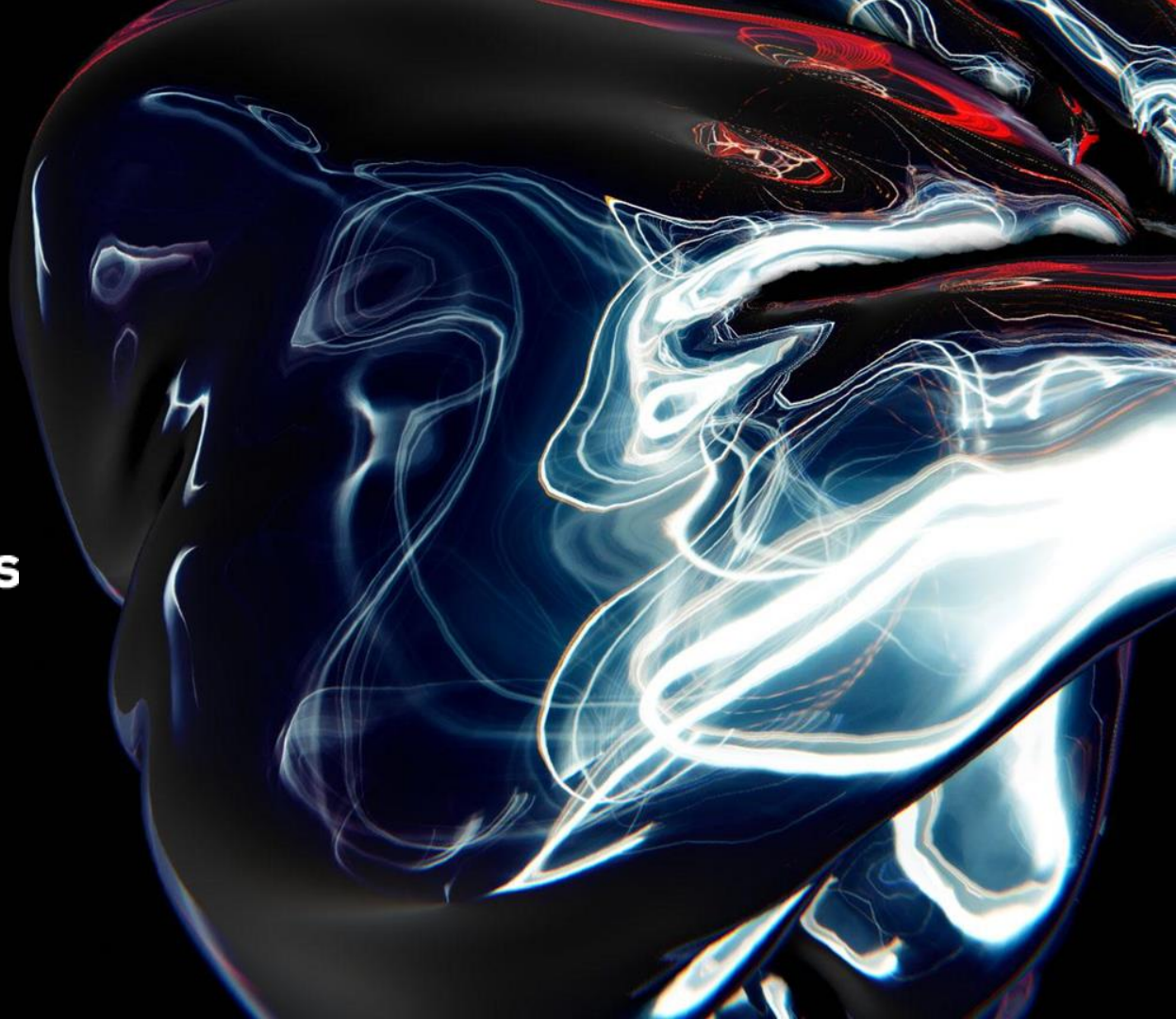
unity

Digital Dragons



Leveraging Ray Tracing Hardware Acceleration In Unity

Digital Dragons ‘19

Anis Benyoub
Graphics Engineer

@Auzaiffe

High Definition Render Pipeline (HDRP)

- Scriptable Render Pipeline
- Physically Based Lighting
- Modern design
- Designed for AAA productions

Book of the dead

https://github.com/Unity-Technologies/ScriptableRenderPipeline

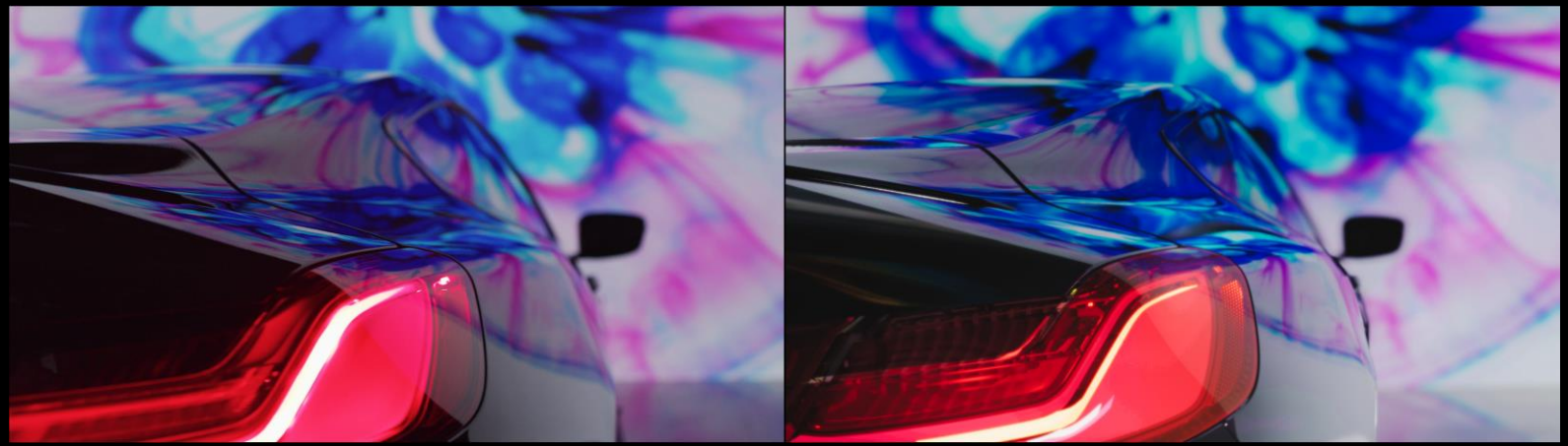
HDRP

The Heretic

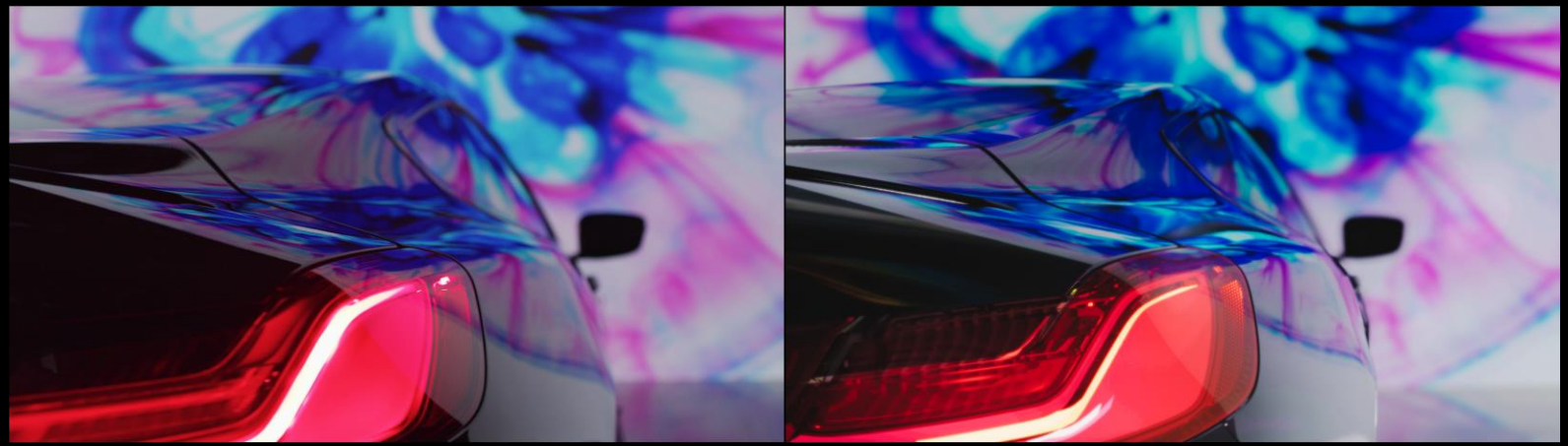
Harold Halibut

Motivation

Ray Tracing Effects



Ray Tracing Effects



Left: real footage. Right rendered with Unity

Ray Tracing Effects

Ambient Occlusion

Ray Tracing Effects

Ambient Occlusion

Indirect Diffuse (GI)

Ray Tracing Effects

Ambient Occlusion

Recursive Tracing

Indirect Diffuse

Ray Tracing Effects

Ambient Occlusion

Recursive Tracing

Indirect Diffuse

Indirect Specular
(Reflections)

Ray Tracing Effects

Ambient Occlusion

Recursive Tracing

Indirect Diffuse

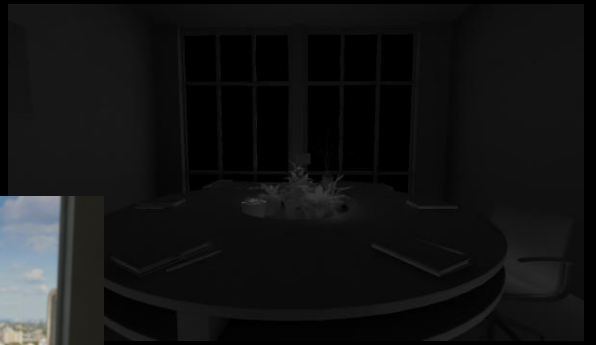
Indirect Specular

Stochastic Area Shadows

Ray Tracing Effects

Ambient Occlusion

Recursive Tracing



Indirect Diffuse

Indirect Specular

Stochastic Area Shadows

Ray Tracing Effects

Ambient Occlusion

Indirect Diffuse

Hybrid (Rasterization/Ray Tracing) Renderer

Recursive Tracing

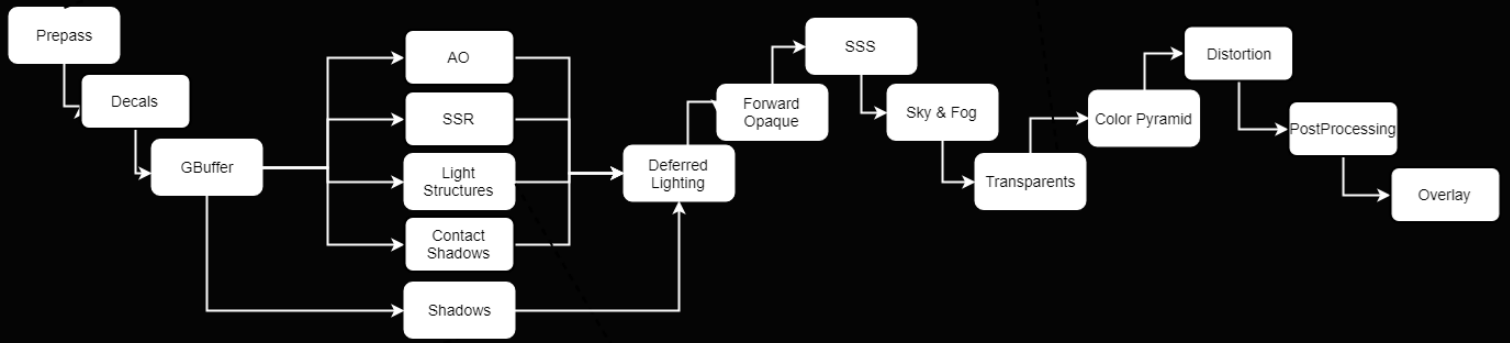
Indirect Specular

Stochastic Area Shadows

HDRP Render Graph

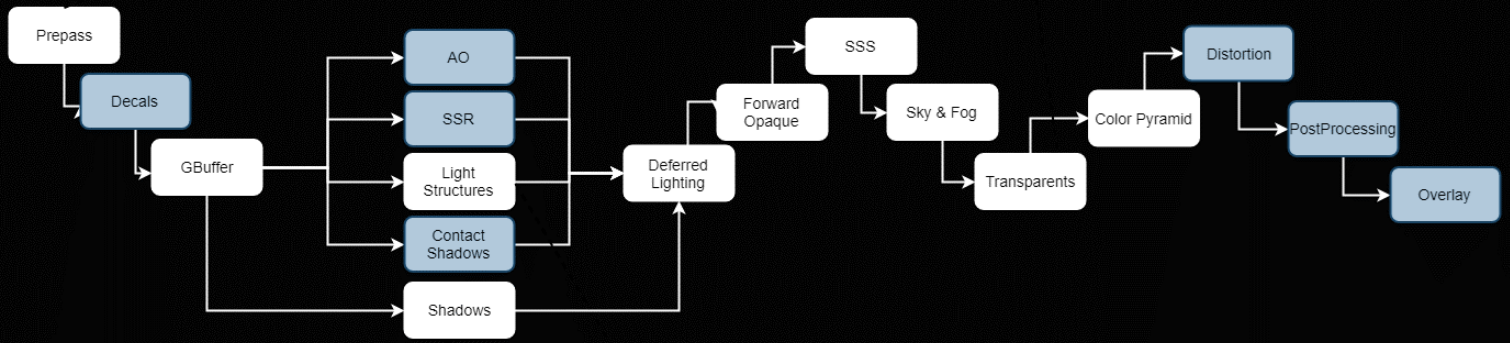
A wild digital dragon appears!

Render Graph - Deferred/Forward



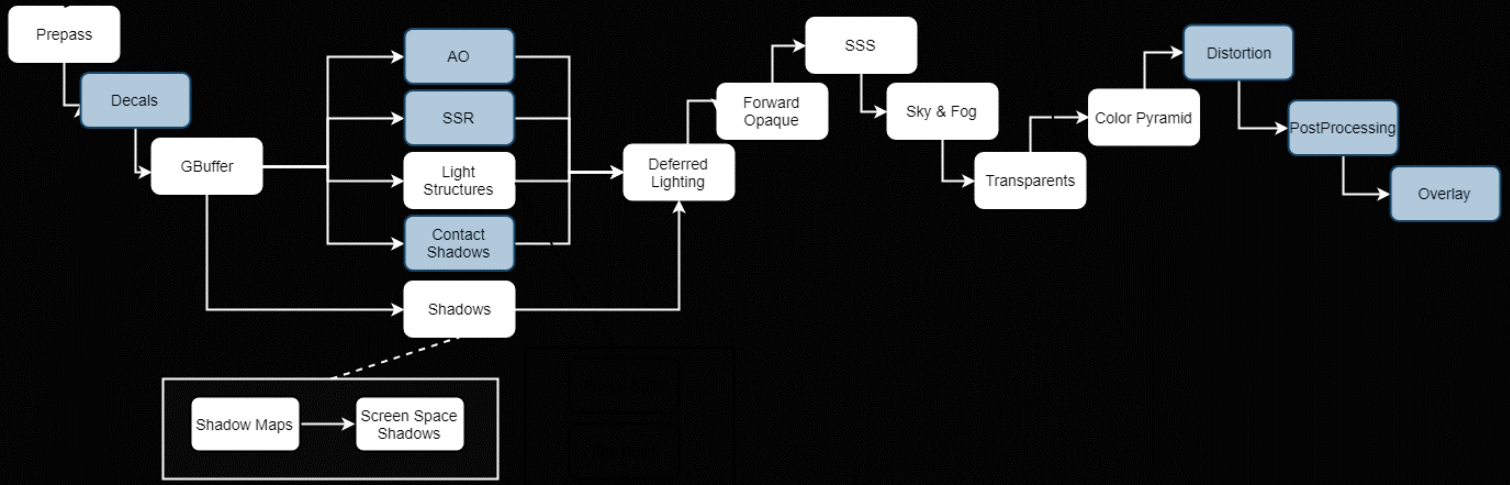
HDRP's hybrid deferred/forward render graph

Render Graph - Deferred/Forward

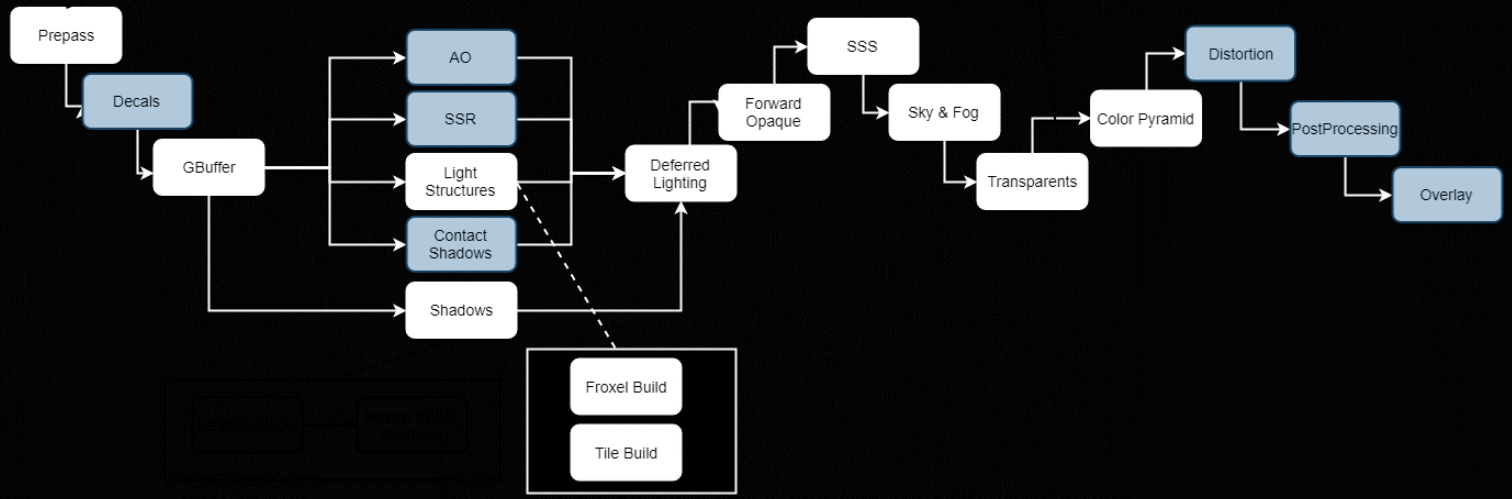


HDRP's hybrid deferred/forward render graph
(blue are optional steps)

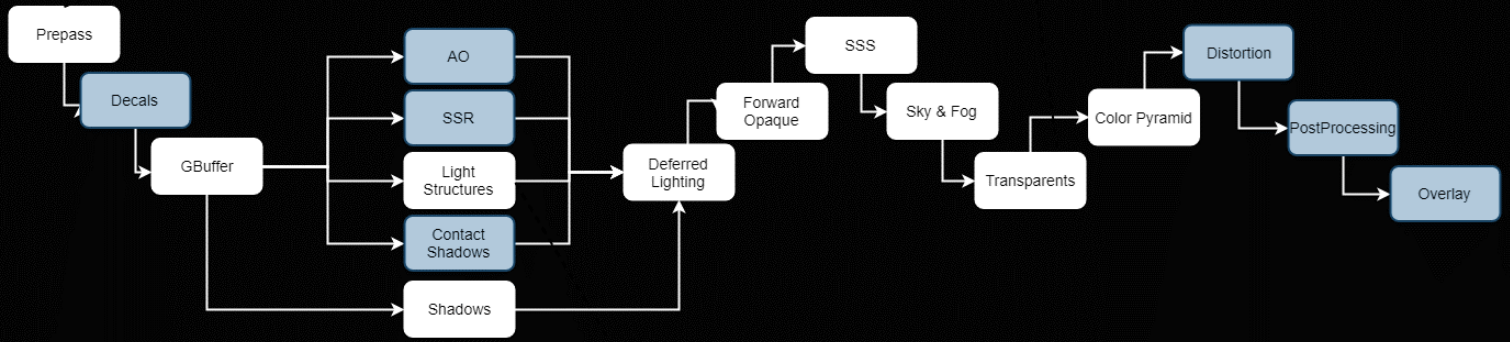
Render Graph - Deferred/Forward



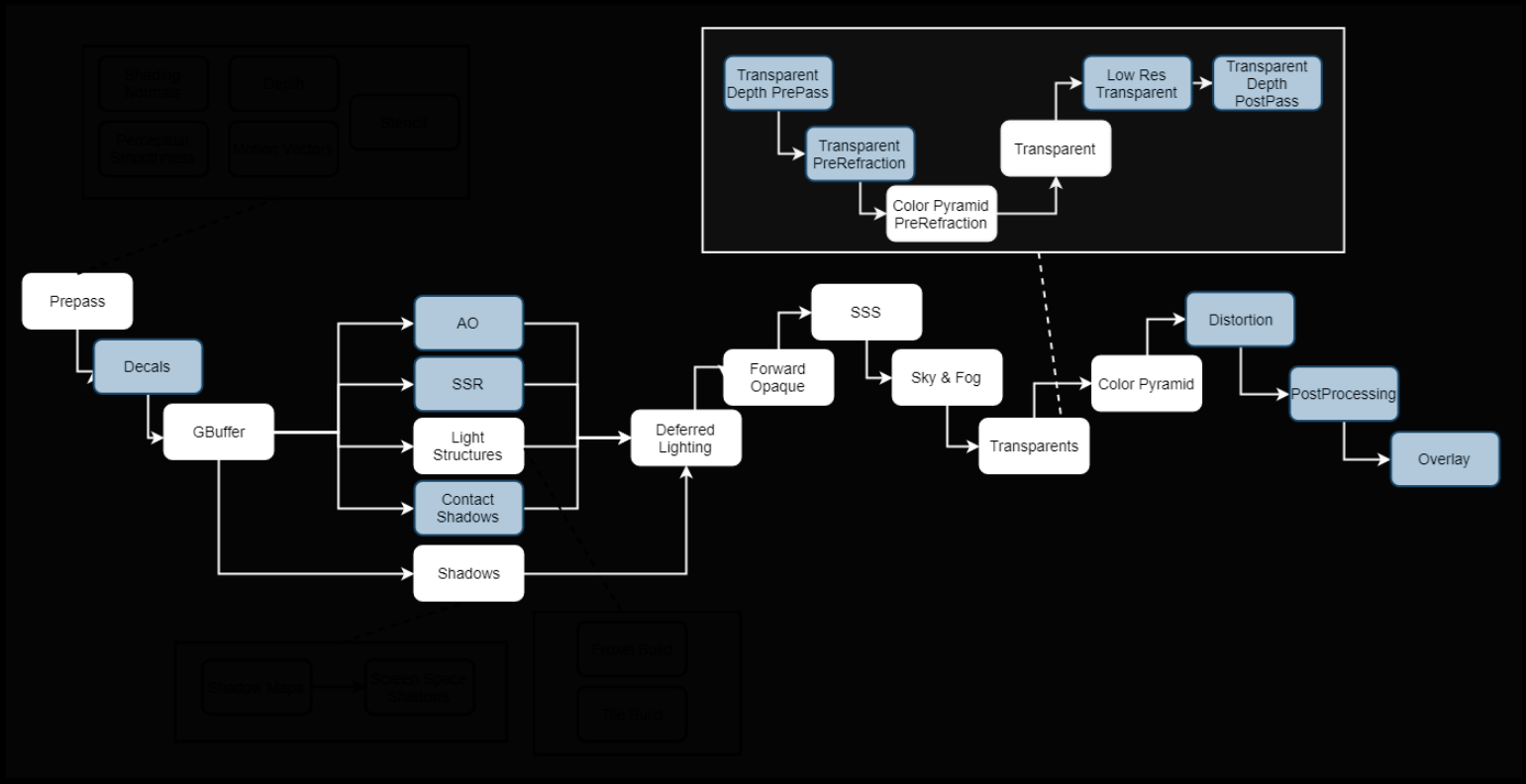
Render Graph - Deferred/Forward



Render Graph - Deferred/Forward

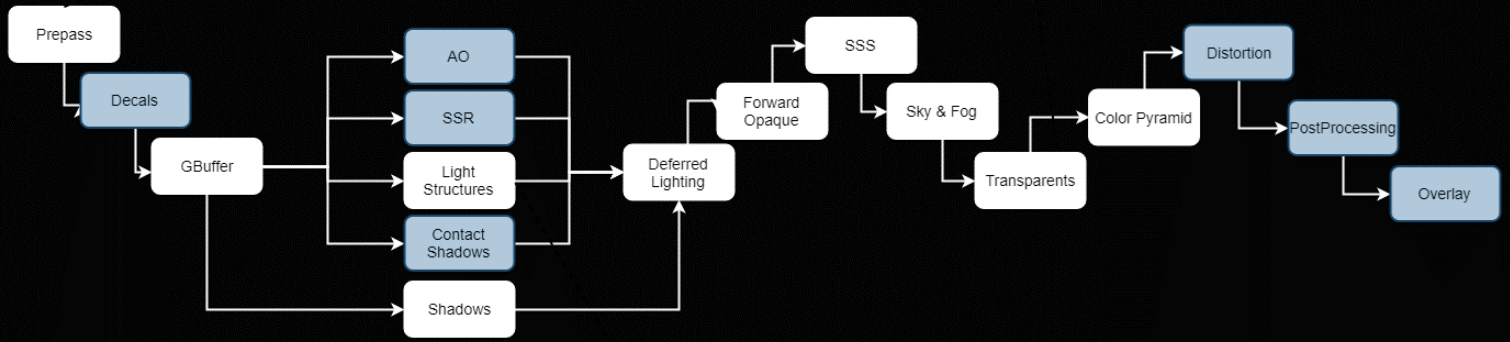


Render Graph - Deferred/Forward

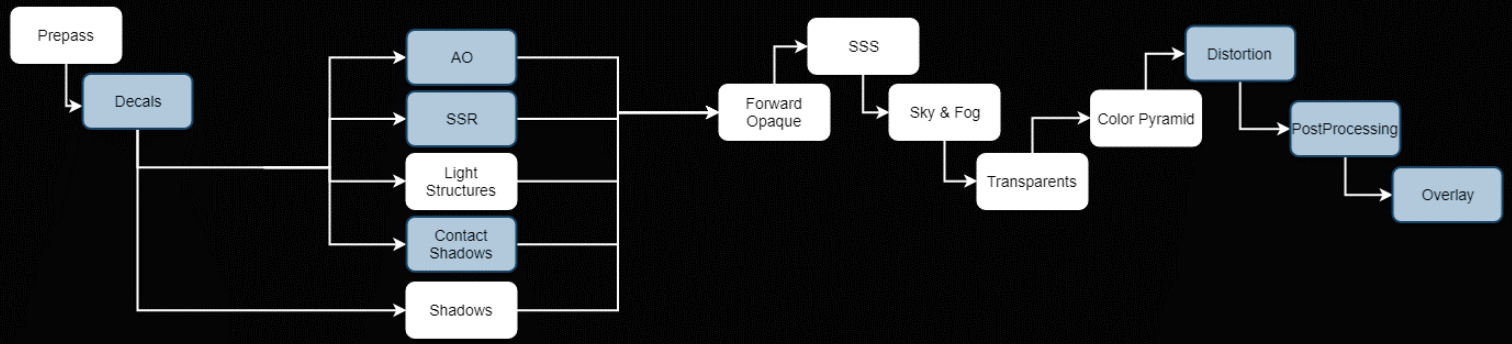


unity

Render Graph - Deferred/Forward

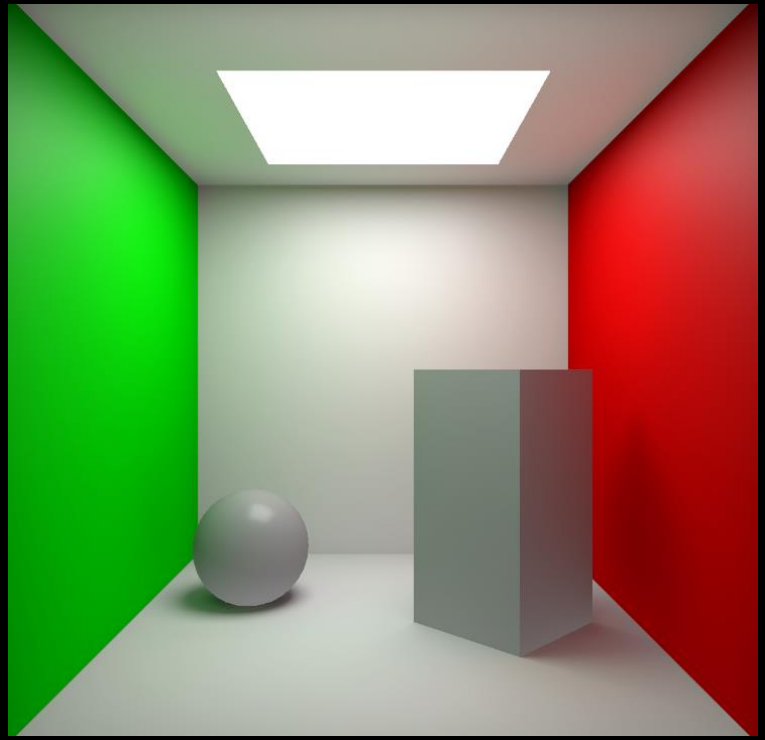


Render Graph - Forward Only



HDRP's forward only render graph

Cornell Box



Cornell Box in HDRP with stochastic area shadows and ray traced indirect diffuse

Cornell Box

Cornell Box

```
gid: sphere,
color: float3(1, 1, 1),
TRS: ...
```

```
gid: plane,
color: float3(0, 1, 0),
TRS: ...
```

```
gid: plane,
color: float3(1, 1, 1),
TRS: ...
```

```
gid: plane,
color: float3(1, 1, 1),
TRS: ...
```

