"IT JUST WORKS": RAY-TRACED REFLECTIONS IN 'BATTLEFIELD V'

Johannes Deligiannis
Jan Schmid
EA DICE

GAME DEVELOPERS CONFERENCE
MARCH 18–22, 2019 | #GDC19
TODAY WE PRESENT

RAYTRACING

• Project background
• GPU Raytracing Pipeline
• Engine integration of DXR
• GPU Performance
BATTLEFIELD V

- FPS set in WWII
- Released Nov 2018
- Raytracing work began Dec 2017
- First DXR game released!
PROJECT BACKGROUND

• ~10 months dev time
• Use DXR in Battlefield V
  • AO
  • GI
  • Shadows
  • Reflections

• Engineering
  • Yasin Uludag (EA DICE)
  • Johannes Deligiannis (EA DICE)
  • Jiho Choi (NVIDIA)
  • Pawel Kozlowski (NVIDIA)
  • And a bunch of other people! 😊
MAIN CHALLANGES

- Not a Tech Demo
  - Content is set
  - Game in full production
  - Scope of Engine changes
  - Performance
  - Denoising vs Ray Count
  - No RTX cards

- Early adopter tax
  - API not final
  - Driver hang/bugs
  - BSoD
  - No capture tool (Nsight, Pix)

- But we shipped it😊
(SIMPLE) RAYTRACING PIPELINE

Generate Rays → Intersect/Material Data → Light Rays → Light Combine
GENERATE RAYS

*Tomasz Stachowiak and Yasin Uludag, Siggraph 2015. “Stochastic Screen-Space Reflections”
RAYTRACING

MAGIC
float4 light(MaterialData surfaceInfo, float3 rayDir) {
    foreach (light : pointLights)
        radiance += calcPoint(surfaceInfo, rayDir, light);

    foreach (light : spotLights)
        radiance += calcSpot(surfaceInfo, rayDir, light);

    foreach (light : reflectionVolumes)
        radiance += calcRef1Vol(surfaceInfo, rayDir, light);

    ...
}

LIGHT RAYS
LIGHT COMBINE

Lookup Texture

Lit Raster result
IMPROVING RAYTRACING PIPELINE

Variable Rate Tracing

Generate Rays → Intersect/Material Data → Light Rays → Light Combine
VARIABLE RATE TRACING
VARIABLE RATE TRACING

256 rays

128 rays

64 rays

32 rays
VARIABLE RATE TRACING

Success!

- More Rays on Water
- More Rays on grazing angles
PROBLEM
IMPROVING RAYTRACING PIPELINE

Variable Rate Tracing

Generate Rays → Intersect/Material Data → Light Rays → Light Combine

Ray Binning
RAY BINNING

Screen Offset

3012 Bin Index

Angle
RAY BINNING
RAY BINNING

- Rays: Ray 1000, Ray 1001, Ray 1002
- Bins: Bin 3011, Bin 3012, Bin 3013
- Local Offsets: 0, 0, 1

Atomic Increment:

- Bin 3011: Atomic Increment 2
  - Local Offset: 0
- Bin 3012: Atomic Increment 0
  - Local Offset: 0
- Bin 3013: Atomic Increment 1
  - Local Offset: 1
*Mark Harris, Shubhabrata Sengupta, and John Owens. “Parallel Prefix Sum (Scan) with CUDA”
PROBLEM
IMPROVING RAYTRACING PIPELINE

- Variable Rate Tracing
- SSR Hybridization

Generate Rays → Intersect/Material Data → Light Rays → Light Combine

Ray Binning
[Stachowiak et al 15] "Stochastic Screen-Space Reflections"
SS-HYBRIDIZATION
SS-HYBRIDIZATION
**PROBLEM**

<table>
<thead>
<tr>
<th>Raytrace</th>
<th>Light Shader Wavefront</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hit</td>
<td>Miss</td>
</tr>
<tr>
<td>Miss</td>
<td>Hit</td>
</tr>
<tr>
<td>Miss</td>
<td>Hit</td>
</tr>
<tr>
<td>Miss</td>
<td>Miss</td>
</tr>
<tr>
<td>Hit</td>
<td>Hit</td>
</tr>
<tr>
<td>Hit</td>
<td>Miss</td>
</tr>
<tr>
<td>Miss</td>
<td>Idle</td>
</tr>
<tr>
<td>Busy</td>
<td>Idle</td>
</tr>
<tr>
<td>Busy</td>
<td>Busy</td>
</tr>
<tr>
<td>Busy</td>
<td>Busy</td>
</tr>
<tr>
<td>Idle</td>
<td>Busy</td>
</tr>
<tr>
<td>Busy</td>
<td>Busy</td>
</tr>
<tr>
<td>Busy</td>
<td>Busy</td>
</tr>
</tbody>
</table>
IMPROVING RAYTRACING PIPELINE

