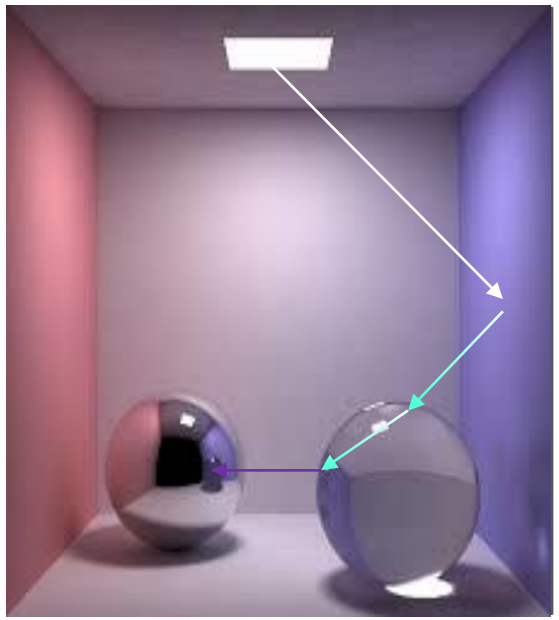
NVIDIA TURING GPU 光栅化管线中的全新渲染技术

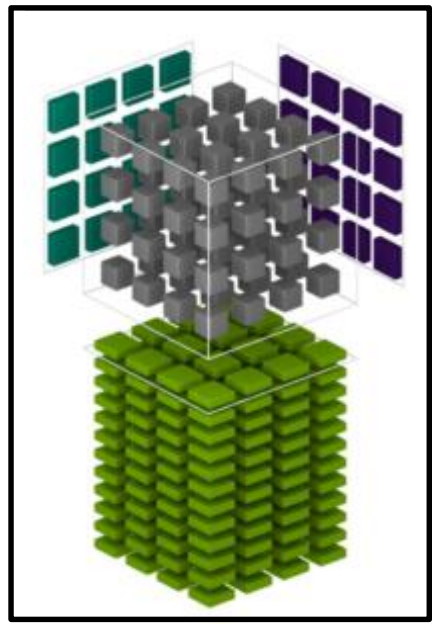
杨雪青, 11月22日

TURING 架构

全新架构



RT CORES



TENSOR CORES/AI

先进的着色技术

概要

网格模型着色器（Mesh Shader）

变频着色（Variable Rate Shading）

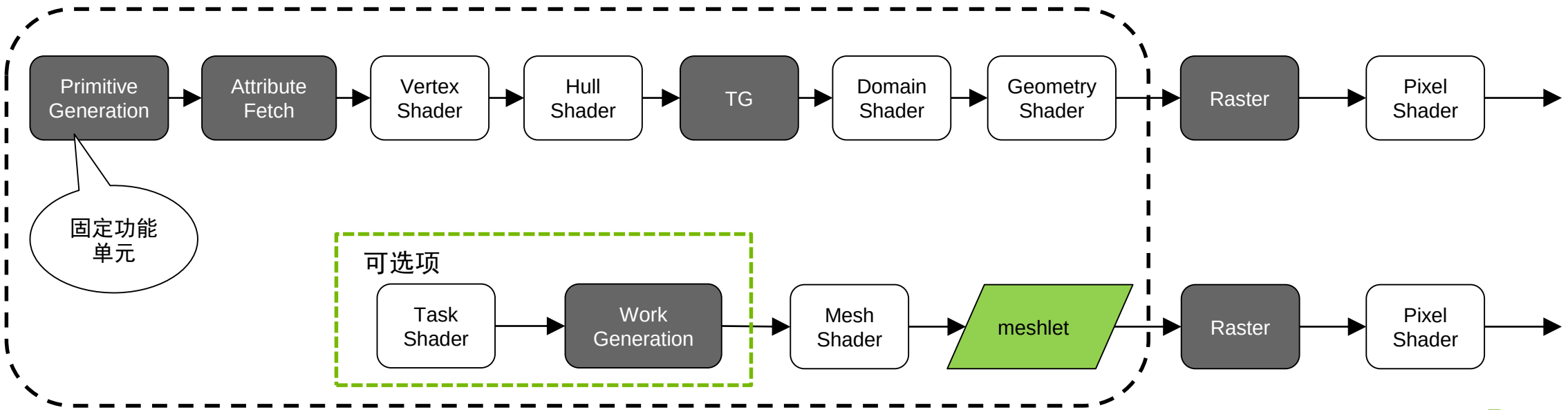
多视图渲染（Multiview Rendering）

贴图空间渲染（Texture space rendering）


网格模型着色器
(MESH SHADER)

MESHLET与MESH SHADER

- 传统光栅化管线的几何处理部分: VS→Tessellation→GS (VTG管线) … →光栅化
- 替代方案: Mesh Shading 管线 … →光栅化
  - Mesh Shader = 能够输出特定数据包(Meshlet)到光栅化器的Compute Shader



为何要启用新的管线

- 传统VTG管线的问题
  - 冗长的可编程+固定功能混合管线，可能造成性能问题
  - 无法并行化处理Index buffer的输入和展开
  - 顶点重用依赖于硬件内置的顶点缓存
  - 大量顶点在VTG后被剪裁和剔除，浪费计算资源

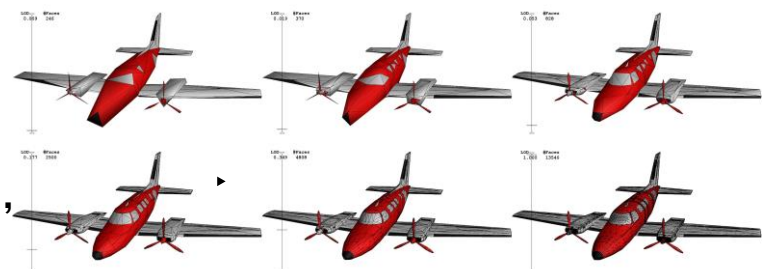
MESH SHADER

- Mesh Shader编程同Compute Shader
  - 线程组(Threadgroup): 32线程
  - 可使用Shared Memory和Barrier
  - 需自行输入数据(无数据格式限制)
  - 可完全处理：输入、剪裁、剔除、曲面细分、顶点间数据共享
- 输出指定格式的Meshlet数据
  - 最多存放256顶点和512图元
  - 必须数据: 顶点属性 + 图元属性
  - 可选数据: 系统变量(View Index，Render Target Index等)

MESH SHADER的用途

- 大规模渲染
  - 大型植被系统：数百至数千不同种类的植被
  - 大量同屏角色：绘制数千至数万动画角色
  - 海量decal系统：多样化、高密度的装饰物件

