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

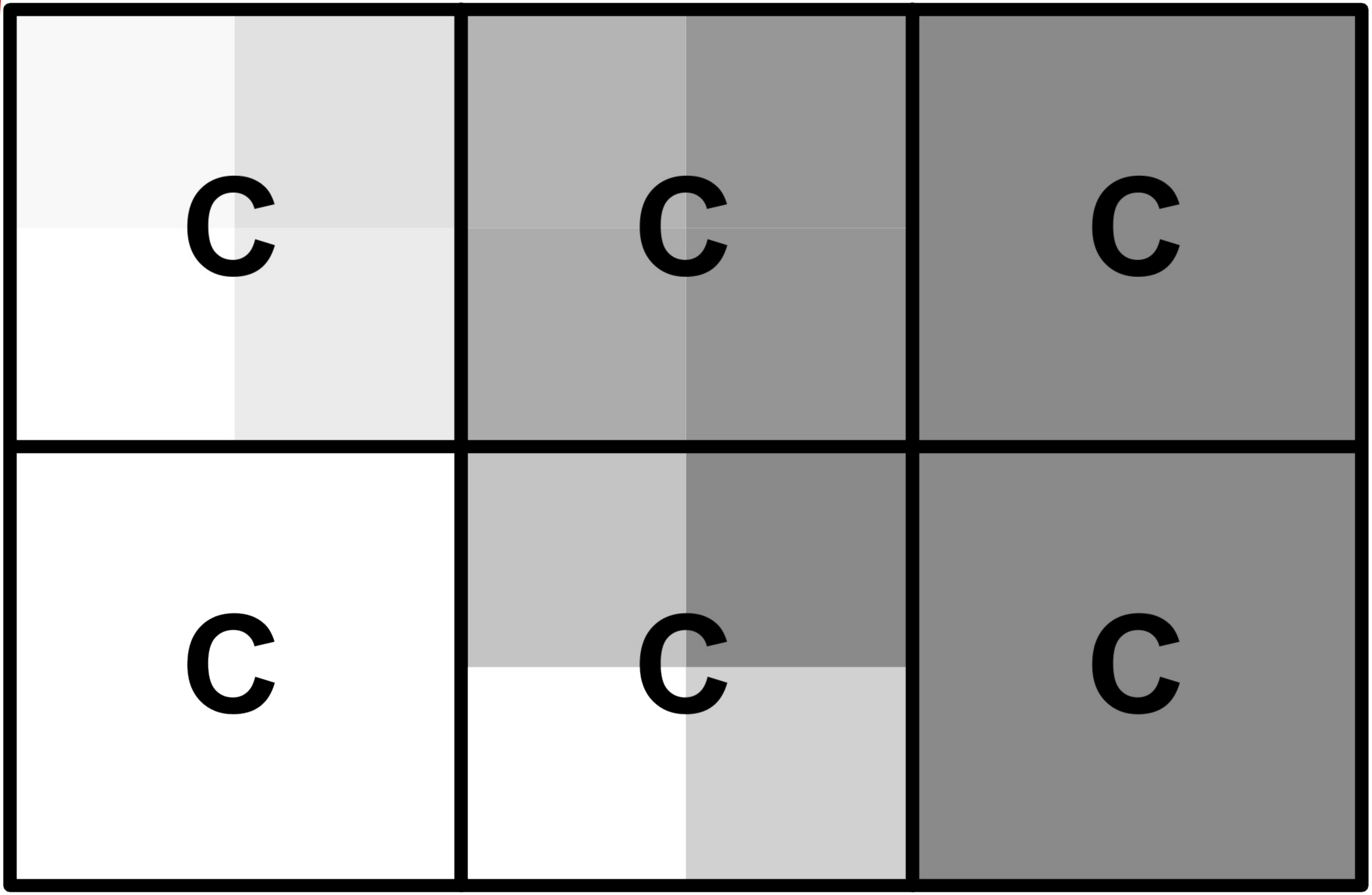
Shader complexity increasing

Rasterize, raytrace, or both?

