
# 多频率着色

袁亚振


01 导引-计算频率

02 几何管线

03 着色管线

04 多频率着色优化

# 绘制管线中的计算频率



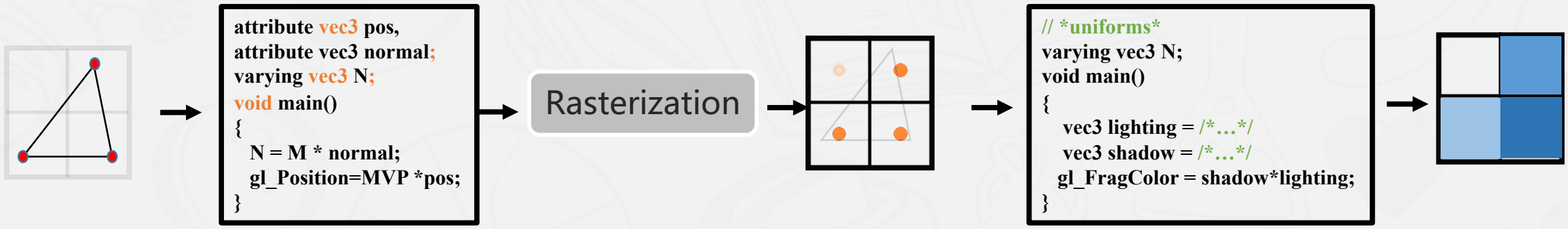
最简单的case：

Vertex Shader

每一个顶点执行一次

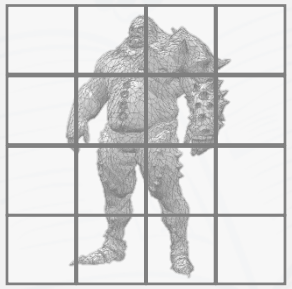
Fragment Shader

每一个像素执行一次

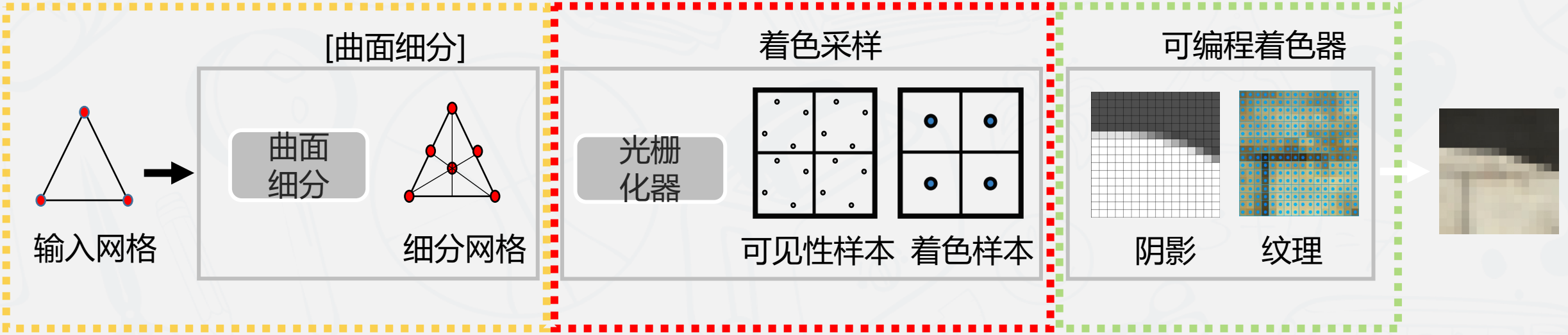


# 绘制管线中的计算频率



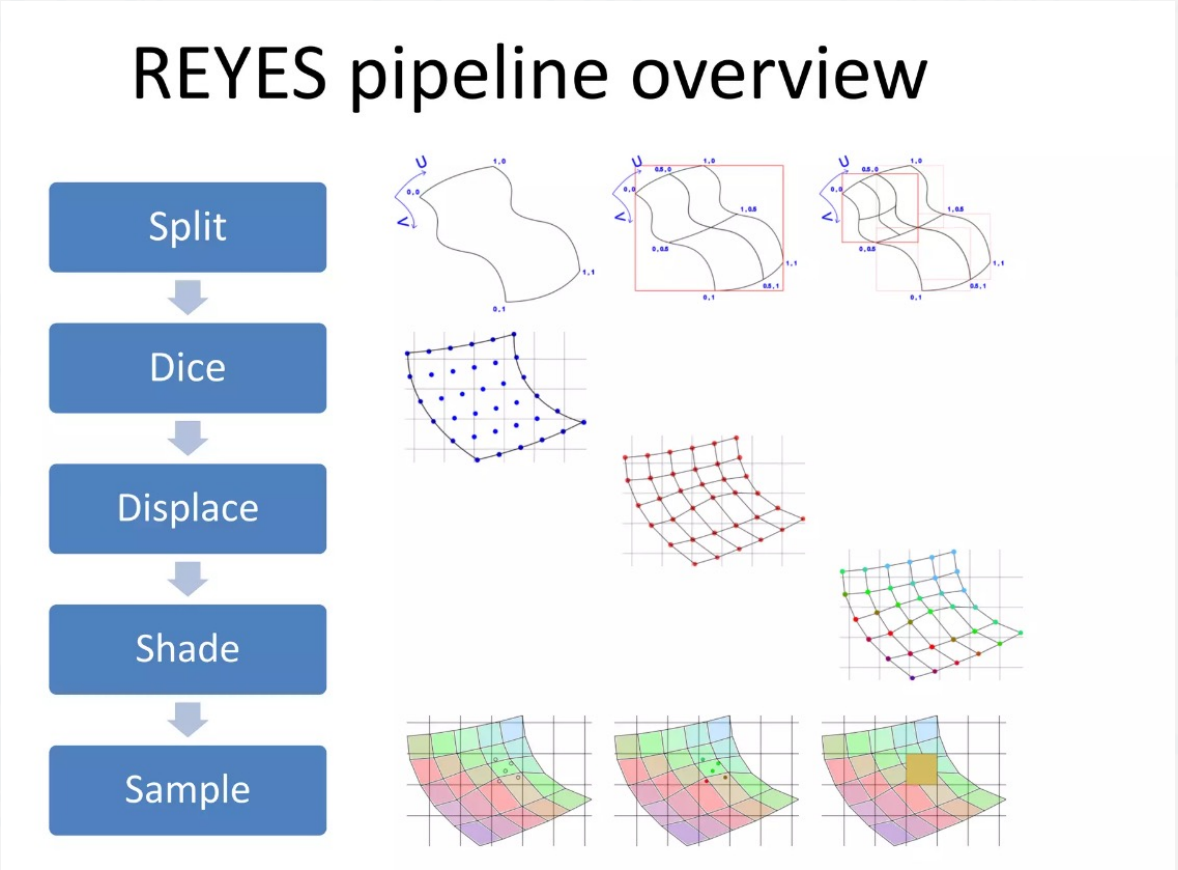


*实际上*

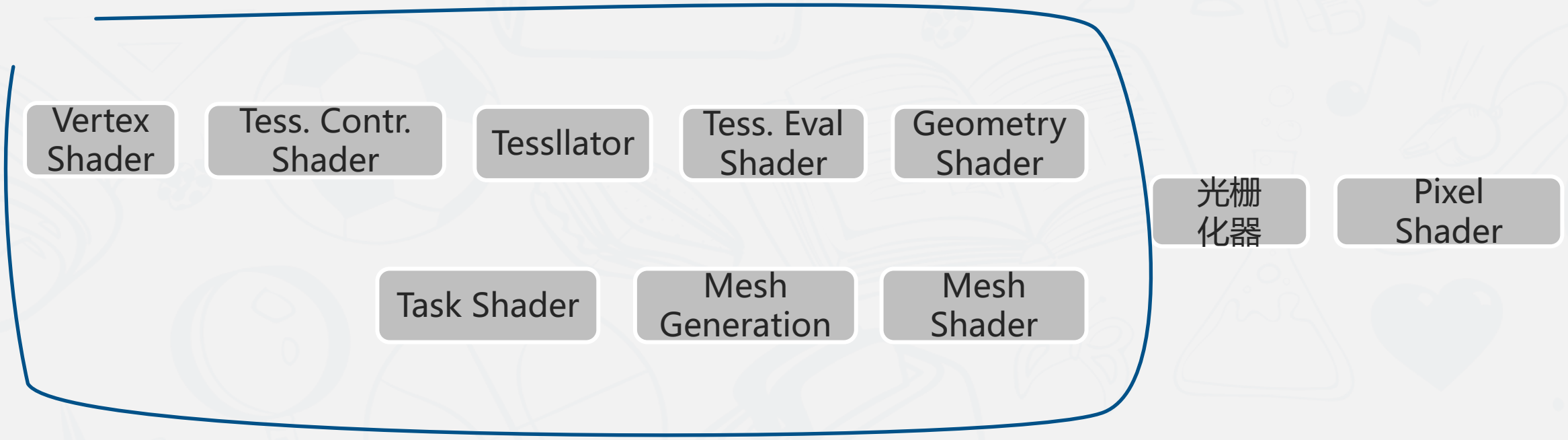


# 几何管线

- Render everything you ever see[1987]
  - 参数曲面，Catmull-Clark subdivision
  - “High-Quality” Movie



