DirectX Ray Tracing in Unity 2019.3

Siggraph 2019

Tianliang Ning, DXR Graphics Dev
paulan@unity3d.com
DXR Integration and Overview
DXR Overview

API designed to leverage hardware-accelerated ray tracing

Why trace rays?

- Off-screen rendering (reflection, refraction)
- Algorithms that call for raycasting

Two major concepts to be concerned with:

- Ray Tracing Acceleration Structures: **What** are we drawing?
- Ray Tracing Shaders: **How** should we draw things?
Acceleration Structure

- **Bottom-Level AS**: Geometry only
- BVH construction done by driver
- **Top-Level AS**: Geometry, materials, transforms, hierarchy
Acceleration Structure

New Unity class: **RayTracingAccelerationStructure**

- May be manually or automatically managed
  - Manual: AddInstance(), UpdateInstanceTransform()
- Specify layer masks on creation to filter which GameObjects may be added
- Call BuildRayTracingAccelerationStructure once a frame
Acceleration Structure Management

New **Renderer** setting: **RayTracingMode**

In order from least to most expensive:

1. Off
2. Static
3. DynamicTransform
4. DynamicGeometry
Ray Tracing Shaders

Raytrace Shaders
- RayGen: First shader executed on dispatch
- Miss: Executes if ray fails to intersect with any geometry that has a hit shader

Surface Shaders
- ClosestHit: Executes on hit nearest to ray origin
- AnyHit*: Executes on every intersection

Callable Shaders
Ray Tracing Shaders

- Execute RayGen Shader
- Execute ClosestHit Shader
- Execute Miss Shader
- Don’t Execute ClosestHit Shader
- May Execute AnyHit Shader (results unused)
- Execute AnyHit Shader
Ray Tracing Shader API

- New shader type: RayTracingShader
  - Extension is .raytrace
- New CommandBuffer API:
  - CommandBuffer.SetRayTracingShaderPass
  - CommandBuffer.SetRayTracingAccelerationStructure
  - CommandBuffer.SetRayTracing*Param
    - e.g. SetRayTracingMatrixParam, SetRayTracingIntParam, etc.
  - CommandBuffer.DispatchRays
  - Analogous bindings also available from RayTracingShader class itself, for immediate execution
// Dispatch shader: Defines at minimum a ray generation shader, often also a miss shader.
// This is the shader that is dispatched, as one would a compute shader,
// for a given ray traced pass.

struct RayPayload { float4 color; uint2 launchIdx; }; // User-defined

[shader("raygeneration")]
void FullResRayGen()
{
    uint2 launchIdx = DispatchRaysIndex().xy; // DXR callback
    uint2 launchDim = DispatchRaysDimensions().xy; // DXR callback
    float2 ndcCoords = (launchIdx / float2(launchDim.x - 1, launchDim.y - 1)) * 2 - float2(1, 1);
    float3 viewDirection = normalize(float3(ndcCoords.x * aspectRatio, ndcCoords.y, -1);
    RayDesc ray; // DXR defined
    ray.Origin = float3(camera_IV[0][3], camera_IV[1][3], camera_IV[2][3]);
    ray.Direction = normalize(mul(camera_IV, viewDirection));
    ray.TMin = 0;
    ray.TMax = 1e20f;
    RayPayload payload;
    payload.color = float4(0, 0, 0, 0);
    TraceRay(accelerationStructure, 0, 0xFF, 0, 1, 0, ray, payload); // DXR callback
}

Ray Tracing Shader Authoring
Ray Tracing Shader Authoring

```c
[shader(“miss”)]
void SampleSkybox(inout RayPayload payload : SV_RayPayload)
{
    rayDirection = WorldRayDirection();
    float4 skyboxColor = skyboxTex.SampleLevel(linearRepeatSampler, rayDirection, 0);
    payload.color = skyboxColor;
}
```