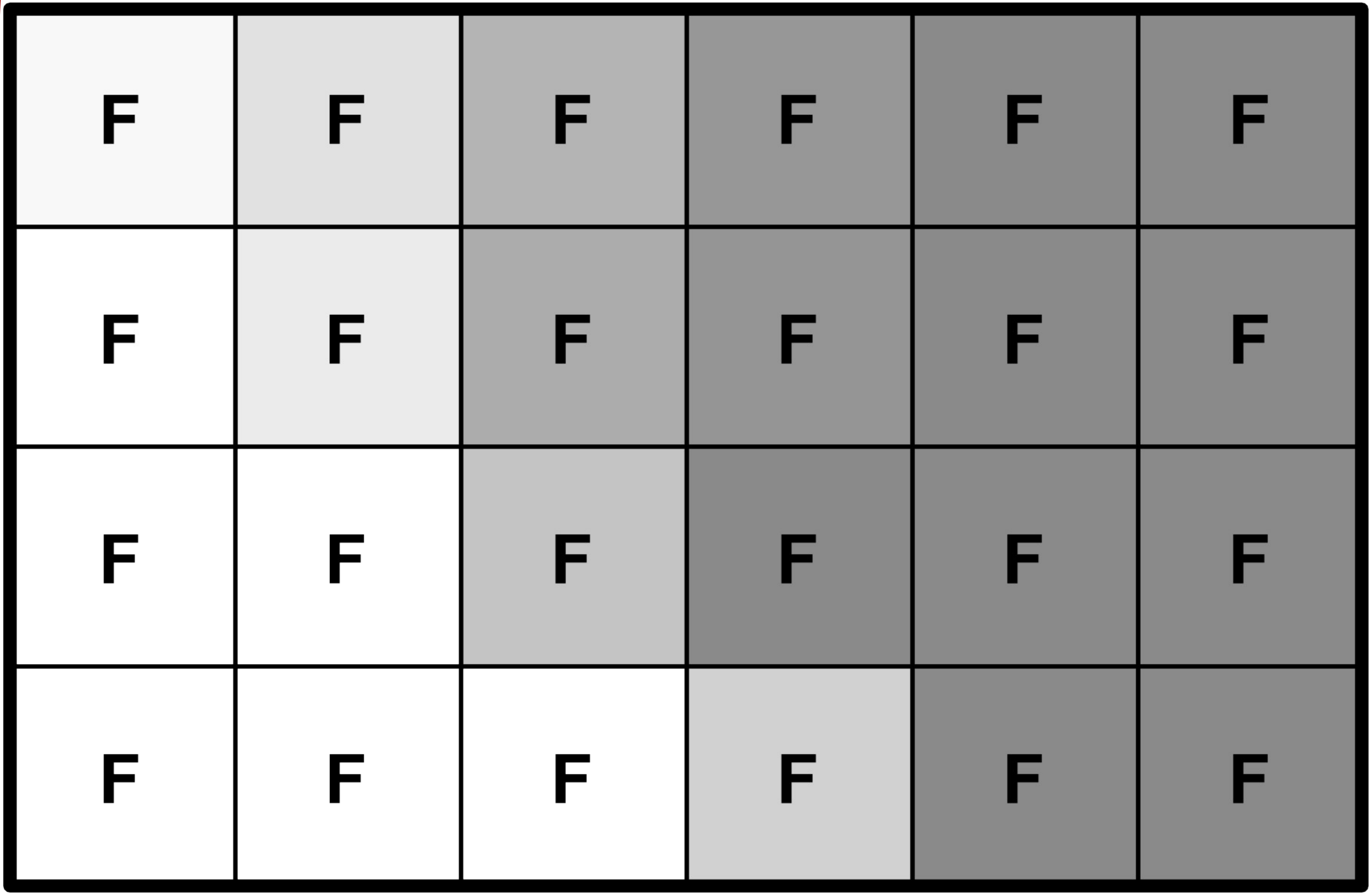
Battery-constrained mobile GPUs

Yet still want cinema experience

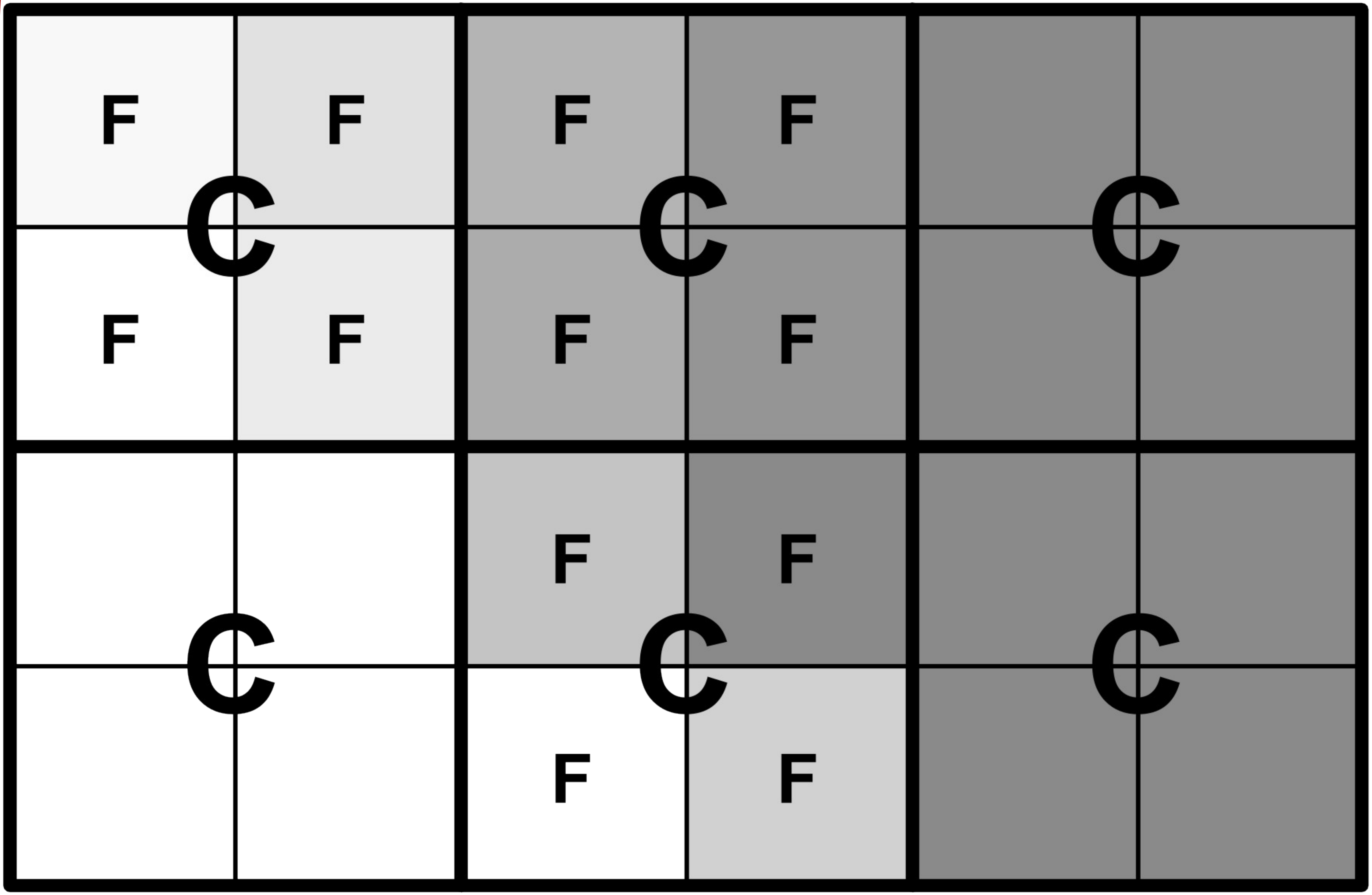
In the beginning, coarse pixels



Fine pixels: clean edges, but slow!



Adaptive pyramidal rendering



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