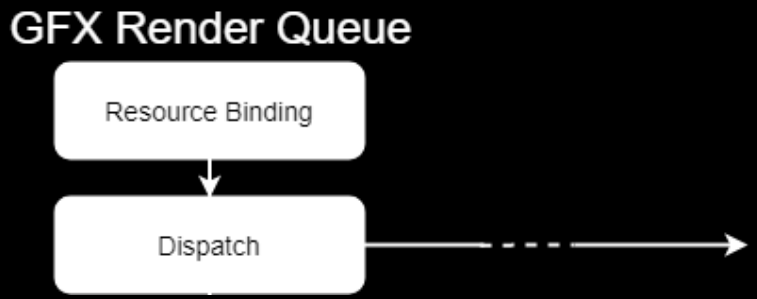
```
gid: cube,
color: float3(1, 1, 1),
TRS: ...
```

```
gid: plane,
color: float3(1, 1, 1),
TRS: ...
```

```
gid: plane,
color: float3(1, 0, 0),
TRS: ...
```

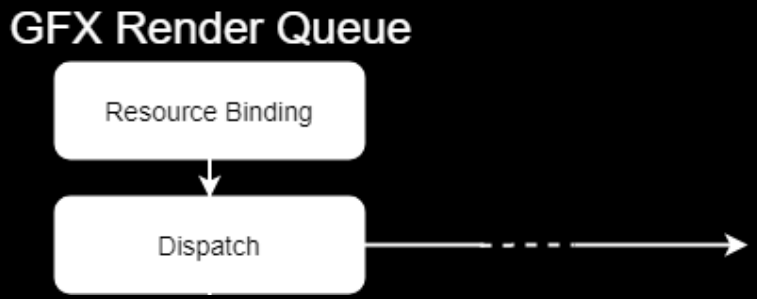
```
gid: plane,
color: float3(1, 1, 1),
TRS: ...
```

Classic Command List




```
[numthreads(COMPUTE_TILE_RESOLUTION, COMPUTE_TILE_RESOLUTION, 1)]
void ComputeShaderExample(uint2 : SV_GroupThreadID, uint2 : SV_GroupID)
```

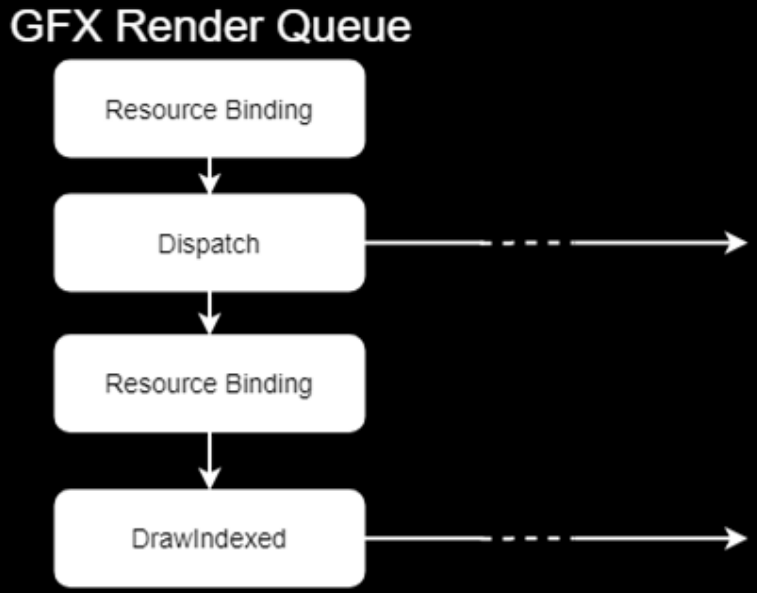
Classic Command List




```
[numthreads(COMPUTE_TILE_RESOLUTION, COMPUTE_TILE_RESOLUTION, 1)]
void ComputeShaderExample(uint2 : SV_GroupThreadID, uint2 : SV_GroupID)
```

SSR, SSAO, Conact Shadows...

Classic Command List

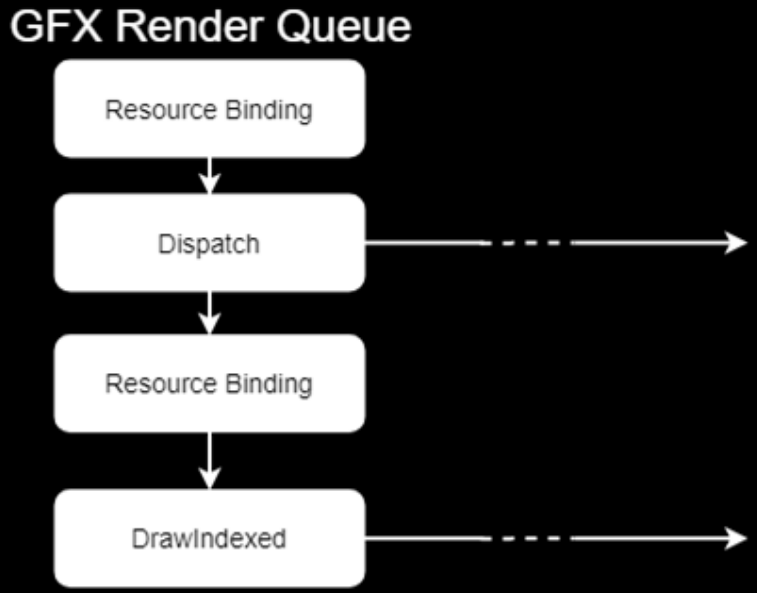



```
[numthreads(COMPUTE_TILE_RESOLUTION, COMPUTE_TILE_RESOLUTION, 1)]
void ComputeShaderExample(uint2 : SV_GroupThreadID, uint2 : SV_GroupID)
```

```
PackedVaryingsType Vert(AttributesMesh)
void Frag(PackedVaryingsToPS, out float4 : SV_Target0, out float: SV_Depth);
```

```
gid: sphere,
color: float3(1, 1, 1),
TRS: ...
```

Classic Command List




```
[numthreads(COMPUTE_TILE_RESOLUTION, COMPUTE_TILE_RESOLUTION, 1)]
void ComputeShaderExample(uint2 : SV_GroupThreadID, uint2 : SV_GroupID)
```

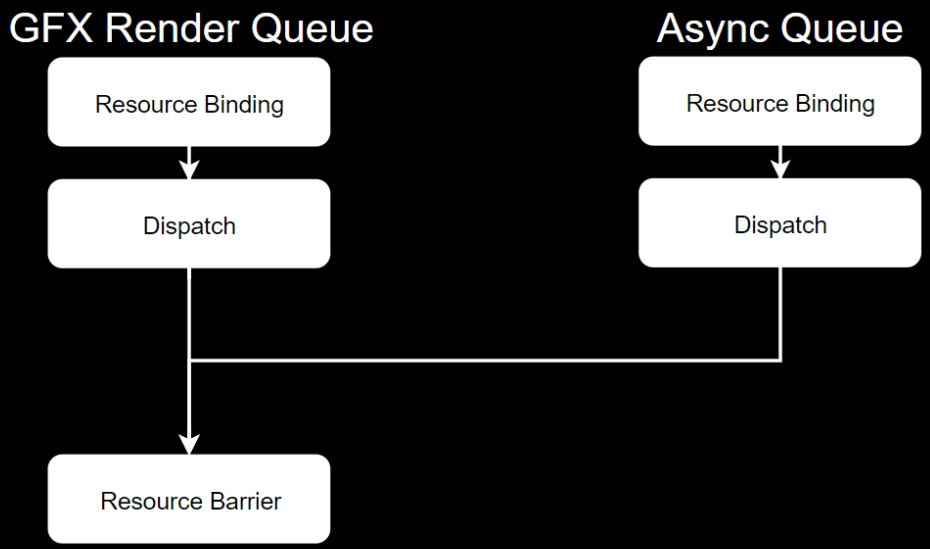
Prepass, GBuffer, Forward

Opaque, Forward Transparents, ...

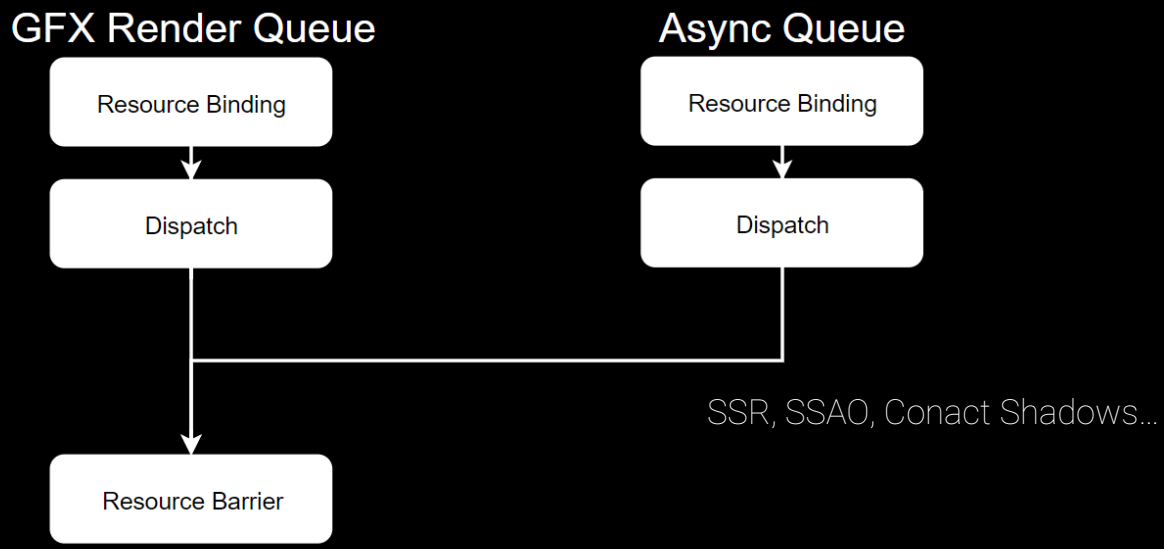
```
PackedVaryingsType Vert(AttributesMesh)
void Frag(PackedVaryingsToPS, out float4 : SV_Target0, out float: SV_Depth);
```

```
gid: sphere,
color: float3(1, 1, 1),
TRS: ...
```

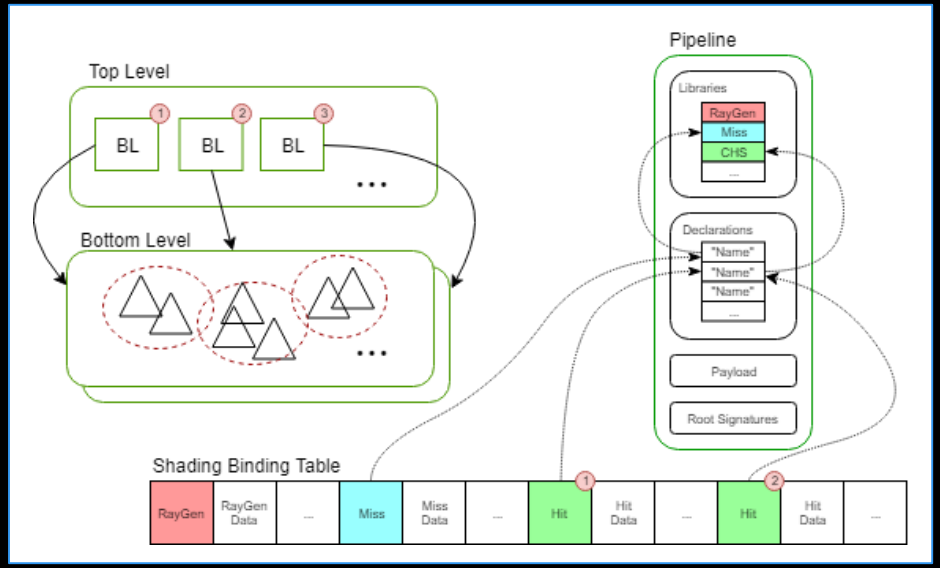
Classic Command List



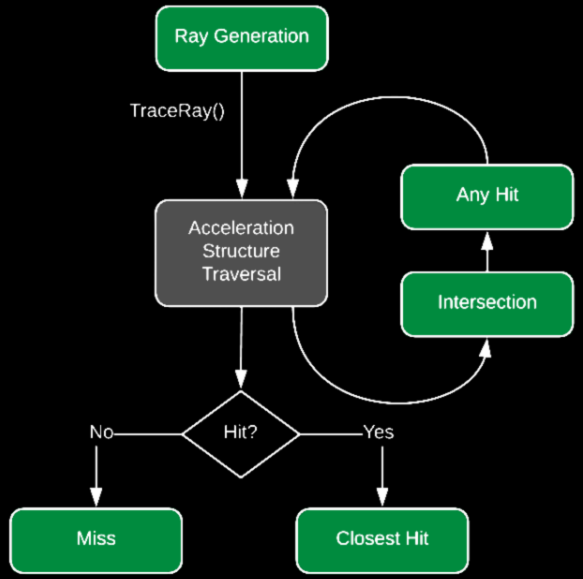
Classic Command List



DXR API

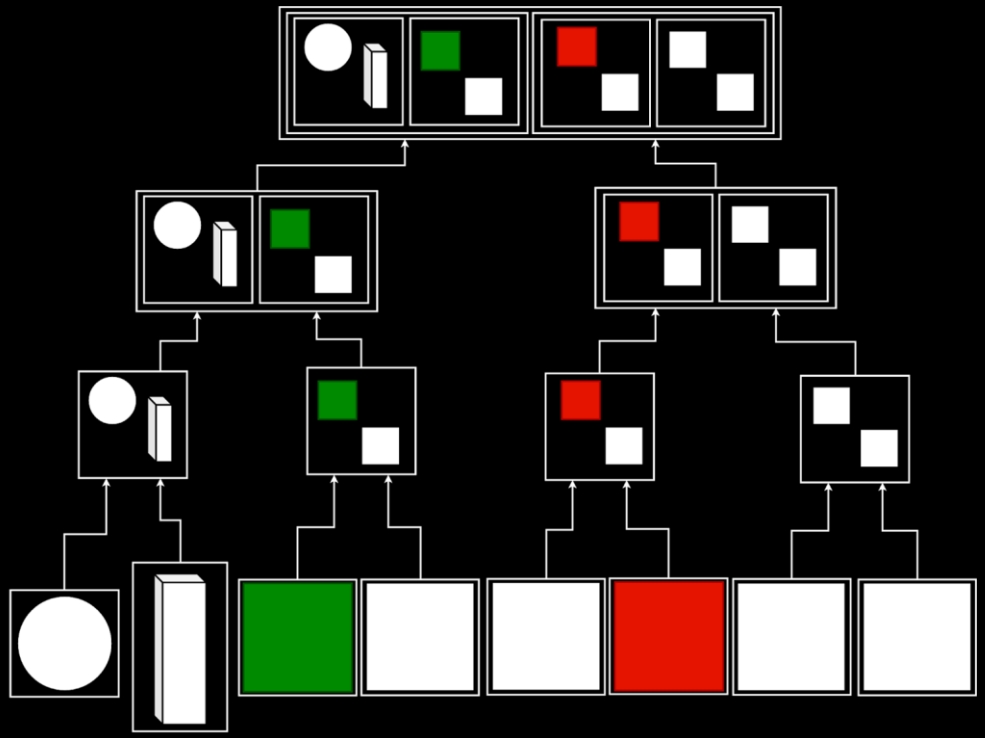


Ray Tracing Acceleration Structure

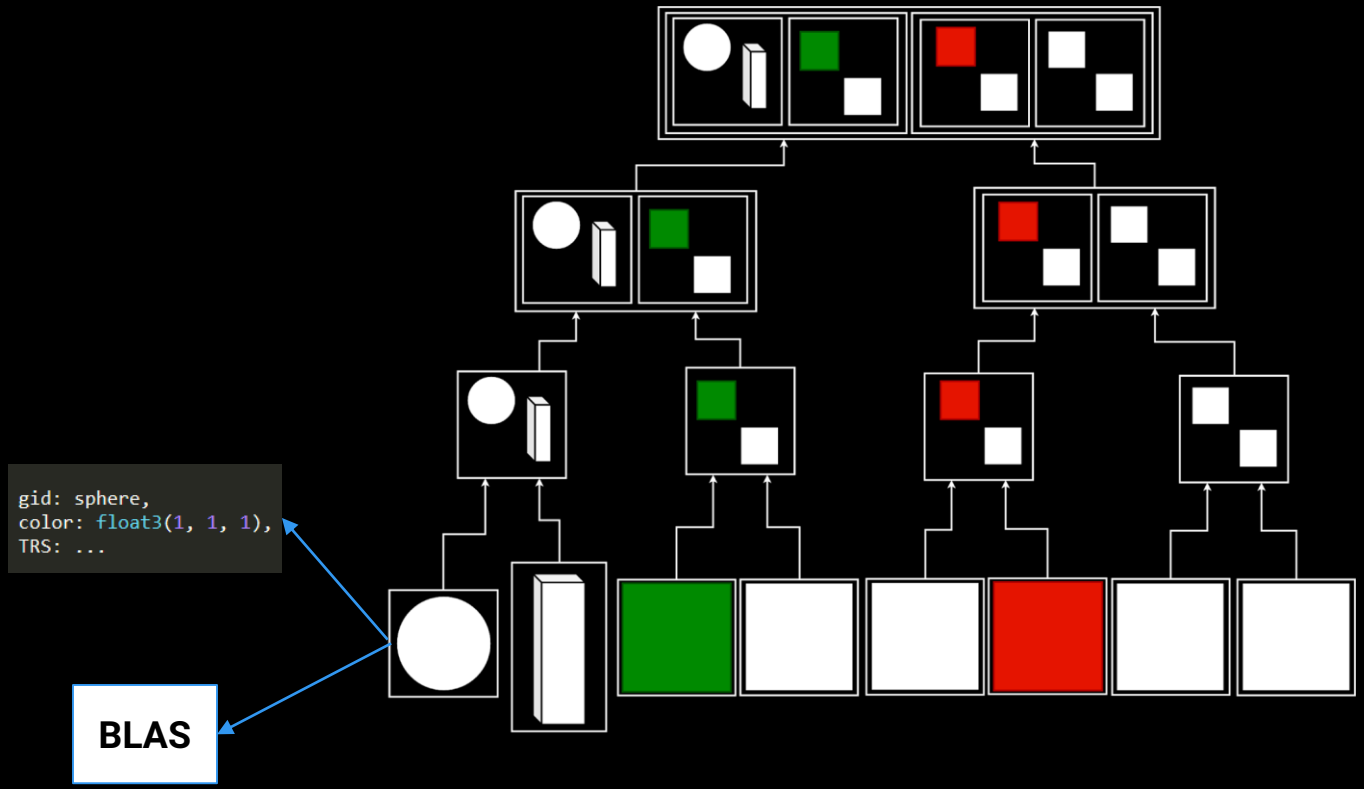


Ray Tracing Shader Pipeline

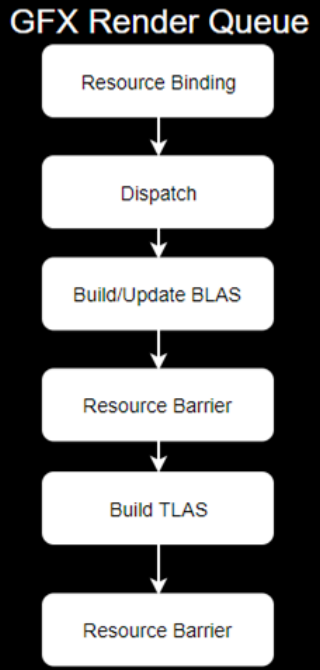
Ray Tracing Acceleration Structure



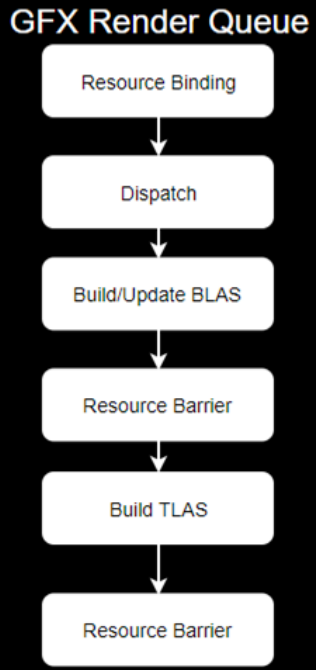
Ray Tracing Acceleration Structure

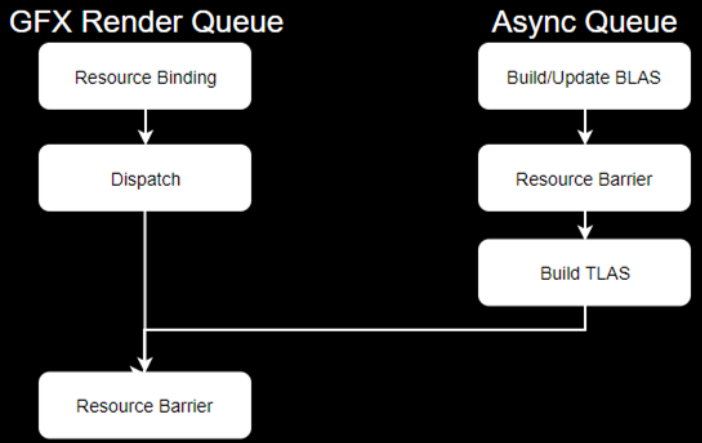


Hybrid Renderer Command List

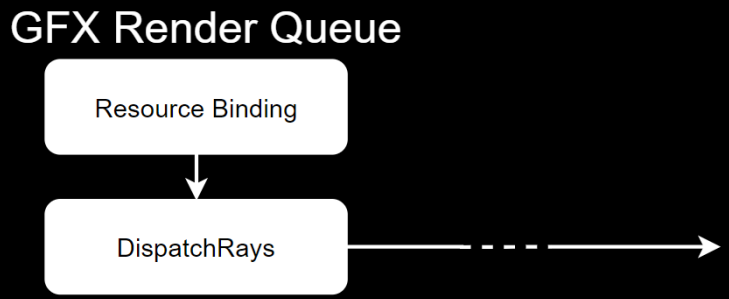


Hybrid Renderer Command List





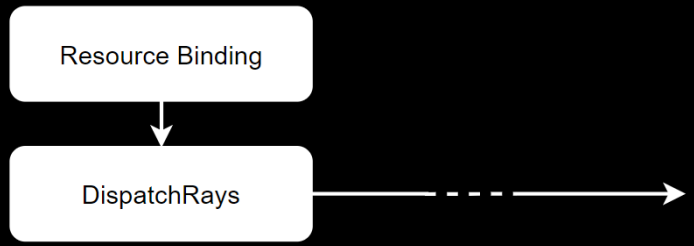
Hybrid Renderer Command List




```
[shader("miss")]
void ExampleMiss(inout : SV_RayPayload)

[shader("raygeneration")]
void ExampleRayGen()

[shader("anyhit")]
void ExampleDefaultAnyHit(inout : SV_RayPayload, : SV_IntersectionAttributes)

[shader("closesthit")]
void ExampleDefaultClosestHit(inout : SV_RayPayload, : SV_IntersectionAttributes)
```

