DXR Tutorial 14

Refit

# Overview

So far, we’ve only handled static meshes. We created the top-level acceleration structure once and assumed that the scene stays static.

In this tutorial, we will learn how to handle dynamic objects by making the outer triangles rotate.

# Rebuild vs Refit

We can animate objects by manipulating their transformation matrix used when creating the TLAS (D3D12\_RAYTRACING\_INSTANCE\_DESC::Transform).

There are 2 options to update the TLAS:

* Rebuild – Creates the TLAS from scratch. Doesn’t use any information from previous builds.
* Refit – **Update** an existing TLAS.

According to the spec, there are different pros and cons for each option. The refit operation is usually faster than rebuild, but traversing a TLAS that supports updates might be slower. As we’ll see in a second, it’s straightforward to switch between the 2 options. This makes it very simple to benchmark both options.

We already know how to build (and therefore rebuild) a TLAS. This tutorial will focus on refit.

# Refitting a TLAS

The code for refitting a TLAS is almost identical to the code creating a TLAS. We need to go through the same steps – allocating scratch, result, and instance-desc buffers, initializing the instance descriptors, and calling **BuildRaytracingAccelerationStructure()**).

There are 3 differences in the arguments we pass to **BuildRaytracingAccelerationStructure()**:

1. We need to create the TLAS with the D3D12\_RAYTRACING\_ACCELERATION\_STRUCTURE\_BUILD\_FLAG\_ALLOW\_UPDATE flag. We also need to pass this flag to **GetRaytracingAccelerationStructurePrebuildInfo()**.
2. When refitting, we need to set the D3D12\_RAYTRACING\_ACCELERATION\_STRUCTURE\_BUILD\_FLAG\_PERFORM\_UPDATE of D3D12\_BUILD\_RAYTRACING\_ACCELERATION\_STRUCTURE\_DESC.
3. When refitting, we need to set a source TLAS buffer into the SourceAccelerationStructureData field of D3D12\_BUILD\_RAYTRACING\_ACCELERATION\_STRUCTURE\_DESC.

Conceptually, these are the only differences. That’s good news, as it means we already have most of what we need to support animation.

# Code Walkthrough

First, let’s change the code that creates the TLAS. We renamed it and changed the signature.

void buildTopLevelAS(ID3D12DevicePtr pDevice,

ID3D12GraphicsCommandListPtr pCmdList,

ID3D12ResourcePtr pBottomLevelAS[2],

uint64\_t& tlasSize,

float rotation,

bool update,

DxrtSample::AccelerationStructureBuffers& buffers)

The last 3 arguments are new:

* rotation – Rotation in radians relative to the Y axis. We will apply this rotation to the 2 outer triangles.
* update – True if this is a refit operation, otherwise false. Remember that we must create the TLAS once before we can update it.
* buffers – Up to now we’ve only stored the result buffer. To avoid reallocating the scratch and instance-desc buffers every frame, we will store them as members.

First, we query for the required buffer sizes for a TLAS that supports updating by passing the D3D12\_RAYTRACING\_ACCELERATION\_STRUCTURE\_BUILD\_FLAG\_ALLOW\_UPDATE flag.

D3D12\_GET\_RAYTRACING\_ACCELERATION\_STRUCTURE\_PREBUILD\_INFO\_DESC prebuildDesc = {};

prebuildDesc.DescsLayout = D3D12\_ELEMENTS\_LAYOUT\_ARRAY;

prebuildDesc.Flags = D3D12\_RAYTRACING\_ACCELERATION\_STRUCTURE\_BUILD\_FLAG\_ALLOW\_UPDATE;

prebuildDesc.NumDescs = 1;

prebuildDesc.Type = D3D12\_RAYTRACING\_ACCELERATION\_STRUCTURE\_TYPE\_TOP\_LEVEL;

Note that we only allocate the buffers if this is a create operation (update == false).

Next, when creating the instance descriptors, you can see we apply rotation to the outer triangles.

mat4 rotationMat = eulerAngleY(rotation);

transformation[1] = translate(mat4(), vec3(-2, 0, 0)) \* rotationMat;

transformation[2] = translate(mat4(), vec3(2, 0, 0)) \* rotationMat;

Finally,if this is an update operation, we set the source buffer and the perform-update flag into the D3D12\_BUILD\_RAYTRACING\_ACCELERATION\_STRUCTURE\_DESC struct used when calling **BuildRaytracingAccelerationStructure()**.

// If this is an update operation, set the source buffer and the perform\_update flag

if(update)

{

asDesc.Flags |= D3D12\_RAYTRACING\_ACCELERATION\_STRUCTURE\_BUILD\_FLAG\_PERFORM\_UPDATE;

asDesc.SourceAccelerationStructureData = buffers.pResult->GetGPUVirtualAddress();

}

**NOTE**: There’s a limitation with the current implementation where the source buffer must also be the result buffer.

Finally, we can record a build command.

Notice that we use the ALLOW\_UPDATE flag, pass update and use the source buffer we computed before.

Now it’s time to use this function.

# Load Time TLAS Creation

The only thing that changed in **createAccelerationStructures()** is the fact that we now call the function by its new name and request a `create` operation (see line 465).

# Render-Time TLAS Refit

We added 4 lines of code to the beginning of **onFrameRender()**.

buildTopLevelAS(mpDevice, mpCmdList, mpBottomLevelAS, mTlasSize, mRotation, true, mTopLevelBuffers);

mRotation += 0.005f;

We call **buildTopLevelAS()** and request an update operation and update the rotation.

And we’re done. No shader changes are required. Launch the application and you should see the 2 outer triangles rotate.