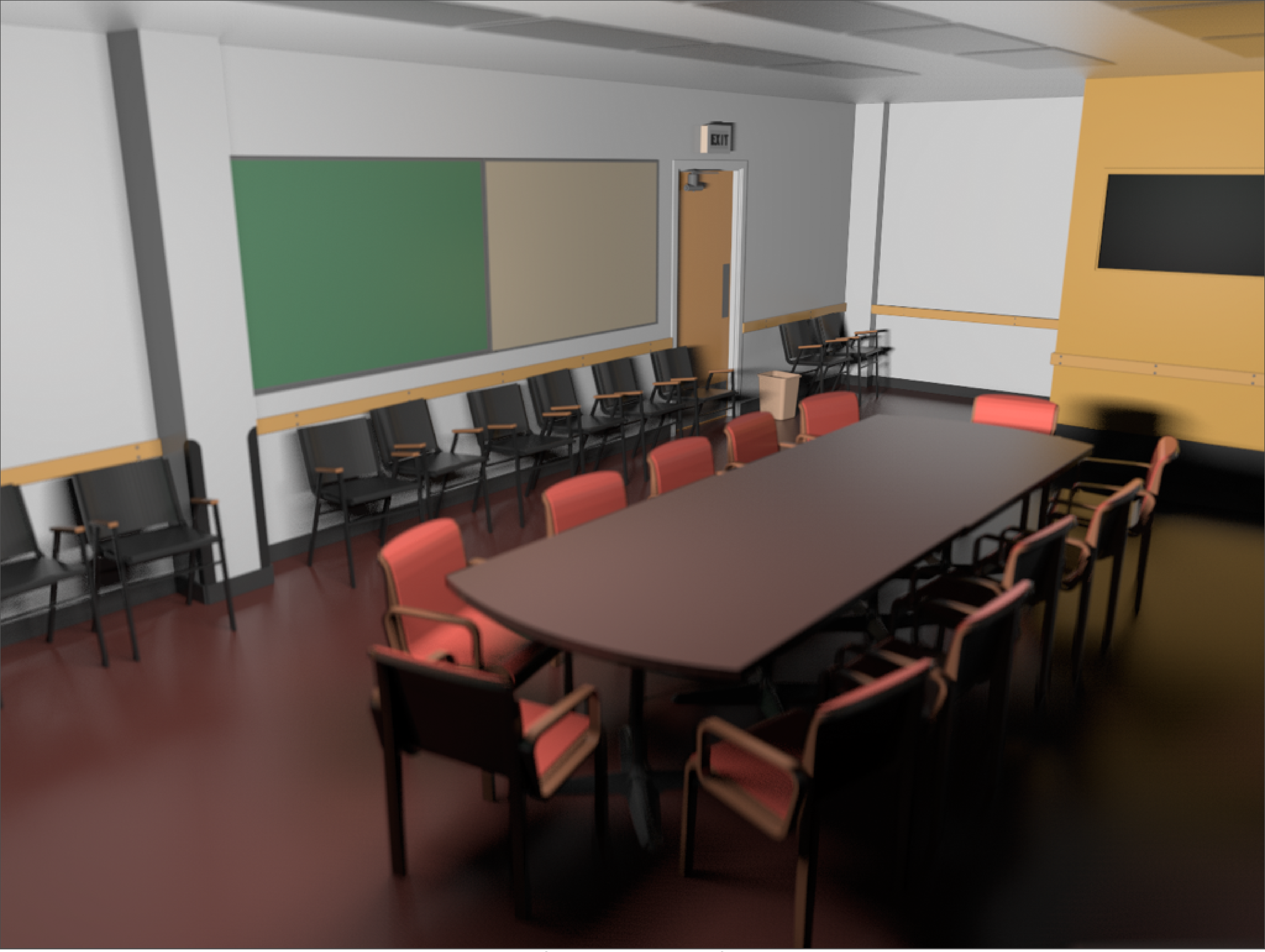
Packet-Based Whitted and Distribution Ray Tracing

Solomon Boulos, David Edwards, J. Dylan Lacewell, Joe Kniss, Jan Kautz, Peter Shirley, Ingo Wald

University of Utah

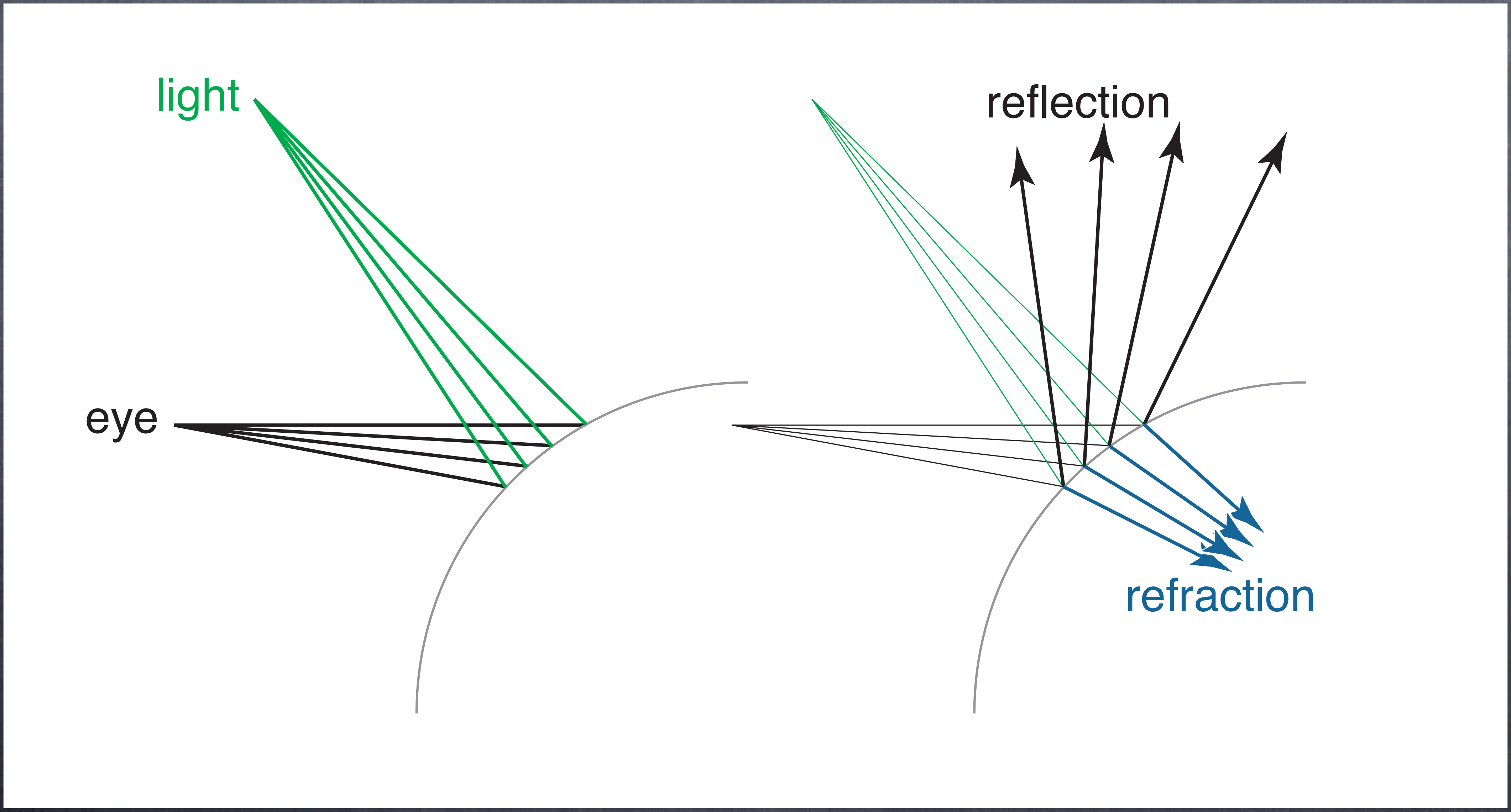
I'm Solomon Boulos from the University of Utah. I'll be presenting my work on interactive ray tracing involving real secondary rays using packets.



