SIGGRAPH2009 NEW ORLEANS

Shiny, Blurry Things

Justin Hensley

Office of the CTO
Advanced Micro Devices

justin.hensley@amd.com

Outline

- Approximate Image Based Lighting
- Depth of Field *(short-short version)*
- Experiences with Real-Time Ray Tracing
- Demos

Approximate Image Based Lighting

SIGGRAPH Asia'08 sketch

New Orleans, LA (August 2009)

AMD
The future is fusion

Overview & Benefits

- Summed-Area Tables
  - What they are, and how to construct them
- Higher-order summed-area tables
  - "Repeated Integration" [heckbert86]
- Approximating BRDFs
- **Key Benefits of the technique**
  - All data computed on the fly - no pre-processing needed
  - Filters can be varied spatially per rendered pixel
  - Approximate image-based lighting
  - All lighting in example images computed completely by filtering HDR environment maps

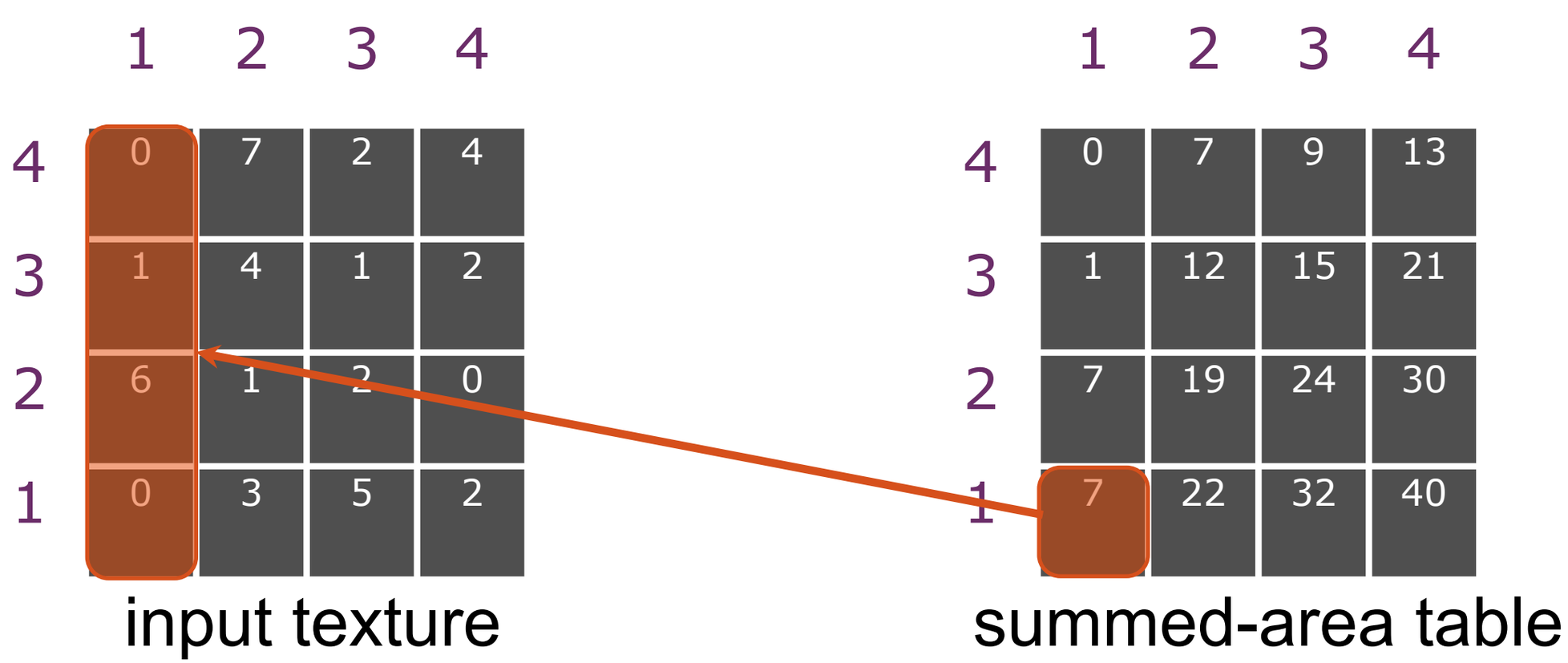
Summed-Area Tables (SATs)

- For a left-handed coordinate system, each element S_{mn} of a summed-area table S contains the sum of all elements above and to the left of the original table/texture T [Crow84]

$$S_{mn} = \sum_{i=1}^{m} \sum_{j=1}^{n} t_{ij}$$

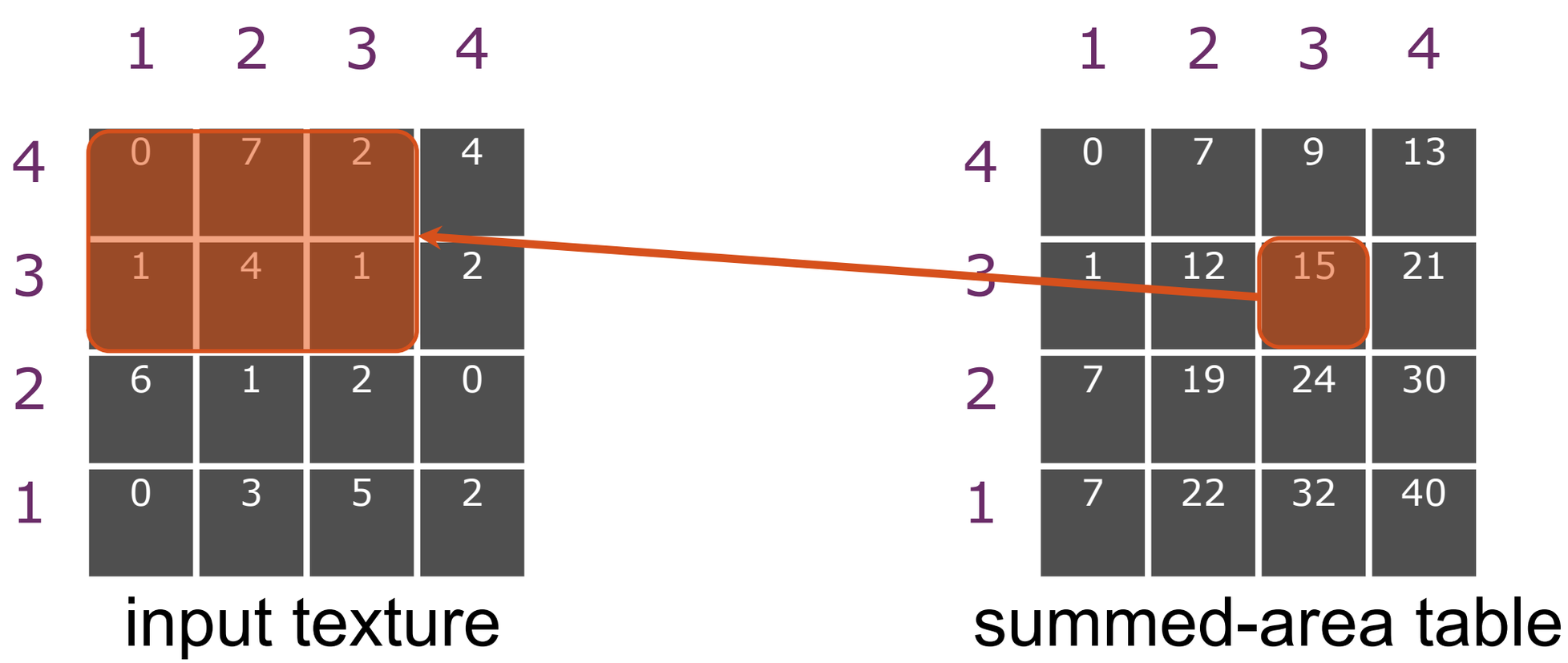
Summed-Area Tables (SATs)

- For a left-handed coordinate system, each element S_{mn} of a summed-area table ***S*** contains the sum of all elements above and to the left of the original table/texture ***T*** [Crow84]



Summed-Area Tables (SATs)

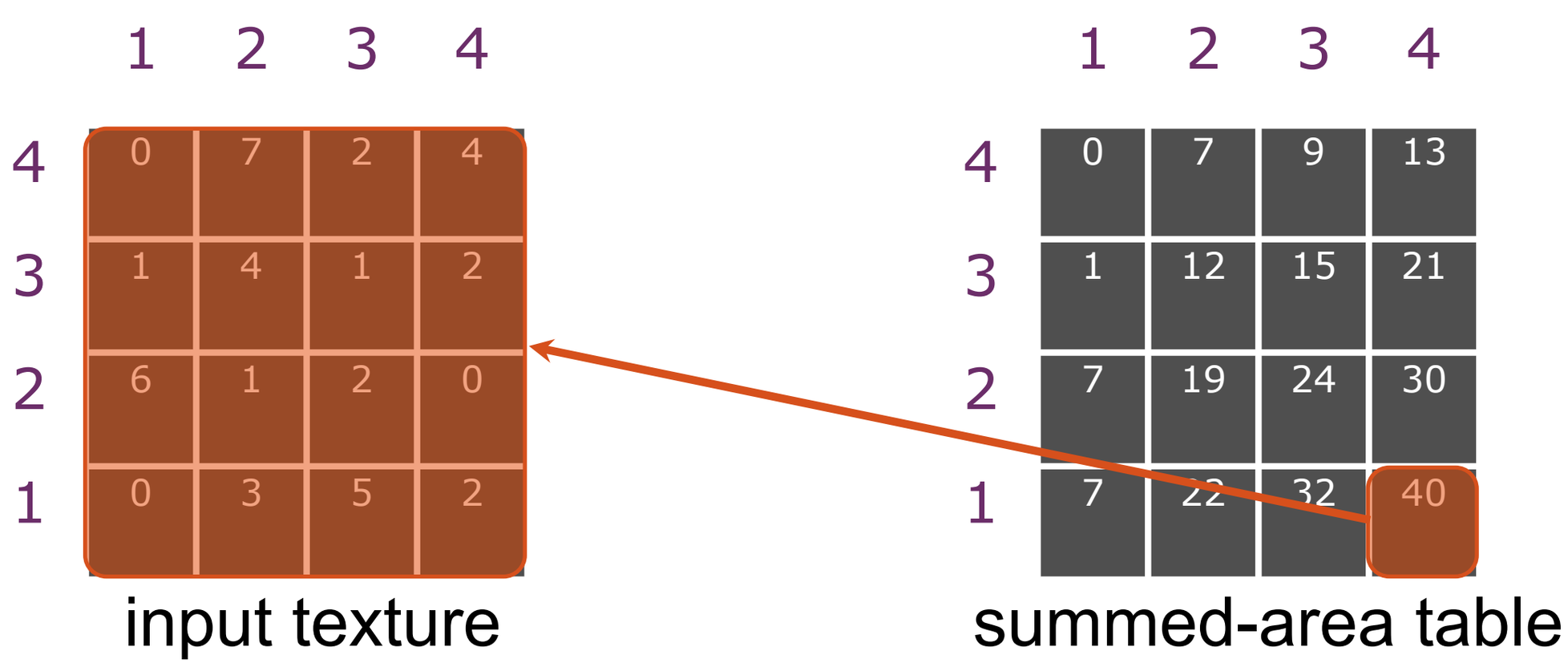
- For a left-handed coordinate system, each element S_{mn} of a summed-area table ***S*** contains the sum of all elements above and to the left of the original table/texture ***T*** [Crow84]



AMD
The future is fusion

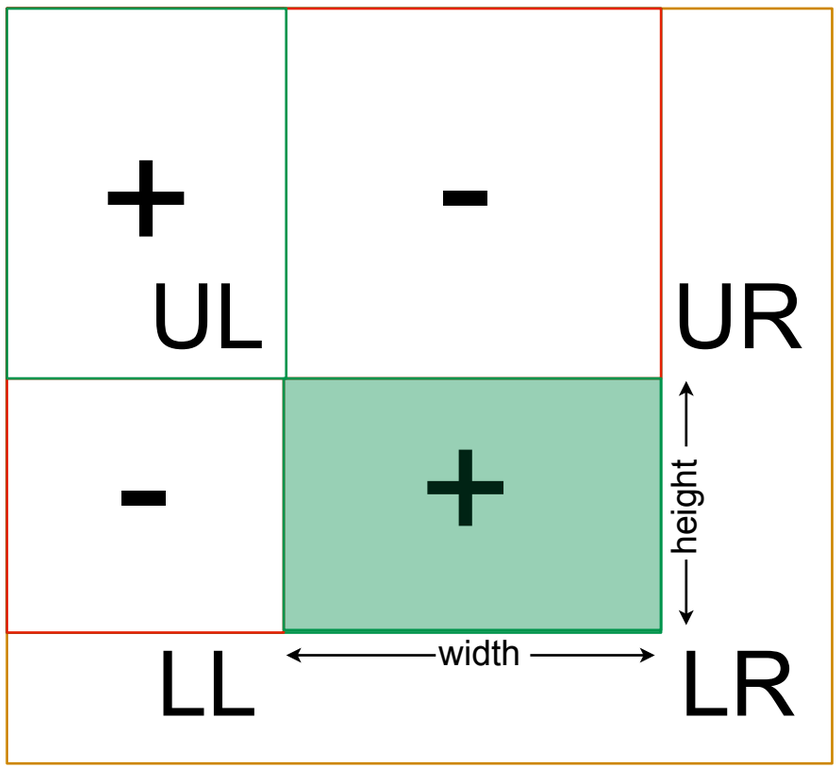
Summed-Area Tables (SATs)