# 几何管线

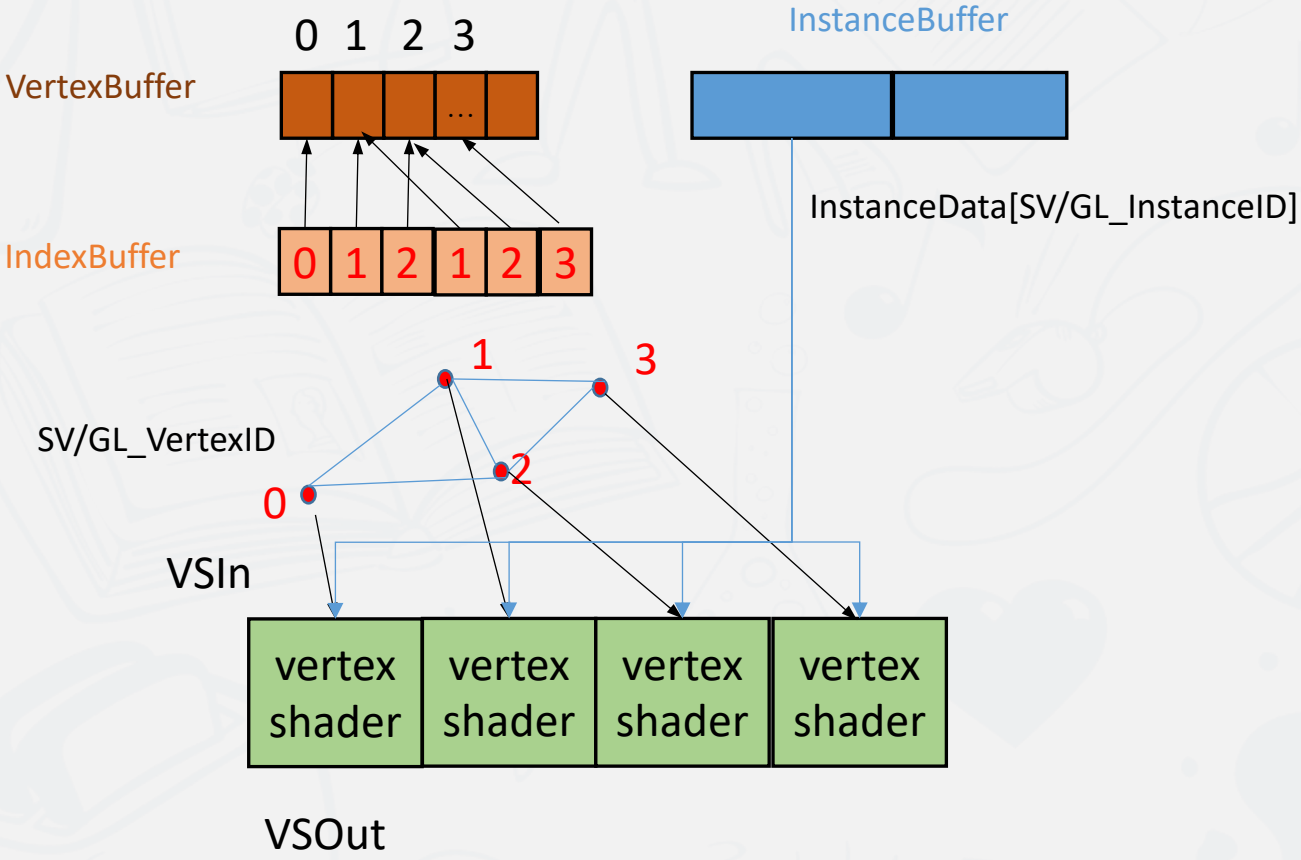

Vertex Shader
Tess. Contr. Shader
Tessllator
Tess. Eval Shader
Geometry Shader
Task Shader
Mesh Generation
Mesh Shader
光栅化器
Pixel Shader

几何管线

# Input Assembly & Vertex Shader



- Primitive Topology
  - VkPrimitiveTopology
    - VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST
- Provide Vertex Input Data
  - VkVertexInputRate
    - VK_VERTEX_INPUT_RATE_VERTEX
    - VK_VERTEX_INPUT_RATE_INSTANCE
- Semantics
  - SV/GL_VertexID
  - SV/GL_InstanceID

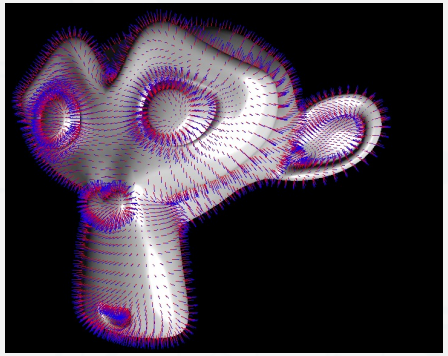


# Geometry Shader

# 几何管线

- Vertex Shader
  - *只能在CPU端设置几何*
- Geometry Shader
  - 自由的修改几何
  - EmitVertex()
  - EndPrimitive()
- Kind of Failure
  - 过于自由，速度不高

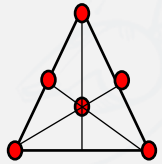
```
#version 450

layout (triangles) in;
layout (line_strip, max_vertices = 6) out;

layout (binding = 1) uniform UBO
{
    mat4 projection;
    mat4 model;
} ubo;

layout (location = 0) in vec3 inNormal[];

layout (location = 0) out vec3 outColor;

void main(void)
{
    float normalLength = 0.02;
    for(int i=0; i<gl_in.length(); i++)
    {
        vec3 pos = gl_in[i].gl_Position.xyz;
        vec3 normal = inNormal[i].xyz;

        gl_Position = ubo.projection * (ubo.model * vec4(pos, 1.0));
        outColor = vec3(1.0, 0.0, 0.0);
        EmitVertex();

        gl_Position = ubo.projection * (ubo.model * vec4(pos + normal * normalLength, 1.0));
        outColor = vec3(0.0, 0.0, 1.0);
        EmitVertex();

        EndPrimitive();
    }
}
```



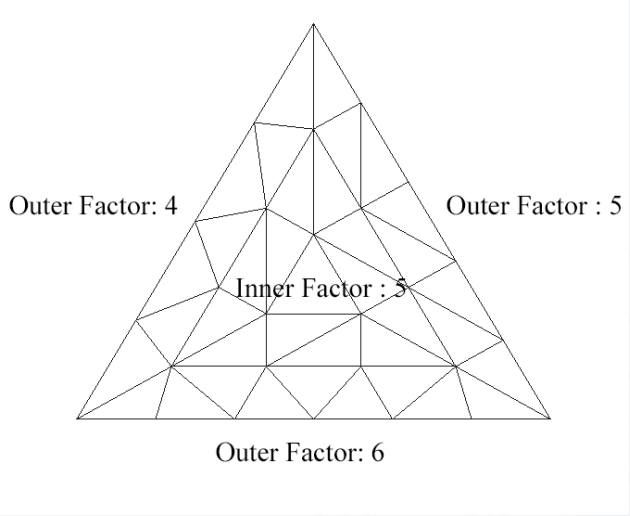
Tessellation Control Shader

曲面细分

Tessellation Evaluation Shader



TessFatcor[0] = 2
TessFatcor[1] = 2
TessFatcor[2] = 2
TessFatcor[3] = 2



# Tessellation Shader



- Semantic
  - GL_TessLevelInner
    - 中间的细分系数
  - GL_TessLevelOuter
    - 三条边的细分系数
  - GL_TesssCoord
    - 细分得到的重心坐标
- InputAssembly
  - VK_PRIMITIVE_TOPOLOGY_PATCH_LIST
- 计算频次
  - Tessellation Control Shader
    - 每个点一次
    - Tessllation Factors?
  - Tessllation Evaluation Shader
    - 每一个细分点一次

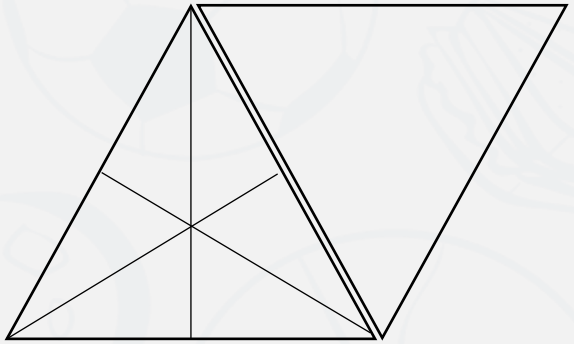
```
#version 450      saschawillems, 7 years ago • Added Vulkan examples sources!

layout (vertices = 3) out;

layout (location = 0) in vec3 inNormal[];
layout (location = 1) in vec2 inUV[];

layout (location = 0) out vec3 outNormal[3];
layout (location = 1) out vec2 outUV[3];

void main(void)
{
    if (gl_InvocationID == 0)
    {
        gl_TessLevelInner[0] = 2.0;
        gl_TessLevelOuter[0] = 2.0;
        gl_TessLevelOuter[1] = 2.0;
        gl_TessLevelOuter[2] = 2.0;
    }

    gl_out[gl_InvocationID].gl_Position = gl_in[gl_InvocationID].gl_Position;
    outNormal[gl_InvocationID] = inNormal[gl_InvocationID];
    outUV[gl_InvocationID] = inUV[gl_InvocationID];
}
```

Tessellation Control Shader

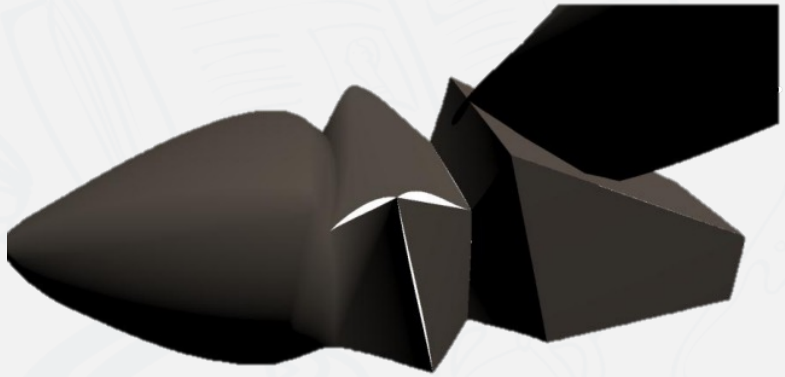
```
layout (triangles, fractional_odd_spacing, cw) in;

layout (binding = 1) uniform UBO
{
    mat4 projection;
    mat4 model;
    float tessAlpha;
} ubo;

layout (location = 0) in vec3 inNormal[];
layout (location = 1) in vec2 inUV[];

layout (location = 0) out vec3 outNormal;
layout (location = 1) out vec2 outUV;

void main(void)
{
    gl_Position = (gl_TessCoord.x * gl_in[0].gl_Position) +
                  (gl_TessCoord.y * gl_in[1].gl_Position) +
                  (gl_TessCoord.z * gl_in[2].gl_Position);
    gl_Position = ubo.projection * ubo.model * gl_Position;

    outNormal = gl_TessCoord.x*inNormal[0] + gl_TessCoord.y*inNormal[1] + gl_TessCoord.z*inNormal[2];
    outUV = gl_TessCoord.x*inUV[0] + gl_TessCoord.y*inUV[1] + gl_TessCoord.z*inUV[2];
}
```

Tessellation Evaluation Shader

- Semantic
- InputAssembly
- 计算频次
- 用途
  - 更平滑的几何
    - PNTriangles
  - 增加几何细节
    - Displacement Map

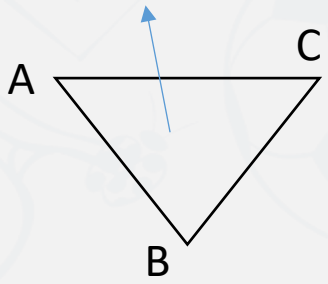


Edge Tessfactor



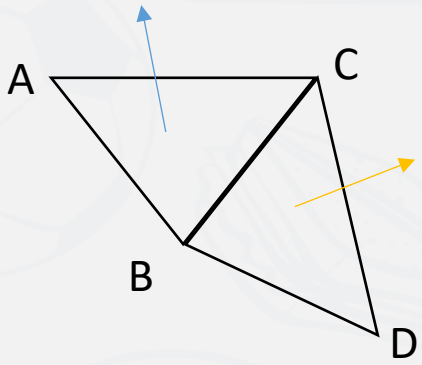
Vertex Displacement 不匹配
(Normal, Offset)