This is the goal. A nice looking distribution ray trace of the standard conference scene. Every material in here has been made glossy to really stress the system (but still produce a reasonable looking image). Our system traces slightly less than 1M ray segments per second per Opteron core and this image requires just under 300M ray segments. So this is a 5 minute image on 1 core. The latest 8-core mac pro is about twice as fast per core as the Opteron used for this timing, so this is about a 20 second image on modern hardware.

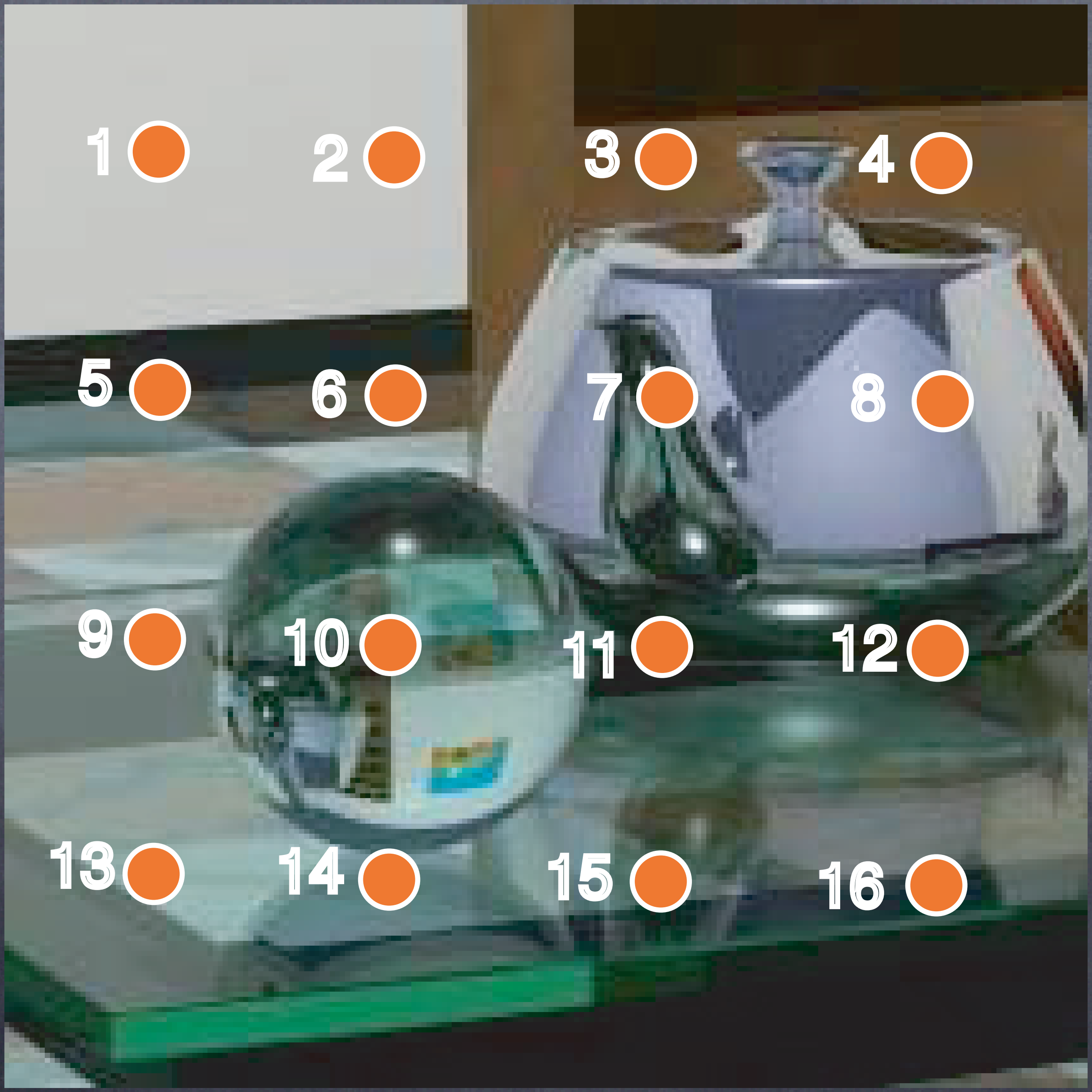
Ray Casting vs Whitted

- In ray casting: packets can amortize constants



Recent interactive ray tracing performance has been mainly derived from the use of ray packets. Larger ray packets allow for significant amortization of both computations and memory accesses. Successful systems such as OpenRT, Intel's MLRTA and our own Coherent Grid and Dynamic BVH are examples of systems that attempt to take advantage of ray packets to avoid redoing work. Each of these systems relies on more than just "using SIMD" and takes advantage of some clever geometric or mathematic technique due to common origin, common sign, etc. These are available when perform ray casting or even shadows from a point light (we call this Ray Casting with Shadows). In Whitted's illumination model, however, even reflection and refraction rays from a pinhole camera yield non-constant origin or signs. This yields a situation where how rays are grouped into packets is non-obvious.