- For a left-handed coordinate system, each element S_{mn} of a summed-area table S contains the sum of all elements above and to the left of the original table/texture T [Crow84]



Using a Summed-Area Table

- Summed-area tables enable the averaging rectangular regions of pixel with a constant number of reads



$$average = \frac{LR - LL - UR + UL}{width * height}$$

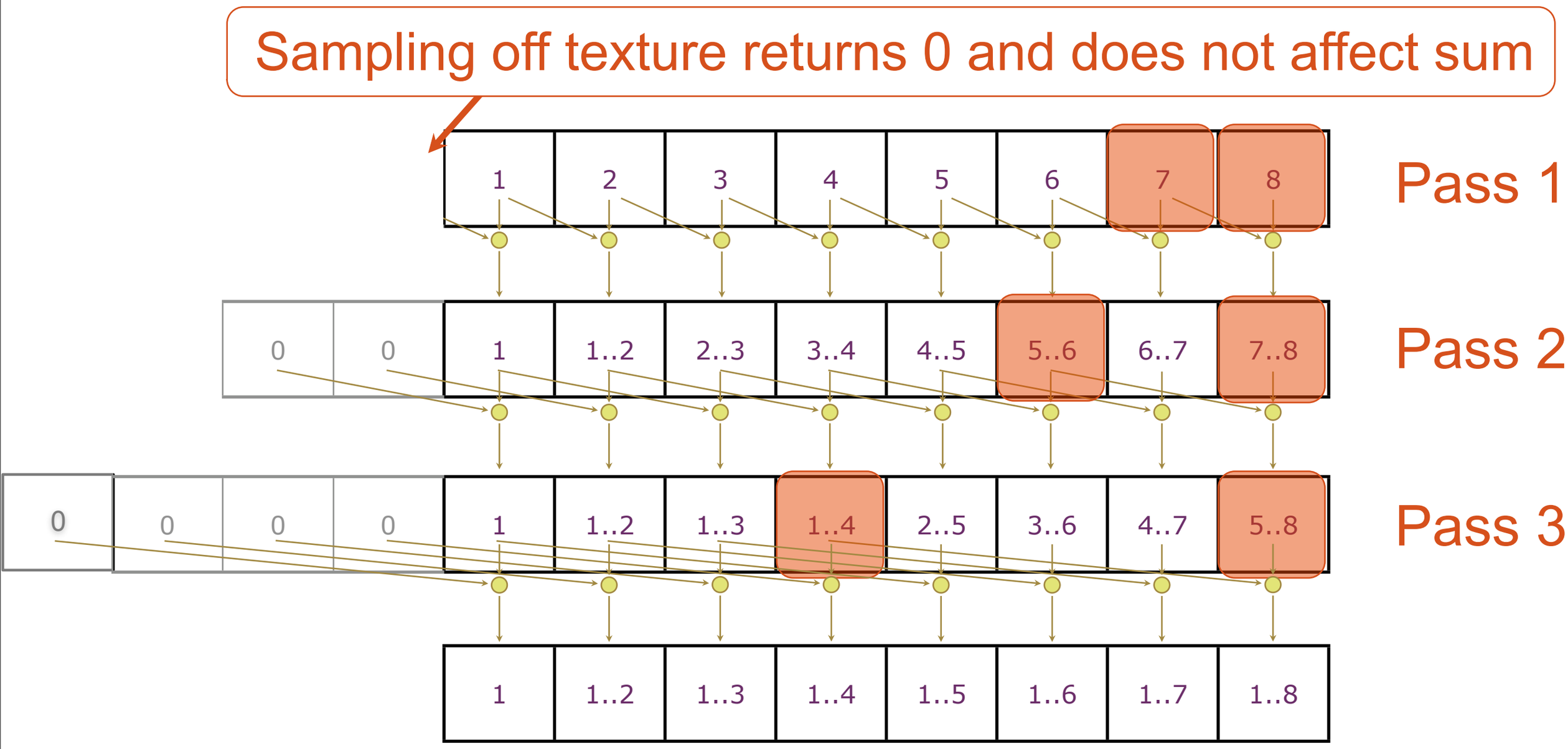
Efficient SAT Creation - Graphics APIs

- Summed-area table construction is linear and separable
  - –Decompose into horizontal and vertical phase each with $log_2(texture\ size)$ passes
- Borrow technique from high performance computing - **recursive doubling**
- Each pass adds two elements from previous pass, *ping-ponging* between render targets. Horizontal pass:

$$P_i(x,y) = P_{i-1}(x,y) + P_{i-1}(x - 2^{passindex}, y)$$

AMD
The future is fusion

Horizontal Phase (1D Example)



Saving Render Passes

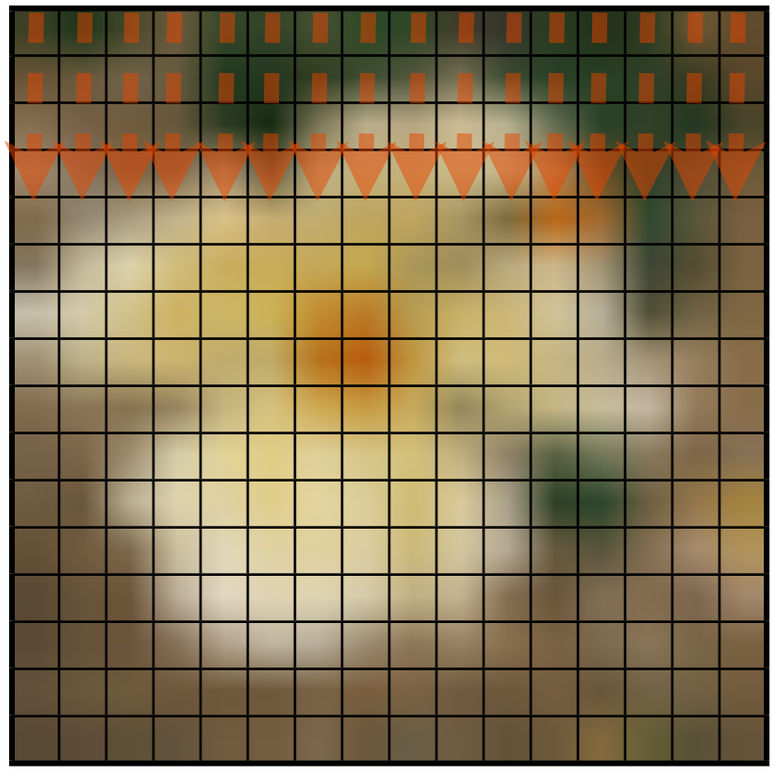
- Two samples per pass requires 16 passes for a 256x256 texture

→ $2*log_2(256)$

- Reduce number of passes by adding more samples per pass
  - **passes** $=\lceil log_{\#samples}(width)+log_{\#samples}(height)) \rceil$
- Only need 4 passes to convert a 256x256 texture into a summed-area table if 16 samples / pass used
  - Trade extra work and memory pressure versus reducing number of context switches

SAT Creation via Compute API (OpenCL or DX11)

Image (*logical*)

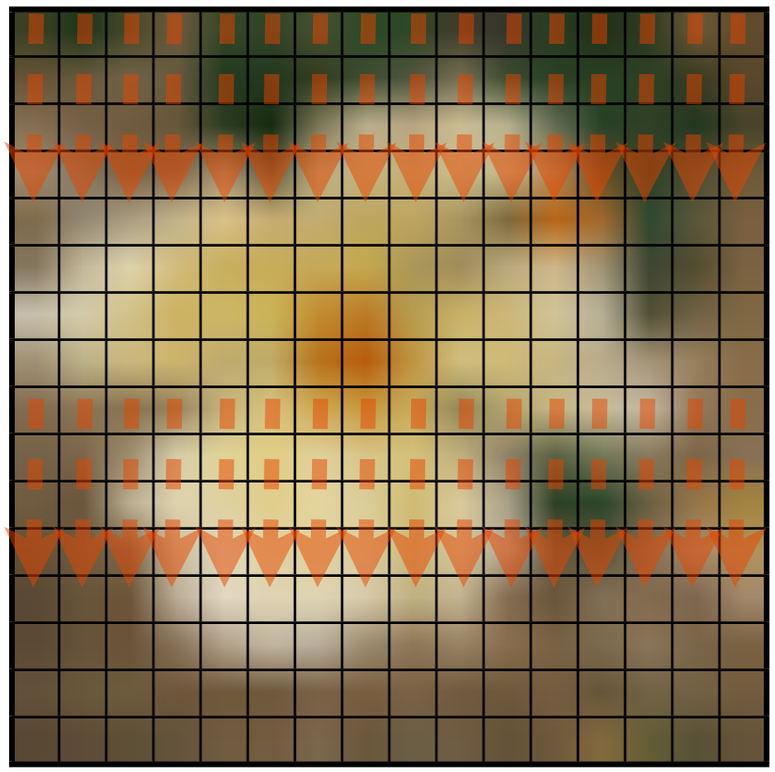


```
__kernel void verticalSAT( __global float *out,
                           __global float *in,
                           int width, int height  )
{
    const int idx = get_global_id( 0 );

    int i, index = idx;
    float sum = 0.0;

    for( i = 0;i < height;i++ )
    {
        sum = sum + in[index];
        out[index] = sum;
        index = index + width;
    }
}
```

Linear Buffers in Memory (*reality*)

SAT Creation via Compute API (OpenCL or DX11)

Image (*logical*)

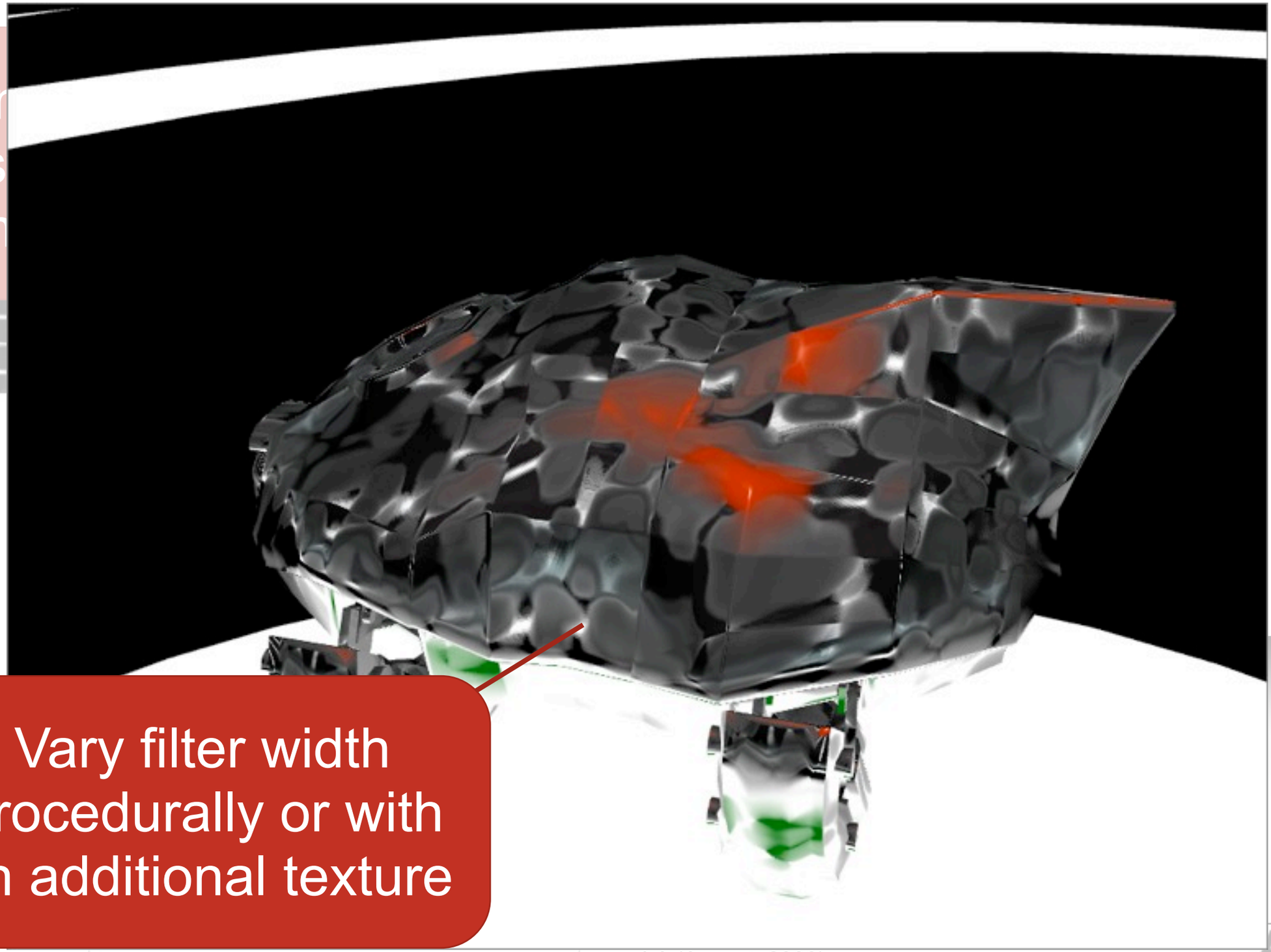


- Horizontal SAT very inefficient for linear buffers
  - Tiled buffers are okay
  - Transpose image to use vertical
- For small images not enough waves launched
  - Perform partial scans on image
  - Resolve with final pass

