DirectX 12 Optimization Techniques in Capcom’s RE ENGINE

Ojiro Tanaka
Rendering Engineer
Capcom

Ashley Smith
Developer Technology Engineer
AMD
Agenda

- Tools
  - RGP
  - RGA
  - Tips

- Optimizations
  - Optimization methods
  - Optimizations for DirectX 12

- Tips
  - Pre-bake PSO
  - QA
RGP

- Overview pages
- Pipeline state
- Context rolls
- Barriers
Let's investigate this draw.
RGP – Pipeline State

Be careful of scratch memory!

2 out of 10 wavefronts

Reduce VGPR by 1: 3 out of 10 wavefronts
RGP – Pipeline State

• Reducing register usage
  • min16float
  • min16int
  • min16uint
• No need to check for support
• Will default to lowest precision
• How do we investigate?
  • RGA
RGA

struct PSInput {
    float4 color : COLOR;
};

float4 PSMain(PSInput input) : SV_TARGET {
    return float4(pow(abs(input.color.rgb), 2.2), input.color.a);
}
<table>
<thead>
<tr>
<th>Line</th>
<th>Ra</th>
<th>Reg State</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>::</td>
<td>label_basic_block_1: s_mov_b32 m0, s2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>::</td>
<td>s_nop 0x0000</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>v:^</td>
<td>v_interp_p1_f32 v2, v0, attr0.x</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>v:^</td>
<td>v_interp_p2_f32 v2, v1, attr0.x</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>v::^</td>
<td>v_interp_p1_f32 v3, v0, attr0.y</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>v::^</td>
<td>v_interp_p2_f32 v3, v1, attr0.y</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>v:::</td>
<td>v_interp_p1_f32 v4, v0, attr0.z</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>v:::</td>
<td>v_interp_p2_f32 v4, v1, attr0.z</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>:::::</td>
<td>v_log_f32 v2, abs(v2)</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>:::::</td>
<td>v_log_f32 v3, abs(v3)</td>
</tr>
<tr>
<td>11</td>
<td>5</td>
<td>:::::</td>
<td>v_log_f32 v4, abs(v4)</td>
</tr>
<tr>
<td>12</td>
<td>5</td>
<td>:::::</td>
<td>v_mul_f32 v2, 0x400ccccd, v2</td>
</tr>
<tr>
<td>13</td>
<td>5</td>
<td>:::::</td>
<td>v_mul_f32 v3, 0x400ccccd, v3</td>
</tr>
<tr>
<td>14</td>
<td>5</td>
<td>:::::</td>
<td>v_mul_f32 v4, 0x400ccccd, v4</td>
</tr>
<tr>
<td>15</td>
<td>5</td>
<td>:::::</td>
<td>v_exp_f32 v2, v2</td>
</tr>
<tr>
<td>16</td>
<td>5</td>
<td>:::::</td>
<td>v_exp_f32 v3, v3</td>
</tr>
<tr>
<td>17</td>
<td>5</td>
<td>:::::</td>
<td>v_exp_f32 v4, v4</td>
</tr>
<tr>
<td>18</td>
<td>5</td>
<td>:::::</td>
<td>v_interp_p1_f32 v0, v0, attr0.w</td>
</tr>
<tr>
<td>19</td>
<td>5</td>
<td>:::::</td>
<td>v_interp_p2_f32 v0, v1, attr0.w</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>:::::</td>
<td>v_cvt_pkrtz_f16_f32 v1, v2, v3</td>
</tr>
<tr>
<td>21</td>
<td>3</td>
<td>:::::</td>
<td>v_cvt_pkrtz_f16_f32 v0, v4, v0</td>
</tr>
<tr>
<td>22</td>
<td>2</td>
<td>vv</td>
<td>exp mrt0, v1, v1, v0, v0</td>
</tr>
<tr>
<td>23</td>
<td>0</td>
<td></td>
<td>s_endpgm</td>
</tr>
</tbody>
</table>

Maximum # VGPR used 5, # VGPR allocated: 5
struct PSInput {
    float4 color : COLOR;
};

float4 PSMain(PSInput input) : SV_TARGET {
    return float4(pow(abs(input.color.rgb), 2.2), input.color.a);
}

