20220816 Project Meeting

Juhyeon Kim

2022.08.16.

PEARL ABYSS

Contents

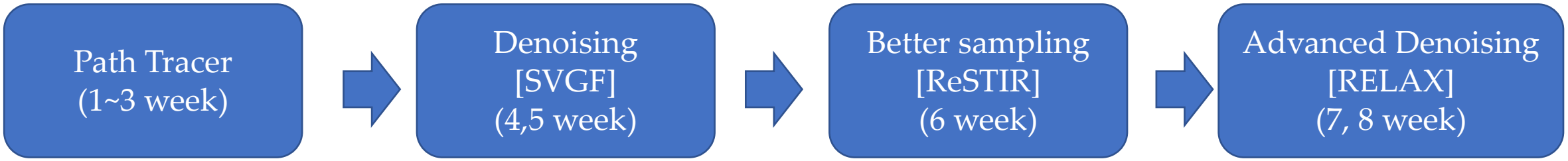
PEARL ABYSS

- Project Overview
- Path Tracer Component Overview
- Render Pass Overview
  - Render Pass Decomposition
  - RELAX (next version of SVGF)
- Result / Analysis

Project Overview

PEARL ABYSS

- Project Goal : Implement a **1-spp real-time** path tracer with denoising & better sampling technique.



- Implement a **real-time path tracer using DX12**.
- Study basics of DX12 and physically based rendering.

- Implement **denoising** technique for a path-traced image.
- Choose to implement SVGF (2017).

- Implement **sampling quality enhancement** technique.
- Choose to implement ReSTIR (2020)

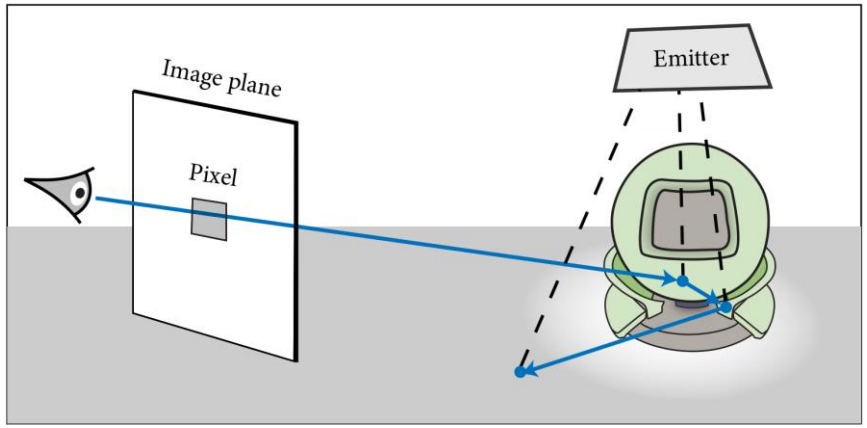
- Implement advanced denoising technique that can handle various materials.
- Choose to implement RELAX (2021)

Path Tracer Component Overview

Path Tracer Component Overview

PEARL ABYSS

- **Mitsuba** styled path tracer
  - Shape - rectangle, cube, mesh, sphere, …
  - BSDF - to be explained in detail
  - Emitter - area, envmap
  - Sensor - perspective only
  - Texture - bitmap image only



Plugin reference

- Plugin reference
- Shapes
- BSDFs
- Phase functions
- Emitters
- Sensors
- Textures
- Spectra
- Integrators
- Samplers
- Films
- Reconstruction filters

Program structure example of Mitsuba2 renderer

Scene Format

PEARL ABYSS

```
<integrator type="path" >
    <integer name="maxDepth" value="65" />
    <boolean name="strictNormals" value="true" />
</integrator>
<sensor type="perspective" >
    <float name="fov" value="19.5" />
    <transform name="toWorld" >
        <matrix value="-1 0 0 0 0 1 0 1 0 0 -1 6.8 0 0 0 1"/>
    </transform>
```

Sensor

```
<bsdf type="twosided" id="TallBox" >
    <bsdf type="diffuse" >
        <rgb name="reflectance" value="0.725, 0.71, 0.68"/>
    </bsdf>
</bsdf>
```

BSDF

```
<shape type="rectangle" >
    <transform name="toWorld" >
        <matrix value="-4.37114e-008 1 4.37114e-008 0 0 -8.74228e-008 2 0 1 4.37114e-008 1.91069e-015 0 0 0 0 1"/>
    </transform>
    <ref id="Floor" />
</shape>
```

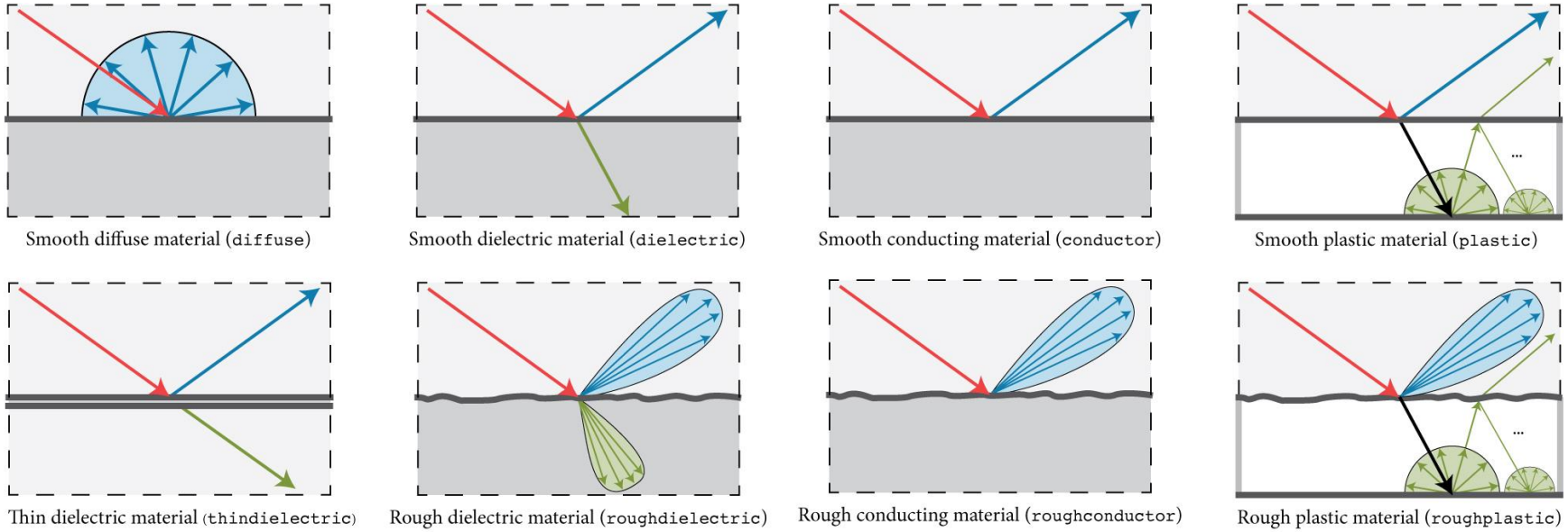
Shape

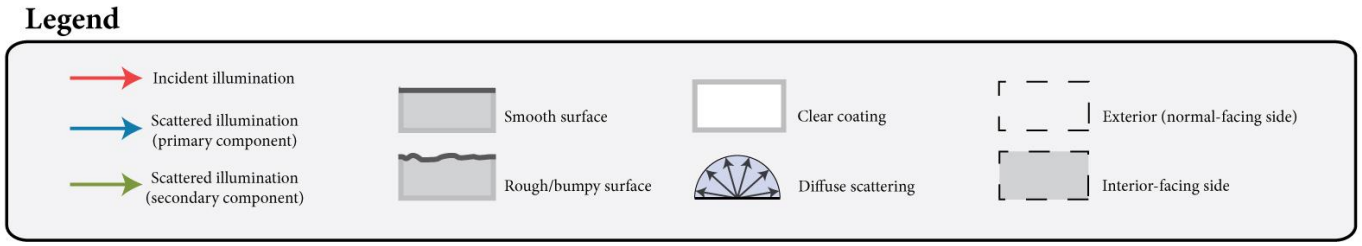
```
<shape type="rectangle" >
    <transform name="toWorld" >
        <matrix value="0.235 -1.66103e-008 -7.80685e-009 -0.005 -2.05444e-008 3.90343e-009 -0.0893 1.98 2.05444e-008 0.19 8.30516e-009 -0.03 0 0 0 1"/>
    </transform>
    <ref id="Light" />
    <emitter type="area" >
        <rgb name="radiance" value="17, 12, 4"/>
    </emitter>
</shape>
```

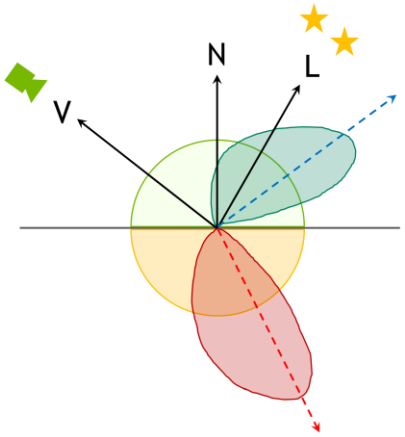
Shape + Emitter

BSDF Overview

PEARL ABYSS



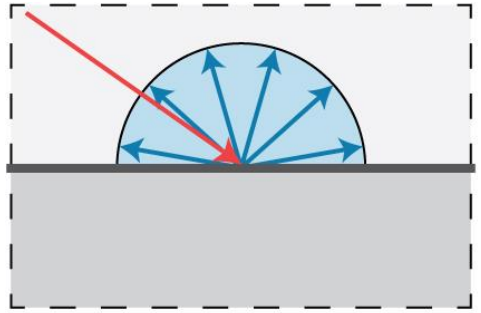




Diffuse/glossy/delta reflection

Diffuse/glossy/delta transmission

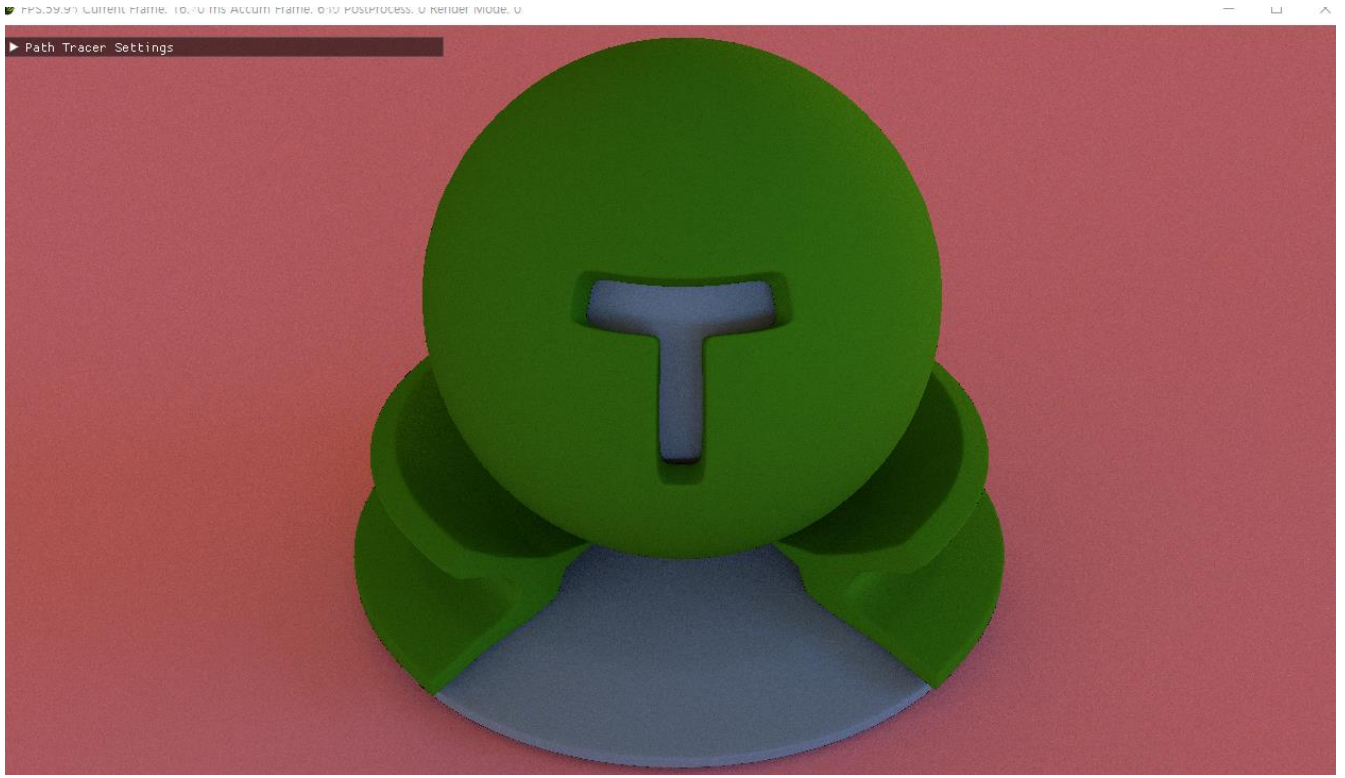
BSDF - Diffuse



Smooth diffuse material (`diffuse`)