Variable Rate Tracing

SSR Hybridization

Generate Rays

Intersect/Material Data

Light Rays

Light Combine

Ray Binning

Defrag
**DEFFRAG**

Exclusive Parallel Sum *

*Mark Harris, Shubhabrata Sengupta, and John Owens. “Parallel Prefix Sum (Scan) with CUDA”*
## Problem

### Light Shader

<table>
<thead>
<tr>
<th>Busy</th>
<th>Busy</th>
<th>Busy</th>
<th>Busy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Busy</td>
<td>Busy</td>
<td>Busy</td>
<td>Busy</td>
</tr>
<tr>
<td>Busy</td>
<td>Busy</td>
<td>Busy</td>
<td>Busy</td>
</tr>
<tr>
<td>Busy</td>
<td>Busy</td>
<td>Busy</td>
<td>Busy</td>
</tr>
</tbody>
</table>

2.0ms
IMPROVING RAYTRACING PIPELINE

- Variable Rate Tracing
- SSR Hybridization
- Generate Rays
- Intersect/Material Data
- Light Rays
- Light Combine
- Ray Binning
- Defrag
PER CELL LIGHT LISTS
PROBLEM
IMPROVING RAYTRACING PIPELINE

Variable Rate Tracing
SSR Hybridization

Generate Rays
Intersect/Material Data
Per Cell Light List Lighting
Light Combine

Ray Binning
Defrag
Denoise
DENOISING

- Reuse Spatial Information
- Reuse Temporal Information

[Stachowiak et al 15] "Stochastic Screen-Space Reflections"

BRDF Filter

Temporal Filter
BRDF DENOISE FILTER

$$L_0 \approx FG \frac{\sum_{k=1}^{N} L_i(l_k) f_s(l_k \rightarrow v) \cos \Theta_{l_k}}{p_k}$$

$$\frac{\sum_{k=1}^{N} f_s(l_k \rightarrow v) \cos \Theta_{l_k}}{p_k}$$
BRDF DENOISE FILTER
BRDF DENOISE FILTER
BRDF DENOISE FILTER

Frame N

Frame N -1
### BRDF Denoise Filter

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Pad</td>
<td>Pad</td>
<td>Pad</td>
<td>Pad</td>
<td>Pad</td>
<td>Pad</td>
<td>Pad</td>
<td>Pad</td>
</tr>
<tr>
<td>Pad</td>
<td>Pad</td>
<td>Pad</td>
<td>Pad</td>
<td>Pad</td>
<td>Pad</td>
<td>Pad</td>
<td>Pad</td>
</tr>
<tr>
<td>Pad</td>
<td>Pad</td>
<td>Thread</td>
<td>Thread</td>
<td>Thread</td>
<td>Thread</td>
<td>Pad</td>
<td>Pad</td>
</tr>
<tr>
<td>Pad</td>
<td>Pad</td>
<td>Thread</td>
<td>Thread</td>
<td>Thread</td>
<td>Thread</td>
<td>Pad</td>
<td>Pad</td>
</tr>
<tr>
<td>Pad</td>
<td>Pad</td>
<td>Thread</td>
<td>Thread</td>
<td>Thread</td>
<td>Thread</td>
<td>Pad</td>
<td>Pad</td>
</tr>
<tr>
<td>Pad</td>
<td>Pad</td>
<td>Thread</td>
<td>Thread</td>
<td>Thread</td>
<td>Thread</td>
<td>Pad</td>
<td>Pad</td>
</tr>
<tr>
<td>Pad</td>
<td>Pad</td>
<td>Pad</td>
<td>Pad</td>
<td>Pad</td>
<td>Pad</td>
<td>Pad</td>
<td>Pad</td>
</tr>
<tr>
<td>Pad</td>
<td>Pad</td>
<td>Pad</td>
<td>Pad</td>
<td>Pad</td>
<td>Pad</td>
<td>Pad</td>
<td>Pad</td>
</tr>
</tbody>
</table>

