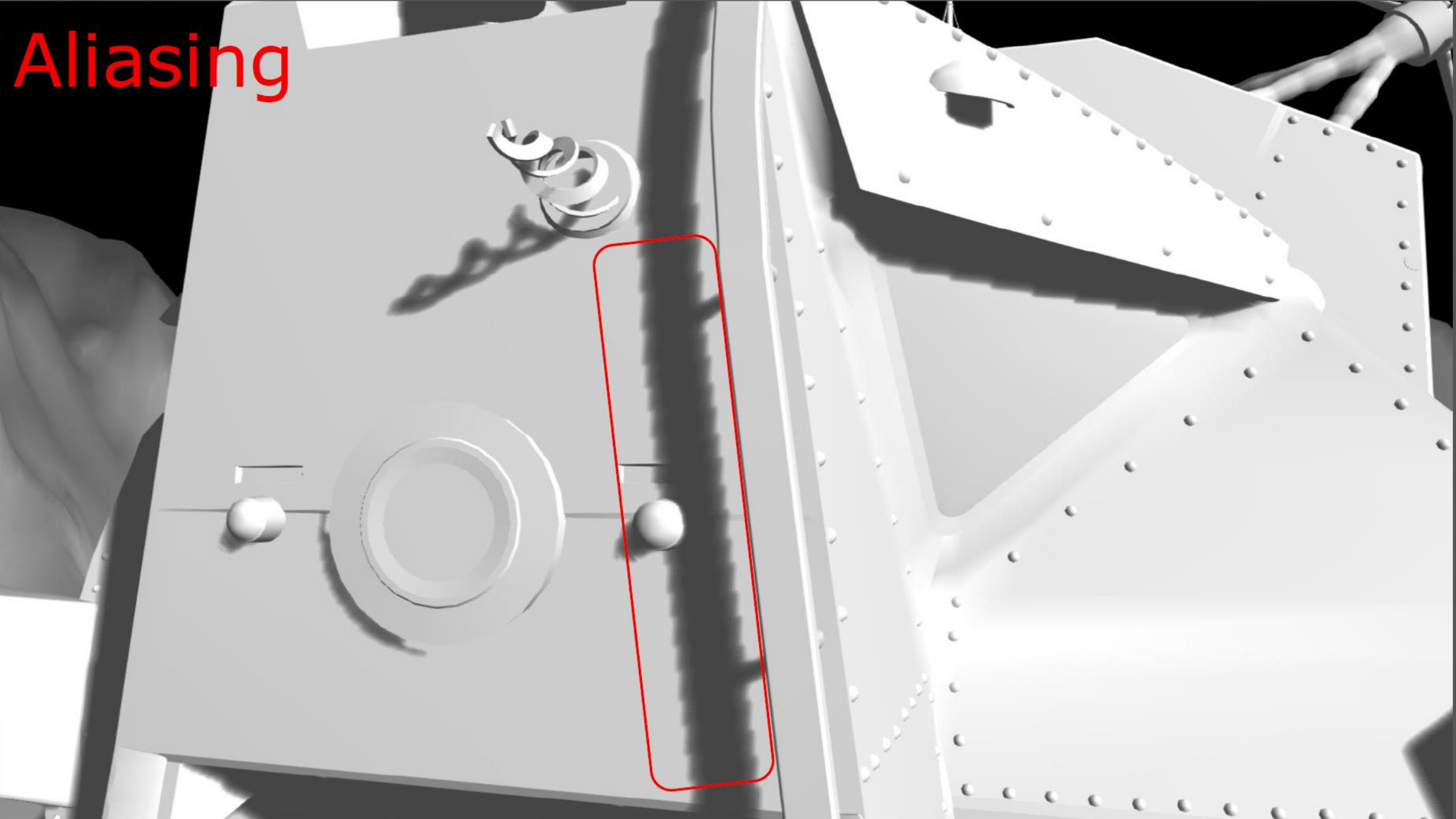
Shadow Map中的瑕疵

- 自阴影
- Peter-panning
- 锯齿

- 调节好所有细节需要很大的工作量…


Acne


Peter-panning

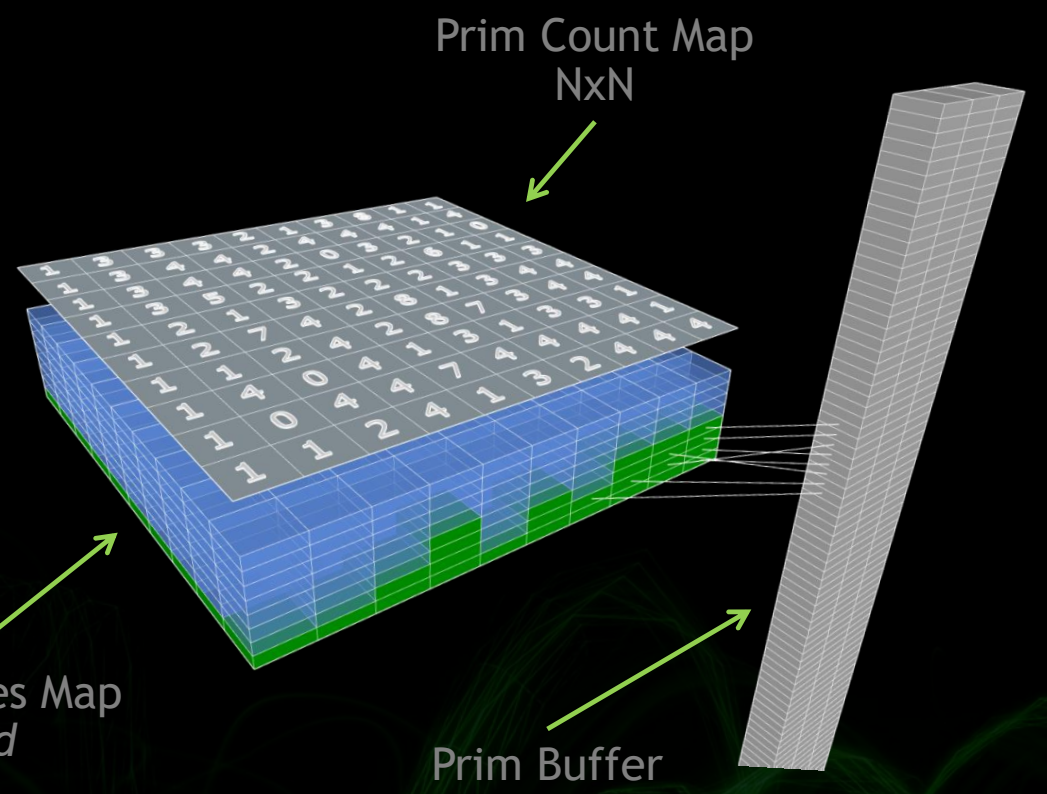

Aliasing

传统的空间划分结构

- 传统的解决方案
  - KD树
  - Bounding Volume Hierarchy
  - 均匀网格
- 实用性较差
  - 无论是重建和更新开销都非常大
  - 树结构的遍历非常慢

光线跟踪阴影的创建

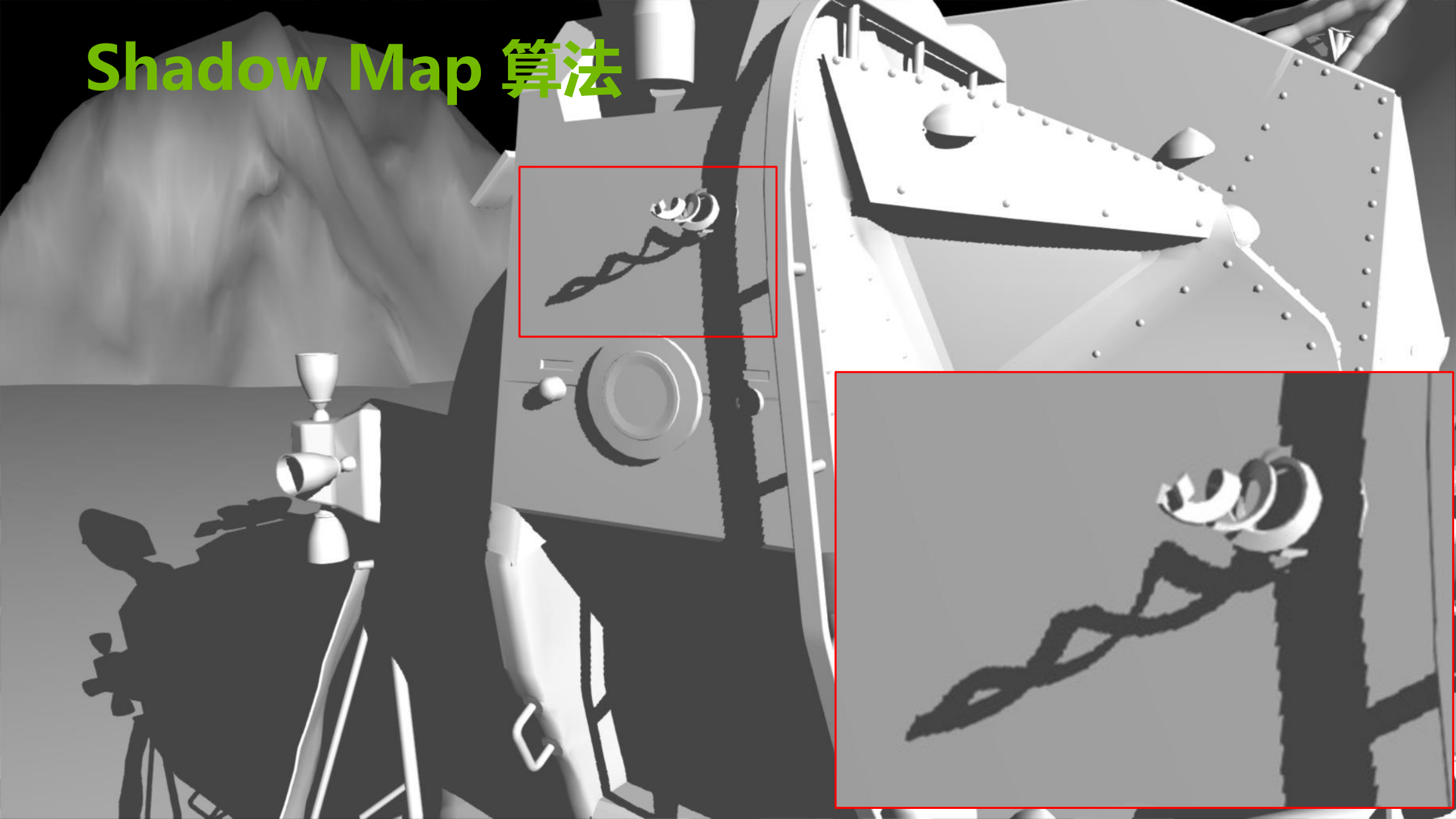
- 图元缓冲 – 三角形顶点数据
- 图元索引缓冲 – 图元缓冲中的三角形索引
- 图元数量缓冲 – 每个像素被多少图元所覆盖
- 在后续的Pass中进行光线跟踪计算



