Deferred Rendering for Current and Future Rendering Pipelines

Andrew Lauritzen
Advanced Rendering Technology (ART)
Intel Corporation
Overview

- Forward shading
- Deferred shading and lighting
- Tile-based deferred shading
- Deferred multi-sample anti-aliasing (MSAA)
Forward Shading

• Do everything we need to shade a pixel
  – for each light
    • Shadow attenuation (sampling shadow maps)
    • Distance attenuation
    • Evaluate lighting and accumulate
• Multi-pass requires resubmitting scene geometry
  – Not a scalable solution
Forward Shading Problems

• Ineffective light culling
  – Object space at best
  – Trade-off with shader permutations/batching
• Memory footprint of all inputs
  – Everything must be resident at the same time (!)
• Shading small triangles is inefficient
  – Covered earlier in this course: [Fatahalian 2010]
Conventional Deferred Shading

• Store lighting inputs in memory (G-buffer)
  – for each light
    • Use rasterizer to scatter light volume and cull
    • Read lighting inputs from G-buffer
    • Compute lighting
    • Accumulate lighting with additive blending

• Reorders computation to extract coherence
Modern Implementation

• Cull with screen-aligned quads
  – Cover light extents with axis-aligned bounding box
    • Full light meshes (spheres, cones) are generally overkill
    • Can use oriented bounding box for narrow spot lights
  – Use conservative single-direction depth test
    – Two-pass stencil is more expensive than it is worth
    – Depth bounds test on some hardware, but not batch-friendly
Lit Scene (256 Point Lights)
Quad-Based Light Culling
Deferred Shading Problems

• Bandwidth overhead when lights overlap
  – for each light
    • Use rasterizer to scatter light volume and cull
    • Read lighting inputs from G-buffer $\leftarrow$ overhead
    • Compute lighting
      • Accumulate lighting with additive blending $\leftarrow$ overhead

• Not doing enough work to amortize overhead
Improving Deferred Shading

• Reduce G-buffer overhead
  – Access fewer things inside the light loop
  – Deferred lighting / light pre-pass
• Amortize overhead
  – Group overlapping lights and process them together
  – Tile-based deferred shading
Deferred Lighting / Light Pre-Pass

- Goal: reduce G-buffer overhead
- Split diffuse and specular terms
  - Common concession is monochromatic specular
- Factor out constant terms from summation
  - Albedo, specular amount, etc.
- Sum inner terms over all lights
Deferred Lighting / Light Pre-Pass

• Resolve pass combines factored components
  – Still best to store all terms in G-buffer up front
  – Better SIMD efficiency
• Incremental improvement for some hardware
  – Relies on pre-factoring lighting functions
  – Ability to vary resolve pass is not particularly useful
• See [Hoffman 2009] and [Stone 2009]
Tile-Based Deferred Shading

• Goal: amortize overhead

• Use screen tiles to group lights
  – Use tight tile frusta to cull non-intersecting lights
    • Reduces number of lights to consider
  – Read G-buffer once and evaluate all relevant lights
    • Reduces bandwidth of overlapping lights

• See [Andersson 2009] for more details
Lit Scene (1024 Point Lights)
Tile-Based Light Culling

Beyond Programmable Shading, SIGGRAPH 2010
Quad-Based Lighting Culling
Light Culling Only at 1080p

- Tile setup dominates

Frame Time (ms)

Number of Point Lights

Slope ~ 7 µs / light
Slope ~ 0.5 µs / light

Quad (ATI 5870)
Quad (NVIDIA 480)
Tiled (NVIDIA 480)
Tiled (ATI 5870)
Total Performance at 1080p

Deferred lighting slightly faster, but trends similarly

Slope ~ 20 µs / light
Slope ~ 4 µs / light

Few lights overlap

Frame Time (ms)