**Actual:** 6

**Actual:** 16

**Actual:** up to 13
Is it a good sample?
If only...
BRDF Denoiser!
TEMPORAL DENOISE FILTER

Still Noisy
Generate LUT
\{ \text{angle, roughness} \}

to
\{ \text{width, height} \}
for unit length ray
IMAGE DENOISE FILTER

* =
IMAGE DENOISE FILTER

\[ \frac{1}{\sqrt{2}} \ast \frac{1}{\sqrt{2}} = \]
IMAGE DENOISE FILTER
NEW PIPELINE

Variable Rate Tracing: 0.37ms
Generate Rays: 0.19ms
Ray Binning: 0.15ms
Screen Space Hybrid: 0.36ms
Intersect/Material Data: 1.98ms
NEW PIPELINE

6.29ms total

Intersect/Material Data 1.98ms
Defrag 0.08ms
‘Improved’ Lighting 0.46ms
Spatial Filter 1.45ms
Temporal Filter 0.24ms
Image Filter 1.00ms
DXR – a.k.a "BLACK BOX"
• **BLAS** - Bottom Level Acceleration Structure

• **TLAS** - Top Level Acceleration Structure

• **CS**
  • Skinning, Destruction
  • Compute shader
  • Update each frame
  • Blas can update incrementally
ACCELERATION STRUCTURE

- Which objects?
- Frustum Culling
- Occlusion Culling
- Easy... no culling!
ACCELERATION STRUCTURE – FIRST PASS

- Rotterdam
- 20200 TLAS instances...
- 5000 BLAS rebuilds...
- GPU rebuild 64 ms (!)
WHAT TO DO?

• Idea: Reduce instance count
• Use a culling heuristic
• Accept (some) minor artifacts
CULLING HEURISTIC

• Assumption:
  • Far away objects not important
    • Except for large objects
    • Bridge, building etc
  • Need some kind of measurement...
• Project bounding sphere

\[ \theta = \tan \frac{r}{d} \]

• If \( \theta^\circ < \text{Threshold}^\circ \): Cull
CULLING

reference — no culling

$\theta = 4^\circ$

$\theta = 15^\circ$
Culling

\[ \theta = 4^\circ \]

Culled Objects

Reference – no culling

\[ \text{Culled Objects} \]
CULLING - RESULTS

• 4 deg culling
• 5000 -> 400 BLAS rebuilds each frame
• 20000 -> 2800 TLAS instances
• TLAS + BLAS build (GPU): 64 ms -> 14.5 ms

• Pros
  • Faster

• Cons
  • Occasional popping
  • Missing objects
BLAS UPDATE
OPTIMIZATIONS

• Still expensive! More ideas:

1. **Stagger** full and incremental BLAS rebuild
   • N frames incremental before full rebuild

2. **D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_PREFER_FAST_BUILD**

3. Avoid **redundant** rebuilds
   • Check CS input (bone matrix)
   • 400 -> 50
   • **Overlap** BLAS update with GFX
     • Gbuffer, shadowmaps
RESULTS

• TLAS + BLAS build (GPU): 14.5 ms → 1.15 ms

• RayGen (GPU): 0.71 ms → 0.81 ms (staggered refit + flags)

• Much better 😊
SHADING (OPAQUE)
RAYTRACING REQUIREMENTS

• Shader output must match!
• ClosestHit Shader
• AnyHit Shader
SHADERS IN FROSTBITE

- VS – Handwritten
- PS – Shader Graphs
  - Graph -> .hls
- Manual conversion... no
  - 1000s of shaders
- Auto VS + PS to HitGroup
# Hit Shader Template

**VS - VertexFragment**
- World Space Normal

**PS -**
- `#define Sample(s, uv)`
- `#define ddx(x) x`
- `#define ddy(x) x`

```
void chMain()
{
    IA0 = iaMain(id + 0)
    IA1 = iaMain(id + 1)
    IA2 = iaMain(id + 2)
    V0 = vsMain(IA0)
    V1 = vsMain(IA1)
    V2 = vsMain(IA2)
    V = lerp(V0, V1, V2, U, V, W)
    P = psMain(V)
    writePayload(P)
}
```