Diffuse/glossy/delta reflection

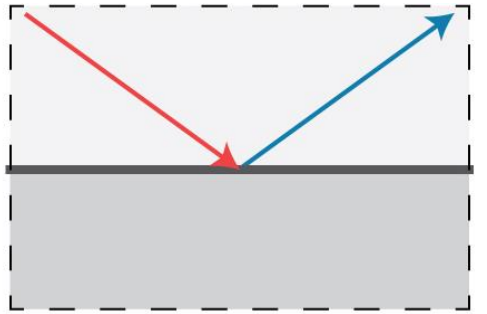
Diffuse/glossy/delta transmission



Diffuse

BSDF - Conductor

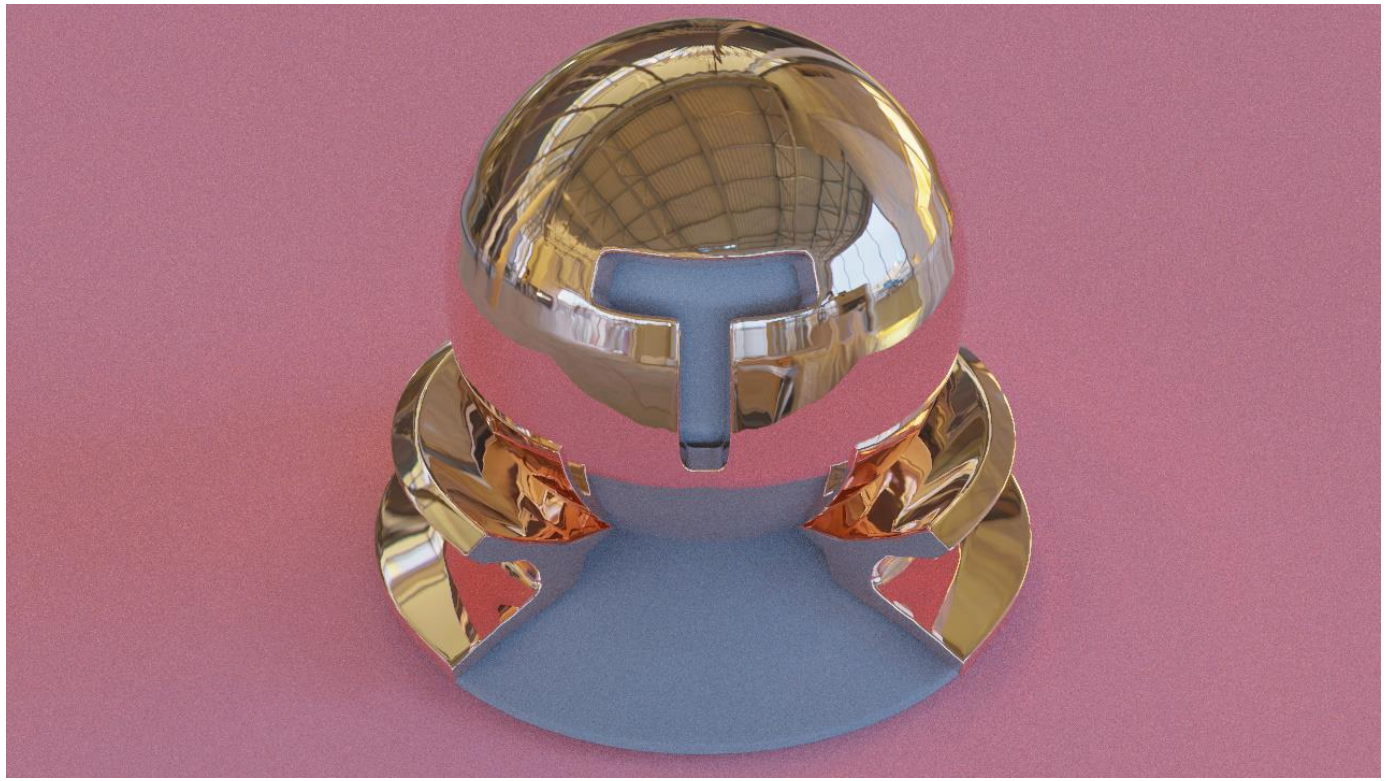
PEARL ABYSS



Smooth conducting material (conductor)

Diffuse/glossy/**delta** reflection

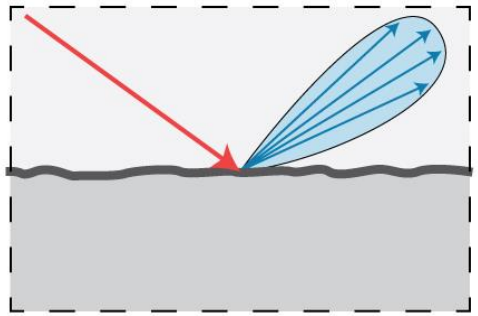
Diffuse/glossy/delta transmission



Conductor

BSDF - Rough Conductor

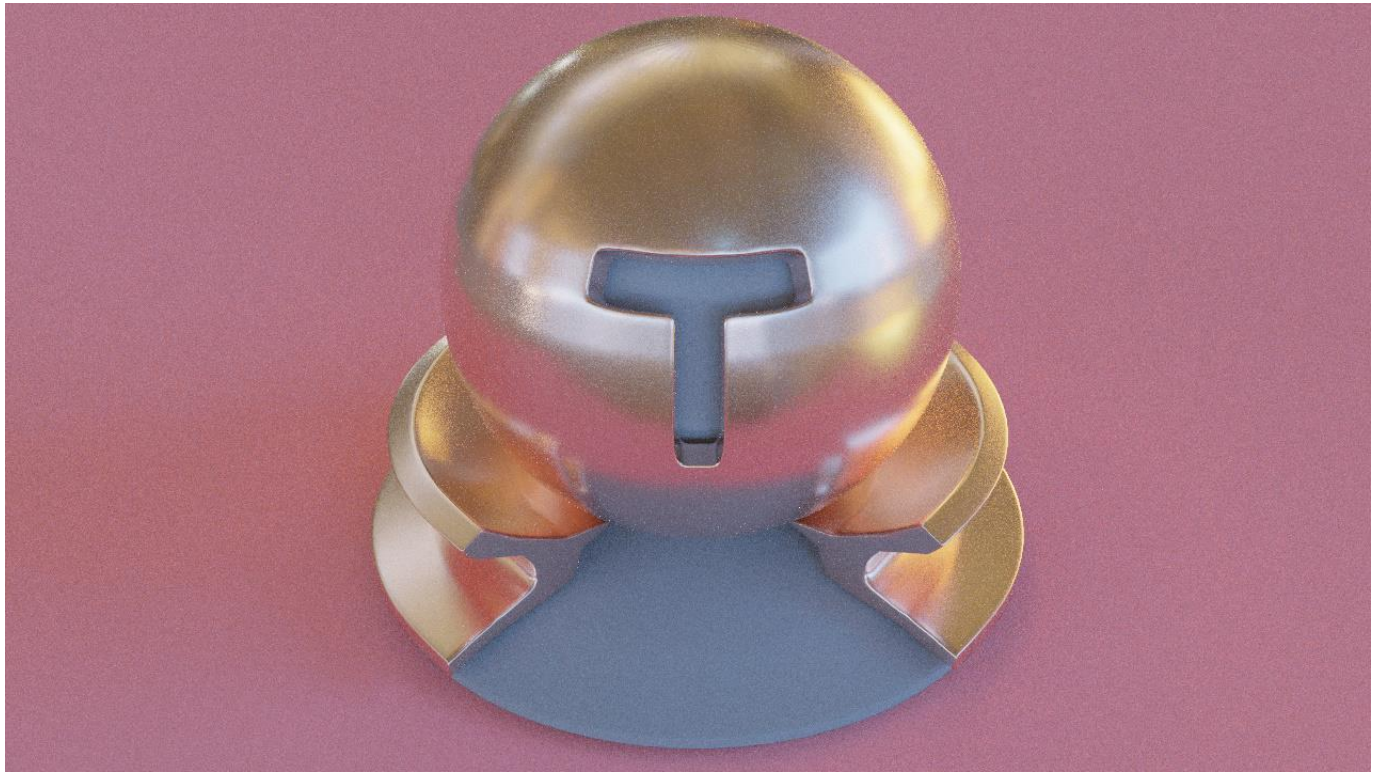
PEARL ABYSS



Rough conducting material (roughconductor)

Diffuse/glossy/delta reflection

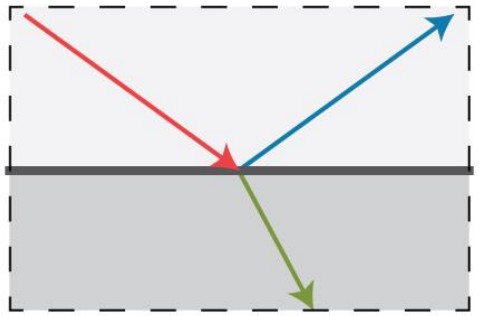
Diffuse/glossy/delta transmission



Rough Conductor

BSDF - Dielectric

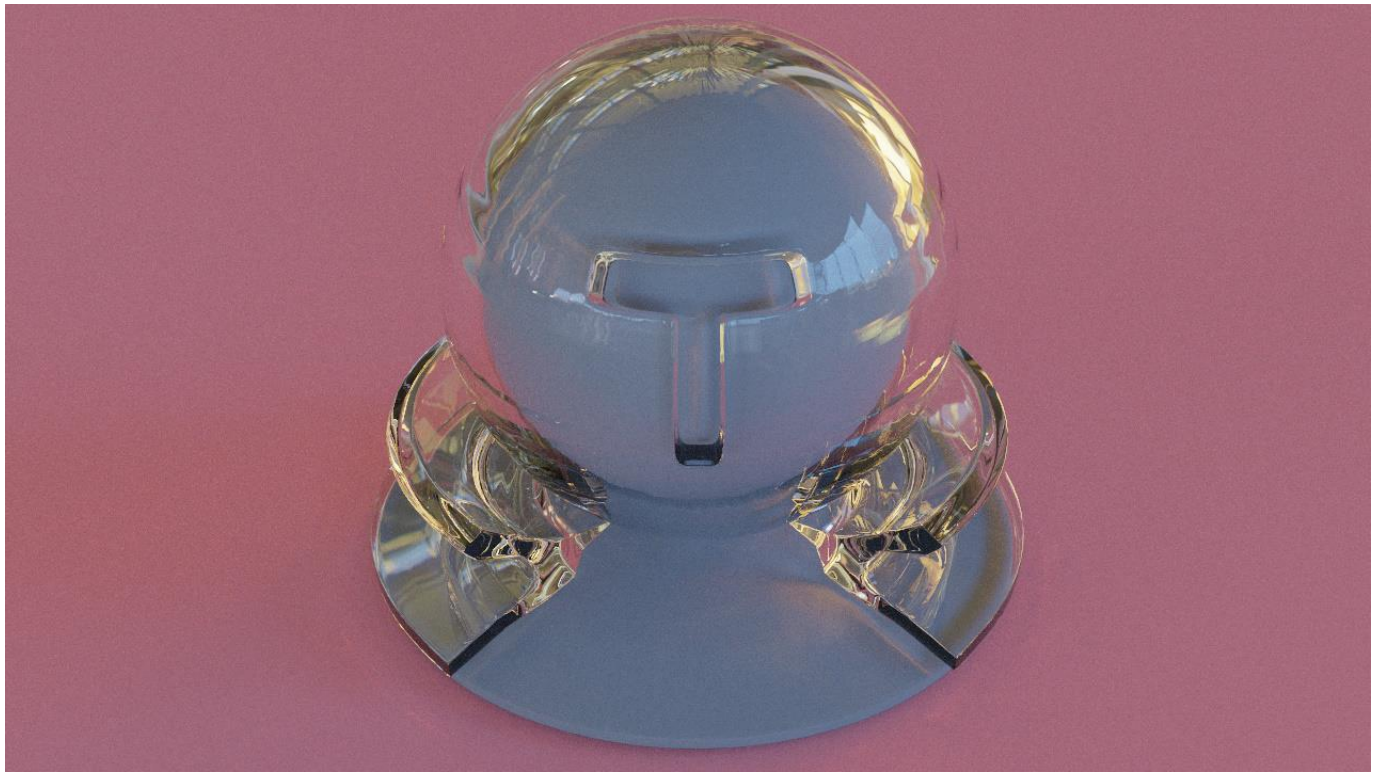
PEARL ABYSS



Smooth dielectric material (`dielectric`)

Diffuse/glossy/**delta** reflection

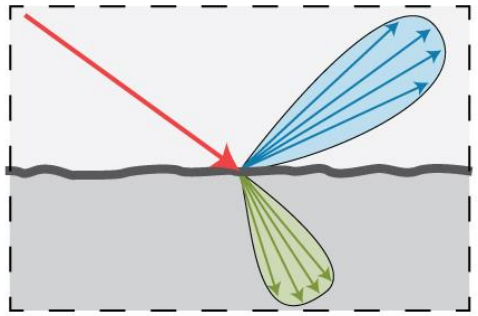
Diffuse/glossy/**delta** transmission



Dielectric

BSDF - Rough Dielectric

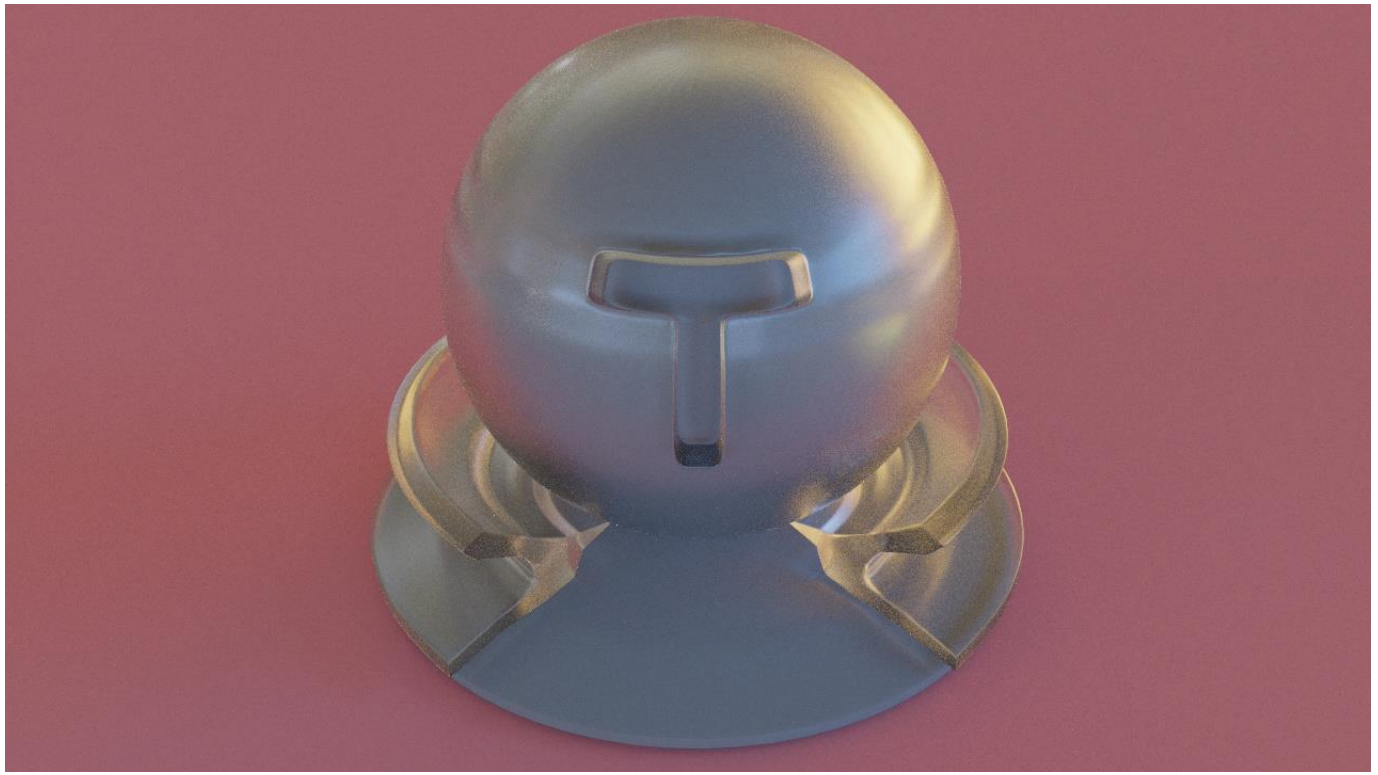
PEARL ABYSS



Rough dielectric material (`roughdielectric`)