# Shared Vertex Data

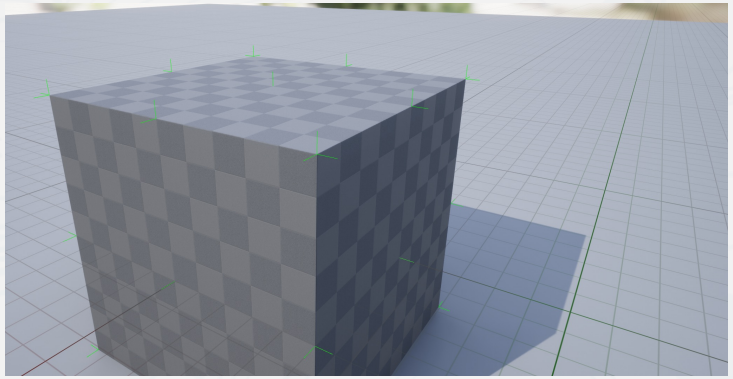



$$F_n = cross(AB, AC)$$
$$B_n = F_n$$

Smoothed Vertex Normal




$$B_n = \text{normalize}(F_{n1} + F_{n2})$$



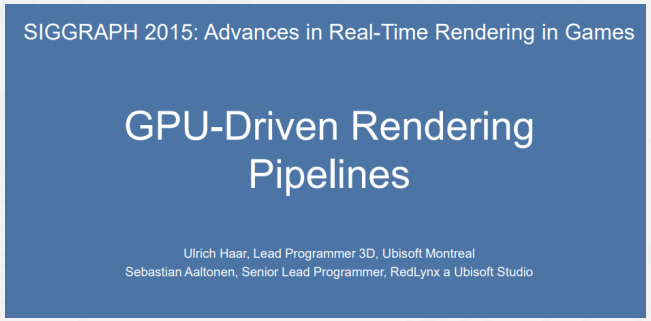
Hard Edge
UV Atlas Edge

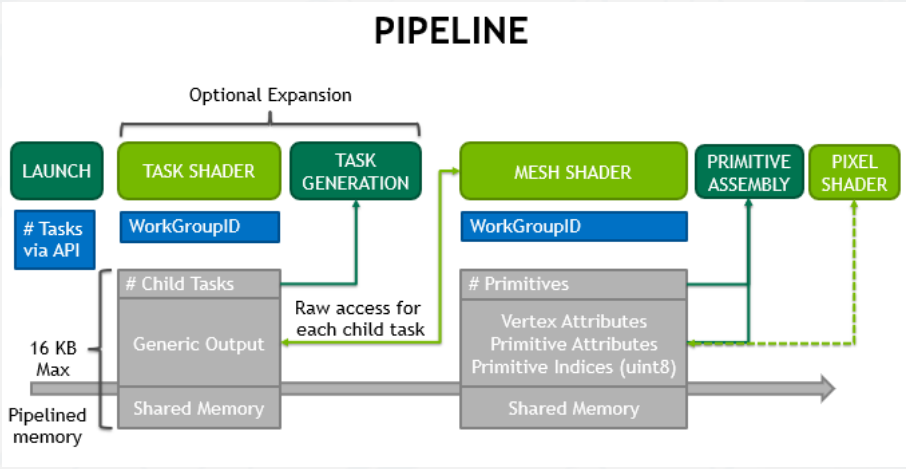
Dominant Vertex

# Mesh Shader

# 几何管线

- Tessellation
  - Fixed pattern
- 背景
  - Compute Shader
  - GPU driven rendering pipeline
- Mesh Shader
  - Task Shader
  - Mesh Shader



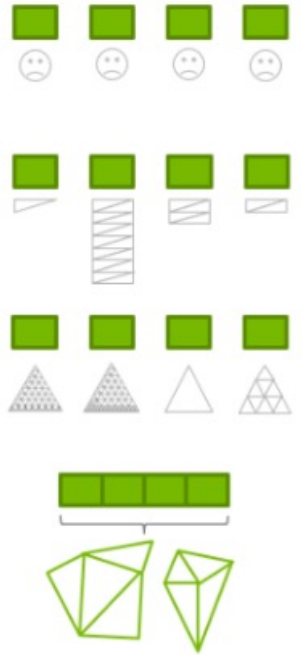



# Overall

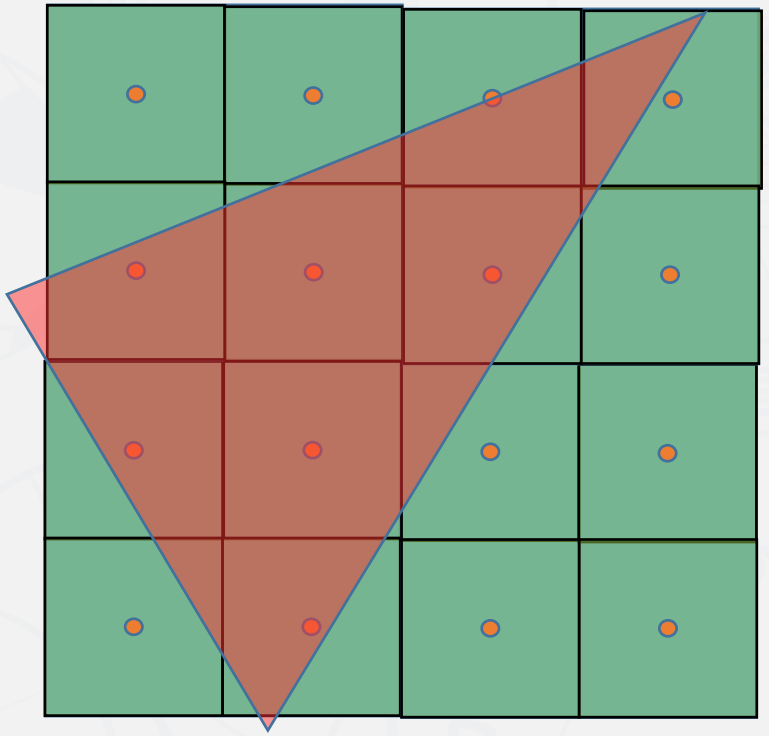
# 几何管线

- 计算粒度的不同
- 并行度的不同
- 找到合适的应用

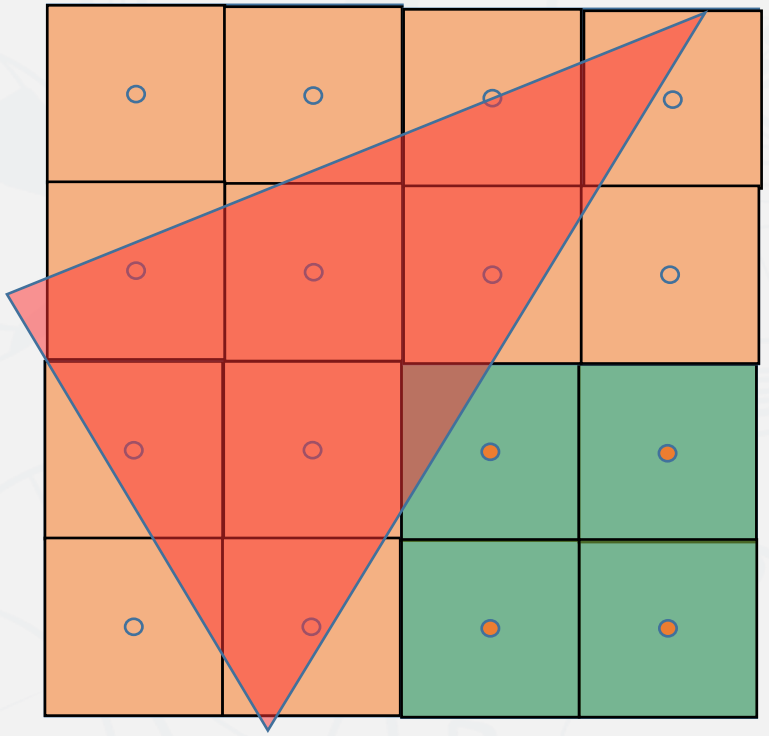
| Shader | | Thread Mapping | Topology |
|---|---|---|---|
| Vertex Shader | No access to connectivity | 1 Vertex | No influence |
| Geometry Shader | Variable output doesn't fit HW well | 1 Primitive / 1 Output Strip | Triangle Strips |
| Tessellation Shader | Fixed-function topology | 1 Patch / 1 Evaluated Vertex | Fast Patterns |
| Mesh Shader | Compute shader features | Flexible | Flexible within work group allocation |