Hybrid Renderer Command List

GFX Render Queue




```
[shader("miss")]
void ExampleMiss(inout : SV_RayPayload)

[shader("raygeneration")]
void ExampleRayGen()

[shader("anyhit")]
void ExampleDefaultAnyHit(inout : SV_RayPayload, : SV_IntersectionAttributes)

[shader("closesthit")]
void ExampleDefaultClosestHit(inout : SV_RayPayload, : SV_IntersectionAttributes)
```

```
[shader("anyhit")]
void ExampleAnyHit(inout : SV_RayPayload, : SV_IntersectionAttributes)

[shader("closesthit")]
void ExampleClosestHit(inout : SV_RayPayload, : SV_IntersectionAttributes)
```

```
gid: sphere,
color: float3(1, 1, 1),
TRS: ...
```

```
[shader("anyhit")]
void ExampleAnyHit(inout : SV_RayPayload, : SV_IntersectionAttributes)

[shader("closesthit")]
void ExampleClosestHit(inout : SV_RayPayload, : SV_IntersectionAttributes)
```

```
gid: plane,
color: float3(1, 0, 0),
TRS: ...
```

Missing Features

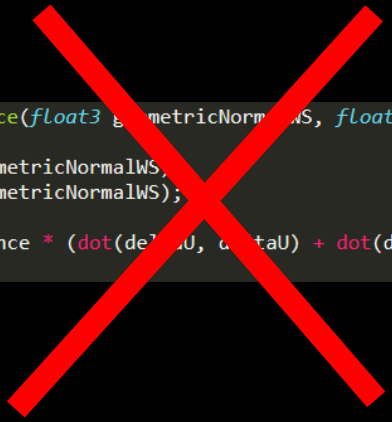
```
float GeometricNormalVariance(float3 geometricNormalWS, float screenSpaceVariance)
{
    float3 deltaU = ddx(geometricNormalWS);
    float3 deltaV = ddy(geometricNormalWS);

    return screenSpaceVariance * (dot(deltaU, deltaU) + dot(deltaV, deltaV));
}
```

Missing Features

```
float GeometricNormalVariance(float3 geometricNormalWS, float screenSpaceVariance)
{
    float3 deltaU = ddx(geometricNormalWS);
    float3 deltaV = ddy(geometricNormalWS);

    return screenSpaceVariance * (dot(deltaU, deltaU) + dot(deltaV, deltaV));
}
```

Missing Features

```
float GeometricNormalVariance(float3 g metricNorm WS, float screenSpaceVariance)
{
    float3 deltaU = ddx(geometricNormalWS);
    float3 deltaV = ddy(geometricNormalWS);

    return screenSpaceVariance * (dot(del U, d taU) + dot(deltaV, deltaV));
}
```



Missing Features

```
float GeometricNormalVariance(float3 g metricNorm WS, float screenSpaceVariance)
{
    float3 deltaU = ddx(geometricNormalWS);
    float3 deltaV = ddy(geometricNormalWS);

    return screenSpaceVariance * (dot(del U, d taU) + dot(deltaV, deltaV));
}
```

```
#if USE_WAVE_INTRINSICS

#define LDS_SIZE  (TILE_SIZE * TILE_SIZE) / WAVE_SIZE

#else

#define LDS_SIZE TILE_SIZE * TILE_SIZE

#endif

groupshared float gs_minMotionVec[LDS_SIZE];
groupshared uint gs_maxMotionVec[LDS_SIZE];
```

Missing Features

```
float GeometricNormalVariance(float3 g metricNorm WS, float screenSpaceVariance)
{
    float3 deltaU = ddx(geometricNormalWS,
    float3 deltaV = ddy(geometricNormalWS);

    return screenSpaceVariance * (dot(del U, d taU) + dot(deltaV, deltaV));
}
```

```
#if USE_WAVE_INT SICS

#define LDS_SIZE  (T SIZE * T _SIZE) / WAVE_SIZE

#else

#define LDS_SIZE TILE_SI LE_SIZE

#endif

groupshared floa gs_minMotionVec[L SIZE];
groupshared ui gs_maxMotionVec[LDS_ E];
```

Missing Features

```
float GeometricNormalVariance(float3 g metricNorm WS, float screenSpaceVariance)
{
    float3 deltaU = ddx(geometricNormalWS,
    float3 deltaV = ddy(geometricNormalWS);

    return screenSpaceVariance * (dot(del U, d taU) + dot(deltaV, deltaV));
}
```

```
#if USE_WAVE_INT SICS

#define LDS_SIZE  (T SIZE * T _SIZE) / WAVE_SIZE

#else

#define LDS_SIZE TILE_SI ILE_SIZE

#endif

groupshared floa gs_minMotionVec[L SIZE];
groupshared ui gs_maxMotionVec[LDS_ E];
```

```
if (test.w == 0.0)
    return _MainTex.Sample(sampler_MainTex, i.texcoordStereo);
```

Missing Features

```
float GeometricNormalVariance(float3 g metricNorm WS, float screenSpaceVariance)
{
    float3 deltaU = ddx(geometricNormalWS);
    float3 deltaV = ddy(geometricNormalWS);

    return screenSpaceVariance * (dot(del U, d taU) + dot(deltaV, deltaV));
}
```

```
#if USE_WAVE_INT SICS

#define LDS_SIZE  (T SIZE * T _SIZE) / WAVE_SIZE

#else

#define LDS_SIZE TILE_SI ILE_SIZE

#endif

groupshared floa gs_minMotionVec[L SIZE];
groupshared ui gs_maxMotionVec[LDS_ E];
```

```
if (test.w == 0.0)
    return _MainTex.Sampl (samp r_MainTex, i.texcoordStereo);
```

Surface vs Effect Type Shaders

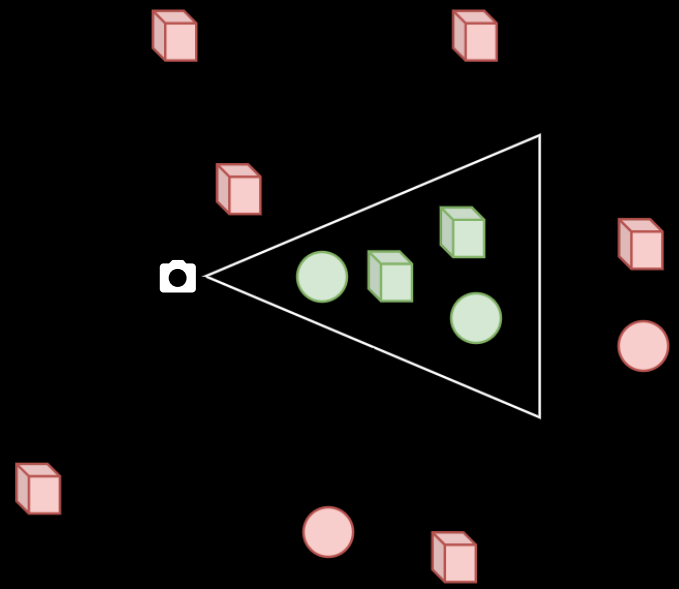
```
SubShader
{
    Pass
    {
        Name "GBuffer"
        PackedVaryingsType Vert(AttributesMesh inputMesh);
        void Frag(PackedVaryingsToPS packedInput, OUTPUT_GBUFFER(outGBuffer));
    }

    Pass
    {
        Name "ShadowCaster"
        PackedVaryingsType Vert(AttributesMesh inputMesh);
        void Frag(PackedVaryingsToPS packedInput, out float outputDepth : SV_Depth);
    }

    Pass
    {
        Name "DepthOnly"
        PackedVaryingsType Vert(AttributesMesh inputMesh);
        void Frag(PackedVaryingsToPS packedInput, out float outputDepth : SV_Depth);
    }

    Pass
    {
        Name "MotionVectors"
        PackedVaryingsType Vert(AttributesMesh inputMesh);
        void Frag(PackedVaryingsToPS packedInput, out float4 outMotionVector : SV_Target0);
    }
}
```

Surface vs Effect Type Shaders

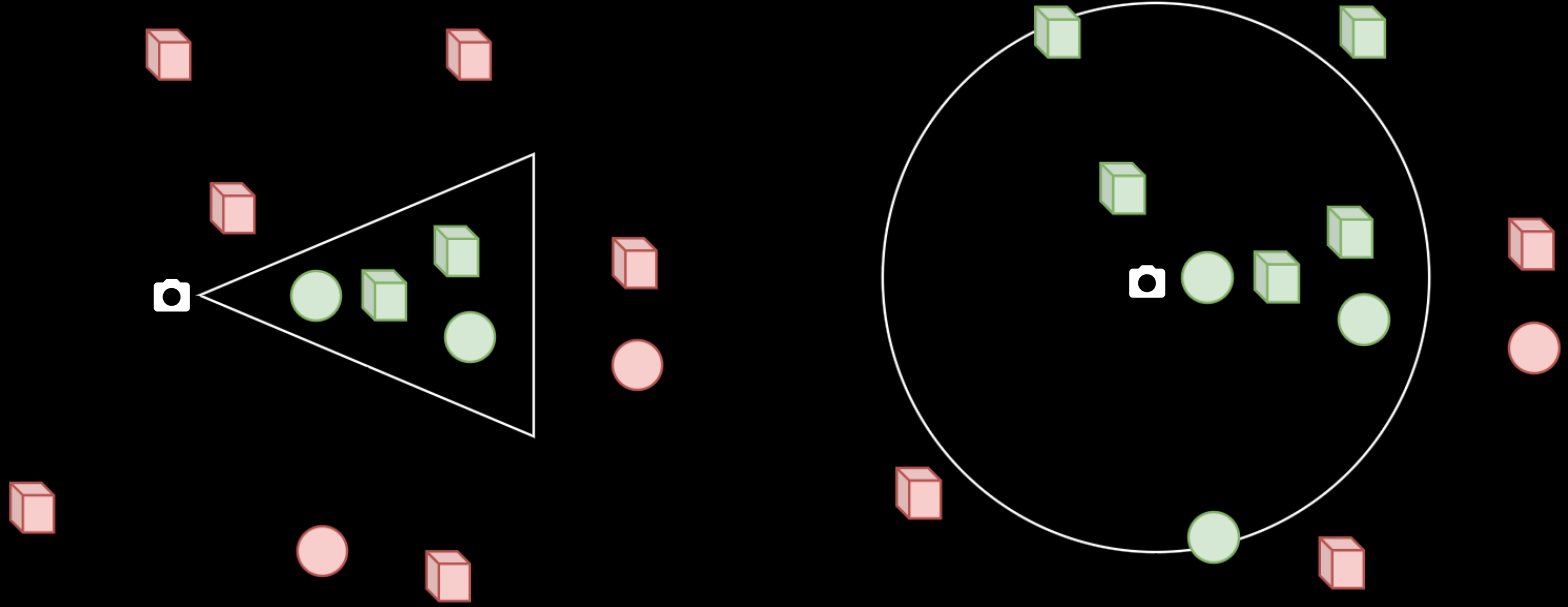
```
SubShader
{
    Pass
    {
        Name "GBuffer"
        PackedVaryingsType Vert(AttributesMesh inputMesh);
        void Frag(PackedVaryingsToPS packedInput, OUTPUT_GBUFFER(outGBuffer));
    }

    Pass
    {
        Name "ShadowCaster"
        PackedVaryingsType Vert(AttributesMesh inputMesh);
        void Frag(PackedVaryingsToPS packedInput, out float outputDepth : SV_Depth);
    }

    Pass
    {
        Name "DepthOnly"
        PackedVaryingsType Vert(AttributesMesh inputMesh);
        void Frag(PackedVaryingsToPS packedInput, out float outputDepth : SV_Depth);
    }

    Pass
    {
        Name "MotionVectors"
        PackedVaryingsType Vert(AttributesMesh inputMesh);
        void Frag(PackedVaryingsToPS packedInput, out float4 outMotionVector : SV_Target0);
    }
}
```

```
SubShader
{
    Pass
    {
        Name "IndirectDXR"

        [shader("closesthit")]
        void ClosestHitIndirect(inout : SV_RayPayload, : SV_IntersectionAttributes);
        [shader("anyhit")]
        void AnyHitIndirect(inout : SV_RayPayload, : SV_IntersectionAttributes);
    }

    Pass
    {
        Name "ForwardDXR"
        [shader("closesthit")]
        void ClosestHitForward(inout : SV_RayPayload, : SV_IntersectionAttributes);
        [shader("anyhit")]
        void AnyHitForward(inout : SV_RayPayload, : SV_IntersectionAttributes);
    }

    Pass
    {
        Name "VisibilityDXR"
        [shader("anyhit")]
        void AnyHitVisibility(inout : SV_RayPayload, : SV_IntersectionAttributes);
    }
}
```

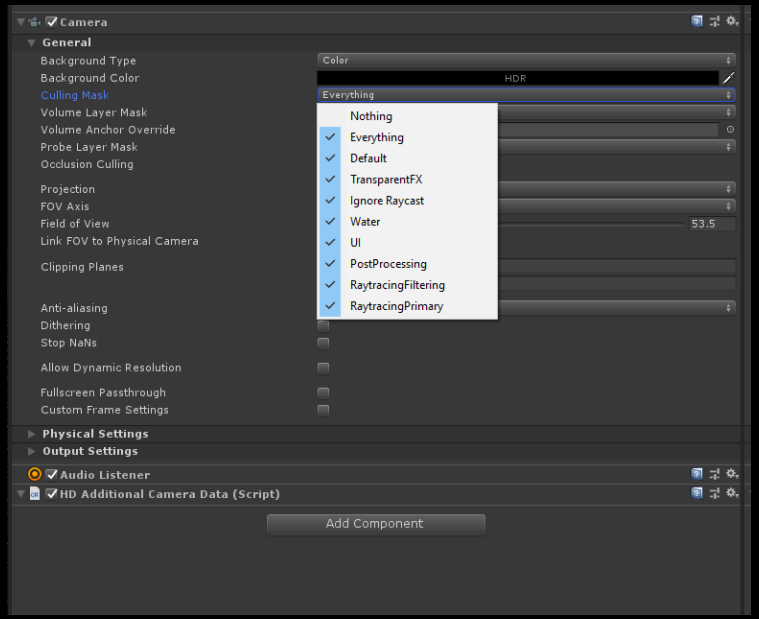
Frustum Culling vs Volume Culling



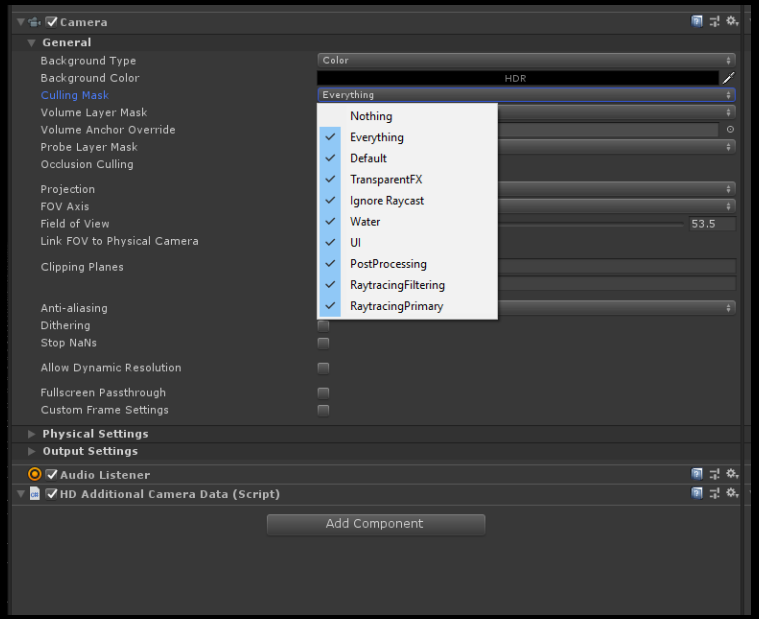
Frustum Culling vs Volume Culling

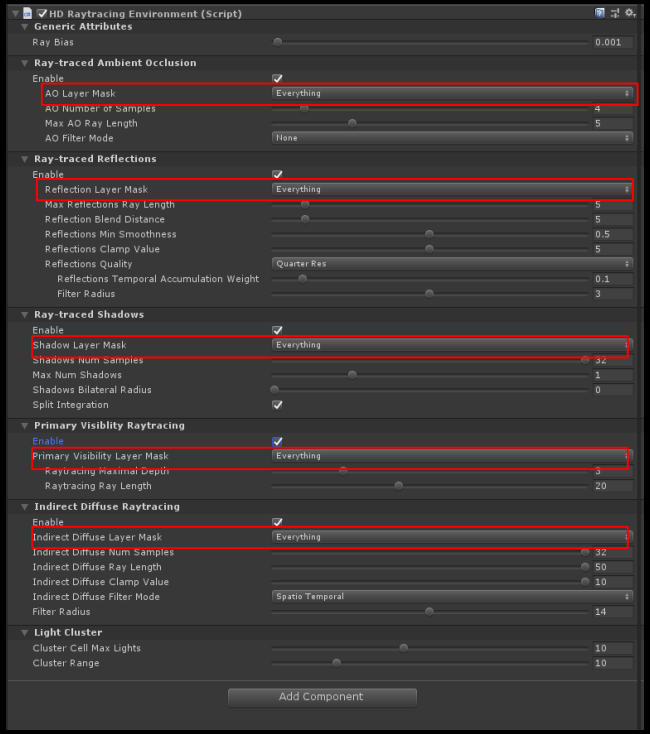


Camera Filtering vs Per Effect Filtering

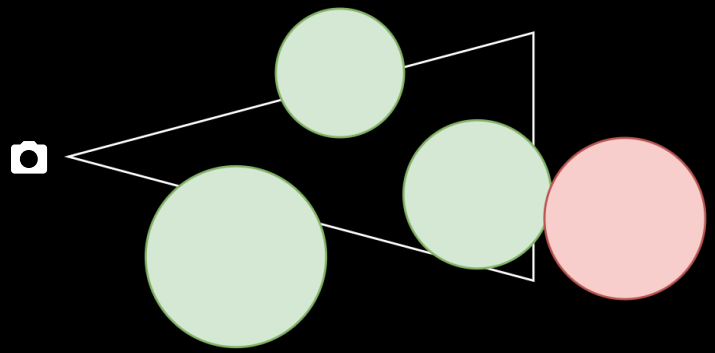


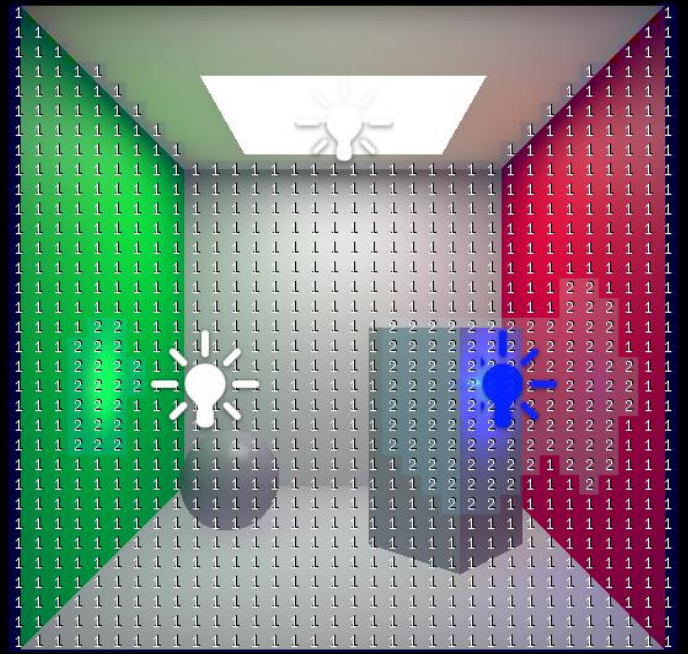
Camera Filtering vs Per Effect Filtering



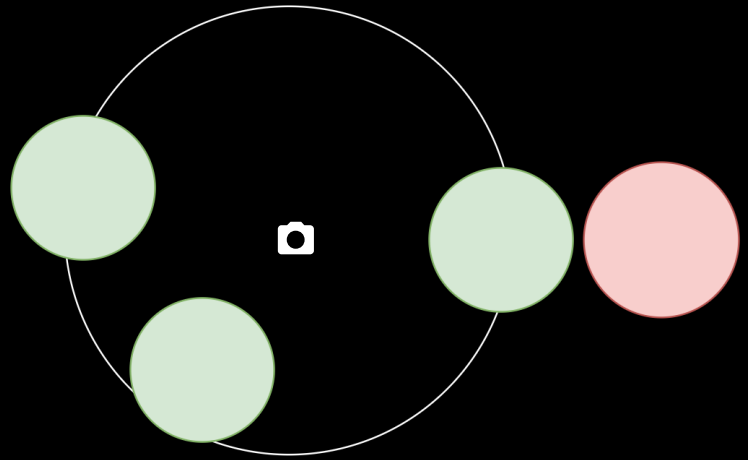


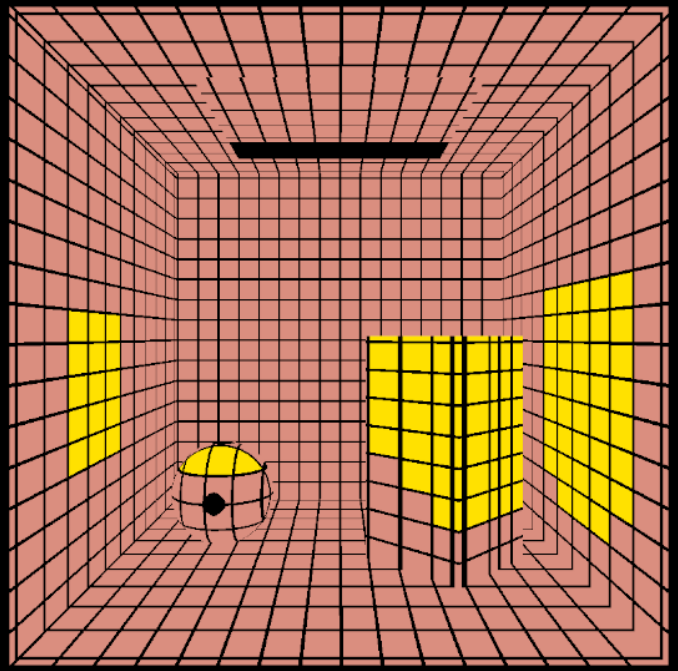
Tiled Lighting



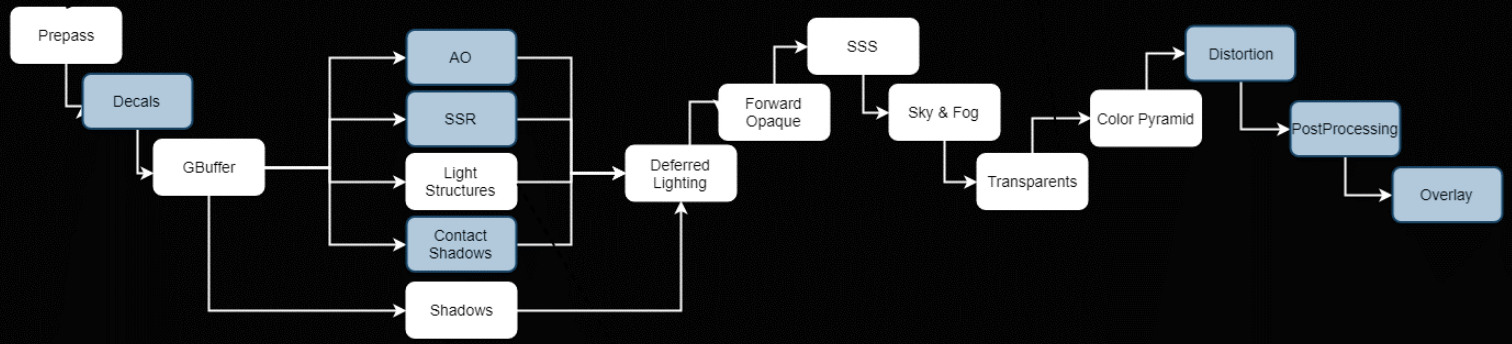


Cluster Lighting



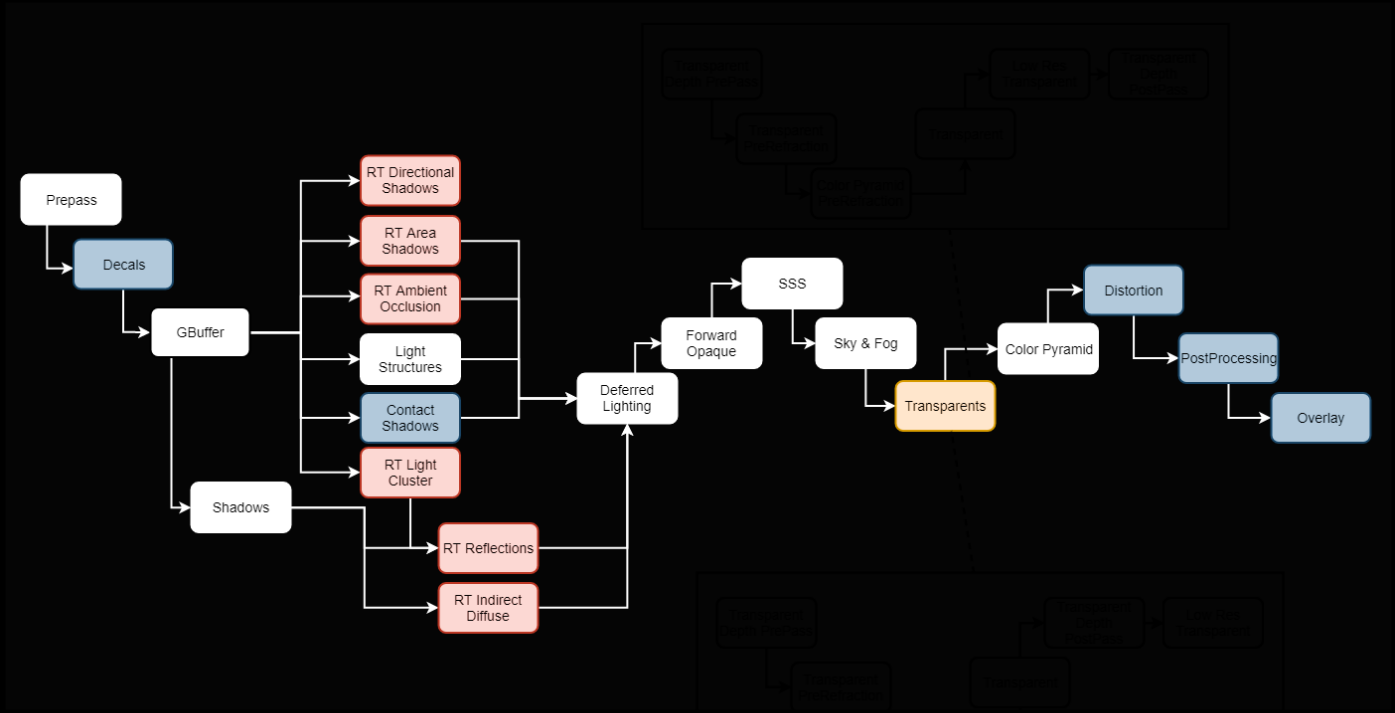


Render Graph - Deferred/Forward



HDRP's hybrid deferred/forward render graph

Hybrid Render Graph



HDRP's hybrid rasterization/ray tracing render graph

Hybrid Pipeline

$$L_o(\mathbf{x}, \omega_o, \lambda, t) = L_e(\mathbf{x}, \omega_o, \lambda, t) + \int_\Omega f_r(\mathbf{x}, \omega_i, \omega_o, \lambda, t)\, L_i(\mathbf{x}, \omega_i, \lambda, t)\, (\omega_i \cdot \mathbf{n})\, d\omega_i$$