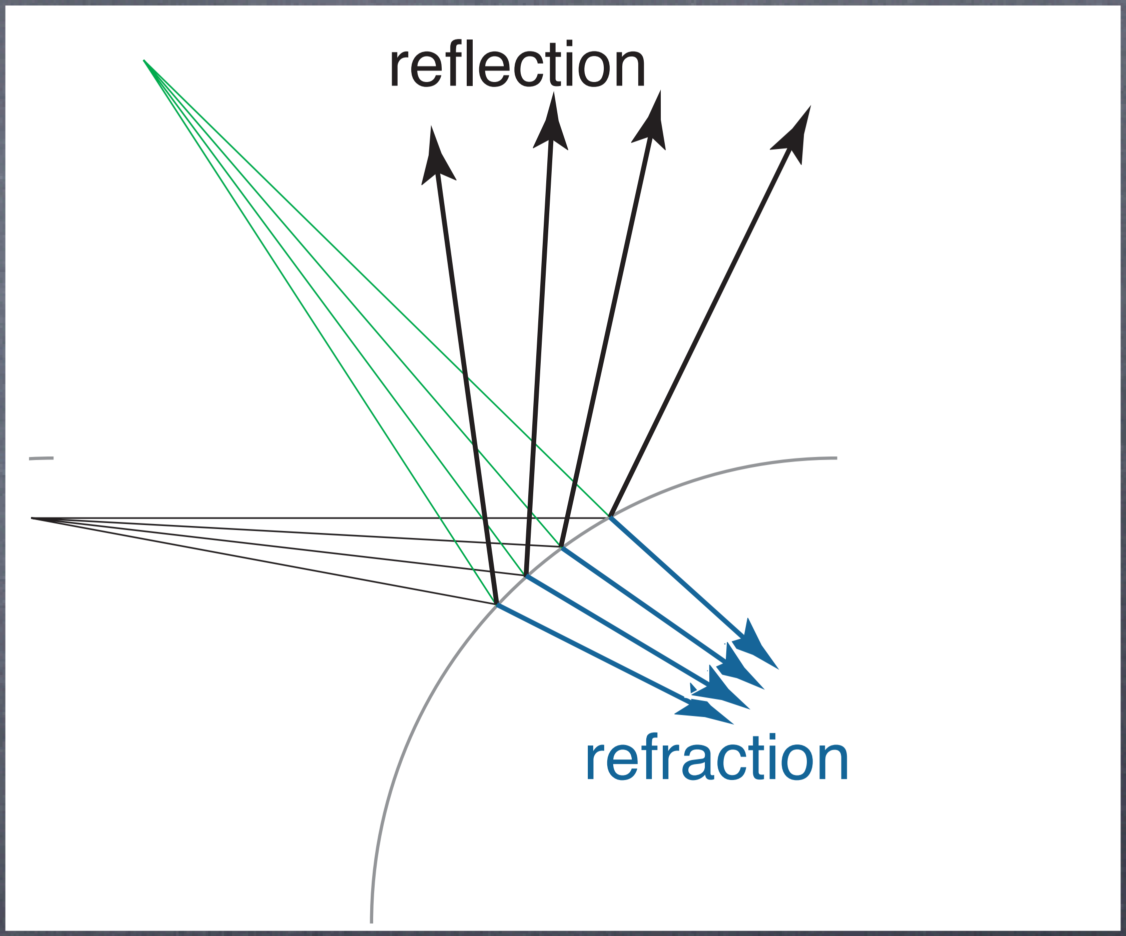
Assembling Packets



As an example, consider 16 primary rays in this historical Utah scene. Rays 1-4 hit diffuse surfaces and require shadow rays. Rays 5 and 6 hit the shiny floor with a phong model and require both reflection and shadow rays. Rays 7, 8, 11, and 12 all hit a metal teapot and only generate reflection rays. All other rays hit glass and generate both reflection and refraction rays.

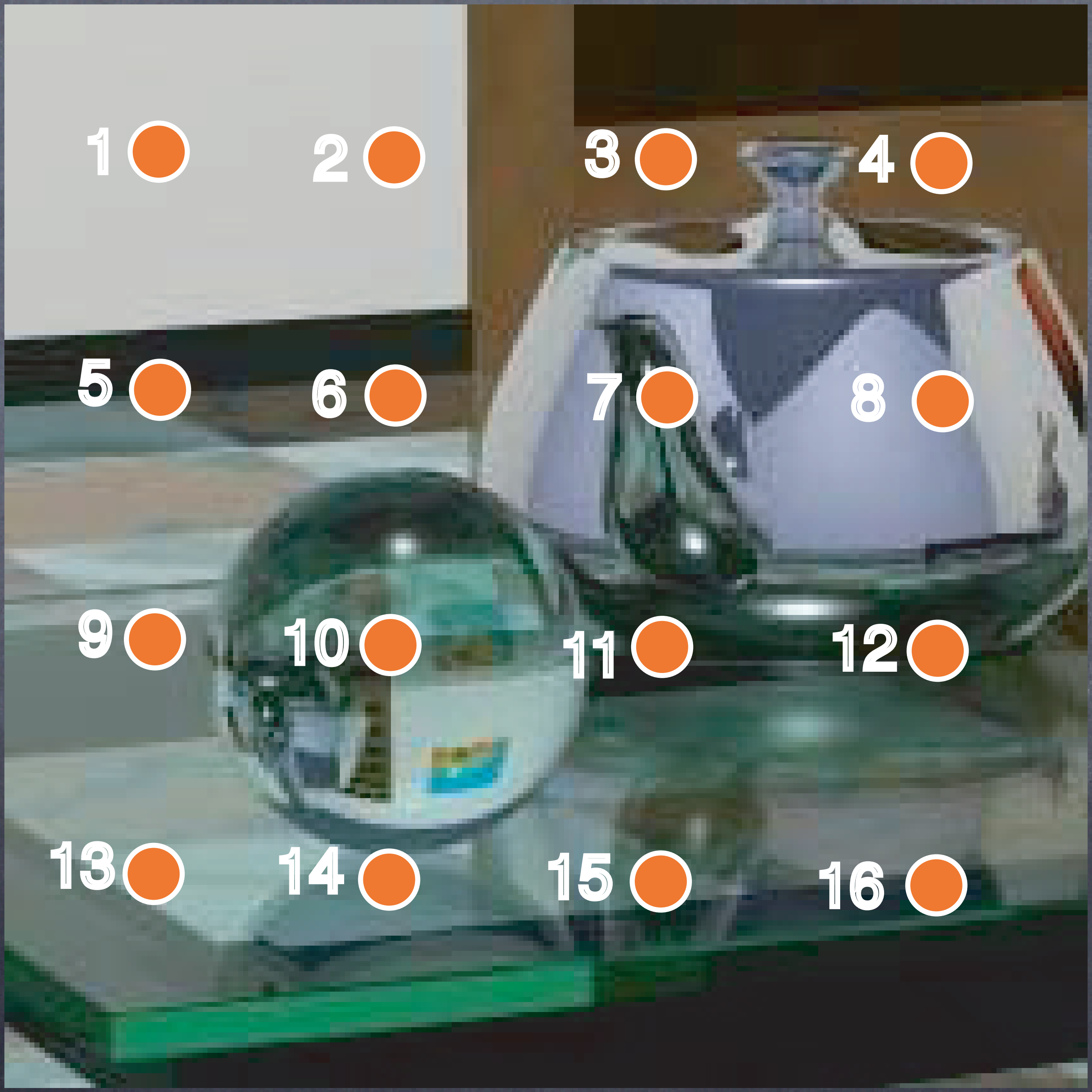
Three Basic Options

- No Packets / Single Ray
- Blind / All Together
- Groups
  - Runs (material type)
  - Ray Type



There are three basic methods for taking our previous situation and choosing how to form new packets of rays. The first is to not use any packets at all, which is what the OpenRT system has traditionally done. In this case, primary rays use packets but all shading is done with traditional single ray code. The complete opposite of this is to just put all the rays together. However, as we remember from the earlier example this would produce awful packets. The basic point of this work is to explore reasonable assembly methods. The two specific methods are runs based on material type and grouping based on ray type.