Linear Buffers in Memory (*reality*)

Dynamic Glossy Environmental Reflections [hensley05]

Sour
but is
r



Vary filter width procedurally or with an additional texture

Higher-order Summed-Area Tables

- Summed-area table → integral of image
- Box filter sampling → derivative of filter kernel

$$f \otimes g = f' \otimes \int g$$

- Summed-area tables extend to higher-order filters [Heckbert86]

$$f \otimes g = f'' \otimes \iint g$$

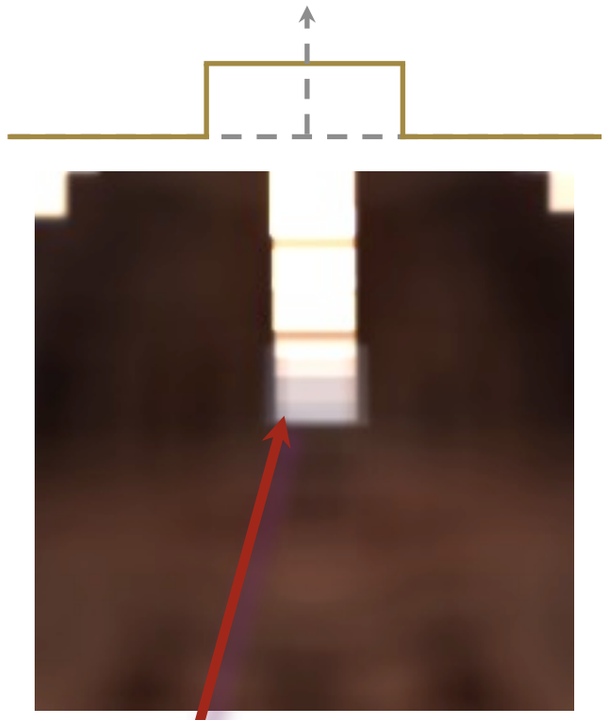
$$f \otimes g = f'^{n} \otimes \int^{n} g$$

- Precision becomes a problem
  - —Larger filters average error out

AMD
The future is fusion

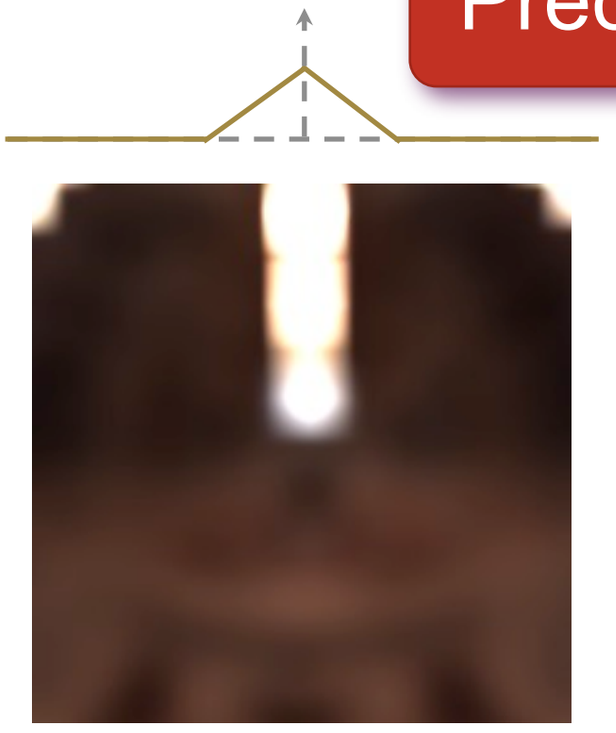
Filtering HDR images

original



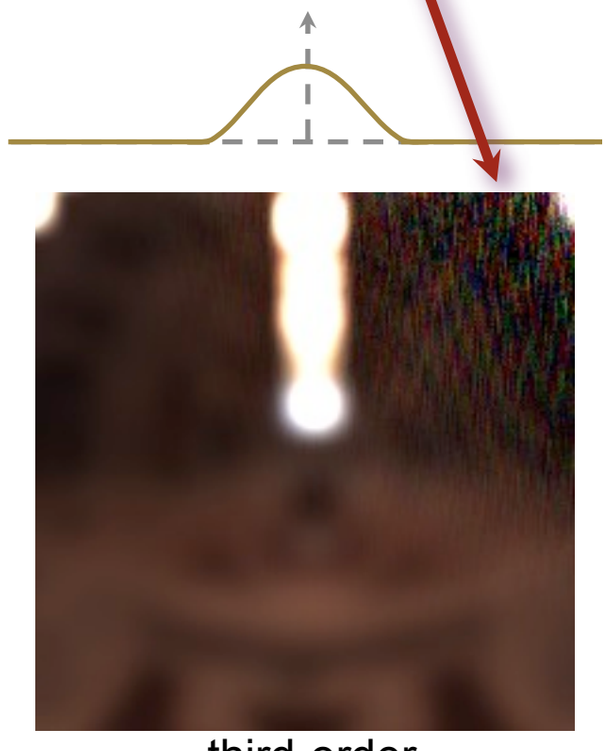
first-order
(box filter)

HDR data “blows out” filter



second-order
(bartlet filter)

Precision error

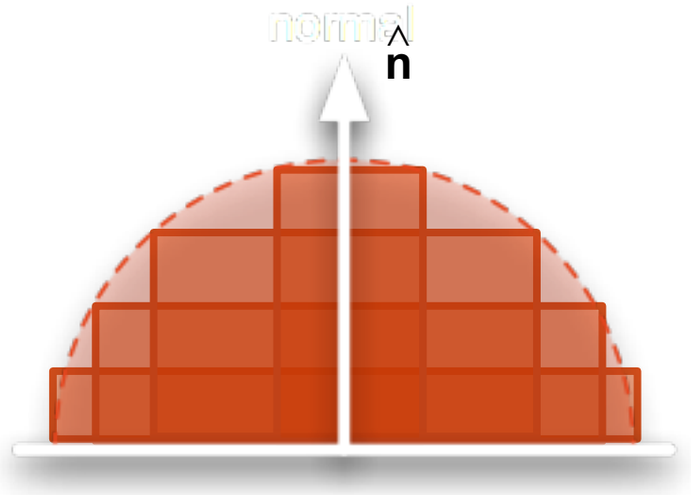


third-order
(quadric filter)

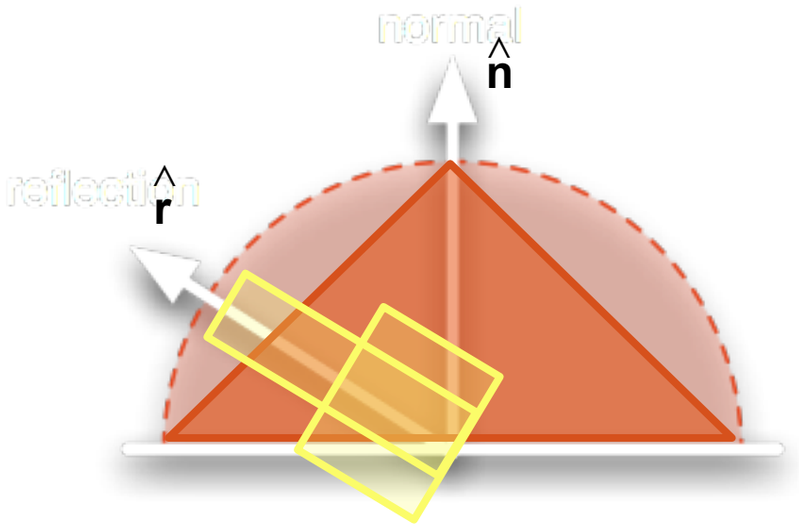
Lightprobe data courtesy of Paul Debevec

Approximating BRDFs

- Average several SAT samples to approximate smoother filter kernels

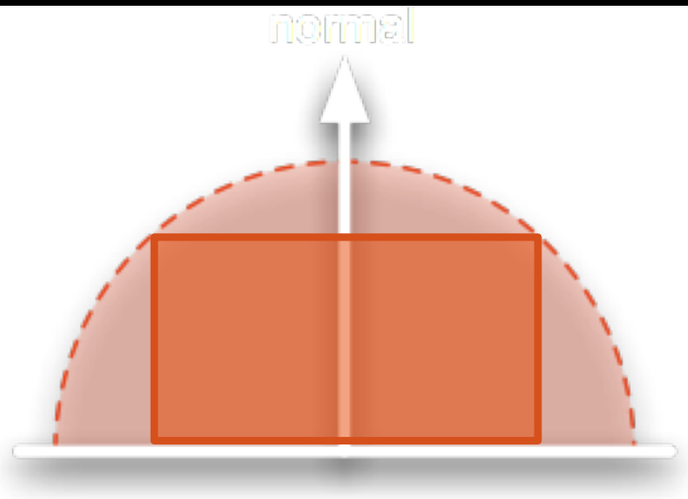


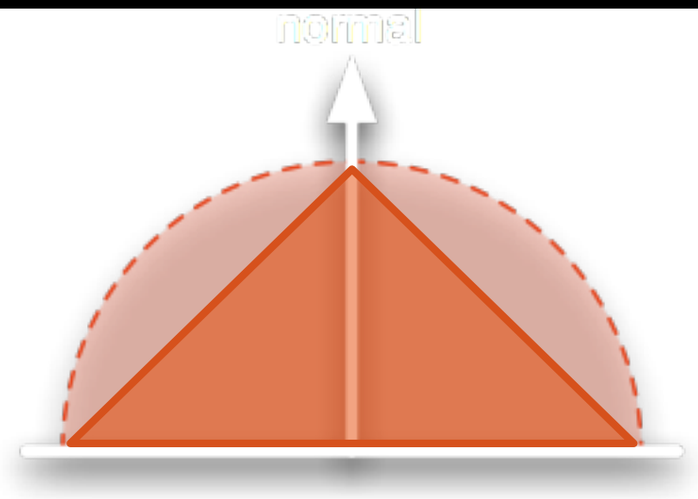
- Approximate a Phong BRDF by combining samples from the normal direction and the reflection direction



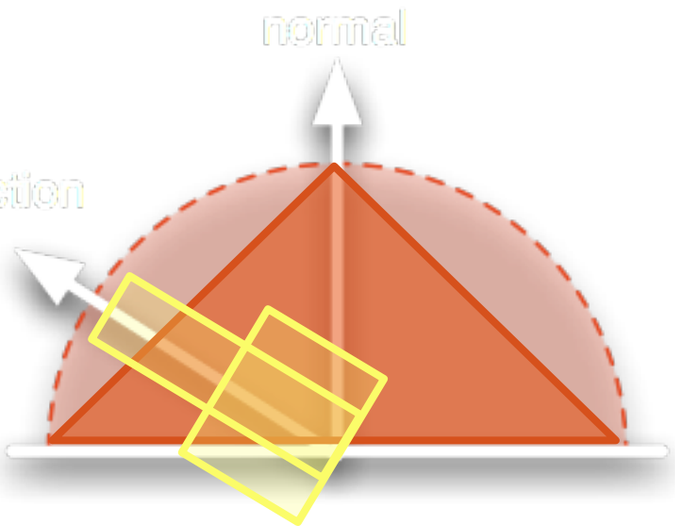
- **Use higher-order SATs to get better filtering**
  - –Precision can be an issue for higher-order SATs

