GPU-Driven Rendering Pipelines

Ulrich Haar, Lead Programmer 3D, Ubisoft Montreal
Sebastian Aaltonen, Senior Lead Programmer, RedLynx a Ubisoft Studio
Topics

- Motivation
- Mesh Cluster Rendering
- Rendering Pipeline Overview
- Occlusion Depth Generation
- Results and future work
GPU-Driven Rendering?

• GPU controls what objects are actually rendered
• "draw scene" GPU-command
  – n viewports/frustums
  – GPU determines (sub-)object visibility
  – No CPU/GPU roundtrip
• Prior work [SBOT08]

SIGGRAPH 2015: Advances in Real-Time Rendering course
Motivation (RedLynx)

- Modular construction using in-game level editor
- High draw distance. Background built from small objects.
- No baked lighting. Lots of draw calls from shadow maps.
- CPU used for physics simulation and visual scripting
Motivation

Assassin’s Creed Unity

• Massive amounts of geometry: architecture
Motivation

Assassin’s Creed Unity

• Massive amounts of geometry: seamless interiors
Motivation

Assassin’s Creed Unity

• Massive amounts of geometry: crowds
Motivation
Assassin’s Creed Unity

• Modular construction (partially automated)
• ~10x instances compared to previous Assassin’s Creed games
• CPU scarcest resource on consoles
Mesh Cluster Rendering

- Fixed topology (64 vertex strip)
- Split & rearrange all meshes to fit fixed topology (insert degenerate triangles)
- Fetch vertices manually in VS from shared buffer [Riccio13]
- DrawInstancedIndirect
- GPU culling outputs cluster list & drawcall args
Mesh Cluster Rendering

- Arbitrary number of meshes in single drawcall
- GPU-culled by cluster bounds
  [Greene93] [Shopf08] [Hill11]
- Faster vertex fetch
- Cluster depth sorting
Mesh Cluster Rendering (ACU)

• Problems with triangle strips:
  – Memory increase due to degenerate triangles
  – Non-deterministic cluster order

• MultiDrawIndexedInstanceIndirect:
  – One (sub-)drawcall per instance
  – 64 triangles per cluster
  – Requires appending index buffer on the fly
Rendering Pipeline Overview

- **Coarse Frustum Culling**
- **Build Batch Hash**
- **Update Instance GPU Data**
- **Batch DrawCalls**
- **Instance Culling** (Frustum/Occlusion)
- **Cluster Chunk Expansion**
- **Cluster Culling** (Frustum/Occlusion/Triangle Backface)
- **Index Buffer Compaction**
- **Multi-Draw**

SIGGRAPH 2015: Advances in Real-Time Rendering course
Rendering pipeline overview

- CPU quad tree culling
- Per instance data:
  - E.g. transform, LOD factor...
  - Updated in GPU ring buffer
  - Persistent for static instances
- Drawcall hash build on non-instanced data:
  - E.g. material, renderstate, ...
- Drawcalls merged based on hash

SIGGRAPH 2015: Advances in Real-Time Rendering course
Rendering Pipeline Overview

Instance Culling (Frustum/Occlusion)

Instance0 | Instance1 | Instance2 | Instance3 | …

- Transform
- Bounds
- Mesh

This stream of instances contains a list of offsets into a GPU-buffer per instance that allows the GPU to access information like transform, instance bounds, etc.

Cluster Chunk Expansion

Cluster Culling (Frustum/Occlusion/Triangle Backface)

Index Buffer Compaction

Multi-Draw
Rendering Pipeline Overview

INSTANCE CULLING (FRUSTUM/OCCLUSION)

CLUSTER CHUNK EXPANSION

CLUSTER CULLING (FRUSTUM/OCCLUSION/TRIANGLE BACKFACE)

INDEX BUFFER COMPACTION

MULTI-DRAW

SIGGRAPH 2015: Advances in Real-Time Rendering course
Rendering Pipeline Overview