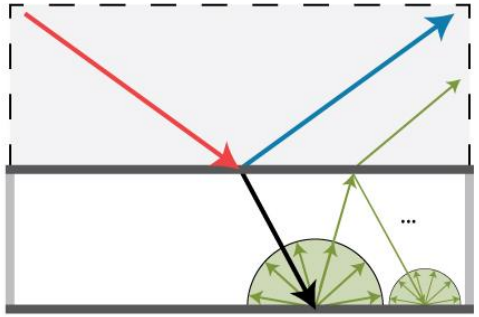
Diffuse/**glossy**/delta reflection

Diffuse/**glossy**/delta transmission



Rough Dielectric

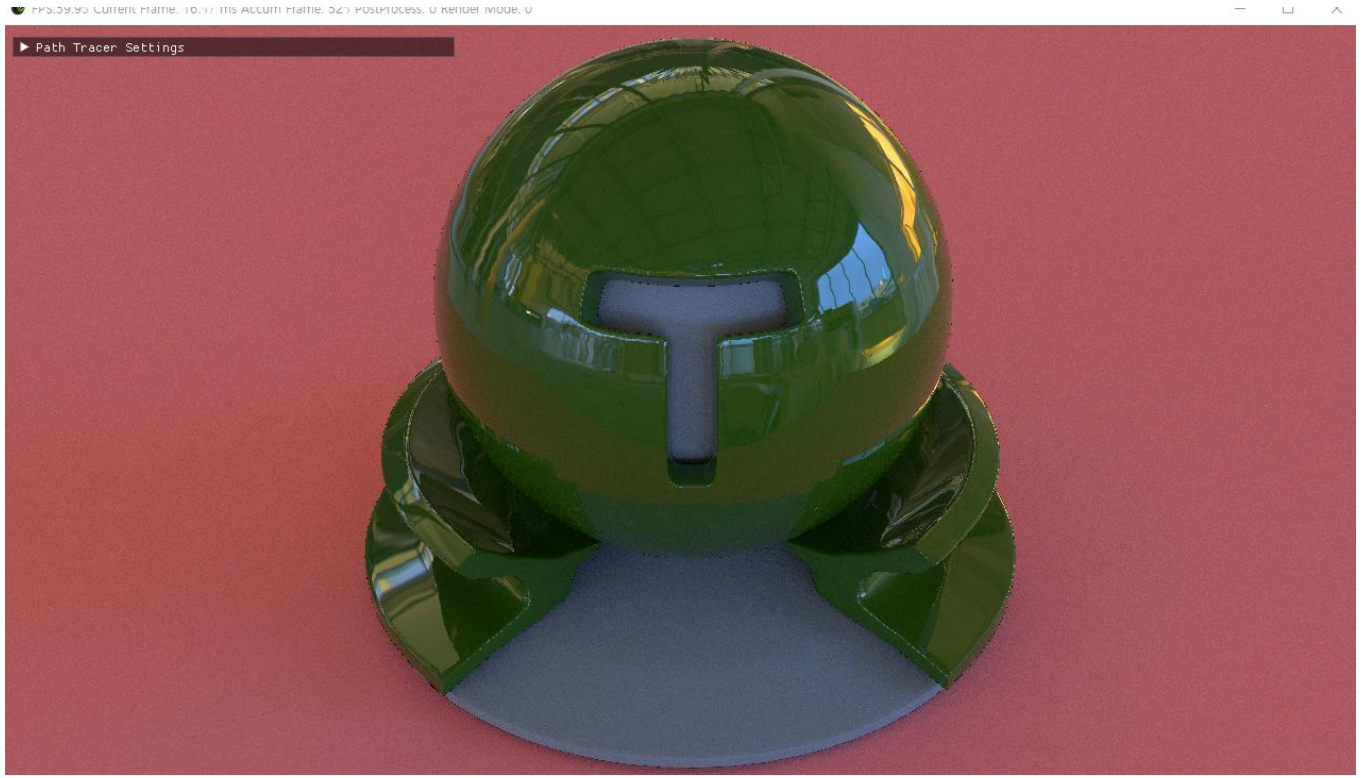
BSDF - Plastic



Smooth plastic material (`plastic`)

Diffuse/glossy/**delta** reflection

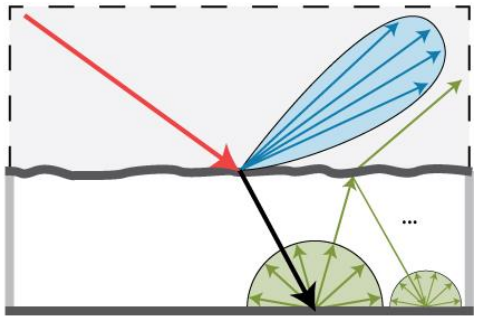
Diffuse/glossy/delta transmission



Plastic

BSDF - Rough Plastic

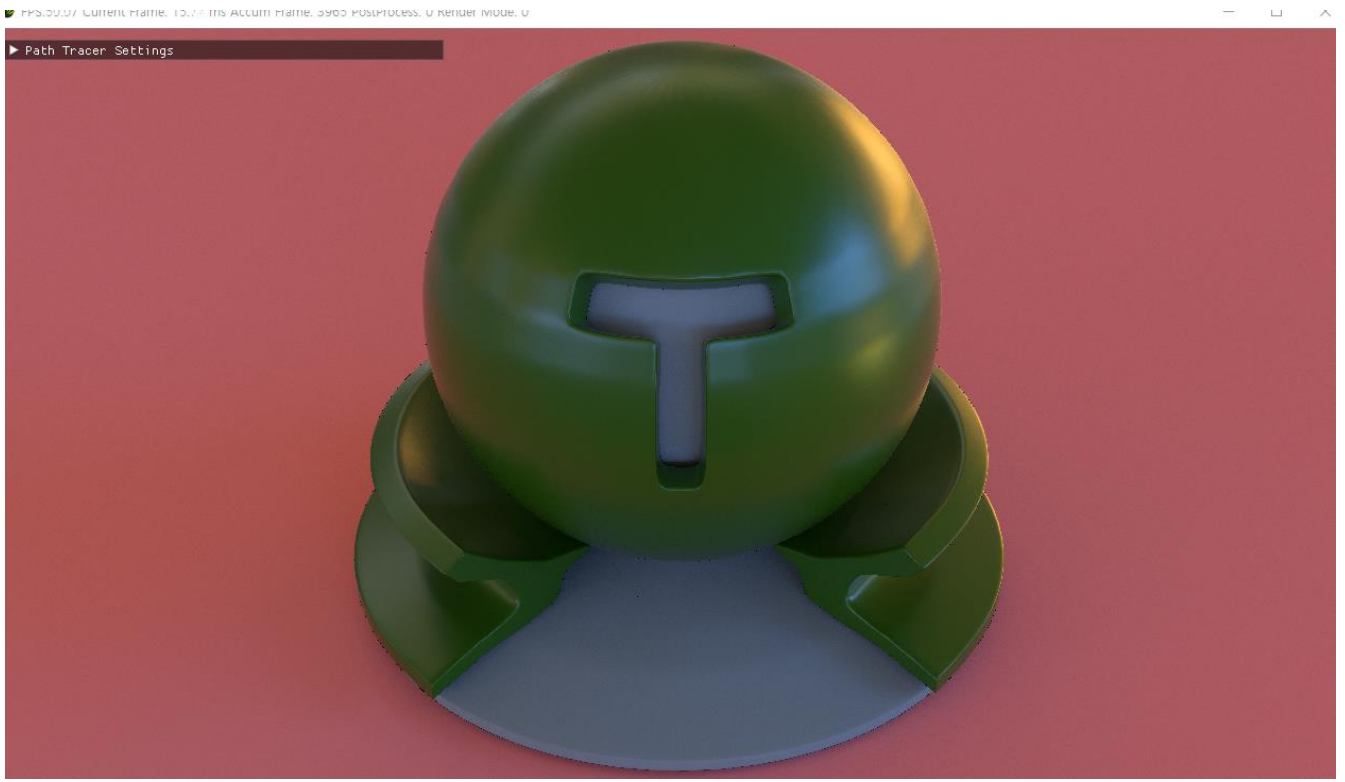
PEARL ABYSS



Rough plastic material (roughplastic)

Diffuse/**glossy**/delta reflection

Diffuse/glossy/delta transmission



Rough Plastic

Render Pass Overview

Path Tracer

PEARL ABYSS

- Standard path tracer with NEE / MIS (direct light sampling) + BSDF Importance sampling

Problem → far from satisfying quality if 1spp is used

Render Pass Overview

PEARL ABYSS

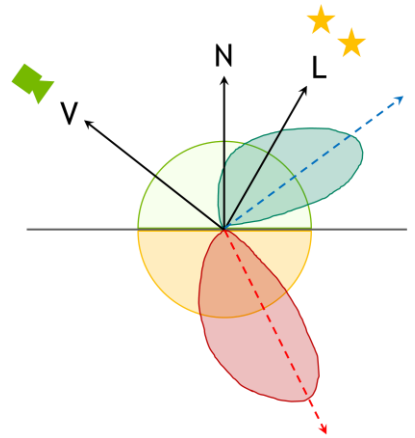
Output =

gDiffuseReflectance * gDiffuseIllumination + gSpecularReflectance * gSpecularIllumination + Emission

+ gDeltaReflectionReflectance * gDeltaReflectionIllumination + gDeltaReflectionEmission

+ gDeltaTransmissionReflectance * gDeltaTransmissionIllumination + gDeltaTransmissionEmission

+ gResidualRadiance



Noise-free Input

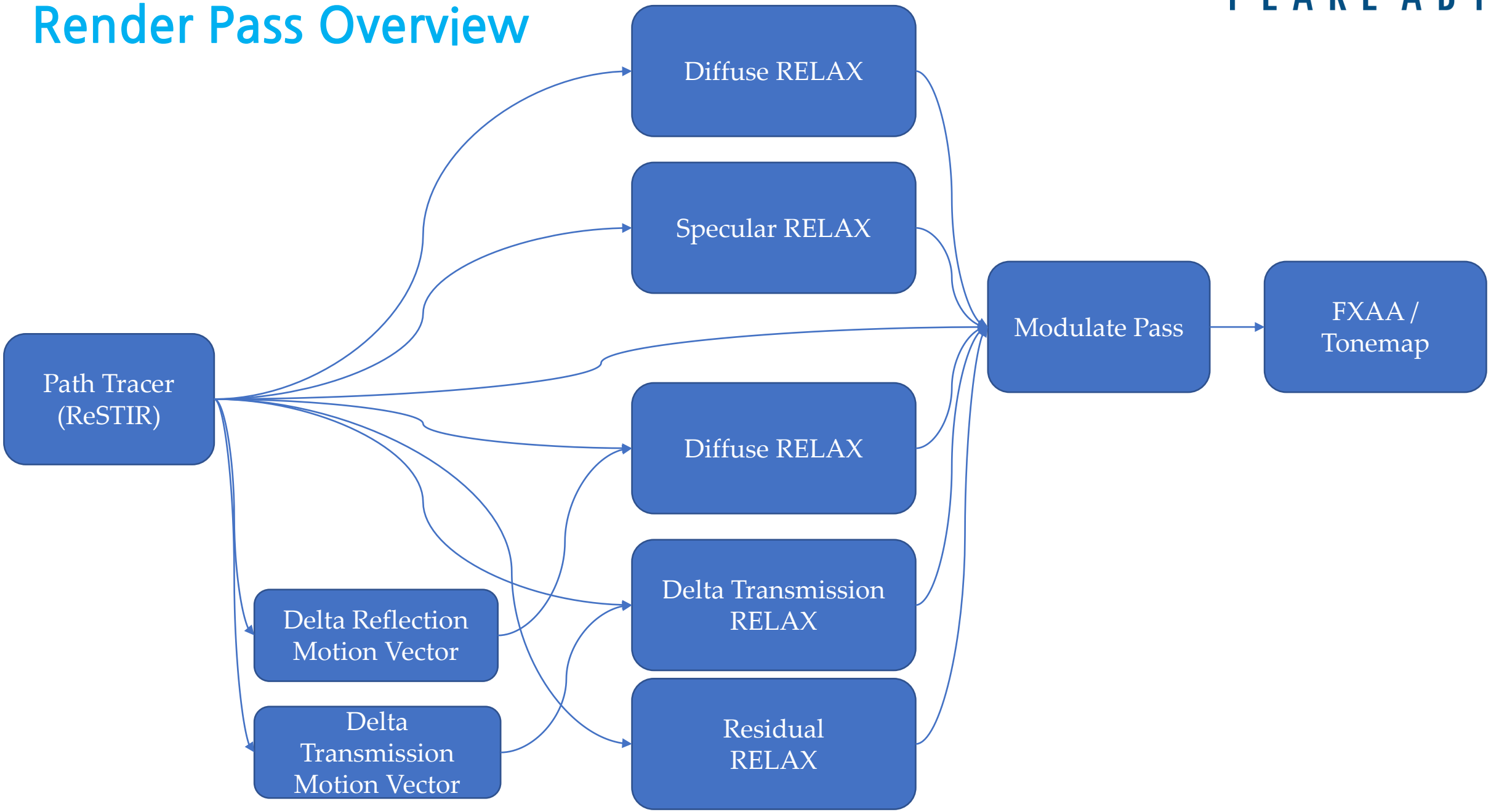
Noisy Input (to be filtered)