GPU Pixel Shader

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

Error Metric

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

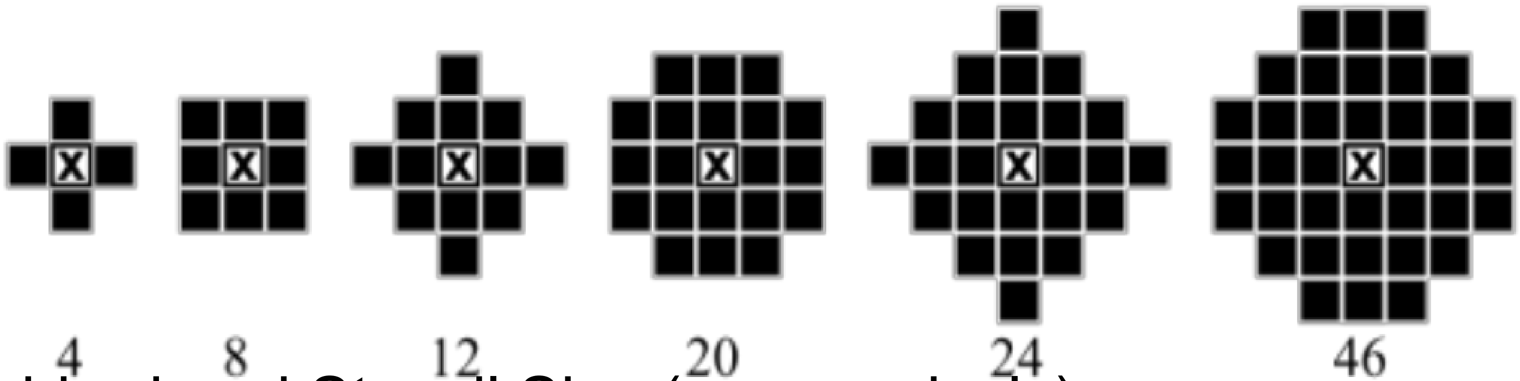
The error metric examines the coarse pixels to decide between interpolation and sampling to make fine pixels.