Runs (by example)



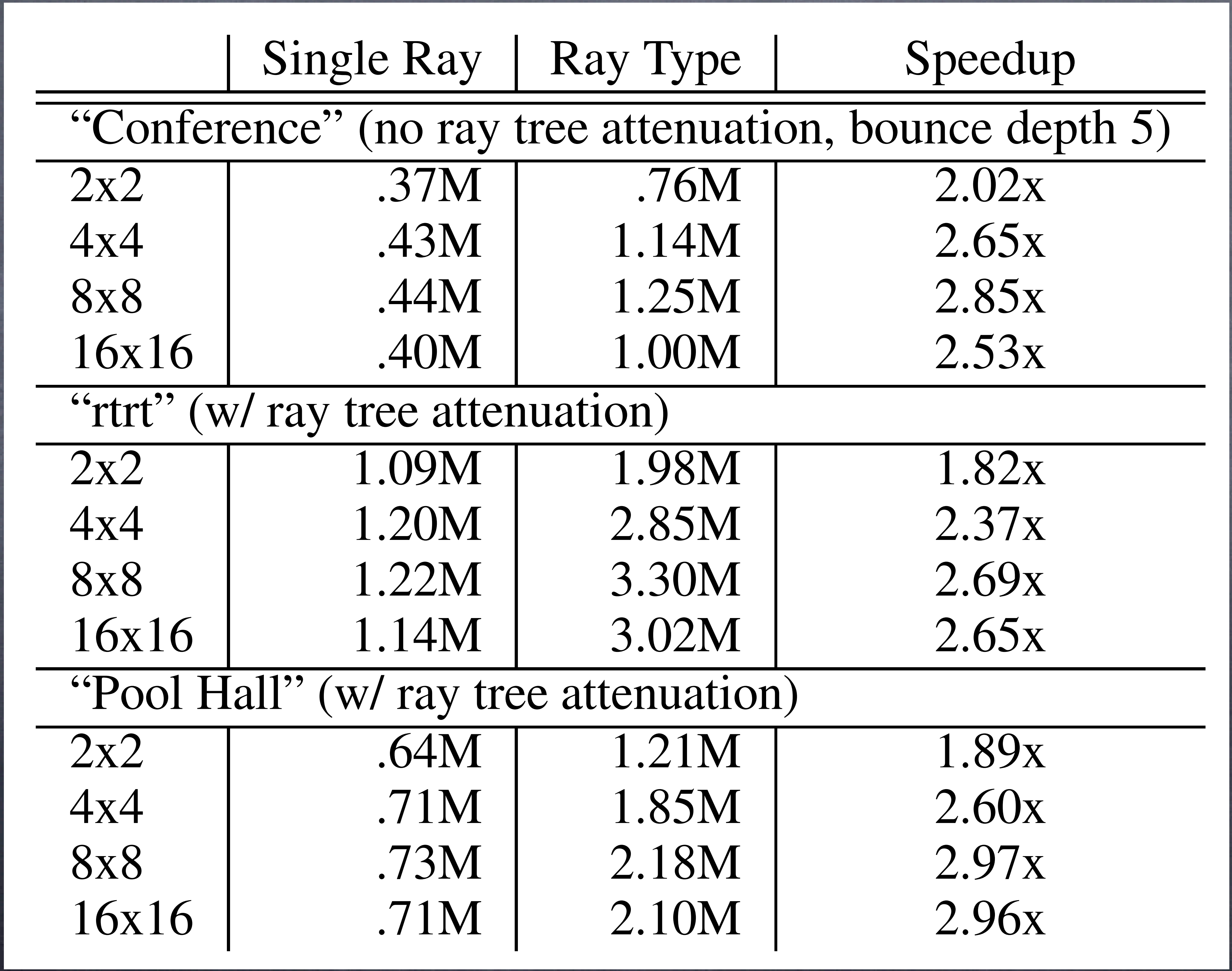
Runs grouping is the style of grouping we have used in the Manta Interactive Ray Tracer. The basic idea is to loop over input rays and build runs of rays that share some common property. In this case, we have chosen to use the material pointer to determine rays that agree. For this scene, rays 1-4 would shade together, 5-6 together, 7-8 together, 9-10, 11-12, and then 13-16. This simple example demonstrates a possible failing for runs grouping: it is inherently sensitive to ordering.

Ray Type

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

In ray type grouping, all input rays first fill in queues of rays for shadows, reflections and refractions. This grouping completely ignores whether a diffuse or phong model has generated a shadow ray: all are treated the same. An important feature of this specific ray type grouping is that the number of groups is bounded at 3: shadows, reflections and refractions. More general grouping methods might be possible, but could require either immense stack space or “thin out” input packets so much as to be equivalent to single ray.

Faster than single ray?

