



Introduction to Real-Time Shading with Many Lights

Part 1/4 - 60 min

Presenter: Ola Olsson
Chalmers University of Technology

SA2014.SIGGRAPH.ORG Efficient Real-Time Shading with Many Lights 2014 SPONSORED BY  

- The first part then is going to be presented by me.
- In this part we will provide an overview of techniques that have been, and still are, used in real-time shading.
- The main focus will be on clustered shading, as this is the most advanced and efficient technique today.

More Lights Please!

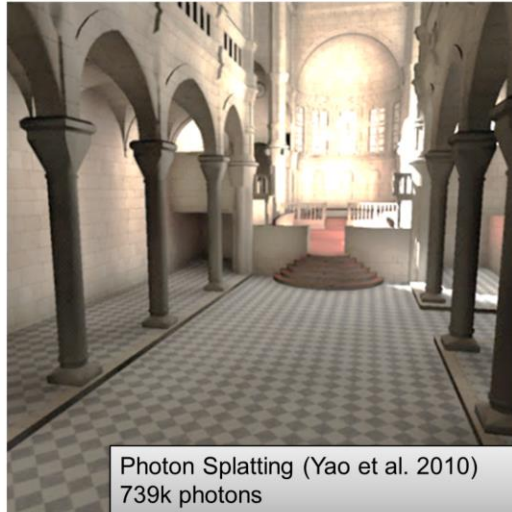


▶ Global Illumination

- ▶ Virtual Point Lights
- ▶ Photon Splatting

▶ Complex Lights

- ▶ Area / Volume



Photon Splatting (Yao et al. 2010)
739k photons



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- So. While I think that you, being a quite advanced audience, have fairly concrete ideas on what many lights can be used for.
- I'll give some illustration as to things that I keep in the back of my head at times like these.
- To the right here we see an image generated using Photon Splatting to visualize the results of global light transport.
- With enough lights, these kind of techniques become possible.

More Lights Please!



► Artistic Freedom

► Lighting design.

Need For Speed: The Run, ~2600 lights [1]



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- More artistic renderings.
- This scene from Need For Speed: The Run, contains around 2600 lights.
- In movies (animated and otherwise), lots of lights are used to achieve a particular aesthetic.
- This is increasingly going to be the case for games too.
- A benefit to using lights to represent scene illumination, as opposed to baking, is that dynamic geometry is affected

More Lights Please!

► Dynamic Lights

- Explosions.
- Particle Effects.
- Headlights.
- Torches.
- Lasers.
- And so on...

Starcraft 2 [Blizzard 2010]

+25
lights



- In games there are a practically limitless number of things that could emit light,
- if we had an efficient way to compute their contributions.
- It's safe to say that current games have not exhausted the possibilities.
- In the starcraft example, there is apparently only a single light <click>.
- However all the muzzle flashes are only additive billboards.

Talk Outline



- ▶ Forward Shading
 - ▶ Lights and geometry batched together.
- ▶ Deferred Shading
 - ▶ Decouples lights from geometry.
- ▶ Tiled Deferred/Forward Shading
 - ▶ Removes bandwidth bottleneck.
- ▶ Clustered Deferred/Forward Shading
 - ▶ Better robustness, scaling, transparency, MSAA.



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



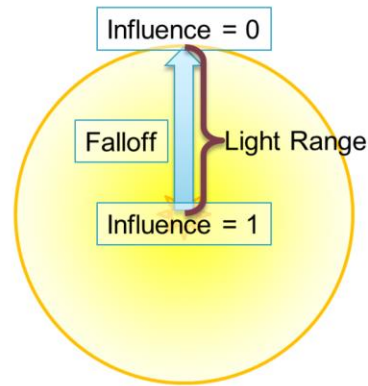
- Our historical and architectural re-cap starts with traditional forward shading, which was predominant until not so long ago.
 - And still is in the mobile space.
- Next we look at deferred shading, which was the first widespread technique to bring many lights to games.
- Then we will discuss tiled deferred shading which has seen adoption in many modern high-end games.
- More recently Clustered shading further improves efficiency and scalability over tiled shading.
- We will also discuss forward shading, using both clustered and tiled shading...
- which allows the use of transparency and MSAA, while retaining much of the goodness with deferred techniques.

Problem Definition



► Real-time shading algorithm

- Thousands of lights
 - Limited range light
 - No shadows
 - Actually, shadows later!
- Fully dynamic
 - Lights and Geometry



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- The algorithms we are going to talk about target thousands of lights in real time,
- The lights have a limited range, with some falloff which goes to 0 at the boundary.
- This means that lights are not physical, but this is the normal procedure in games.
- We also do not consider shadows, which is covered later in the course.
- There is no pre-computation so all geometry and lights are allowed to change freely from frame to frame.

Light Assignment – *Not* Shading!



- ▶ Question: *Which lights affect what geometry.*
- ▶ Solution: *Light Assignment*
 - ▶ (*Light Culling*)



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- Despite the title of this course, it is not actually about shading at all!
- This is a name we have inherited from forward shading / deferred shading etc.
- In reality the course is about the above question,
- That is, working out which lights affect what pieces of geometry.
- We call this 'light assignment', but is also know as 'light culling'.