<table>
<thead>
<tr>
<th>Line</th>
<th>Ra</th>
<th>Reg State</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>::</td>
<td>label_basic_block_1: s_mov_b32 m0, s2</td>
</tr>
<tr>
<td></td>
<td>23</td>
<td>0</td>
<td>s_endpgm</td>
</tr>
</tbody>
</table>

Maximum # VGPR used 5, # VGPR allocated: 5
struct PSInput {
    uint16 float4 color : COLOR;
};

float4 PSMain(PSInput input) : SV_Target {
    return float4(pow(abs(input.color.rgb), 2.2), input.color.a);
}

<table>
<thead>
<tr>
<th>Line</th>
<th>Ra</th>
<th>Reg State</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>::</td>
<td>label_basic_block_1: s_mov_b32 m0, s2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>// ...</td>
</tr>
<tr>
<td>24</td>
<td>4</td>
<td>:^v:</td>
<td>v_cvt_f32_f16 v1, v2</td>
</tr>
<tr>
<td>25</td>
<td>4</td>
<td>::^v</td>
<td>v_cvt_f32_f16 v2, v3</td>
</tr>
<tr>
<td>26</td>
<td>4</td>
<td>:::x</td>
<td>v_cvt_f32_f16 v3, v3</td>
</tr>
<tr>
<td>27</td>
<td>4</td>
<td>x:::</td>
<td>v_cvt_f32_f16 v0, v0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>// ...</td>
</tr>
<tr>
<td>33</td>
<td>0</td>
<td></td>
<td>s_endpgm</td>
</tr>
</tbody>
</table>

Maximum # VGPR used  4, # VGPR allocated:  4 5
struct PSInput {
    min16float4 color : COLOR;
};
float4 PSMain(PSInput input) : SV_TARGET {
    return float4(pow(abs(input.color.rgb), 2.2), input.color.a);
}

Line | Ra | Reg State | Instruction
--------------------------------------------
 1 | 2 | ::        | label_basic_block_1: s_mov_b32 m0, s2
// ...
 24 | 4 | :^v:     | v_cvt_f32_f16 v1, v2
 25 | 4 | ::^v      | v_cvt_f32_f16 v2, v3
 26 | 4 | :::x      | v_cvt_f32_f16 v3, v3
 27 | 4 | x:::      | v_cvt_f32_f16 v0, v0
// ...
 33 | 0 |           | s_endpgm

Maximum # VGPR used  4, # VGPR allocated:  4 5
RGP – Context Rolls

cmdBuf->RSSetViewports(a);
cmdBuf->Draw(1);
cmdBuf->RSSetViewports(b);
cmdBuf->Draw(2);
cmdBuf->RSSetViewports(a);
cmdBuf->Draw(3);
RGP Profiler – Context Rolls

Color by hardware context

Could be running more draws here
RGP – Context Rolls

How do we check context rolls?

- Every draw caused a context roll 😊
- Can see what state change caused context roll

---

<table>
<thead>
<tr>
<th>State which causes context rolls</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
</tr>
<tr>
<td>D3D12_GRAPHICS_PIPELINE_STATE_VS</td>
</tr>
<tr>
<td>D3D12_GRAPHICS_PIPELINE_STATE_PS</td>
</tr>
<tr>
<td>TextKill</td>
</tr>
<tr>
<td>Input Count</td>
</tr>
<tr>
<td>D3D12_GRAPHICS_PIPELINE_STATE.BlendState</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
RGP – Context Rolls

```c
cmdBuf->RSSetViewports(a);
cmdBuf->Draw(1);
cmdBuf->RSSetViewports(b);
cmdBuf->Draw(2);
cmdBuf->RSSetViewports(a);
cmdBuf->Draw(3);
```

```c
(cmdBuf->RSSetViewports(a);
cmdBuf->Draw(1);
(cmdBuf->Draw(3);
(cmdBuf->RSSetViewports(b);
cmdBuf->Draw(2);
```
RGP – Barriers