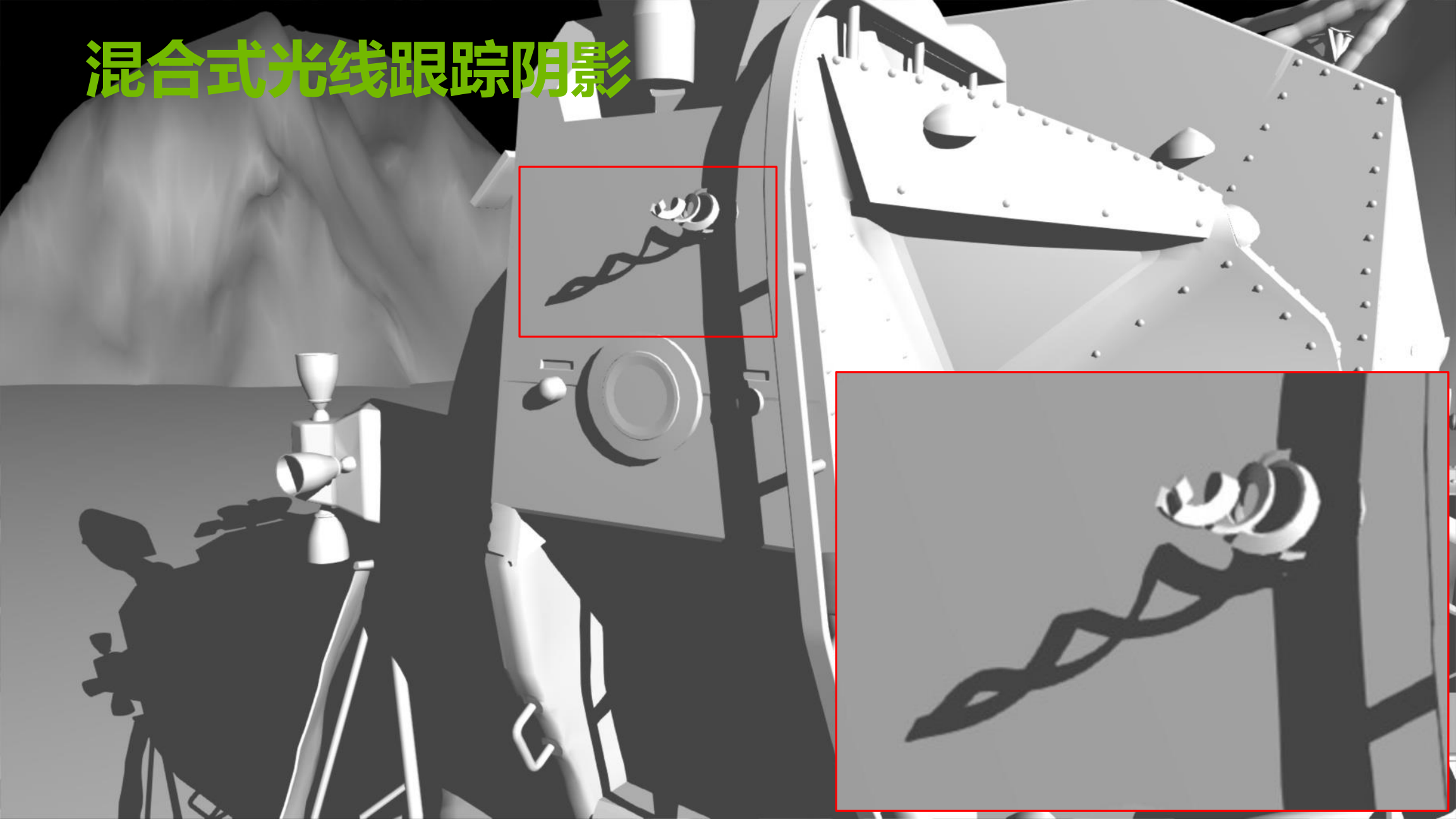
保守光栅化

- 硬件支持
  - DX12
  - NVAPI
- 软件模拟算法
  - http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter42.html

Shadow Map 算法



混合式光线跟踪阴影



混合式光线跟踪阴影

- 结合传统的阴影算法与光线跟踪阴影算法
- 使用类似PCSS等类似的高级过滤算法
- 根据投影物体的距离确定插值系数
- 当距离趋近与0的时候，光线跟踪阴影比重最大