- **INSTANCE CULLING (FRUSTUM/OCCLUSION)**
- **CLUSTER CHUNK EXPANSION**
  - Chunk1_0
  - Chunk2_0
  - Chunk2_1
  - Chunk2_2
  - …
- **CLUSTER CULLING (FRUSTUM/OCCLUSION/TRIANGLE BACKFACE)**
- **INDEX BUFFER COMPACTION**
- **MULTI-DRAW**
Rendering Pipeline Overview

- INSTANCE CULLING (FRUSTUM/OCCLUSION)
- CLUSTER CHUNK EXPANSION
- CLUSTER CULLING (FRUSTUM/OCCLUSION/TRIANGLE BACKFACE)
- INDEX BUFFER COMPACTATION
- MULTI-DRAW

Triangle Mask Read/Write Offsets
Rendering Pipeline Overview

INSTANCE CULLING (FRUSTUM/OCCLUSION)

CLUSTER CHUNK EXPANSION

CLUSTER CULLING (FRUSTUM/OCCLUSION/TRIANGLE BACKFACE)

INDEX BUFFER COMPACTION

INDEX COMPACTION

Instance0  Instance1  Instance2

0  1  0  1  0  1  2  ...

Compacted index buffer

MULTI-DRAW

SIGGRAPH 2015: Advances in Real-Time Rendering course
Rendering Pipeline Overview

- INSTANCE CULLING (FRUSTUM/OCCLUSION)
- CLUSTER CHUNK EXPANSION
- CLUSTER CULLING (FRUSTUM/OCCLUSION/TRIANGLE BACKFACE)
- INDEX BUFFER COMPACTION

MULTI-DRAW

SIGGRAPH 2015: Advances in Real-Time Rendering course
Rendering Pipeline Overview

- **INSTANCE CULLING (FRUSTUM/OCCULSION)**
- **CLUSTER CHUNK EXPANSION**
- **CLUSTER CULLING (FRUSTUM/OCCULSION/TRIANGLE BACKFACE)**
- **INDEX BUFFER COMPACTION**

---

**Multi-Draw**

Drawcall 0

Drawcall 1

Drawcall 2

0 1 1 1

1 64 3 8

SIGGRAPH 2015: Advances in Real-Time Rendering course
Static Triangle Backface Culling

- Bake triangle visibility for pixel frustums of cluster centered cubemap
- Cubemap lookup based on camera
- Fetch 64 bits for visibility of all triangles in cluster
Static Triangle Backface Culling
Static Triangle Backface Culling

• Only one pixel per cubemap face (6 bits per triangle)
• Pixel frustum is cut at distance to increase culling efficiency (possible false positives at oblique angles)
• 10-30% triangles culled
Occlusion Depth Generation
Occlusion Depth Generation

- Depth pre-pass with best occluders
- Rendered in full resolution for High-Z and Early-Z
- Downsampled to 512x256
- Combined with reprojection of last frame’s depth
- Depth hierarchy for GPU culling
Occlusion Depth Generation

- 300 best occluders (~600us)
- Rendered in full resolution for High-Z and Early-Z
- Downsampled to 512x256 (100us)
- Combined with reprojection of last frame’s depth (50us)
- Depth hierarchy for GPU culling (50us)

(*PS4 performance *)

SIGGRAPH 2015: Advances in Real-Time Rendering course
Shadow Occlusion Depth Generation

• For each cascade
• Camera depth reprojection (~70us)
• Combine with shadow depth reprojection (10us)
• Depth hierarchy for GPU culling (30us)
Camera Depth Reprojection
Camera Depth Reprojection

SIGGRAPH 2015: Advances in Real-Time Rendering course
Camera Depth Reprojection
Camera Depth Reprojection
Camera Depth Reprojection

Light Space Reprojection
Camera Depth Reprojection

Reprojection “shadow” of the building
Camera Depth Reprojection

• Similar to [Silvennoinen12]
• But, mask not effective because of fog:
  – Cannot use min-depth
  – Cannot exclude far-plane
• 64x64 pixel reprojection
• Could pre-process depth to remove redundant overdraw
Results

CPU:
• 1-2 Orders of magnitude less drawcalls
• ~75% of previous AC, with ~10x objects

GPU:
• 20-40% triangles culled (backface + cluster bounds)
• Only small overall gain: <10% of geometry rendering
• 30-80% shadow triangles culled