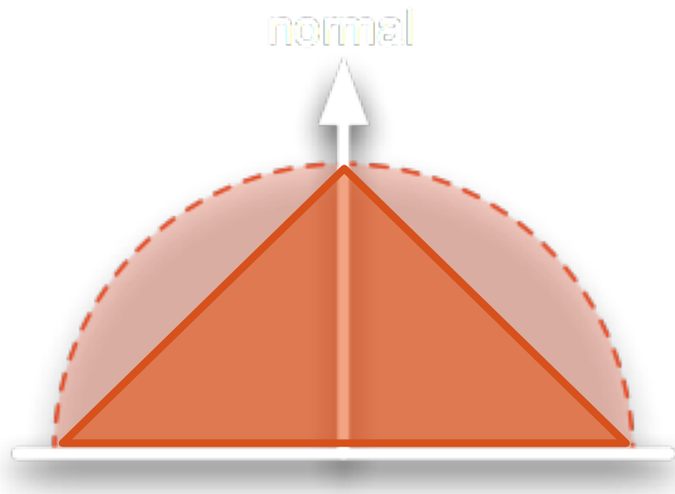
Diffuse Approximation - single filter





Approximate Image-Based Lighting





Combine with ambient occlusion

New Orleans, LA (August 2009)

AMD
The future is fusion

Video

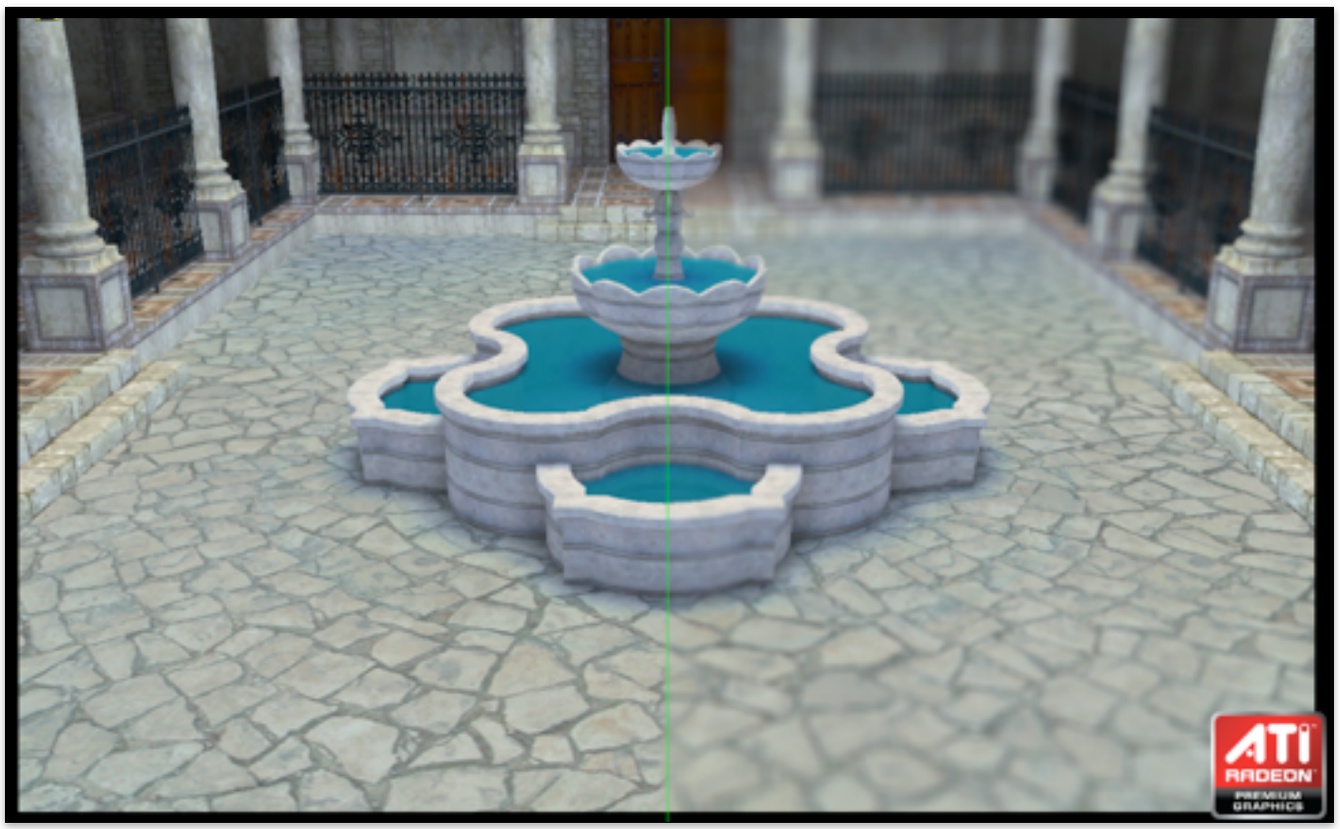
performance: 40 fps when recomputing SATs every frame on an **ATI X1900XT**
120+ fps when environment map is static

This is a GPU over two generations old!

New Orleans, LA (August 2009)

AMD
The future is fusion

Depth of Field

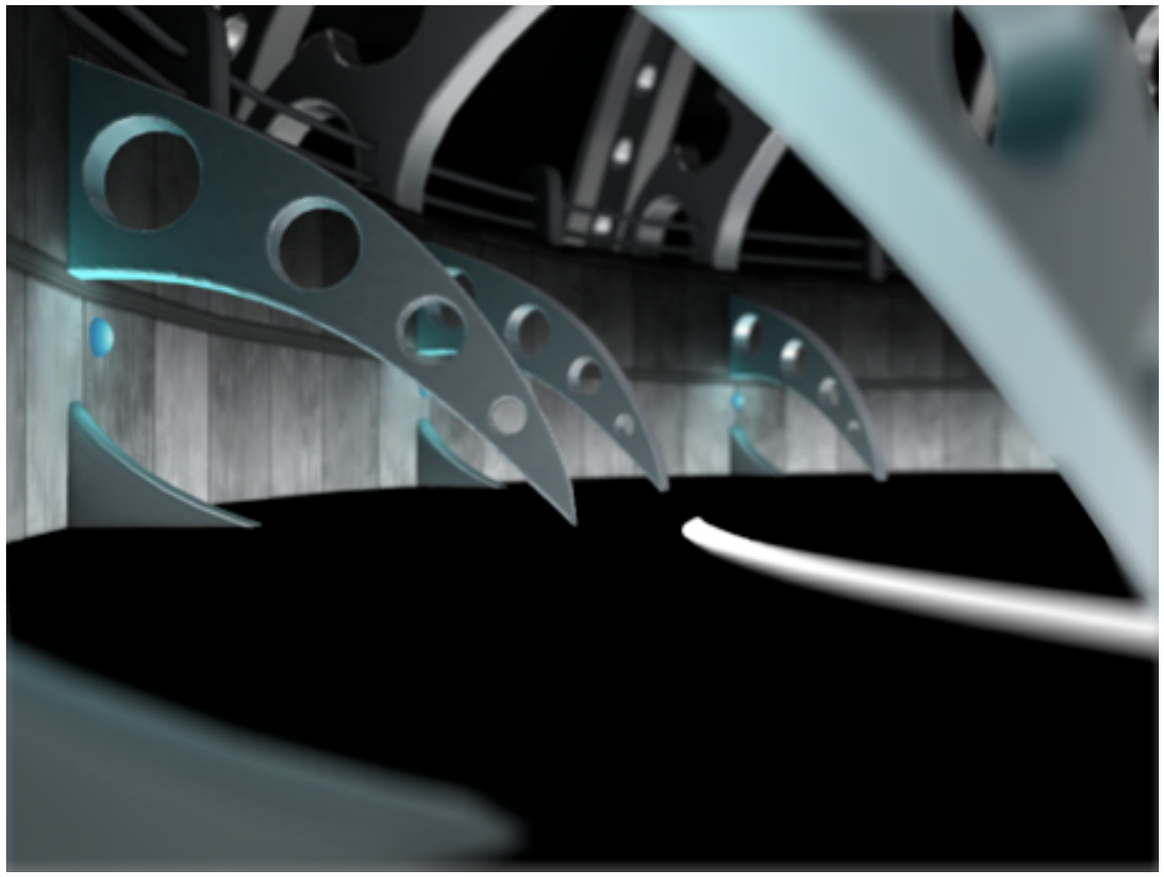


New Orleans, LA (August 2009)

AMD
The future is fusion

“Gather” based Depth of Field

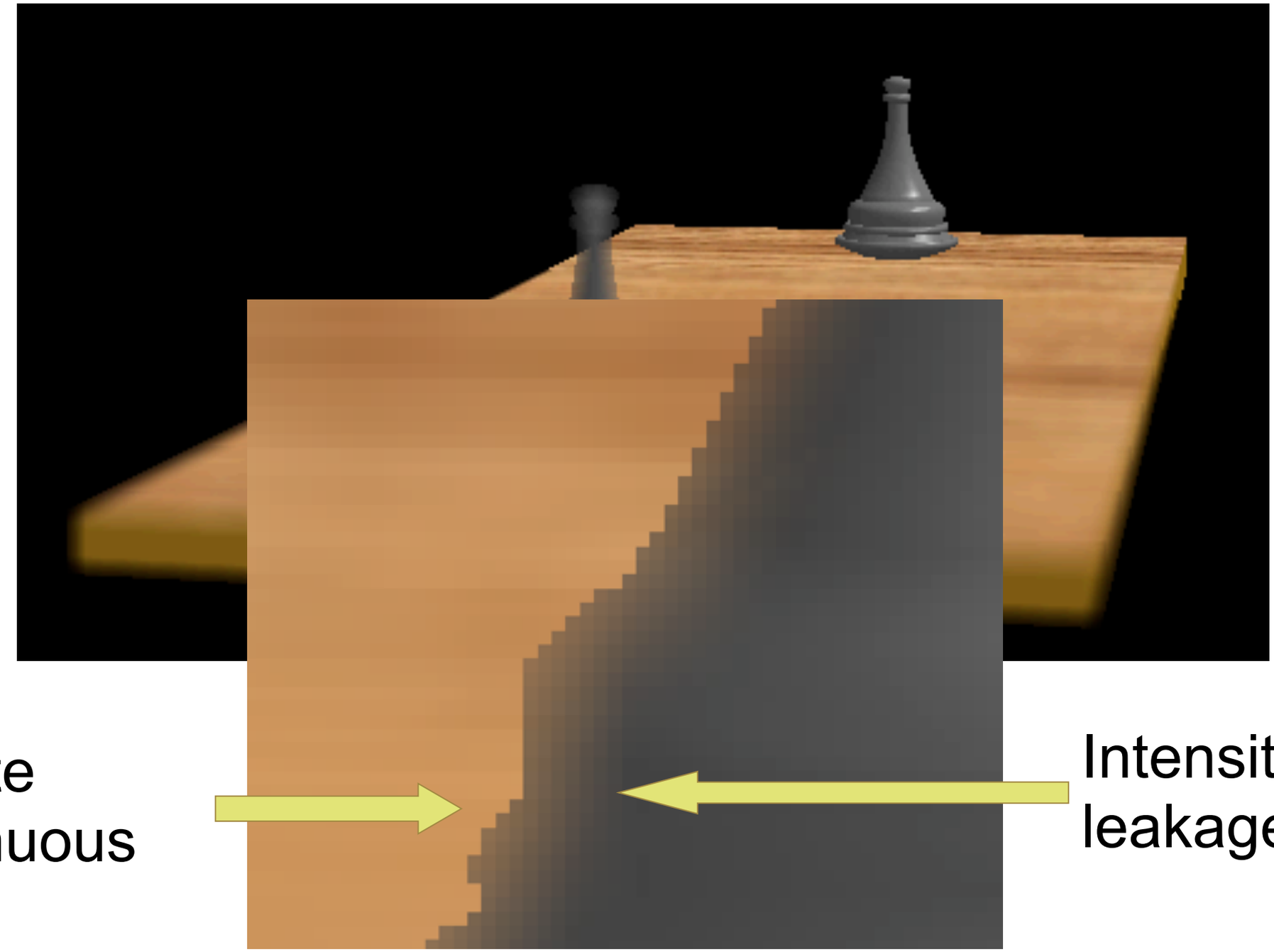
- Render frame to backbuffer
- Compute SAT of framebuffer
- Sample SAT with filter size modulated by z-buffer
  - Approximates cameras circle of confusion