6% of time spent in barriers

List of barriers
## RGP – Barriers

<table>
<thead>
<tr>
<th>Event Numbers</th>
<th>Duration</th>
<th>Drain Time</th>
<th>Stalls</th>
<th>Depth/Stencil Decompress</th>
<th>HIZ Range</th>
<th>DCC Decompress</th>
<th>FMask Decompress</th>
<th>Fast Clear Eliminates</th>
<th>Ini Mask RAM</th>
<th>Invalidated</th>
<th>Flushed</th>
<th>Barrier type</th>
<th>Reason for Barriers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.001 ms</td>
<td>0.004 ms</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>DRIVER</td>
<td>DRIVER</td>
<td>Before CS</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.002 ms</td>
<td>0.004 ms</td>
<td>CS</td>
<td>CS</td>
<td>CS</td>
<td>CS</td>
<td>CS</td>
<td>CS</td>
<td>CS</td>
<td>DRIVER</td>
<td>DRIVER</td>
<td>After CS</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.002 ms</td>
<td>0.002 ms</td>
<td>DSMR</td>
<td>DSMR</td>
<td>DSMR</td>
<td>DSMR</td>
<td>DSMR</td>
<td>DSMR</td>
<td>DSMR</td>
<td>DSMR</td>
<td>DSMR</td>
<td>DSMR</td>
<td></td>
</tr>
<tr>
<td>5..6</td>
<td>0.011 ms</td>
<td>0.002 ms</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>DRIVER</td>
<td>DRIVER</td>
<td>Before CS</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0.001 ms</td>
<td>0.001 ms</td>
<td>CS</td>
<td>CS</td>
<td>CS</td>
<td>CS</td>
<td>CS</td>
<td>CS</td>
<td>CS</td>
<td>DRIVER</td>
<td>DRIVER</td>
<td>After CS</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0.001 ms</td>
<td>0.001 ms</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>DRIVER</td>
<td>DRIVER</td>
<td>Before CS</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0.002 ms</td>
<td>0.001 ms</td>
<td>CS</td>
<td>CS</td>
<td>CS</td>
<td>CS</td>
<td>CS</td>
<td>CS</td>
<td>CS</td>
<td>DRIVER</td>
<td>DRIVER</td>
<td>After CS</td>
<td></td>
</tr>
<tr>
<td>19..20</td>
<td>0.009 ms</td>
<td>0.008 ms</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>DRIVER</td>
<td>DRIVER</td>
<td>Before CS</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>0.002 ms</td>
<td>0.008 ms</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>DRIVER</td>
<td>DRIVER</td>
<td>After CS</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>0.001 ms</td>
<td>0.005 ms</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>DRIVER</td>
<td>DRIVER</td>
<td>DSMR</td>
<td></td>
</tr>
<tr>
<td>22..23</td>
<td>0.008 ms</td>
<td>0.008 ms</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>DRIVER</td>
<td>DRIVER</td>
<td>DSMR</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>0.035 ms</td>
<td>0.001 ms</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>DRIVER</td>
<td>DRIVER</td>
<td>DSMR</td>
<td></td>
</tr>
</tbody>
</table>
RGP – Barriers

• Depth/stencil decompress
RGP – Barriers

- HiZ range resummarize
RGP – Barriers

- DCC decompress

Compared to RGP, compression is enabled or disabled by many factors due to various format, flags, and usage scenarios.
RGP – Barriers

• DCC decompress

• Example:

```c
ResourceBarrier(D3D12_RESOURCE_STATE_RENDER_TARGET,
                 D3D12_RESOURCE_STATE_COPY_DEST);
```

```c
ResourceBarrier(D3D12_RESOURCE_STATE_COPY_DEST,
                 D3D12_RESOURCE_STATE_RENDER_TARGET);
```
Tips

- Fast clears
- Debugging
Tips – Fast clears

ClearRenderTargetView()  
ClearDepthStencilView()  
pOptimizedClearValue

Stick to 1.0f or 0.0f for depth  
Black or white for color
Tips – Debugging

Breadcrumbs / WriteBufferImmediate()