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



Neighborhood Stencil Size (coarse pixels) ->

Interpolating Polynomial order ->

	4	8	12	20	24	46
P^1						
P^3						
P^5						
P^9						

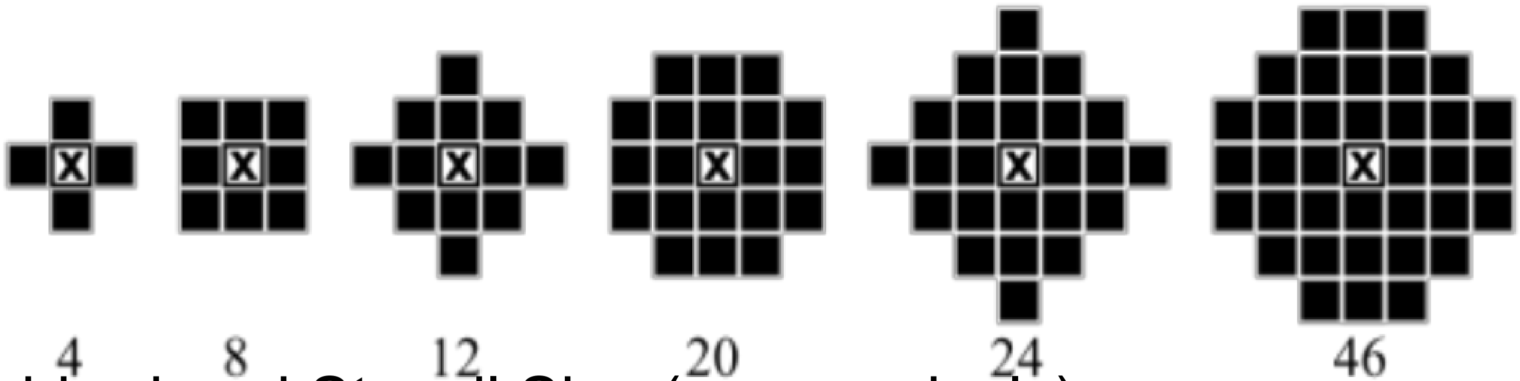
Constant

Linear

Expectation: more is better!

Quadratic

Metric stencil and order vs error



Neighborhood Stencil Size (coarse pixels) ->

Interpolating Polynomial order ->

	4	8	12	20	24	46
P^1	2.31%	2.32%	2.36%	2.48%	2.53%	2.63%
P^3	2.31%	2.26%	2.29%	2.34%	2.35%	2.41%
P^5	*[4]	2.54%	2.25%	2.31%	2.31%	2.34%
P^9	*	*	2.28%	2.30%	2.29%	2.30%

Error Metric Surprises

The effect of the particular polynomial and stencil is <u>weak</u>: if the image is smooth, any reasonable metric correctly shows it should be interpolated.

Large stencils cause "false positives", expanding detail <u>far</u> from the true cause.

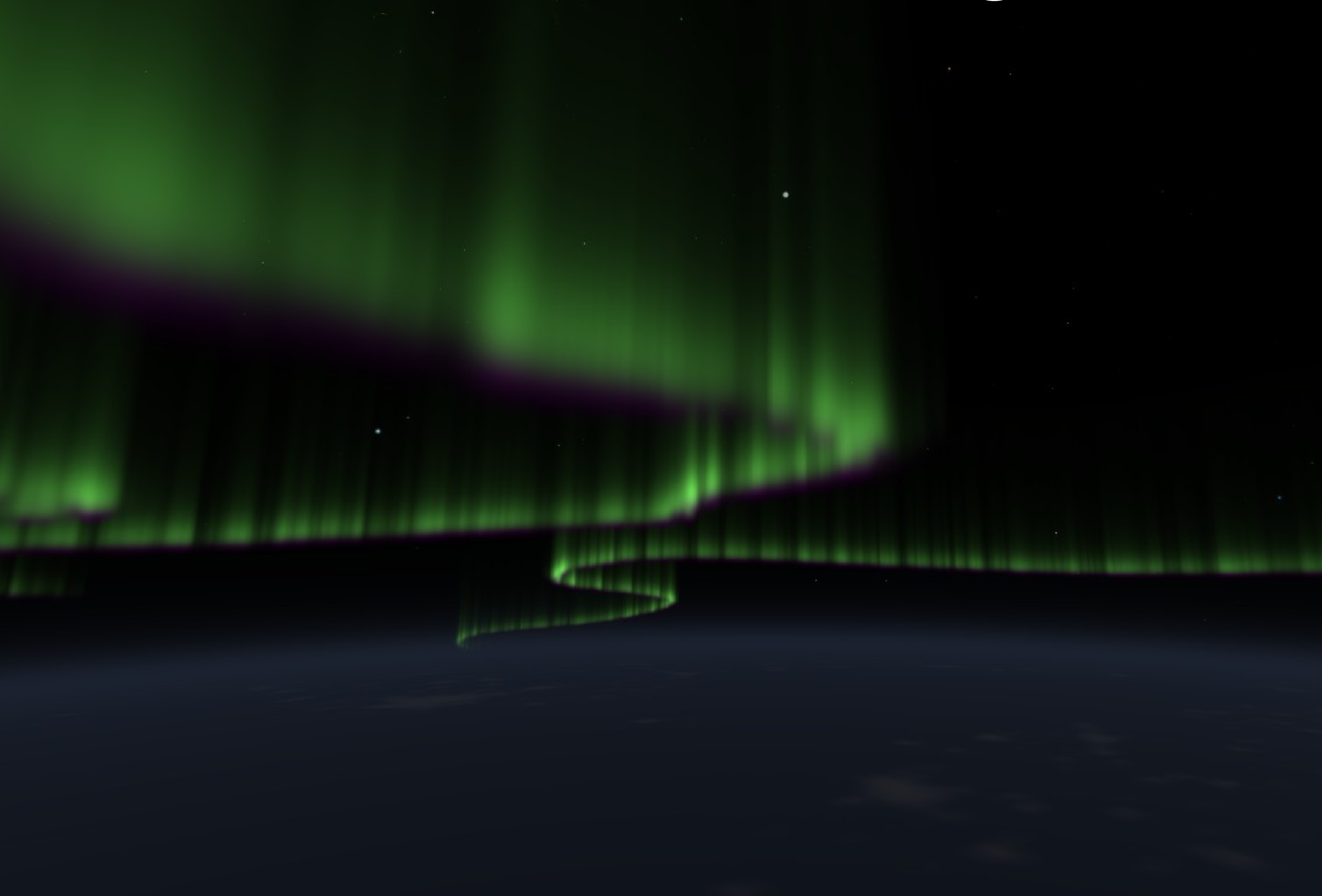
High order polynomials cause "false negatives", assuming a <u>smooth</u> high-order curve where something more subtle is happening.

