DirectX Raytracing

Matt Sandy
Program Manager
DirectX
Agenda

• Why Raytracing?
• DXR Deep Dive
• Tools & Helpers
• Applied Raytracing (EA/SEED)
• Get Started
3D Graphics is a Lie

- Solving the Visibility Problem
Emergence of Exceptions

• Dynamic Shadows
• Environment Mapping
• Reflections
• Global Illumination
A Brief History of Pixels

• 1999 – Hardware T&L
• 2000 – Simple Programmable Shaders
• 2002 – Complex Programmable Shaders
• 2008 – Compute Shaders
• 2014 – Asynchronous Compute
• 2018...
A Brief History of Pixels

• 1999 – Hardware T&L
• 2000 – Simple Programmable Shaders
• 2002 – Complex Programmable Shaders
• 2008 – Compute Shaders
• 2014 – Asynchronous Compute
• 2018 – DirectX Raytracing
Raytracing 101

1. Construct a 3D representation of a scene.
2. Trace rays into the scene from a point of interest (e.g. camera).
3. Accumulate data about ray intersections.
4. Optional: go to step 2.
5. Process the accumulated data to form an image.
Case Study: SSR
Case Study: SSR
Raytracing Requirements

• Scene geometry representation
• Trace rays into scene and get intersections
• Determine and execute material shaders
Raytracing Requirements

- Scene geometry representation
- Trace rays into scene and get intersections
- Determine and execute material shaders
Acceleration Structures

- Opaque buffers that represent a scene
- Constructed on the GPU
- Two-level hierarchy
Geometries

- Triangles
  - Vertex Buffer (float16x3 or float32x3)
  - Index Buffer (uint16 or uint32)
  - Transformation matrix

- Programmable Geometry
  - Defined using shader code
  - Specify enclosing AABBs
Bottom-Level Acceleration Structure

- Defined by a set of geometries
- Built on the GPU, written to opaque buffer
Top-Level Acceleration Structures

- Defined by a set of instances of bottom-level structures
- Built on the GPU, written to opaque buffer
- Transformation matrix for each instance
Acceleration Structure Details

• Prebuild info
  • Query driver for allocation requirements
  • Returns conservative result and scratch sizes required

• Postbuild info available
  • Query compacted size for reallocation

• Updates supported
  • Incrementally update top/bottom level structs
  • Can do async full rebuild when drifting too far
Raytracing Requirements

- Scene geometry representation
- Trace rays into scene and get intersections
- Determine and execute material shaders
Raytracing Requirements

- Scene geometry representation
- Trace rays into scene and get intersections
- Determine and execute material shaders
Ray Generation Shader

- Invoked via CommandList::DispatchRays()
  - Specify 2D grid of threads
- Emit any number of rays per thread
  - Use TraceRay intrinsic
- Write traversal results to UAVs
TraceRay Intrinsic

- Origin
- Direction
- $T_{\text{Min}}/T_{\text{Max}}$
- App-defined "payload"
Determining Intersections

• Triangle Geometry
  • Determination: Automatic
  • Attributes: Barycentrics

• Programmable Geometry
  • Determination: Intersection Shaders
  • Attributes: Application-defined
• Everything new requires the DXIL compiler
  ✔ dxc.exe / dxcompiler.dll
  ❌ fxc.exe / d3dcompiler_47.dll
  • Get it here: http://aka.ms/HLSL
    • For DXR, use the pre-built binary in the experimental SDK
Raytracing Requirements

- Scene geometry representation
- Trace rays into scene and get intersections
- Determine and execute material shaders
Raytracing Requirements

• Scene geometry representation
• Trace rays into scene and get intersections
• **Determine and execute material shaders**
Hit Shaders

• Invoked at geometry intersection points
• Access to intersection attributes (tri: barycentrics)
• Read/write access to app-defined ray payload
• Call TraceRay() for recursive traversal
Any-Hit Shaders