Diffuse/glossy/delta reflection

Diffuse/glossy/delta transmission

Render Pass Overview

PEARL ABYSS



PEARL ABYSS

Diffuse Reflectance

Diffuse Illumination

Diffuse Radiance

PEARL ABYSS

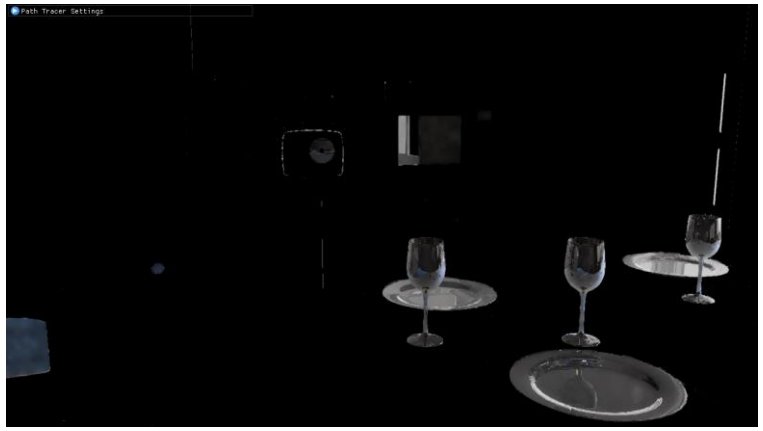
Specular Reflectance

Specular Illumination

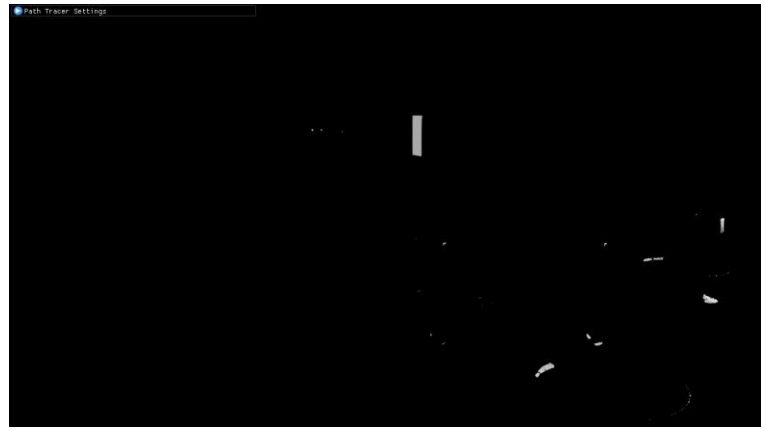
Specular Radiance

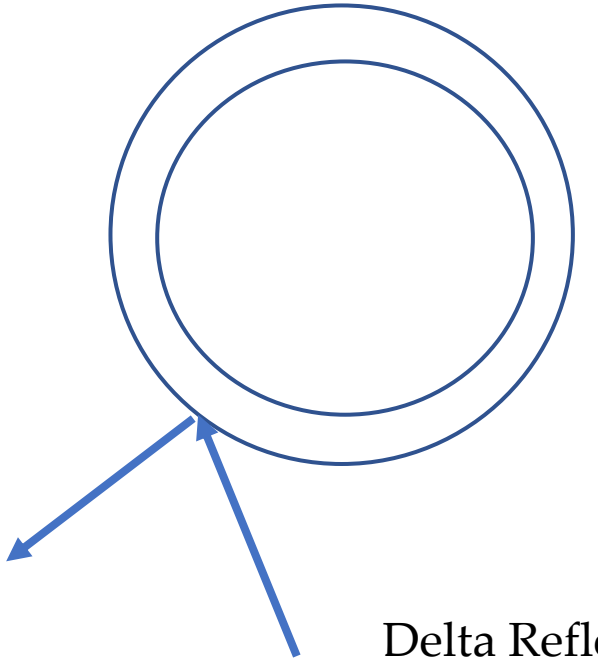
Delta Reflection Reflectance

Delta Reflection Illumination



Delta Reflection Emission

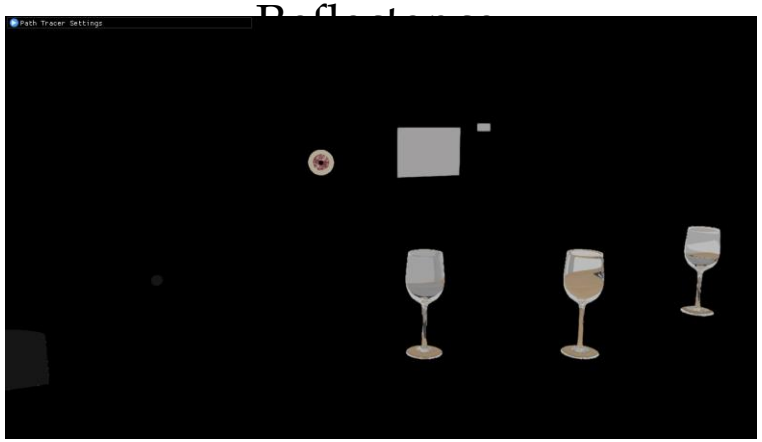




Delta Reflection Only

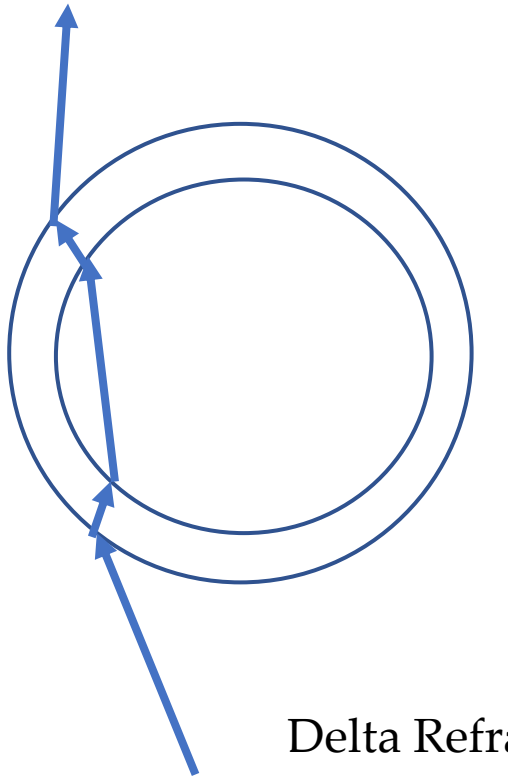
Delta Reflection Radiance

Delta Transmission Reflection



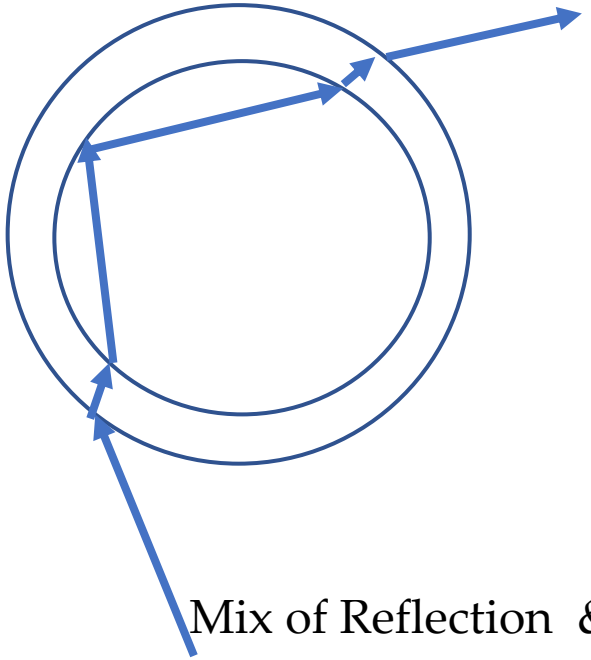
Delta Transmission Illumination

Delta Transmission Emission



Delta Refraction Only

Delta Transmission Radiance



Mix of Reflection & Refraction

Residual Radiance


Path Tracer Settings

Path Tracer Overview

```
PrimaryPath(ray, pathResult, payloadPrimary);
Material material = g_materialInfo[payloadPrimary.materialIndex];
uint materialReflectionLobe = bsdf::getReflectionLobe(material);

if (materialReflectionLobe & (BSDF_LOBE_DELTA_REFLECTION)) {
    RayPayload payload = payloadPrimary;
    PathTraceDeltaReflectance(ray, seed, pathResult, payload);
}
if (materialReflectionLobe & BSDF_LOBE_DELTA_TRANSMISSION) {
    RayPayload payload = payloadPrimary;
    PathTraceDeltaTransmission(ray, seed, pathResult, payload);
}

PathTrace(ray, seed, pathResult, payloadPrimary, true);
```

PEARL ABYSS

If delta-reflection Lobe

If delta-transmission Lobe

Primary Path (G-Buffer) → **Standard Path Tracing** → **Delta Reflection Only** → **Delta Transmission Only**

- Diffuse Reflectance
- Specular Reflectance
- Primary hit position
- Primary hit normal
- …

- Radiance

- Delta Reflection Reflectance
- Delta Reflection Pos/Norm

- Delta Transmission Reflectance
- Delta Transmission Pos/Norm

- Delta Reflection Reflectance : first hit non-delta material

RELAX - Temporal Accumulation

PEARL ABYSS

- Find Temporal Consistency based on previous frame pixel
  - Position
  - Normal
  - Mesh ID

- Calculate previous frame pixel (motion vector) is easy !!

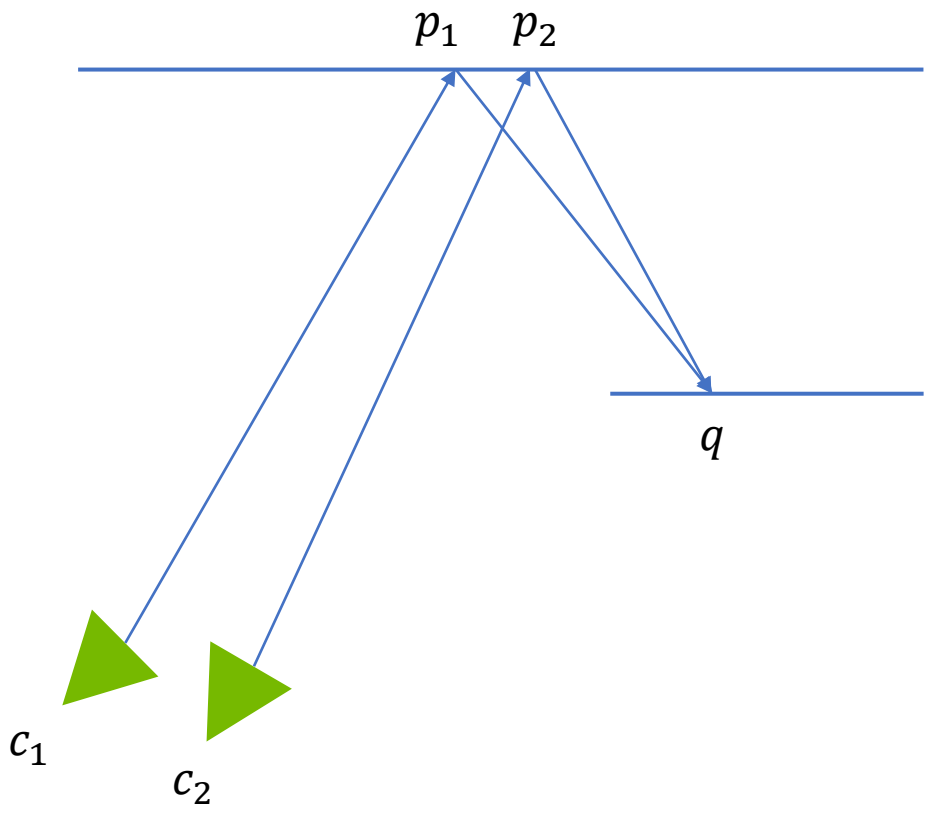
```
// Reprojection
    float4 projCoord = mul(float4(position, 1.0f), g_frameData.previousProjView);
    projCoord /= projCoord.w;
    float2 prevPixel = float2(projCoord.x, -projCoord.y);
    prevPixel = (prevPixel + 1) * 0.5;
```

But for reflection / transmission?

Motion Vector - Delta Reflection

PEARL ABYSS

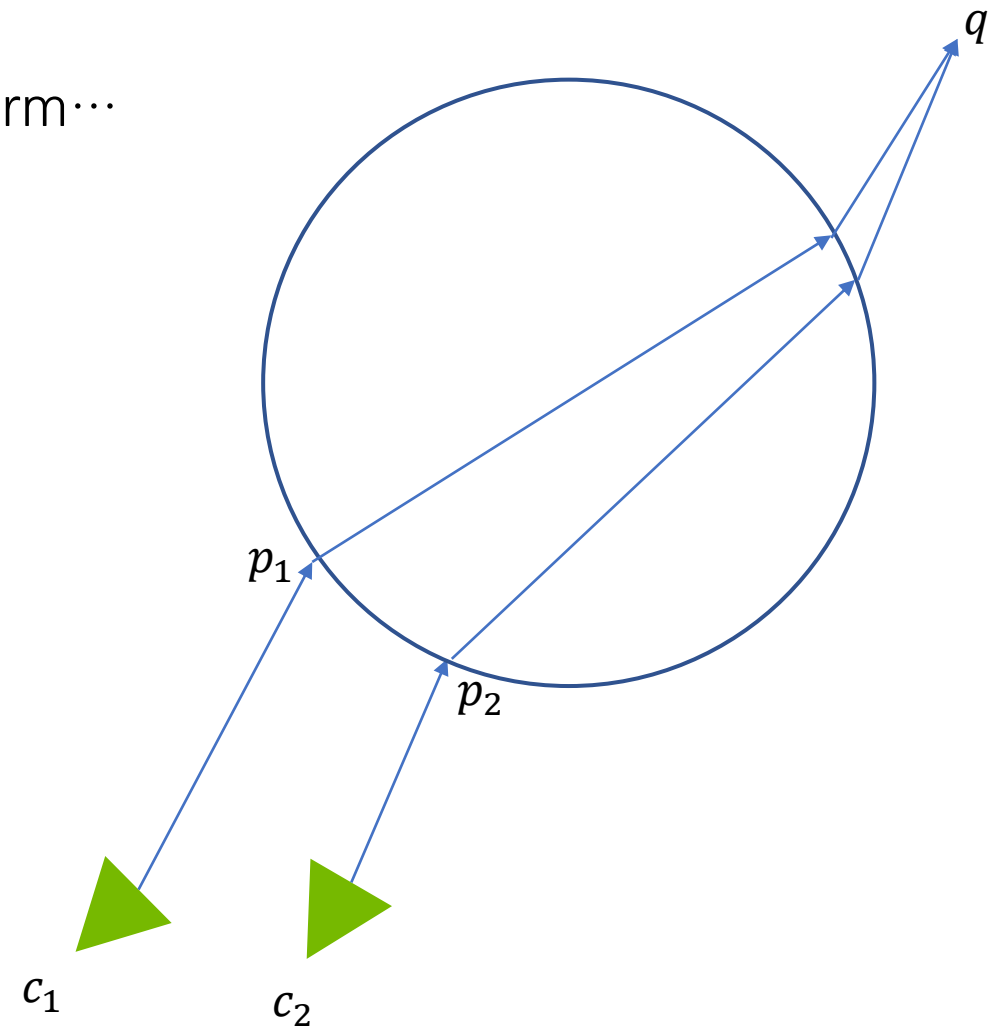
- Can calculate in closed form!