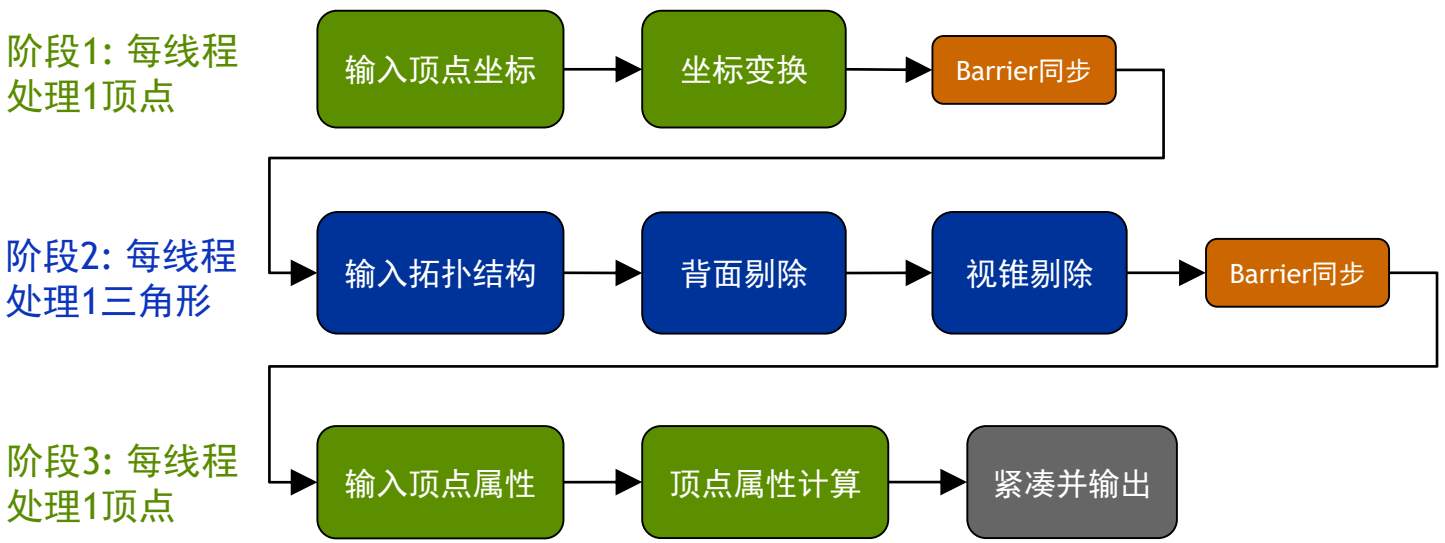
- 程序化模型
  - 渐进模型：动态融合LOD，达到平滑过渡
  - 模型压缩：自定义网格模型的压缩与解压缩
  - 体素模型的网格化：提取等值面和隐式曲面，转化为三角形



- 建模与可视化软件
  - 高模的显示与编辑：提高顶点复用率；增加批量剔除

应用举例(1)：三角形批量剔除

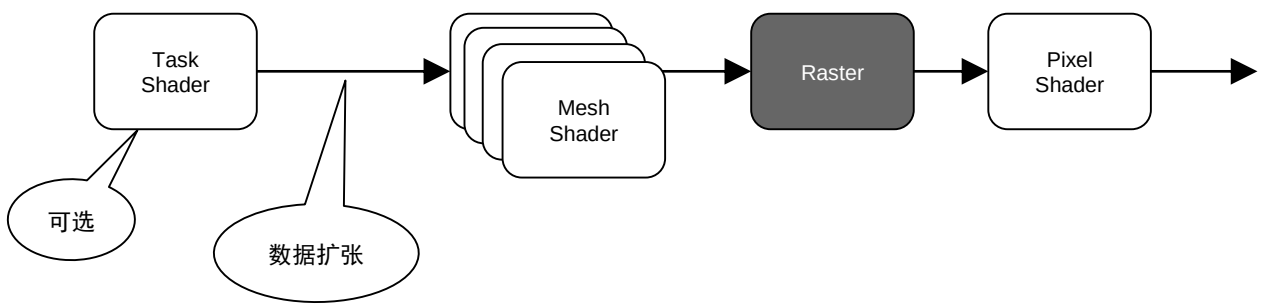
Mesh shader流程



- 先剔除再计算属性，避免了不必要顶点的计算浪费
- 输入模型需预先处理，提高邻近三角形的剔除效率

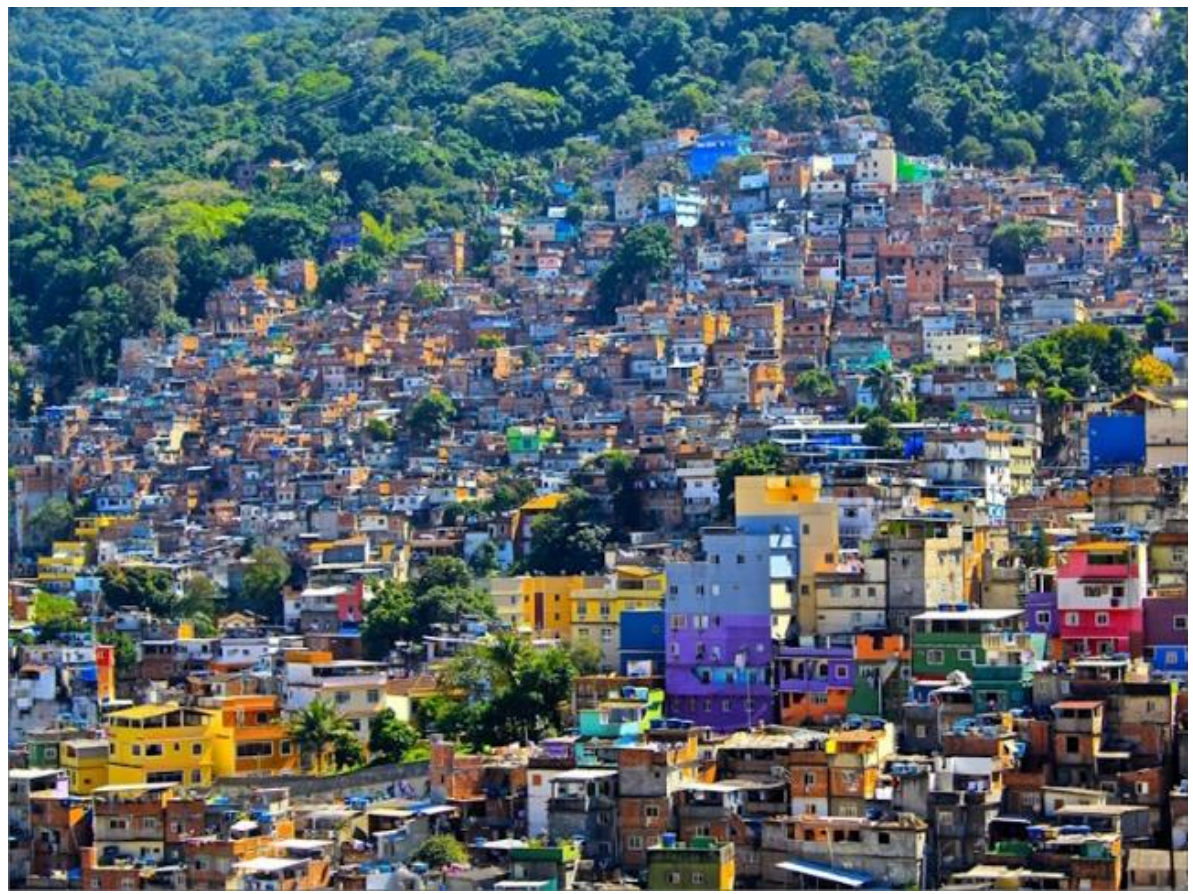
可选单元: 任务着色器(TASK SHADER)

- 在管线最前端引入Task Shader来实现数据扩张
  - 同Compute Shader. 可输出变长数据，创建并调用一系列Mesh Shader
  - 可动态控制Mesh Shader的数量



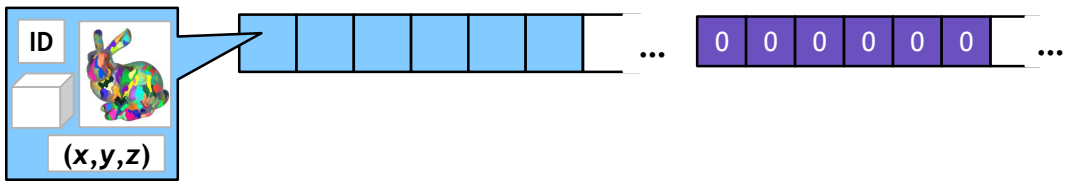
应用举例(2): 复杂INSTANCING与LOD

- 丰富的视觉元素
  - 海量Decal
  - 大型植被系统
  - 大量同屏角色
- 传统方法难以实现
  - 200,000物体同屏: 每个物体需要在10ns内完成模型选择、LOD计算和Draw Call