- Invoked for all intersections along ray path
- Read attributes, modify ray payload for subsequent hit shaders
- May call IgnoreHit() / AcceptHitAndEndSearch()
Closest-Hit Shaders

- Invoked for closest accepted intersection along ray path
- Read attributes, modify ray payload for TraceRay() caller
Closest-Hit Shaders

- Invoked for closest accepted intersections along ray path
- Read attributes, modify ray payload for TraceRay() caller
- Trace more rays
Miss Shader

- Invoked for rays with no accepted hits through TMax
- May trace more rays (e.g. into lower-LOD acceleration structure)
- Return transparent-black, sample skybox, etc.
Which Shaders to Run?

- Rays can intersect any geometry, need any shader, any resource
- ✔ Bindless resources (arbitrarily indexable table)
- ✨ Bindless shaders
Which Shaders to Run?

- Rays can intersect any geometry, need any shader, any resource
- Bindless resources (arbitrarily indexable table)
- Bindless shaders: shader tables
Shader Tables

- GPU buffer of "shader records"
  - Shader ID
  - Root arguments
- Flexible indexing in DXR
  - Instance properties
  - DispatchRays arguments
  - TraceRay arguments
- Shader IDs acquired from "state objects"
State Objects (PSOs v2)

• Set of shaders and root signatures
• Associate root signatures with DXIL library exports
• Create pipeline-specific sub-objects and associations
• Flexibility to support future pipelines

• **State Object Properties** interface for post-compile information
Raytracing State Objects

• Configure maximum ray recursion depth
• Configure ray payload and attribute size
• Create "hit groups" from individual shaders
  • 0/1 intersection shader
  • 0/1 any-hit shader
  • 0/1 closest-hit shader
• Use state object properties interface to get:
  • Ray generation shader IDs
  • Miss shader IDs
  • Hit group IDs
Raytracing Requirements

- Scene geometry representation
- Trace rays into scene and get intersections
- Determine and execute material shaders
Putting it All Together

- Create state objects with set of potential material shaders
- Create top/bottom level acceleration structures
- Create shader tables with hit groups / root parameters
- Call **DispatchRays**
  - Invoke ray-generation shader, call **TraceRay()**
  - Execute hit shaders, write results into a UAV
- Incorporate UAV results into final scene render
Tools

• PIX support available now
  • See also: Direct3D Graphics Debugging and Optimization
  • Thursday 12:45 PM, Room 2009, West Hall (this room)

• Fallback layer
  • Open-source reference implementation
  • Compute shader based (requires DXIL support)

• VS/PS → Hit Group conversion
  • Reuse existing shader content

• Raytracing helper header
  • Very useful for building state objects
DirectX: Evolving Microsoft's Graphics Platform

Johan Andersson & Colin Barré-Brisebois
Electronic Arts
“PICA PICA”

Exploratory mini-game & world

- For our self-learning AI agents to play, not for humans 😊
- Uses SEED’s *Halcyon* R&D engine
- Goals
  - Explore hybrid raytracing with DXR
  - Clean and consistent visuals
  - Procedurally-generated worlds
  - No precomputation
Why raytracing?

- Flexible new tool in the toolbox
- Solve sparse & incoherent problems
- Unified API + performance (DXR + RTX)
- Simple high quality - easy ground truth
Hybrid Rendering Pipeline

Deferred shading (raster)

Direct shadows (raytrace or raster)

Direct lighting (compute)

Reflections (raytrace)

Global Illumination (raytrace)

Ambient occlusion (raytrace or compute)

Transparency & Translucency (raytrace)

Post processing (compute)
Live demo
- **Spawn a Mesh?**
  - DXR: build its bottom acceleration structure
  - Multiple geometries for multiple materials
    - Triangles, AABBs, custom
  - Mesh instances specified in top acceleration

- **Move a Mesh?**
  - Update the instance’s position/orientation in the top acceleration