Averaging is 10% better than maximum error.

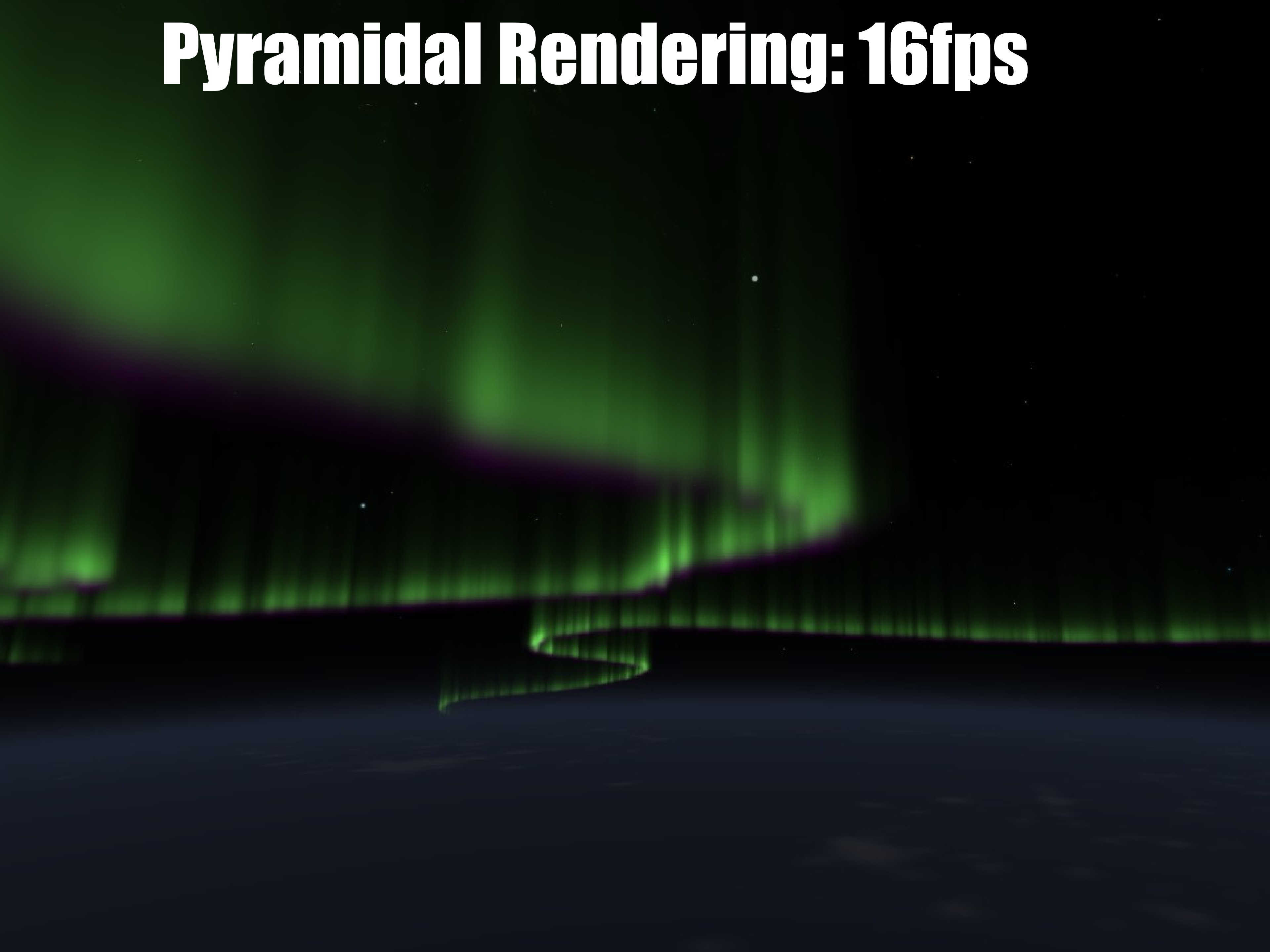
Some common choices, like "contrast metric" [Mitchell 87], are over 30% worse than P1-4!