使用MESH SHADER实现大规模渲染

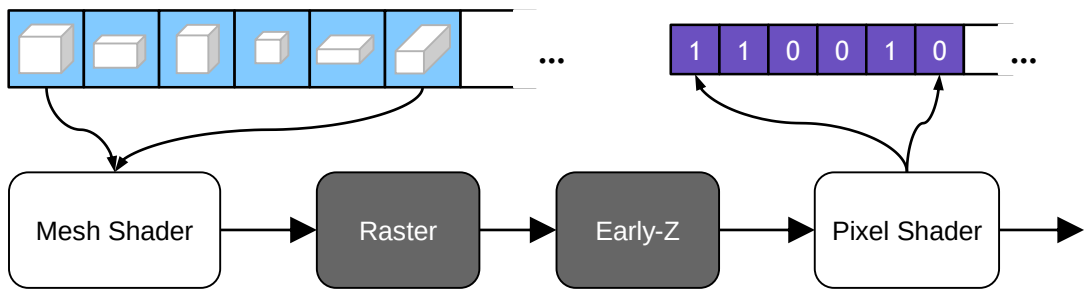
使用一维数组存储物体信息
使用UAV数组表示各物体的可见性



遮挡剔除

Mesh Shader: 绘制包围盒

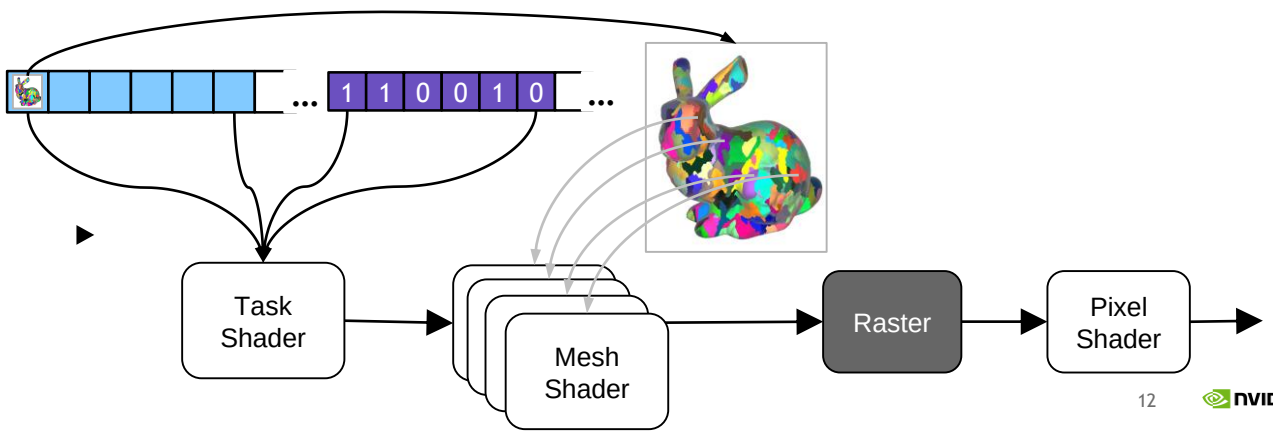
Pixel Shader: 将UAV中对应标记写入1



绘制

Task Shader: 模型选择→LOD计算→视锥剔除→骨骼计算→创建mesh shaders

Mesh Shader: 面片读取→蒙皮计算→背面剔除→顶点属性计算→输出meshlet包




变频着色
(VARIABLE RATE SHADING)

粗粒度着色(COARSE SHADING)

多个像素组成一个**粗粒度像素**(Coarse Pixel),执行一次像素着色器

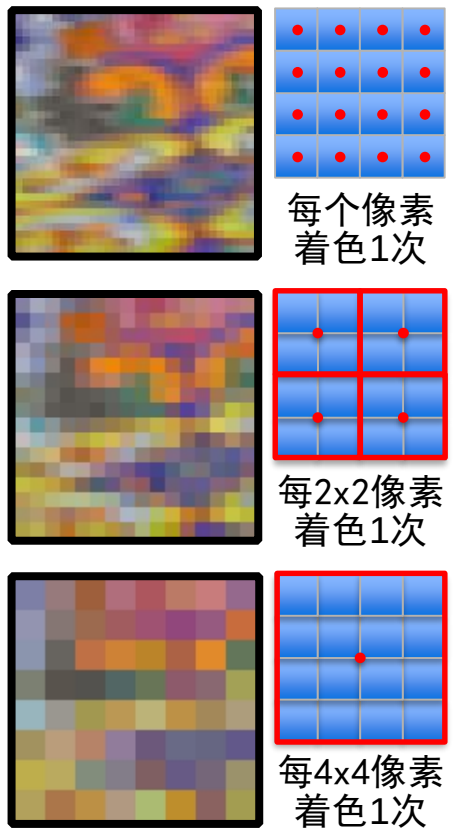
与MSAA有相似之处

图像上并不是所有部分都需要做高采样率的渲染

使用粗粒度着色可以提高效率

但是粗粒度着色会不会把整张图都搞糊?

希望能够只对需要的部分进行粗粒度着色



变频着色

应用程序用一个着色频率表面(Shading Rate Surface)来为不同的**16x16 像素区域**设置像素着色频率

深度值始终在全像素频率下进行光栅化

类似于MSAA:

- 像素着色器线程的数量取决于着色频率
  - 由于几何复杂度，实际的比率会有所上升
- 光栅化器(Raster)会把输出的颜色推广到所有被覆盖的像素/样点

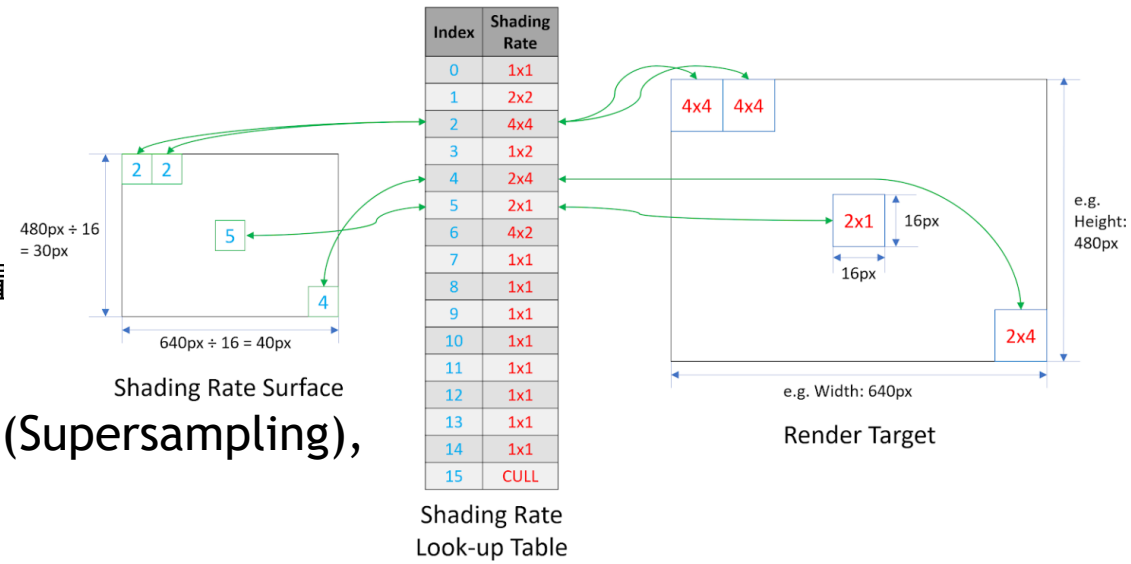
但是更加的灵活...