- Blue solved using an integrator (path tracer, photon mapper, etc.)
- Needs to be rephrased for our hybrid approach

Hybrid Pipeline

$$L_o(\mathbf{x}, \omega_o, \lambda, t) = L_e(\mathbf{x}, \omega_o, \lambda, t) + L_{direct}(\mathbf{x}, \omega_o, \lambda, t) + L_{indirect}(\mathbf{x}, \omega_o, \lambda, t)$$

- Red evaluated using rasterization (almost)
- Blue evaluated using rasterization or raytracing

Hybrid Pipeline

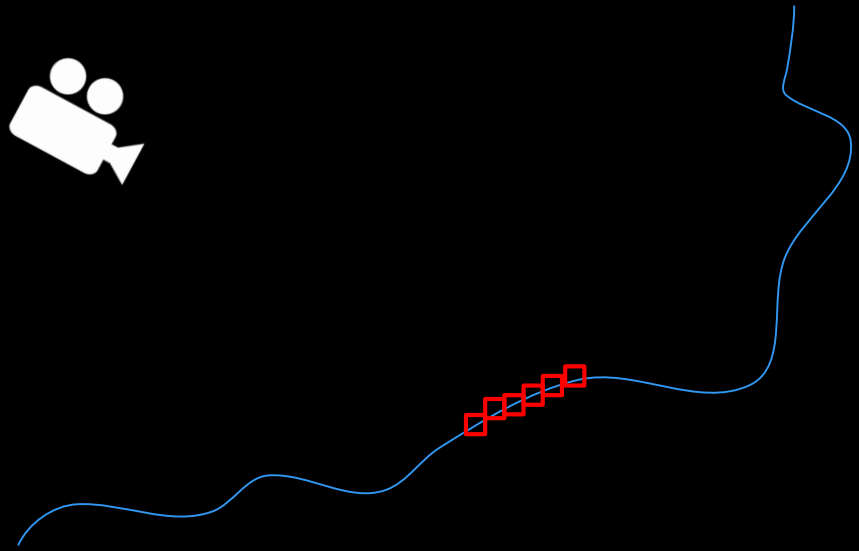
$$L_{indirect}(\mathbf{x}, \omega_o, \lambda, t) = L_{specular}(\mathbf{x}, \omega_o, \lambda, t) + L_{diffuse}(\mathbf{x}, \omega_o, \lambda, t)$$

$$L_{\text{indirect specular}} \approx \int_\Omega \frac{F\,G\,D}{4(\omega_i \cdot \mathbf{n})(\mathbf{v} \cdot \mathbf{n})} L_{\text{indirect}}\,(\omega_i \cdot \mathbf{n})\,d\omega_i$$

$$L_{\text{indirect diffuse}} \approx \int_\Omega \frac{\rho}{\pi} L_{\text{indirect}}\,(\omega_i \cdot \mathbf{n})\,d\omega_i$$

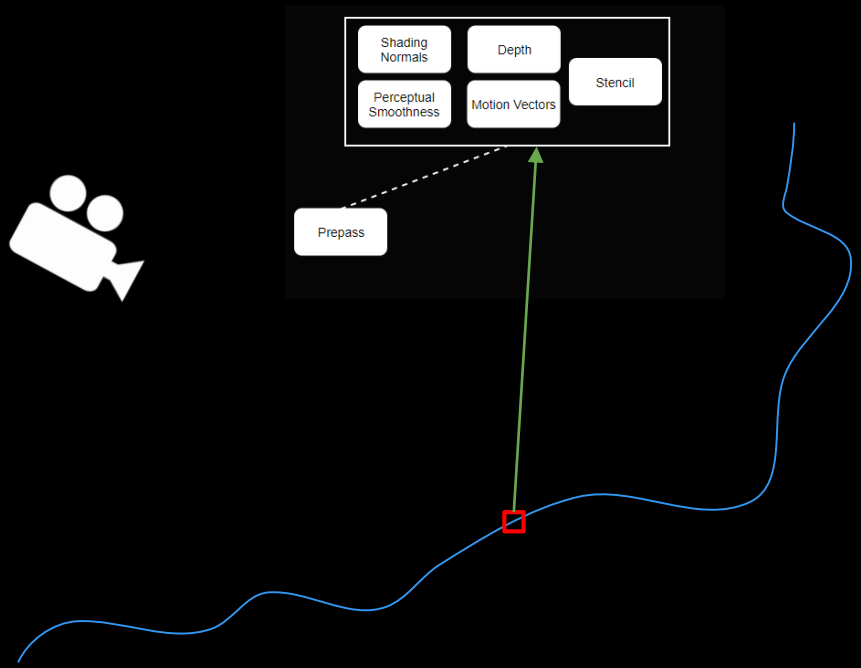
- Indirect specular is approximated as an Isotropic GGX lobe
- Indirect diffuse is approximates as a Lambert lobe
- Ambient occlusion affect indirect diffuse or both

Ray Generation



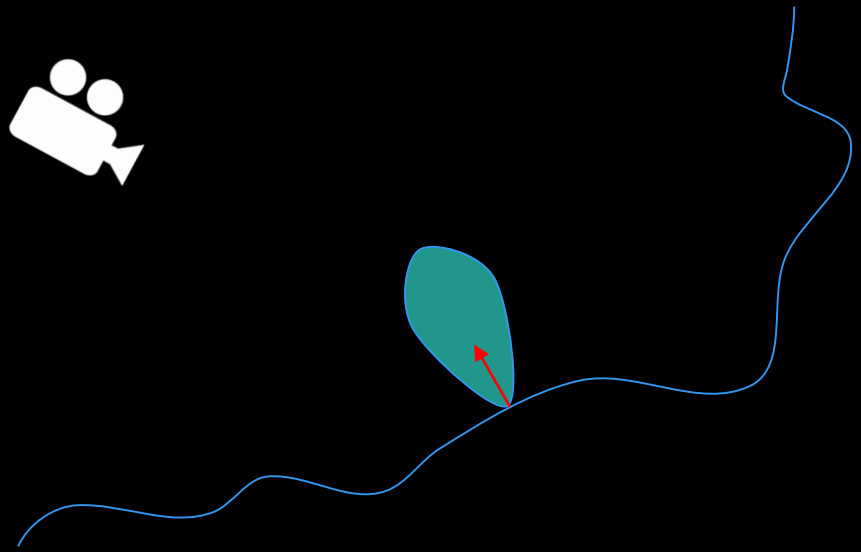
Depth Buffer

Ray Generation



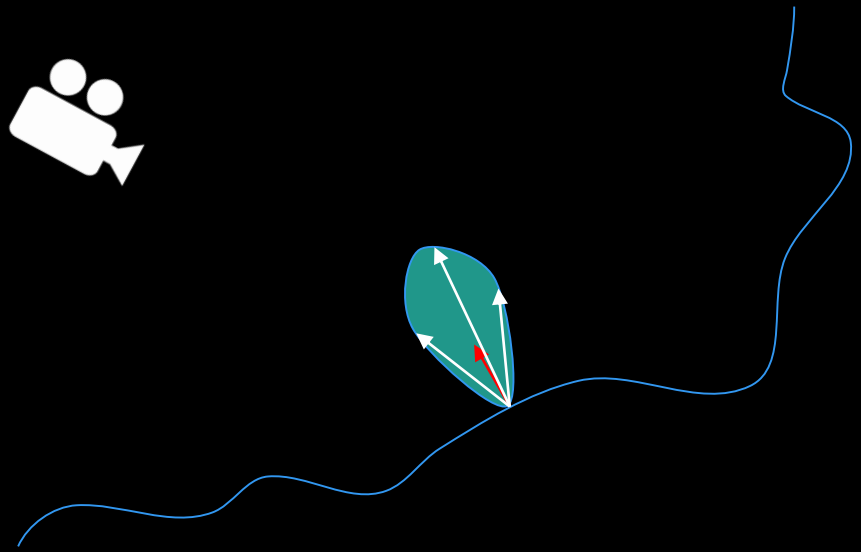
Depth Buffer

Ray Generation



Depth Buffer

Ray Generation



Depth Buffer

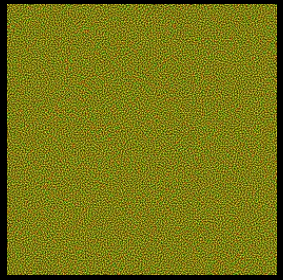
Sampling Based Integration

- Our budget is 1spp per effect per frame (or less)
- We use spatio-temporal reprojection/accumulation to increase the coverage
  - Spatial coverage improved by *“Blue-noise Dithered Sampling” [Georgiev and Fajardo 2016]*
  - Temporal coverage improved by iterating over a Sobol sequence

Sampling Based Integration

```
uint2 ScramblingValue(uint i, uint j)
{
    i = i % 256;
    j = j % 256;
    return clamp((uint2)(_ScramblingTexture[uint2(i, j)] * 256.0f), uint2(0,0), uint2(255, 255));
}

float GetRaytracingNoiseSample(uint sampleIndex, uint sampleDimension, uint scramblingValue)
{
    // Make sure arguments are in the right range
    sampleIndex = sampleIndex % 256;
    sampleDimension = sampleDimension % 4;

    // Fetch the matching Value sequence
    uint value = clamp((uint)(_OwenScrambledTexture[uint2(sampleIndex, 0)][sampleDimension] * 256.0f), 0, 255);

    // Scramble the value
    value = value ^ scramblingValue;

    // convert to float and return
    float v = (0.5f + value) / 256.0f;
    return v;
}
```

Owen Scrambled Sobol Sequence 256x1x4



Scrambling Tile 256x256x2

Indirect Specular

Indirect Specular

$$L_{\text{indirect specular}} \approx \int_{\Omega} \frac{\mathrm{F}\,\mathrm{G}\,D}{4(\omega_i \cdot \mathbf{n})(\mathbf{v} \cdot \mathbf{n})} L_{\text{indirect}}\,(\omega_i \cdot \mathbf{n})\, d\omega_i$$

Indirect Specular

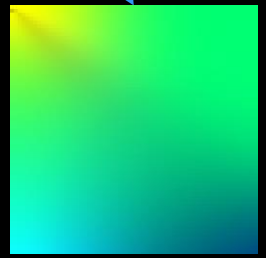
$$L_{\text{indirect specular}} \approx \int_{\Omega} \frac{\mathrm{F}\,\mathrm{G}\,D}{4(\omega_i \cdot \mathbf{n})(\mathbf{v} \cdot \mathbf{n})} L_{\text{indirect}}(\omega_i \cdot \mathbf{n})\, d\omega_i$$

$$L_{\text{indirect specular}} \approx specularFGD \int_{\Omega} \frac{(\mathbf{h} \cdot \mathbf{n})\, D}{4(\mathbf{h} \cdot \mathbf{n})} L_{\text{indirect}}(\omega_i \cdot \mathbf{n})\, d\omega_i \; *$$

Indirect Specular

$$L_{\text{indirect specular}} \approx \int_{\Omega} \frac{\mathrm{F}\,\mathrm{G}\,D}{4(\omega_i \cdot \mathbf{n})(\mathbf{v} \cdot \mathbf{n})} L_{\text{indirect}}(\omega_i \cdot \mathbf{n})\, d\omega_i$$

$$L_{\text{indirect specular}} \approx specularFGD \int_{\Omega} \frac{(\mathbf{h} \cdot \mathbf{n})\, D}{4(\mathbf{h} \cdot \mathbf{n})} L_{\text{indirect}}(\omega_i \cdot \mathbf{n})\, d\omega_i \; *$$



Indirect Specular

$$L_{\text{indirect specular}} \approx \int_{\Omega} \frac{\mathrm{F}\,\mathrm{G}\,D}{4(\omega_\mathrm{i}\cdot\mathbf{n})(\mathbf{v}\cdot\mathbf{n})} L_{\text{indirect}}\,(\omega_\mathrm{i}\cdot\mathbf{n})\,d\omega_\mathrm{i}$$

$$L_{\text{indirect specular}} \approx specularFGD \int_{\Omega} \frac{(\mathbf{h}\cdot\mathbf{n})\,D}{4(\mathbf{h}\cdot\mathbf{n})} L_{\text{indirect}}\,(\omega_\mathrm{i}\cdot\mathbf{n})\,d\omega_\mathrm{i} \;*$$

- Split Sum approximation *[Karis 2013]* is useful for variance reduction
- Blue is precomputed
- Red evaluated with rasterization or ray tracing

Real Shading in Unreal Engine 4 *(Brian Karis)*

Indirect Specular

Small office scene rendered with ray traced indirect specular

Indirect Specular

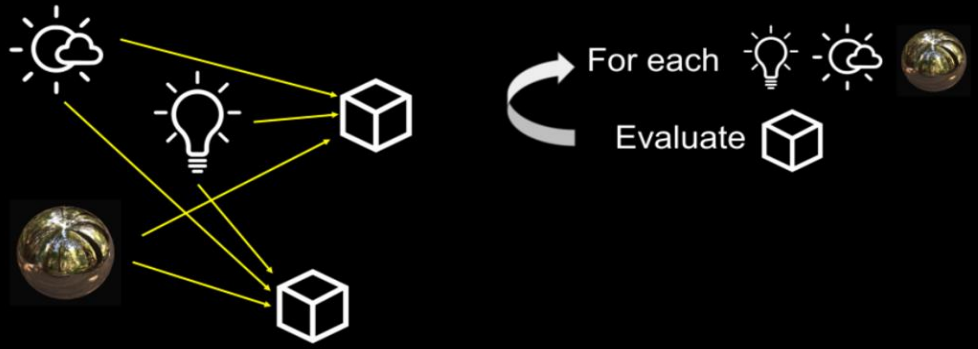
*

Ray traced indirect specular term

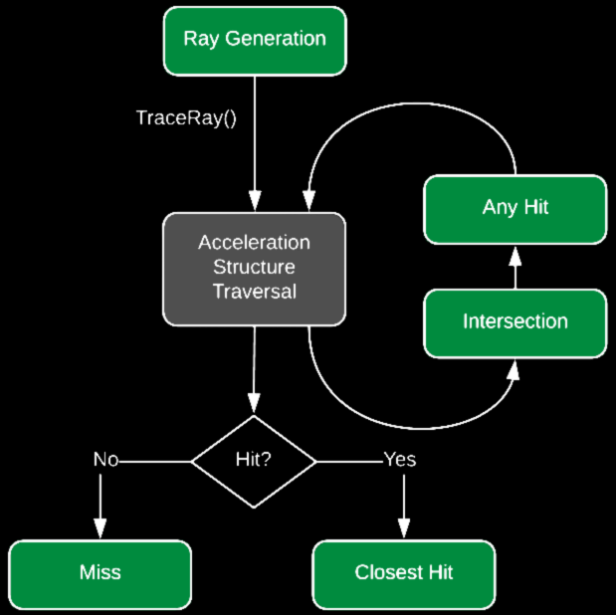
Indirect Specular

Ray traced indirect specular term

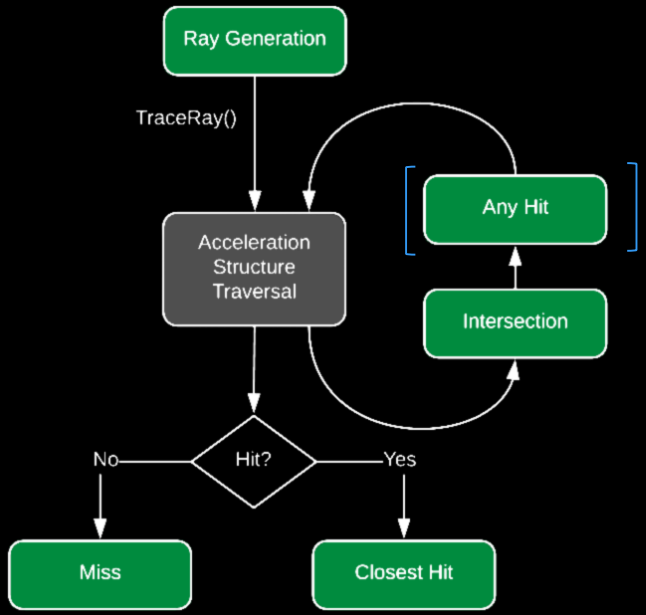
Indirect Specular



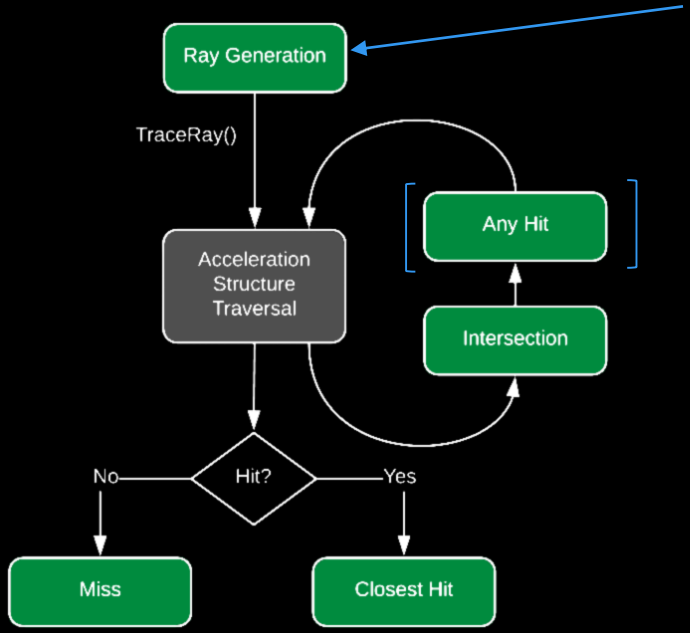
Indirect Specular



Indirect Specular

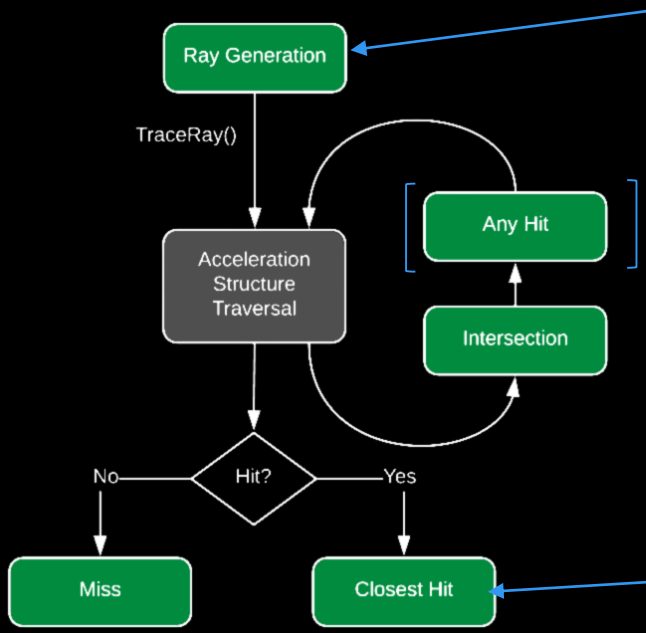


Indirect Specular




```
for(int sampleIdx = 0; sampleIdx < numSamples; ++sampleIdx)
{
    newRay = GenerateRay();
    TraceRay(newRay);
    totalLighting += newRay.lighting;
}
finalColor = totalLighting / (numSamples);
```

Indirect Specular



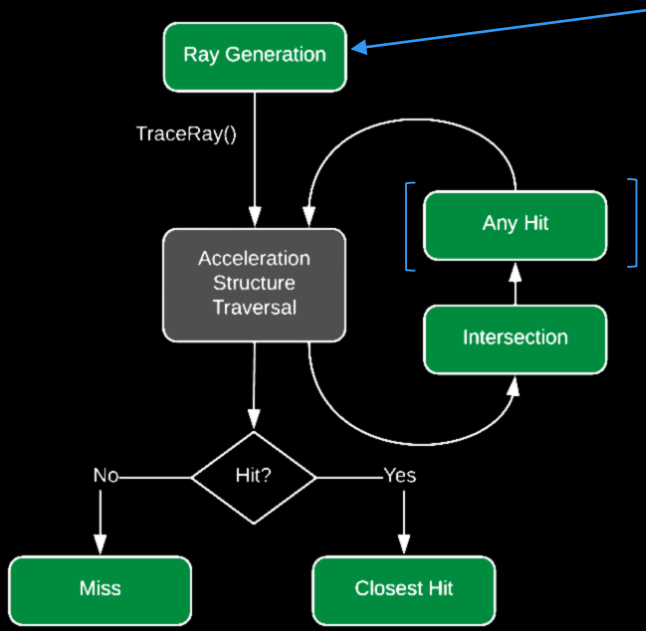

```
for(int sampleIdx = 0; sampleIdx < numSamples; ++sampleIdx)
{
    newRay = GenerateRay();
    TraceRay(newRay);
    totalLighting += newRay.lighting;
}
finalColor = totalLighting / (numSamples);
```

```
for(int lightIdx = 0; lightIdx < numLights; ++lightIdx)
{
    totalLighting += EvaluateLighting();
}
```

Indirect Specular

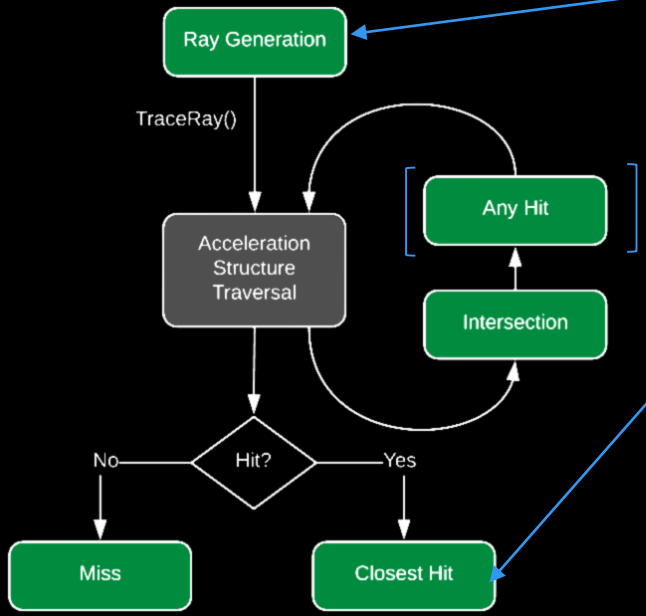
2080Ti @ 1920x1080

Indirect Specular




```
for(int sampleIdx = 0; sampleIdx < numSamples; ++sampleIdx)
{
    newRay = GenerateRay();
    TraceRay(newRay);
    totalLighting += LightLoop(newRay.gbuffer0, newRay.gbuffer1,...);
}
finalColor = totalLighting / (numSamples);
```

Indirect Specular




```
for(int sampleIdx = 0; sampleIdx < numSamples; ++sampleIdx)
{
    newRay = GenerateRay();
    TraceRay(newRay);
    totalLighting += LightLoop(newRay.gbuffer0, newRay.gbuffer1,...);
}
finalColor = totalLighting / (numSamples);
```

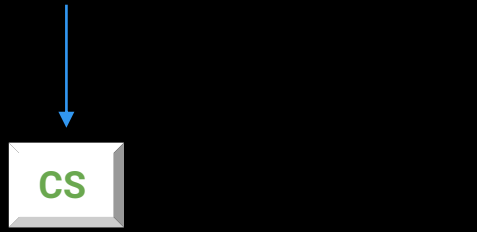
```
currentRay.gbuffer0 = ...
currentRay.gbuffer1 = ...
currentRay.gbuffer2 = ...
currentRay.gbuffer3 = ...
```

Indirect Specular

2080Ti @ 1920x1080

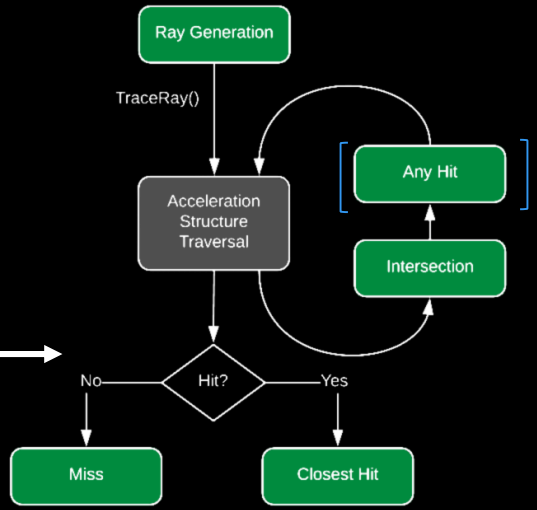
Indirect Specular

```
newRay = GenerateRay();
directionUAV = newRay.dir;
```