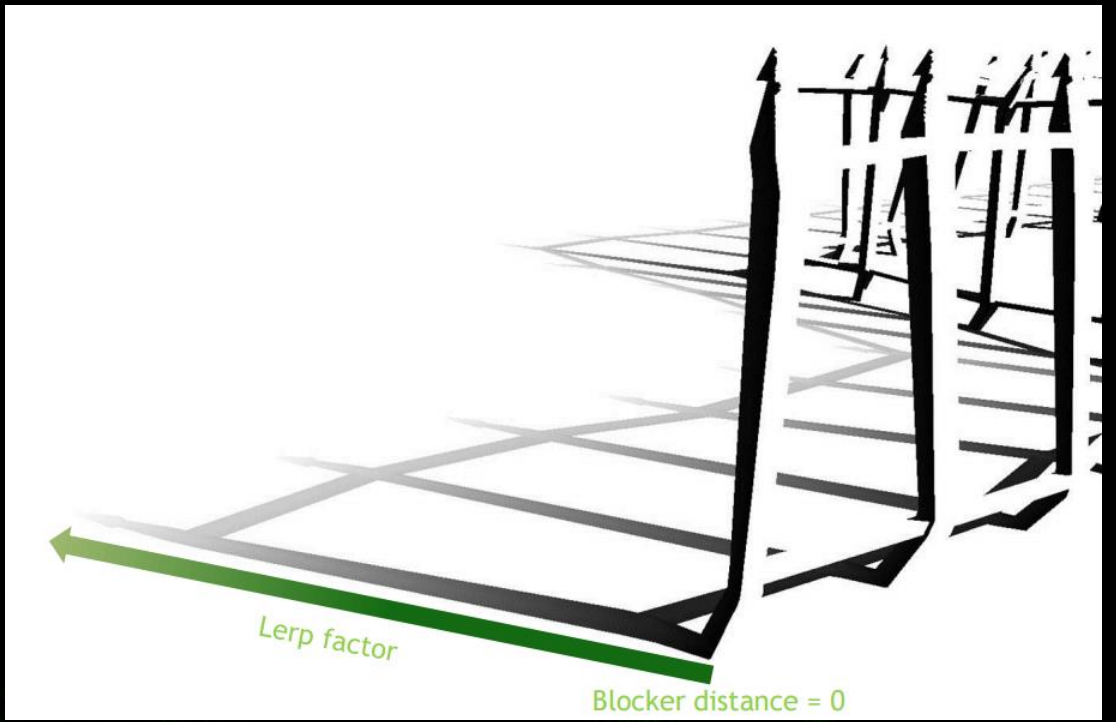
插值系数

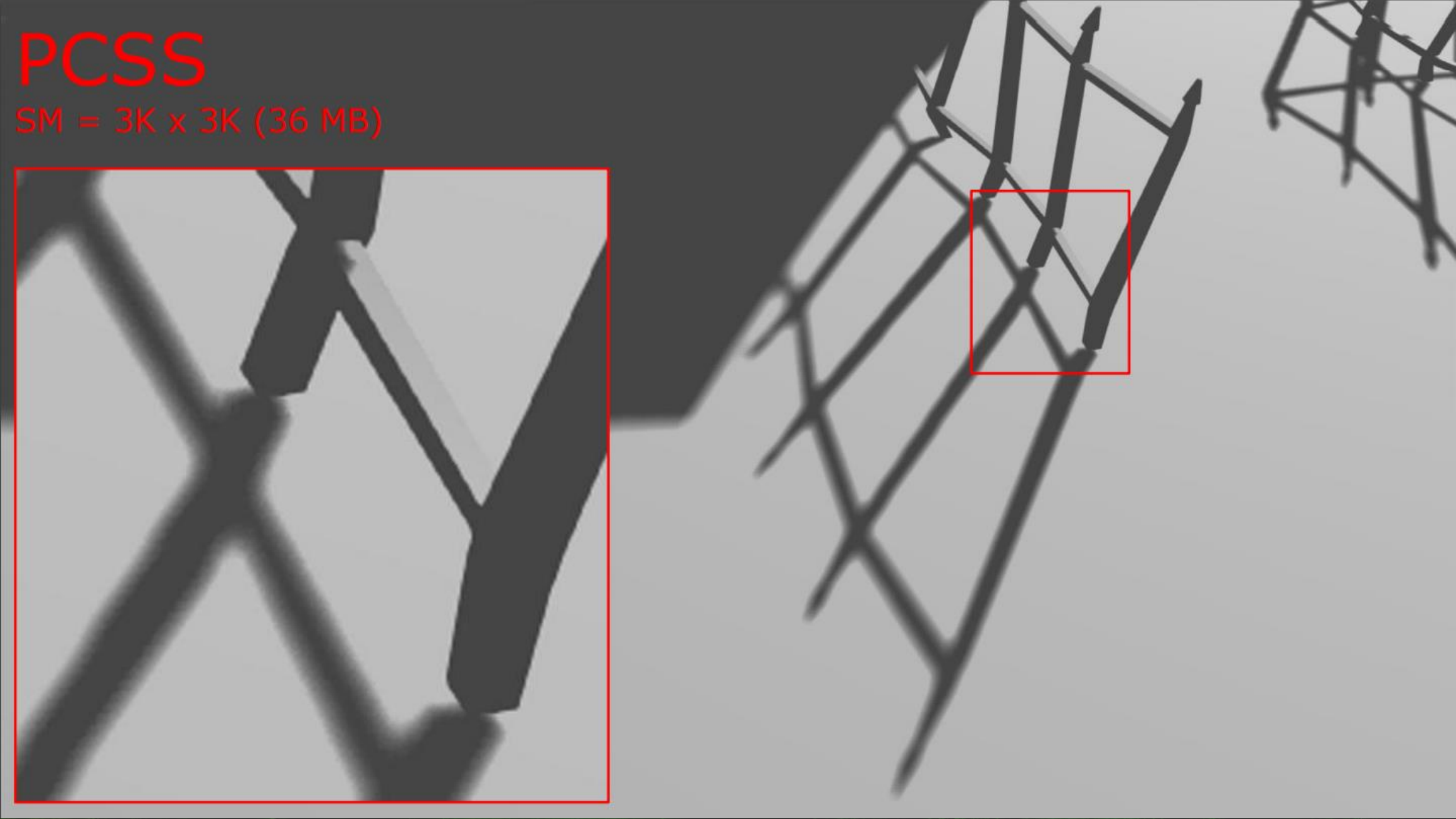
- L = saturate(BD / WSS * PHS)

- L: 插值系数
- BD: 遮挡距离（从光线源头开始）
- WSS: 世界空间中的缩放系数
- PHS: 理想的‘硬’阴影权重

- FS = lerp(RTS, PCSS, L)

- FS: 最终的阴影效果
- RTS: 光线跟踪阴影效果(0 or 1)
  PCSS: PCSS+阴影效果 (0 to 1)




PCSS
SM = 3K x 3K (36 MB)

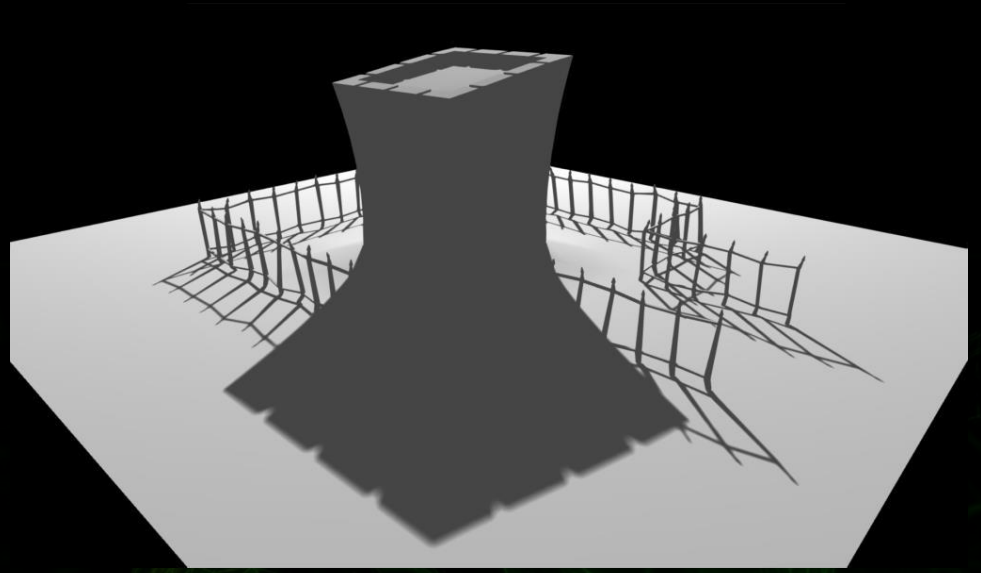
Hybrid Ray Traced + PCSS

SM = 3K x 3K (36 MB)

PM = 1K x 1K x 32 (128 MB)

混合式光线跟踪阴影性能

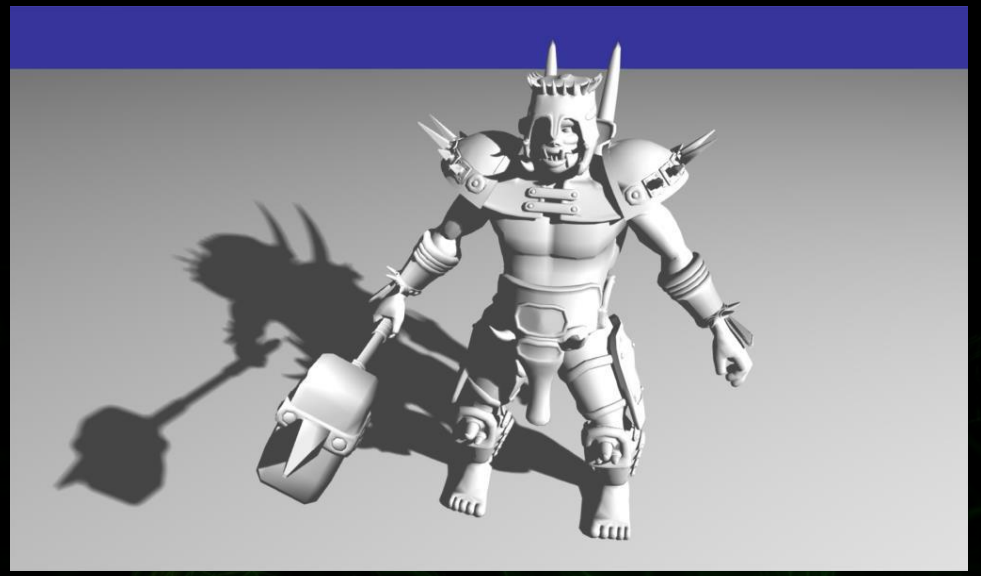
- 图元数量: ~10K
- Shadow Map: 3K x 3K (36 MB)
- Primitive Map: 1K x 1K x 32 (128 MB)
- Primitive Buffer: ~360K
- 阴影缓冲: 1920 x 1080



	GTX 980
Primitive Map + HW CR	0.4
Primitive Map + SW CR	0.5
Ray Trace	0.4
PCSS	1.3
PCSS + Ray Trace	1.8

混合式光线跟踪阴影性能

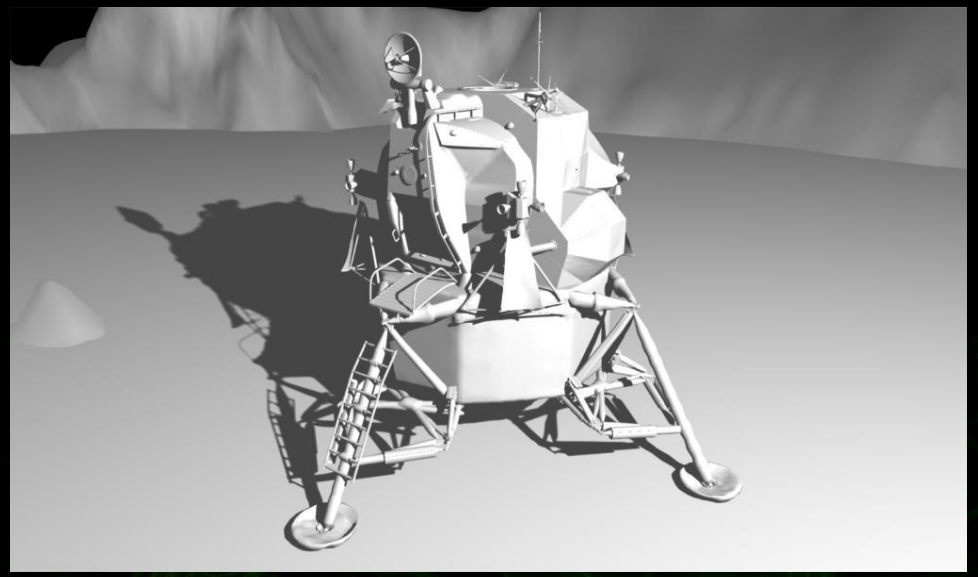
- 图元数量: ~65K
- Shadow Map: 3K x 3K (36 MB)
- Primitive Map: 1K x 1K x 64 (256 MB)
- Primitive Buffer: ~2.2 MB
- 阴影缓冲: 1920 x 1080