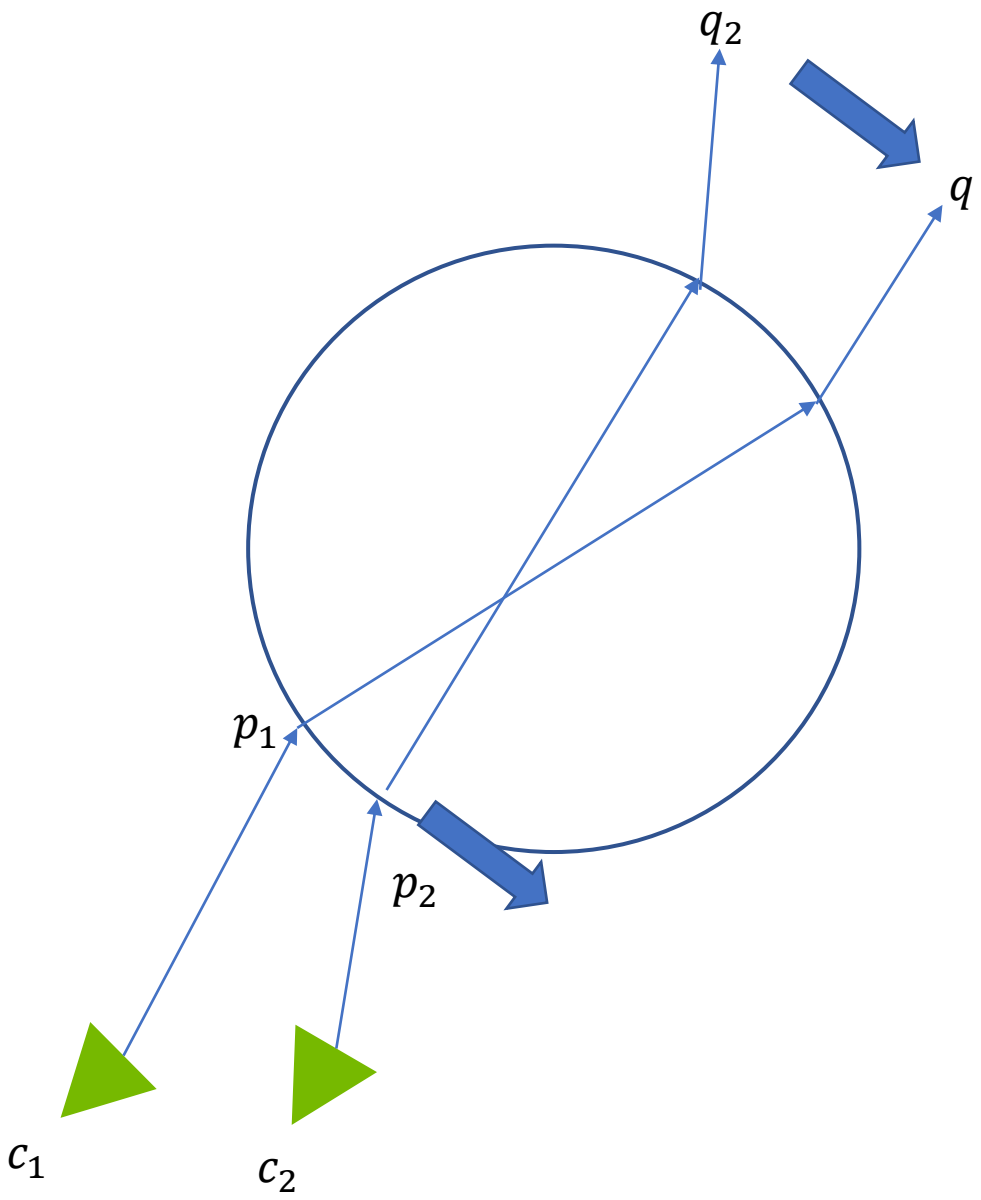


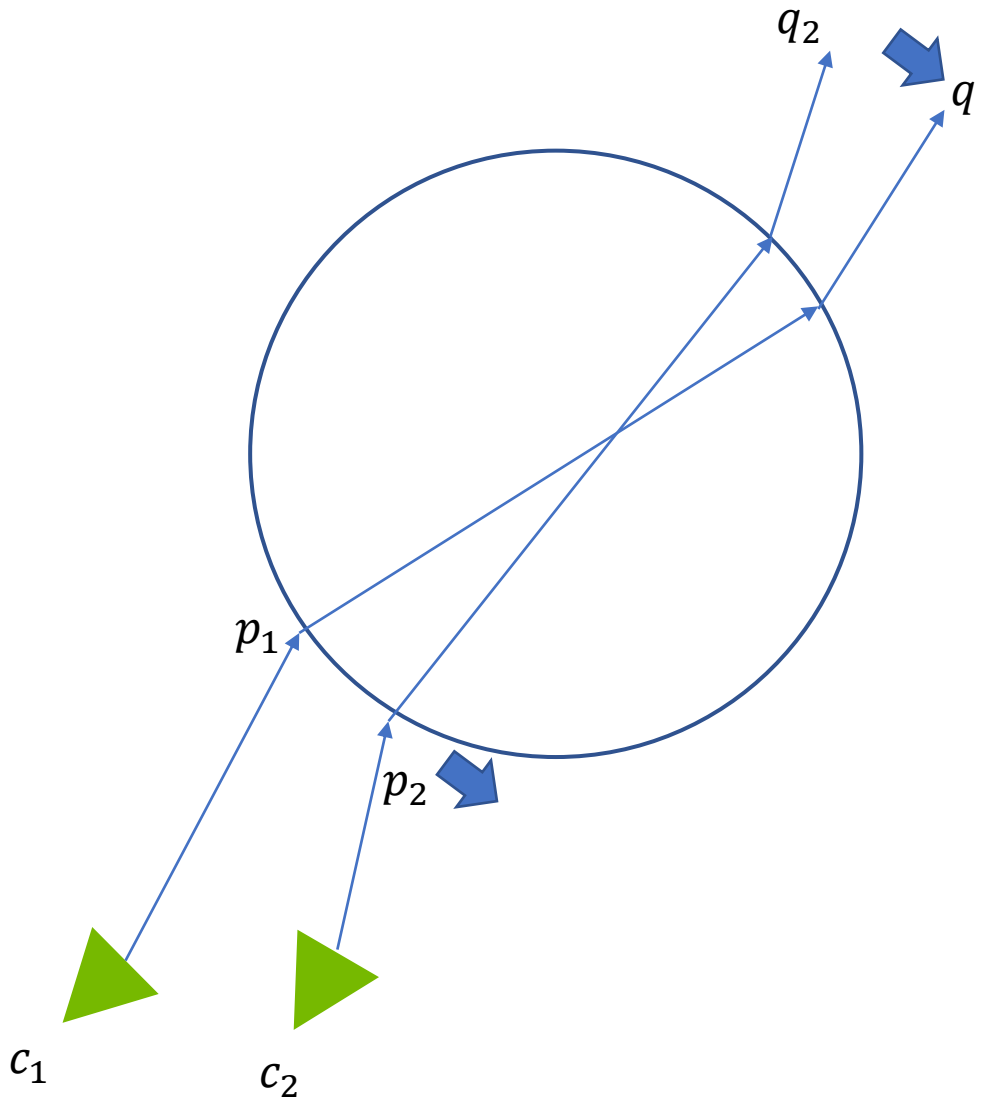
Motion Vector - Delta Transmission

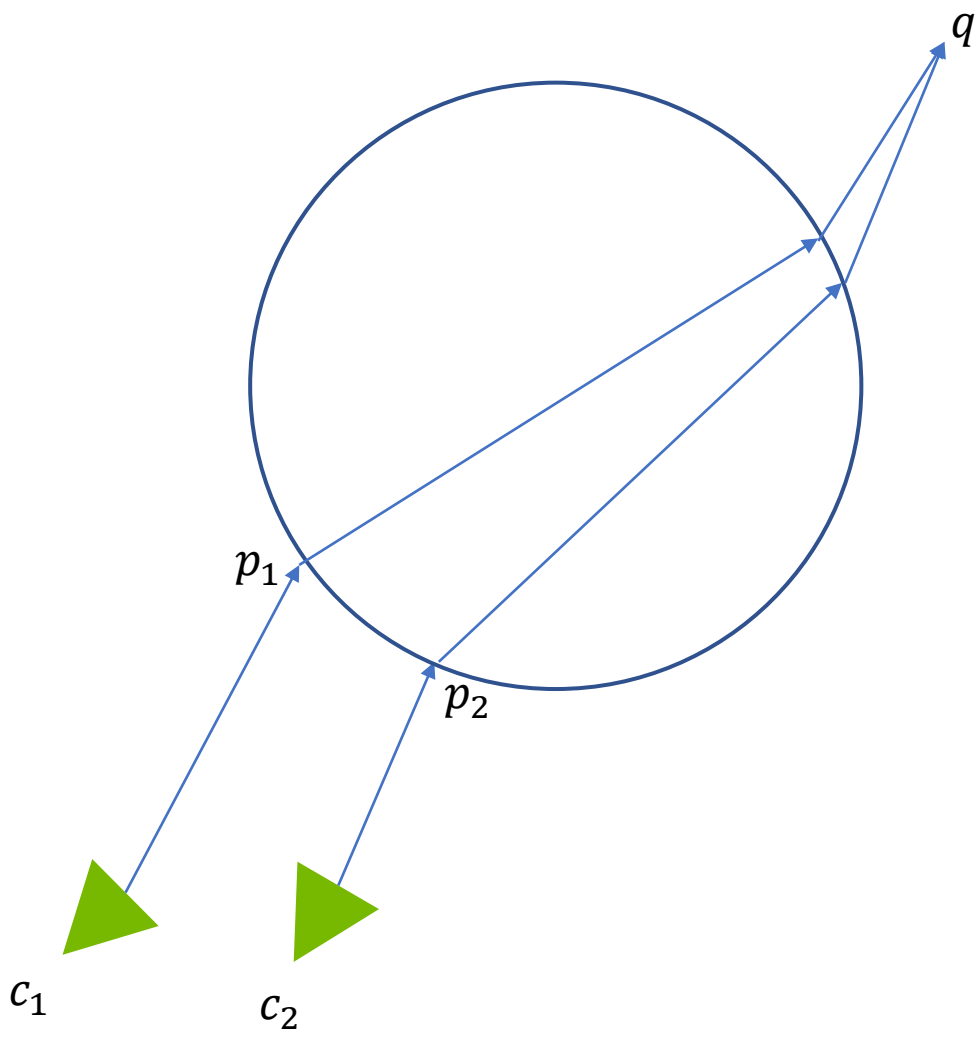
PEARL ABYSS

- Cannot solve in closed form…




q_2
q
p_1
p_2
c_1
c_2

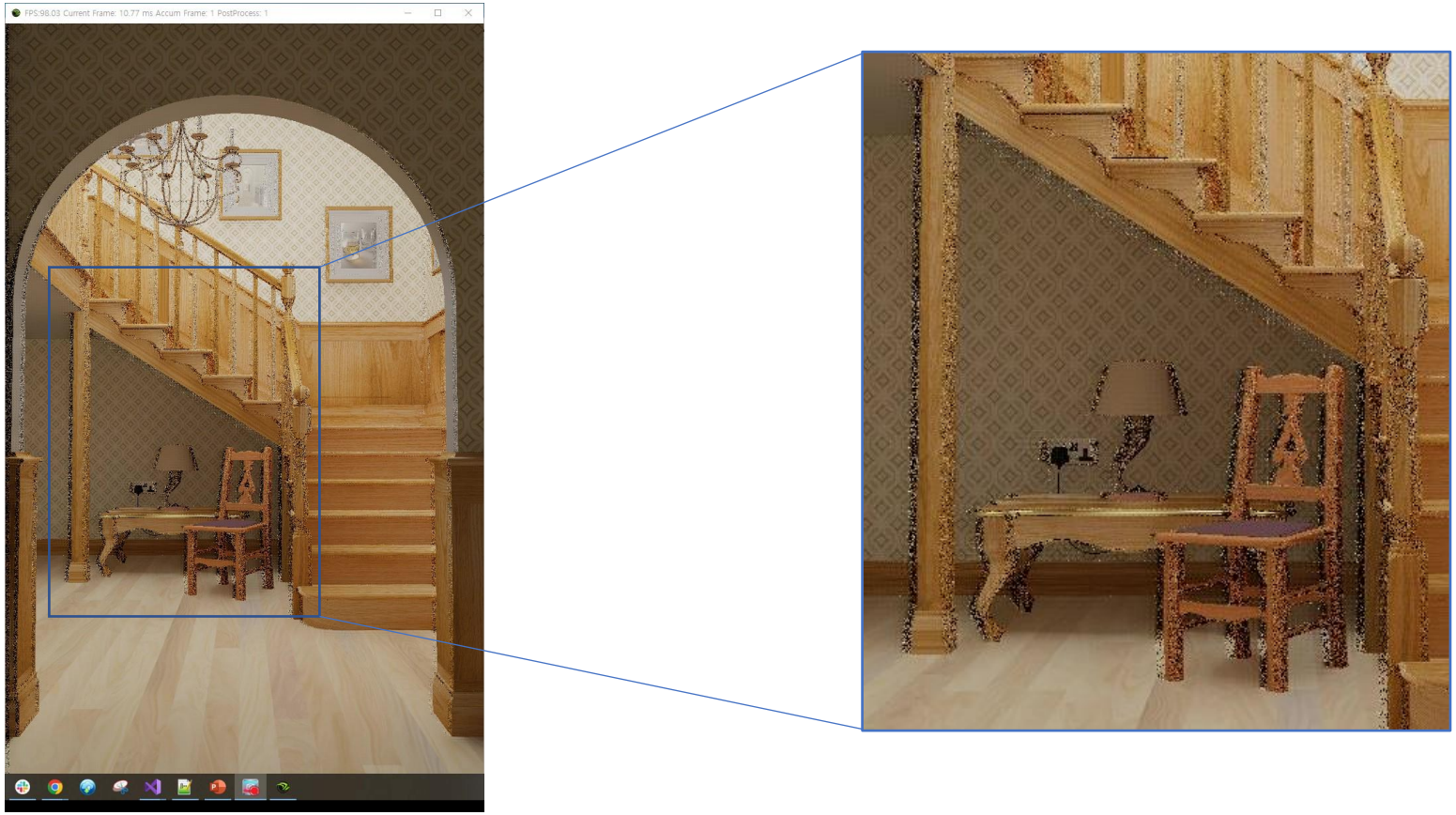

q_2
q
p_1
p_2
c_1
c_2


q
p_1
p_2
c_1
c_2

RELAX - Disocclusion Fix

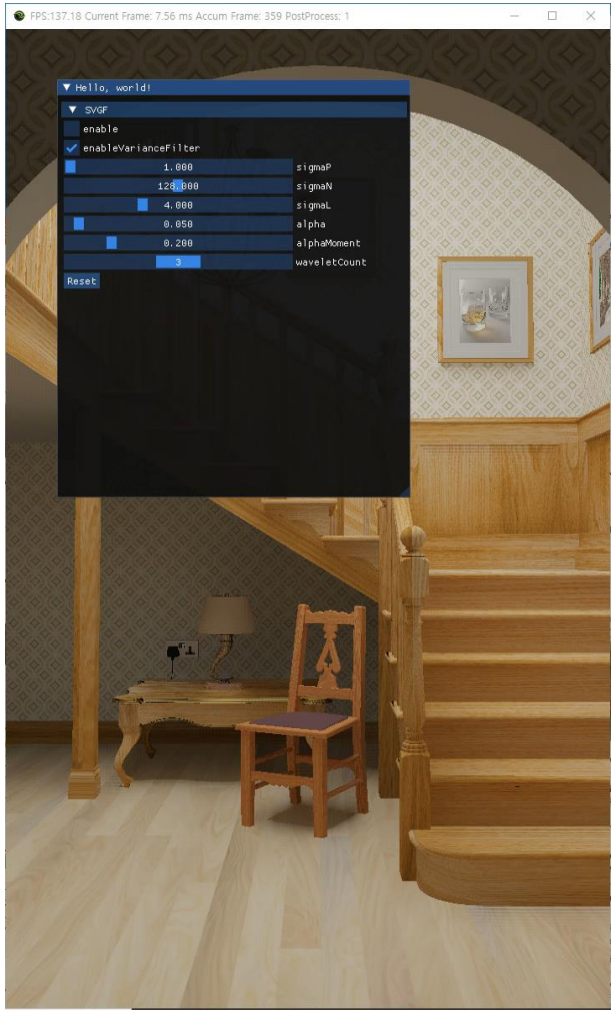
PEARL ABYSS

- Just after disocclusion, temporal variance doesn't work.
- Solution → estimate variance spatially if history length < 4.



RELAX - Disocclusion Fix

PEARL ABYSS



RELAX - A-trous Wavelet Filter

PEARL ABYSS

- Spatial filter is required. → A-trous wavelet filter (3~5 times)

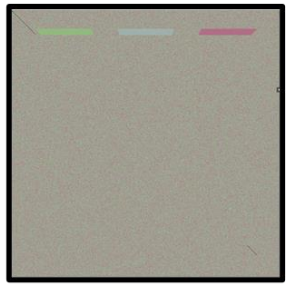
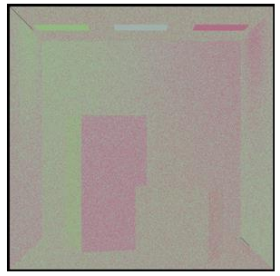
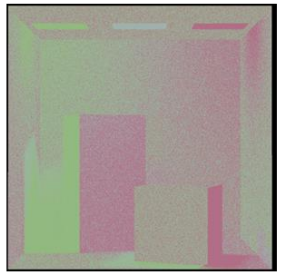
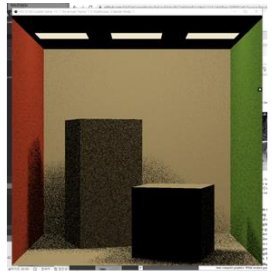
