Bandwidth-Efficient Rendering

Marius Bjørge

ARM
Agenda

- Efficient on-chip rendering
- Post-processing
  - Bloom
  - Blur filters
Efficient on-chip rendering

- Extensions
  - Framebuffer fetch
  - Pixel Local Storage

- Why extensions?
  - Surely mobile GPUs are already bandwidth-efficient?
Framebuffer fetch

- Read the current fragment’s previous color value
- ARM also supports reading the previous depth and stencil values of the current fragment

- Useful for
  - Programmable blending
  - Programmable depth/stencil testing
Pixel Local Storage (PLS)

- Per-pixel storage that is persistent throughout the lifetime of the frame
  - Read/write access
  - Storage stays on-chip
  - Storage layout declared per fragment shader invocation – does not depend on framebuffer format

- Useful for
  - Deferred shading
  - Order Independent Transparency [1]
  - Volume rendering
Pixel Local Storage (PLS)

- Rendering pipeline changes slightly when PLS is enabled
  - Writing to PLS bypasses blending
- Note
  - Fragment order
  - PLS and color share the same memory location
At this point we change the layout of the PLS.
Post-processing
Post-processing

• High-end mobile devices typically have small displays with massive resolutions

• Rendering at native resolution is often out of the question, especially if you add post-processing to the mix

• Solution: mixed resolution rendering
  – Go as low as you can without sacrificing quality, and then upscale
Mobile post-processing

On-chip
- Color Grading
- Tonemapping

Off-chip
- Anti-aliasing
- Bloom
- Depth of Field
- Screen Space Ambient Occlusion
- Screen Space Reflections
Bloom

- Doesn’t have to be physically correct
- Wide + thin
Blur

• What makes a good blur filter?

• Goal:
  – High quality
  – Stable
  – High performance
Box blur

- 5x5 box blur = 25 samples
- Separate the blurs
  - 5 + 5 = 10 samples
Gaussian blur

- Convolve a gaussian function over the image
- Separable just like the box filter
Linear sampling optimization [2]

- Reduce number of texture lookups by exploiting the HW texture unit
  - Modify sample offsets and gaussian weights
- Get 9x9 at similar cost as 5x5
Mixing resolutions
Mixing resolutions

• Gets increasingly complicated when using separable kernels
Kawase blur [3]
Kawase blur
“Dual filtering”
“Dual filtering”
Comparing filters
Comparison setup

- 97x97 blur
- Gaussian used as reference
- Kawase
  - First downsample to 1/16\textsuperscript{th} resolution
  - Setup with 0, 1, 2, 3, 4, 4, 5, 6, 7 distances passes
- “Dual filtering” setup with 8 passes
- Naïve method which relies on \texttt{glGenerateMipmap}
<table>
<thead>
<tr>
<th>Input</th>
<th>Reference</th>
<th>Dual</th>
<th>Kawase</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Input Image" /></td>
<td><img src="image2.png" alt="Reference Image" /></td>
<td><img src="image3.png" alt="Dual Image" /></td>
<td><img src="image4.png" alt="Kawase Image" /></td>
</tr>
<tr>
<td><img src="image5.png" alt="Input Image" /></td>
<td><img src="image6.png" alt="Reference Image" /></td>
<td><img src="image7.png" alt="Dual Image" /></td>
<td><img src="image8.png" alt="Kawase Image" /></td>
</tr>
</tbody>
</table>

**PSNR:**

- Dual: 49.78 dB
- Kawase: 50.02 dB
Stability comparison
<table>
<thead>
<tr>
<th>Input</th>
<th>Reference</th>
<th>Dual</th>
<th>Kawase</th>
<th>Naive</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Input Image" /></td>
<td><img src="image2.png" alt="Reference Image" /></td>
<td><img src="image3.png" alt="Dual Image" /></td>
<td><img src="image4.png" alt="Kawase Image" /></td>
<td><img src="image5.png" alt="Naive Image" /></td>
</tr>
</tbody>
</table>
Performance comparison
Performance (ms)

Tested on a Mali-T760 MP8
Cache utilization

<table>
<thead>
<tr>
<th>Method</th>
<th>Hit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear sampling</td>
<td>2%</td>
</tr>
<tr>
<td>5x5 gaussian reduction</td>
<td>70%</td>
</tr>
<tr>
<td>Kawase</td>
<td>63%</td>
</tr>
<tr>
<td>Dual</td>
<td>71%</td>
</tr>
</tbody>
</table>
Summary

• On-chip rendering
  – Please use the extensions

• Bloom
  – Multi-pass mixed resolution
  – “Dual filter” blur

• Next steps
  – Work on getting on-chip rendering into future core APIs
  – Look into alternative data flows for doing blurs
Thanks!

• Questions?
  – Marius.Bjorge@arm.com

• References
  1. Efficient Rendering with Tile Local Storage [Siggraph 2014]