// These slides have a good introduction to built-in DXR callbacks:
// Material/Surface shader: Hit shaders should be defined as a pass in a shader used for a
// material in the scene.
Shader "FlatColor"
{
    SubShader {
        Pass {
            CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag
            v2f vert (appdata v) { return UnityObjectToClipPos(v.vertex); }
            fixed4 frag (v2f i) : SV_Target { return albedo; }
            END CGPROGRAM
        }
    }
    SubShader {
        Pass {
            Name "DefaultRTPass"
            HLSLPROGRAM
            #pragma raytracing
            struct AttributeData { float2 barycentrics; }; // User-defined
            [shader(“closesthit”)]
            void FullResRayGen(inout RayPayload payload : SV_RayPayload,
                                AttributeData attribs : SV_IntersectionAttributes)
            {
                // A trivial hit shader that populates a bound RT with albedo of hit object
                payload.color = albedo;
                outputRT[payload.launchIdx] = albedo;
            }
            ENDHLSL
        }
    }
}
Setup Requirements for DXR in Unity

- Windows 10 v1809+
- Unity 2019.3b1+
- Graphics card with latest drivers:

Unity Project settings:
- Select DX12 as Windows Graphics API

<table>
<thead>
<tr>
<th>BASIC RT EFFECTS</th>
<th>COMPLEX RT EFFECTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOW RAY COUNT</td>
<td>multiple RT effects</td>
</tr>
<tr>
<td></td>
<td>high ray count</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PASCAL</th>
<th>TURING</th>
<th>TURING RTX</th>
</tr>
</thead>
<tbody>
<tr>
<td>TITAN XP</td>
<td>GTX 1660 TI</td>
<td>TITAN RTX</td>
</tr>
<tr>
<td>TITAN X</td>
<td>GTX 1660</td>
<td>RTX 2080 Ti</td>
</tr>
<tr>
<td>GTX 1080 Ti</td>
<td>GTX 1080</td>
<td>RTX 2080</td>
</tr>
<tr>
<td>GTX 1080</td>
<td>GTX 1070</td>
<td>RTX 2070</td>
</tr>
<tr>
<td>GTX 1070 Ti</td>
<td>TITAN V</td>
<td>RTX 2060</td>
</tr>
<tr>
<td>GTX 1070</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GTX 1060 6GB</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

credit: nvidia
Ray Tracing Setup for HDRP

- Everything on the previous slide
- Clone HDRP from Github
- Windows > Render Pipeline > HDRP Wizard > check everything under DXR additional configuration, which takes care of the following:
  - Sets DX12 as graphics API if you haven’t already
  - In Project Settings > Player > Scripting Define Symbols, add REALTIME_RAYTRACING_SUPPORT
  - In HDRP Asset > Rendering, enable Realtime Raytracing
- Find ShaderConfig.hlsl in your local copy of the high-definition-config package, and change #define SHADEROPTIONS_RAYTRACING to (1)
- Add a Game Object > Rendering > Ray Tracing Environment to your scene
- For ray traced shadows: enable screen space shadows in HDRP Asset
State of Unity DXR

- Ray Tracing API is pipeline-agnostic
  - However, it’s only officially supported for HDRP
    - HDRP is also the only pipeline that actually implements features using ray tracing
    - In ShaderGraph, HDRP master nodes for Lit, Unlit, and Fabric support ray tracing
    - Users can still use the public C# API to build their own features!
- Unsupported in 19.3:
  - Intersection shaders
  - Animated meshes
  - Procedural geometry
Ray Tracing Features in the High Definition Render Pipeline
Primary rays for most effects are computed from depth/normal buffers
Cluster-based lighting added to HDRP for ray tracing
Render graph here is simplified and omits many HDRP stages
Ray Traced Effects

Indirect Lighting

Shadows

Transparents
Ray Traced Indirect Lighting

- **Global Illumination (Indirect Diffuse)**
  - Lambert lobe sampling
  - Multiple bounces
  - Temporally accumulated