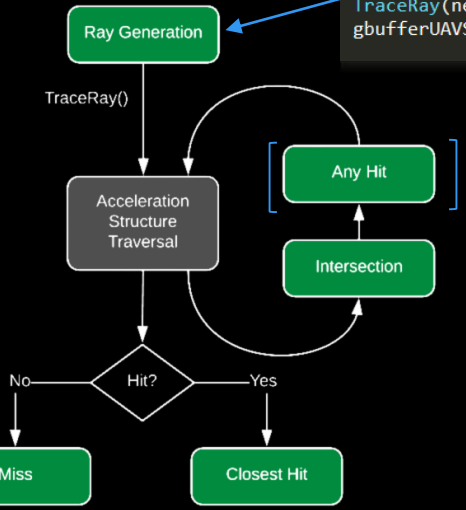
Indirect Specular

```
newRay = GenerateRay();
directionUAV = newRay.dir;
```



Indirect Specular

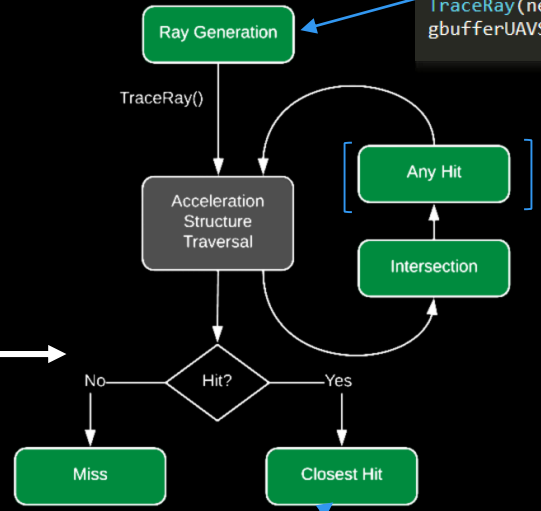
```
newRay = GenerateRay();
directionUAV = newRay.dir;
```




```
newRay.dir = directionUAV;
newRay.origin = readPosition(DepthBuffer);
TraceRay(newRay);
gbufferUAVS = newRay.gbuffer0_1_2_3;
```

Indirect Specular

```
newRay = GenerateRay();
directionUAV = newRay.dir;
```

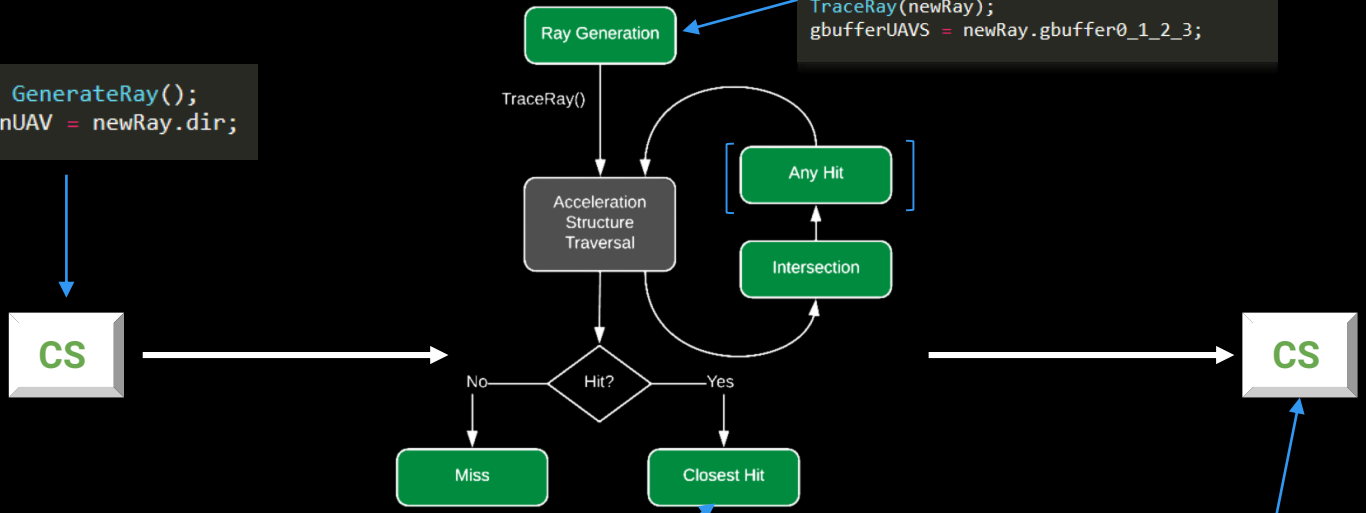



```
newRay.dir = directionUAV;
newRay.origin = readPosition(DepthBuffer);
TraceRay(newRay);
gbufferUAVS = newRay.gbuffer0_1_2_3;
```

```
currentRay.gbuffer0 = ...
currentRay.gbuffer1 = ...
currentRay.gbuffer2 = ...
currentRay.gbuffer3 = ...
```

Indirect Specular

```
newRay = GenerateRay();
directionUAV = newRay.dir;
```



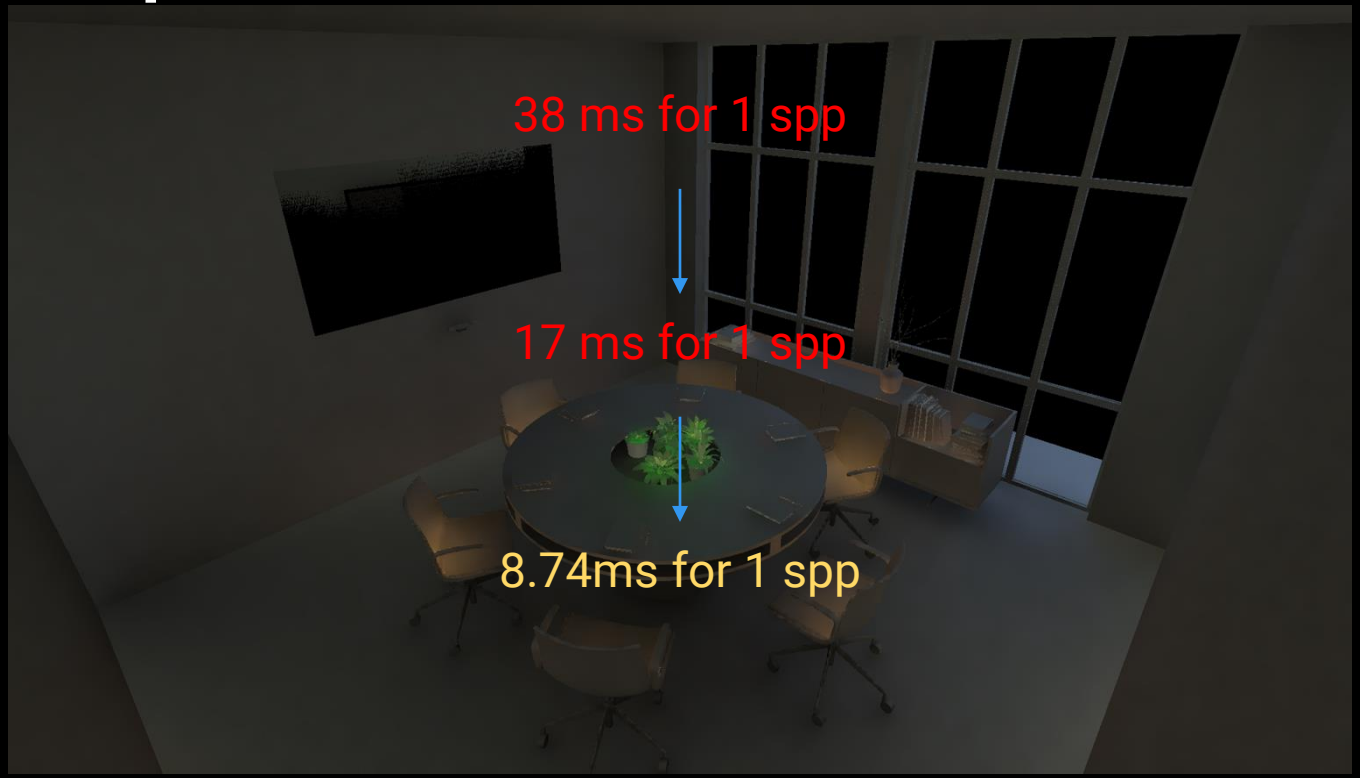

```
newRay.dir = directionUAV;
newRay.origin = readPosition(DepthBuffer);
TraceRay(newRay);
gbufferUAVS = newRay.gbuffer0_1_2_3;
```

```
currentRay.gbuffer0 = ...
currentRay.gbuffer1 = ...
currentRay.gbuffer2 = ...
currentRay.gbuffer3 = ...
```

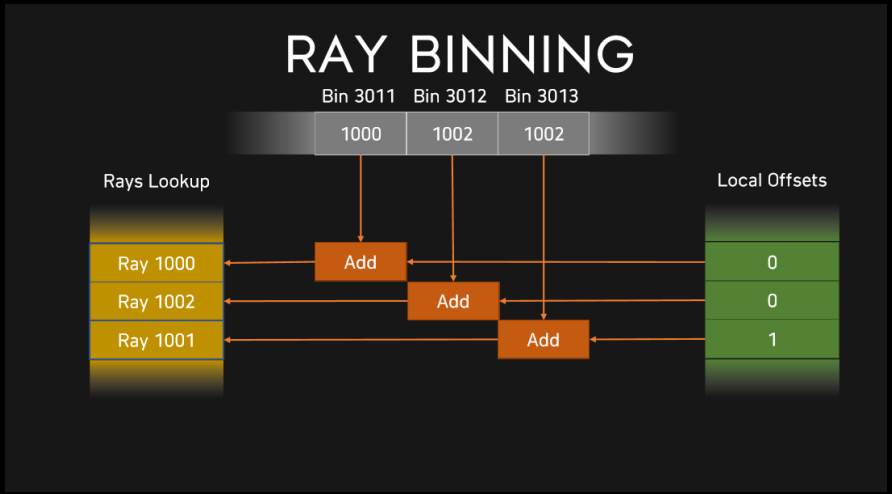
```
gbuffer0, gbuffer1, gbuffer2, gbuffer3 = gbufferUAVS
totalLightingUAV += LightLoop(gbuffer0, gbuffer1,...) / numSamples;
```

Indirect Specular

2080Ti @ 1920x1080

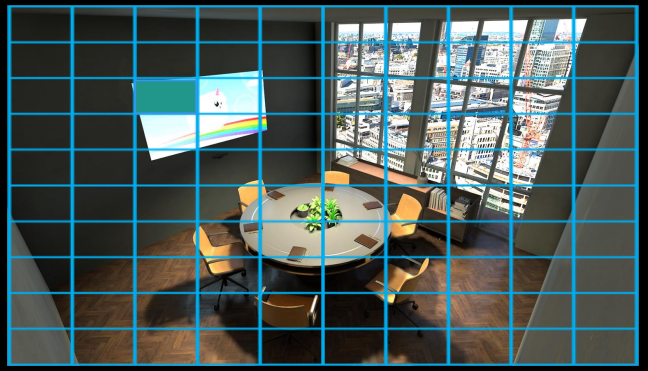


Ray Binning



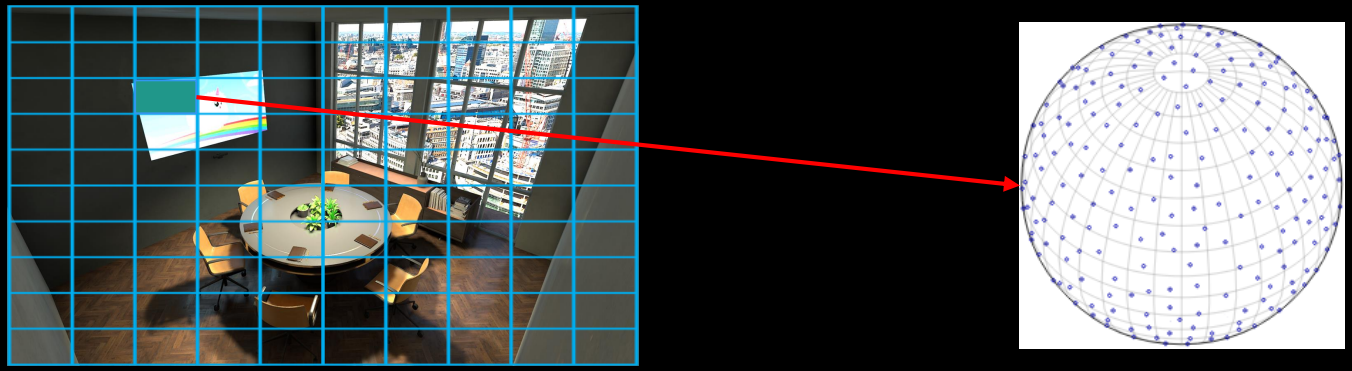
"It Just Works": Ray-Traced Reflections in 'Battlefield V'
Johannes Deligiannis Jan Schmid EA DICE

Ray Binning



Screen tiles of 32x32 pixels for binning

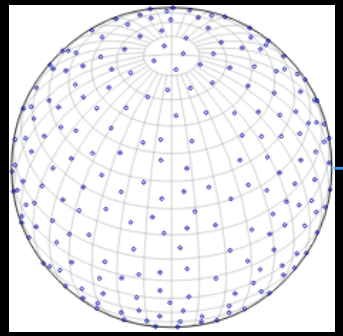
Ray Binning



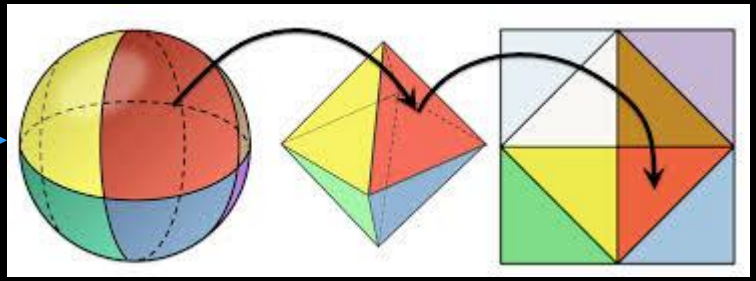
Screen tiles of 32x32 pixels for binning

Generated rays in the unit sphere for a tile

Ray Binning



Generated rays in the unit sphere for a tile

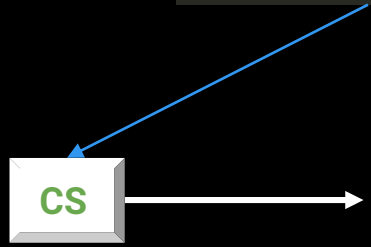


Octahedral space for ray direction binning

Indirect Specular

Indirect Specular

```
newRay = GenerateRay();
directionUAV = newRay.dir;
```



Indirect Specular

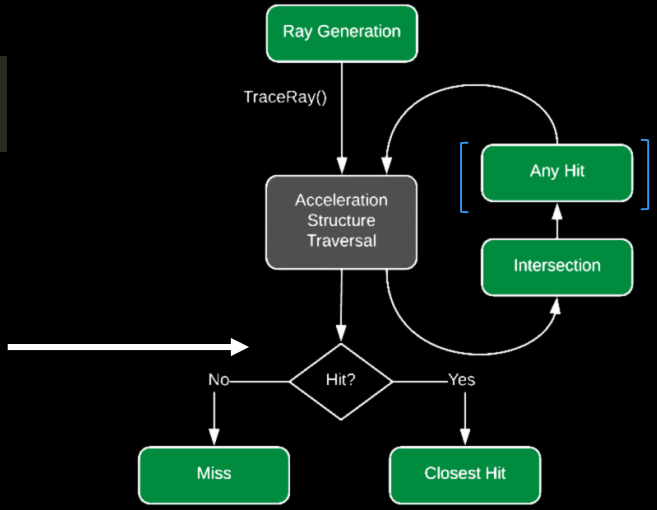
```
newRay = GenerateRay();
directionUAV = newRay.dir;
```


```
rayBinIndex = GetRayBinIndex(directionUAV);

atomic_add(lds[rayBinIndex]);

barrier();

ComputePixelOffsets();

barrier();

binResultUAV = offset;
```

Indirect Specular

```
newRay = GenerateRay();
directionUAV = newRay.dir;
```


```
rayBinIndex = GetRayBinIndex(directionUAV);

atomic_add(lds[rayBinIndex]);

barrier();

ComputePixelOffsets();

barrier();

binResultUAV = offset;
```



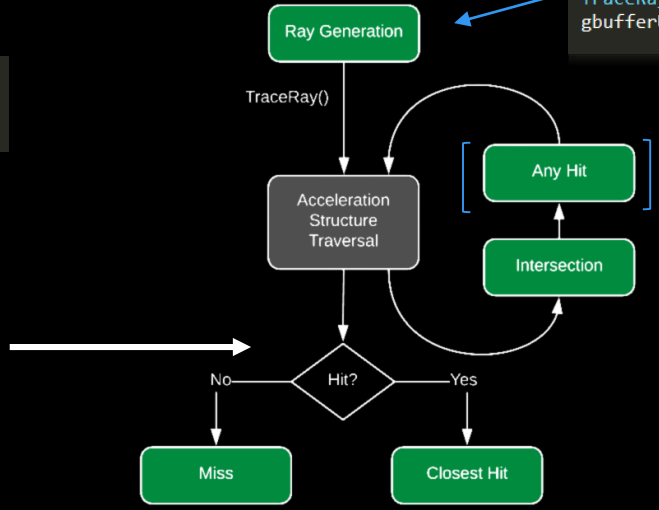
unity

Indirect Specular

```
newRay = GenerateRay();
directionUAV = newRay.dir;
```


```
rayBinIndex = GetRayBinIndex(directionUAV);

atomic_add(lds[rayBinIndex]);

barrier();

ComputePixelOffsets();

barrier();

binResultUAV = offset;
```

```
newRay.dir = directionUAV;
newRay.origin = readPosition(DepthBuffer);
TraceRay(newRay);
gbufferUAVS = newRay.gbuffer0_1_2_3;
```



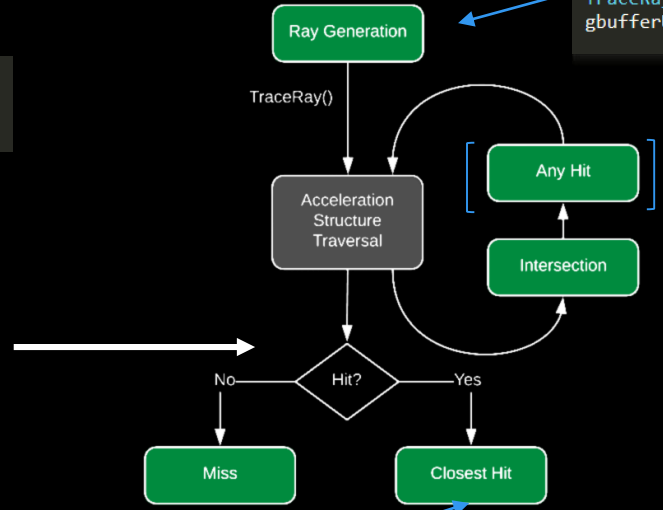
unity

Indirect Specular

```
newRay = GenerateRay();
directionUAV = newRay.dir;
```


```
rayBinIndex = GetRayBinIndex(directionUAV);

atomic_add(lds[rayBinIndex]);

barrier();

ComputePixelOffsets();

barrier();

binResultUAV = offset;
```

```
newRay.dir = directionUAV;
newRay.origin = readPosition(DepthBuffer);
TraceRay(newRay);
gbufferUAVS = newRay.gbuffer0_1_2_3;
```

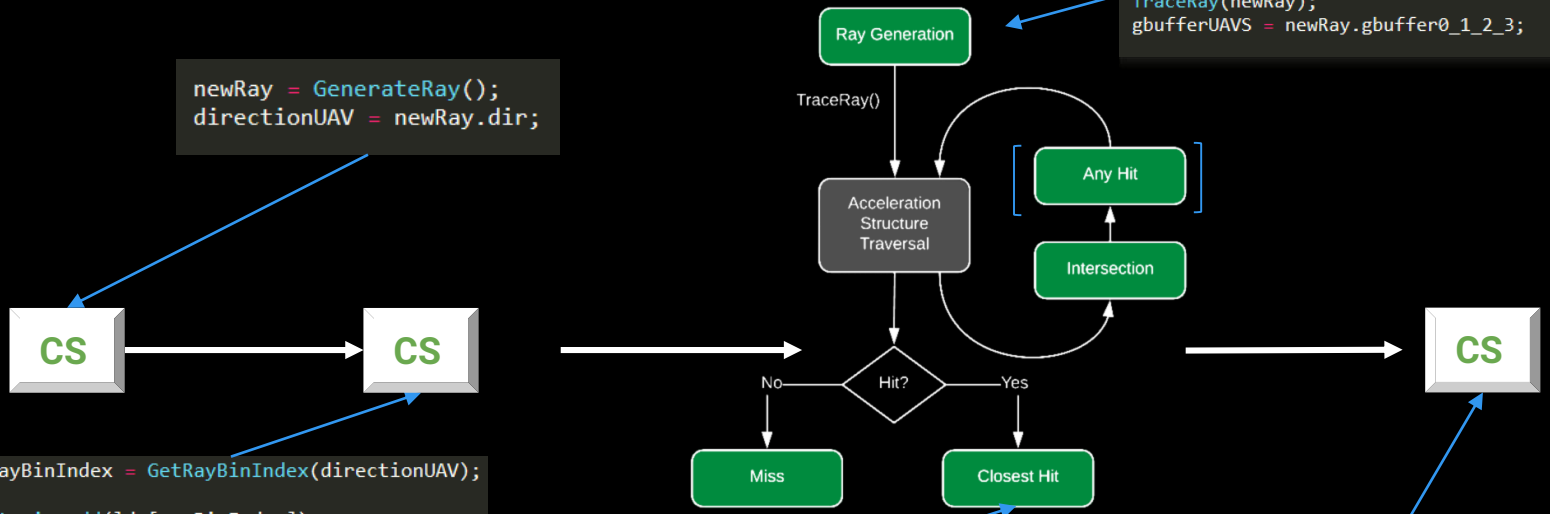



```
currentRay.gbuffer0 = ...
currentRay.gbuffer1 = ...
currentRay.gbuffer2 = ...
currentRay.gbuffer3 = ...
```

Indirect Specular

```
newRay = GenerateRay();
directionUAV = newRay.dir;
```

```
newRay.dir = directionUAV;
newRay.origin = readPosition(DepthBuffer);
TraceRay(newRay);
gbufferUAVS = newRay.gbuffer0_1_2_3;
```



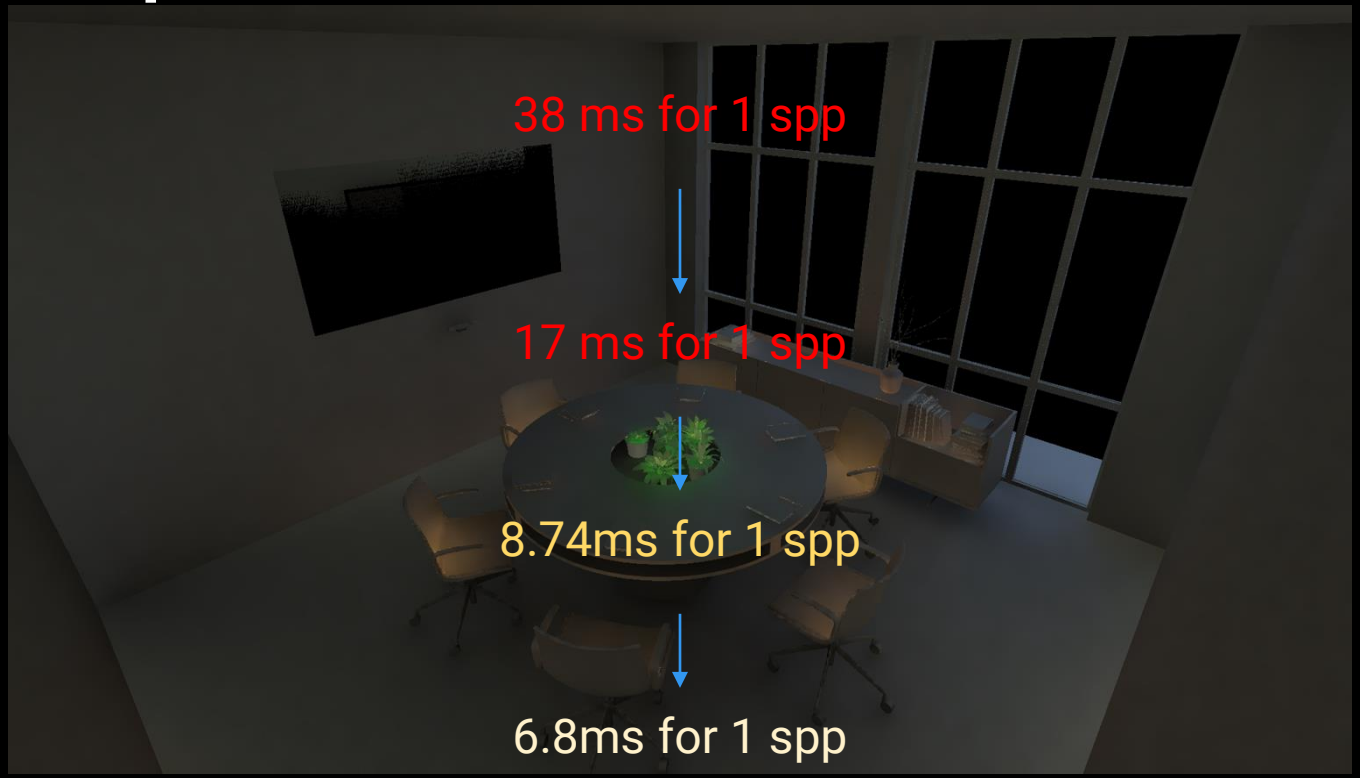

```
rayBinIndex = GetRayBinIndex(directionUAV);

atomic_add(lds[rayBinIndex]);

barrier();

ComputePixelOffsets();

barrier();

binResultUAV = offset;
```

```
currentRay.gbuffer0 = ...
currentRay.gbuffer1 = ...
currentRay.gbuffer2 = ...
currentRay.gbuffer3 = ...
```

```
gbuffer0, gbuffer1, gbuffer2, gbuffer3 = gbufferUAVS
totalLightingUAV += LightLoop(gbuffer0, gbuffer1,...) / numSamples;
```