Definition of Gathering

- Filtering is usually considered a process of taking averages
  - Weighted average of adjacent pixels taken to produce a blurred pixel
- We call this approach "gathering"
  - Many pixels are gathered together to produce one output
- Another approach, "spreading" that is better for depth of field
  - Similar to splatting, but constant time with size

Gather Depth of Field Errors



Silhouette discontinuous

Intensity leakage

Filter Spreading

UC Berkeley Tech Report - Kosloff09

- Spreading filters operate as follows:

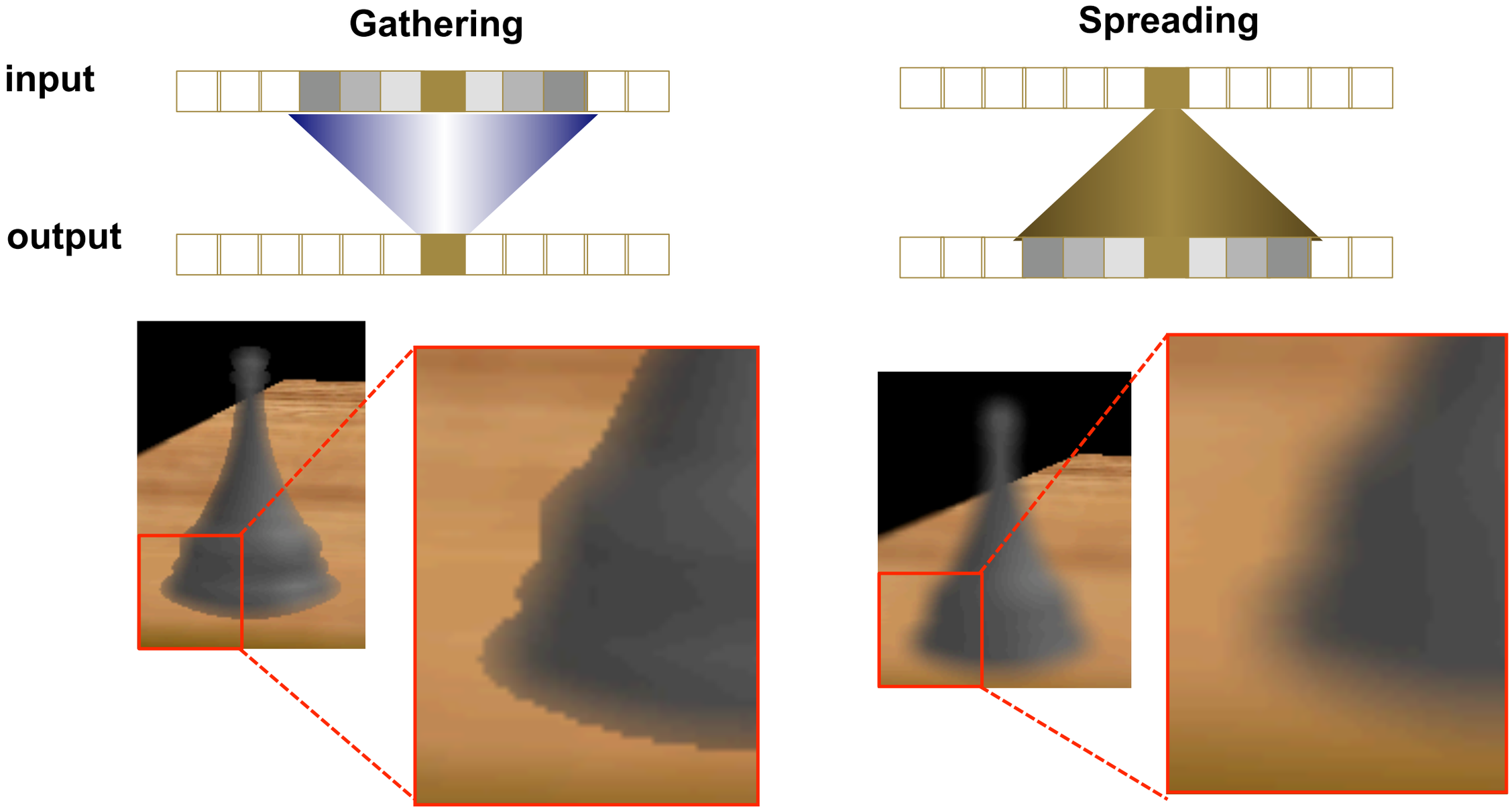
  For each pixel in the input image filter

  Accumulate filter kernel in the output image

  ("spread" pixel data, modulated by kernel, to neighborhood)

- Spreading more accurately simulates the physics of depth of field

Gathering vs. Spreading

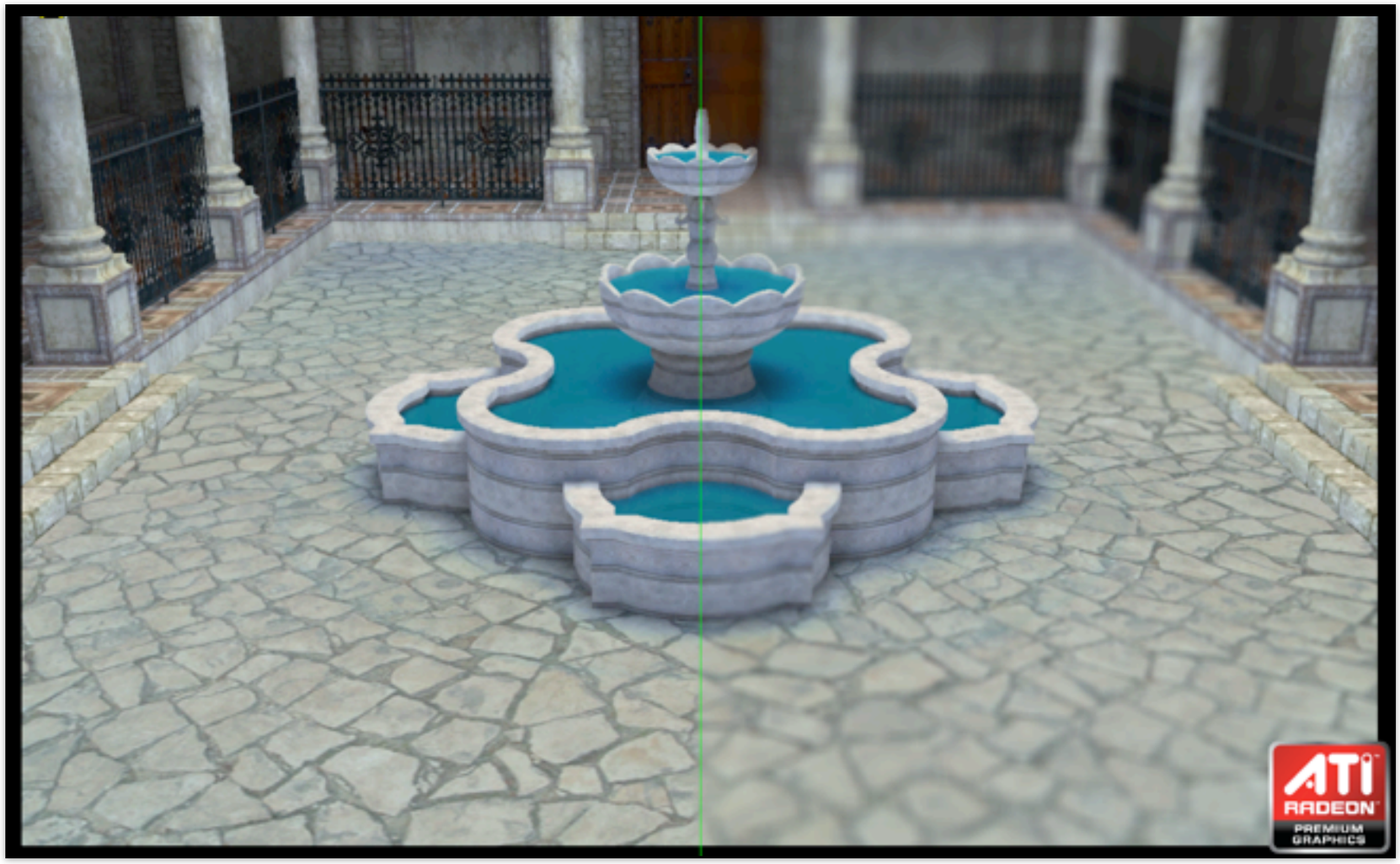


- Spreading more correctly blurs the silhouette edge
- Gathering and spreading the same for **constant** sized filter

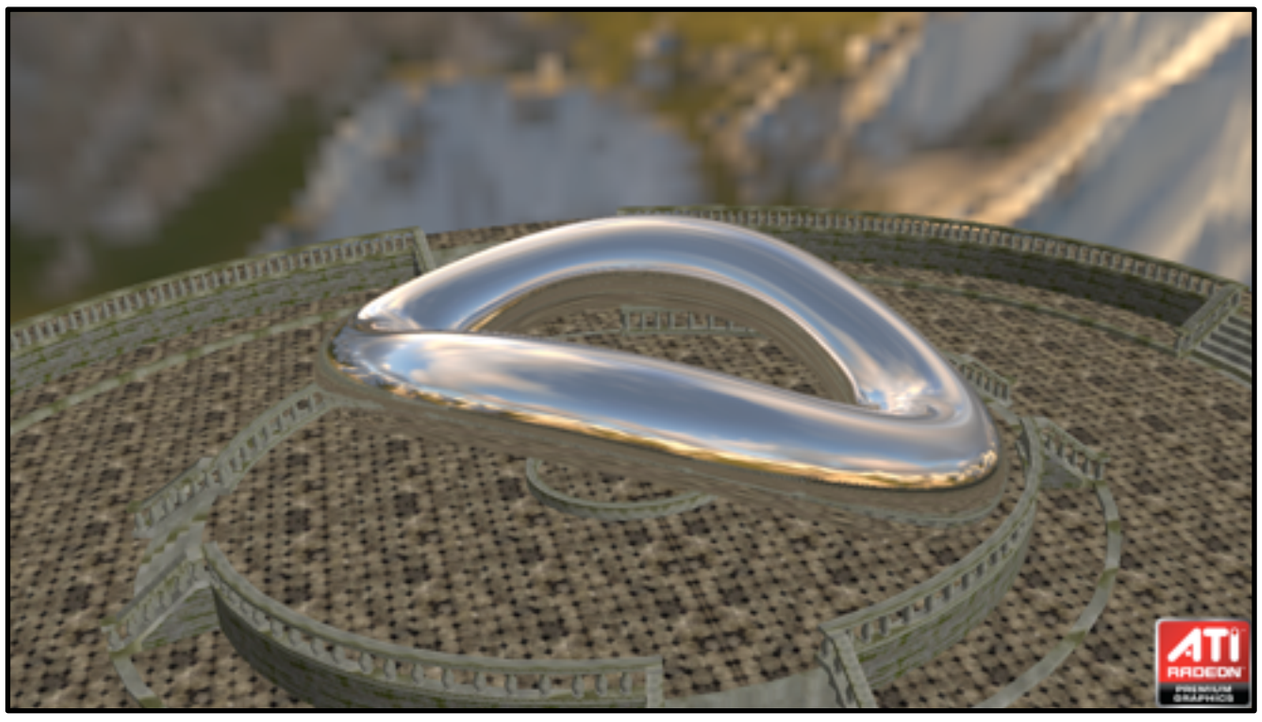
Fast Spreading

- Naïve implementations of filter spreading require O(N^2) time in the number of image pixels (very slow).
- Just talked about constant time "gather" filters
  - SATs one possible technique
- Modify SAT methodology to implement fast spreading
  - Basically run operations in "reverse"
    - 1. For each pixel, accumulate filter delta functions in a buffer
    - 2. Integrate result

Screenshot - demo later



Hybrid Rendering Using a Standard Graphics API



New Orleans, LA (August 2009)

AMD
The future is fusion

Ray Tracing vs. Rasterization

- Rasterization is still "faster" than ray tracing
- Ray tracing can handle some effects better than rasterization
  - Reflections
  - Soft shadows
  - Global illumination

Gratuitous Car Shot!