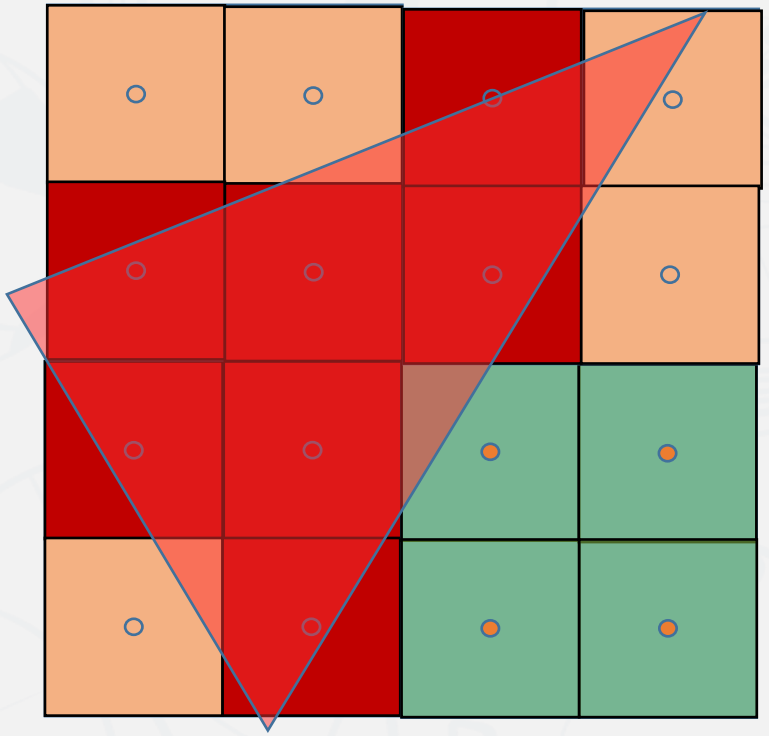
# 着色管线

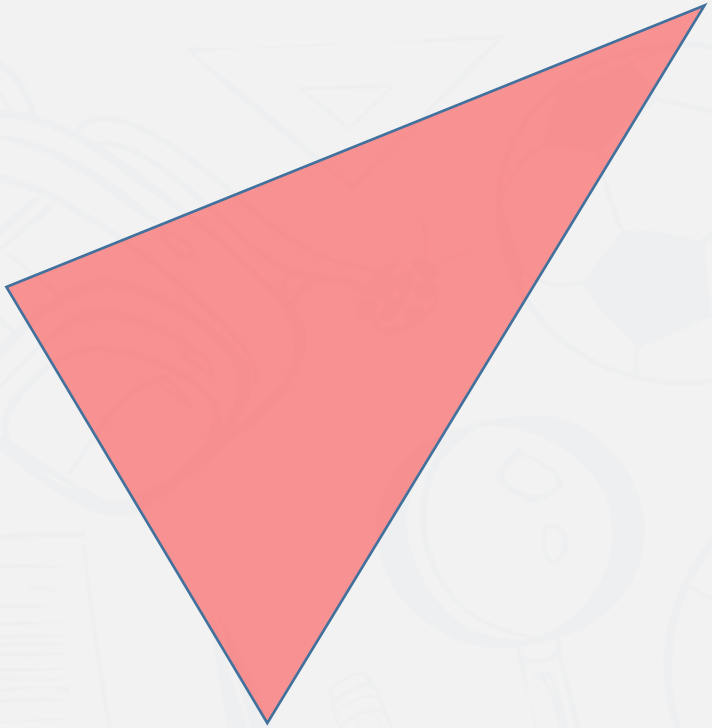


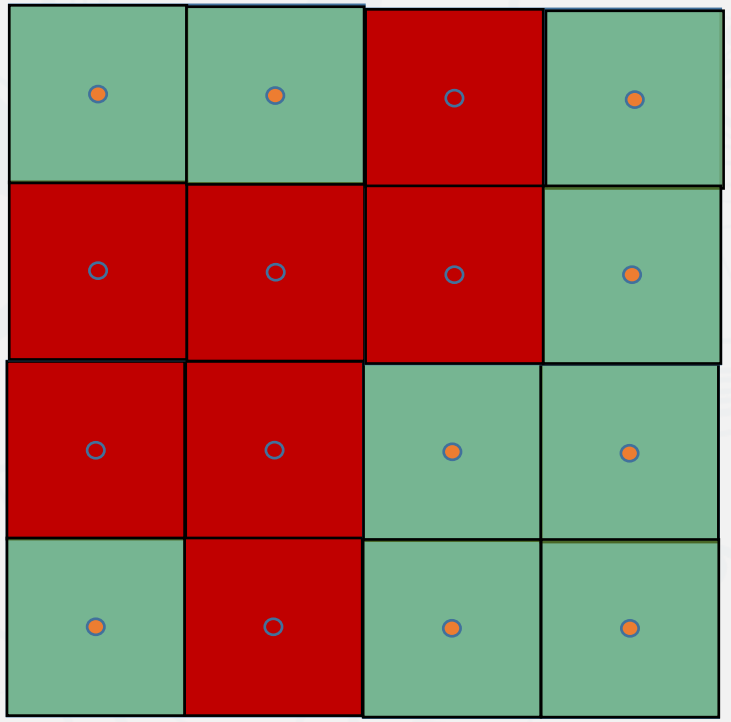
哪些像素会执行Pixel Shader?
哪些像素会写入？



以Quad为单位执行，Why?

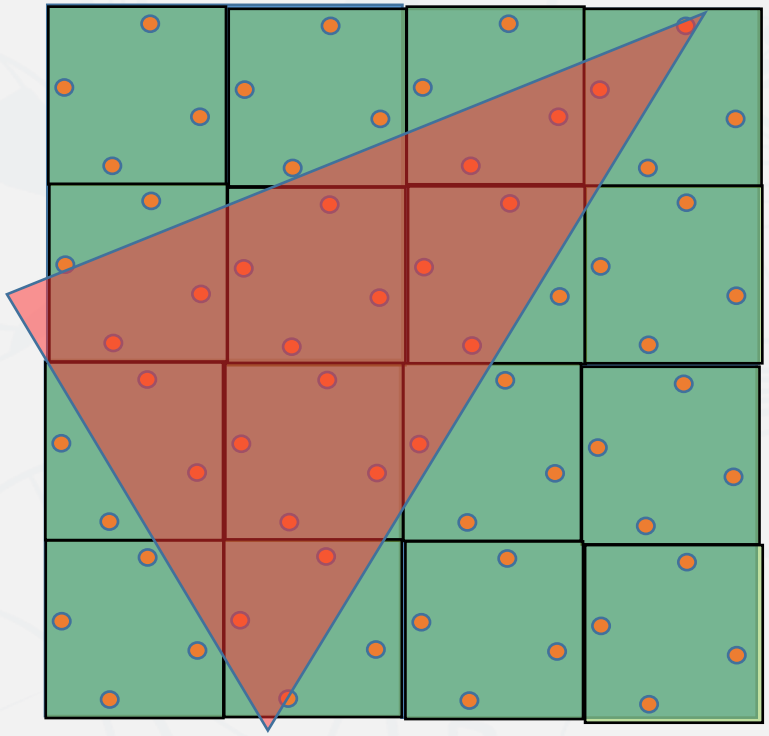




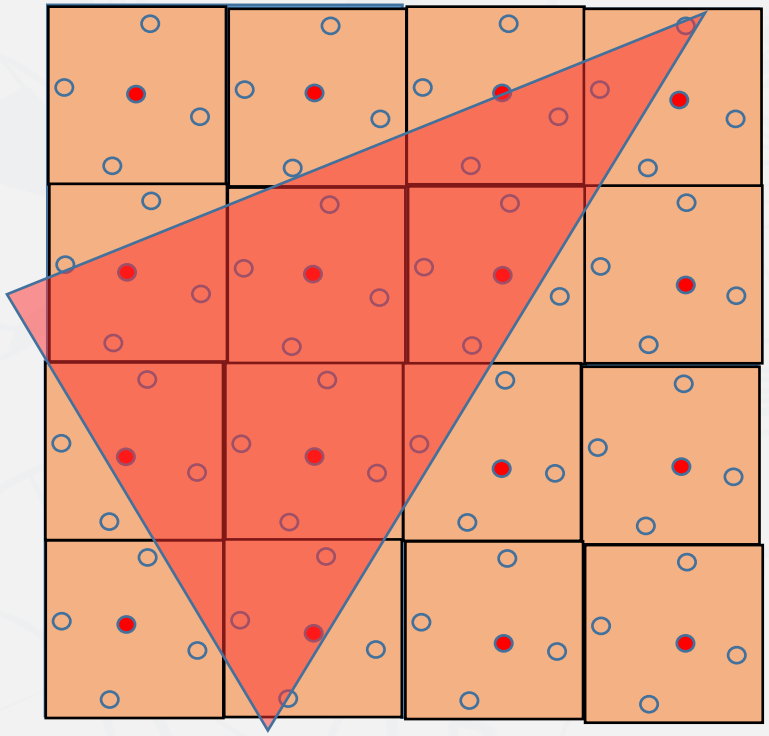


锯齿！

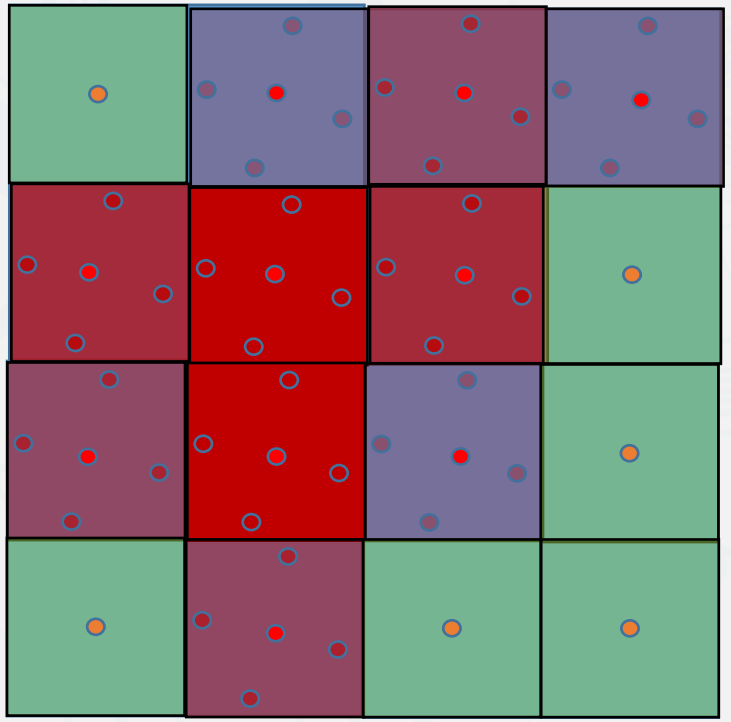
# MSAA(Multiple Sample Anti-Aliasing)



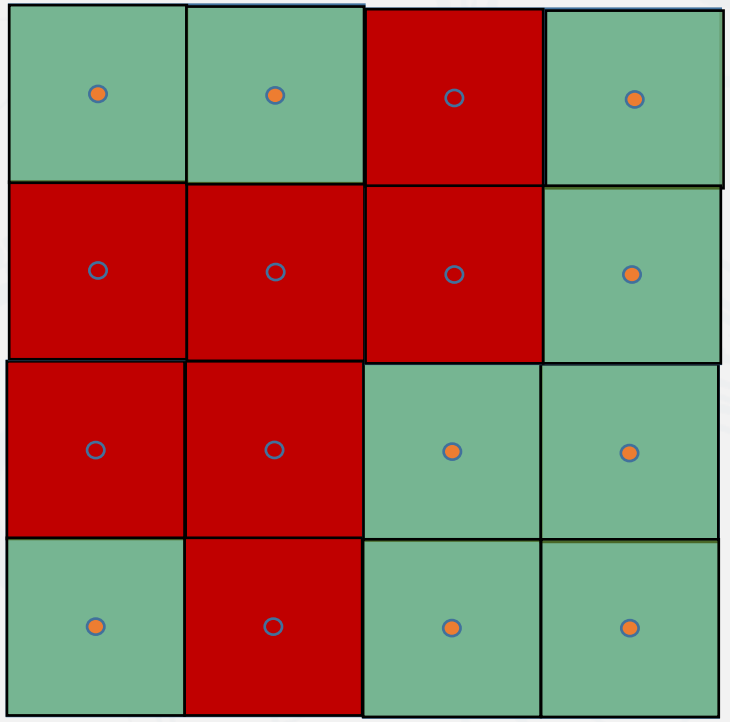
增加可见性样本.
获得三角形对这个像素的覆盖比例(Coverage
Semantic: SV_Coverage