变频着色

光栅化硬件从一个二维查询表格中读取着色频率

应用程序通过设置**枚举值**来声明着色频率

为每个16x16的像素区域都使用一个枚举值来设置所需的着色频率



枚举值所对应的模式包括：剔除(Culling)，超采样(Supersampling)，常规渲染及粗粒度着色

CULL (16x16区域内的像素都不会被着色)

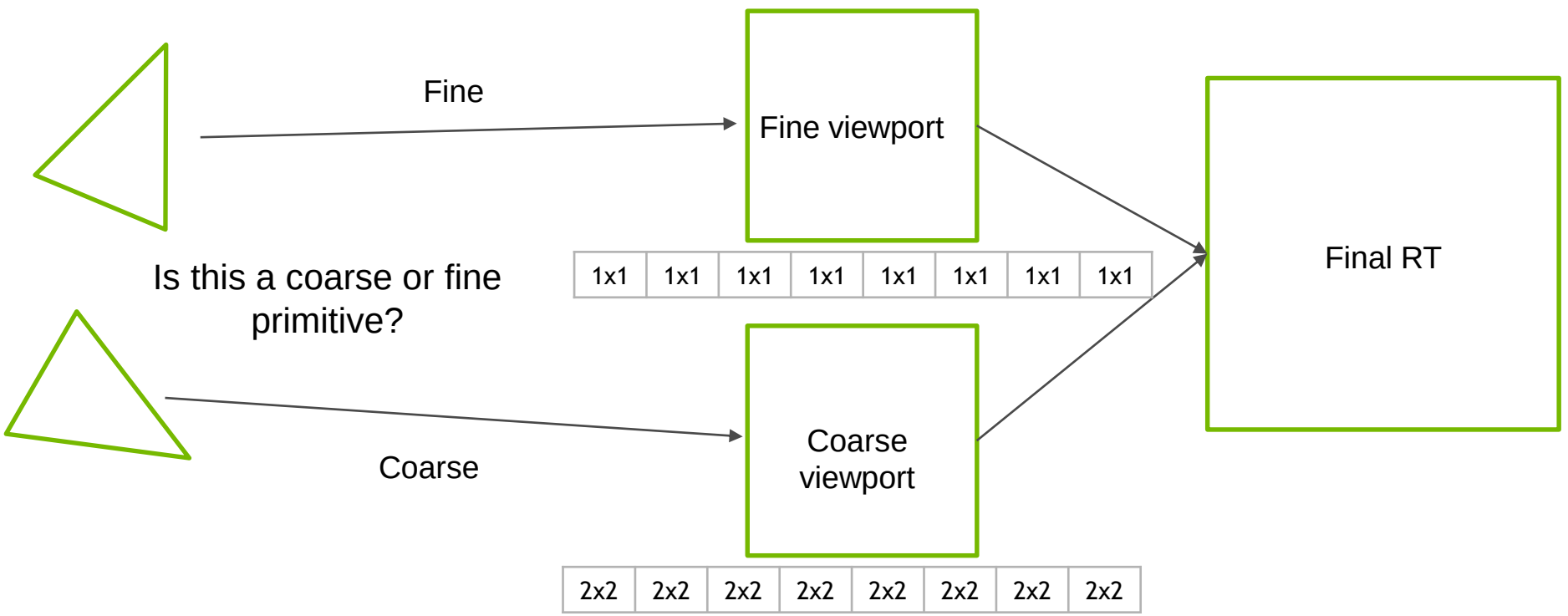
每像素对应8, 4, 2个着色线程 (超采样)

每个像素对应1个着色线程 (常规渲染/MSAA)

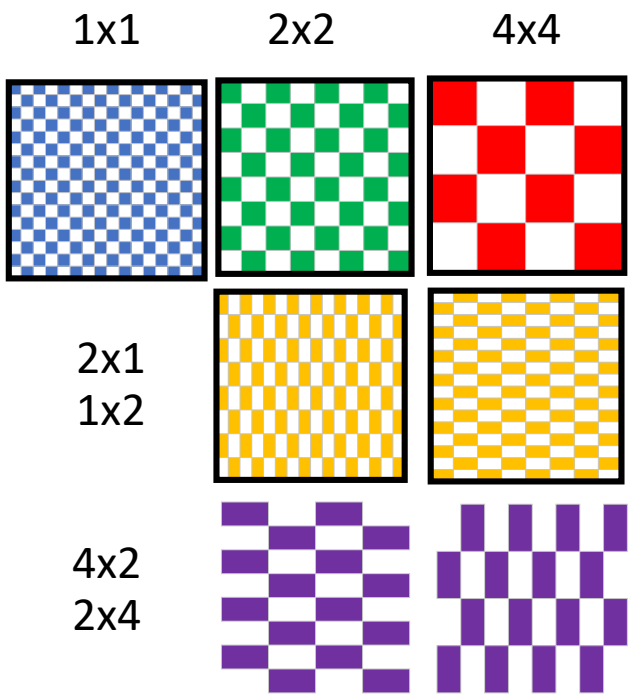
每个着色线程对应2x1, 1x2, 2x2, 4x2, 2x4, 4x4 个像素(粗粒度着色)