Work in progress:
• More GPU-driven for static objects
• More batch friendly data
Future

- Bindless textures
- GPU-driven vs. DX12/Vulkan
RedLynx Topics

- Virtual Texturing in GPU-Driven Rendering
- Virtual Deferred Texturing
- MSAA Trick
- Two-Phase Occlusion Culling
- Virtual Shadow Mapping
Virtual Texturing

- **Key idea:** Keep only the visible texture data in memory [Hall99]
- Virtual $256k^2$ texel atlas
- $128^2$ texel pages
- $8k^2$ texture page cache
  - 5 slice texture array: Albedo, specular, roughness, normal, etc.
  - DXT compressed (BC5 / BC3)
GPU-Driven Rendering with VT

• Virtual texturing is the biggest difference between our and AC: Unity’s renderer

• Key feature: All texture data is available at once, using just a single texture binding

• No need to batch by textures!
Single Draw Call Rendering

- Viewport = single draw call (x2)
- Dynamic branching for different vertex animation types
  - Fast on modern GPUs (+2% cost)
- Cluster depth sorting provides gain similar to depth prepass
- Cheap OIT with inverse sort
Additional VT Advantages

- Complex material blends and decal rendering results are stored to VT page cache
- Data reuse amortizes costs over hundreds of frames
- Constant memory footprint, regardless of texture resolution and the number of assets
Virtual Deferred Texturing

- **Old Idea:** Store UVs to the G-buffer instead of texels [Auf.07]
- **Key feature:** VT page cache atlas contains all the currently visible texture data
- 16+16 bit UV to the $8k^2$ texture atlas gives us $8 \times 8$ subpixel filtering precision
Gradients and Tangent Frame

• Calculate pixel gradients in screen space. UV distance used to detect neighbors.
• No neighbors found → bilinear
• Tangent frame stored as a 32 bit quaternion [Frykholm09]
• Implicit mip and material id from VT. Page = UV.xy / 128.
Recap & Advantages

- 64 bits. Full fill rate. No MRT.
- Overdraw is dirt cheap
  - Texturing deferred to lighting CS
- Quad efficiency less important
- Virtual texturing page ID pass is no longer needed
Gradient reconstruction quality

Ground truth  Reconstructed  Difference (x4)

SIGGRAPH 2015: Advances in Real-Time Rendering course
**MSAA Trick**

- **Key Observation:** UV and tangent can be interpolated
- **Idea:** Render the scene at 2x2 lower resolution (540p) with ordered grid 4xMSAA pattern
- Use `Texture2DMS.Load()` to read each sample separately in the lighting compute shader

\[
P_1 = A + \frac{1}{4} \overrightarrow{AB} + \frac{1}{4} \overrightarrow{AC}
\]
\[
P_2 = B + \frac{1}{4} \overrightarrow{BA} + \frac{1}{4} \overrightarrow{BD}
\]
\[
P_3 = C + \frac{1}{4} \overrightarrow{CA} + \frac{1}{4} \overrightarrow{CD}
\]
\[
P_4 = D + \frac{1}{4} \overrightarrow{DC} + \frac{1}{4} \overrightarrow{DB}
\]
1080p Reconstruction

- Reconstruct 1080p into LDS
- Edge pixels are perfectly reconstructed. MSAA runs the pixel shader for both sides.
- Interpolate the inner pixels’ UV and tangent
- Quality is excellent. Differences are hard to spot.

\[ P_1 = A + \frac{1}{4} \overrightarrow{AB} + \frac{1}{4} \overrightarrow{AC} \]
\[ P_2 = B + \frac{1}{4} \overrightarrow{BA} + \frac{1}{4} \overrightarrow{BD} \]
\[ P_3 = C + \frac{1}{4} \overrightarrow{CA} + \frac{1}{4} \overrightarrow{CD} \]
\[ P_4 = D + \frac{1}{4} \overrightarrow{DC} + \frac{1}{4} \overrightarrow{DB} \]
8xMSAA Trick Benchmark

- 128 bpp G-Buffer
- One pixel is a 2x2 tile of “2xMSAA pixels”
- Xbox One: 1080p + MSAA + 60 fps 😊