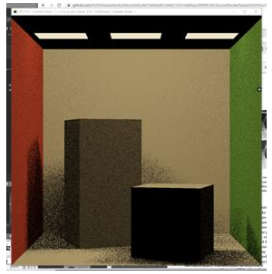
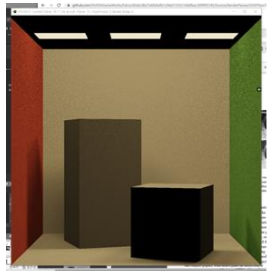
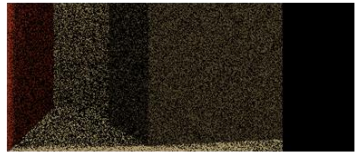
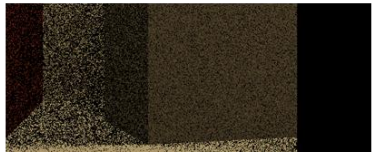
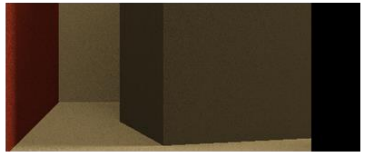
- Position difference
- Normal difference
- Luminance difference

- (specular only) Specular lobe difference

Without spatial filtering

ReSTIR

- Implemented ReSTIR (temporal reuse only) → but not that useful…

No RIS (uniform)	RIS w/o temporal reuse	RIS w/ temporal reuse
$x_i \sim u$	$x_i \sim \rho L_e G$	$x_i \sim \rho L_e G V$
		
		
		

u : uniform pdf
ρ : BRDF
L_e : emission
G : Geometric term
V : Visibility

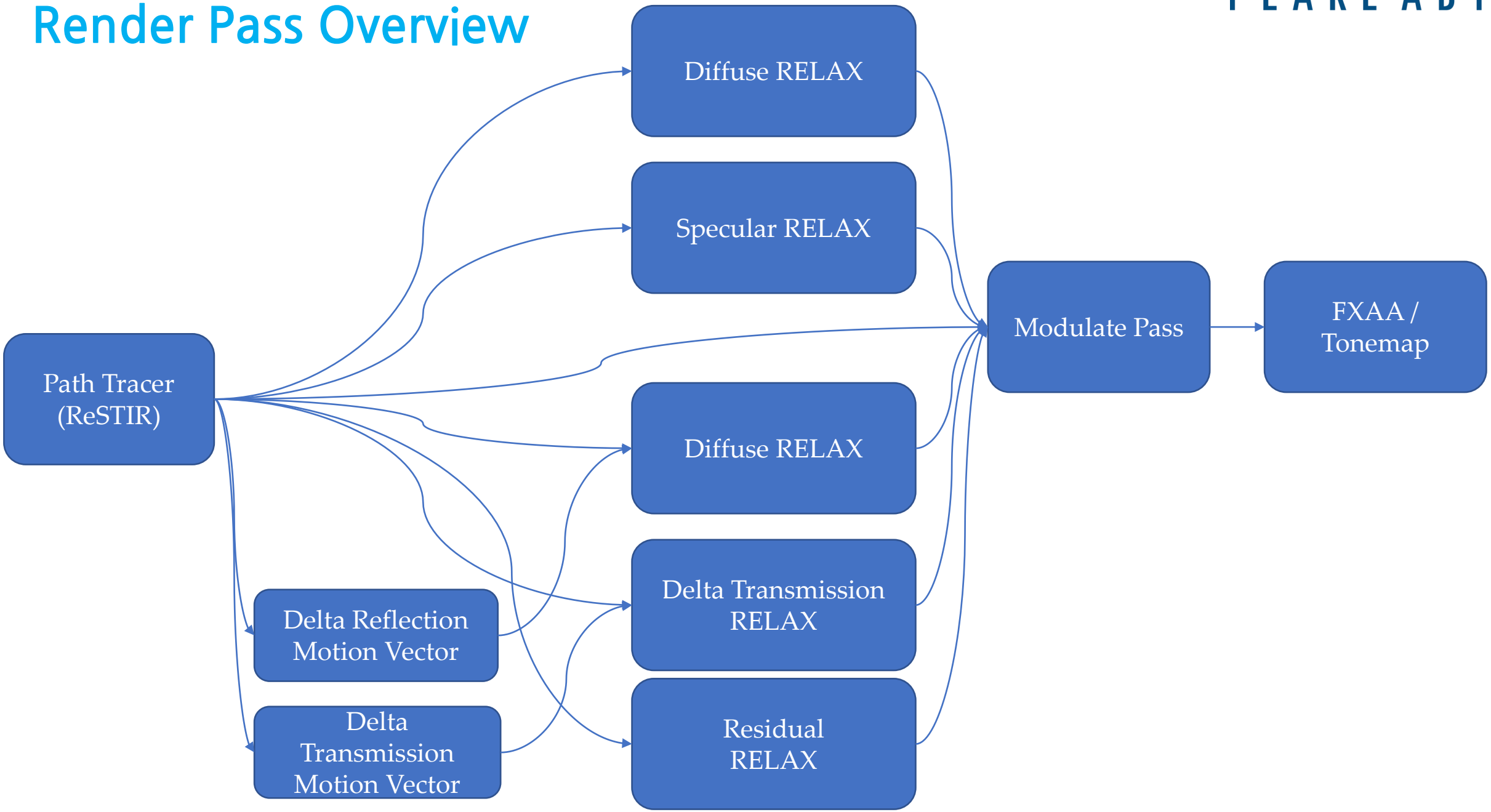
Light Weight

1-spp
Rendered Image

(no accumulation)

Render Pass Overview

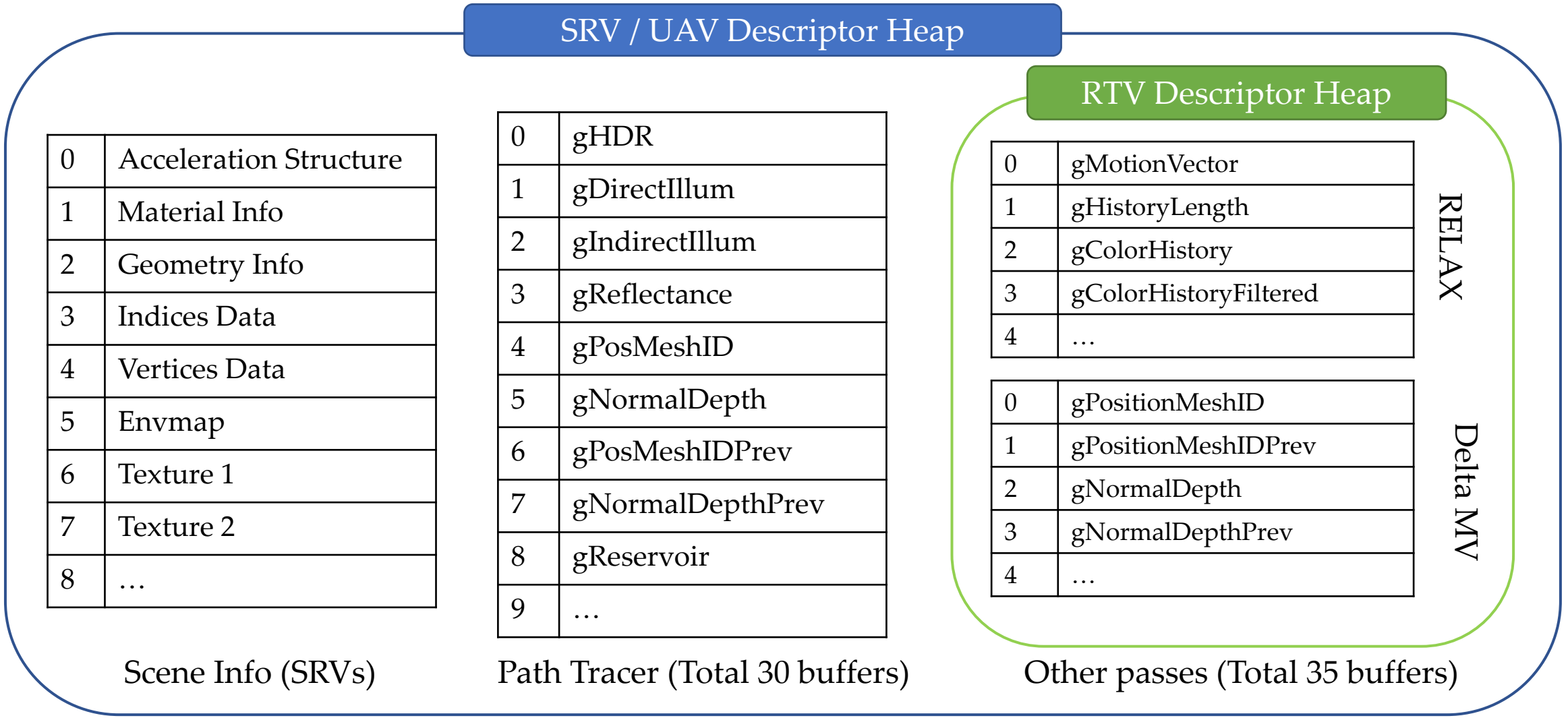
PEARL ABYSS



Pipeline Overview

※ Why single descriptor heap?