- **Spawn [some] Rays?**
  - Multiple Hit and Miss shaders possible
Raytraced Reflections

- Rasterize primary visibility
- Launch rays from the G-Buffer
- Raytrace at half resolution
- Reconstruct at full resolution
  - Spatiotemporal filtering
- Works on both flat and curved surfaces
Reflection Rays

Let’s launch some reflection rays:

1. Select one of the (2x2) pixels to trace
2. Reconstruct position and vectors
3. Initialize Halton & random number seq.
4. Initialize the payload
5. Prepare a new ray
6. Trace
7. Gather results from ray payload
   - Reflection Color, Direction, HitT, 1/pdf

Reflections Raytracing HLSL Pseudo-Code

```cpp
RaytracingAccelerationStructure g_rtScene : register<t0, space0>
ConstantBuffer(RaytracingConstants) g_rt : register<t0, space0>
RTexture2D<float4> g_text0 : register<00, space0>
RTexture2D<float4> g_text1 : register<01, space0>

[shader("raygeneration")]
void reflectionRaygen()
{
    uint2 px = DispatchRaysIndex();
    uint2 launchDim = DispatchRaysDimensions();

    // Select one of the four full-res pixels to trace from
    const uint2 noiseCoord = (px / 2 + (g_rt.frameIndex / 2) * uint2(3, 7)) & 31;
    const uint subpixIndex = g_BlueNoise(noiseCoord.x, noiseCoord.y) & 32 + g_rt.frameIndex;
    const uint2 gbufferPx = px + 2 = uint2(subpixIndex & 1, (subpixIndex >> 1) & 1);

    // Reconstruct position and the various vectors
    const float depth = g_depth[gbufferPx];
    float2 uvPos = (gbufferPx / 0.5f / g_rt.viewDimensions.x, y);
    float4 csPos = float4(uvPos, depth, 1.0f);
    float4 vsPos = mul(g_rt.clipToWorld, csPos);
    const float3 P = vsPos.xyz / vsPos.w;
    const float3 E = g_r.eyeWorldPosition;
    const float3 V = normalize(E - position);
    const float3 N = g_buffer.normal;

    // Initialize the Halton sequence for each ray and random number generator to rotate the sequence
    uint seed = randomInit(px);

    // Initialize a new payload
    Payload payload = payloadInit(seed, halton);

    // Initialize a new ray
    RayDesc ray;
    ray.origin = position + r * max(1, length(frame.position)) * 1e-4f; // epsilon
    ray.direction = sampleDirectionFromPdf((gbuffer, V, N, halton, seed)); // stochastic BRDF
    ray.Min = 0.5;
    ray.Max = 1000; // perf: ray stops at 1000 meters

    // Launch the ray
    TraceRay(g_rtScene, RAY_FLAG_CULL_FRONT_FACING_TRIANGLES, RaytracingInstanceMaskAll,
             HitType_Primary, SbtRecordStride, MissType_Primary,
             ray, hitData);
    g_text0[px] = float4(hitData.lighting.rgb, depth);
    g_text1[px] = float4(ray.Direction, hitData.hit, 1.0f / brdfSample.pdf);
}````
Reflection Filtering

Inspired by *Stochastic Screen-Space Reflections* [Stachowiak 2015]

- For every full-res pixel, sample 16 pixels in half-res ray results
  - Blue Noise offsets, decorrelated every 2x2 pixels

- Build color bounding box of ray-hit results
  - Clamp temporal history to bounding box

- Followed by a variance-driven bilateral filter
  - Helps with rough reflections

Unfiltered (Top) and Filtered (Bottom) Results
Ambient Occlusion (AO) [Langer 1994] [Miller 1994] maps and scales directly with real-time ray tracing:

- Integral of the visibility function over the hemisphere $\Omega$ for the point $\vec{p}$ on a surface with normal $\hat{n}$ with respect to the projected solid angle.