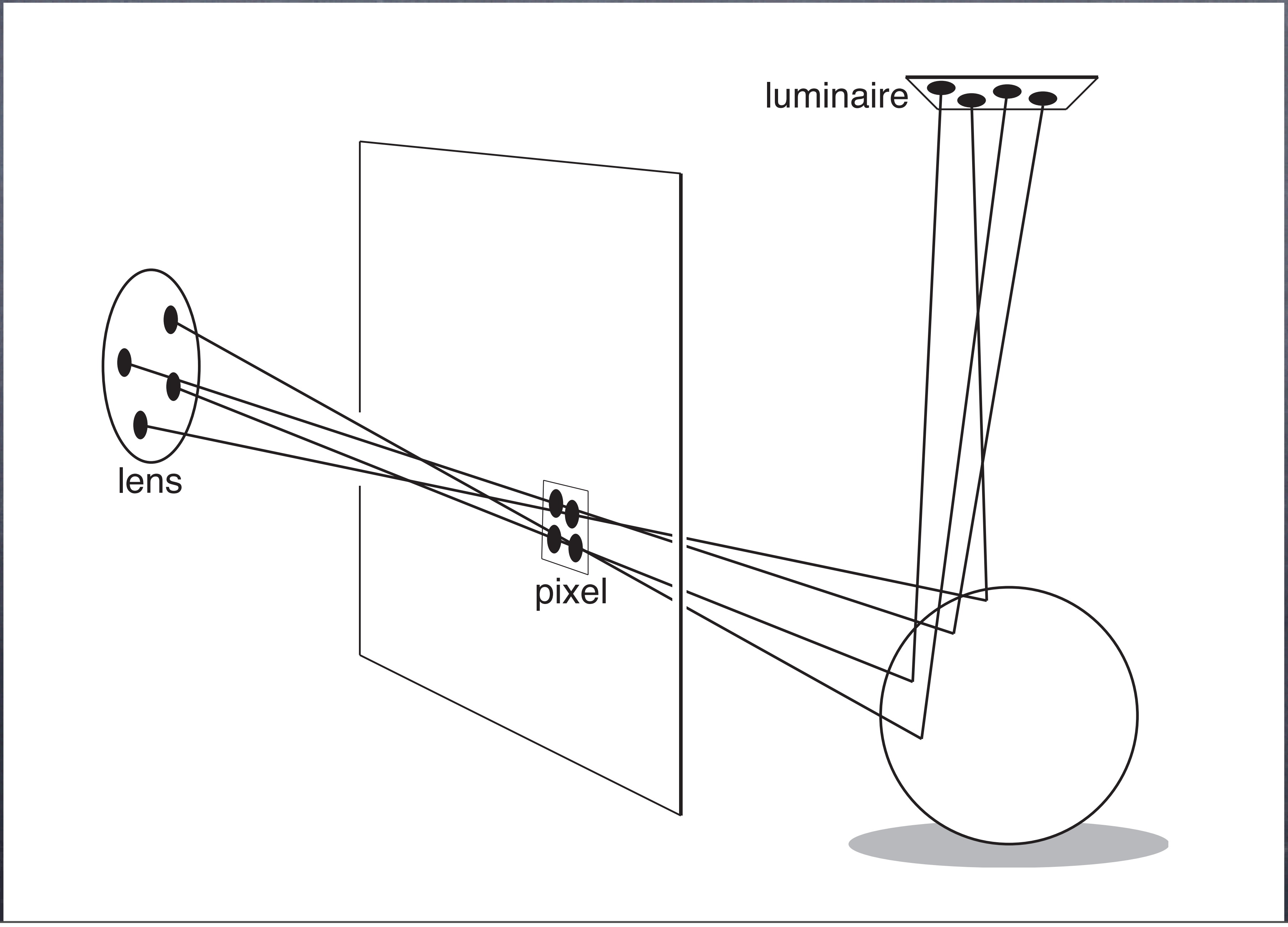


	Single Ray	Ray Type	Speedup
"Conference" (no ray tree attenuation, bounce depth 5)			
2x2	.37M	.76M	2.02x
4x4	.43M	1.14M	2.65x
8x8	.44M	1.25M	2.85x
16x16	.40M	1.00M	2.53x
"rtrt" (w/ ray tree attenuation)			
2x2	1.09M	1.98M	1.82x
4x4	1.20M	2.85M	2.37x
8x8	1.22M	3.30M	2.69x
16x16	1.14M	3.02M	2.65x
"Pool Hall" (w/ ray tree attenuation)			
2x2	.64M	1.21M	1.89x
4x4	.71M	1.85M	2.60x
8x8	.73M	2.18M	2.97x
16x16	.71M	2.10M	2.96x

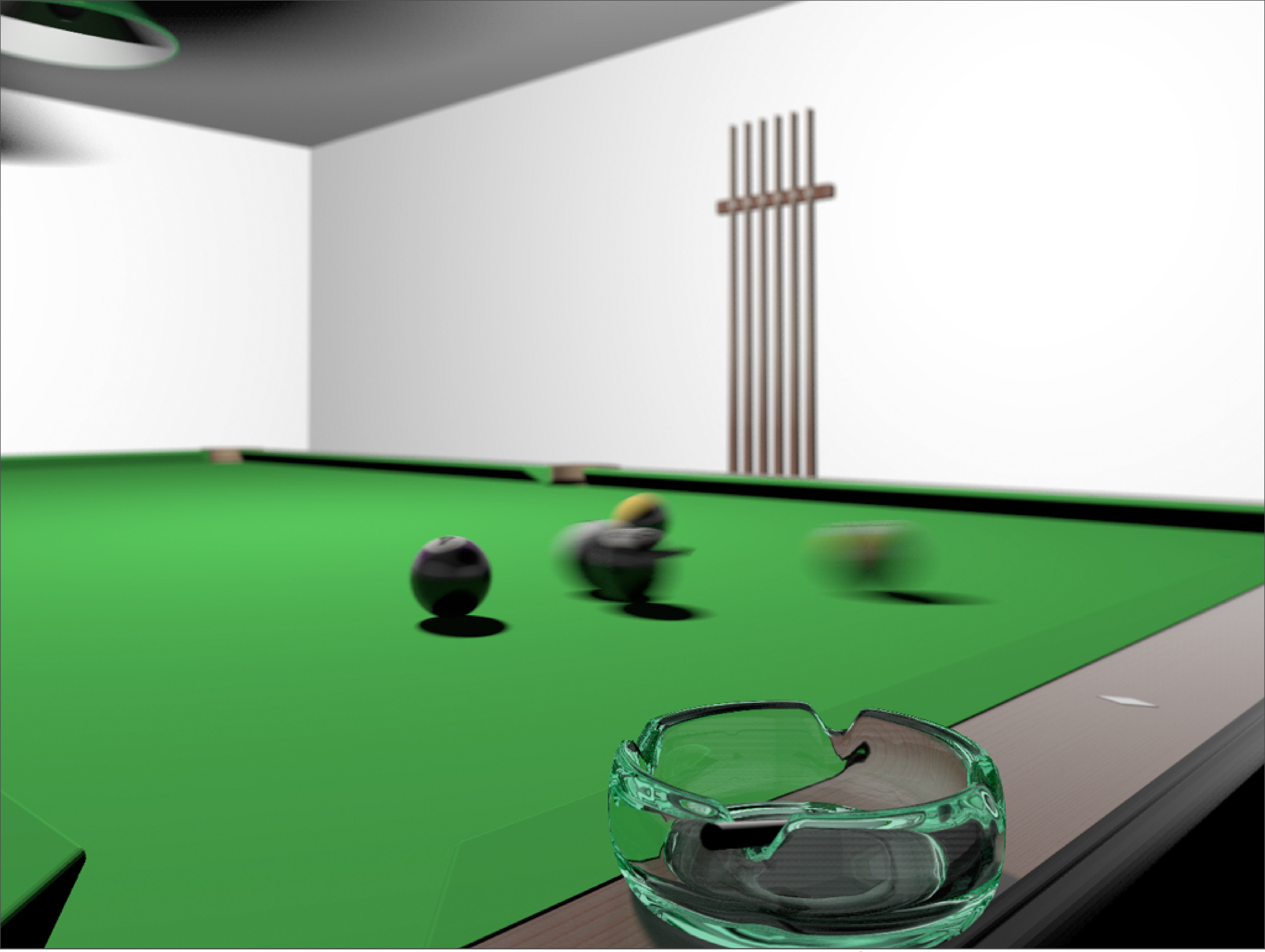
An important question for any packet based work is whether or not it actually matters. In more direct terms, is it actually better than just single ray code? Each row within the subtables compares single ray grouping for secondary rays against the proposed ray type grouping. It should be noted that even the "Single Ray" code uses packets for primary rays where they are assumed to be beneficial. This table demonstrates that for whitted style ray tracing, the benefits of packet tracing still exist. And indeed even algorithmic amortization beyond simple SIMD are present in the table. It should be noted that comparing the best RayType grouping to the 2x2 single ray, we can achieve nearly a 4x speedup for WRT. But then again, Whitted style ray tracing isn't really the goal...