**Vertex buffer UV, Normal**
• **AnyHit Shader:**

• If (AlphaTest(alphaValue))

  ```cpp
  IgnoreHit();
  ```
ALPHA TEST

ANY HIT OFF

ANY HIT ON
ALPHA TEST
Summary Opaque

- **Closest Hit** Shader
  - Always

- **Any Hit** Shader (Optional)
  - Alpha tested

- **Compute** Shader (Optional)
  - Skinning, destruction etc
RAY PAYLOAD

- Payload: returned on ray intersection
- Same format as Gbuffer RTV
- Contains Material Data
  - Normal
  - Base Color
  - Smoothness
  - ...etc

```c
struct GbufferPayloadPacked
{
    uint data0;  // R10G10B10A2_UNORM
    uint data1;  // R8G8B8A8_SRGB
    uint data2;  // R8G8B8A8_UNORM
    uint data3;  // R11G11B10_FLOAT
    float hitT;  // Ray length
};
```
VERIFYING CORRECTNESS

1. Rasterizer output
2. Shoot primary rays into scene
3. Compare Payload with Gbuffer
4. Non-zero output? Bug!
5. Fix bug
SHADER COMPILATION

• All shaders generated 😊
• ~3000 per level
• ~250 per frame
• Single RT PSO
• Runtime compile times?

Color coded Closest Hit Shaders
• Dx12 GFX PSO...
• ... DXR 3000 shaders ☺
• Compile times?
  • Majority > 100 ms
• Cold cache
  • 7 min 30 sec thread time
  • 6 threads: 1 min 30 sec
• Warm cache
  • 1 min 30 sec thread time
  • 6 threads: 15 sec

PSO GENERATION

Compile time histogram
Smoke, Fire and Explosions. Important elements in BFV!
PARTICLES

• Particle = Transparent+Billboard

• Basic algorithm

1. Shoot ray in Opaque TLAS
2. Shoot again in Particle TLAS
   (Max ray length from Opaque)
3. Blend particles with opaque hit
THE PROBLEM WITH PARTICLES

• Camera aligned billboards 😊
• Rotate odd particles 90 deg around Y?

Billboards visible when viewed from the side.
Before: Billboards visible in reflection

After: Rotating odd quads produces a more volumetric look
Accumulate intersections along ray

1 rpp => N rpp 😊

RayGen loop

 Sounds... expensive?

RayGen Shader

```cpp
... init ray using opaqueHitT and currT*
for (hitCount = 0; hitCount < MaxIntersectionCount; ++hitCount)
{
...
ForwardPayloadPacked forwardPayloadPacked;
initForwardPayloadPacked(forwardPayloadPacked);
TraceRay(g_tlasParticles, 0, 0xFF, 0, 1, 0, ray, forwardPayloadPacked);
ForwardPayload forwardPayload = unpackForwardPayload(forwardPayloadPacked);
if (forwardPayload.hitT <= 0.0f) // Miss, tracing done
    break;

* ... update ray using forwardPayload.hitT, accumulate color, alpha *
```
THE (SECOND) PROBLEM WITH PARTICLES

RayGen loop \(0.96\text{ms}\)
Optimizing Particles

- Loop Idea: AnyHit shader?
- Same... but different
- Inspired by WBOIT*
  - **Weight = max(luminance, r, g, b, alpha)**
  - Emissive, fire

---

**RayGen Shader**

```shadertalk
... init ray using maxT and currT
TraceRay(g_tlasPartices, 0, 0xFF, 0, 1, 0, ray, forwardPayloadPacked);
* ... process payload and calculate weighted average *
```

**Any Hit Shader**

```shadertalk
struct Attributes { float2 barycentrics; };
[shader("anyhit")]
void main(inout ForwardPayloadPacked payloadPacked, in Attributes attributes)
{
  /* ... Calculate color, transparency */
  payloadPacked.alpha += alpha * weight
  payloadPacked.color += color * weight;
  payloadPacked.weight += weight;
  IgnoreHit();
}
```

---

‘Naive’ Closest Hit **0.96ms**
Slow but accurate

Order Independent AnyHit **0.34ms**
Really fast, but slightly different look
THANK YOU!

...ANY QUESTIONS?