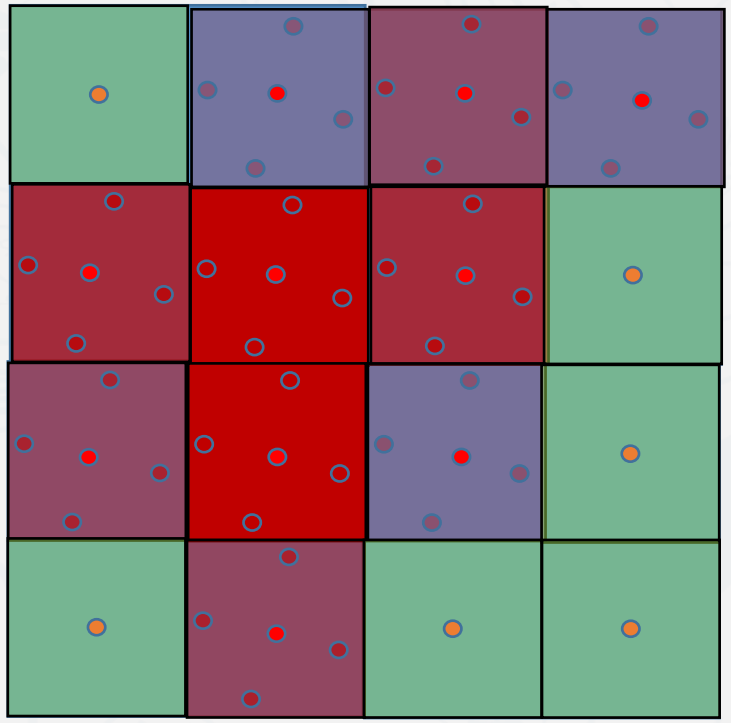
在屏幕中心计算颜色像素颜色C.



在屏幕中心计算颜色像素颜色C.
Resolve:写入C * Coverage



普通Rasterization
Depth样本: 16
Pixel Shader 执行次数: 12次
写入颜色个数： 7

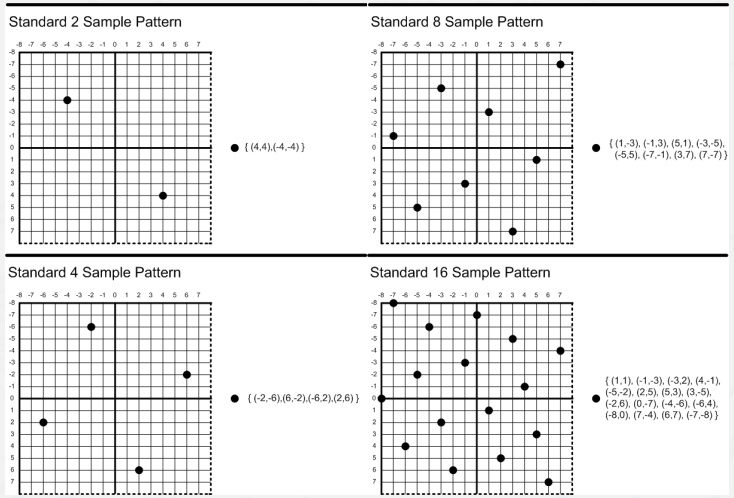


MSAA(4x)
深度样本: 16 * 4
Pixel Shader 执行次数： 16次
写入颜色个数: 10

# MSAA



- MSAA
  - 更多的可见性样本点
  - 一个像素一次Shading
    - 多个三角形覆盖同一个像素？
- 重要接口
  - VkImageCreateInfo::samples
  - VkPipelineMultisampleStateCreateInfo::samples & many more
    - SampleShading
    - SampleMask

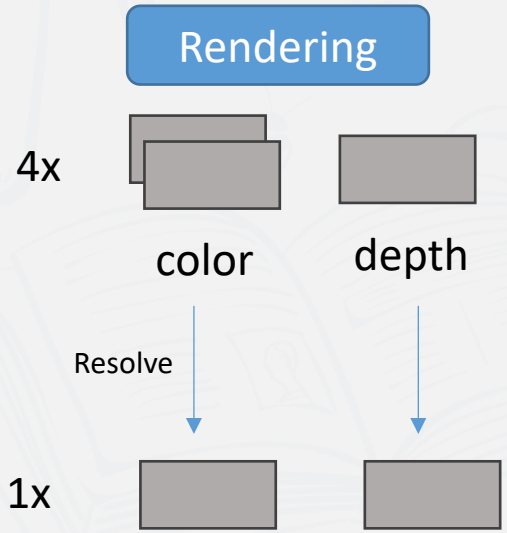



```
// Provided by VK_VERSION_1_0
typedef struct VkPipelineMultisampleStateCreateInfo {
    VkStructureType                         sType;
    const void*                             pNext;
    VkPipelineMultisampleStateCreateFlags   flags;
    VkSampleCountFlagBits                   rasterizationSamples;
    VkBool32                                sampleShadingEnable;
    float                                   minSampleShading;
    const VkSampleMask*                     pSampleMask;
    VkBool32                                alphaToCoverageEnable;
    VkBool32                                alphaToOneEnable;
} VkPipelineMultisampleStateCreateInfo;
```

# MSAA



- MSAA
- 重要接口
- **Resolve**
  - 硬件自动Resolve
    - Average
    - Depth: min/max/sample0
  - 手动 VKcmdResolveImage
  - 手动写shader
    - Texture2DMS




```
// Provided by VK_VERSION_1_0
typedef struct VkSubpassDescription {
    VkSubpassDescriptionFlags       flags;
    VkPipelineBindPoint             pipelineBindPoint;
    uint32_t                        inputAttachmentCount;
    const VkAttachmentReference*    pInputAttachments;
    uint32_t                        colorAttachmentCount;
    const VkAttachmentReference*    pColorAttachments;
    const VkAttachmentReference*    pResolveAttachments;
    const VkAttachmentReference*    pDepthStencilAttachment;
    uint32_t                        preserveAttachmentCount;
    const uint32_t*                 pPreserveAttachments;
} VkSubpassDescription;
```

```
// Provided by VK_VERSION_1_2
typedef struct VkSubpassDescriptionDepthStencilResolve {
    VkStructureType                 sType;
    const void*                     pNext;
    VkResolveModeFlagBits           depthResolveMode;
    VkResolveModeFlagBits           stencilResolveMode;
    const VkAttachmentReference2*   pDepthStencilResolveAttachment;
} VkSubpassDescriptionDepthStencilResolve;
```

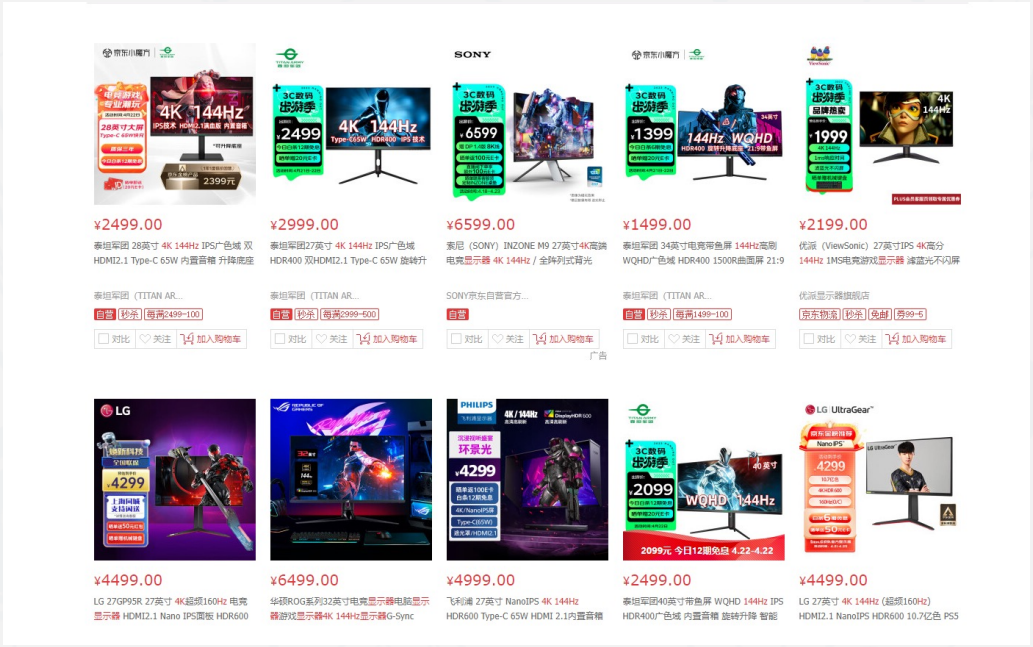
```
// Provided by VK_VERSION_1_3
typedef struct VkRenderingAttachmentInfo {
    VkStructureType         sType;
    const void*             pNext;
    VkImageView             imageView;
    VkImageLayout           imageLayout;
    VkResolveModeFlagBits   resolveMode;
    VkImageView             resolveImageView;
    VkImageLayout           resolveImageLayout;
    VkAttachmentLoadOp      loadOp;
    VkAttachmentStoreOp     storeOp;
    VkClearValue            clearValue;
} VkRenderingAttachmentInfo;
```

