Exploiting Spatial Redundancy with Adaptive Pyramidal Rendering

Dr. Orion Sky Lawlor
lawlor@alaska.edu
& Dr. Jon Genetti
U. Alaska Fairbanks
WSCG 2014-06-04
Why Pyramidal Rendering?

Higher resolution displays
300ppi smartphone
5 megapixel Apple Retina
30+ megapixel powerwalls

Lower & tighter latency requirements
Head tracking: 120hz 1080p or better

Shader complexity increasing
Rasterize, raytrace, or both?

Battery-constrained mobile GPUs
Yet still want cinema experience
In the beginning, coarse pixels
Fine pixels: clean edges, but slow!

<table>
<thead>
<tr>
<th>F</th>
<th>F</th>
<th>F</th>
<th>F</th>
<th>F</th>
<th>F</th>
<th>F</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
# Adaptive pyramidal rendering

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

Dr. Lawlor, U. Alaska
Pyramidal Rendering Steps

1  Render initial coarse image
2  For each pixel in finer image
   3   If coarser image is smooth enough
   4   Interpolate from coarser image
   5     else
   6     Render finer image pixel
7  Repeat from 2 until fine enough
Pyramidal Rendering Steps

1. Render initial coarse image

2. For each pixel in finer image

3. If coarser image is smooth enough

4. Interpolate from coarser image

5. else

6. Render finer image pixel

7. Repeat from 2 until fine enough
Pyramidal Rendering Steps

1. Render initial coarse image

2. For each pixel in finer image

3. If coarser image is smooth enough

4. Interpolate from coarser image

5. else

6. Render finer image pixel

7. Repeat from 2 until fine enough
Pyramidal Rendering Steps

1. Render initial coarse image

2. For each pixel in finer image
3. If coarser image is smooth enough
4. Interpolate from coarser image
5. else
6. Render finer image pixel
   Raytracer, or Rasterizer with early exit
7. Repeat from 2 until fine enough
Error Metric

The **error metric** examines the coarse pixels to decide between interpolation and sampling to make fine pixels.

Can use **any** pixel data here: RGB, UV, XYZ, eye tracking, mesh attribute, ...

Typically, you'd compare the local coarse neighborhood (a stencil) against an interpolant (e.g., a polynomial).

- A good fit means interpolation is safe
- A poor fit means you should sample
Metric stencil and order vs error

<table>
<thead>
<tr>
<th>Interpolating Polynomial order</th>
<th>4</th>
<th>8</th>
<th>12</th>
<th>20</th>
<th>24</th>
<th>46</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P^1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P^3$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P^5$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P^9$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Neighborhood stencil size (coarse pixels)</td>
<td>4</td>
<td>8</td>
<td>12</td>
<td>20</td>
<td>24</td>
<td>46</td>
</tr>
</tbody>
</table>

Expectation: more is better!
Metric stencil and order vs error

<table>
<thead>
<tr>
<th>Neighborhood Stencil Size (coarse pixels)</th>
<th>4</th>
<th>8</th>
<th>12</th>
<th>20</th>
<th>24</th>
<th>46</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interpolating Polynomial order -&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P^1$</td>
<td>2.31%</td>
<td>2.32%</td>
<td>2.36%</td>
<td>2.48%</td>
<td>2.53%</td>
<td>2.63%</td>
</tr>
<tr>
<td>$P^3$</td>
<td>2.31%</td>
<td>2.26%</td>
<td>2.29%</td>
<td>2.34%</td>
<td>2.35%</td>
<td>2.41%</td>
</tr>
<tr>
<td>$P^5$</td>
<td>*4</td>
<td>2.54%</td>
<td>2.25%</td>
<td>2.31%</td>
<td>2.31%</td>
<td>2.34%</td>
</tr>
<tr>
<td>$P^9$</td>
<td>*</td>
<td>*</td>
<td>2.28%</td>
<td>2.30%</td>
<td>2.29%</td>
<td>2.30%</td>
</tr>
</tbody>
</table>

* indicates interpolation and further sampling.
The effect of the particular polynomial and stencil is weak: if the image is smooth, any reasonable metric correctly shows it should be interpolated.

Large stencils cause “false positives”, expanding detail far from the true cause.

High order polynomials cause “false negatives”, assuming a smooth high-order curve where something more subtle is happening.

Averaging is 10% better than maximum error.

Some common choices, like “contrast metric” [Mitchell 87], are over 30% worse than P1-4!
Examples of Adaptive Pyramidal Rendering
Pyramidal Rendering: 16fps
Coarse Rendering: 40fps
Adaptive Reconstruction
Black = fine, White = coarse
Speed vs Image Quality Tradeoff

Average Absolute Color Error (%) vs Samples Per Finished Pixel

Rendering Speed
Benchmark Image Bank

Raster order by increasing interpolation accuracy.

Internet Ray Tracing Contest winners & honorable mention images.
Adaptive Reconstruction
Black = fine, White = coarse
Conclusions

Interpolate smooth areas of image
Definition of “smooth” is up to you!
Easy to implement in shader
2x speedup with good image quality

This trick should be in your toolbox
Any raytracer or fill-limited renderer

See examples in WSCG Short/J37 zip,
or code at: tinyurl.com/WSCGpyramid

Future work: spectral raytracing? GI?