Indirect Specular

2080Ti @ 1920x1080



Indirect Specular

Indirect Specular

2080Ti @ 1920x1080

unity

Indirect Specular – Additional compromises

Area Shadow

Area Shadow

$$L_o(\mathbf{x}, \omega_o, \lambda, t) = L_e(\mathbf{x}, \omega_o, \lambda, t) + L_{direct}(\mathbf{x}, \omega_o, \lambda, t) + L_{indirect}(\mathbf{x}, \omega_o, \lambda, t)$$

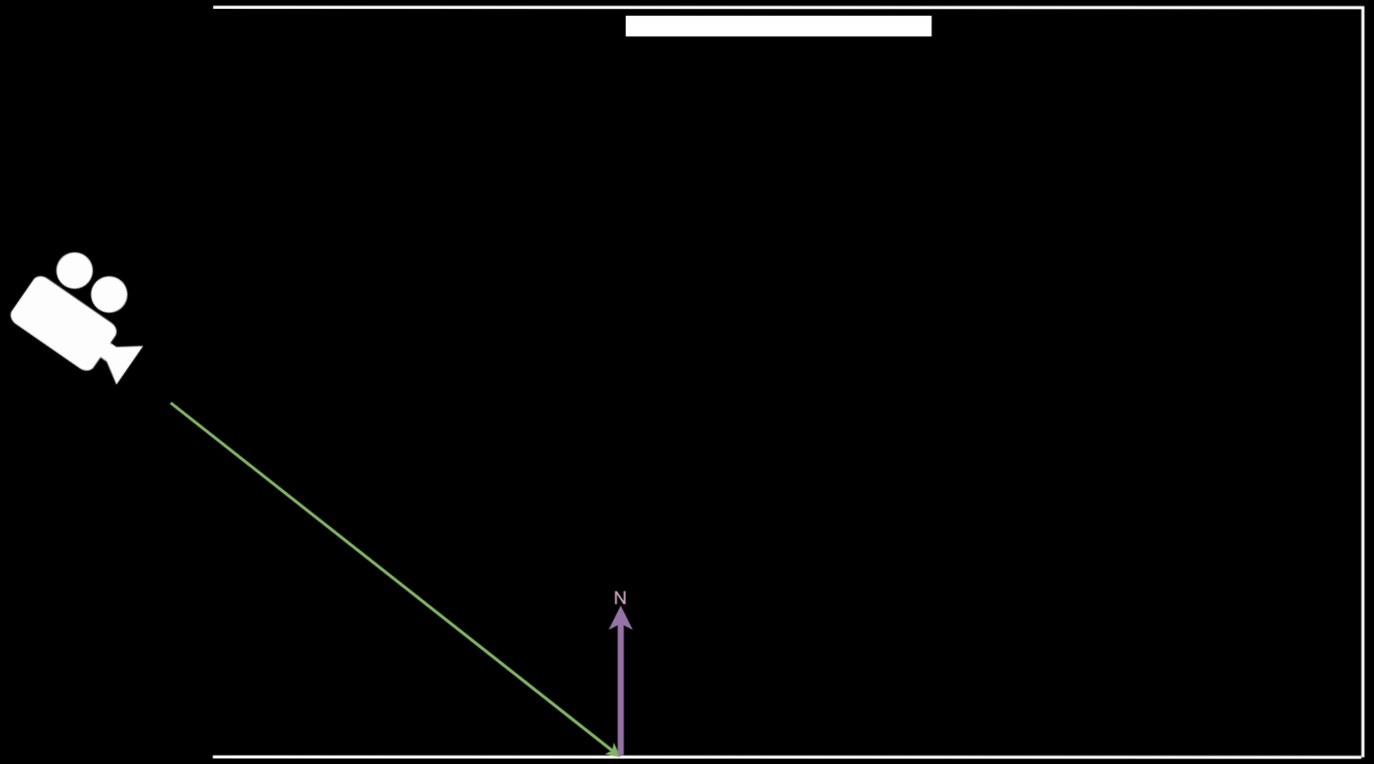
$$L_o = \int_\Omega BSDF \cdot L_i \cdot V \, d\omega$$

Area Shadow

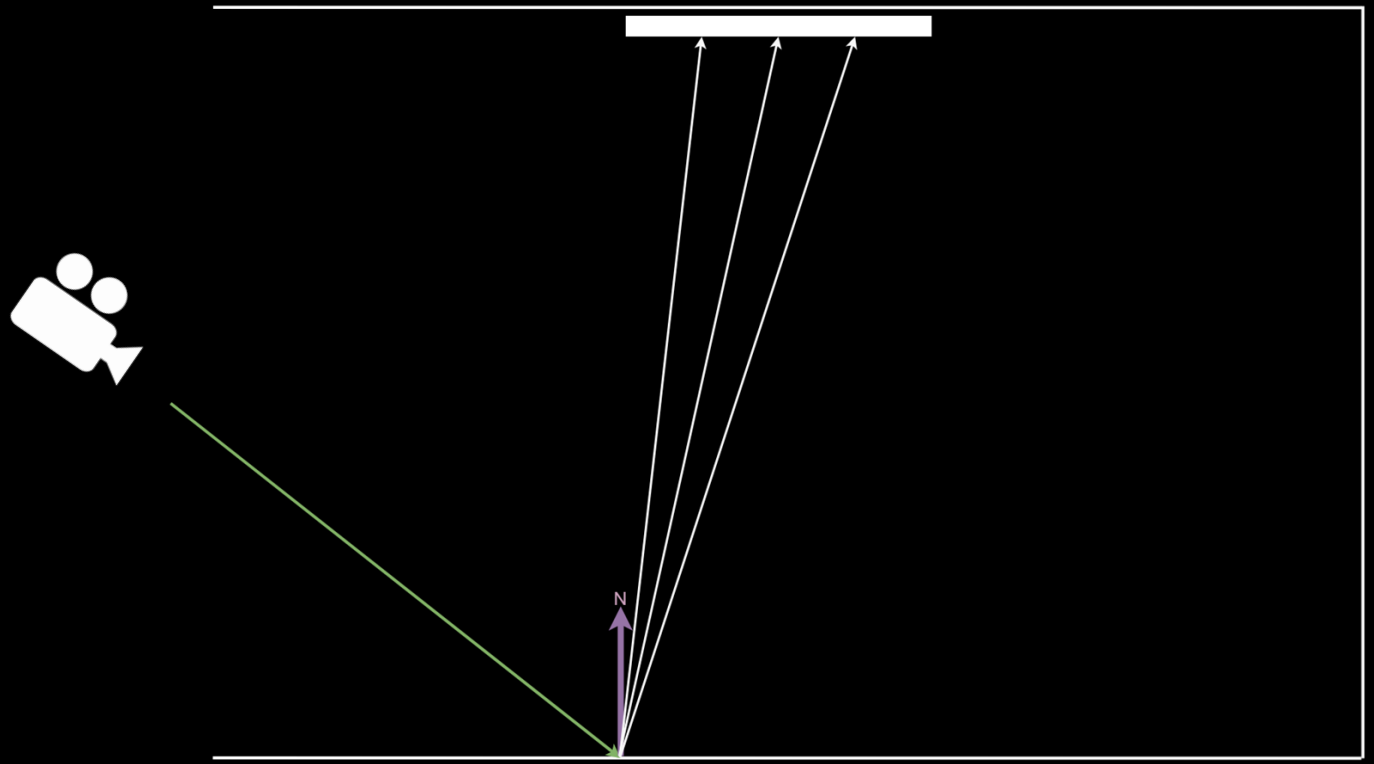
$$L_o = \int_\Omega BSDF \cdot L_i \cdot V \, d\omega$$

Sponza Scene - Reference

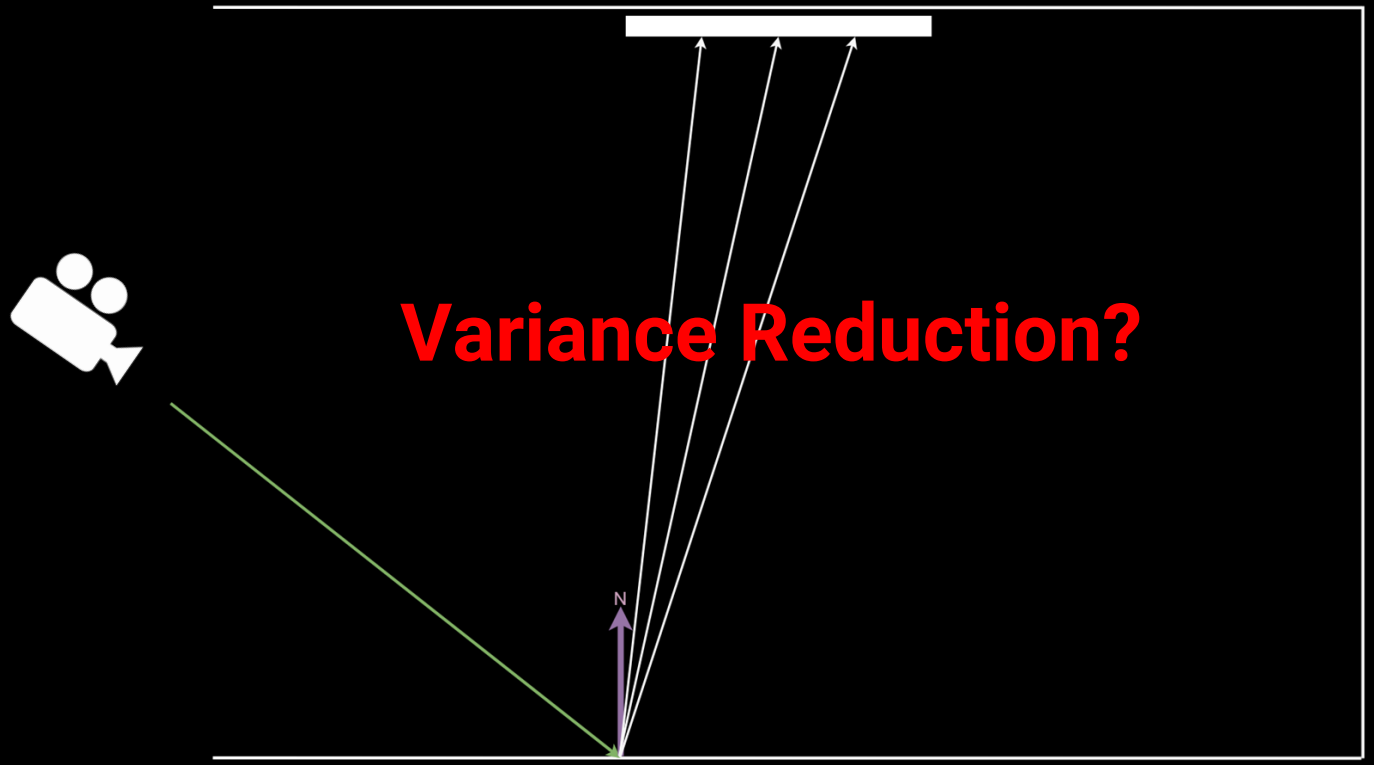
Area Shadow



Area Shadow



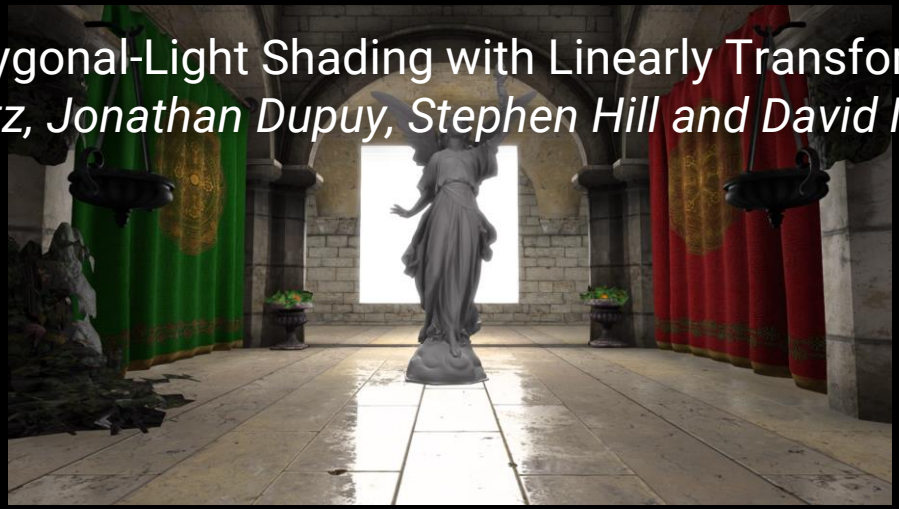
Area Shadow



Area Shadow

$$L_o \approx \int_\Omega BSDF \cdot L_i \, d\omega$$

“Real-Time Polygonal-Light Shading with Linearly Transformed Cosines”
Eric Heitz, Jonathan Dupuy, Stephen Hill and David Neubelt

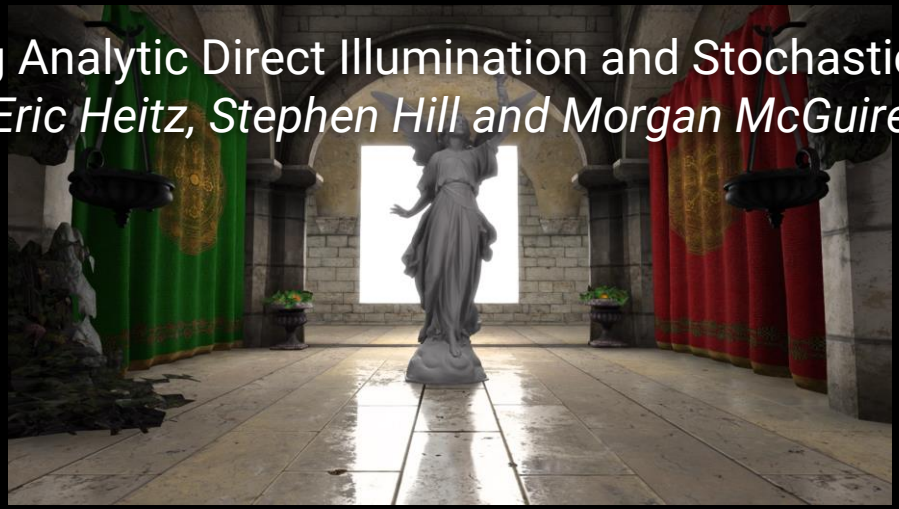


Sponza Scene - LTC only

Area Shadow

$$L_o \approx \frac{|| \int_\Omega BSDF \cdot L_i \cdot V ||}{|| \int_\Omega BSDF \cdot L_i ||} \int_\Omega BSDF \cdot L_i$$

“Combining Analytic Direct Illumination and Stochastic Shadows”
Eric Heitz, Stephen Hill and Morgan McGuire



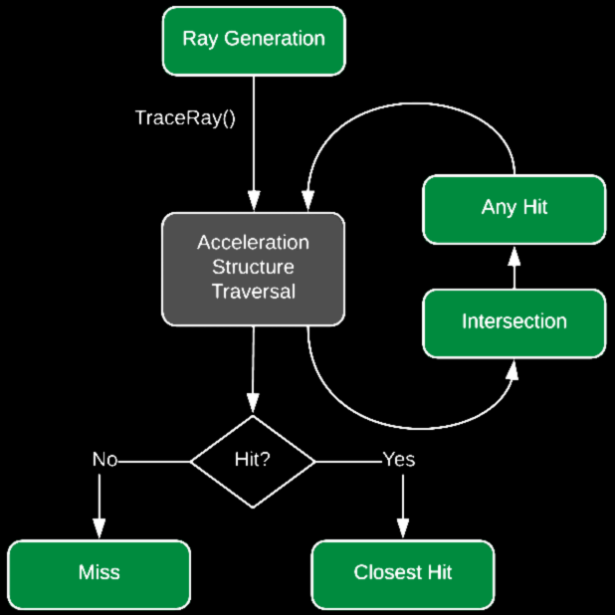
Sponza Scene - LTC + Stochastic Shadow

Area Shadow - Visibility

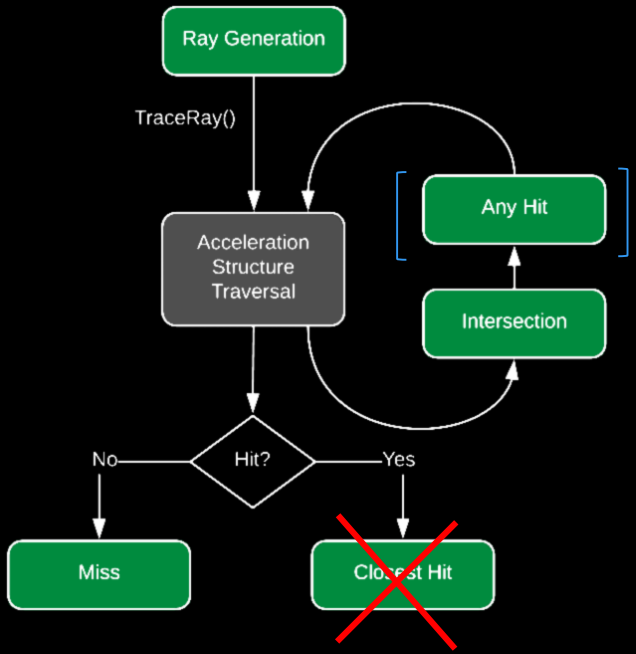
$$L_o = \int_\Omega BSDF \cdot L_i \, d\omega \int_\Omega L_i \cdot V \, d\omega$$

Area Shadow - Stochastic

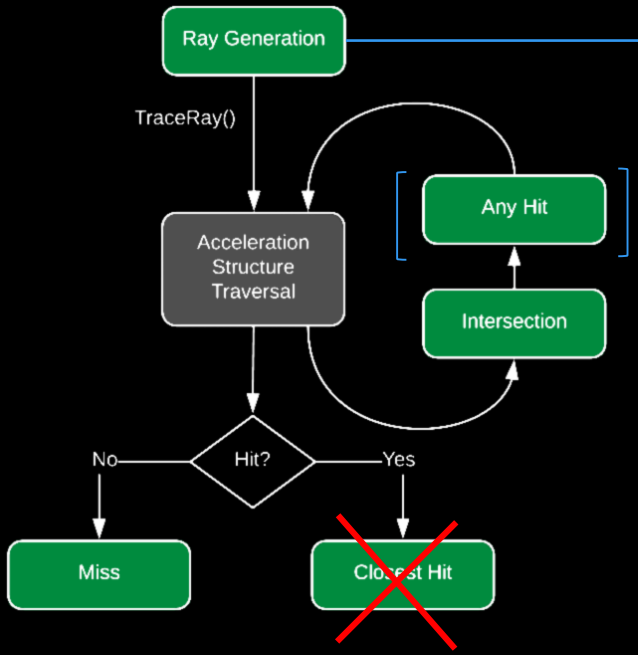
Area Shadow - Stochastic



Area Shadow - Stochastic

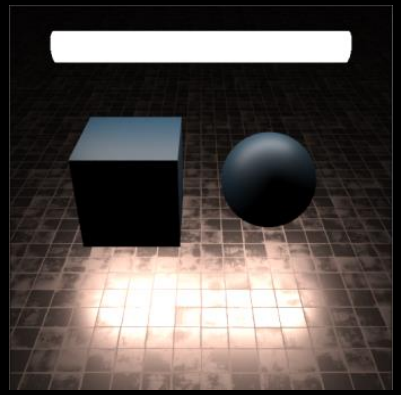


Area Shadow - Stochastic

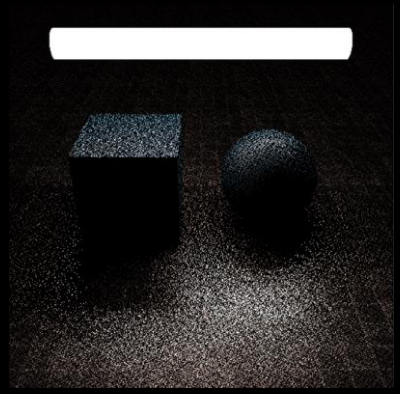



```
sq = InitSphericalQuad();

for(int sampleIdx = 0; sampleIdx < numSamples; ++sampleIdx)
{
    sample = generateSample();
    ray = generateRay();
    TraceRay(ray);
    Un += BSDF;
    Sn += BSDF * ray.hit;
}
UnUAV = Un / numSamples;
SnUAV = Sn / numSamples;
```

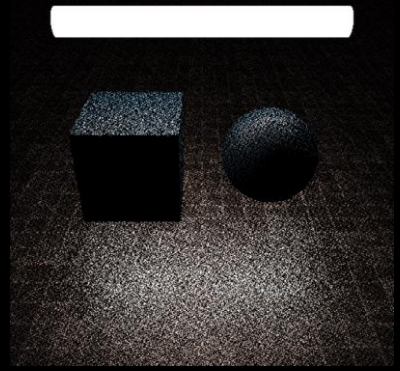
Area Shadow



LTC

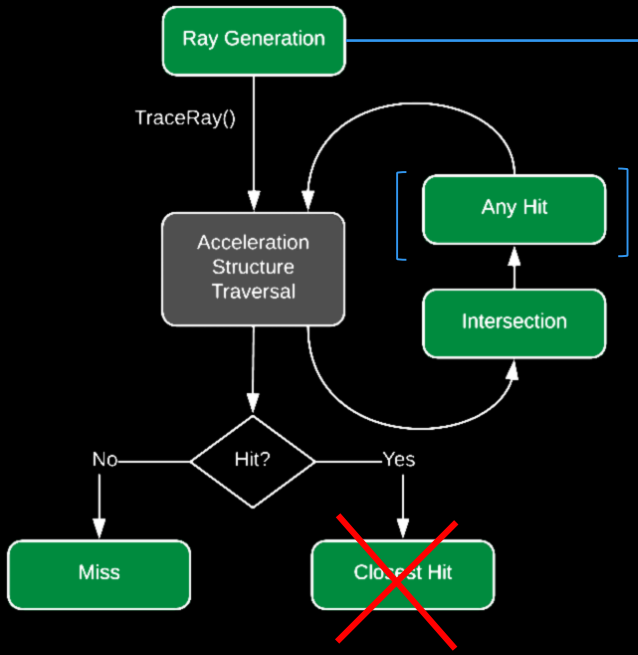


Sn



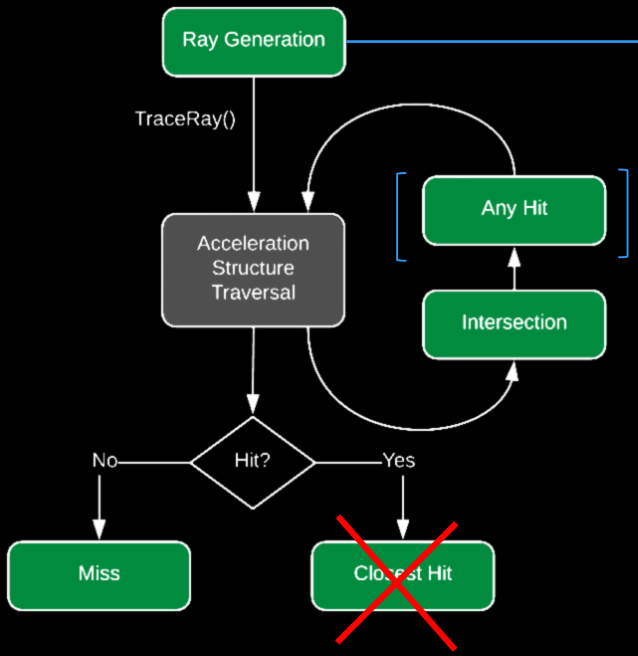
Un

Area Shadow - Stochastic




```
sq = InitSphericalQuad();

for(int sampleIdx = 0; sampleIdx < numSamples; ++sampleIdx)
{
    sample = generateSample();
    ray = generateRay();
    TraceRay(ray);
    Un += BSDF;
    Sn += BSDF * ray.hit;
}
UnUAV = Un / numSamples;
SnUAV = Sn / numSamples;
```

Area Shadow - Stochastic




```
sq = InitSphericalQuad();

for(int sampleIdx = 0; sampleIdx < numSamples; ++sampleIdx)
{
    sample = generateSample();
    ray = generateRay();
    TraceRay(ray);
    Un += BSDF;
    Sn += BSDF * ray.hit;
}
UnUAV = Un / numSamples;
SnUAV = Sn / numSamples;
```

CS

```
denoiseHorizontal(Un);
denoiseHorizontal(Sn);
```

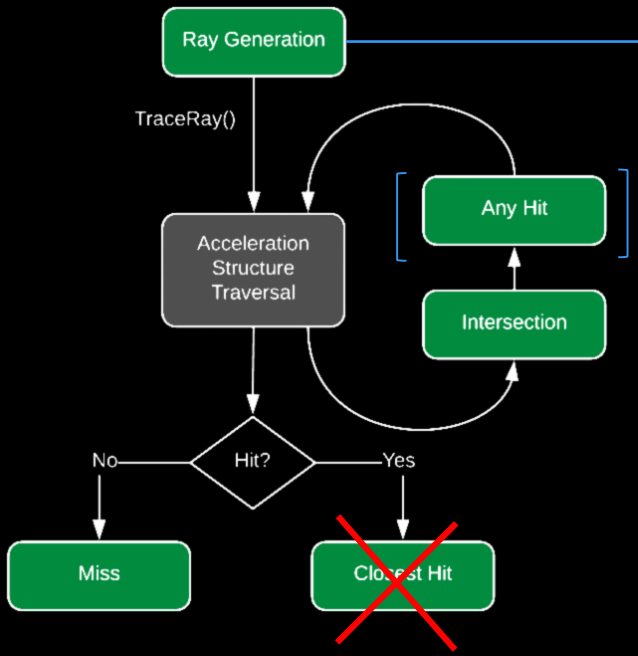
CS

```
U = denoiseVertical(Un);
S = denoiseVertical(Sn);
shadowUAV = S / U;
```