# Variable Shading Rate

# 着色管线

*分辨率涨的太快了！*
*我们需要少画一些像素*

90’s Gameboy, 160x144 pixels,
2*10^5 FLOPS



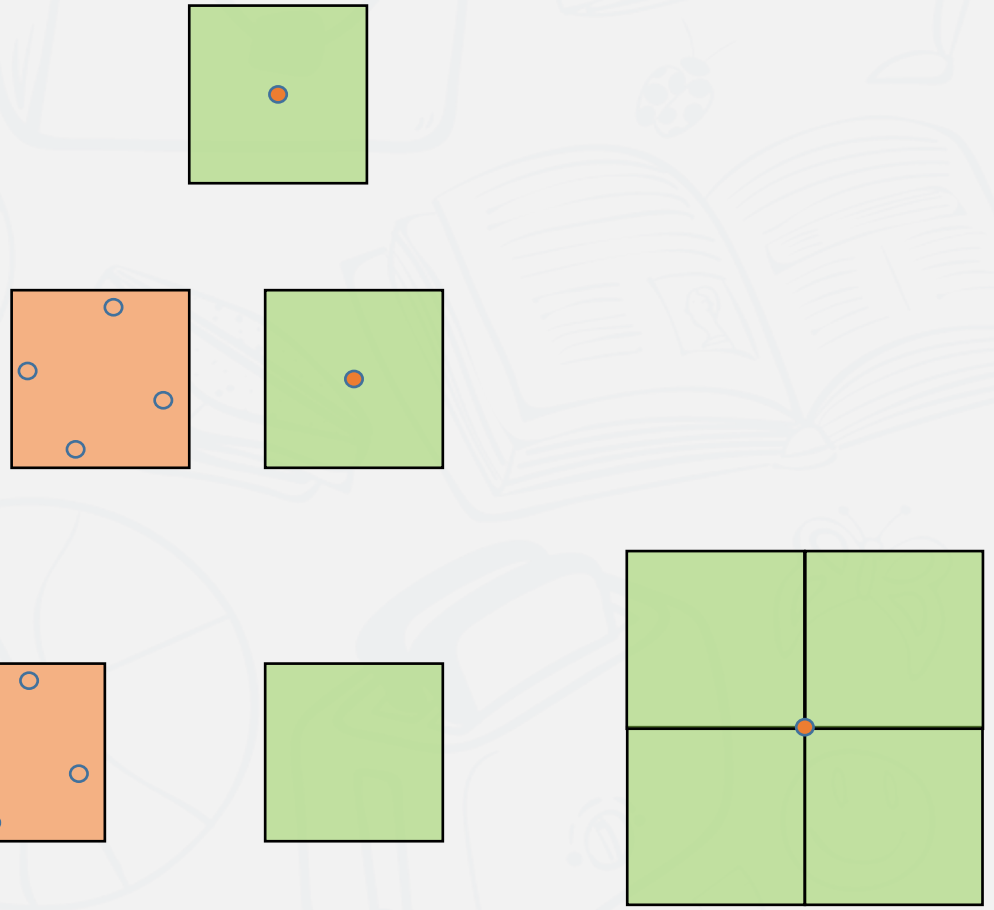
# Variable Shading Rate-原理



Pixel Shading

MSAA

Variable Shading Rate

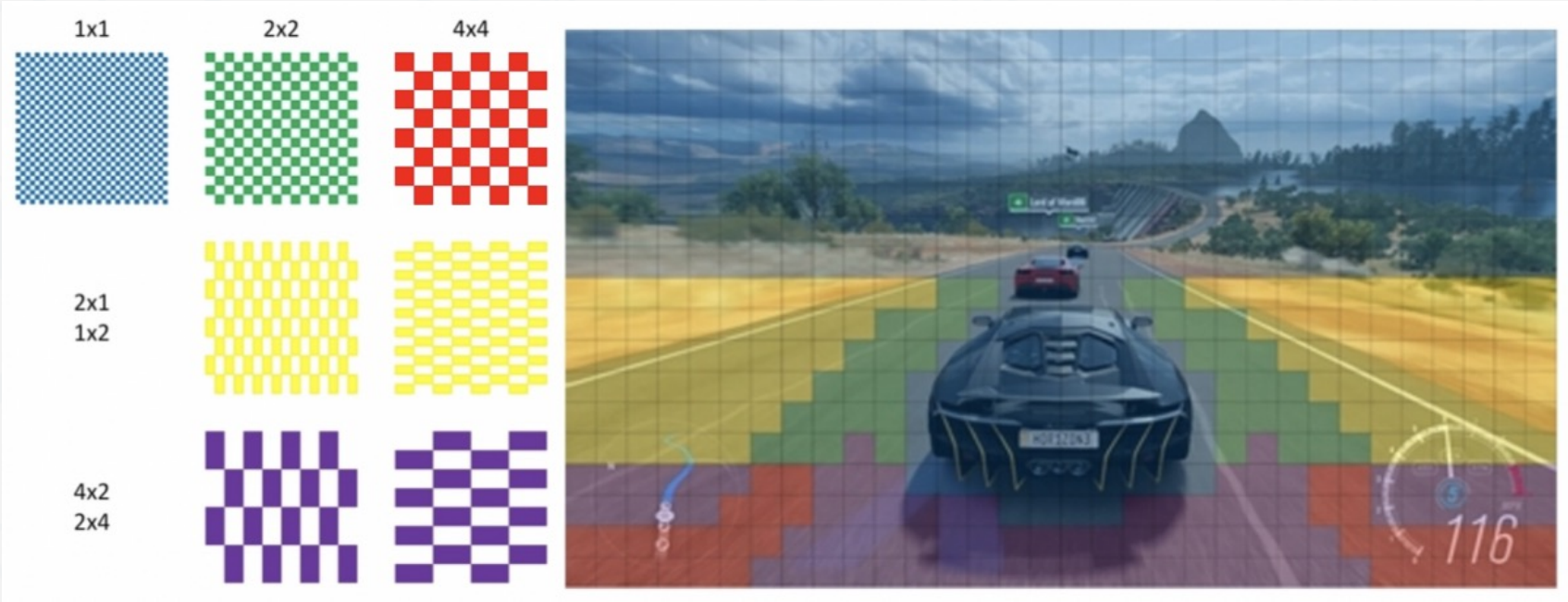


在比像素更小的尺度上
进行可见性/着色计算

在比像素更大的尺度上
进行着色计算

# Variable Shading Features

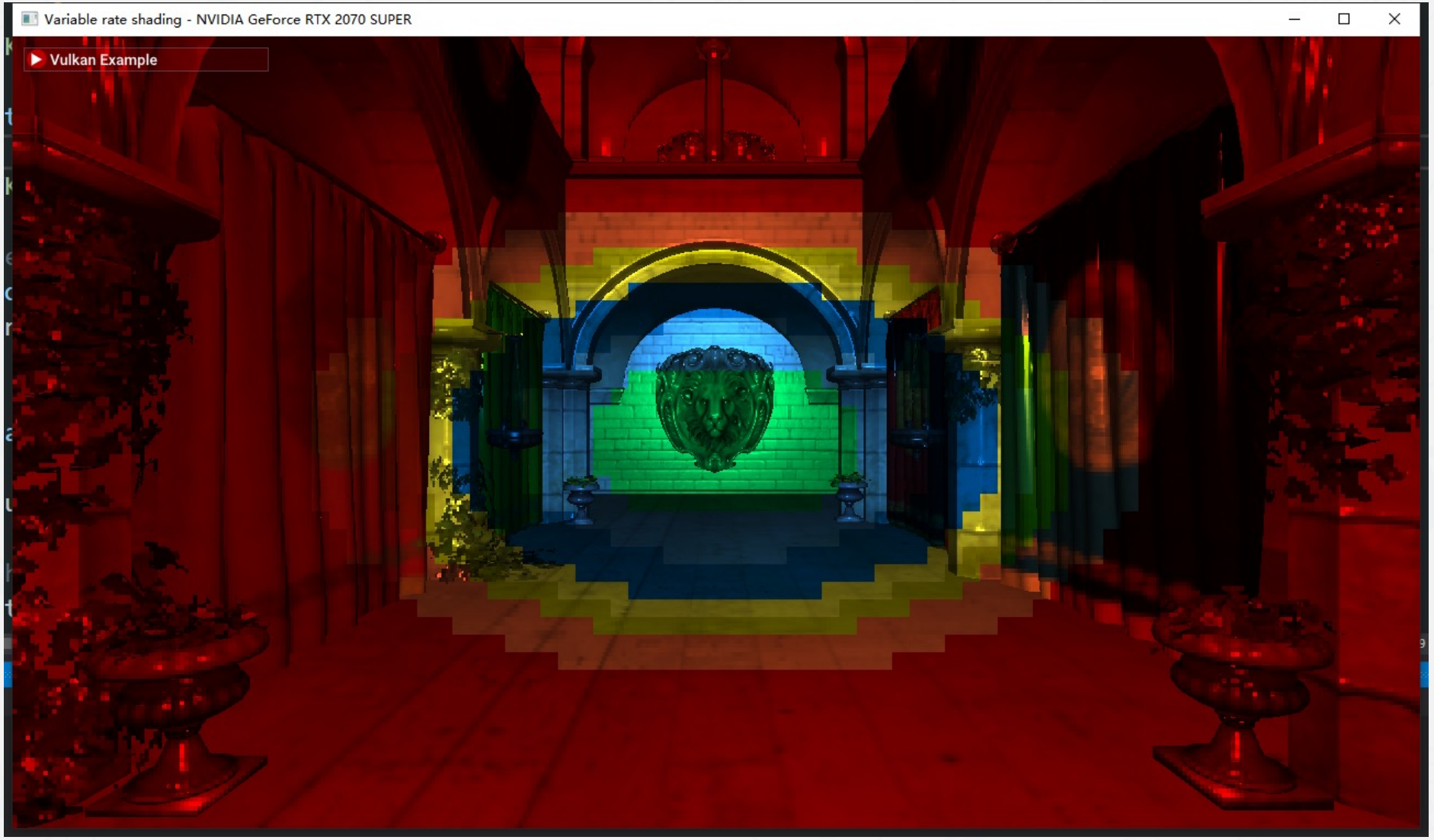
- Per-Draw Basis
  - Object Tag
- Shading Rate Image
  - Static Shading rate
  - Adaptive Shading rate
- Combine with MSAA




```
// Provided by VK_NV_shading_rate_image
typedef struct VkPipelineViewportShadingRateImageStateCreateInfoNV {
    VkStructureType                 sType;
    const void*                     pNext;
    VkBool32                        shadingRateImageEnable;
    uint32_t                        viewportCount;
    const VkShadingRatePaletteNV*   pShadingRatePalettes;
} VkPipelineViewportShadingRateImageStateCreateInfoNV;
```