	GTX 980
Primitive Map + HW CR	0.5
Primitive Map + SW CR	0.7
Ray Trace	0.7
PCSS	1.3
PCSS + Ray Trace	2.8

混合式光线跟踪阴影性能

- 图元数量: ~240K
- Shadow Map: 3K x 3K (36 MB)
- Primitive Map: 1K x 1K x 64 (256 MB)
- Primitive Buffer: ~8.2 MB
- 阴影缓冲: 1920 x 1080



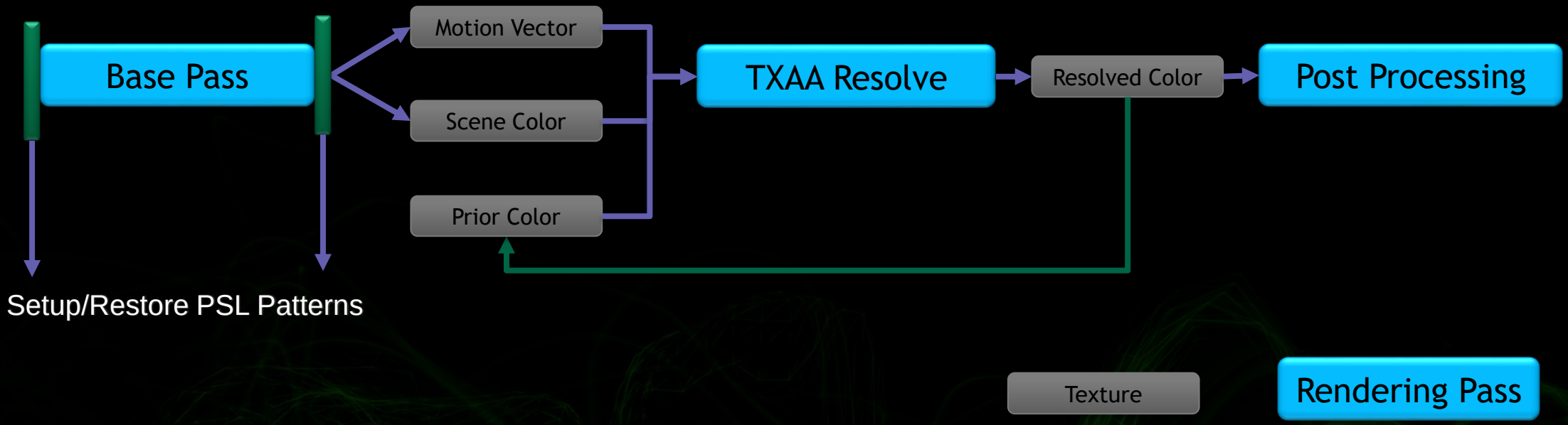
	GTX 980
Primitive Map + HW CR	3.4
Primitive Map + SW CR	4.1
Ray Trace	1.0
PCSS	1.3
PCSS + Ray Trace	3.4

次时代TXAA图形特效

- 基于时间的动态采样点位置 (PSL)
- 更简单的过滤方法
  - 不再使用高斯过滤，不再会有过度模糊的现象
  - 更快的渲染速度
- 支持所有MSAA模式
  - 支持8x MSAA
  - 支持1x MSAA 或者非MSAA模式

TXAA 3.0 核心算法

- TXAA 3.0 = [MSAA] + Temporal AA + MFAA



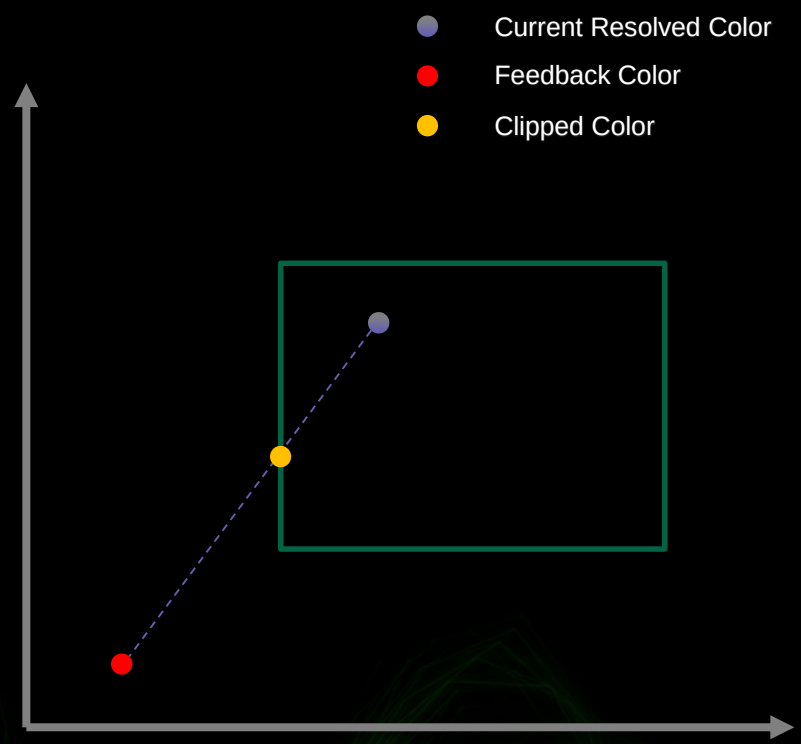
指数和

- TXAA的数学基础
  - $TXAA_n = \alpha MSAA_n + (1-\alpha)TXAA_{n-1}$
- 数学证明:
  - $MSAA_n = MSAA_{n-k}$
    - *对于静止的场景是成立的, k是采样模式的格式.*
  - $TXAA_n = \frac{\alpha}{1-(1-\alpha)^k}\Sigma_{t=0}^{k-1}(1-\alpha)^t MSAA_{n-t}$
  - $\lim_{\alpha\to 0}\left(\frac{\alpha}{1-(1-\alpha)^k}\Sigma_{t=0}^{k-1}(1-\alpha)^t MSAA_{n-t}\right) = \frac{1}{k}\Sigma_{t=0}^{k-1} MSAA_{n-t}$
    - *对于静止的场景而言，这是一个完美的结果.*

- 很遗憾，上面公式仍然有一些瑕疵:
  - α 永远不会为0
  - 如果 α 依赖于 MSAA_n的话，上述公式并不成立
  - 第一个公式对于运动场景不一定成立

颜色空间裁剪

- 为了消除‘鬼影’
- 根据周围邻居像素建立一个轴对齐的包围盒
  - **Min = min(c , n , s , w , e)**
  - **Max = max(c , n , s , w , e)**
- 上一帧的像素颜色会根据这个轴对齐包围盒裁剪



一个简单的示例，TXAA的颜色裁剪工作方式类似，不过是在YCgCo空间

可编程的采样点位置

- 图形程序员可以通过该接口控制每个采样点的位置
- 用最小的开销，最大程度的提高了抗锯齿的效果

- 抗锯齿的效果依赖于采样点模式的数量，一般来说4个或者8个表现已经足够了

必要的时候需要禁止PSL

- 生成Shadow Map
- 生成反射贴图
- 渲染UI
- 后处理渲染

TXAA 3.0 集成

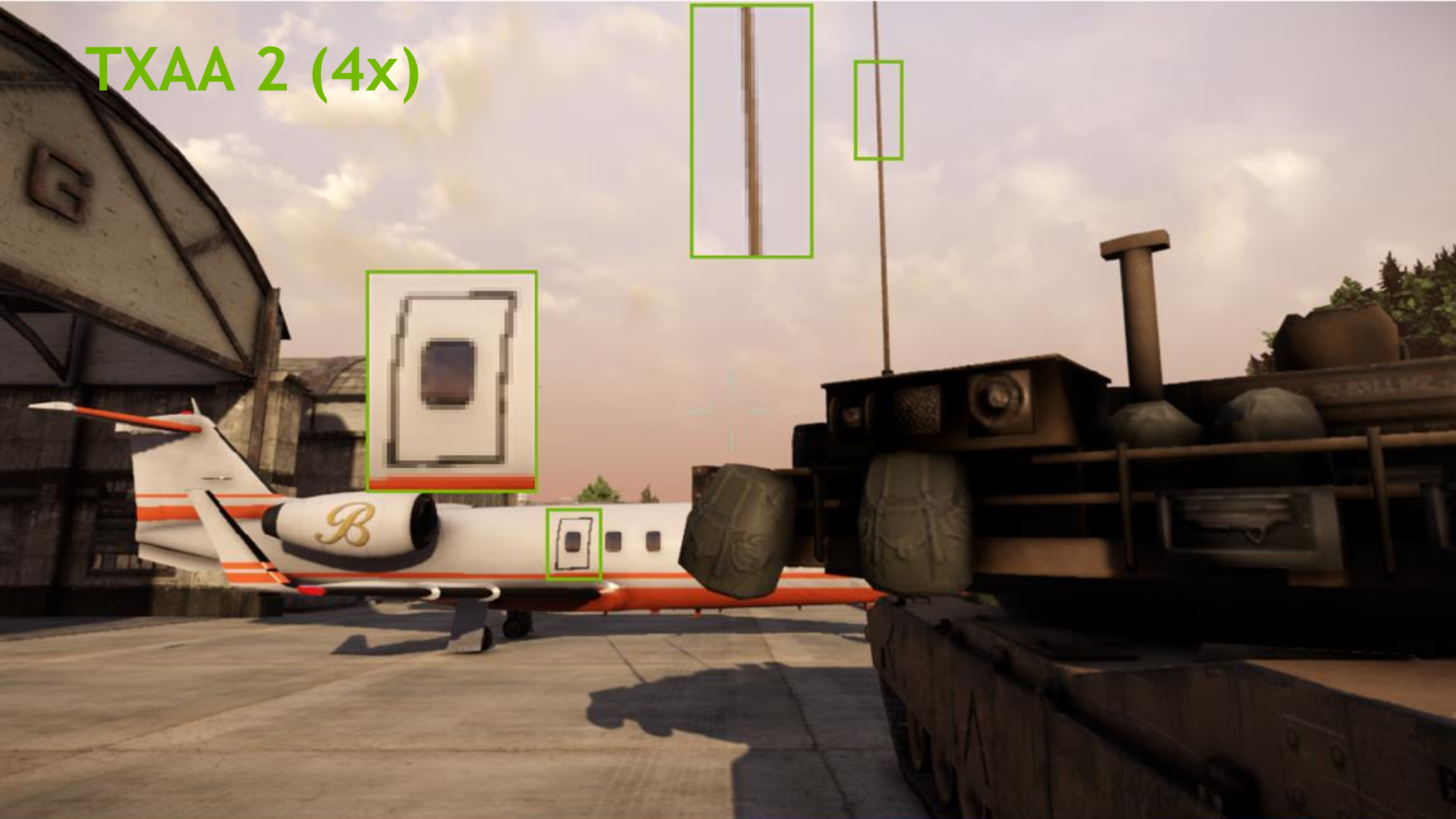
- 集成TXAA相对比较容易，只有三个接口:
  - GFSDK_TXAA_DX11_InitializeContext(&txaaContext, device)
    - 初始化TXAA3.0, 在Kepler之前的硬件上会失败
  - GFSDK_TXAA_DX11_ResolveFromMotionVectors(&resolveParameters, &motion)
    - 执行TXAA 3.0 Resolve过程
  - GFSDK_TXAA_DX11_ReleaseContext(&txaaContext)
    - 释放TXAA 3.0 资源