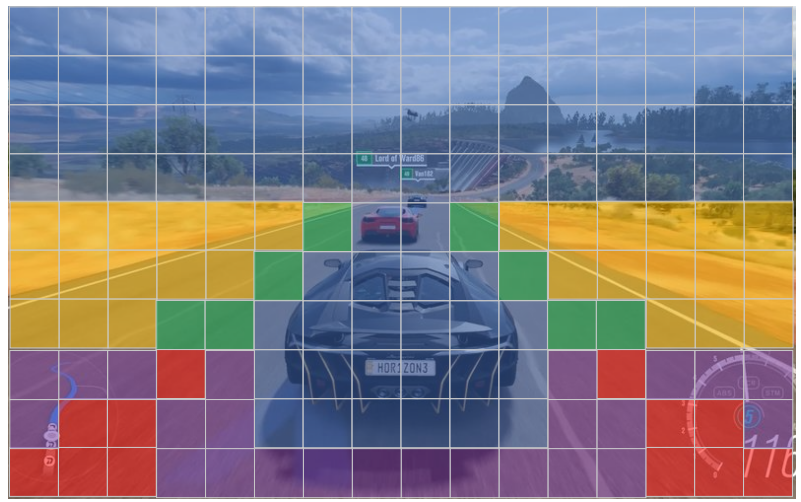
变频着色

还可以针对每个物体或三角形设置着色频率



变频着色

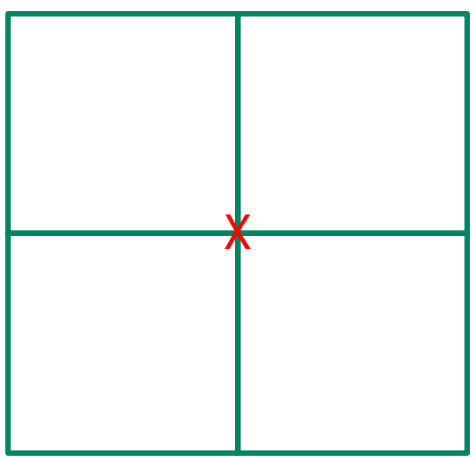




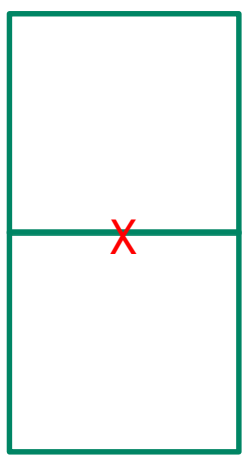
变频着色

类似MSAA，粗粒度像素(Coarse Pixel)的属性通过粗粒度像素的中心进行插值计算

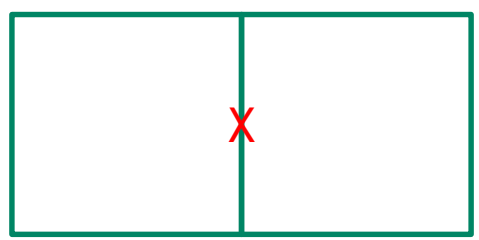
Centroid模式可以用来调整插值中心，但是梯度计算可能会受影响



2x2



1x2



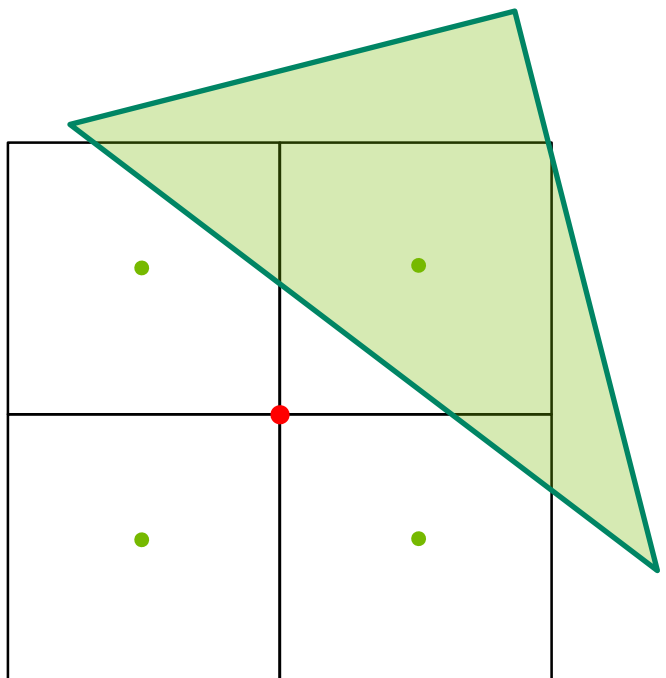
2x1

变频着色

以2x2 的粗粒度像素为例

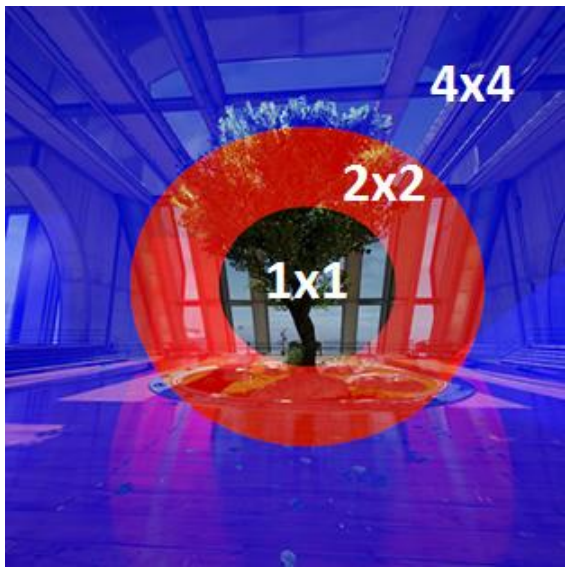
没有用 Centroid

使用 Centroid

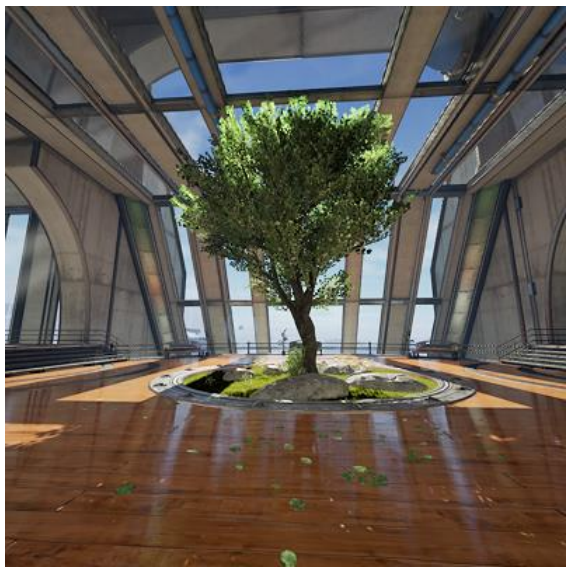


变频着色

在需要的地方重点着色



眼球追踪渲染
(Foveated Rendering)



内容自适应着色
(Content Adaptive Shading)

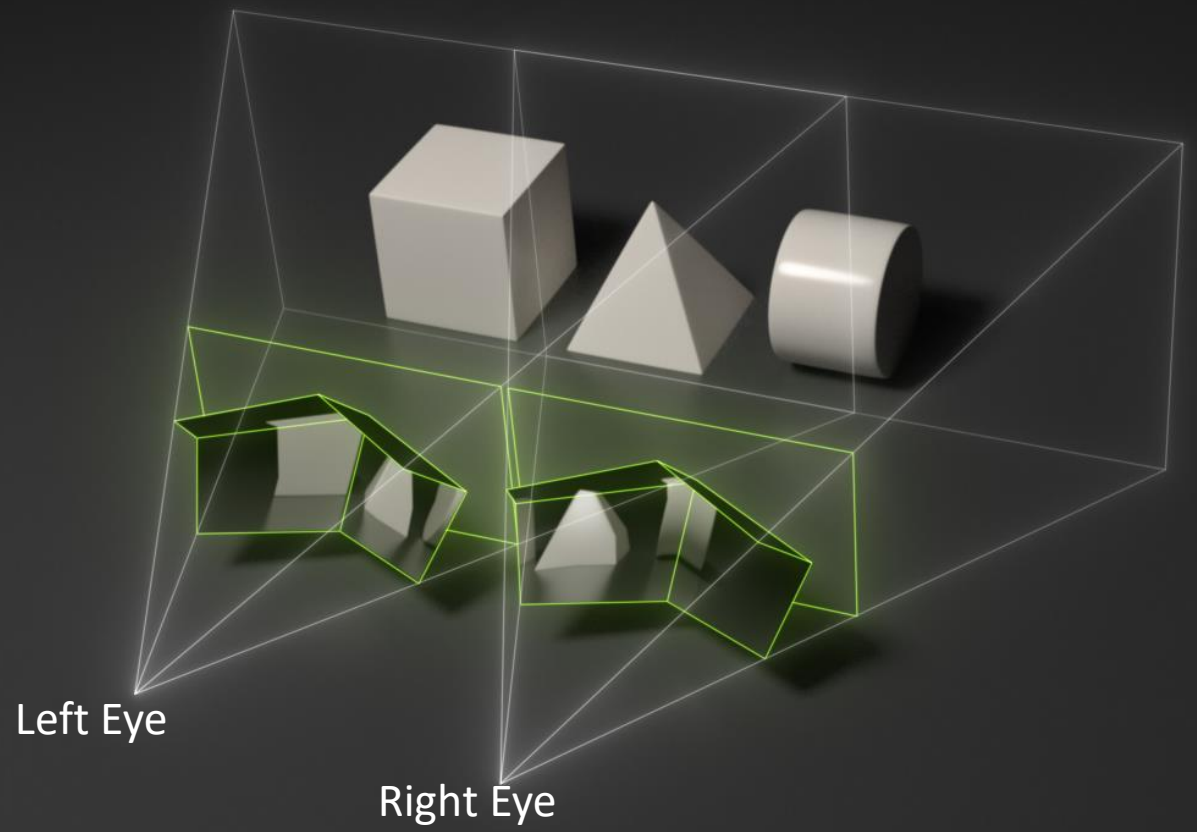
运动自适应着色
(Motion Adaptive Shading)