Examples of
Adaptive Pyramidal Rendering

Aurora Volume Rendering: 5fps



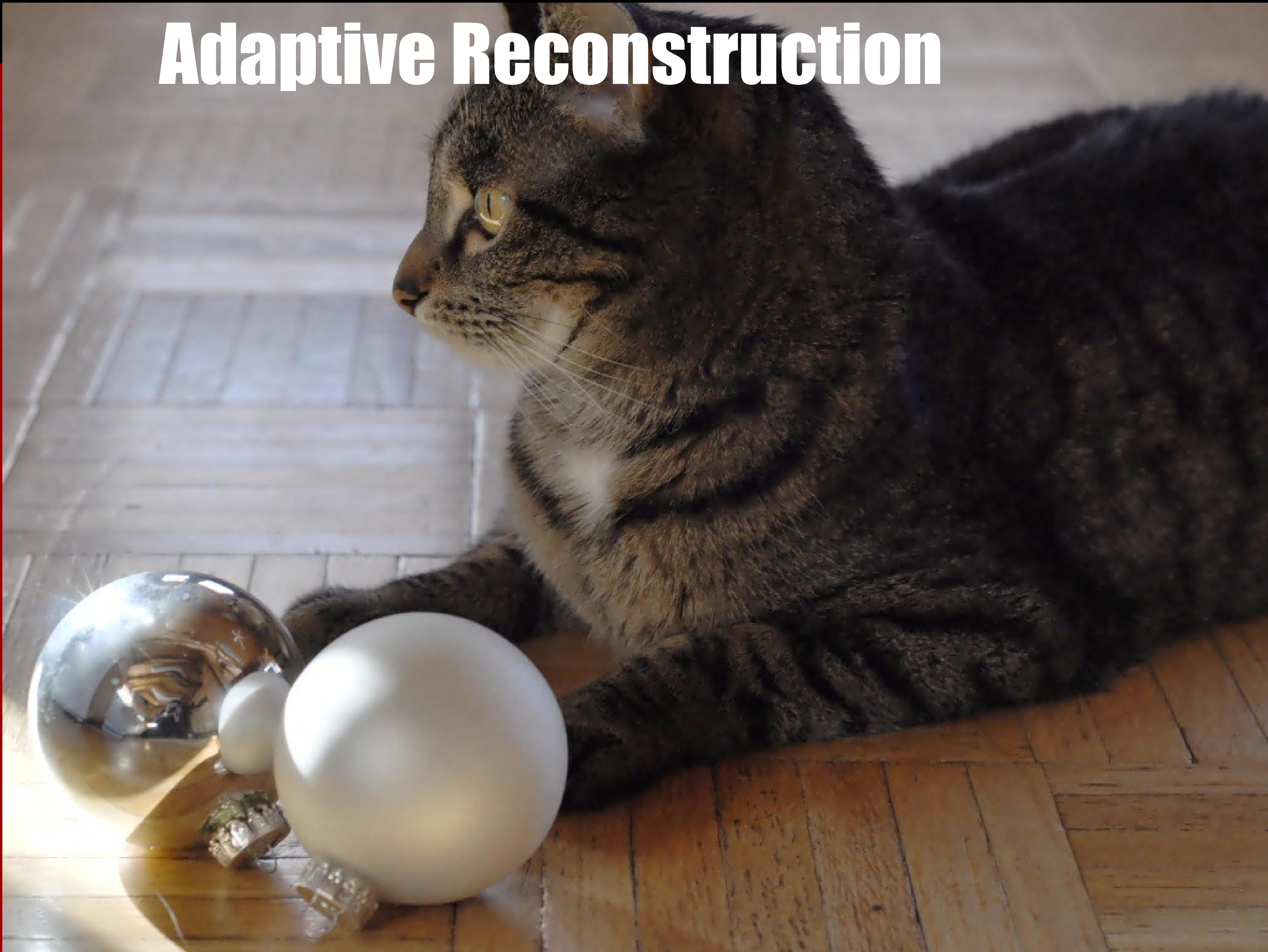
Pyramidal Rendering: 16fps



Coarse Rendering: 40fps

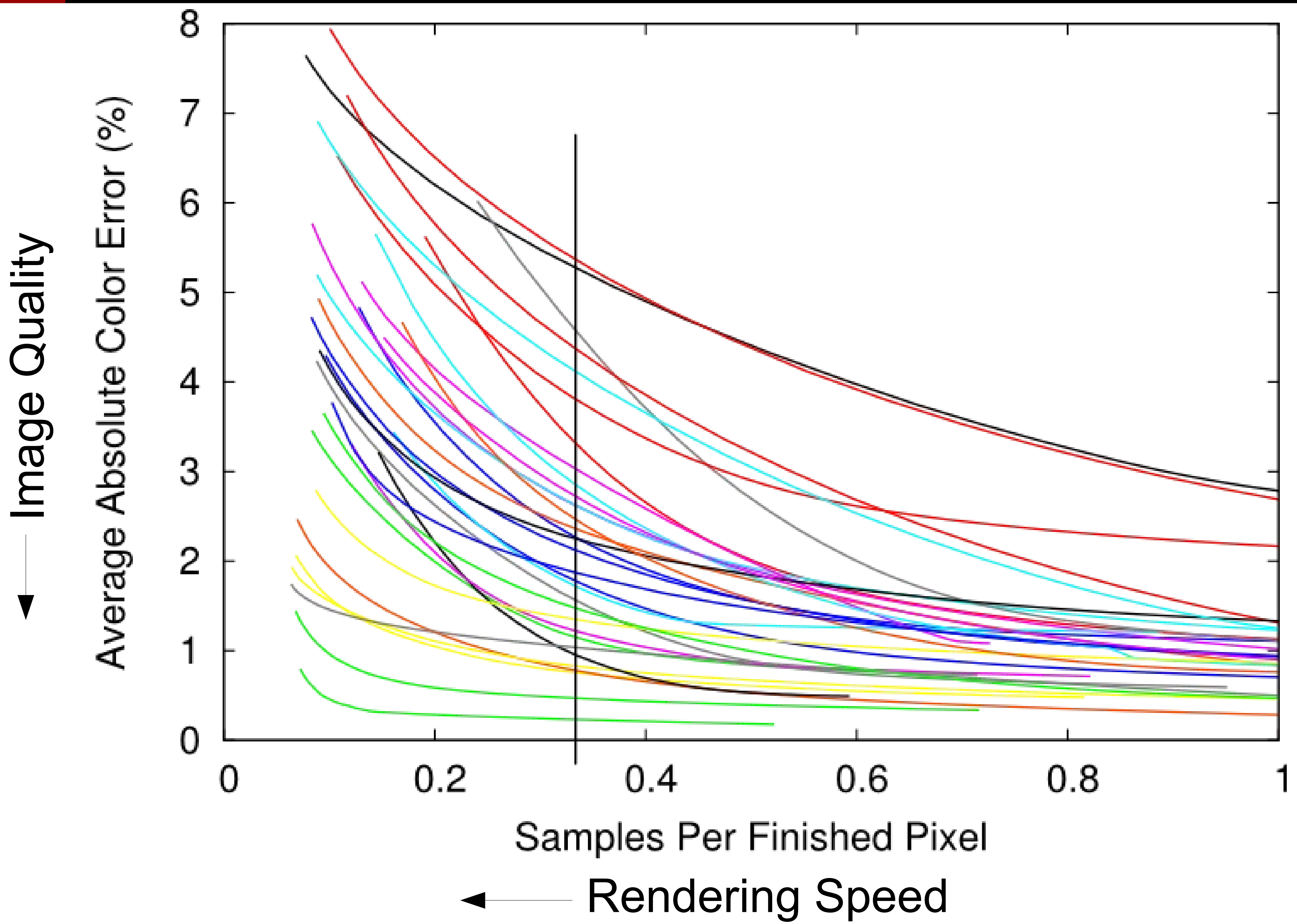