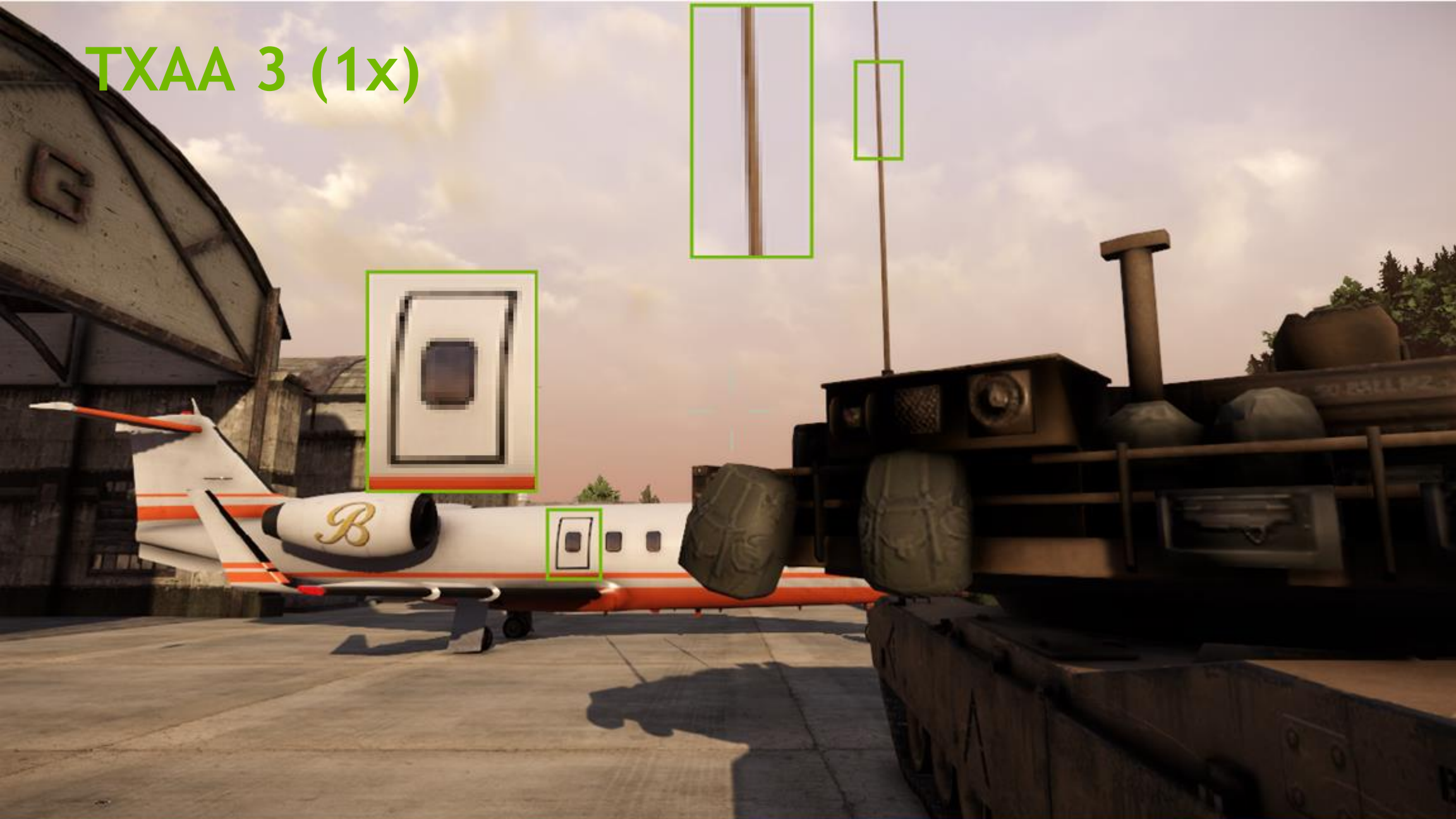
- 其它:
  - TXAA 3.0同样提供了用于debug的特性
  - 除了TXAA3本身，我们另外提供了一些帮助接口便于集成
    - 边缘检测
    - 根据深度和相机信息生成位移向量


No AA


4x MSAA

TXAA 2 (4x)




TXAA 3 (1x)

TXAA 3.0 性能

- 相对于其它抗锯齿特效，TXAA 3.0的开销非常小，几乎可以忽略不计
- 由于TXAA只有一个Draw Call，相对来说开销很小
  - 在GTX 980上，开销小于1ms
  - 即使开启了4x MSAA