WriteMarker(TopOfPipe, 1)
Draw(x)
WriteMarker(BottomOfPipe, 2)
WriteMarker(TopOfPipe, 3)
Draw(y)
WriteMarker(BottomOfPipe, 4)
Tips – Debugging

Breadcrumbs / WriteBufferImmediate()

WriteMarker(TopOfPipe, 1)

Draw(x) < TDR happens here

WriteMarker(BottomOfPipe, 2)

WriteMarker(TopOfPipe, 3)

Draw(y)

WriteMarker(BottomOfPipe, 4)
Tips – Debugging

Breadcrumbs / WriteBufferImmediate()
WriteMarker(TopOfPipe, 1)
Draw(x) < TDR happens here
WriteMarker(BottomOfPipe, 2)
WriteMarker(TopOfPipe, 3)
Draw(y)
WriteMarker(BottomOfPipe, 4)
// ...
< Crash reported afterwards
Tips – Debugging

Breadcrumbs / WriteBufferImmediate()

WriteMarker(TopOfPipe, 1)
Draw(x) < TDR happens here
WriteMarker(BottomOfPipe, 2)
WriteMarker(TopOfPipe, 3)
Draw(y)
WriteMarker(BottomOfPipe, 4)
// ...
< Crash reported afterwards
Tips – Debugging

Breadcrumbs / WriteBufferImmediate()

```c
WriteMarker(TopOfPipe, 1) // 1
Draw(x) < TDR happens here
WriteMarker(BottomOfPipe, 2) // 0
WriteMarker(TopOfPipe, 3) // 0
Draw(y)
WriteMarker(BottomOfPipe, 4) // 0
// ...
< Crash reported afterwards
```

We know what caused the TDR now
Tips – Debugging

Breadcrumbs / WriteBufferImmediate()
DX11: AGS on github

DX12: WriteBufferImmediate()

Only for debugging (May cause stalls!)
RE ENGINE
Optimization
Agenda

• Optimization
  • Adaptation of console optimizations to PC
  • Optimization for DirectX 12

• Tips
Background of in-house engine

- **RE ENGINE**
  - Capcom’s in-house engine
  - Targets consoles and PC

- **Shipped**
  - Resident Evil 7: Biohazard (RE7)
  - Resident Evil 2 (RE2)
  - Devil May Cry 5 (DMC5)
Background of in-house engine

- RE ENGINE uses “Intermediate drawing command”
  - Platform independent commands
  - Allows programmers to write drawing commands without platform knowledge
  - Useful for multi-platform development
  - Able to create drawing commands on multiple threads
  - These “Intermediate drawing commands” are sorted after creation then translated to API commands
  - Drawing order is controlled using priority variable (uint 64 bit value)
  - Allows batch process at the discretion of the user
  - Useful for controlling sync timing of UAVOverlap and AsyncDispatch
Implementation of DirectX 12 in RE ENGINE

• Trials started during RE7 production, but was not implemented

• RE2 and DMC5 implements DirectX 12
Optimization

• Adaptation of console optimizations to PC
  • OcclusionCulling using MultiDraw
  • UAVOverlap
  • Wave Intrinsics
  • Depth Bounds Test

• Optimization for DirectX 12
  • Reduction of resource barrier
  • Buffer update
  • RootSignature
  • Memory management
Comparison of before and after

- 24% frame time saving!
Adaptation of console optimizations to PC
Testing environment

- RE2 (2/15 patch)
- 1080p
- Mainly Radeon RX480, partially Radeon R9 Fury X
- Radeon GPU Profiler 1.3.1.70, OCAT, PIX for Windows
MultiDraw

• In DirectX 12 we use ExecuteIndirect
  • Allows execution of multiple drawing commands at once
  • Aim to reduce the overhead of drawing meshes

• In DirectX 11 MultiDraw is supported by AGS or NVAPI
Any improvements?