Distribution Ray Tracing

- Remove constants

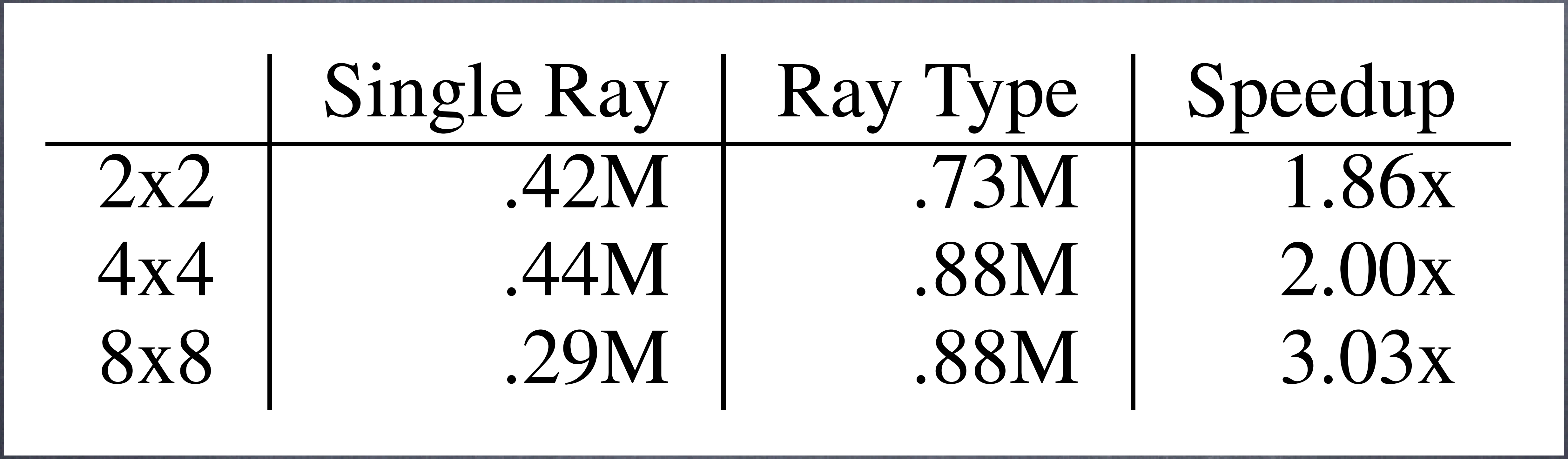


For distribution ray tracing, we add a little bit of blur to each dimension. This immediately removes common origin optimizations for even the camera and further decreases the so called coherence of generated packets (presumably regardless of assembly method). An important question in our work was to actually generate data for this case. Many researchers have assumed that distribution ray tracing leads to reduced coherence because of this blur, but nobody had shown whether or not that was the case.



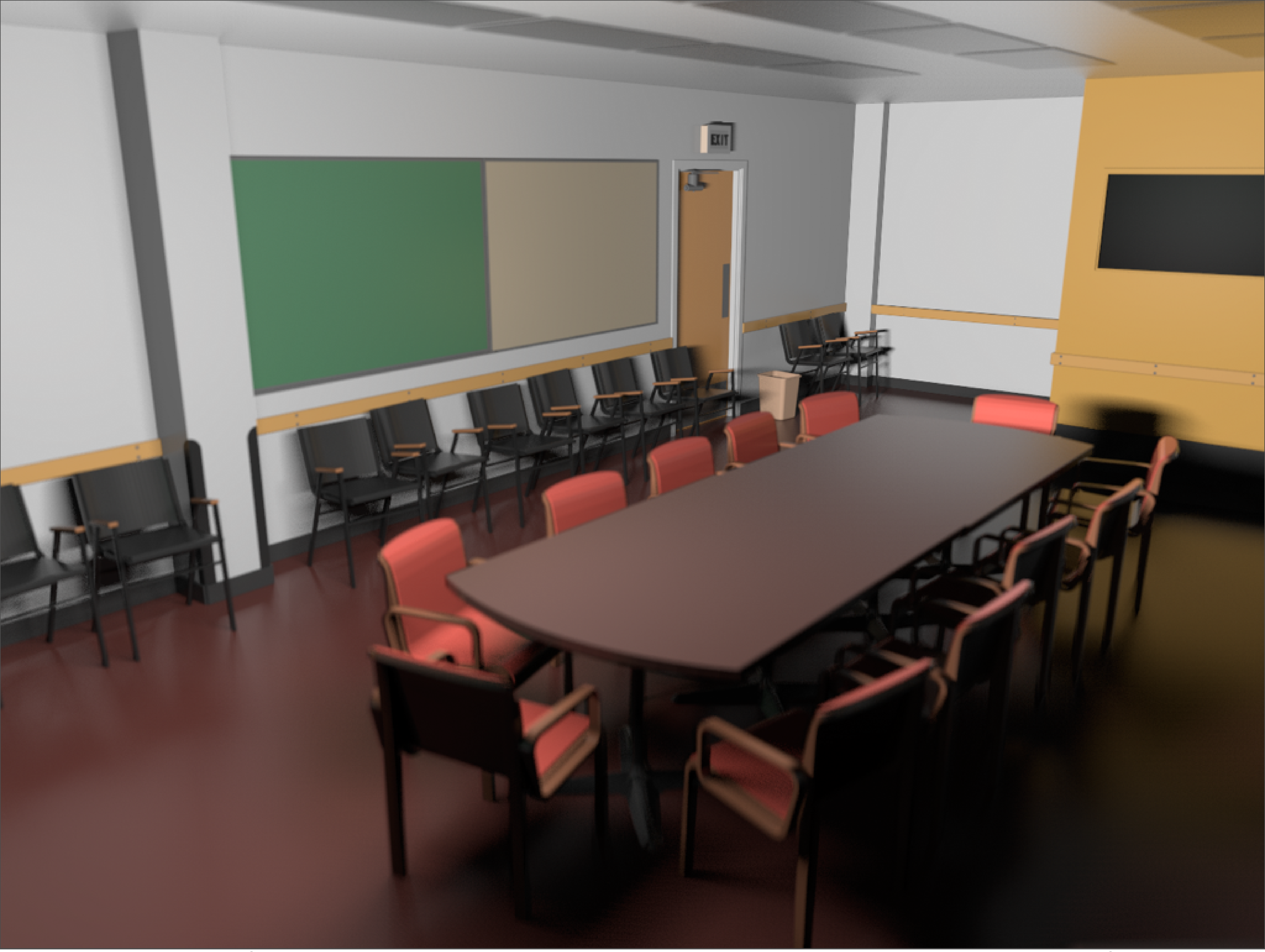
This is our test scene mimicking Cook’s original 1984 distribution ray tracing paper (in fact, we even set up the animation to look like his from above). We’ve added a glass ashtray and a real poolhall so we can demonstrate depth of field as well.