Come on, let's pave the way for our people on the ground.
100
75

多视图渲染
(MULTIVIEW RENDERING)

PASCAL：单遍立体(SINGLE PASS STEREO)

通过一个批次同时完成左右眼的几何体渲染

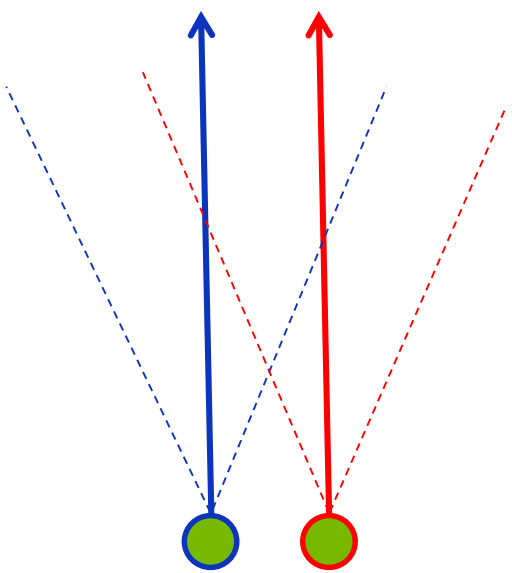


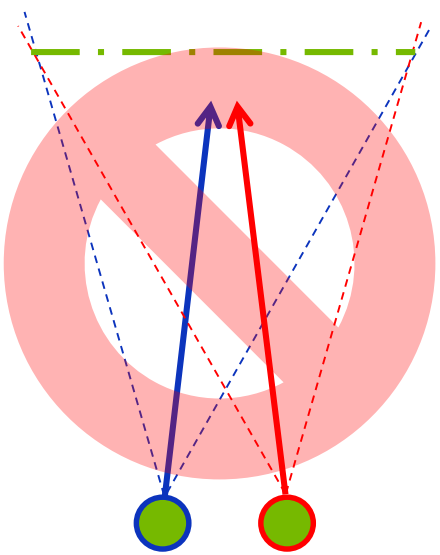
单遍立体

仅能同时渲染两个视图

除了顶点位置的X坐标外不支持其他的视图依赖属性 (view-dependent attributes)

需要在像素着色器中显式地选择与当前视图相匹配的输入属性





多视图渲染

同时渲染最多四个视图(View)

全面支持视图依赖属性 (View-Dependent Attributes)

- 4个位置(Position)属性, 128个通用 (Generic)属性以及 8 个裁剪距离 (Clip Distance)属性
- Viewport Index 及 Render Target Array Index
- **像素着色器可以自然地获取正确的输入属性**