Area Shadow - Stochastic

2080Ti @ 1920x1080

Area Shadow - Stochastic




```
sq = InitSphericalQuad();

for(int sampleIdx = 0; sampleIdx < numSamples; ++sampleIdx)
{
    sample = generateSample();
    ray = generateRay();
    TraceRay(ray);
    Un += BSDF;
    Sn += BSDF * ray.hit;
}
UnUAV = Un / numSamples;
SnUAV = Sn / numSamples;
```

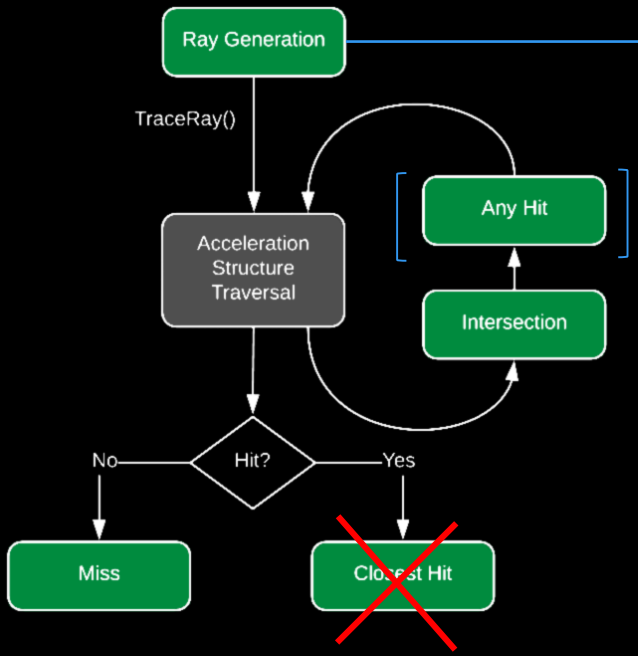
CS

```
denoiseHorizontal(Un);
denoiseHorizontal(Sn);
```

CS

```
U = denoiseVertical(Un);
S = denoiseVertical(Sn);
shadowUAV = S / U;
```

Area Shadow - Stochastic




```
sq = InitSphericalQuad();

for(int sampleIdx = 0; sampleIdx < numSamples; ++sampleIdx)
{
    sample = generateSample();
    ray = generateRay();
    TraceRay(ray);
    Un += BSDF;
    Sn += BSDF * ray.hit;
}
UnUAV = Un / numSamples;
SnUAV = Sn / numSamples;
```

Use simplest shader variant

CS

CS

```
denoiseHorizontal(Un);
denoiseHorizontal(Sn);
```

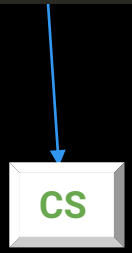
```
U = denoiseVertical(Un);
S = denoiseVertical(Sn);
shadowUAV = S / U;
```

Area Shadow - Stochastic

2080Ti @ 1920x1080

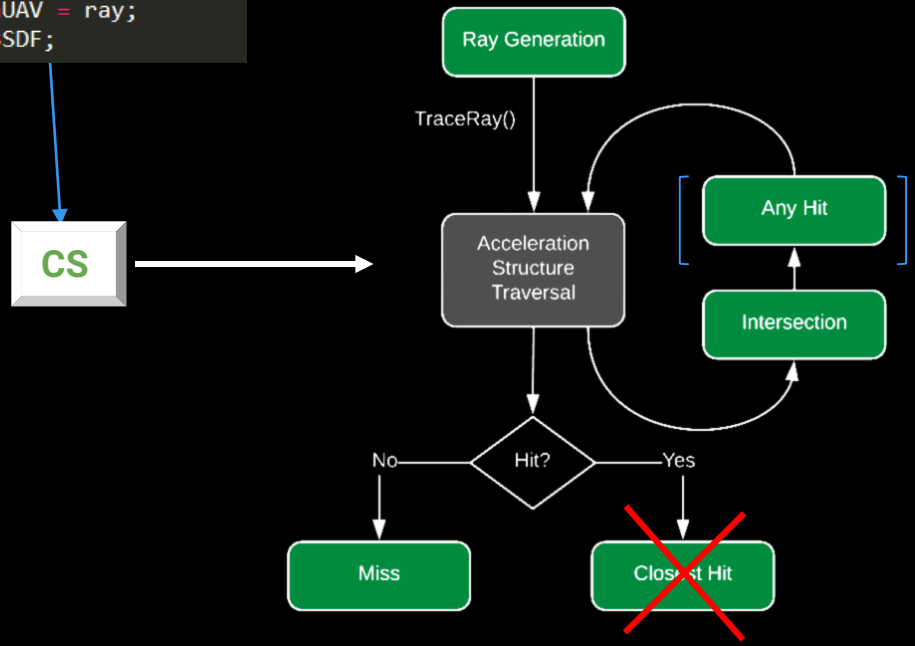
Area Shadow - Stochastic

```
sq = InitSphericalQuad();
sample = pickNextSample();
ray = generateRay();
directionUAV = ray;
UnUAV = BSDF;
```



Area Shadow - Stochastic

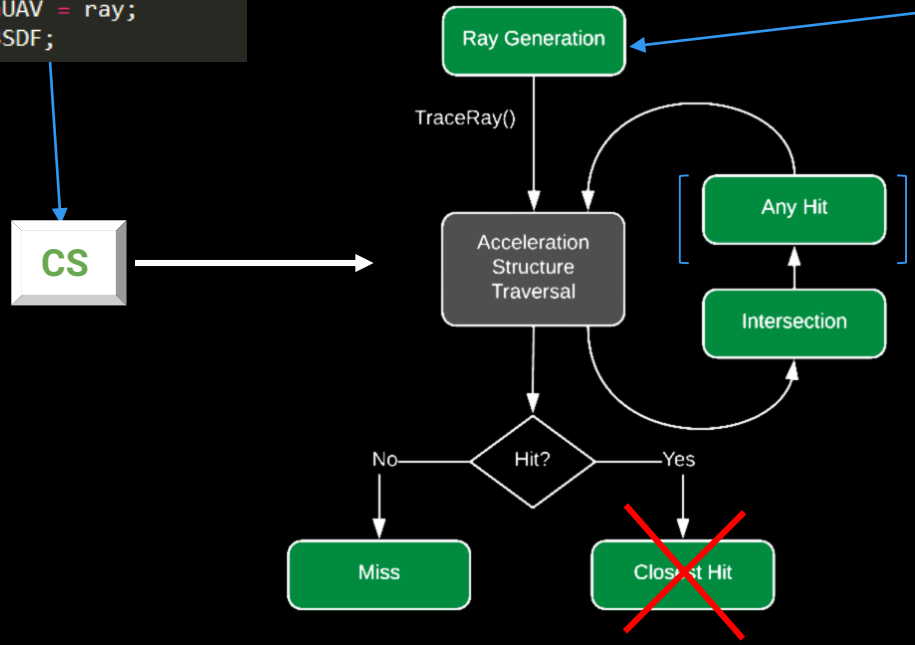
```
sq = InitSphericalQuad();
sample = pickNextSample();
ray = generateRay();
directionUAV = ray;
UnUAV = BSDF;
```



Area Shadow - Stochastic

```
sq = InitSphericalQuad();
sample = pickNextSample();
ray = generateRay();
directionUAV = ray;
UnUAV = BSDF;
```

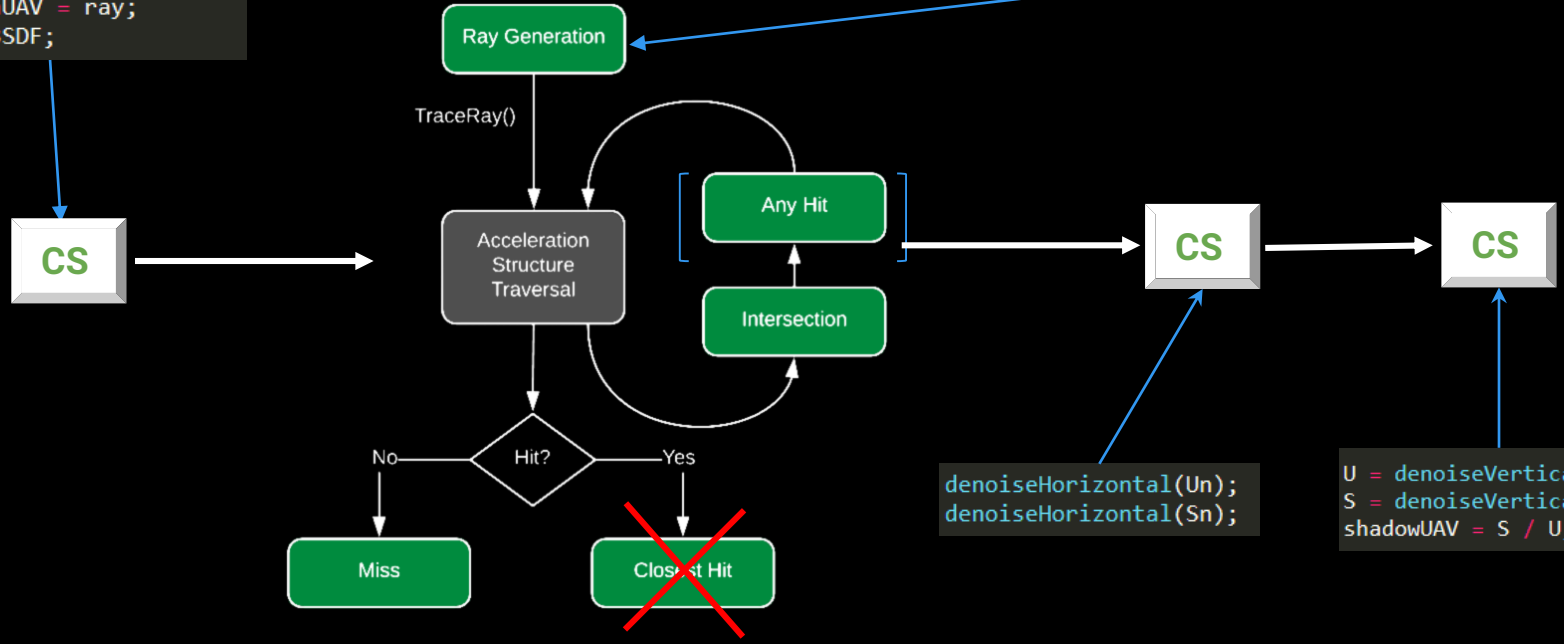
```
newRay.dir = directionUAV;
newRay.origin = readPosition(DepthBuffer);
TraceRay(newRay);
SnUAV = UnSV * newRay.hit;
```



Area Shadow - Stochastic

```
sq = InitSphericalQuad();
sample = pickNextSample();
ray = generateRay();
directionUAV = ray;
UnUAV = BSDF;
```

```
newRay.dir = directionUAV;
newRay.origin = readPosition(DepthBuffer);
TraceRay(newRay);
SnUAV = UnSV * newRay.hit;
```




```
denoiseHorizontal(Un);
denoiseHorizontal(Sn);
```

```
U = denoiseVertical(Un);
S = denoiseVertical(Sn);
shadowUAV = S / U;
```