Motivation

- Investigate hybrid rendering with ray-tracing
  - Ray trace only reflections
  - Rasterize everything else (including primary rays)
  - Lots of people looking at this

- Identify issues game developers must address when integrating into an existing game engine infrastructure
  - Game engines are designed and optimized for GPUs
  - Art asset and material shader management important

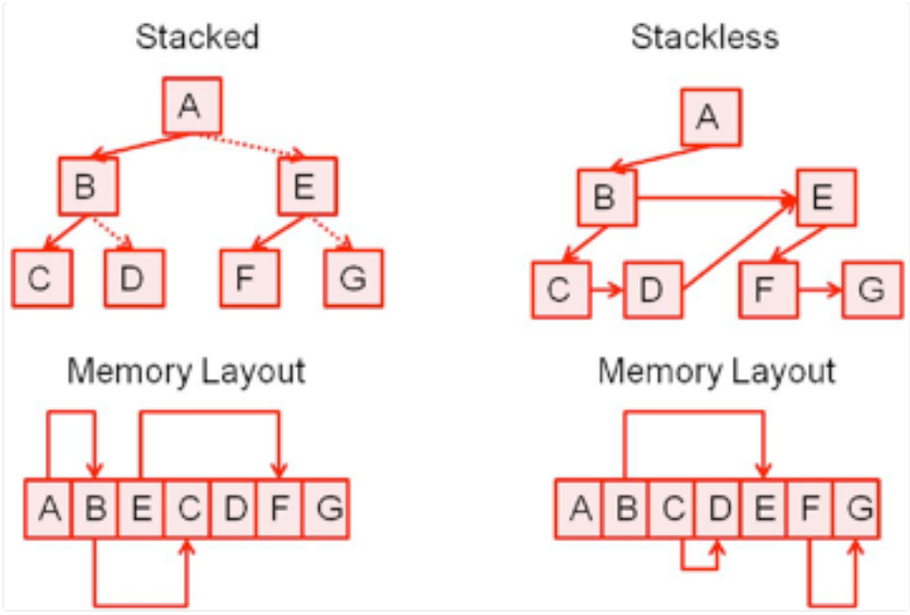
Experimental System Architecture

- Integrated into ATI's rendering engine used for the Ruby demos
- Implement in MS DirectX 10.1
- Tested on Radeon HD 4870

- Does not yet handle deformable objects

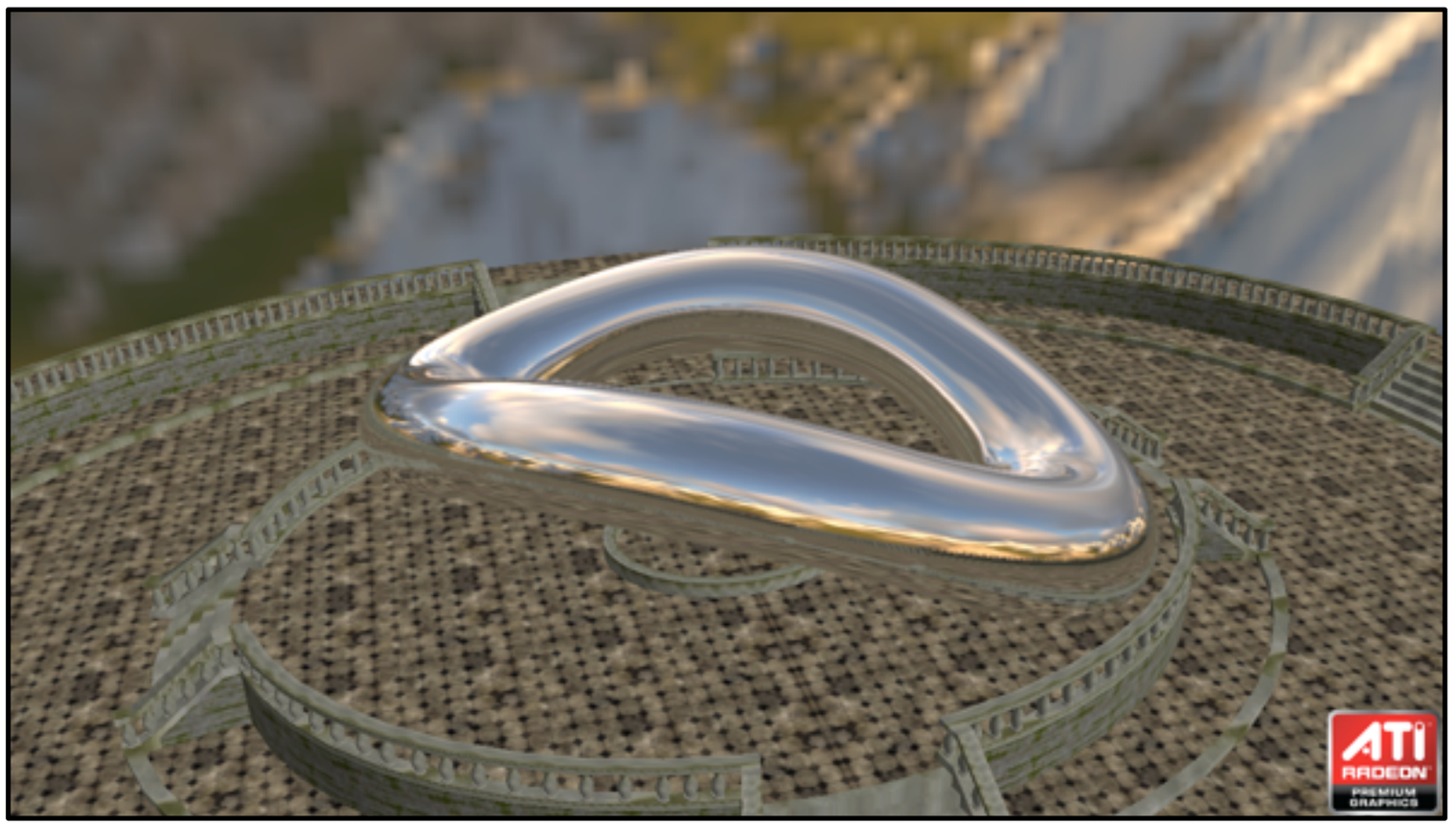
Data Structures

- Bounding Volume Hierarchy
- Stackless Bounding Volume Hierarchy
- KD Tree
- Bounding Interval Hierarchy



Data Structure Test Scene

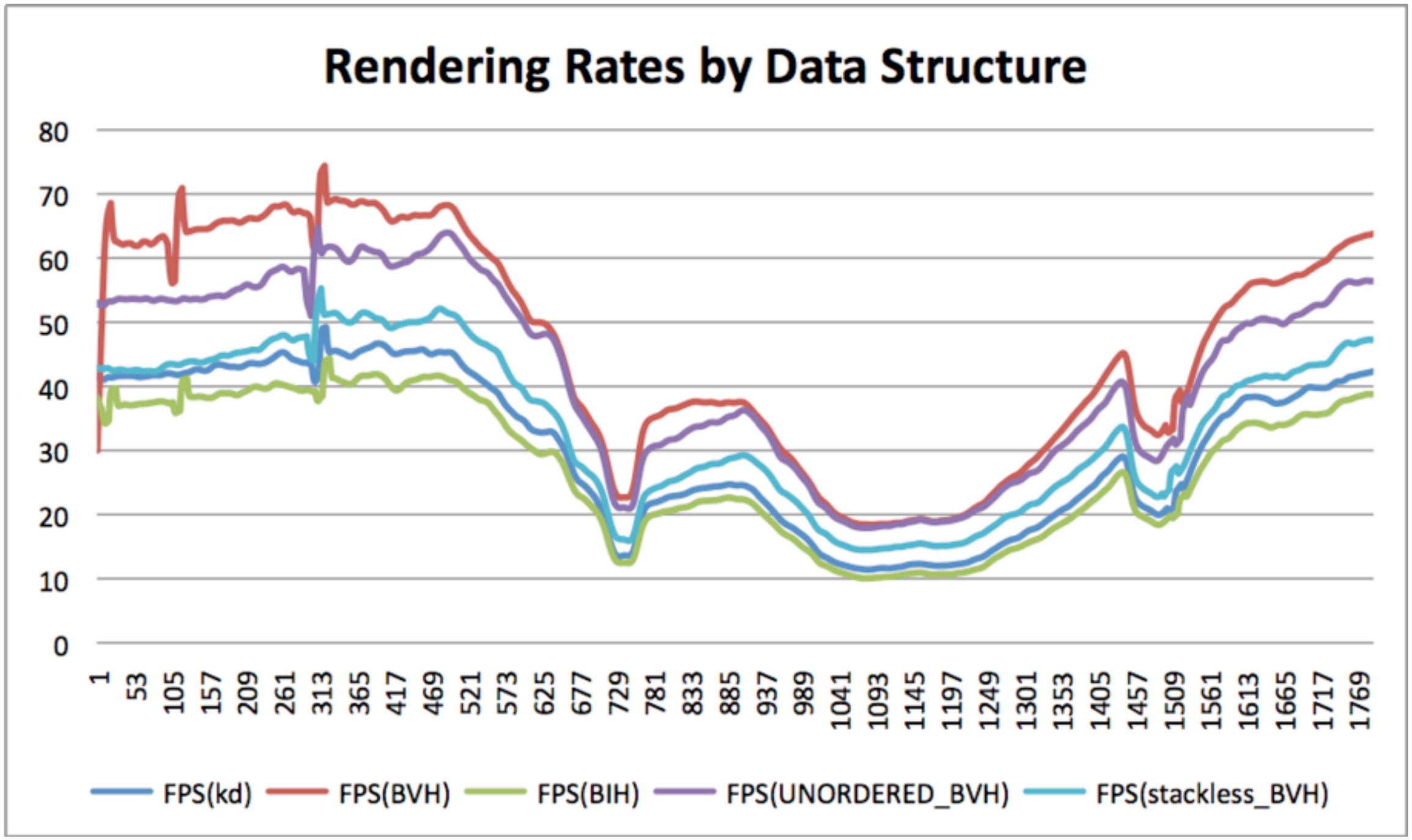
- Four materials → Four separate trees
  - Torus (20,872 triangles)
  - Park (255,488 triangles)



New Orleans, LA (August 2009)

AMD
The future is fusion

Results



AMD
The future is fusion

Metrics

	Nodes	Max Depth	Avg Depth	Triangles	Avg Tris/Leaf	Tree Size (MB)
BVH	113994	21	15.4	288852	5.07	3.48
KD	171048	27	19.8	834996	9.76	4.49
BIH	140944	27	19.1	288852	4.88	2.15

Test Scene Data

Algorithm	ALU Ops (VLIW)	Time (s)	% ALU Util	% TEX Util	Traversals	Writes Per Traversal	GPR count
BVH	7.41555E+12	131.68	46.92	30.57	2.15418E+11	0.112	14
KD	7.79515E+12	184.77	35.15	22.35	3.02104E+11	0.168	18
BVH_Stackless	1.56182E+13	198.73	65.49	37.44	3.7035E+11	0	15
BIH	1.02991E+13	210.26	40.81	15.21	3.28972E+11	0.165	11