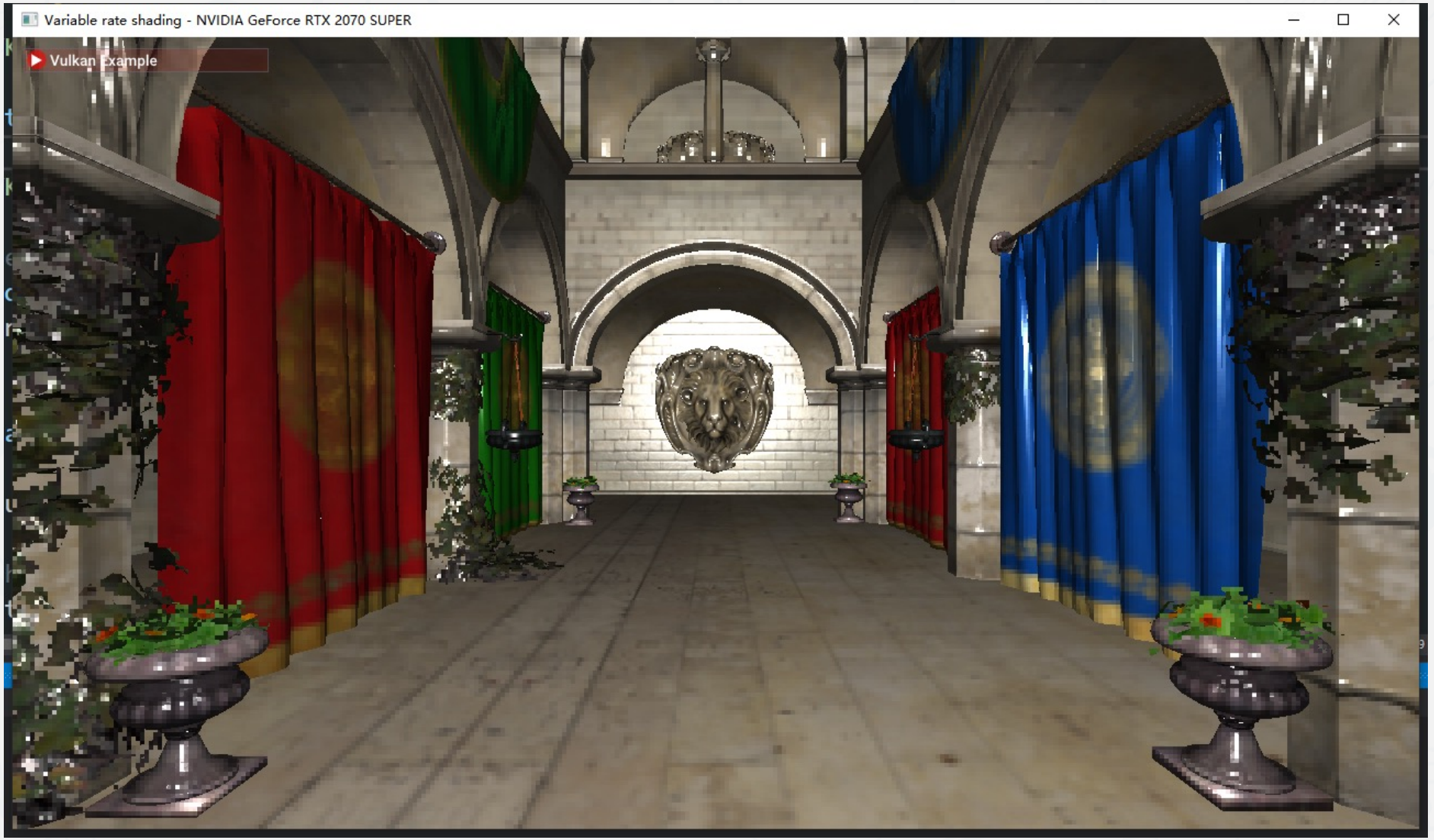
# Static Shading Rate





# 着色管线

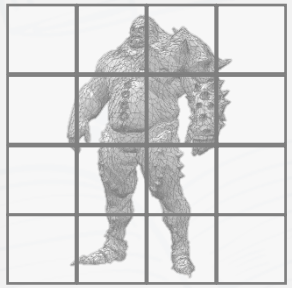
如何让图像质量降得少一点

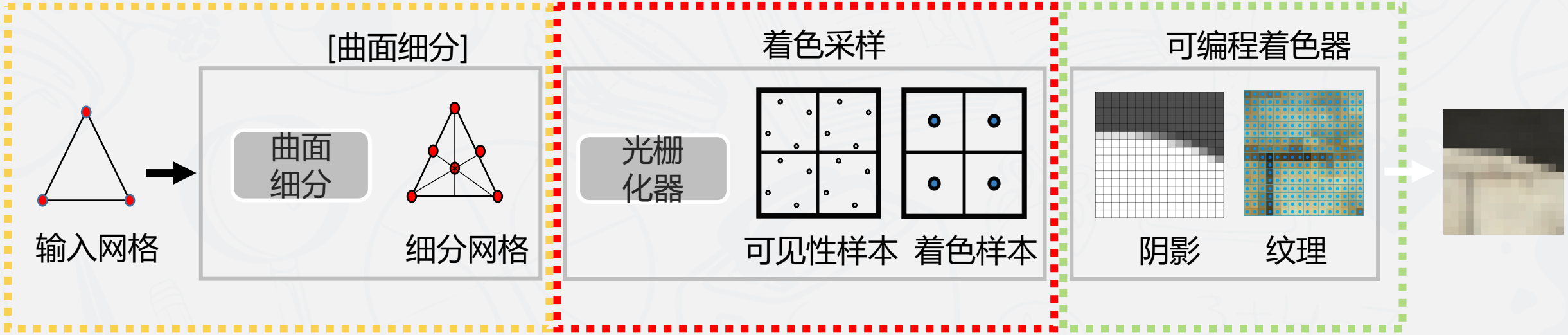


# 总结一下

## 着色管线





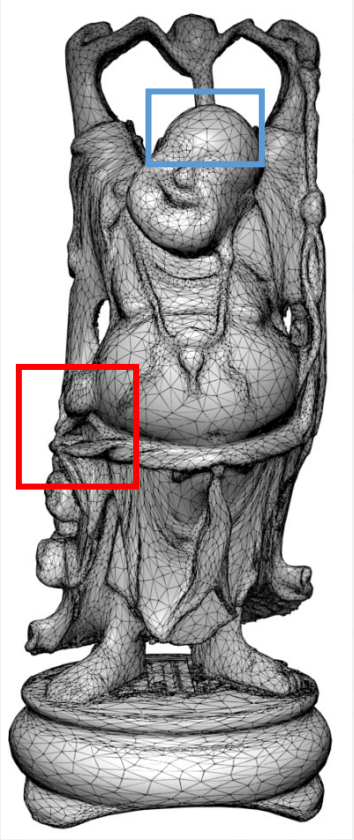




*怎么利用这些管线做优化呢？*

# 多频率着色优化

# 着色频率分析-几何

- 高频
  - 法线变化大
  - 几何细节多
- 低频区域
  - 平坦
  - 光滑

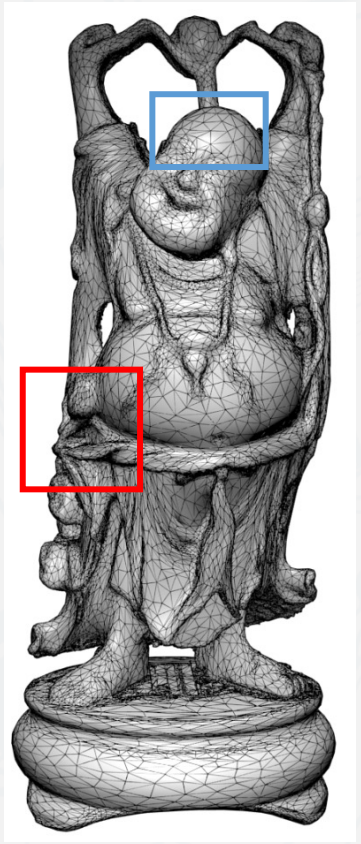


Dense mesh

# 几何优化

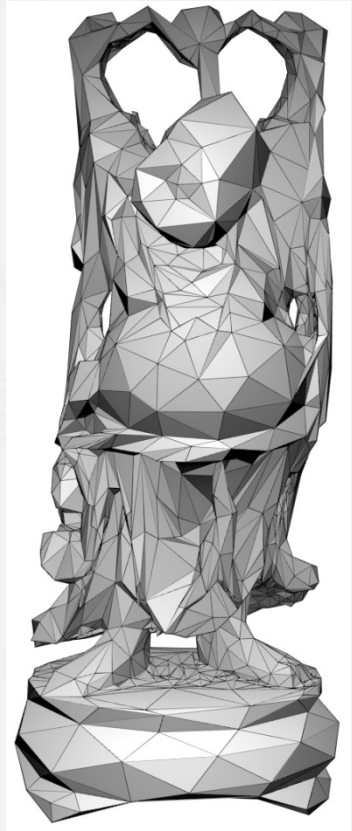


- 高低频分解
  - 简化
  - 烘焙
- 绘制
  - Displacement map
  - Adaptive tessellation
  - PN-triangle



Simplification /Baking

Rendering



Displacement map

Normal Map

Dense mesh

Base Mesh

***Low frequency*** ***High frequency***

# 着色频率分析-屏幕信号



- 高频
  - 光照变化剧烈
  - 纹理细节丰富
- 低频
  - 光照平缓
  - 纹理较为一致
  - 比较”远”

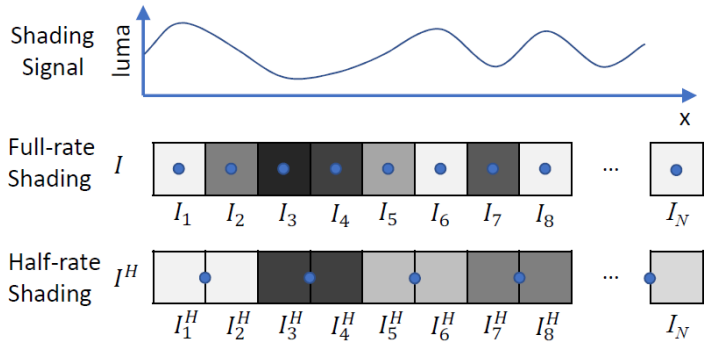
# 着色频率分析-屏幕信号

## 多频率着色优化

**Visually Lossless Content and Motion Adaptive Shading in Games**

LEI YANG, DMITRY ZHDAN, EMMETT KILGARIFF, ERIC B. LUM, YUBO ZHANG, MATTHEW JOHNSON, and HENRIK RYDGÅRD, NVIDIA Corporation


(a) Full rate (stationary) (b) Adaptive (stationary) (c) Shading Rate (stationary) (d) Adaptive (in motion) (e) Shading Rate (in motion)




$$
\begin{aligned}
\mathcal{E}(I, I^H) &= \left\| I - I^H \right\|_2 \\
&= \sqrt{\frac{1}{N}\sum_{j=1}^{N/2}\left(I_{2j-1} - I^H_{2j-1}\right)^2 + \left(I_{2j} - I^H_{2j}\right)^2} \\
&= \sqrt{\frac{1}{N}\sum_{j=1}^{N/2}\left(I_{2j-1} - \frac{I_{2j-1}+I_{2j}}{2}\right)^2 + \left(I_{2j} - \frac{I_{2j-1}+I_{2j}}{2}\right)^2} \\
&= \sqrt{\frac{1}{N}\sum_{j=1}^{N/2} 2\left(\frac{I_{2j}-I_{2j-1}}{2}\right)^2} \\
&\approx \sqrt{\frac{1}{N-1}\sum_{i=2}^{N}\left(\frac{I_i - I_{i-1}}{2}\right)^2}.
\end{aligned}
$$