Changing descriptor heaps can incur a pipeline flush on some hardware. Because of this, it is recommended to use a single shader-visible heap of each type, and set it once per frame, rather than regularly changing the bound descriptor heaps. Instead, use



All resources (**ID3DResource / GPU / CPU Handles**) are managed by dictionary → resouceDictionary[“gNormal”]

Code Example

PEARL ABYSS

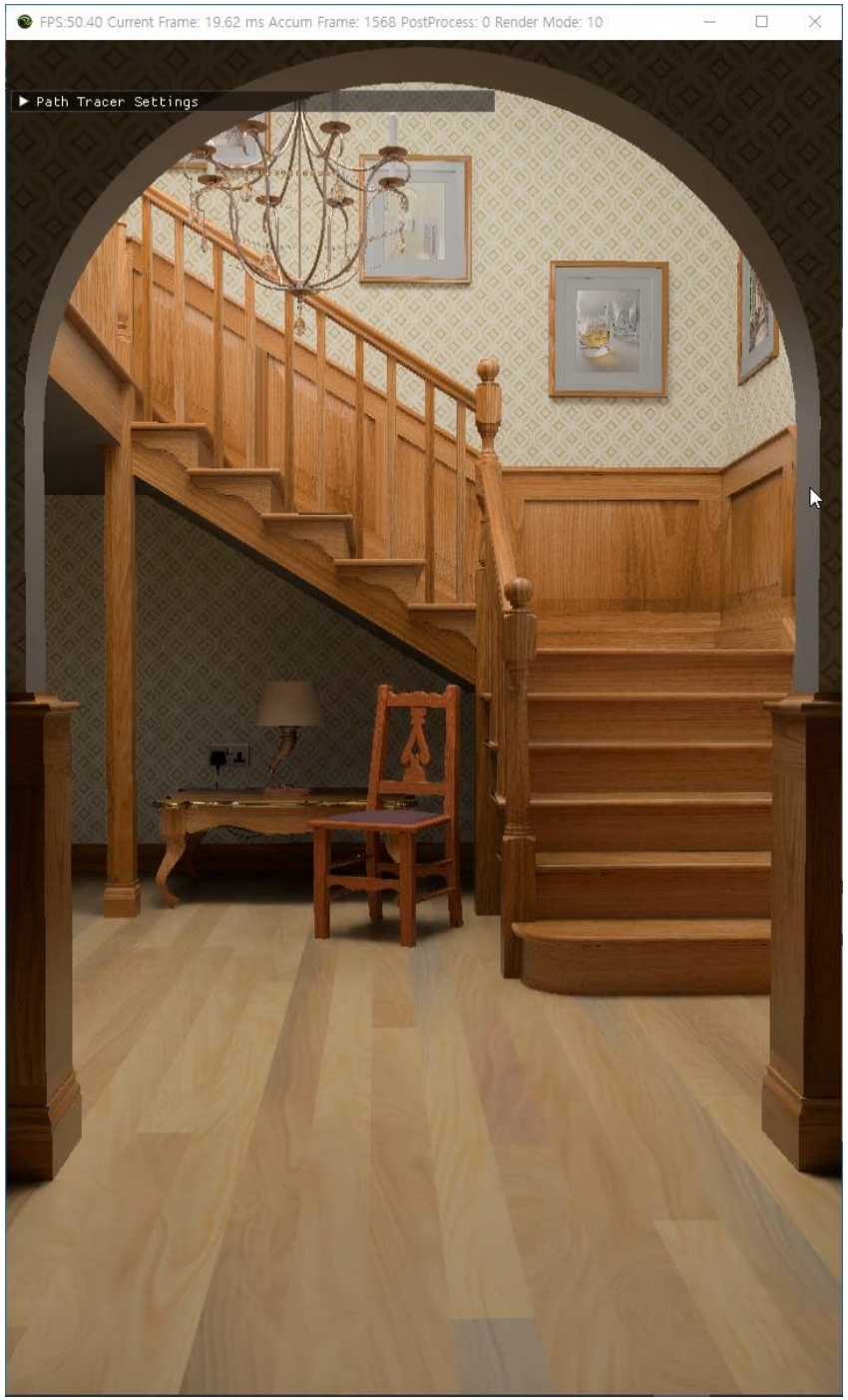
```
RenderData deltaReflectionMotionVectorRenderData;
deltaReflectionMotionVectorRenderData.gpuHandleDictionary["gPositionMeshIDPrev"] = renderDataPathTracer.outputGPUHandleDictionary["gPositionMeshIDPrev"];
deltaReflectionMotionVectorRenderData.gpuHandleDictionary["gNormalDepthPrev"] = renderDataPathTracer.outputGPUHandleDictionary["gNormalDepthPrev"];
deltaReflectionMotionVectorRenderData.gpuHandleDictionary["gPositionMeshID"] = renderDataPathTracer.outputGPUHandleDictionary["gPositionMeshID"];
deltaReflectionMotionVectorRenderData.gpuHandleDictionary["gNormalDepth"] = renderDataPathTracer.outputGPUHandleDictionary["gNormalDepth"];
deltaReflectionMotionVectorRenderData.gpuHandleDictionary["gDeltaReflectionPositionMeshID"] = renderDataPathTracer.outputGPUHandleDictionary["gDeltaReflectionPositionMeshID"];
deltaReflectionMotionVectorRenderData.gpuHandleDictionary["gDeltaReflectionPositionMeshIDPrev"] = renderDataPathTracer.outputGPUHandleDictionary["gDeltaReflectionPositionMeshIDPrev"];
deltaReflectionMotionVectorRenderData.gpuHandleDictionary["gPathType"] = renderDataPathTracer.outputGPUHandleDictionary.at("gPathType");

deltaReflectionMotionVectorPass->forward(&renderContext, deltaReflectionMotionVectorRenderData);
```

Passing required inputs

```
void MotionVectorDeltaReflection::forward(RenderContext* pRenderContext, RenderData& renderData)
{
    ID3D12GraphicsCommandList4Ptr pCmdList = pRenderContext->pCmdList;
    map<string, D3D12_GPU_DESCRIPTOR_HANDLE>& gpuHandles = renderData.gpuHandleDictionary;
    this->setViewPort(pCmdList);

    // Render to motionVectorRenderTexture !!
    pCmdList->SetPipelineState(mpMotionVectorShader->getPipelineStateObject());
    pCmdList->SetGraphicsRootSignature(mpMotionVectorShader->getRootSignature()); // set the root signature

    pCmdList->ResourceBarrier(1, &CD3DX12_RESOURCE_BARRIER::Transition(mpMotionVectorRenderTexture->mResource, D3D12_RESOURCE_STATE_PRESENT, D3D12_RESOURCE_STATE_RENDER_TARGET));

    D3D12_CPU_DESCRIPTOR_HANDLE motionVectorRTV[1] = { mpMotionVectorRenderTexture->mRtvDescriptorHandle };
    pCmdList->OMSetRenderTargets(1, motionVectorRTV, FALSE, nullptr);

    pCmdList->SetGraphicsRootDescriptorTable(1, gpuHandles.at("gPositionMeshIDPrev"));
    pCmdList->SetGraphicsRootDescriptorTable(2, gpuHandles.at("gNormalDepthPrev"));
    pCmdList->SetGraphicsRootDescriptorTable(3, gpuHandles.at("gPositionMeshID"));
    pCmdList->SetGraphicsRootDescriptorTable(4, gpuHandles.at("gNormalDepth"));
    pCmdList->SetGraphicsRootDescriptorTable(5, gpuHandles.at("gDeltaReflectionPositionMeshID"));
    pCmdList->SetGraphicsRootDescriptorTable(6, gpuHandles.at("gDeltaReflectionPositionMeshIDPrev"));
    pCmdList->SetGraphicsRootDescriptorTable(7, gpuHandles.at("gPathType"));

    pCmdList->SetGraphicsRootConstantBufferView(0, pRenderContext->pSceneResourceManager->getCameraConstantBuffer()->GetGPUVirtualAddress());

    pCmdList->DrawInstanced(6, 1, 0, 0);
    pCmdList->ResourceBarrier(1, &CD3DX12_RESOURCE_BARRIER::Transition(mpMotionVectorRenderTexture->mResource, D3D12_RESOURCE_STATE_RENDER_TARGET, D3D12_RESOURCE_STATE_PRESENT));

    renderData.outputGPUHandleDictionary["gMotionVector"] = mpMotionVectorRenderTexture->getGPUSrvHandler();

    return;
}
```

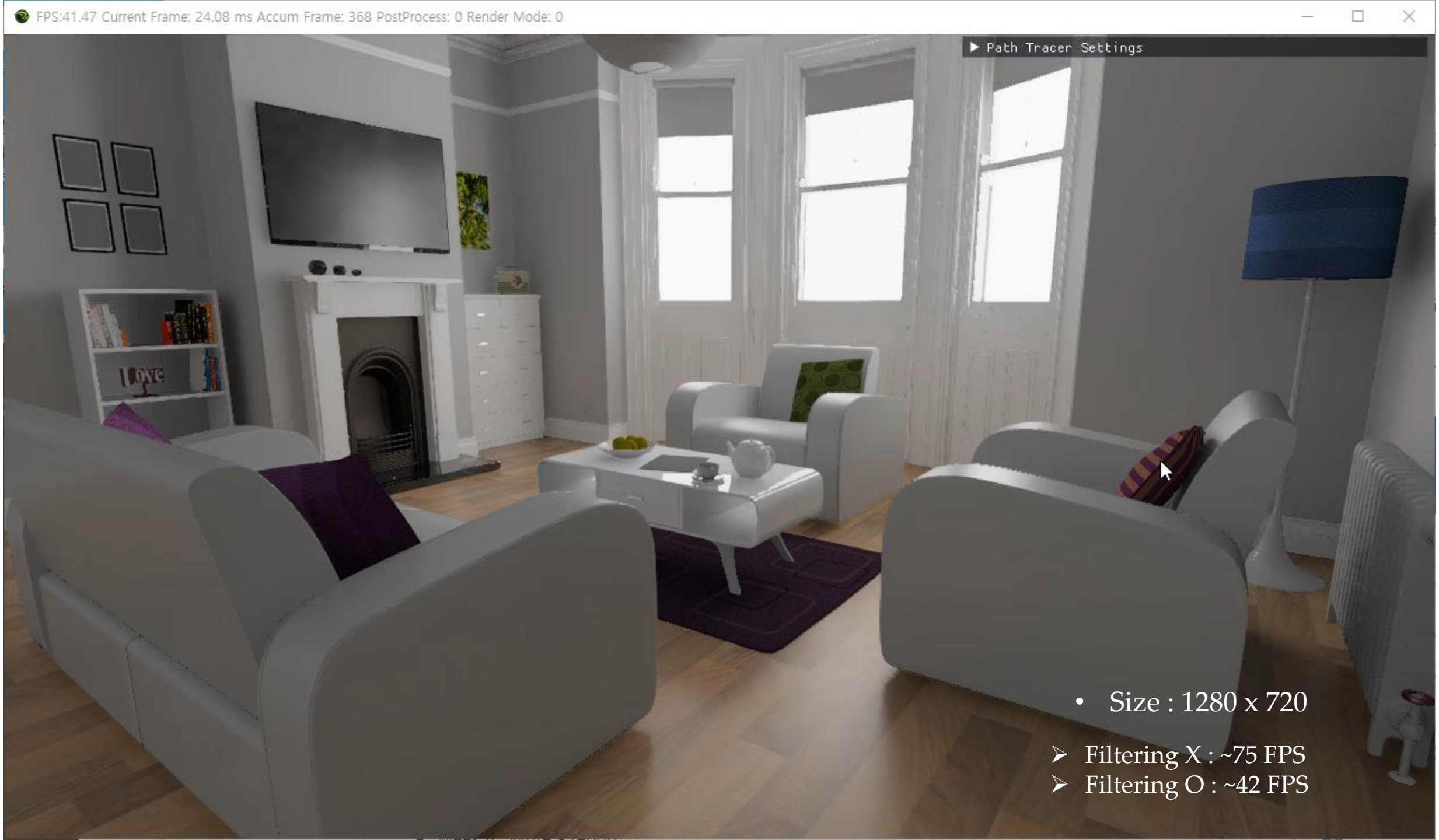
Renderpass.forward()