- Overhead-wise there was not as much improvement as we had hoped
- ExecuteIndirect was useful for implementation of GPU-based occlusion culling
GPU-based occlusion culling OFF
GPU-based occlusion culling ON
FYI

- 2 possible solution; ExecuteIndirect and Predication command

- ExecuteIndirect
  - 4 byte Alignment
  - Controls the number of IndirectArgument executions with CountBuffer

- Predication command
  - 8 byte Alignment - Incompatible with consoles
Data structures - VisibleBuffer

- Visibility managed using “VisibleBuffer”
  - Practically, it is a CountBuffer in RE ENGINE
  -_byteAddressBuffer
  - Number of elements is equal to maximum number of meshes in scene
  - Each element contains per mesh visibility
    - 0xffff for visible, 0x0000 for invisible
Data structures – Mesh data

- **StructuredBuffer**
  - AABB - CPU made or GPU made
  - VisibleBuffer’s byte offset
  - IndirectArgument’s byte offset
Visibility test

- **Draw with EarlyZ**
  - `[earlydepthstencil]` attribute!
  - Store 0xffff into VisibleBuffer
- **Minimize writing to same address in units of Wave[dorobot16]**

```cpp
[earlydepthstencil]
void PS_Culltest(OccludeeOutput I){
    uint hash = WaveCompactValue(I.outputAddress);
    [branch]
    if (hash == 0){
        RWCountBuffer.Store(I.outputAddress, 0xffff);
    }
}
```
Apply visibility test result

- Apply drawing per mesh
- Specify number of draws using `MaxCommandCount`
  - `VisibleBuffer` as `CountBuffer`
    - `CountBuffer 0xffff`: Enable draw (count is `MaxCommandCount`)
    - `CountBuffer 0`: Disable draw

```c
void ExecuteIndirect(
    ID3D12CommandSignature *pCommandSignature,
    UINT MaxCommandCount,
    ID3D12Resource *pArgumentBuffer,
    UINT64 ArgumentBufferOffset,
    ID3D12Resource *pCountBuffer,
    UINT64 CountBufferOffset);
```
### Result on PIX

<table>
<thead>
<tr>
<th>IDX</th>
<th>Command Description</th>
<th>Visible mesh</th>
<th>Invisible mesh</th>
</tr>
</thead>
<tbody>
<tr>
<td>5522</td>
<td><code>ExecuteIndirect</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5523</td>
<td><code>DrawIndexedInstanced</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5524</td>
<td><code>DrawIndexedInstanced</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5525</td>
<td><code>DrawIndexedInstanced</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5526</td>
<td><code>DrawIndexedInstanced</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5527</td>
<td><code>DrawIndexedInstanced</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5528</td>
<td><code>DrawIndexedInstanced</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5529</td>
<td><code>ExecuteIndirect</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5530</td>
<td><code>ExecuteIndirect</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5531</td>
<td><code>ExecuteIndirect</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5532</td>
<td><code>ExecuteIndirect</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5533</td>
<td><code>ExecuteIndirect</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5534</td>
<td><code>ExecuteIndirect</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5535</td>
<td><code>ExecuteIndirect</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5536</td>
<td><code>ExecuteIndirect</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Per mesh occlusion culling OFF
Per mesh occlusion culling ON
Per mesh occlusion culling ON

Occluder plane in green
Per mesh occlusion culling OFF
Per mesh occlusion culling ON
Per mesh occlusion culling ON

Not as much geometry culled as hoped
Room for improvement?

- Effective against props and character mesh
  - Culling methods are effective against smaller AABB units

- Ineffective against large mesh
  - Large meshes are always visible
  - Need to split the mesh finely for better results
Automatic division of large mesh

- Cut out 256 triangles as one batch
- Each batch consists of consecutive Indirect Argument
  - Create AABB per batch
Issues with many micro drawing command

- Almost all draws fall below 768 indices
- Large amounts of batches cause bad performance
  - Depend on the hardware
- Merge commands if adjacent IndirectArguments are continuous
Mesh division OFF
Mesh division ON
Divide mesh OFF
Divide mesh ON
Partial Z-prepass

- To run as few fragment shaders as possible

- Z-prepass with every mesh is expensive
  - Cost can surpass the benefit

- Limiting Z-prepass to meshes close to the camera
  - Reuse auto-division models
## Comparison of each method