- Games often approximate this in screen-space.

- With RT, more grounded & improves visual fidelity!
  - Random directions $\hat{\omega}$
  - Can be temporally accumulated or denoised.

$$A_{\vec{p}} = \frac{1}{\pi} \int_{\Omega} V_{\vec{p}, \hat{\omega}} (\hat{n} \cdot \hat{\omega}) d\omega$$
Raytraced AO
(Same Radius as SSAO)
mGPU

Explicit Heterogenous Multi-GPU

- Parallel Fork-Join Style
- Resources copied through system memory using copy queue
- Minimize PCI-E transfers
- Approach
  - Run ray generation on primary GPU
  - Copy results in sub-regions to other GPUs
  - Run tracing phases on separate GPUs
  - Copy tracing results back to primary GPU
  - Run filtering on primary GPU
Summary

- Just the beginning – important new tool going forward
- Unified API – easy to experiment and integrate
- Flexible but complex tradeoffs - noise vs ghosting vs perf
- Can enable very high quality cinematic visuals
- Lots more to explore – perf, raster vs trace, sparse render, denoising, new techniques
SEED @ GDC 2018

- **Shiny Pixels & Beyond: Rendering Research at SEED (presented by Nvidia)**
  - Johan Andersson and Colin Barré-Brisebois
  - Room 3022, West Hall, Wednesday, March 21st, 5:00pm - 6:00pm

- **Deep Learning - Beyond the Hype**
  - Magnus Nordin
  - Room 2016, West Hall, Thursday, March 22nd, 11:30am - 12:30pm

- **Creativity of Rules and Patterns: Designing Procedural Systems**
  - Anastasia Opara
  - GDC Show Floor, Thursday, March 22nd, 12:30PM-1:00PM and Friday, March 23rd @ 11:00AM-11:30AM
Thanks

- **SEED**
  - Jasper Bekkers
  - Joakim Bergdahl
  - Ken Brown
  - Dean Calver
  - Dirk de la Hunt
  - Jenna Frisk
  - Paul Greveson
  - Henrik Halen
  - Effeli Holst
  - Andrew Lauritzen
  - Magnus Nordin
  - Niklas Nummelin
  - Anastasia Opara
  - Kristoffer Sjöö
  - Tomasz Stachowiak
  - Ida Winterhaven
  - Graham Wihlidal

- **Microsoft**
  - Chas Boyd
  - Ivan Nevraev
  - Amar Patel
  - Matt Sandy

- **NVIDIA**
  - Tomas Akenine-Möller
  - Nir Benty
  - Jiho Choi
  - Peter Harrison
  - Alex Hyder
  - Jon Jansen
  - Aaron Lefohn
  - Ignacio Llamas
  - Henry Moreton
  - Martin Stich
SEED // SEARCH FOR EXTRAORDINARY EXPERIENCES DIVISION

Stockholm – Los Angeles – Montréal – Remote
www.ea.com/seed

We’re Hiring!
How to get started

• Windows Insider Preview (RS4)
• Experimental SDK + spec: http://aka.ms/DXRSDK
• PIX-raytracing: http://aka.ms/DXRPIX
• DXR overview: http://aka.ms/DXR
• Give us feedback (really!): http://forums.directxtech.com
Ray Tracing Gems – Call for Papers

• A new book series with focus on real-time and interactive ray tracing for game development using the DXR API.

• We invite articles on the following topics:
  Basic ray tracing algorithms, effects (shadows, reflections, etc.), non-graphics applications, reconstruction, denoising, & filtering, efficiency and best practices, baking & preprocessing, ray tracing API & design, rasterization and ray tracing, global Illumination, BRDFs, VR, deep learning, etc.

• Important dates:
  • 15th of October 2018: submission deadline for full papers
  • GDC 2019: publication of Ray Tracing Gems (paper version + e-book)
  • Eric Haines and Tomas Akenine-Möller will lead the editorial team
  http://developer.nvidia.com/raytracinggems/
Questions?