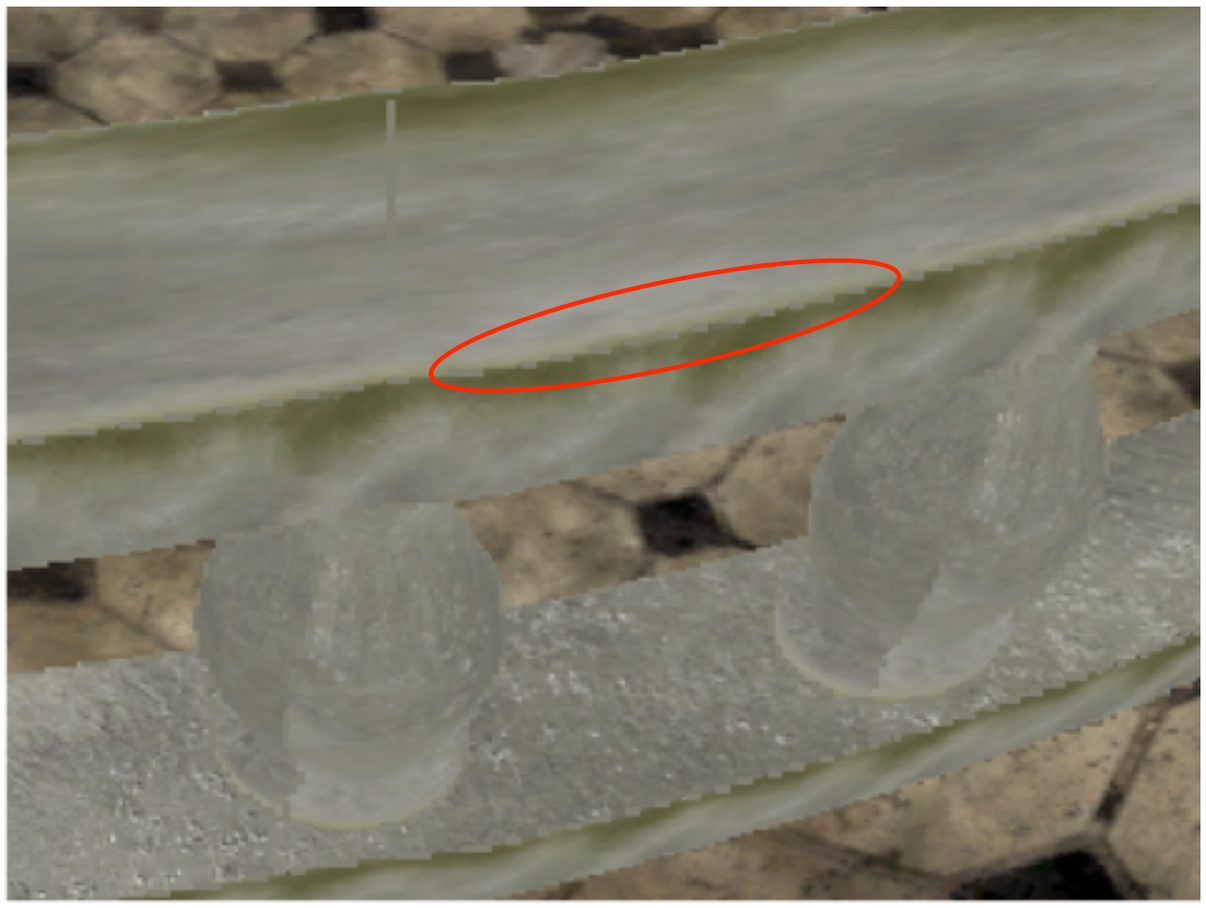
10,000 Random Rays

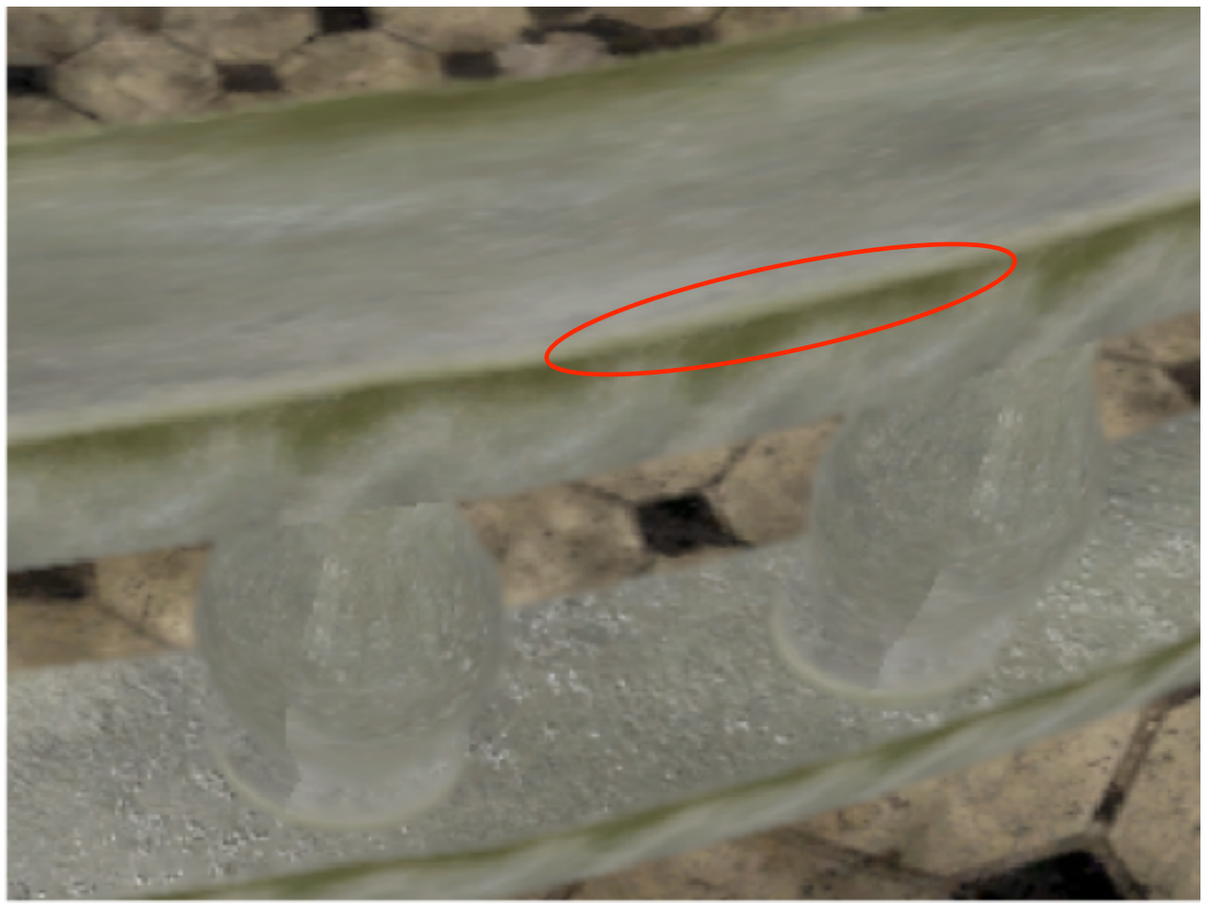
- BVH Advantages
  - Can vectorize tests
  - Handles empty space better
  - Stack is not a bottleneck

How to shade?

- Vertex Attributes must be interpolated across the triangle
  - Normals
  - Texture coordinates
- GPU usually handles this in hardware
- Must do this in shader at ray hit locations

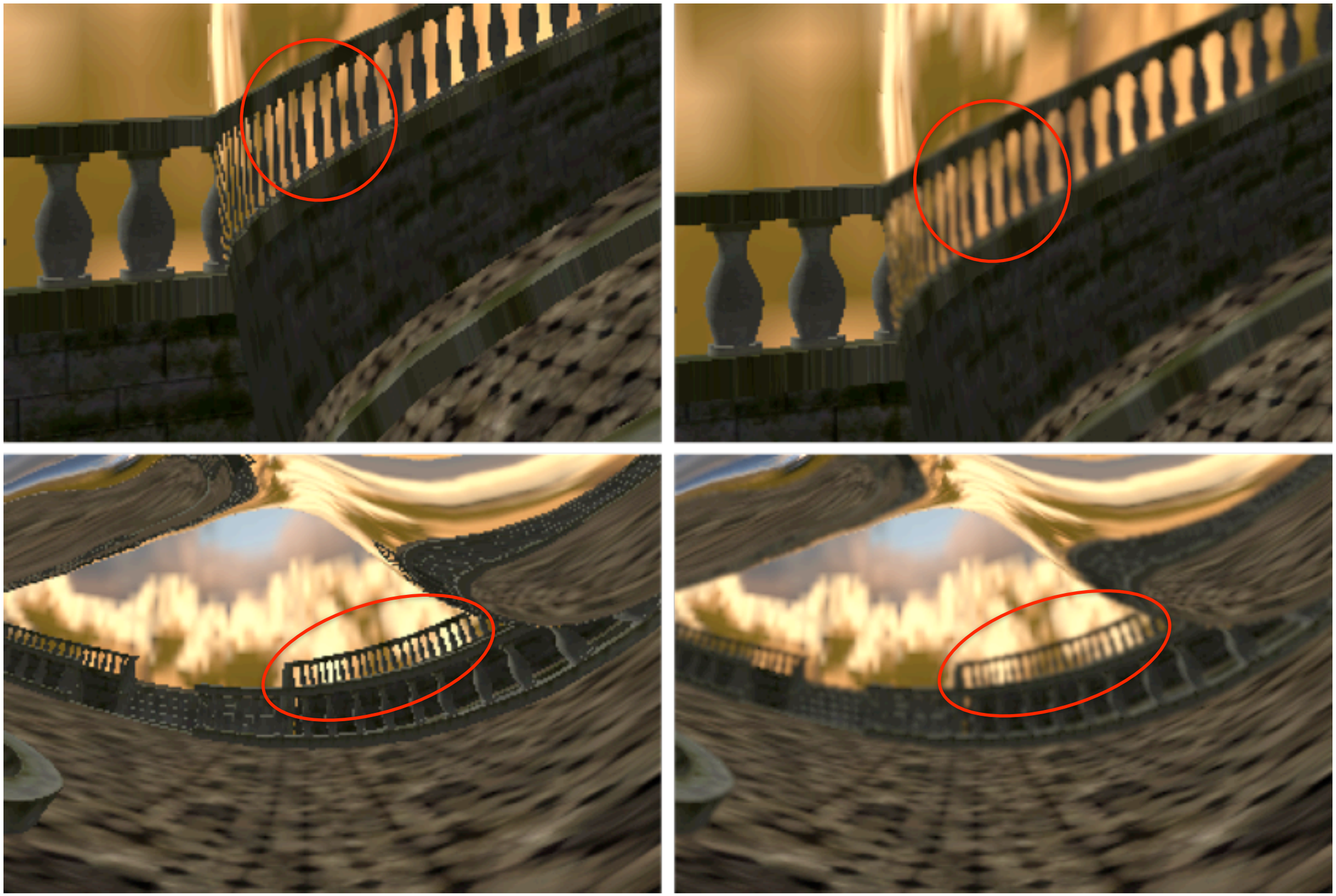
Texture Filtering





- Igehy, H. “Tracing Ray Differentials”. SIGGRAPH 1999.

Geometric Aliasing



AMD
The future is fusion

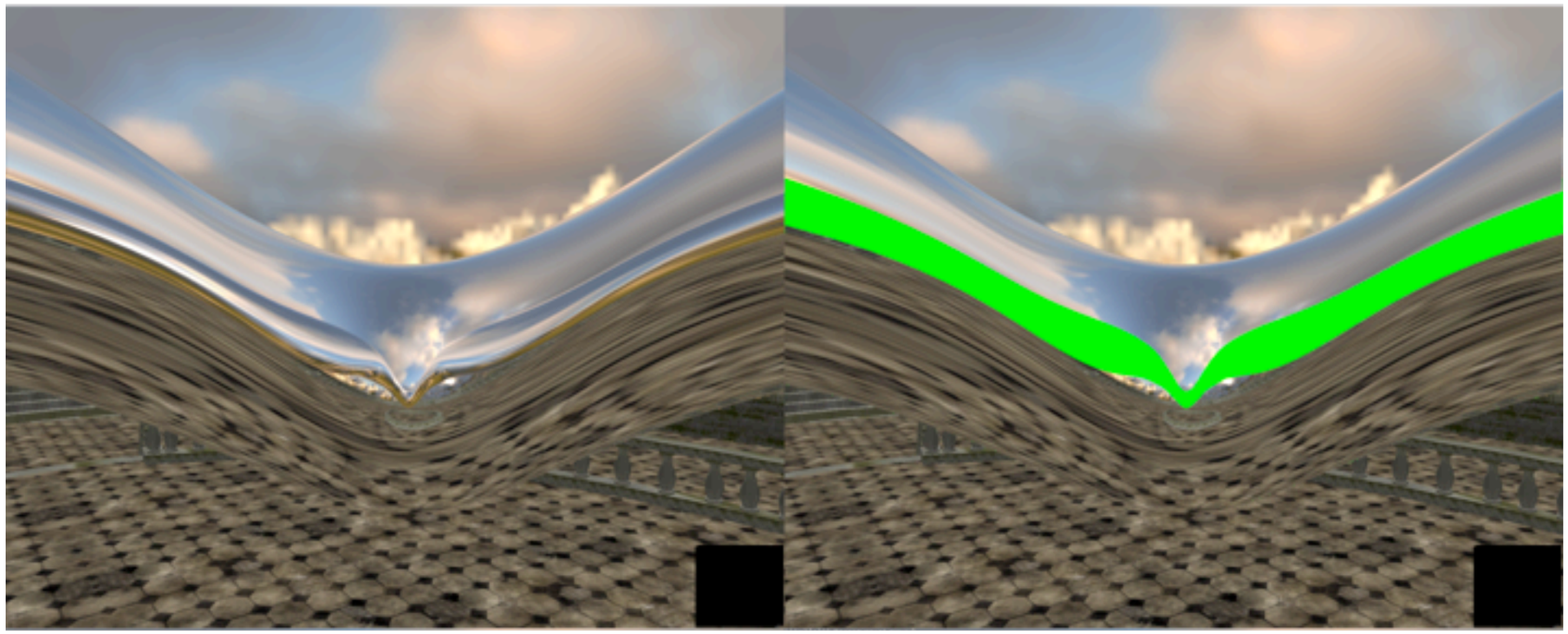
Geometric Aliasing

- Supersample, but expensive
- AA only on edges (similar method of deferred shading)
  - Surface normal variation
  - Depth discontinuities
  - Object edges
  - Judicious use of blurring