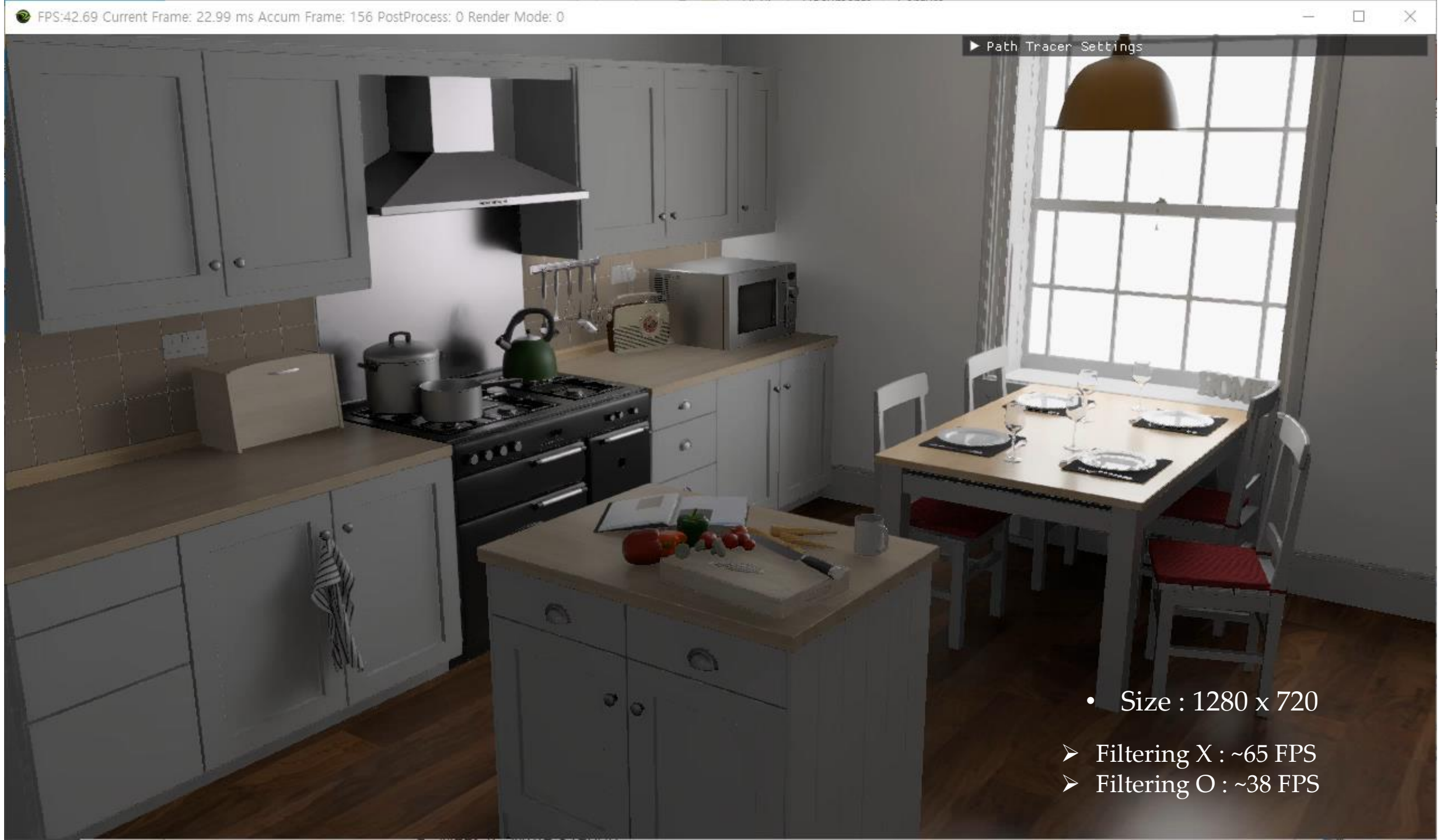
Result & Analysis

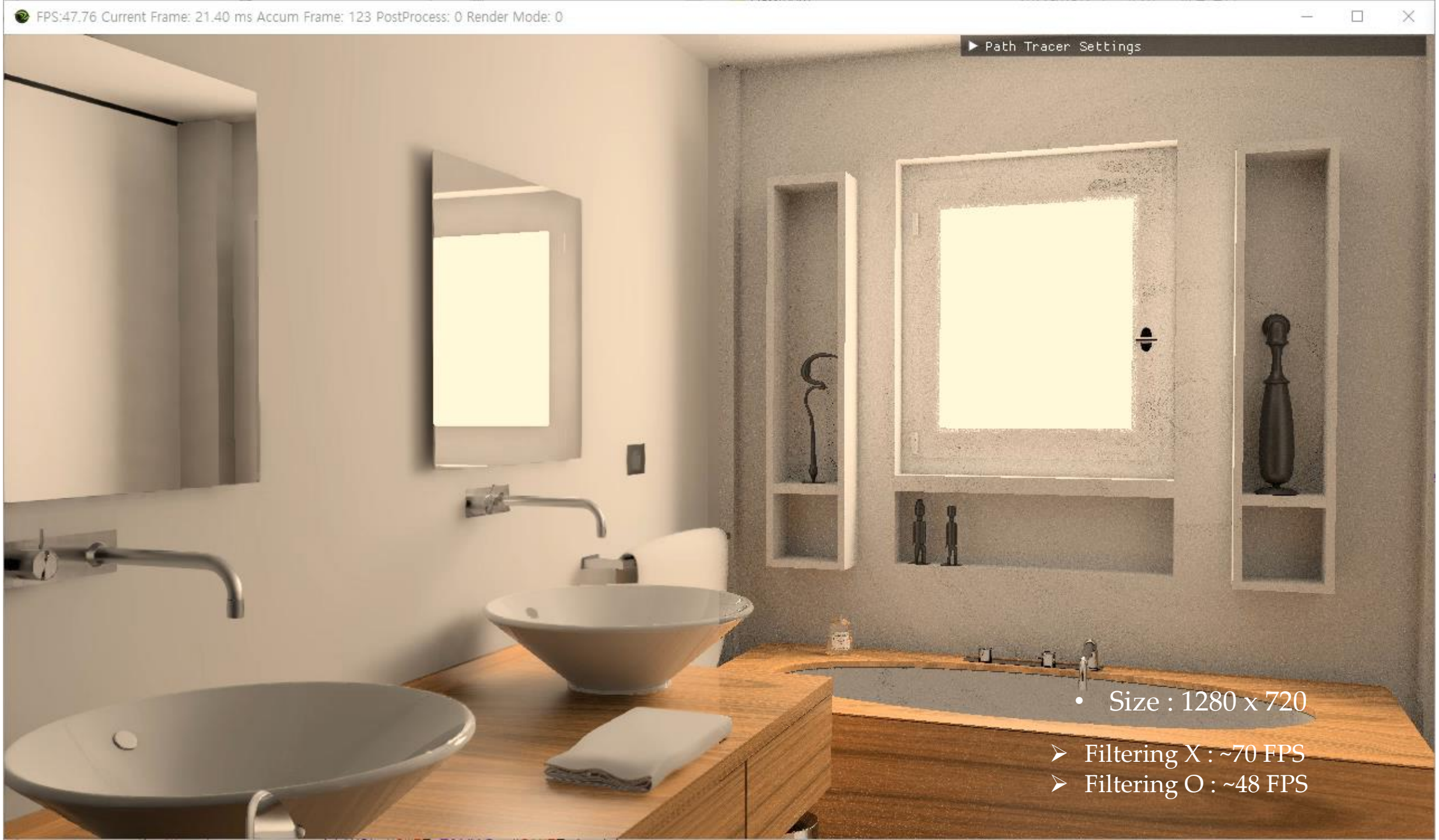


- Size : 720 x 1280

➢ Filtering X : 102.42 FPS
➢ Filtering O : 50.43 FPS


FPS:41.47 Current Frame: 24.08 ms Accum Frame: 368 PostProcess: 0 Render Mode: 0
▶ Path Tracer Settings
• Size : 1280 x 720
➢ Filtering X : ~75 FPS
➢ Filtering O : ~42 FPS


FPS:42.69 Current Frame: 22.99 ms Accum Frame: 156 PostProcess: 0 Render Mode: 0
▶ Path Tracer Settings
• Size : 1280 x 720
➢ Filtering X : ~65 FPS
➢ Filtering O : ~38 FPS


FPS:47.76 Current Frame: 21.40 ms Accum Frame: 123 PostProcess: 0 Render Mode: 0
Path Tracer Settings
• Size : 1280 x 720
➢ Filtering X : ~70 FPS
➢ Filtering O : ~48 FPS

Memory Analysis

PEARL ABYSS

※ Pass Name (# of screen-sized texture)

Path Tracer (30)

```
RWTexture2D<float4> gOutput : register(u0);
RWTexture2D<float4> gOutputHDR : register(u1);
RWTexture2D<float4> gDirectIllumination : register(u2);
RWTexture2D<float4> gIndirectIllumination : register(u3);
RWTexture2D<float4> gDiffuseRadiance : register(u4);
RWTexture2D<float4> gSpecularRadiance : register(u5);
RWTexture2D<float4> gEmission : register(u6);

RWTexture2D<float4> gReflectance : register(u7);
RWTexture2D<float4> gDiffuseReflectance : register(u8);
RWTexture2D<float4> gSpecularReflectance : register(u9);

RWTexture2D<float4> gPositionMeshID : register(u10);
RWTexture2D<float4> gNormalDepth : register(u11);
RWTexture2D<float4> gPositionMeshIDPrev : register(u12);
RWTexture2D<float4> gNormalDepthPrev : register(u13);

RWTexture2D<float4> gDeltaReflectionReflectance : register(u16);
RWTexture2D<float4> gDeltaReflectionEmission : register(u17);
RWTexture2D<float4> gDeltaReflectionRadiance : register(u18);

RWTexture2D<float4> gDeltaTransmissionReflectance : register(u19);
RWTexture2D<float4> gDeltaTransmissionEmission : register(u20);
RWTexture2D<float4> gDeltaTransmissionRadiance : register(u21);

RWTexture2D<float4> gDeltaReflectionPositionMeshID : register(u22);
RWTexture2D<float4> gDeltaReflectionNormal : register(u23);
RWTexture2D<float4> gDeltaTransmissionPositionMeshID : register(u24);
RWTexture2D<float4> gDeltaTransmissionNormal : register(u25);

RWTexture2D<float4> gResidualRadiance : register(u26);
RWTexture2D<uint> gPrimaryPathType : register(u27);
RWTexture2D<float> gRoughness : register(u28);
RWTexture2D<float2> gMotionVector : register(u29);
```

Delta MV (1 x 2 pass)

```
Texture2D gPositionMeshIDPrev : register(t0);
Texture2D gNormalDepthPrev : register(t1);
Texture2D gPositionMeshID : register(t2);
Texture2D gNormalDepth : register(t3);
Texture2D gDeltaReflectionPositionMeshID : register(t4);
Texture2D gDeltaReflectionPositionMeshIDPrev : register(t5);
Texture2D<uint> gPathType : register(t6);
```

RELAX (6 x 5 pass)

Temporal Accum (3)

```
Texture2D gColorVariance : register(t0);
Texture2D gNormalDepth : register(t1);
Texture2D gPositionMeshID : register(t2);
Texture2D gDepthDerivative : register(t3);
Texture2D<uint> gPathType : register(t4);

#ifdef RELAX_SPECULAR
Texture2D gDeltaReflectionNormal : register(t5);
Texture2D gDeltaReflectionPositionMeshID : register(t6);
Texture2D<float> gRoughness : register(t7);
#endif
```

Disocclusion Fix (1)

```
Texture2D gColorVariance : register(t0);
Texture2D gNormalDepth : register(t1);
Texture2D gPositionMeshID : register(t2);
Texture2D gMoments : register(t3);
Texture2D gHistoryLength : register(t4);
Texture2D gDepthDerivative : register(t5);
Texture2D<uint> gPathType : register(t6);
```

Wavelet Filter (2)

```
Texture2D gColorHistory : register(t0);
Texture2D gCurrentColor : register(t1);
Texture2D<float2> gMomentsHistory : register(t2);
Texture2D<float> gHistoryLength : register(t3);
Texture2D<float2> gMotionVector : register(t4);
Texture2D gPositionMeshID : register(t5);
Texture2D gPositionMeshIDPrev : register(t6);
Texture2D gNormalDepth : register(t7);
Texture2D gNormalDepthPrev : register(t8);
Texture2D<uint> gPathType : register(t9);
Texture2D<float> gRoughness : register(t10);
```

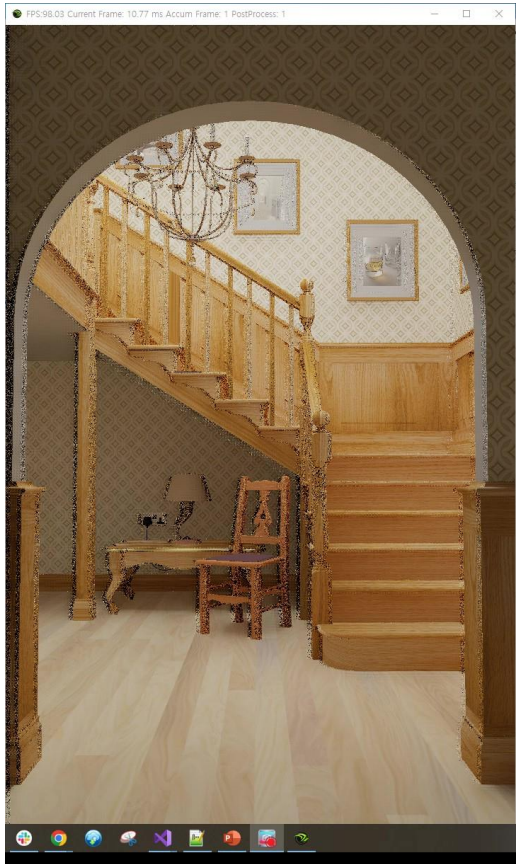
Modulate (1)
FXAA (1)
Tonemap (1)

Total 65

~960 MB (1280x720, RGBA32)

Performance Analysis

PEARL ABYSS



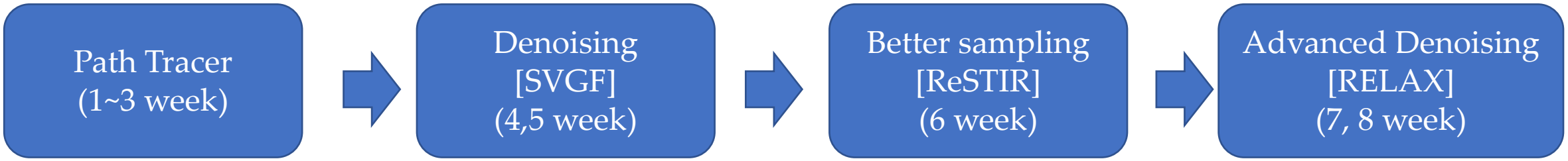
720 x 1280

Pass Name	Elapsed Time (ms)
Path Trace	8.56
Write to G-Buffer	3.69
Delta Motion Vector	2.55
Wavelet	5.31
Wavelet (5)	9.53
Total	20.33

Project Overview

PEARL ABYSS

- Project Goal : Implement a **1-spp real-time** path tracer with denoising & better sampling technique.



- Implement a **real-time path tracer using DX12**.
- Study basics of DX12 and physically based rendering.

- Implement **denoising** technique for a path-traced image.
- Choose to implement SVGF (2017).

- Implement **sampling quality enhancement** technique.
- Choose to implement ReSTIR (2020)

- Implement advanced denoising technique that can handle various materials.
- Choose to implement RELAX (2021)

Limitation

PEARL ABYSS

- Specular RELAX is still not working well.
- Delta motion vector is not accurate.
- Overall performance (especially memory → use too much buffers)

Thank you