What about DRT?



	Single Ray	Ray Type	Speedup
2x2	.42M	.73M	1.86x
4x4	.44M	.88M	2.00x
8x8	.29M	.88M	3.03x

This is a table from the paper for distribution ray tracing on the conference scene comparing ray type grouping to a high performance single ray implementation. Each row shows a different packet size from 2x2-8x8. The 2x2 row is a pure SIMD implementation similar to Ingo Wald's initial "packet tracing" paper. Our results demonstrate that even for distribution ray tracing (every surface in the scene has a phong exponent of 256) both SIMD and algorithmic amortization are possible.



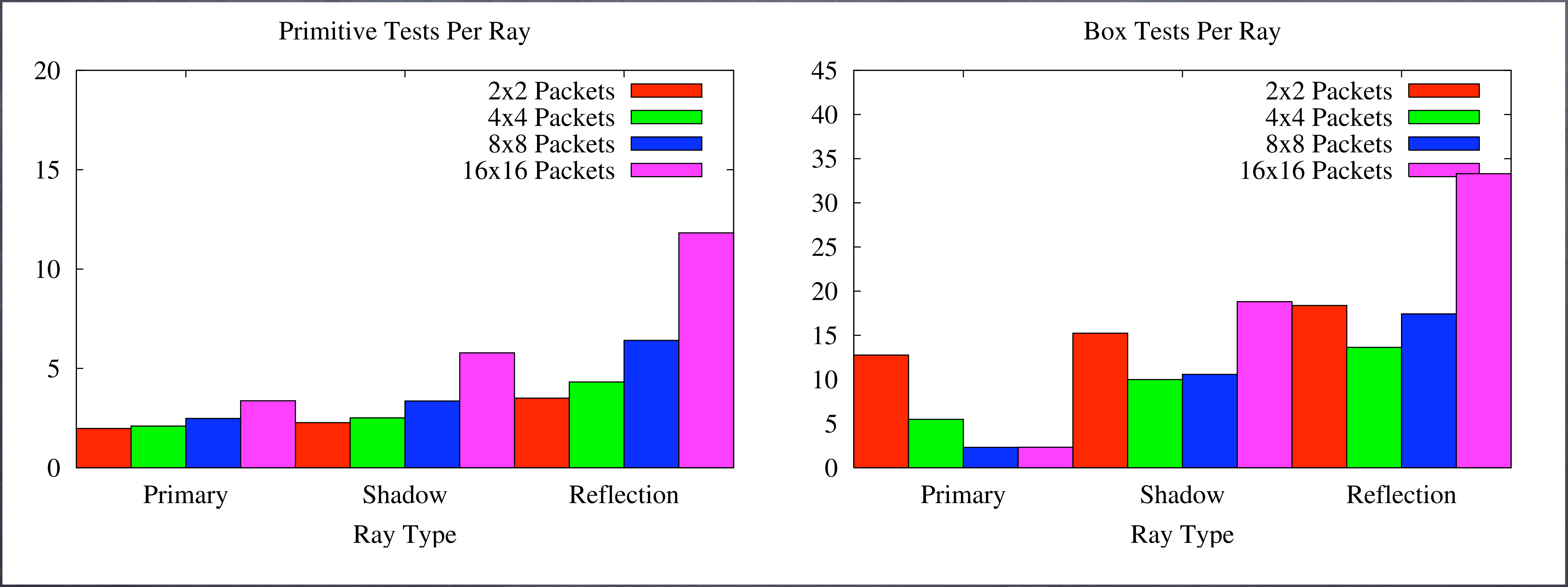
This is just a reminder of how glossy the test actually is. To me this looks a lot like a nice path traced image and for the most part it basically is (a glossy exponent of 256 is low enough to be just sort of shiny).

How does this compare?

	RCS rps	WRT rps	DRT rps
conference	3.25M	1.79M	0.88M
rtrt	3.30M	2.00M	1.53M
poolhall	2.83M	.83M	1.23M

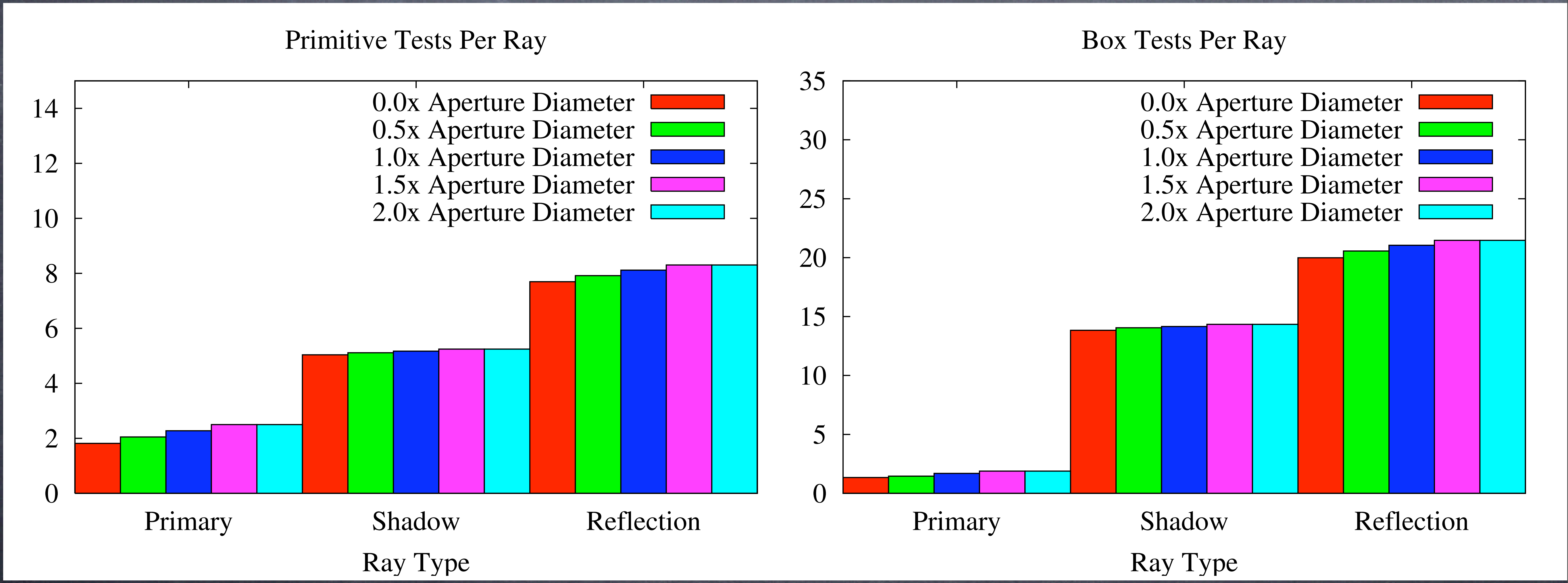
An interesting question is how do different rendering types compare to each other. In this table, we demonstrate that while ray casting with shadows is clearly the fastest our performance per ray does not degrade as severely as one might think. Adding Whitted style illumination clearly reduces performance per ray by nearly a factor of 2, but adding distribution ray tracing on top of this is not an extreme leap. In practice it's at most a factor of 2 (for our stress test scene), but can even have improved performance due to the higher density of primary rays (all our tests use 64 samples per pixel). As an important note, even our RCS time include much more advanced shading than that used in our earlier TOG paper. In this work, we normalize camera rays, evaluate fresnel coefficients, and chose not to use special primitive intersection tricks relying on common origin. This accounts for the approximately 2.5x difference in performance from our TOG paper results.