Original Image

Adaptive Reconstruction

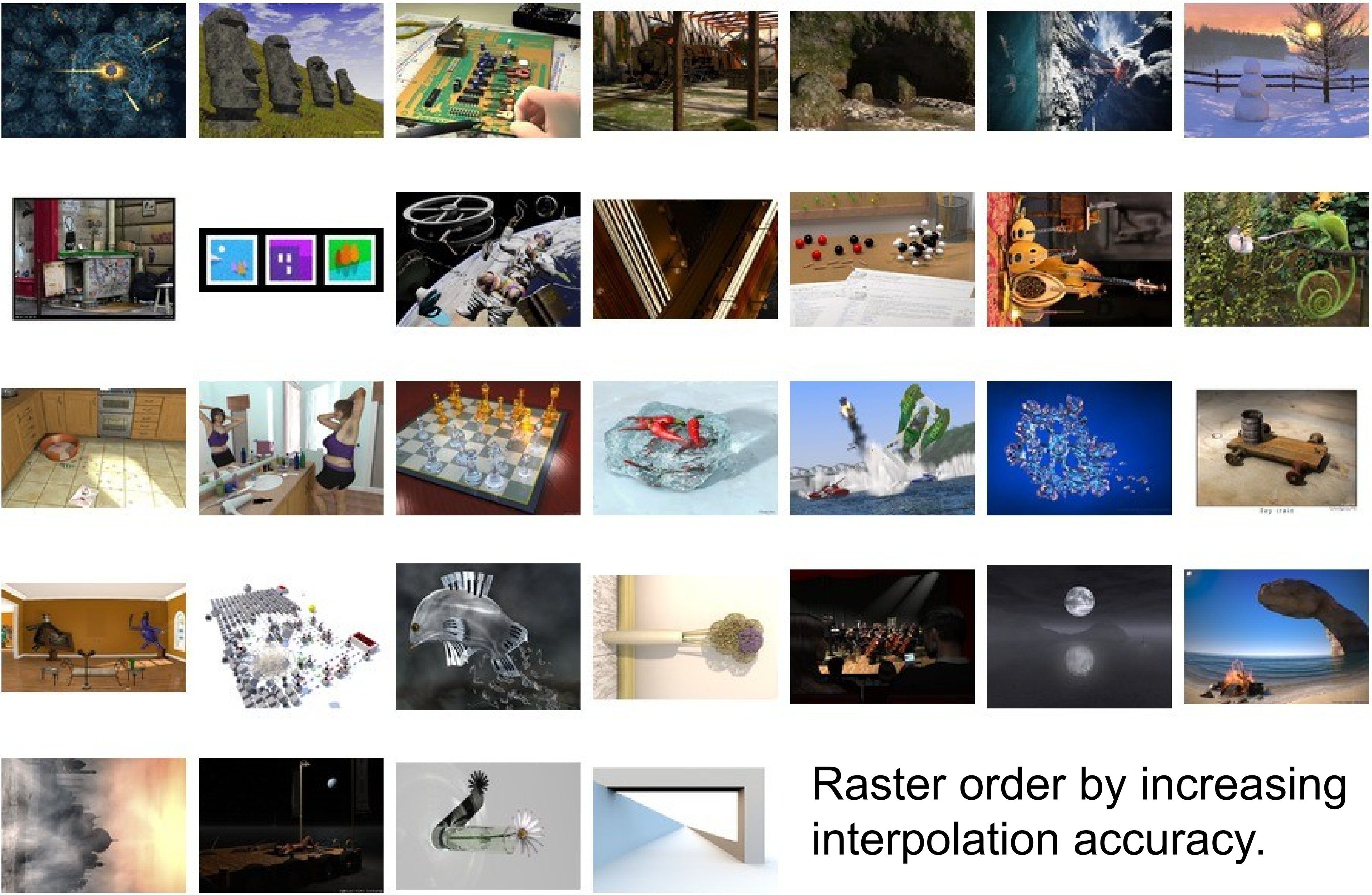


Black = fine, White = coarse

Speed vs Image Quality Tradeoff



Benchmark Image Bank

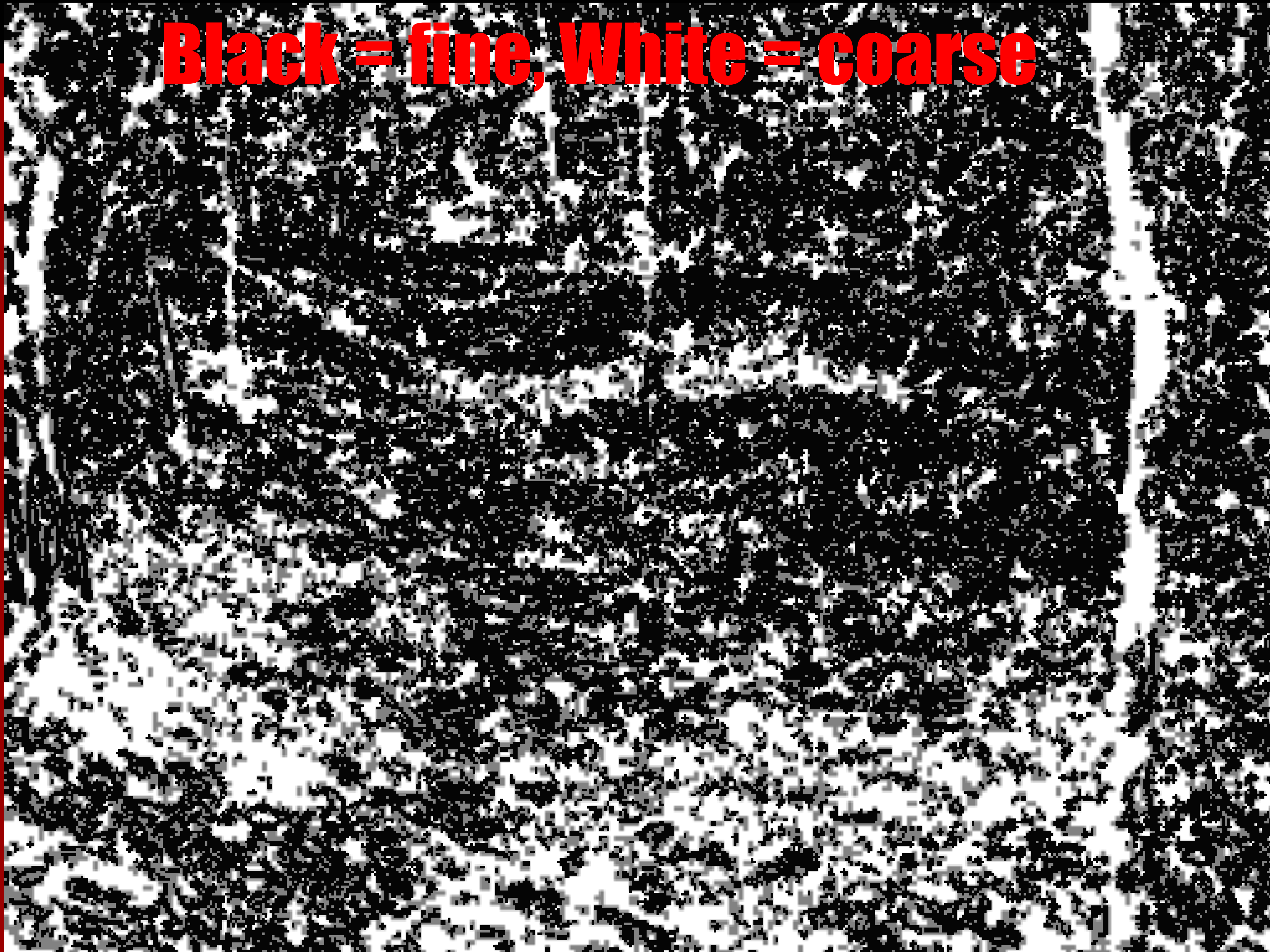


Raster order by increasing interpolation accuracy.

Internet Ray Tracing Contest winners & honorable mention images


Original Image

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?