Ansel

自由视角

滤镜

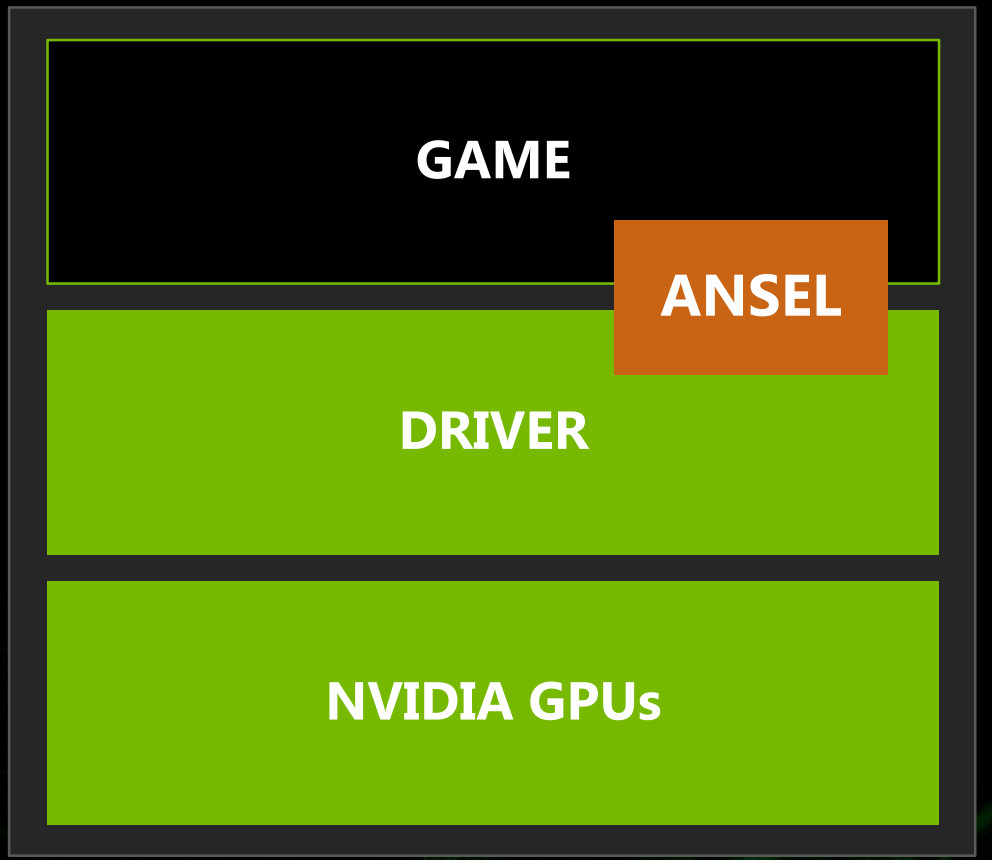
HDR

超高分辨率

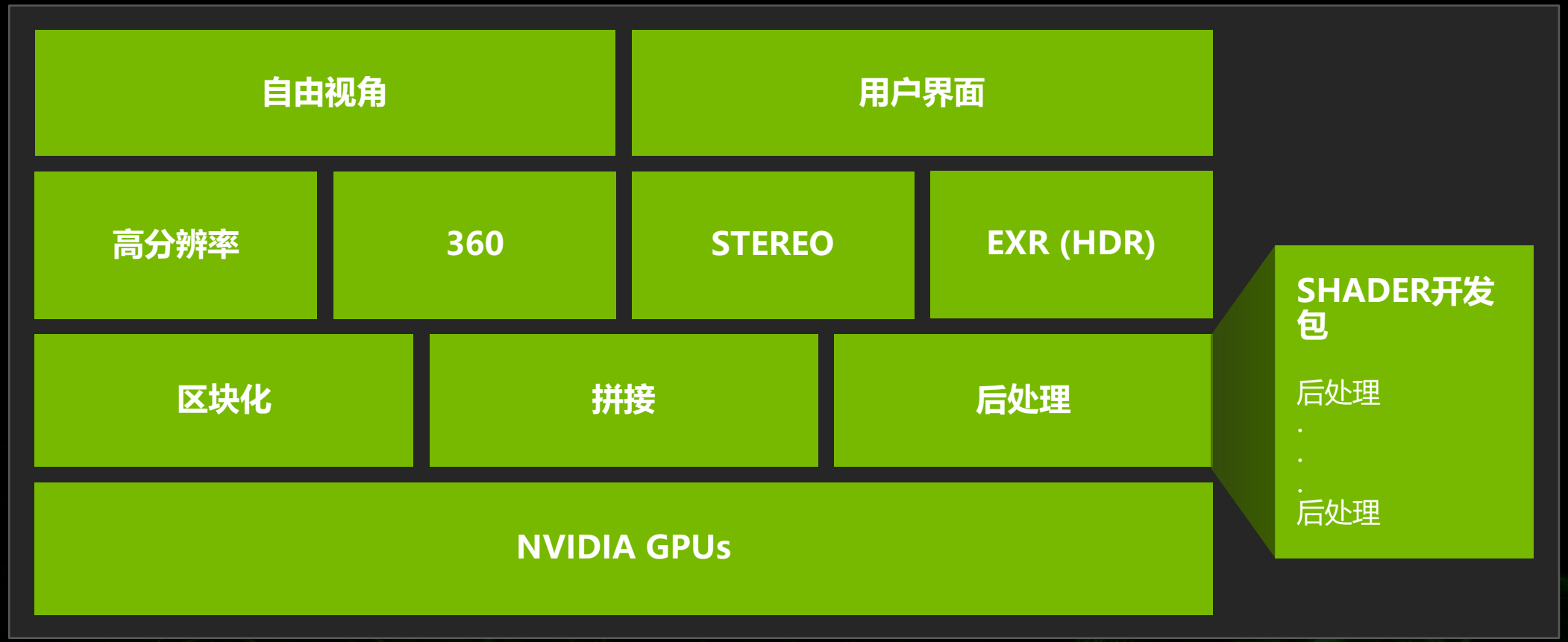
360

ANSEL集成

- 非常容易集成
- 提供UI控制
- ‘见证者’的集成只用了40行代码
- ‘巫师3’的集成只用了150行代码



ANSEL架构


自由视角
用户界面
高分辨率
360
STEREO
EXR (HDR)
区块化
拼接
后处理
NVIDIA GPUs
SHADER开发包
后处理
.
.
.
后处理

自由视角

超高分辨率

ANSEL 后处理特效

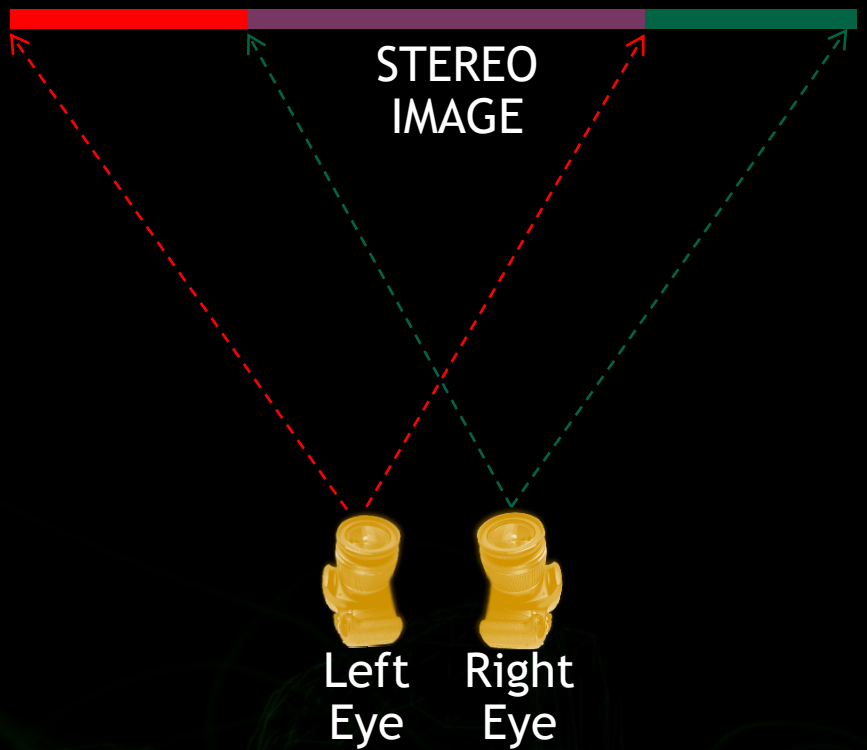
GAME ENGINE → BUFFERS COLOR BUFFER Z BUFFER G BUFFER … → POST PROCESS SHADER →

更多的图形特效!

- Color curves
- Sketch
- Color space transformation
  - Hue shift
  - Desaturation
  - Brightness
- Contrast
- Film grain
- Bloom
- Lens flares
- Anamorphic glare
- Distortion effects
  - Map distortion ("heathaze")
  - Fisheye
  - Explosion
- Color aberration
- Tonemapping schemes
  - Haarm-Peter Duiker
  - Reinhard
  - Hable (Uncharted2)
- Lens dirt
- Lightshafts
- Vignette
- Gamma correction
- Convolution filters
  - Sharpening
  - Edge detection
  - Blur
  - Hipass/lowpass
- FXAA
- Sepia tone
- Halftone
- Deband
- Denoise