Why does it work?



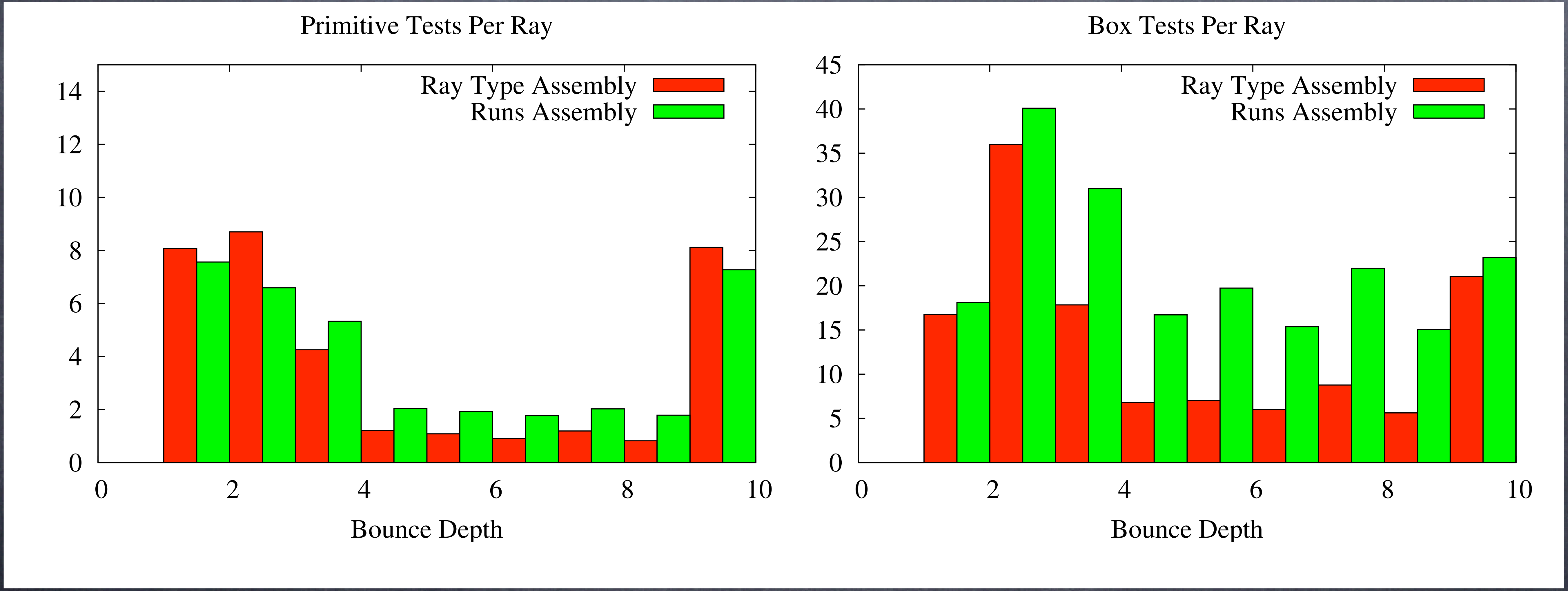
These graphs demonstrate the number of primitive and box tests per ray as we vary the ray packet size for a whitted ray tracing conference scene. The different colors correspond to different ray packet size, while each group of packet sizes is a specific Ray Type (primary, shadow, reflection). As can be seen, the 8x8 packets achieve a sort of sweet spot for this scene balancing amortization of box tests versus increasing “bloat” during primitive intersection. You can also see that while primary rays benefit significantly from growing packet size, shadow and reflection rays respond poorly. This is what I believe is the best metric of “coherence”. I would also like to note that this is probably the most hand-wavy magical part of this and other interactive packet based work: how do you choose packet size?

How does it behave?



Once we demonstrated that Whitted ray tracing was achieving benefits due to packets and basic settings for distribution ray tracing worked as well, it became obvious it may fail for more extreme settings. This graph is one of many we generated showing that even somewhat severe changes in "blur" factors does not totally destroy our approach. The graphs for other effects were similar, although aperture is an interesting effect because it affects all further dimensions (light samples, etc). Just increasing light source size only effects shadow rays, by contrast aperture demonstrates how all ray types differ.

What about runs?



Earlier we proposed two different simplified grouping methods: runs and ray type. In practice, we've found about a 10% overall difference with ray type coming out ahead. I thought this was strange since Ray Type seemed like a better approach due to its invariance to ordering. These graphs demonstrate primitive tests per ray and box tests per ray for the conference scene under distribution ray tracing. Each column demonstrates this variable for increasing bounce depth with RayType in red and Runs in green. The final column shows an average over all depths that takes into account the high number of "early bounce" rays. This is the primary reason that ray type assembly does not usually greatly improve upon runs assembly: ray tree attenuation hides high bounce depths. Runs clearly performs quite a bit more box tests relative to RayType especially for higher bounces, but this is hidden. We believe in the future, ray type grouping will pull further ahead if higher bounce depths are needed as for example highly specular chains.

Questions?

- For more information see my research page:
- http://www.cs.utah.edu/~boulos/research.htm
- Or the SIGGRAPH 2006 course webpage:
- http://www.cs.utah.edu/~shirley/irt/
-