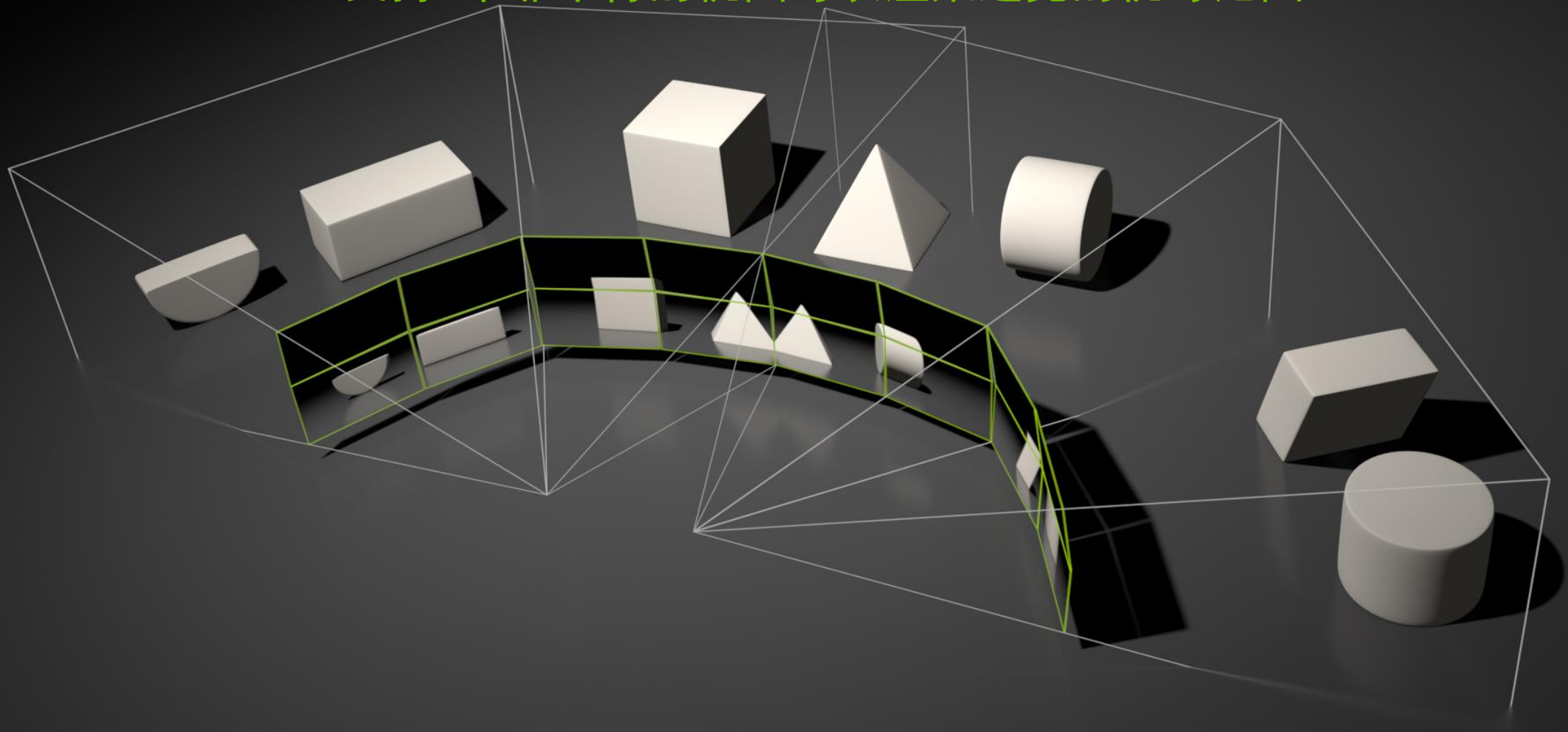
是SPS的超集

为 DX12 中的 View Instancing 特性提供了硬件实现支持

- VTG Shader根据当前的 **SV_ViewID** 来计算不同 View上的坐标和属性

TURING: 多视图渲染

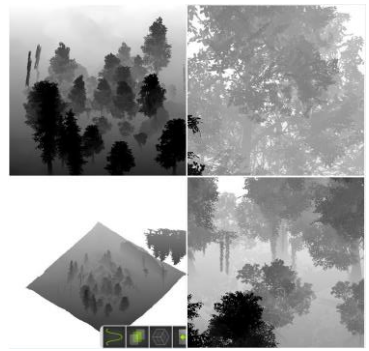
支持4个非平行的视图可以渲染超宽的视野范围



多视图渲染

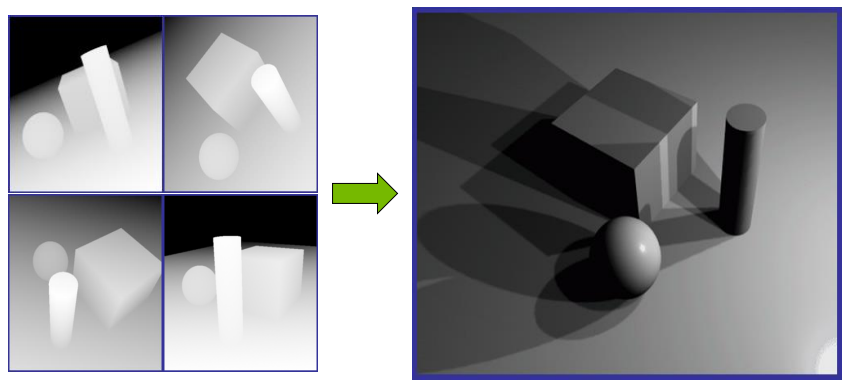
更广泛的应用

Stereo



Cascaded Shadow Maps

Surround



Multiple Shadows/Area Shadows

贴图空间渲染
(TEXTURE SPACE RENDERING)

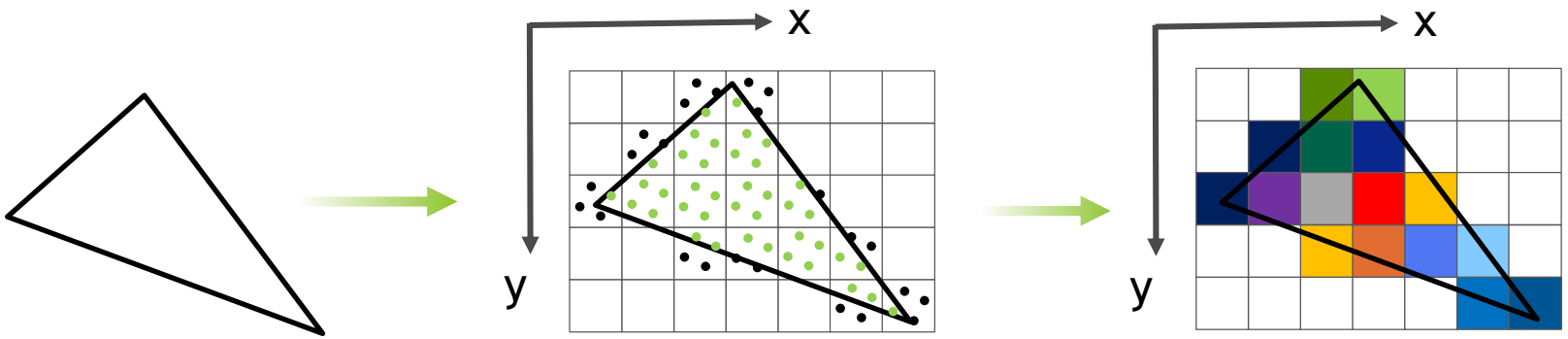
分离着色 (DECOUPLED SHADING)简介

- 一种有别于 forward shading 和 deferred shading 的着色方式
  - 既不是SDK特性也不是硬件特性
  - DX12和硬件提供固件来加速处理
- 目标: 使着色空间与采样空间完全分离
  - 着色可能发生在不同的分辨率下或在不同的时间线上
  - 应用可以显式地控制过采样或着色结果重用

基本上, 能够在不同的区域或时间上重用着色结果

传统着色管线

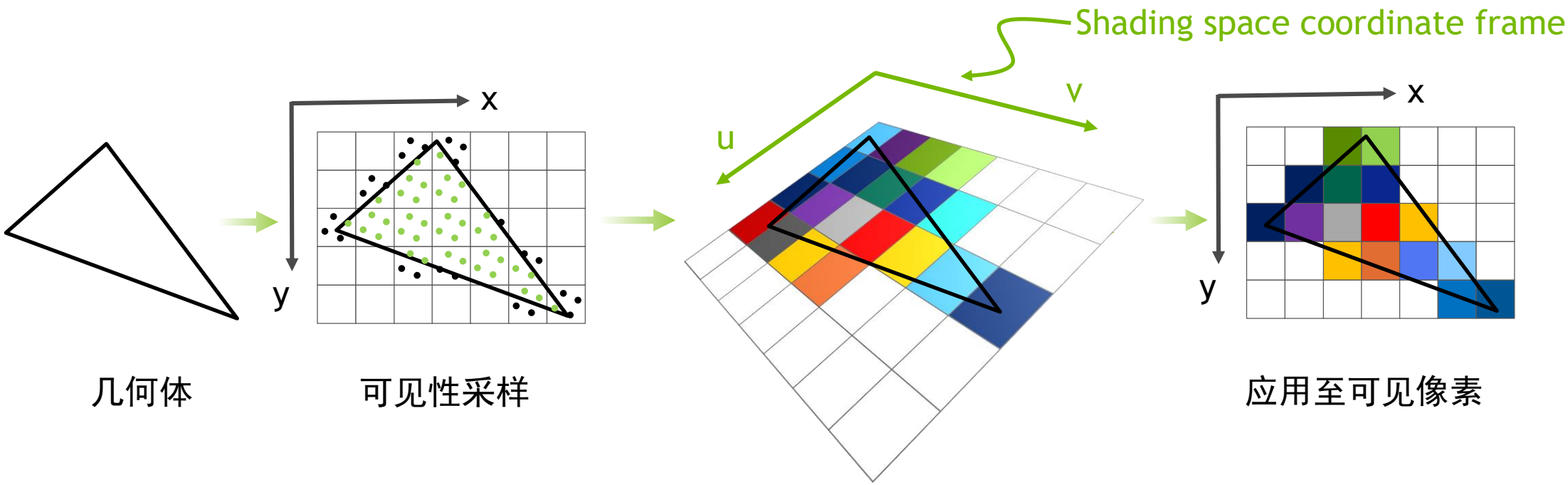
屏幕空间着色



几何体　　可见性采样　　着色采样

贴图空间着色(TEXTURE SPACE SHADING)

