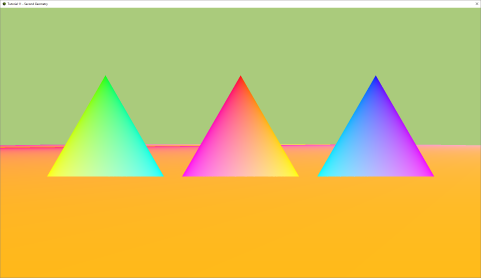
DXR Tutorial 12

Per-Geometry Hit-Shader

# Overview

In the previous tutorial we added a new geometry – a plane. The result wasn’t that impressive – the plane used the same hit-shader and the same vertex colors as for the triangle, resulting in colorful image.



In this tutorial, we will implement a new hit-shader specific to the plane and show how to invoke it when the plane is hit by a ray.

# Plane Hit-Program

For the plane, we will create a simple hit-program which returns a constant color. The following code can be found in ’*12-Shaders.hlsl*’

[shader("closesthit")]

void planeChs(inout RayPayload payload, in BuiltInTriangleIntersectionAttributes attribs)

{

payload.color = 0.9f;

}

We need to make the following changes to **createRtPipelineState():**

* Create a new HitProgram for the plane CHS (line 752).
* Associate the empty-root signature with the new plane hit-group (line 779)
* Associate the shader-config sub-object with the plane hit-group (line 787)

# Shader-Table Layout

We would like the ray-tracing pipeline to invoke the new hit-program when the plane is hit. In tutorial 10 we learned that the hit-program indexing is computed as follows:

(HitStartAddress +

InstanceContributionToHitGroupIndex +

GeometryIndex \* MultiplierForGeometryContributionToShaderIndex +

RayContributionToHitGroupIndex)

To understand how it can be used to invoke a different hit-program, let’s look again at our TLAS.

We have 3 instances:

* Instance 0 with 2 geometries.
  + Geometry 0 – triangle
  + Geometry 1 – plane
* Instance 1 – single geometry, a triangle
* Instance 2 – single geometry, a triangle

Geometries in the same instance share the same InstanceContributionToHitGroupIndex. To direct different geometries to different shader-table records, we need to use GeometryIndex.

Our new shader-table will look like this:

**Hit**

**Instance 2**

**Geom 0**

**Miss**

**Hit**

**Instance 1**

**Geom 0**

**Hit**

**Instance 0**

**Geom 1**

**Hit**

**Instance 0**

**Geom 0**

**RayGen**

Let’s see how this layout works with the hit-program index computation:

* BaseIndex is 2. It’s shared between all instances and geometries.
* InstanceContributionToHitGroupIndex is per instance, specified when building the TLAS.
  + For instance 0 it will be 0.
  + For instance 1 it will be 2 (we need to skip both geometries in instance 0).
  + For instance 2 it will be 3.
* GeometryIndex is generated automatically by the pipeline. This is the index of the geometry within an instance.
  + This value will be 0 for all the triangles, since they are the first geometry in the instance.
  + It will be 1 for the plane, since it’s the second geometry in the first instance.
* MultiplierForGeometryContributionToShaderIndex should be 1.
  + This value doesn’t affect the triangles (their GeometryIndex is 0).
  + For the plane, (GeometryIndex \* MultiplierForGeometryContributionToShaderIndex) will result in 1, which is the required offset of the record relative to the start of the instance.
* RayContributionToHitGroupIndex should be 0.

You can plug these values into the formula above to see the final value for each geometry.

# Shader-Table Changes

Now that we understand the new layout and the indexing, we can make the required code changes.

First, we need to create a larger shader-table. We need 6 entries in total. This happens at the beginning of createShaderTable().

Next, we need to initialize the shader-table hit-program records. The first entry is for the triangle in instance 0:

// Entry 2 - Triangle 0 hit program. ProgramID and constant-buffer data

uint8\_t\* pEntry2 = pData + mShaderTableEntrySize \* 2;

*memcpy*(pEntry2, pRtsoProps->GetShaderIdentifier(kTriHitGroup), progIdSize);

\*(D3D12\_GPU\_VIRTUAL\_ADDRESS\*)(pEntry2 + progIdSize) = mpConstantBuffer[0]->GetGPUVirtualAddress();

This code is similar to the code from the previous tutorials.

Now let’s initialize the entry for the plane. We have no shader resources, so we only need to set the program identifier of the plane hit-program.

// Entry 3 - Plane hit program. ProgramID only

uint8\_t\* pEntry3 = pData + mShaderTableEntrySize \* 3;

*memcpy*(pEntry3, pRtsoProps->GetShaderIdentifier(kPlaneHitGroup), progIdSize);

Entries 4 and 5 are for the 2 other triangles. The code is very similar to the code we used for the first triangle. You can find the code at lines 861-871.

Three final changes:

* We need to change the InstanceContributionToHitGroupIndex for the second and third instances. This happens during TLAS creation, on line 410.
* Hit the ray-generation shader (*12-Shaders.hlsl*), we need to change the **TraceRay()** call. We need to pass `1` as the MultiplierForGeometryContributionToShaderIndex argument.
* In onFrameRender(), set raytraceDesc.HitGroupTable.*SizeInBytes* to mShaderTableEntrySize \* 4.

And that should do it!