Number of Point Lights

Deferred Shading (NVIDIA 480)
Deferred Shading (ATI 5870)
Deferred Lighting (ATI 5870)
Deferred Lighting (NVIDIA 480)
Tiled (NVIDIA 480)
Tiled (ATI 5870)
Anti-aliasing

- Multi-sampling with deferred rendering requires some work
  - Regular G-buffer couples visibility and shading
- Handle multi-frequency shading in user space
  - Store G-buffer at sample frequency
  - Only apply per-sample shading where necessary
  - Offers additional flexibility over forward rendering
Identifying Edges

- Forward MSAA causes redundant work
  - It applies to all triangle edges, even for continuous, tessellated surfaces
- Want to find surface discontinuities
  - Compare sample depths to depth derivatives
  - Compare (shading) normal deviation over samples
Per-Sample Shading Visualization
MSAA with Quad-Based Methods

• Mark pixels for per-sample shading
  – Stencil still faster than branching on most hardware
  – Probably gets scheduled better

• Shade in two passes: per-pixel and per-sample
  – Unfortunately, duplicates culling work
  – Scheduling is still a problem
Per-Sample Scheduling

• Lack of spatial locality causes hardware scheduling inefficiency
MSAA with Tile-Based Methods

• Handle per-pixel and per-sample in one pass
  – Avoids duplicate culling work
  – Can use branching, but incurs scheduling problems
  – Instead, reschedule per-sample pixels
    • Shade sample 0 for the whole tile
    • Pack a list of pixels that require per-sample shading
    • Redistribute threads to process additional samples
    • Scatter per-sample shaded results
Tile-Based MSAA at 1080p, 1024 Lights

![Bar Chart]

Frame Time (ms)

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Frame Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crytek Sponza (ATI 5870)</td>
<td></td>
</tr>
<tr>
<td>2009 Game (ATI 5870)</td>
<td></td>
</tr>
<tr>
<td>Crytek Sponza (NVIDIA 480)</td>
<td></td>
</tr>
<tr>
<td>2009 Game (NVIDIA 480)</td>
<td></td>
</tr>
</tbody>
</table>

- No MSAA
- 4x MSAA (Branching)
- 4x MSAA (Packed)

Beyond Programmable Shading, SIGGRAPH 2010
4x MSAA Performance at 1080p

- Slope ~ 35 µs / light
- Slope ~ 5 µs / light

Tiled takes less of a hit from MSAA

Deferred lighting even less compelling

Beyond Programmable Shading, SIGGRAPH 2010
Conclusions

• Deferred shading is a useful rendering tool
  – Decouples shading from visibility
  – Allows efficient user-space scheduling and culling

• Tile-based methods win going forward
  – Fastest and most flexible
  – Enable efficient MSAA
Future Work

• Hierarchical light culling
  – Straightforward but would need lots of small lights

• Improve MSAA memory usage
  – Irregular/compressed sample storage?
  – Revisit binning pipelines?
  – Sacrifice higher resolutions for better AA?
Acknowledgements

• Microsoft and Crytek for the scene assets
• Johan Andersson from DICE
• Craig Kolb, Matt Pharr, and others in the Advanced Rendering Technology team at Intel
• Nico Galoppo, Anupreet Kalra and Mike Burrows from Intel
References

Questions?

• Full source and demo available at:
  – http://visual-computing.intel-research.net/art/publications/deferred_rendering/
Backup
Quad-Based Light Culling

• Accumulate many lights per draw call
  – Render one point per light
  – Vertex shader computes quad bounds for light
  – Geometry shader expands into two triangles (quad)
  – Pixel shader reads G-buffer and evaluates lighting
Tile-Based Deferred Lighting?

• Can do deferred lighting with tiling...
  – Not usually worth sacrificing the flexibility
  – Bandwidth already minimized
  – Additional resolve pass can make it slower overall

• Exception: hardware considerations
  – SPU lighting on Playstation 3
    • Moving less data across the bus can be an overall win