DXR Tutorial 11

Adding a Second Geometry

# Overview

So far, we used a single mesh, which had a single triangle it. We used instancing to render 3 triangles to the screen. However, in real-world scenarios scenes are comprised of thousands of meshes. In this tutorial we learn how to support multiple geometries by way of adding a plane to our scene.

# Acceleration Structures Revisited

Time to talk a little bit about the acceleration structure hierarchy. During the previous tutorials we mentioned the top-level and bottom-level acceleration structures a lot. We also mentioned briefly that there’s something called geometry. Let’s fully understand what each means conceptually. We will only discuss triangles. We will not cover axis-aligned bounding-box geometries, but know that they exist.

## Geometry

A geometry describes a single mesh. It must have exactly 1 vertex buffer which contains the positions (it can also contain other vertex-attributes). An index-buffer is optional. The only supported topology is triangle-list. We can optionally provide a transformation matrix which will be applied to the positions **at build time**.

A geometry is described using the D3D12\_RAYTRACING\_GEOMETRY\_DESC struct.

## Bottom-Level Acceleration Structure

A BLAS is a collection of geometries. If a geometry describes a mesh, we can think of the BLAS as a model (made up of multiple meshes). The BLAS defines a local-space for the geometries it contains.

## Top-Level Acceleration Structure

If a geometry describes a mesh and a BLAS defines a model, then the TLAS is a scene. It is a collection of BLAS instances. Each instance is described by a BLAS and a transformation matrix.

# Our Goal

At the end of this tutorial we will use all the concept above. We will have 2 bottom-level acceleration structures:

1. Single geometry, containing our trusty triangle.
2. 2 geometries – a triangle and a plane.

We will only make changes to the acceleration structure code. We do not need to change the shader-table.

# Creating the Plane

We need to create a vertex buffer for the plane. This is standard DX12 code – see **createPlaneVB()**.

# Bottom-Level Acceleration Structures

We need 2 bottom-level acceleration structures. The code for both is very similar, so we will be using the same function.

AccelerationStructureBuffers createBottomLevelAS(ID3D12RtDevicePtr pDevice,

ID3D12GraphicsCommandListPtr pCmdList,

ID3D12ResourcePtr pVB[],

const uint32\_t vertexCount[],

uint32\_t geometryCount);

The function now accepts an array of vertex buffers. We call it once with both the plane and the triangle, and once just with the triangle (the calls are in **createAccelerationStructures()**).

The first thing we do is initialize an **array** of D3D12\_RAYTRACING\_GEOMETRY\_DESC.

D3D12\_RAYTRACING\_GEOMETRY\_DESC geomDesc[2] = {};

for (uint32\_t i = 0; i < geometryCount; i++)

{

geomDesc[i].Type = D3D12\_RAYTRACING\_GEOMETRY\_TYPE\_TRIANGLES;

geomDesc[i].Triangles.VertexBuffer.StartAddress = pVB[i]->GetGPUVirtualAddress();

geomDesc[i].Triangles.VertexBuffer.StrideInBytes = sizeof(vec3);

geomDesc[i].Triangles.VertexCount = vertexCount[i];

geomDesc[i].Triangles.VertexFormat = *DXGI\_FORMAT\_R32G32B32\_FLOAT*;

geomDesc[i].Flags = D3D12\_RAYTRACING\_GEOMETRY\_FLAG\_OPAQUE;

}

The rest of the code is similar to the code in the previous tutorials, except we use the array and geometryCount when querying for the prebuild info and when creating the BLAS.

# Top-Level Acceleration Structure

Nothing fancy in the code here.

We are still initializing 3 D3D12\_RAYTRACING\_INSTANCE\_DESC structures. The difference is the we initialize the first D3D12\_RAYTRACING\_INSTANCE\_DESC instance with the bottom-level acceleration structure containing both geometries.

// Create the desc for the triangle/plane instance

instanceDescs[0].InstanceID = 0;

instanceDescs[0].InstanceContributionToHitGroupIndex = 0;

instanceDescs[0].Flags = D3D12\_RAYTRACING\_INSTANCE\_FLAG\_NONE;

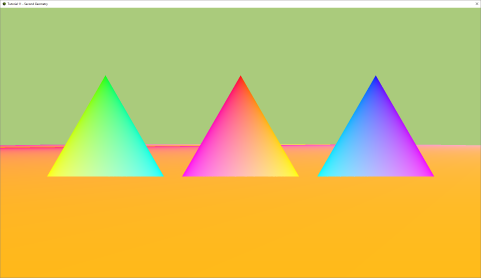
*memcpy*(instanceDescs[0].Transform, &transformation[0], sizeof(instanceDescs[0].Transform));

instanceDescs[0].AccelerationStructure = pBottomLevelAS[0]->GetGPUVirtualAddress();

instanceDescs[0].InstanceMask = 0xFF;

The code for the other instance descs remains the same and will not be repeated here.

Launching the application, we can see the result.



As you can see, the plane uses the same hit-shader and vertex-colors as the first triangle. That’s because they share the same shader-table record. In the next tutorial we will learn how to use a different shader-table record for each geometry, which will allow us to use different resources and even execute a different shader for each geometry.