### Occlusion culling and GBuffer's duration (micro sec)

<table>
<thead>
<tr>
<th>Method</th>
<th>Culling</th>
<th>GBuffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frustum culling</td>
<td>2410.9</td>
<td>2295.4</td>
</tr>
<tr>
<td>Frustum culling + Occlusion culling</td>
<td>2595.2</td>
<td>1976.9</td>
</tr>
<tr>
<td>Frustum culling + Occlusion culling + Auto split</td>
<td>2593.8</td>
<td>1764.4</td>
</tr>
<tr>
<td>Frustum culling + Occlusion culling + Auto split + Partial Z-prepass</td>
<td>2383.7</td>
<td>1560.6</td>
</tr>
</tbody>
</table>

API DirectX 12, GPU AMD Radeon RX480, Radeon Profiler
Comparison of GPU-based occlusion culling

- At this point not gain performance
UAV Overlap

- In DirectX 12 shaders without dependency can execute in parallel

- UAV barrier has ambiguous dependency
  - Unclear whether read or write
  - If each batch writes to a separate location, it can be executed in parallel
    - If WAW (write-after-write) hazard is avoidable
UAVOverlap

- Controllable UAV Synchronization for each compute shader dispatch
  - Parallel execution made possible by disabling synchronization of UAV
  - In DirectX 11, it is possible to introduce equivalent functions using AGS and NVAPI.

```c
void dispatch(u32 threadGroupX, u32 threadGroupY, u32 threadGroupZ, bool uavResourceSyncDisable = false);
void dispatchIndirect(Buffer& buffer, u32 alignedOffsetForArgs, bool uavResourceSyncDisable = false);
```
Comparison: UAV Overlap

- Overall performance improvement

| 15.84ms | 15.28ms |
Wave Intrinsics

- Shader scalarization can improve the rate the threads work in parallel.
- Used for Lighting, GPU-based occlusion culling, SSR...
- For scalarization, refer to [Sousa16]

- Wave Intrinsics improves efficiency of scalarization by removing unnecessary synchronizations.

- Supported in DirectX 11 and DirectX 12
  - Using AGS Intrinsic with Shader Model 5.1
  - Can also be used with Shader Model 6.0
Comparison : Wave Intrinsics

- Overall performance improvement
Depth Bounds Test

- Clamp depth to a specific depth range
  - Mainly used to eliminate extraneous pixel shaders
  - Available with DirectX 12 (Creators Update) and DirectX 11.3
    - DirectX 11 With AGS and NVAPI

- In RE ENGINE, it is used for decals and light shafts
Decals

- Runs on pixels that failed the depth test
- Preferably omit processing when completely occluded
  - Resolved using Depth Bounds Test
Comparison of Depth Bounds Test for decals

![Bar chart showing comparison between DepthBoundsTest OFF and ON]

- **DepthBoundsTest OFF**: GBuffer duration (milli sec) = 3.5 milliseconds
- **DepthBoundsTest ON**: GBuffer duration (milli sec) = 2.5 milliseconds

API DirectX 12, GPU AMD Radeon R9 Fury X, RadeonProfiler
Console optimization comparison

15.84ms

15.35ms
Optimization for DirectX 12
Optimization

- Feedback console optimizations method to PC
  - MultiDraw
  - UAVOverlap
  - Wave Intrinsics
  - Depth Bounds Test

- Optimization for DirectX 12
  - Reduction of resource barriers
  - Buffer update
  - RootSignature
  - Memory management
Reduction of resource barriers

15.35ms → 12.13ms
Resource barrier without optimization

- In our original build without optimization, we inserted resource barrier in batches
- Immediately before executing drawing command, transition the resource barrier required for the current batch
Resource barriers

- Large number of resource barriers
  - One of the reasons GPU-based occlusion culling did not improve performance as much
Resource barriers

- Sections with many resource barriers are not operating efficiently.
Reducing resource barriers

• Optimize by considering the sub resource for each resource

• It is difficult to manually create the best resource barrier from all intermediate drawing commands