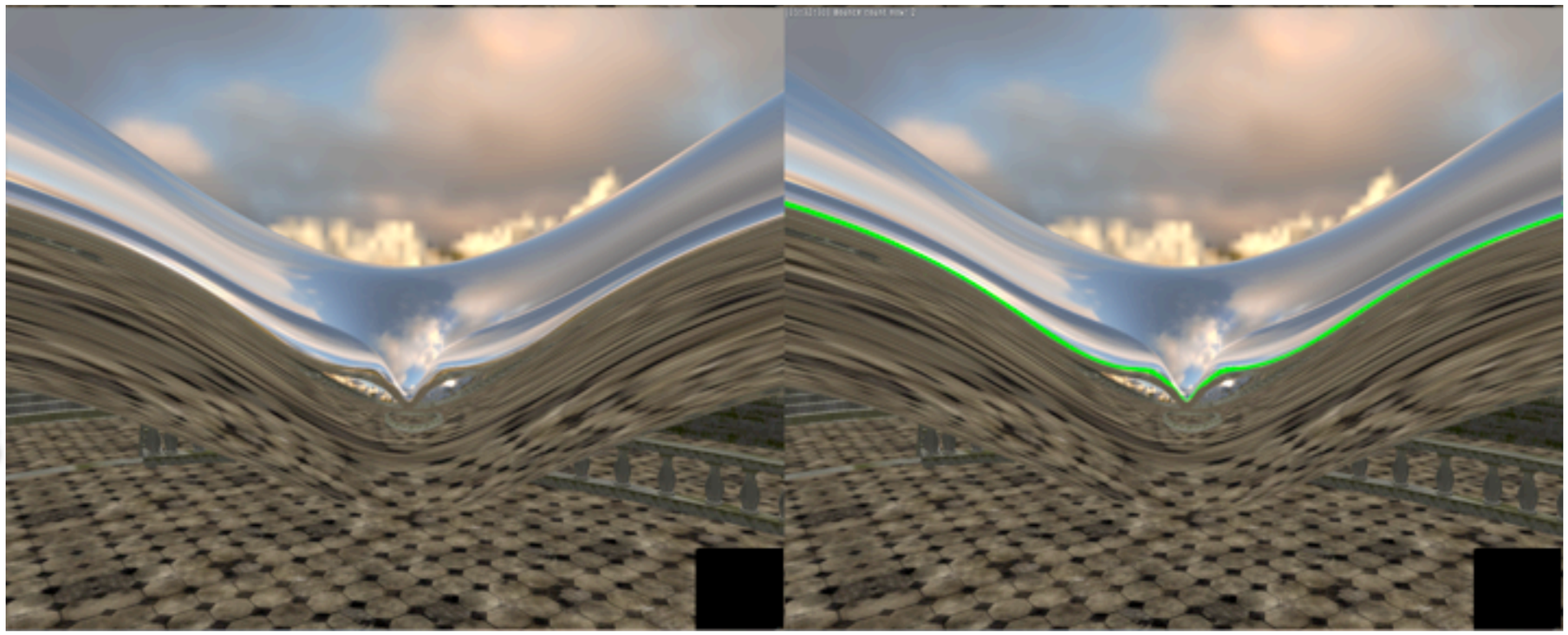
AMD
The future is fusion

Number of Bounces

1 Bounce



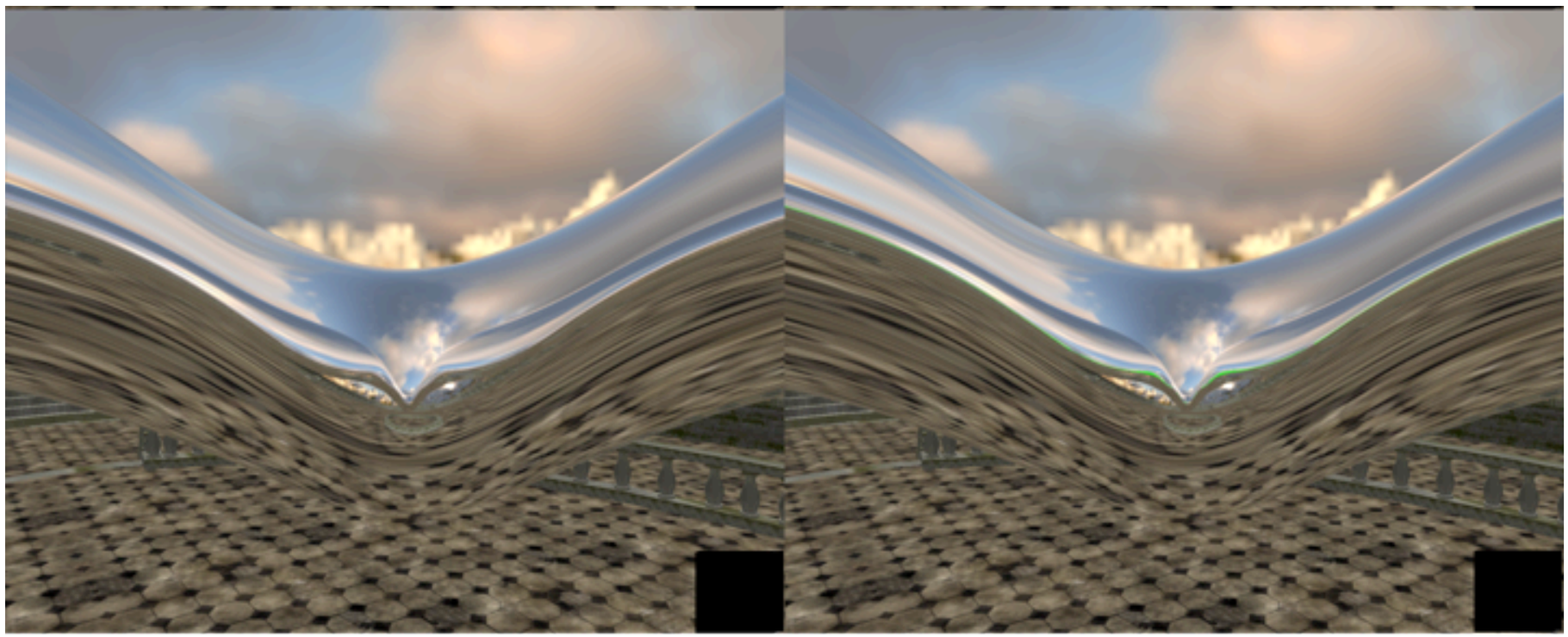
2 Bounces



New Orleans, LA (August 2009)

AMD
The future is fusion

Number of Bounces



3 Bounces

AMD
The future is fusion

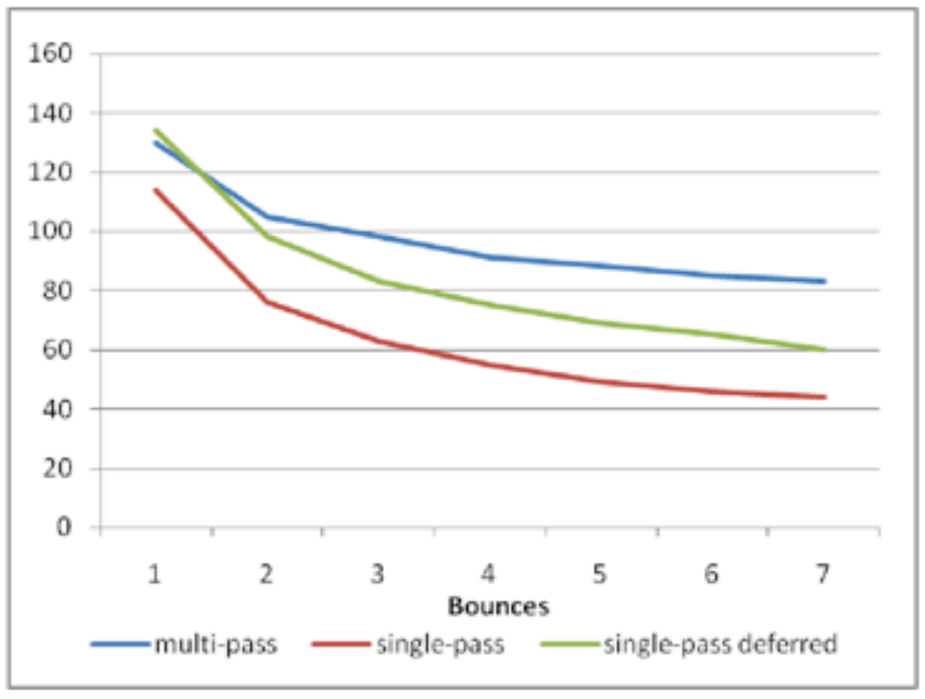
Single Pass vs. Multi Pass Shader

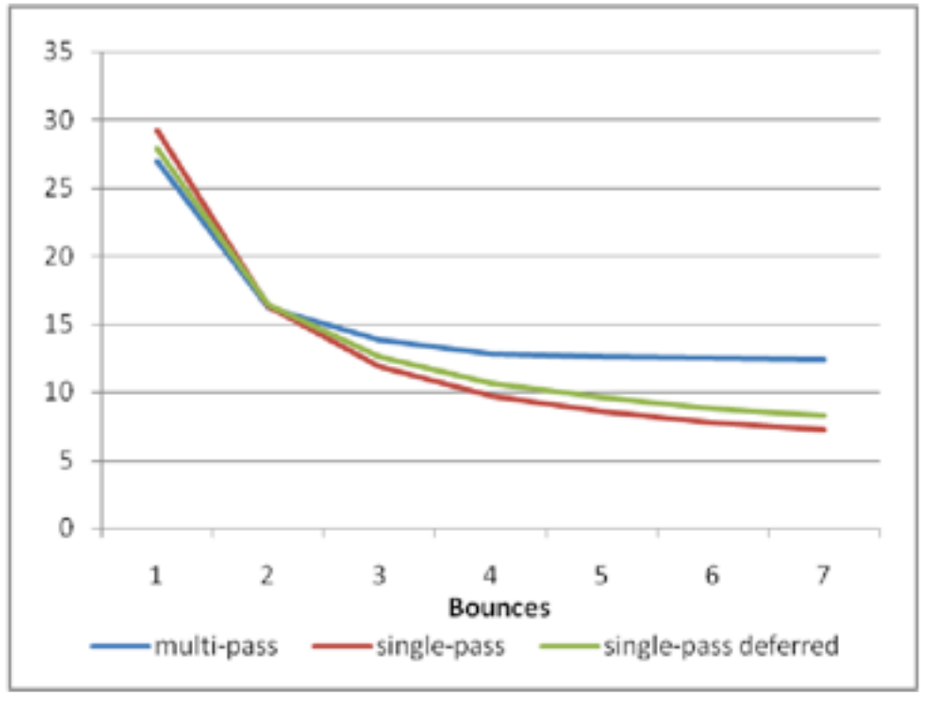
- Single pass
  - One rendering pass per reflective object
  - Easier to manage during rendering
  - Not well suited during production
    - Each material change requires shader update
  - Performance issues
- Multi pass
  - More closely resembles a conventional forward-rendering
  - Much more flexible

Multi Pass Ray Tracing

- Rasterize all objects and save ray reflection info of shiny objects to a MRT and stencil
- Render a quad for each potential object to be hit by a ray
  - Shade every hit and save the hit distance
  - Discard by alpha if distance is further than previous hit
- Distance buffer needs to be ping-ponged due to read-write limitation of DX 10.1, but solved using DX 11
- Cannot clip because distance is needed

Performance Scaling





AMD
The future is fusion

Further Reading on Other Topics

- Iourcha, K., Yang, J., and Pomianowski, A. 2009. A Directionally Adaptive Edge Anti-Aliasing Filter. *High Performance Graphics*.
  - Shipping as a driver feature on ATI Radeon HD GPUs:
    Edge-Detect Custom Filter AA

Demos

New Orleans, LA (August 2009)

AMD
The future is fusion

Acknowledgements

- Jason Yang (AMD)
- Joshua Barczak (Firaxis Games)
- Todd Kosloff (UC Berkeley)

Questions?

Trademark Attribution

AMD, the AMD Arrow logo and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Other names used in this presentation are for identification purposes only and may be trademarks of their respective owners.

©2009 Advanced Micro Devices, Inc. All rights reserved.