360 图像

Stereo 模式截取

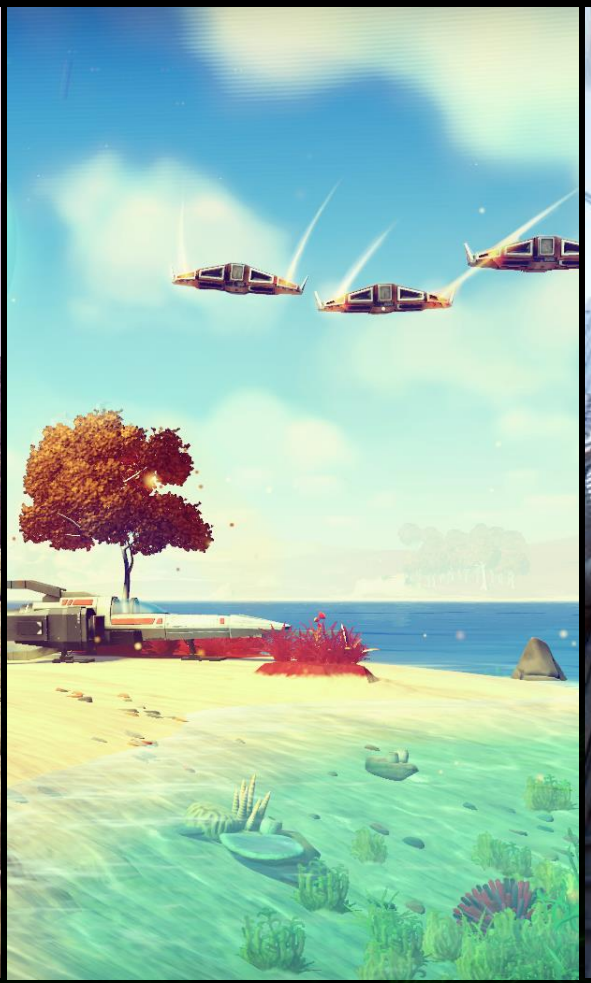


近期支持Ansel的游戏





近期支持Ansel的游戏



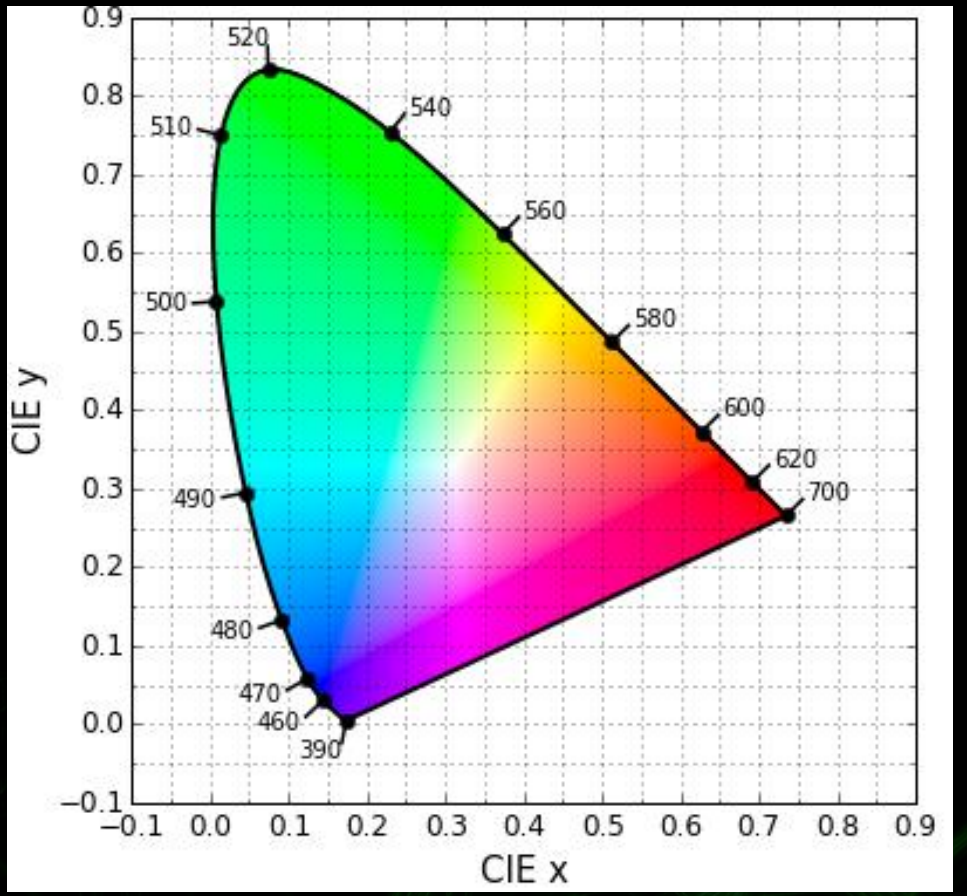
为什么要升级HDR显示器

- LDR显示器显示的色彩范围非常有限
- 非HDR显示器
  - 高光表现平淡，很难与高亮度漫反射的白色区分
  - 高光处会在饱和度方面进行妥协
  - 暗处的阴影会非常暗淡，丢失细节
- HDR显示器
  - 更高的亮度 (1000+nits)
  - 更广的色彩范围