• difficulty
  • Getting maximum GPU performance
  • Keeping it Bug free 😊
Add pre-pass for command analysis

• Calculate the position of resource barrier automatically
  • Analyze intermediate drawing command

• Intermediate drawing commands are sorted by priority
  • Able to track the usage of drawing command chronologically for each resource

• Analysing batches with dependency can easily improve efficiency of GPU by shifting the priority order
Resource barrier compaction

- Search for precursor resource barrier
Resource barrier compaction

- Search for precursor resource barrier
Resource barrier compaction

- Search for precursor resource barrier
  - Bundle if possible
Advantage / Disadvantage

• **Advantage**
  • Need not be as conscious of internal implementation and caching
  • Reduce unnecessary resource barriers

• **Disadvantage**
  • Requires command parsing time
    • PC is super fast!
Comparison: Resource barrier reduction
Still not enough?

- There are still inefficient sections in updating the buffer
Still not enough?

- A large amount of resource barriers caused by the driver in DMA transfer
What was going on?

- Buffer updates on graphics queue

- CopyBufferRegion
  - GPU particle buffer update
  - Updating skinning matrix

- CopyBufferRegion is executed as DMA transfer
What was going on?

- Strong cache flush was operating when DMA transfer was performed
- **L1-Cache, L2-Cache, K-Cache**
  - Batching resource barrier has no effect

### Possible solutions
- Update with CopyQueue if only one update per frame
- Update using compute shader

- We used compute shader
Compute shader based update

StructuredBuffer<
uint> fastCopySource;
RWStructuredBuffer<
uint> fastCopyTarget;

[numthreads(256,1,1)]
void CS_FastCopy( uint groupId : SV_GroupID, uint threadId : SV_GroupThreadID )
{
    fastCopyTarget[(groupId.x * 2 + 0)*256 + threadId.x] = fastCopySource[(groupId.x * 2 + 0)*256 + threadId.x];
    fastCopyTarget[(groupId.x * 2 + 1)*256 + threadId.x] = fastCopySource[(groupId.x * 2 + 1)*256 + threadId.x];
}
Optimization of constant buffer update

• Update all constant buffer via upload heap
  • Updates to the same Constant Buffer needs resource-barrier and CopyBufferRegion(DMA transfer)
  • Store new value into upload heap and get upload heap offset address

• Shaders that use ConstantBuffer only needs reference offset address
  • Resource barrier and CopyBufferRegion are no longer needed
CopyBufferRegion reduction comparison

- Successfully removed inefficiency!
Comparison of each method

Occlusion culling and Gbuffer’s duration (micro sec)

Frustum culling
Frustum culling + Occlusion culling
Frustum culling + Occlusion culling + Auto split
Frustum culling + Occlusion culling + Auto split + Partial Z-prepass + Reduction resource barrier

Culling GBuffer

<table>
<thead>
<tr>
<th>Method</th>
<th>Culling</th>
<th>GBuffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frustum culling</td>
<td>2410.9</td>
<td>2295.4</td>
</tr>
<tr>
<td>Frustum culling + Occlusion culling</td>
<td>2595.2</td>
<td>1976.9</td>
</tr>
<tr>
<td>Frustum culling + Occlusion culling + Auto split</td>
<td>2593.8</td>
<td>829.4</td>
</tr>
<tr>
<td>Frustum culling + Occlusion culling + Auto split + Partial Z-prepass</td>
<td>2383.7</td>
<td>823.1</td>
</tr>
<tr>
<td>Frustum culling + Occlusion culling + Auto split + Partial Z-prepass + Reduction resource barrier</td>
<td>1877.9</td>
<td>1592.5</td>
</tr>
</tbody>
</table>