# 着色频率分析-屏幕信号

$$
\begin{aligned}
\mathcal{E}(I, I^H) &= \left\| I - I^H \right\|_2 \\
&= \sqrt{\frac{1}{N}\sum_{j=1}^{N/2}\left(I_{2j-1} - I^H_{2j-1}\right)^2 + \left(I_{2j} - I^H_{2j}\right)^2} \\
&= \sqrt{\frac{1}{N}\sum_{j=1}^{N/2}\left(I_{2j-1} - \frac{I_{2j-1}+I_{2j}}{2}\right)^2 + \left(I_{2j} - \frac{I_{2j-1}+I_{2j}}{2}\right)^2} \\
&= \sqrt{\frac{1}{N}\sum_{j=1}^{N/2} 2\left(\frac{I_{2j}-I_{2j-1}}{2}\right)^2} \\
&\approx \sqrt{\frac{1}{N-1}\sum_{i=2}^{N}\left(\frac{I_i - I_{i-1}}{2}\right)^2}.
\end{aligned}
$$

频率分析

$$\mathcal{E}(I, I^Q) \approx k \cdot \mathcal{E}(I, I^H).$$

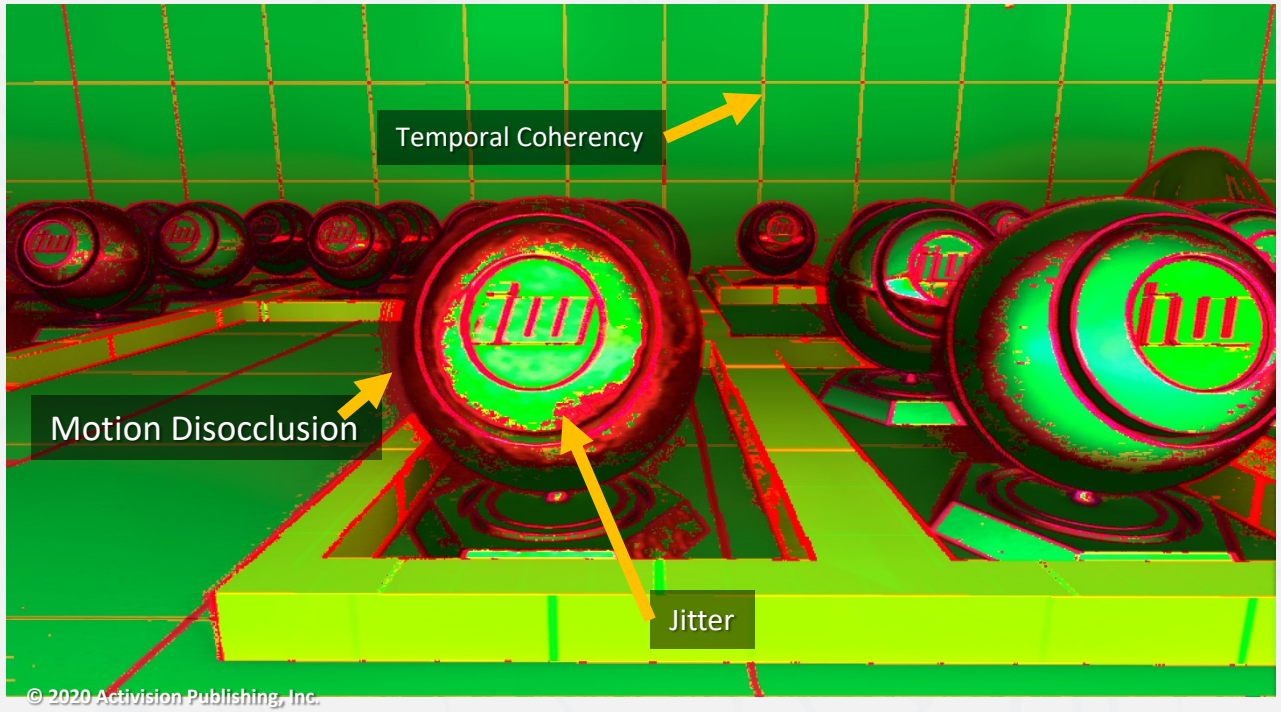
着色率决策

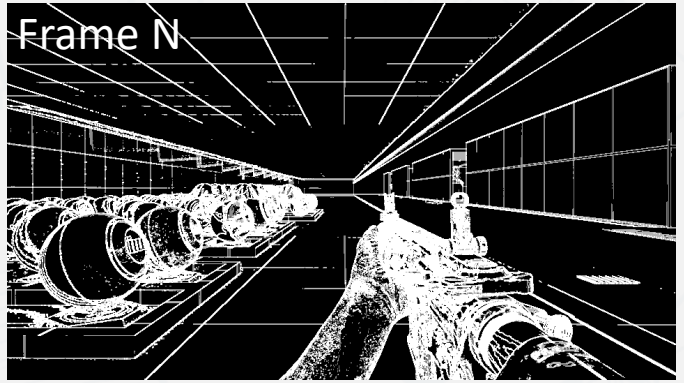
$$\tau_I = t \cdot (I_{\text{avg}} + l),$$

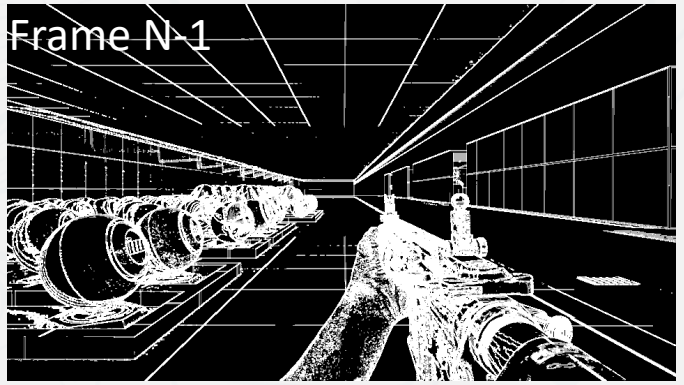
$$S_I = \begin{cases} \text{Full}, & \text{if } \mathcal{E}(I, I^H) \geq \tau_I; \\ \text{Quarter}, & \text{if } k \cdot \mathcal{E}(I, I^H) < \tau_I; \\ \text{Half}, & \text{otherwise.} \end{cases}$$

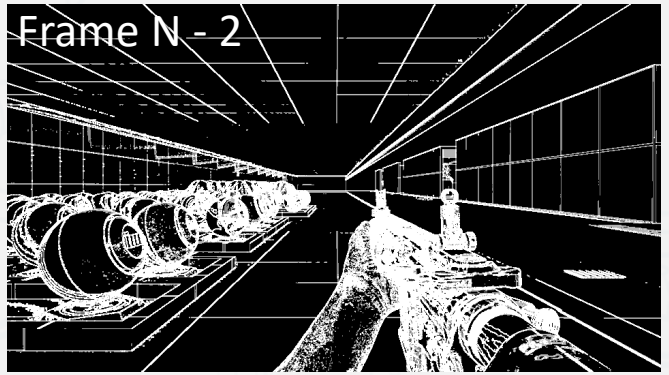
# 着色频率分析-屏幕信号

- COD 20



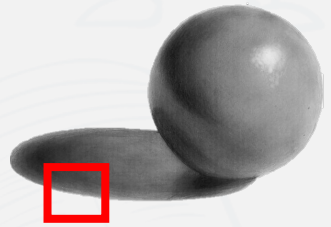



# 着色频率分析-着色项



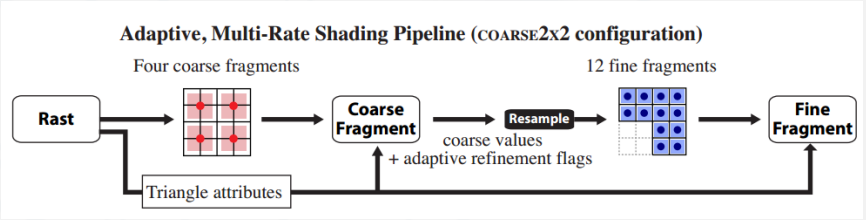
- 高频
  - 高光的Lobe中心
  - 影音的半影去
- 低频
  - 漫反射
  - 远离高光中心
  - 完全被影子遮敝

Diffuse



Shadow
(umbra region)

Specular
( far from reflection
low freq. region)




He et al. 2014

# 着色频率分析-分辨率

*现代引擎有很多可以调着色分辨率的地方*

AO

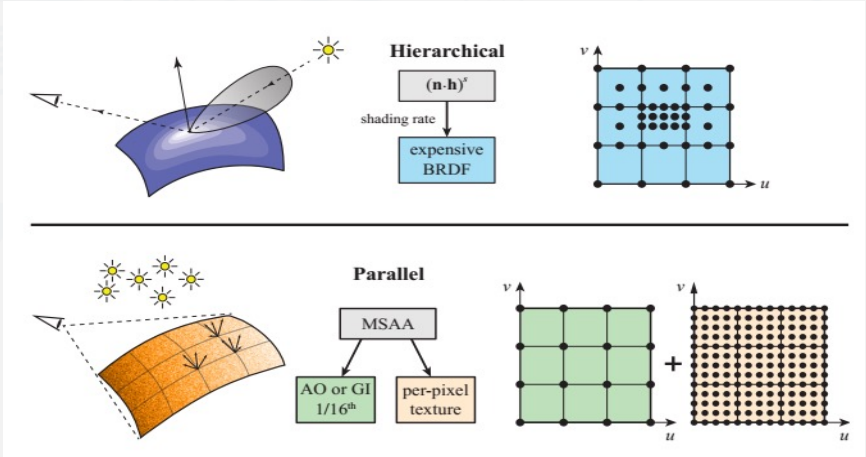
Global Illumination

# 着色频率分析-着色方程



- Move to Vertex/Tessllation
  - Per-Vertex lighting
  - Shader simplifictation
- Multi-rate shading
  - Coarse Pixel shader
  - Object-Multi Rate shading


Hu et al. 2022




Clarberg et al. 2014

- 绘制管线本身提供了不同计算粒度，且越来越自由
  - 几何管线
  - 着色管线
- 正确理解着色信号的产生过程，做出合理的优化选择
  - 几何过于复杂
  - 像素过多
  - 着色信号过于高频

# 作业

- Extend the variable shading rate example
  - Content Adaptive
  - Motion Adaptive

**Visually Lossless Content and Motion Adaptive Shading in Games**

LEI YANG, DMITRY ZHDAN, EMMETT KILGARIFF, ERIC B. LUM, YUBO ZHANG, MATTHEW JOHNSON, and HENRIK RYDGÅRD, NVIDIA Corporation


(a) Full rate (stationary) (b) Adaptive (stationary) (c) Shading Rate (stationary) (d) Adaptive (in motion) (e) Shading Rate (in motion)