- **Reflections (Indirect Specular)**
  - **Isotropic GGX** lobe sampling
  - **Split sum approximation**
  - Multiple bounces
  - Temporally accumulated

- **Ambient Occlusion**
  - Not technically lighting
  - Same as GI but only visibility/no color
  - 1 bounce
Cluster Based Light Lists

- Ray Tracing must look up light list given 3D intersection location in scene
- Populated with lights within culling radius of camera
- Debug view shows # of lights affecting a given cluster
Ray Traced Shadows

Directional Lights:
- Ray-traced screen space soft shadows
- Sun modeled as adjustable-size disk

Area Lights:
- Rays cast across surface of area light
- Combined with analytic lighting using a ratio estimator
Transparencies

- Need angle of incidence, so trace primary rays (rather than constructing first hit from GBuffer)
- Each bounce generates 2 rays: one for transmission, one for reflection
- More overlapping transparent layers require more bounces
Spatiotemporal Sampling

Sample count is configured per-effect.

1. **Ray pixel coordinates** are used to sample spatial noise from a **dithered blue-noise texture**
2. Spatial noise results and **frame index used** to sample temporal noise from a looping **Sobol sequence**
3. Resulting value is **mapped to the appropriate PDF** for each effect to calculate **raycast direction**
Denoising

Denoising is done per-effect in a compute shader:

- Temporal sample accumulation
  - Use accumulated samples across previous 8 frames
  - Previous frames reprojected to correct for camera motion
- Separable Bilateral Gaussian filtering
  - Uses depth/normal buffers detect and avoid artifacting at edges
  - Incompatible with transparencies
Optimization Knobs

Ray tracing effects may be accessed in the volume inspector

Per-Effect config

- Ray length
- # samples
- # bounces

Content management

- Mesh count
- Per-effect and per-camera layer masks
- Selective application of effects
Acknowledgements

Anis Benyoub
Sebastien Lagarde
Justin Lee
Mike Lee
Ionut Nedelcu
Marc Scattergood
Soner Sen
Natasha Tatarchuk
Joel de Vahl
We are hiring
Together, we empower real-time creativity around the world.

careers.unity.com

277 open positions across 25 locations in 10 departments
Want to learn more?

Tutorial: **Introduction to the Post Processing Stack**

Tutorial: **Post-Processing Effects (2018.x+)**

Tutorial: **Particle System: Lights**

Unity experts, live sessions, and advanced learning at [learn.unity.com](http://learn.unity.com)
Thank you.

#unity3d
Resources
Ray Tracing Shader Execution

https://microsoft.github.io/DirectX-Specs/d3d/Raytracing.html

AnyHit:

“The TMin value tracked by the system never changes over the lifetime of a ray. On the other hand, as intersections are discovered (in arbitrary spatial order), the system reduces TMax to reflect the closest intersection so far. When all intersections are complete, TMax represents the closest intersection, the relevance of which appears later...”
AnyHit

- IgnoreHit()
- AcceptHitAndEndSearch()
- Otherwise, implicitly accepts hit and continues traversal
Ray Tracing Shaders

- Don’t Execute ClosestHit Shader
- May Execute AnyHit Shader (results unused)
- May Execute Miss Shader (results unused)
- Execute RayGen Shader
- Execute ClosestHit Shader
- Execute AnyHit Shader
- Execute Custom Intersection Shader
- Execute Miss Shader
Hybrid Render Graph
Indirect Specular

\[ L_{\text{indirect specular}} \approx \int_{\Omega} \frac{FGD}{4(\omega_i \cdot n)(v \cdot n)} L_{\text{indirect}} (\omega_i \cdot n) \, d\omega_i \]

\[ L_{\text{indirect specular}} \approx \text{specular} \frac{FGD}{4(h \cdot n)} \int_{\Omega} \frac{(h \cdot n) D}{4(h \cdot n)} L_{\text{indirect}} (\omega_i \cdot n) \, d\omega_i \]