<table>
<thead>
<tr>
<th></th>
<th>2xMSAA</th>
<th>MSAA trick</th>
<th>Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>G-buffer rendering time</td>
<td>3.03 ms</td>
<td>2.06 ms</td>
<td>-32%</td>
</tr>
<tr>
<td>Pixel shader waves</td>
<td>83016</td>
<td>36969</td>
<td>-55%</td>
</tr>
<tr>
<td>DRAM memory traffic ESRAM (18 MB partial)</td>
<td>76.3 MB 60.9 MB</td>
<td>15.0 MB 29.1 MB</td>
<td>-20%</td>
</tr>
</tbody>
</table>

SIGGRAPH 2015: Advances in Real-Time Rendering course
Two-Phase Occlusion Culling

- No extra occlusion pass with low poly proxy geometry
- Precise WYSIWYG occlusion
- Based on depth buffer data
- Depth pyramid generated from HTILE min/max buffer
- $O(1)$ occlusion test (gather4)

SIGGRAPH 2015: Advances in Real-Time Rendering course
Two-Phase Occlusion Culling

1\textsuperscript{st} phase
- Cull objects & clusters using last frame’s depth pyramid
- Render visible objects

2\textsuperscript{nd} phase
- Refresh depth pyramid
- Test culled objects & clusters
- Render false negatives
Benchmark

- “Torture” unit test scene
  - 250,000 separate moving objects
  - 1 GB of mesh data (10k+ meshes)
  - 8k² texture cache atlas

- DirectX 11 code path
  - 64 vertex clusters (strips)
  - No ExecuteIndirect / MultiDrawIndirect

- Only two DrawInstancedIndirect calls
# Benchmark Results

## Xbox One, 1080p

<table>
<thead>
<tr>
<th>GPU time</th>
<th>1(^{\text{st}}) phase</th>
<th>2(^{\text{nd}}) phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object culling + LOD</td>
<td>0.28 ms</td>
<td>0.26 ms</td>
</tr>
<tr>
<td>Cluster culling</td>
<td>0.09 ms</td>
<td>0.04 ms</td>
</tr>
<tr>
<td>Draw (G-buffer)</td>
<td>1.60 ms</td>
<td>&lt; 0.01 ms</td>
</tr>
<tr>
<td>Pyramid generation</td>
<td>0.06 ms</td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td>2.3 ms</td>
</tr>
</tbody>
</table>

**CPU time:** 0.2 milliseconds (single Jaguar CPU core)
Virtual Shadow Mapping

• $128k^2$ virtual shadow map
• $256^2$ texel pages
• Identify needed shadow pages from the z-buffer [Fernando01].
• Cull shadow pages with the GPU-driven pipeline.
• Render all pages at once.
VTSM Quality and Performance

- Close to 1:1 shadow-to-screen resolution in all areas
- **Measured**: Up to 3.5x faster than SDSM [Lauritzen10] in complex “sparse” scenes
- Virtual SM slightly slower than SDSM & CSM in simple scenes
GPU-Driven Rendering + DX12

NEW DX12 (PC) FEATURES
- ExecuteIndirect
- Asynchronous Compute
- VS RT index (GS bypass)
- Resource management
- Explicit multiadapter
- Tiled resources + bindless
- Conservative raster + ROV

FEATURES IN OTHER APIs
- Custom MSAA patterns
- GPU side dispatch
- SIMD lane swizzles
- Ordered atomics
- SV_Barycentric to PS
- Exposed CSAA/EQAA samples
- Shading language with templates

SIGGRAPH 2015: Advances in Real-Time Rendering course
References


[Silvennoinen12] Chasing Shadows, GDMag Feb/2012.


Acknowledgements

• Stephen Hill
• Roland Kindermann
• Jussi Knuuttila
• Jalal Eddine El Mansouri
• Tiago Rodrigues
• Lionel Berenguier
• Stephen McAuley
• Ivan Nevraev
GPU-Driven Rendering Pipelines

Ulrich Haar, Lead Programmer 3D
Ubisoft Montreal

Sebastian Aaltonen, Senior Lead Programmer
RedLynx, a Ubisoft Studio