API DirectX 12, GPU AMD Radeon RX480, Radeon Profiler
Root Signature

- DirectX12 uses similar RootSignature to DX11 & Consoles
- Determined at runtime, not at shader build
  - To provide customized optimization for each IHV
  - For AMD, use RootParameter as table
  - For NVIDIA, use RootParameter to optimize ConstantBuffer access

```
Root(AMD and Intel)
DescriptorTable(CBV 0-14)
DescriptorTable(SRV 0-32)
DescriptorTable(UAV 0-8)
DescriptorTable(Sampler)
```

```
Root(NVIDIA)
RootCBV(0)
RootCBV(1)
RootCBV(2)
RootCBV(3)
DescriptorTable(CBV 4-14)
DescriptorTable(SRV 0-32)
DescriptorTable(UAV 0-8)
DescriptorTable(Sampler)
```
Memory management

• In the first implementation, memory Evict started at around 50% memory usage
  • Pretty conservative

• Many spikes occurred during gameplay
  • In Resident Evil 2, controls loading and disposal for each room caused spikes every time the character moved
  • Even occurred when loading UI for pause menus
Memory management

• Do not Evict until memory is exhausted
  • To prevent micro Evicts

• When the memory usage rate exceeds 90%, unreferenced memory is Evicted
Comparison after all optimizations

- 24% frame time saving!
Comparison of DirectX 11 / DirectX 12

- Profile Resident Evil 2 in game

Frame time (milli sec)

DirectX 11: 13.58
DirectX 11 with AGS: 11.49
DirectX 12: 11.43

GPU: AMD Radeon RX480, OCAT
Future works

- AsyncCompute
  - Used for Consoles
  - Implementation was incompatible for PC

- Shader model 6.0
  - Some tests and trials were done
  - Not enough time to ensure stability
Optimization recap

• Although optimizations from console are useful, it may be inadequate by itself

• Reducing resource barrier is important
  • Big impact on performance!
  • Effectiveness of other optimization methods can be affected by the resource barrier

• Paging spikes decreased when memory management was done all at once rather than doing it in small increments
  • May be due to game design
  • Worked well even at around 90% utilization
Tips
Pre-bake PipelineStateObject

- Creating PipelineStateObject at runtime is slow
- It would be better if we can Pre-bake PipelineStateObject beforehand.
Pre-bake PipelineStateObject

- We pre-bake PSO before the final package
  - Included in assets created on the engine
  ```json
  {
    "Name":"LightCulling",
    "CS": ["lightCulling.hlsl","CS_LightCulling"],
  },
  {
    "Name":"IndirectIllumination",
    "CS": ["deferred.hlsl","IndirectIlluminationCS"],
  }
  ```
- RTV, DSV and index bit stride are not included at first
- We use the collected information to pre-bake the PSO for the final package
  - Much smoother for the end-user.
Load PipelineStateObject at runtime

- Compile in the background during asset loading
  - Compute shader: Create immediately on another thread
  - Other shaders: Create if it is on the collected information.

- However, if the build of PipelineStateObject is not completed beforehand, the CPU is blocked
Quality Assurance (QA)

• Quality Assurance for PC version frequently suffer from GPU crashes
  • Various factors such as CPU, GPU, display, etc

• However, crash dumps were not useful for debugging GPU crashes
  • No way to trace
  • RE ENGINE does not offer functions to replay command lists… yet

• In DirectX 12, use WriteBufferImmediate
  • Read back executing shader name to the buffer for each drawing command
  • Able to know the shader name that was running at the time of crash
  • In DirectX 11, AGS supports BreadcrumbBuffer as same function.
Acknowledgments

- Big thanks to RE ENGINE dev team’s contribution and to the support of IHVs
  - Many bugs were fixed by the driver team!
Questions?
References

- GPU-Driven Rendering Pipelines, Ulrich Haar (Ubisoft Entertainment), Sebastian Aaltonen (Ubisoft Entertainment)
- Optimizing the Graphics Pipeline with Compute, Graham Wihlidal
- Improved Culling for Tiled and Clustered Rendering, Michal Drobot
- The Devils in the details, Tiago Sousa (idTech), Jean Geffroy (idTech) Siggraph 2016
- AMD GeometryFX
- Rendering with Conviction, Stephen Hill
- Moving to DirectX 12: Lessons Learned, Tiago Rodrigues
- Graphics optimization of the latest title in Capcom, Hitoshi Mishima CEDEC 2018