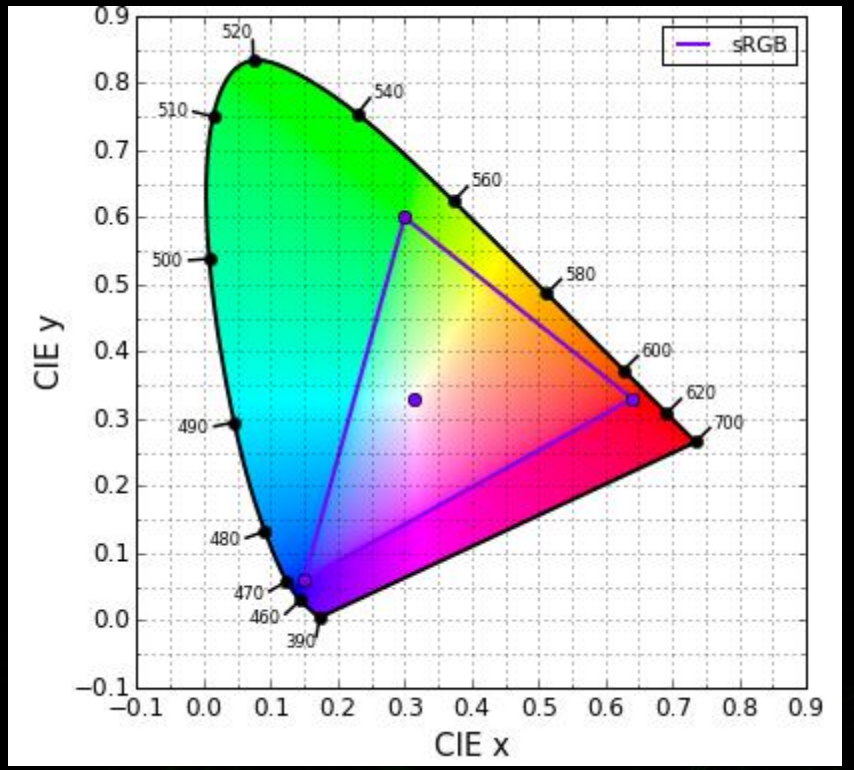
LDR 显示器的图像

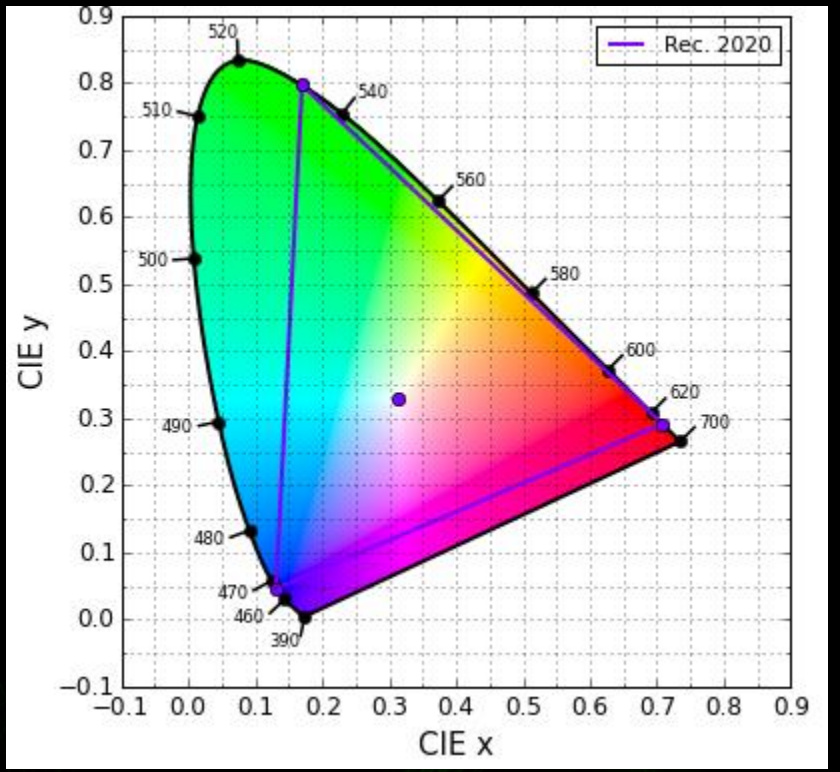
CIE 色度图

- 它表示了对人眼可见的颜色范围
- 边缘的颜色具有较高的饱和度
- 任何内部的颜色都可以由边缘的颜色插值而成



sRGB vs BT.2020/Rec.2020

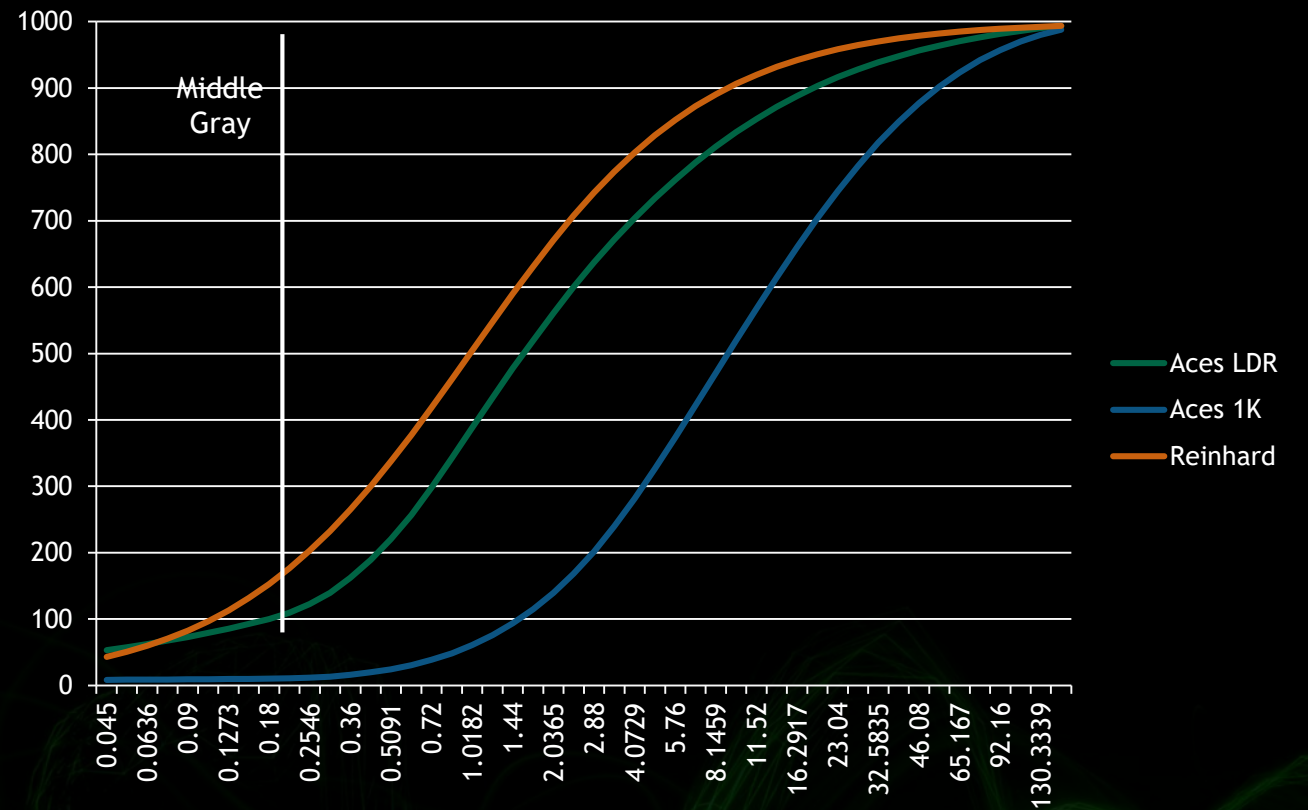




- sRGB
- BT.2020/Rec.2020

传统的ToneMapping的方式是否仍然使用？

- 压缩过度的图像
- 丢失了高光处的细节
- 即使原本较暗的细节也会变的很亮
  - Middle-Grey会表现为亮白色



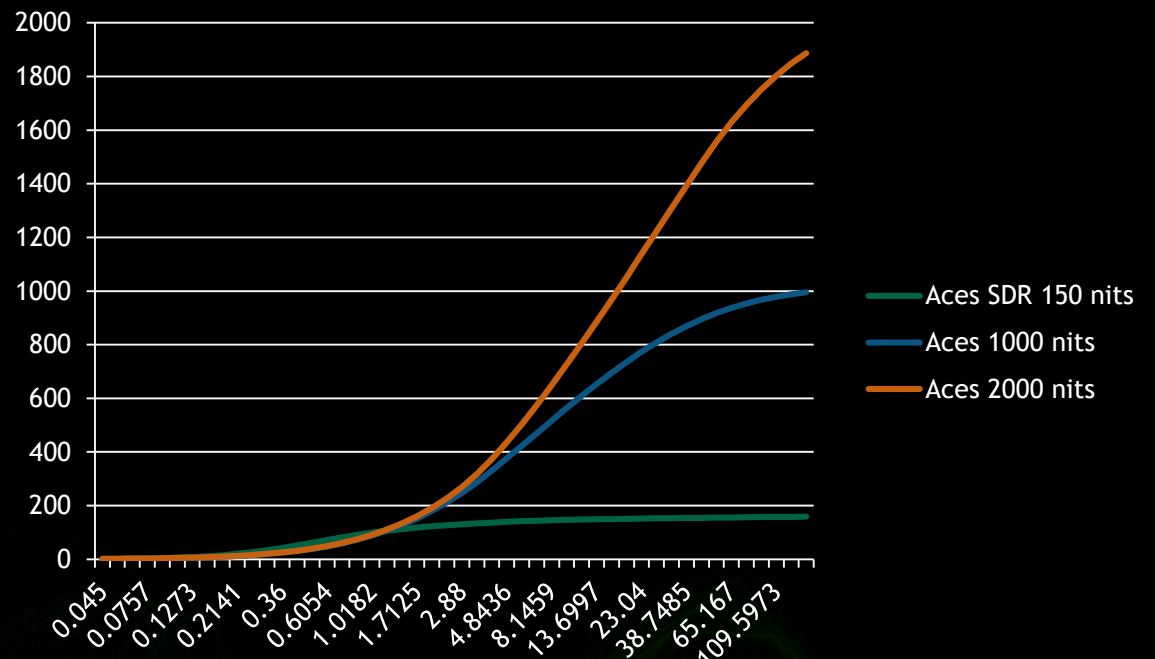
可以移除ToneMapping么？

- 既然已经有HDR显示器，ToneMapping还有意义么？
- 是的.
  - 最好的HDR显示器也远远不能够表现自然界中的颜色范围
  - 第一代的HDR显示器只能够生成1000 nits亮度的光线
  - BT. 2020 标准最多支持的光线强度为10000 nits.

不同的条件	亮度 (cd/m^2, or nits)
日出	600000
60瓦的灯泡	120000
晴朗的天空	8000
典型的办公室	100-1000
LDR显示器	80-100
多云的夜晚	0.25

一种新的ToneMapping算法

- Academy Color Encoding System (ACES)中的ToneMapping算法
  - 呈S状
  - 对于每个通道分别操作
- 我们已经为大家实现好了该算法



UI 混合

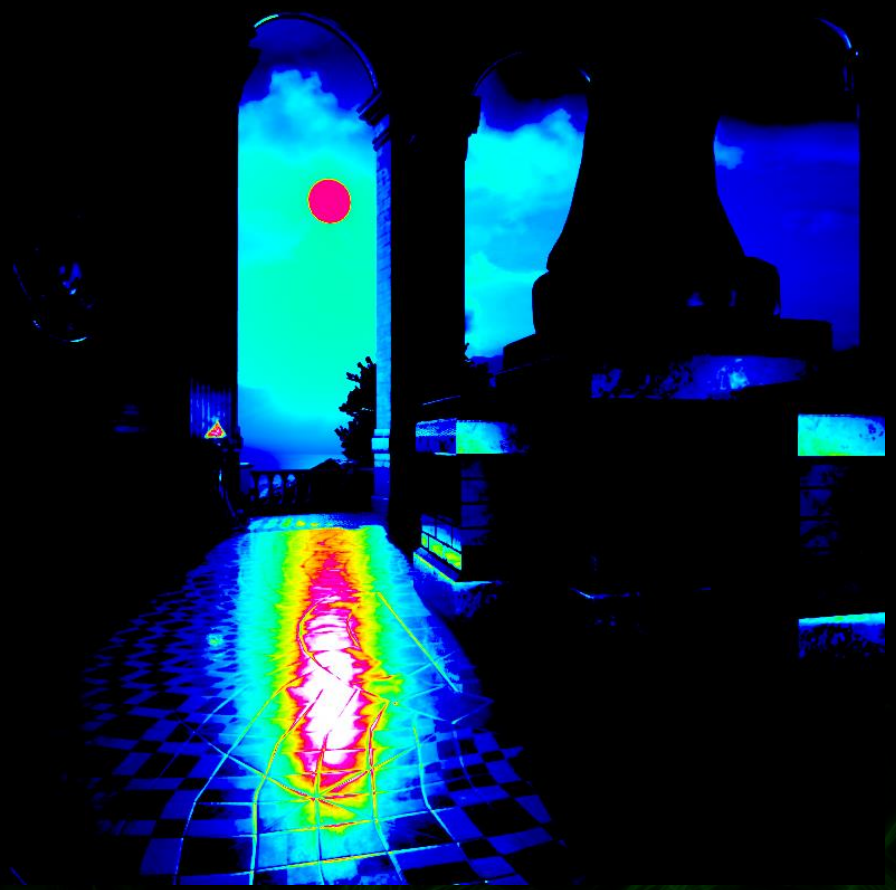
- 通常是在sRGB空间中进行的，发生在3D渲染后
- 在HDR显示器中，工作原理类似
- 提示:
  - 不要用最大亮度的颜色
  - 建议增加一个颜色缩放的参数
  - 需要注意Alpha Blending，可以对背景使用简单的tone mapper(Reinhard)

基于物理的光照渲染

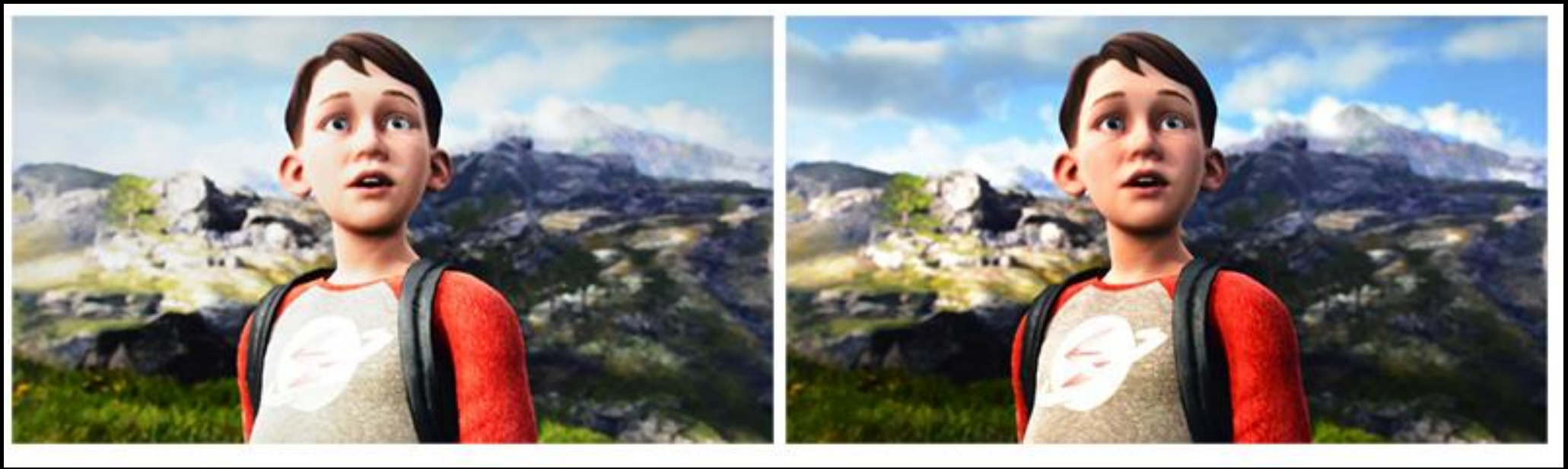
- 可以生成亮度更为丰富的图像

注意‘光源’几何体

- ‘光源’几何体的亮度可能会小于反射光线的亮度
- 在LDR显示器中，不会有严重问题因为两者都达到了最大的颜色亮度



LDR vs HDR



- Left : LDR渲染的图像
- Right : 真实相机拍摄下的HDR显示器渲染的图像

Q&A

THANKS