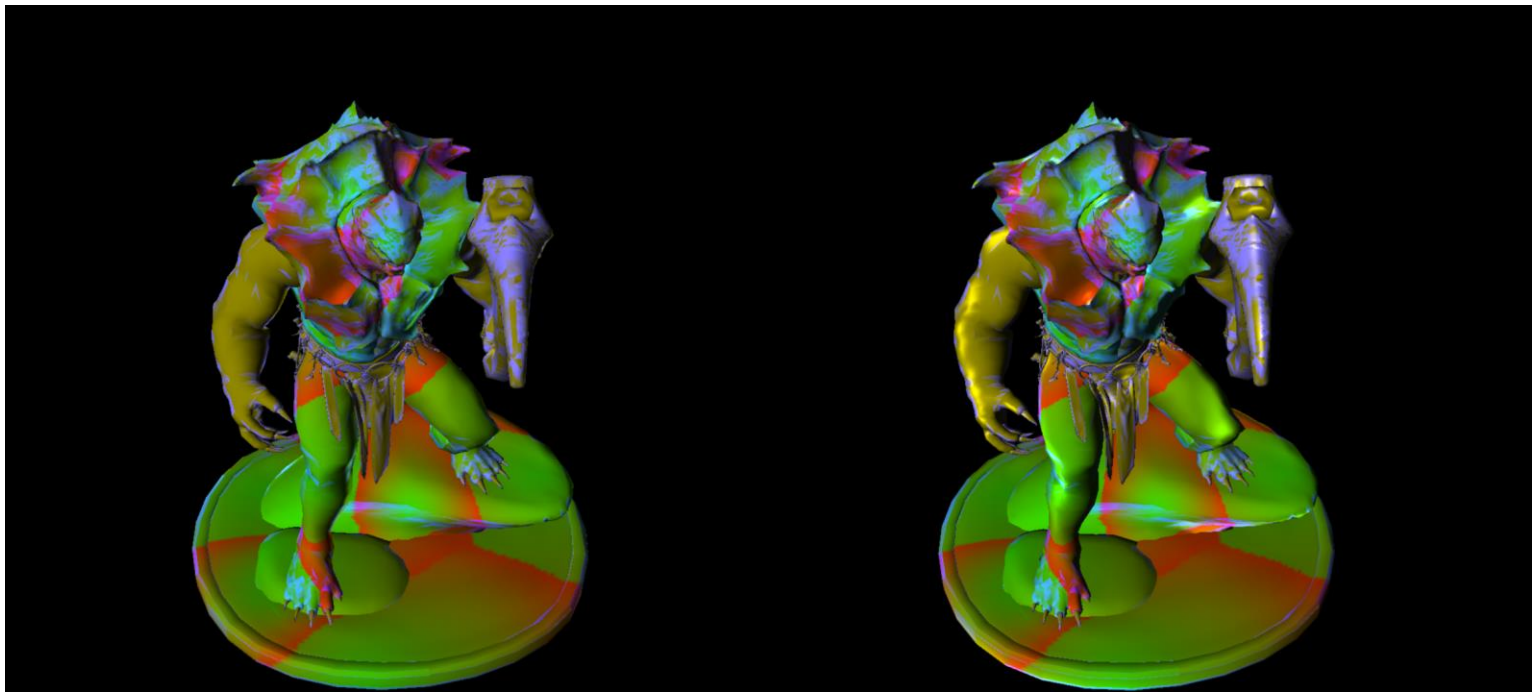
分离着色的一种



应用举例

立体渲染着色结果重用

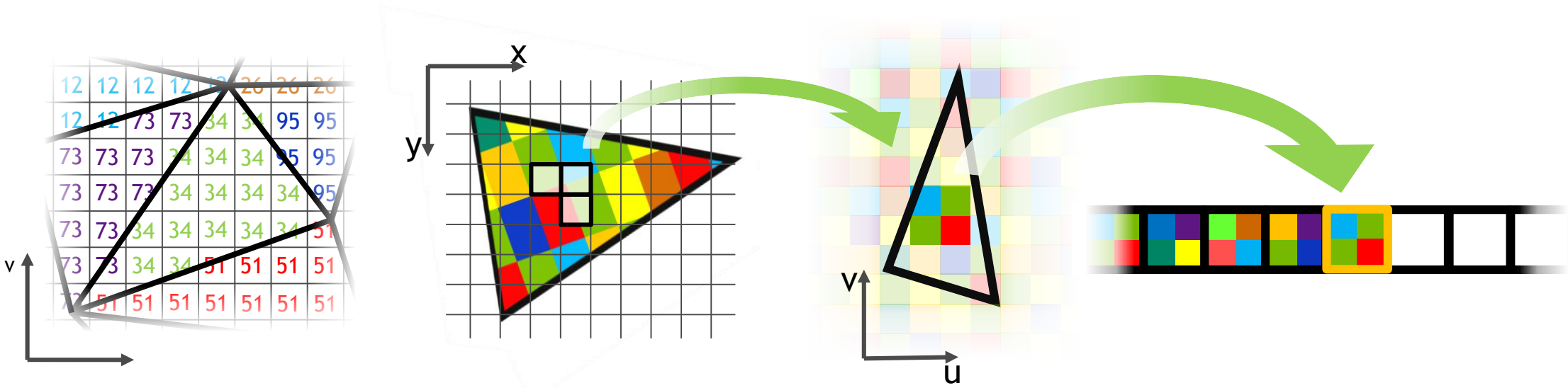
- 在大多数立体渲染中，双眼间的着色差异仅~5%



应用举例

立体渲染着色结果重用

1. 渲染V-Buffer: 三角形ID, 贴图坐标, LOD信息
2. 数据收集: 将可见像素映射到贴图空间，产生着色请求
3. 重复数据剔除: 剔除对应相同贴图区域的着色请求



应用举例

立体渲染着色结果重用

4. 队列: 将去重复后的着色请求放入着色请求队列
5. 着色: 对队列中的请求进行着色处理
6. 使用: 将着色结果应用至屏幕上的几何体

相关的TURING特性

Texture footprint

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	**1**	0	0	0	0
0	0	0	0	**1**	0	0	0
0	0	0	0	0	**1**	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

使用**Status Bit Surface**来剔除重复着色请求

Texture footprint

- 看上去像一次常规的贴图读取(Texture Fetch)
- 返回一个**64b**的掩码(Mask)
- Bitmask与贴图基本对齐
- 贴图读取代表每个图素组的Bit
  - 操作粒度(Granularity)是可配置的

*not always

相关的TURING特性

新的Warp-wide操作

更新Status Bit Surface时，会出现多个线程更新同一个Bit的现象

更新操作是原子操作，对同一个目的地进行原子操作会影响性能

找出一个Warp(32个线程)中操作同一个 8x8 Tile的线程

将这些线程中的 Footprint Bitmask合并

发送一个原子操作来更新Status Bit Surface

相关的TURING特性

HTEX

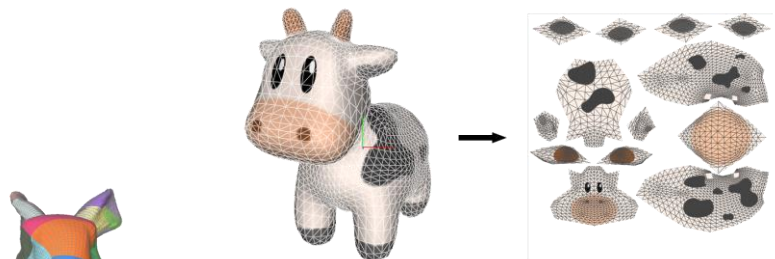
定义贴图空间

需要恰当地构建着色空间

必须要**唯一(unique)**

由美术设置

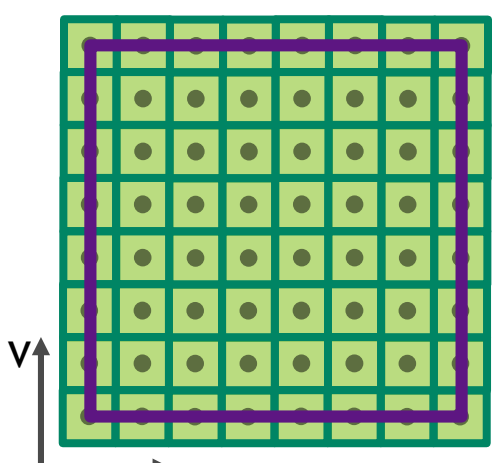
根据模型网格拓扑结构设置：可以自动生成



HTEX

对贴图上的图素的定位方式进行了调整

使得在接壤的面片(Patch)的边缘或角上可以对应相同的图素



应用范围

灵活的多频率着色 - **性能/质量**

高质量画面效果 - **性能/质量**

着色结果重用 - **性能**

提高微小三角形的渲染性能 - **性能**

Motion Blur

DOF

立体渲染


谢谢！
NVIDIA