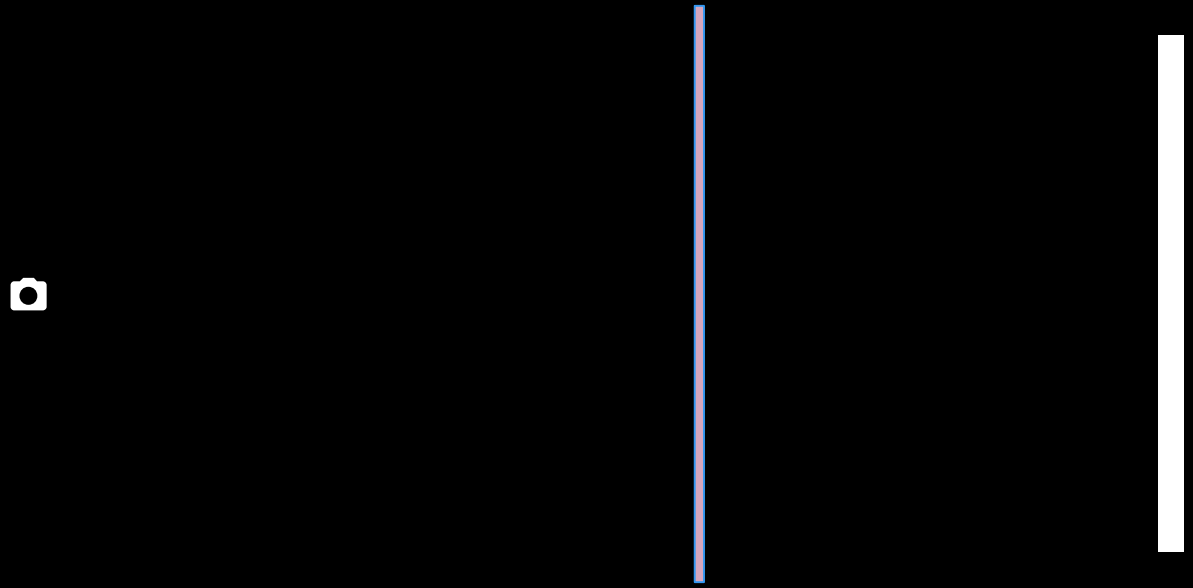
Area Shadow - Stochastic

2080Ti @ 1920x1080

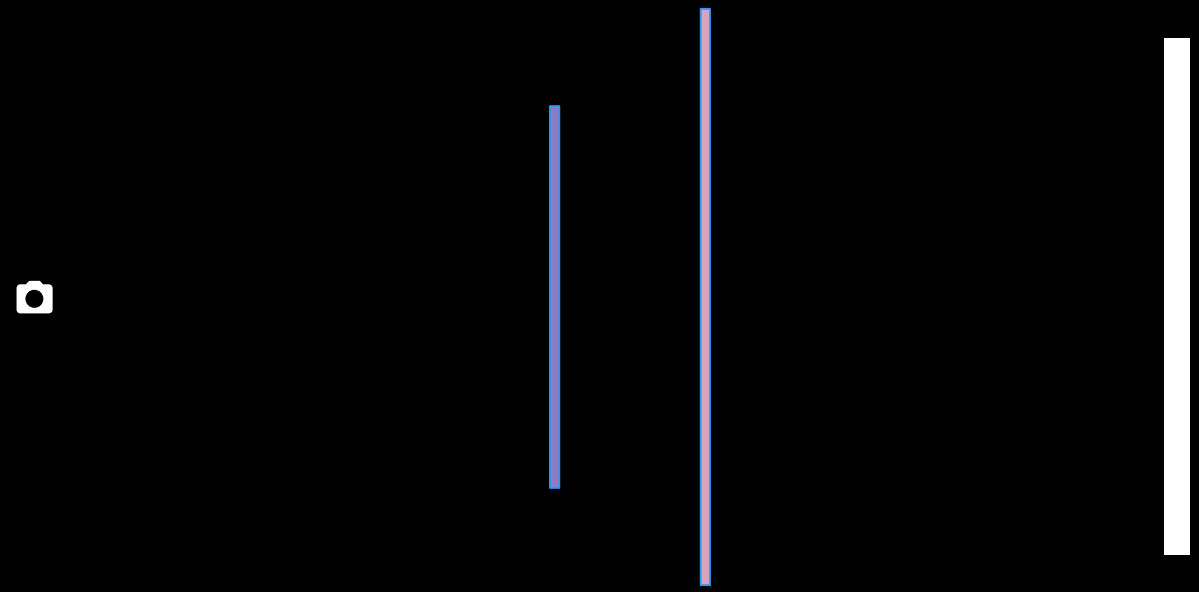
Recursive tracing

Alpha Blend Transparency

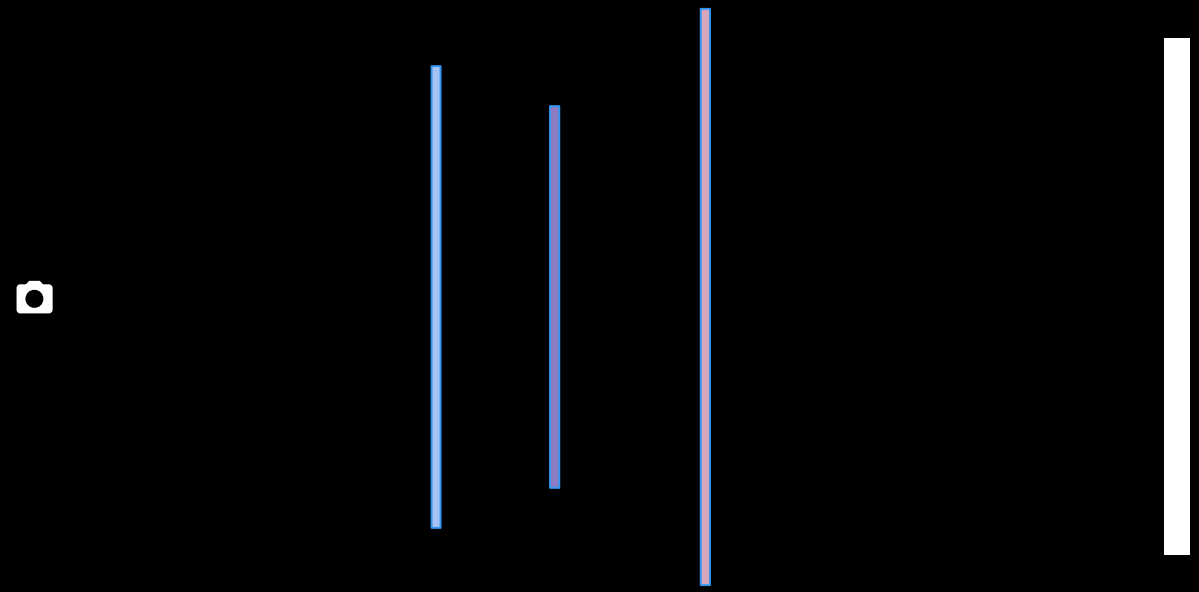
Alpha Blend Transparency



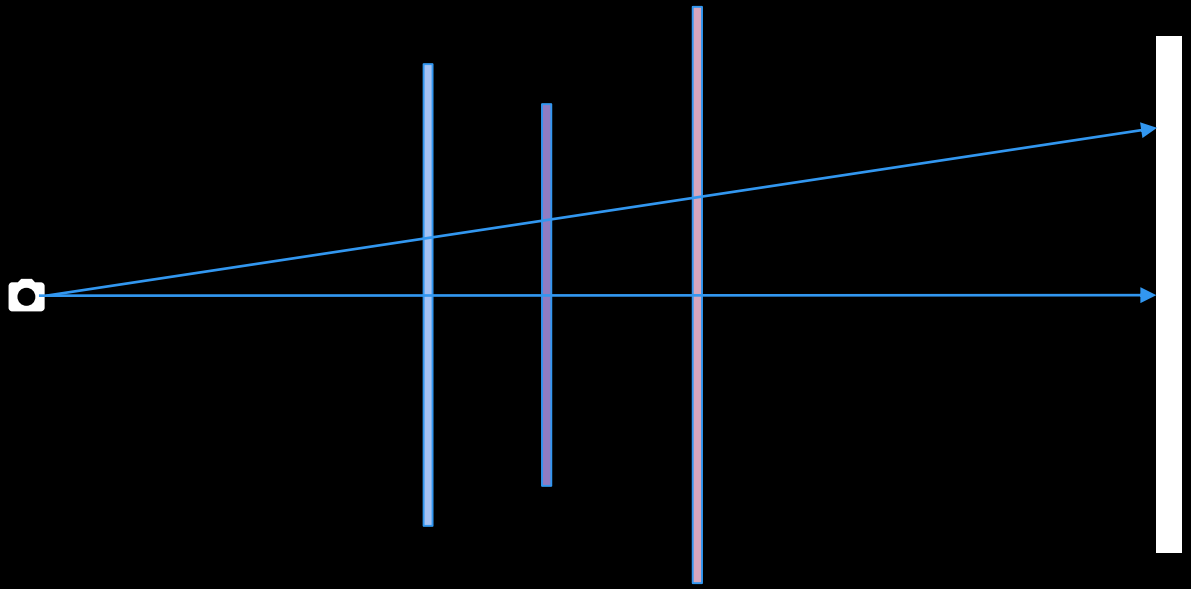
Alpha Blend Transparency



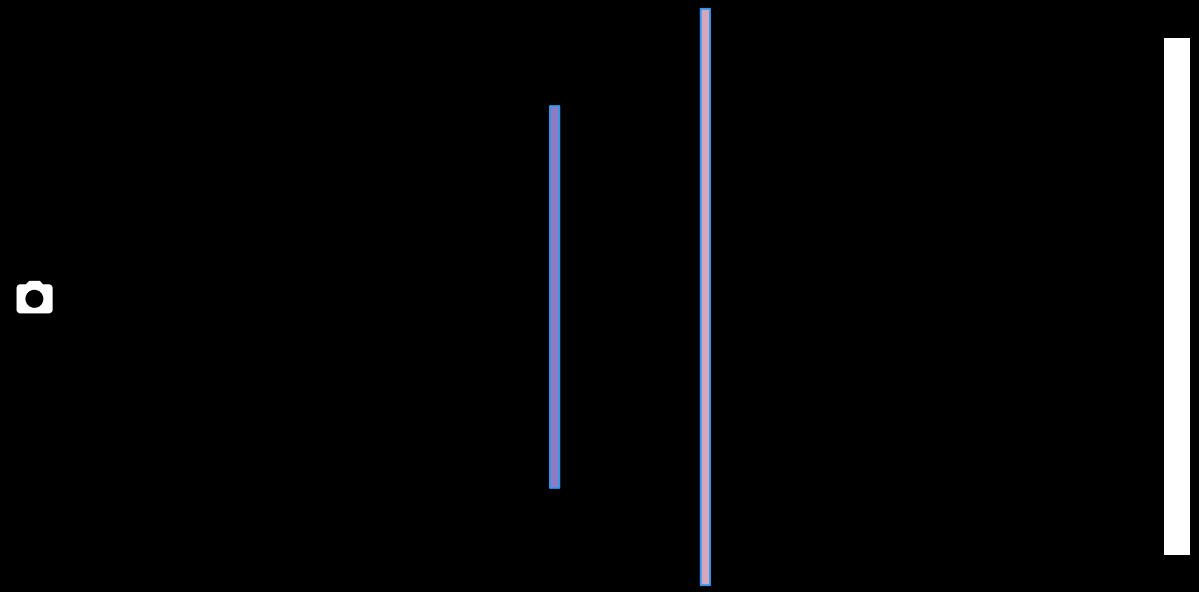
Alpha Blend Transparency



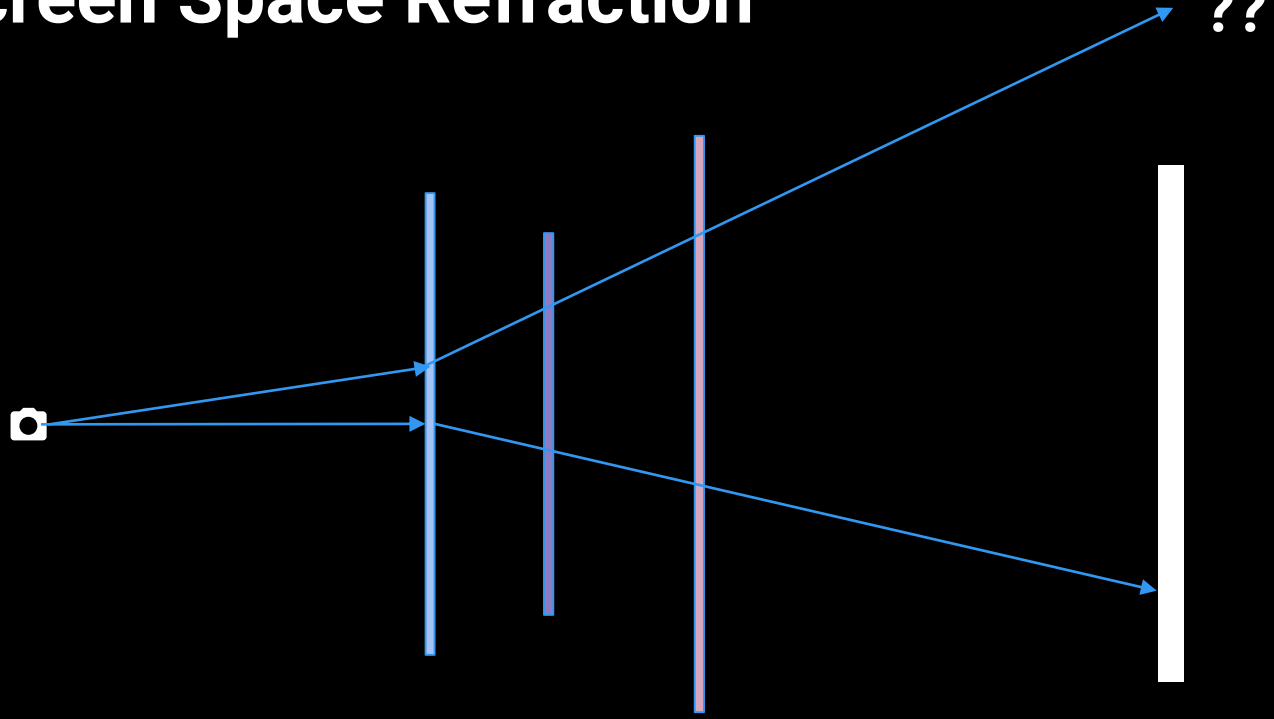
Alpha Blend Transparency



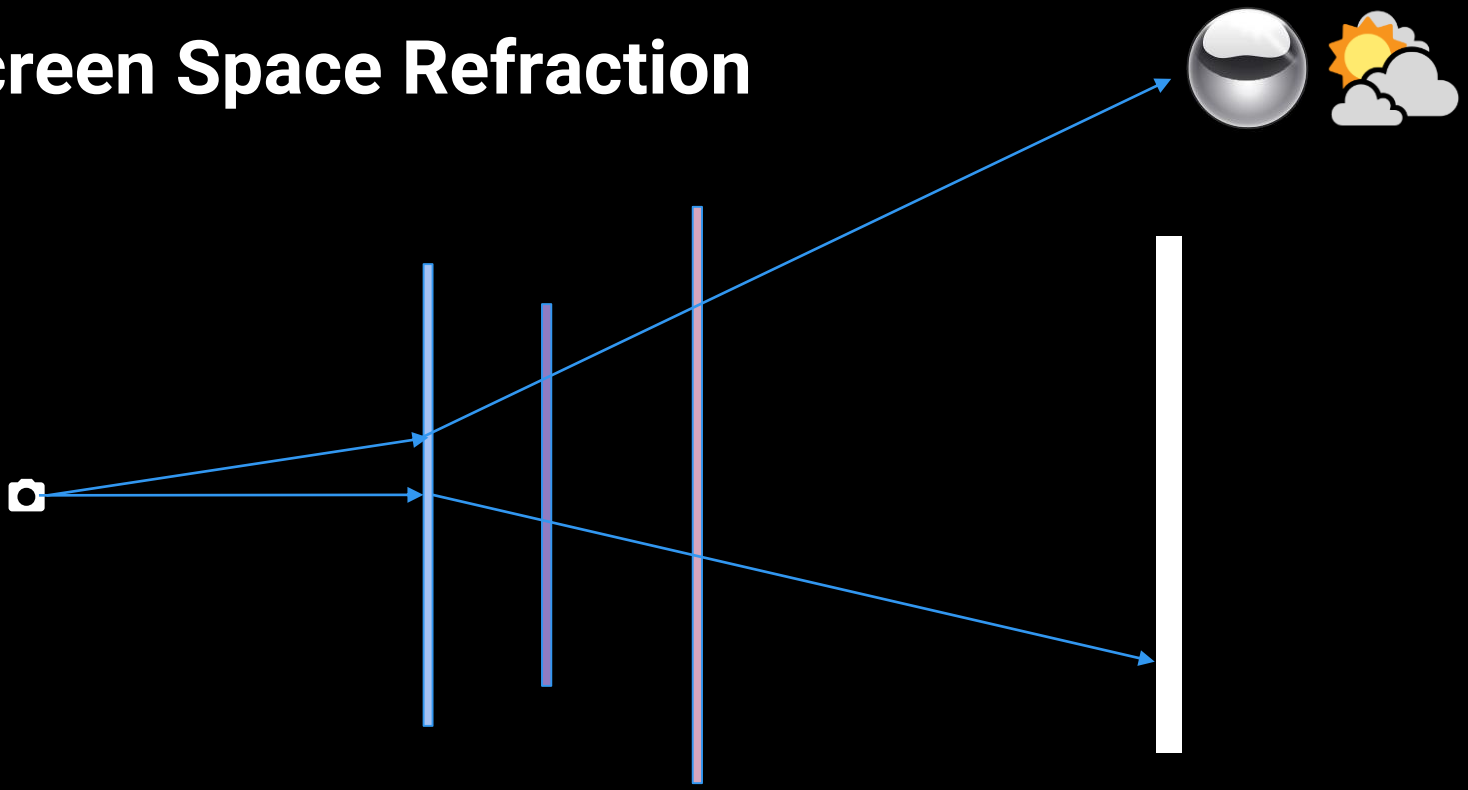
Screen Space Refraction



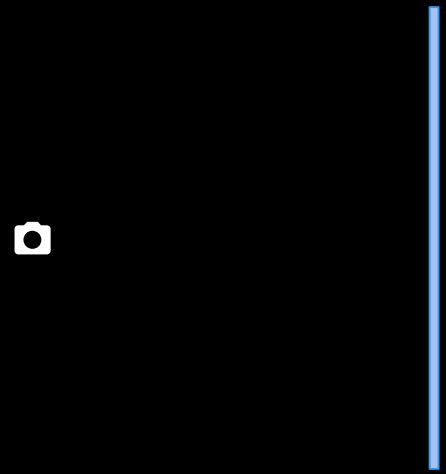
Screen Space Refraction



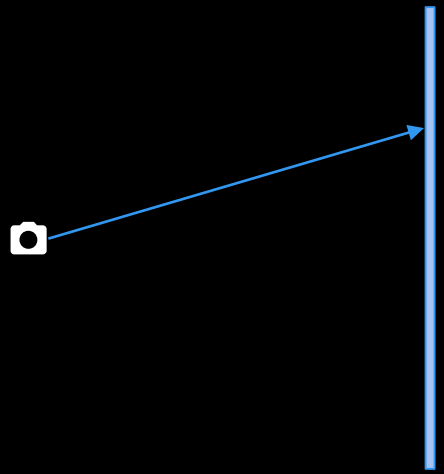
Screen Space Refraction



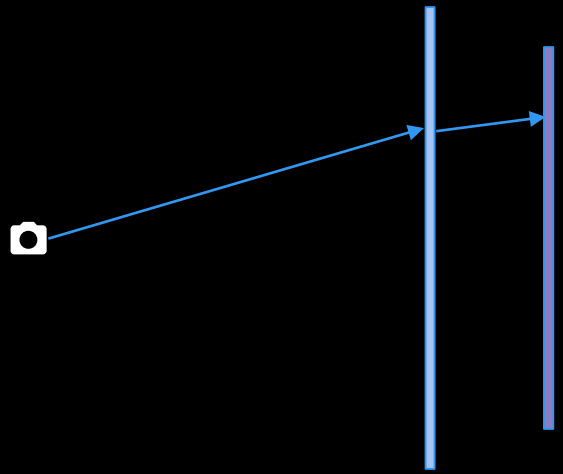
Ray Traced Transparency



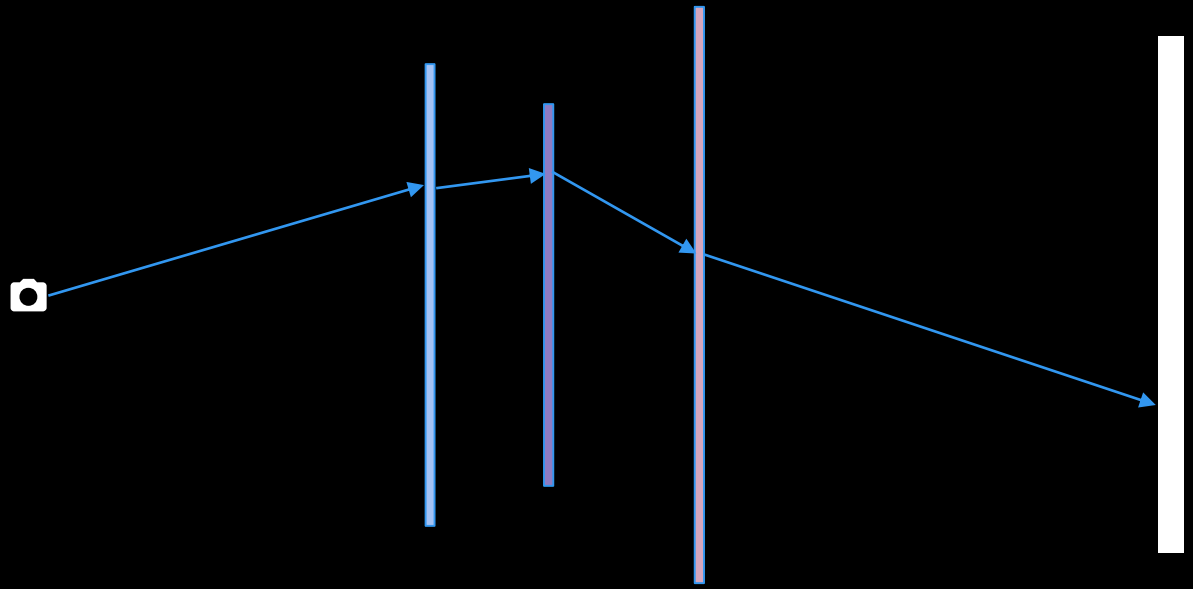
Ray Traced Transparency



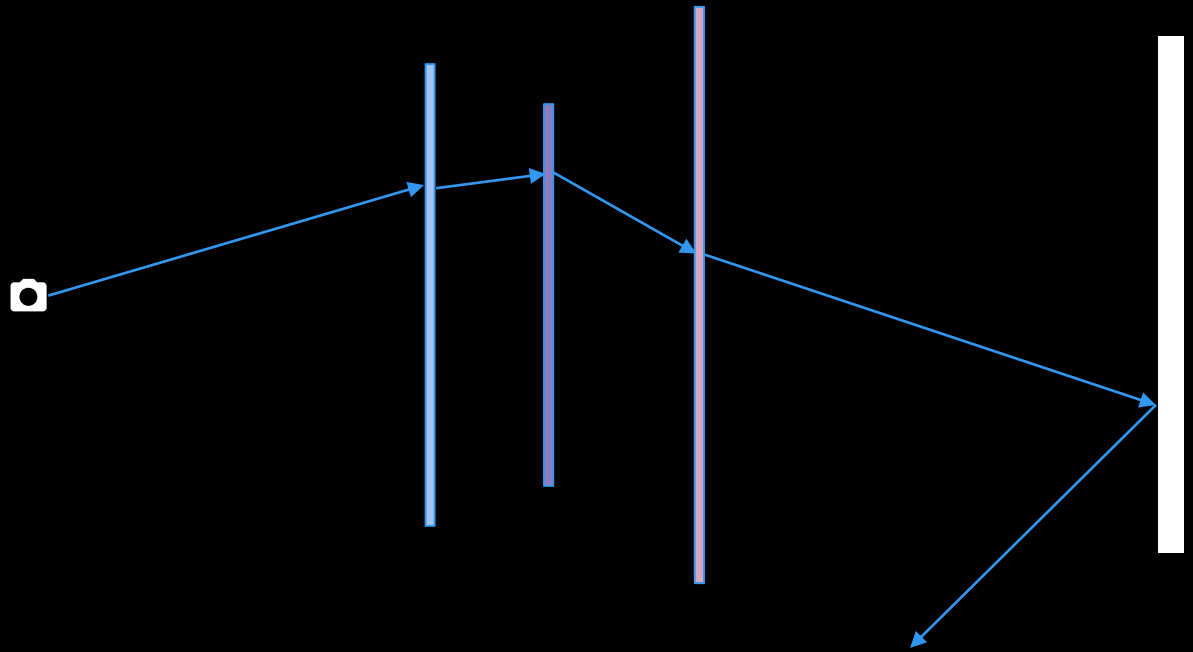
Ray Traced Transparency



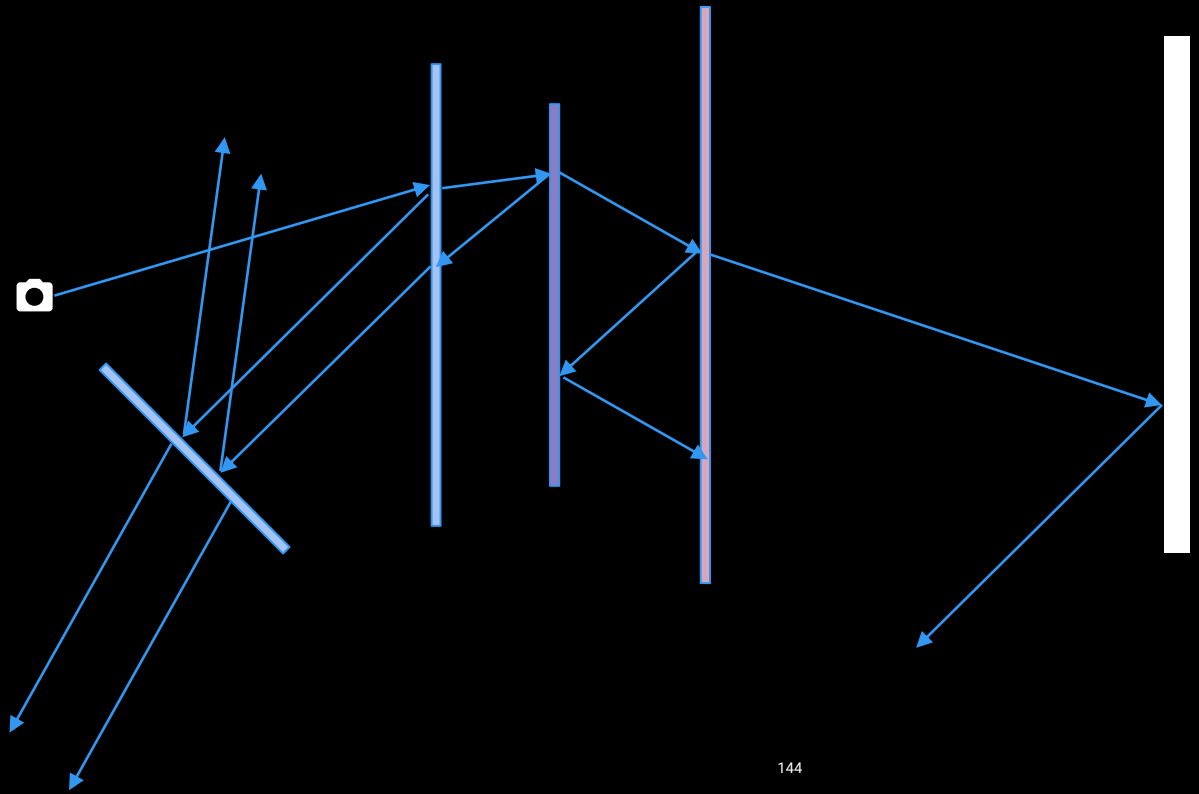
Ray Traced Transparency



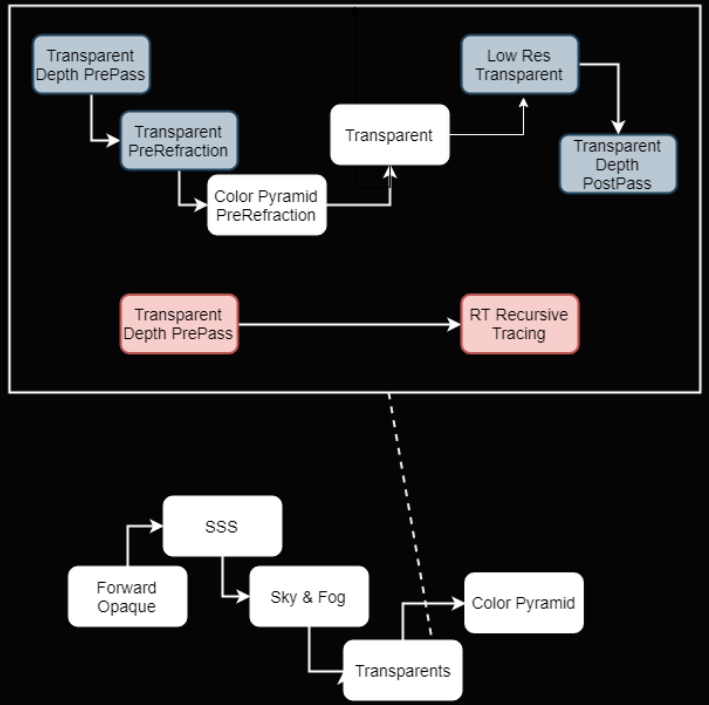
Ray Traced Transparency



Ray Traced Transparency



Hybrid Transparent Render Graph



Takeaways

- Naive is possible but not viable performance wise
- Reduce variance with analytic approximation/precomputation
- Forward materials are a real problem
- Sometimes rasterization is good enough

Takeaways

- Bin rays when distribution requires it
- Avoid `anyhit` as much as possible
- Take advantage of compute shaders when possible
- Some rasterization paradigms are incompatible with ray tracing

Takeaways

- Manage resources lifetime to avoid render target allocation
- Temporal reprojection/accumulation is key for denoising
- For real time, ray budget is extremely tight
- It might not make your image prettier, simply more accurate

Future Improvements

- Mix ray traced effects with their rasterized variant
- Lower resolution effects and up-sample all effects
- Use ray tracing for non rendering applications
- Path tracer for reference
- Skinned meshes and particle support

Acknowledgement

Unity Graphics Team
Sebastien Lagarde (Unity)
Kate McFadden (Unity)
Dany Ayoub (Unity)
Alexandre Cepisul (L&S)
Awen Couellan (L&S)
Aymeric du Chéné (L&S)
Mike Geig (Unity)
Emmanuel Turquin (Unity)
Laurent Harduin (Unity)
Ionut Nedelcu (Unity)
Arnaud Carré (Unity)
Joel de Vahl (Unity)
Tim Cooper (Unity)
Eric Heitz (Unity)
Francesco Cifariello Ciardi (Unity)
Antoine Lelievre (Unity)
Cyril Jover (Unity)
Lewis Jordan (Unity)
Natasha Tatarchuk (Unity)
Jesper Mortensen (Unity)
Tian Ning (Unity)
Melissa Chou (Unity)

Questions

https://github.com/Unity-Technologies/ScriptableRenderPipeline

Additional materials

Indirect Specular

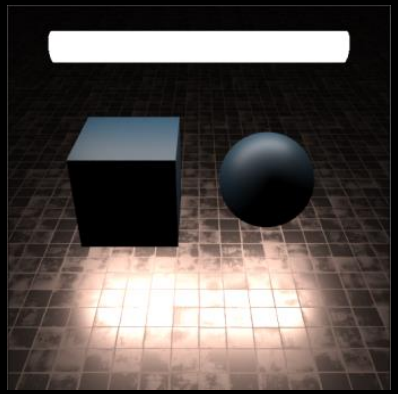
Small Room Scene - Roughness override 1.0 - 1SPP - Min Smoothness 0.0								
Test ID	Deferred	Binning	Tile Size (resolution)	Bin Resolution	Ray Binning Pass (ms)	Ray Trace (ms)	Deferred Shading (ms)	Effect Cost (ms)
Base	Off	Off	0	0	0	38	0	38
A0	On	Off	0	0	0	7.3	1.44	8.74
A1	On	On	16	16	0.141	5.8	1.437	7.378
A2	On	On	32	16	0.126	5.3	1.44	6.866

Small Room Scene - Roughness override 1.0 - 1SPP - Min Smoothness 0.0				
Test ID	Type	Ray Trace (ms)	Deferred Shading (ms)	Effect Cost (ms)
Base	Default	38	0	38
A0	Prepass + SemiDeferred	17	0	17
A1	Prepass + Deferred	7.5	1.4	8.9

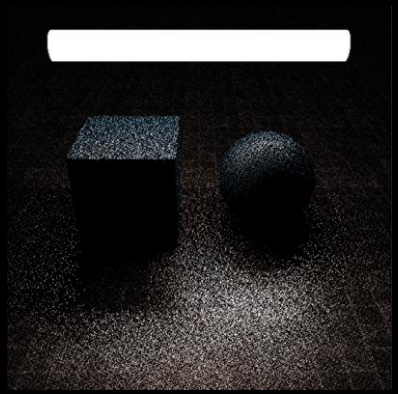
Indirect Specular

Small Room Scene - Variable Roughness - 1SPP - Min Smoothness 0.6								
Test ID	Deferred	Binning	Tile Size (resolution)	Bin Resolution	Ray Binning Pass (ms)	Ray Trace (ms)	Deferred Shading (ms)	Effect Cost (ms)
Base	Off	Off	0	0	0	8.6	0	8.6
A0	On	Off	0	0	0	1.961	0.633	2.594
A1	On	On	16	16	0.132	1.8	0.634	2.566
A2	On	On	32	16	0.1085	1.69	0.637	2.4355

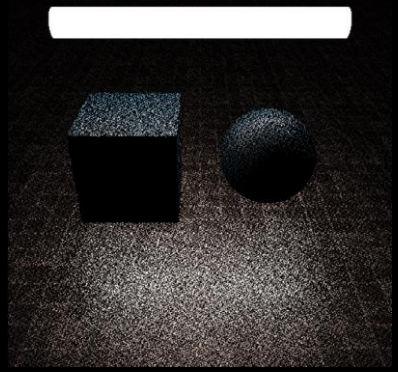
Area Shadow



LTC

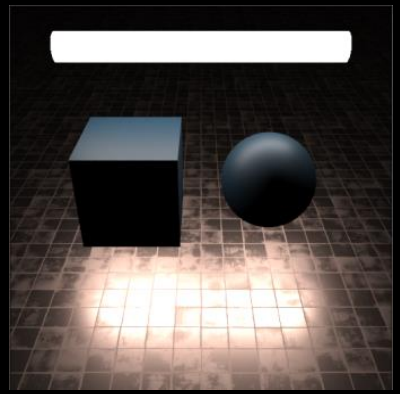


Sn

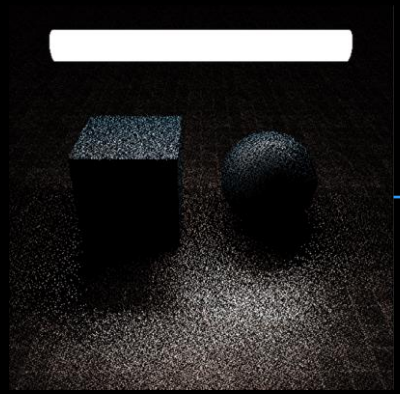


Un

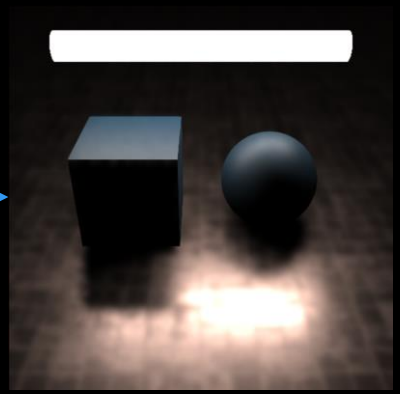
Area Shadow



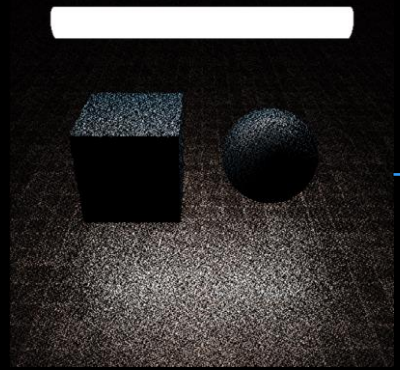
LTC



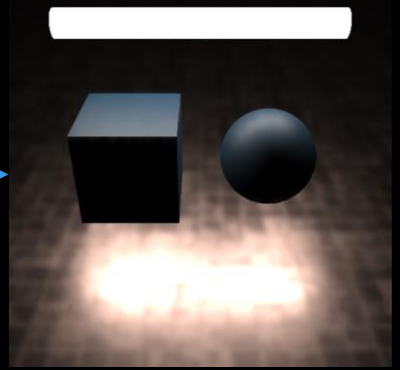
Sn



denoise(Sn)

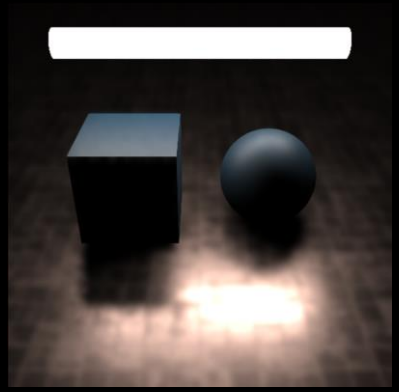


Un

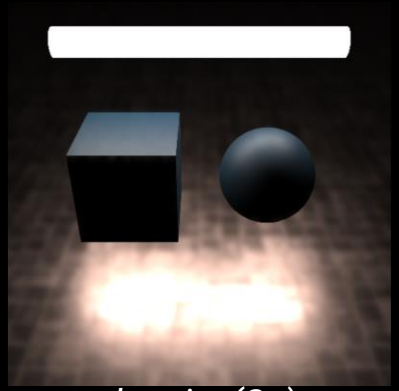


denoise(Un)

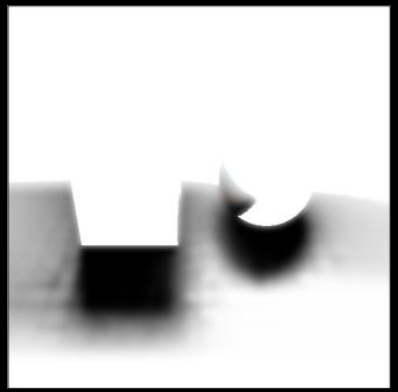
Area Shadow



denoise(Un)

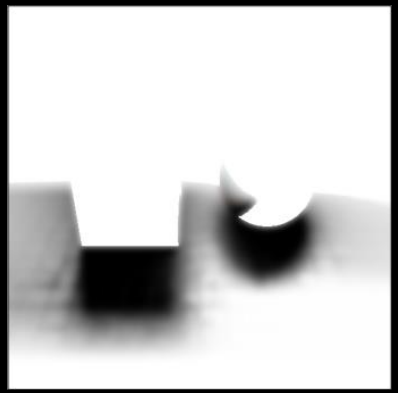


denoise(Sn)

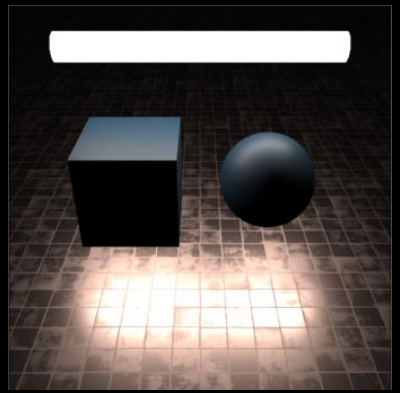


denoise(Sn) / denoise(Un)

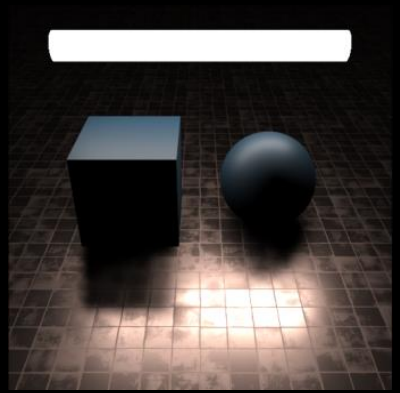
Area Shadow



denoise(Sn) / denoise(Un)



LTC



LTC + Stochastic Shadow