Traditional Forward Shading

SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

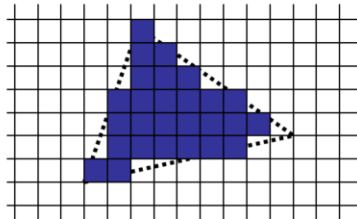
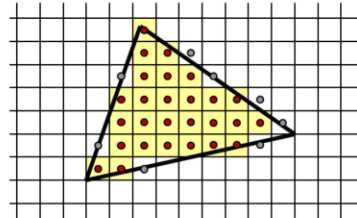
SPONSORED BY



(Traditional) Forward Shading



- ▶ The traditional approach
 - ▶ Single pass
 - ▶ Shading in shaders
- ▶ Rasterization generates fragments
 - ▶ Each fragment is shaded
 - ▶ Needs set of lights.
 - ▶ Produces a *color*



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- The traditional method for real-time shading is called forward shading and used to be the only method during the first decade, or so, of consumer GPUs.
- It is still dominant on mobile hardware.
- In this technique, there is only a single pass over the geometry drawing into a frame buffer accumulating the final image.
- Geometry is rasterized, shading is performed and the frame buffer is updated.
- Shading is performed in the fragment shaders.

Forward Shading



- ▶ Which lights to use?
 - ▶ Gather before draw call.
- ▶ Want minimal set of lights.
 - ▶ Small batches.
- ▶ Want quick drawing
 - ▶ Few batches!
 - ▶ Large batches.
 - ▶ Lower GPU/API overhead.
 - ▶ Fewer material changes.



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY

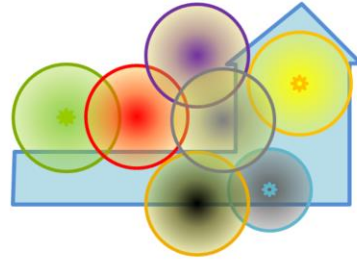


- Thus, we need to round up the set of relevant lights before each draw call.
- and assign lights per chunk/batch of primitives.
- To minimize number of lights, we want to make sure a batch is small, geometrically.
- But to draw fast, keep the GPU busy and avoid API overhear, we want large batches.
- And don't care so much about geometric shape.
- This is a fundamental conflict, where the best we can manage is a compromise. But it is a difficult one to strike.

Forward Shading -Problem Examples



- ▶ Few large objects
- ▶ Many small lights



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY

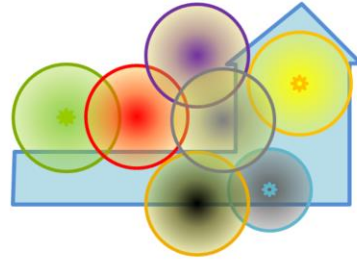


- Now, this conflict runs deeper than simply batch size.
- The basic problem is that we have to assign lights based on the size of geometry chunks.
- Therefore, on the one hand, we might have a situation with a few large objects, and many small lights

Forward Shading - Problem Examples



- ▶ Few large objects
- ▶ Many small lights
- ▶ Redundant shading work



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY

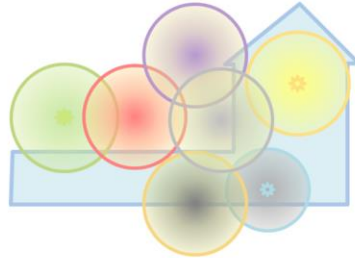


- This leads to lots of wasted effort in shading lights that only affect a portion of the geometry.

Forward Shading - Problem Examples



- ▶ Few large objects
- ▶ Many small lights
- ▶ Redundant shading work
- ▶ Many small objects



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY

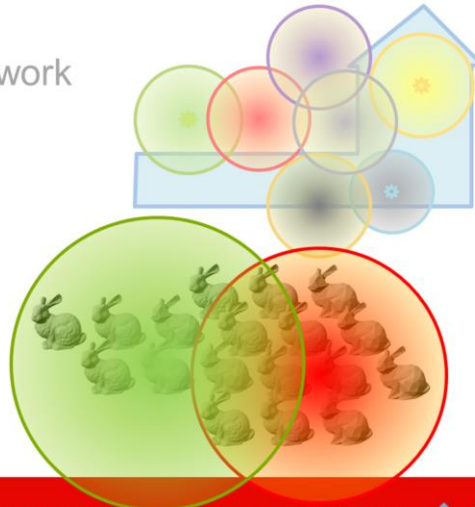


- On the other hand, we may have many small objects.

Forward Shading - Problem Examples



- ▶ Few large objects
- ▶ Many small lights
- ▶ Redundant shading work
- ▶ Many small objects
- ▶ Few large lights



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY

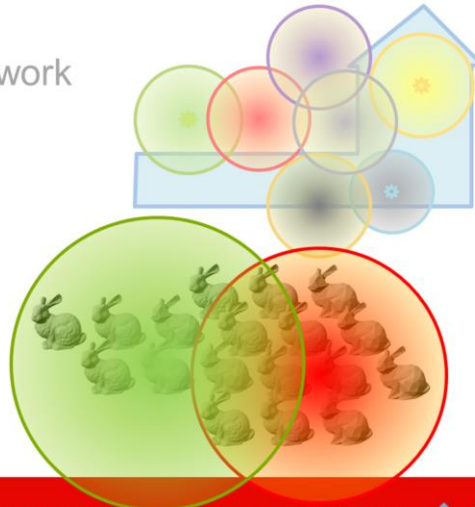


- And a few large lights, <click>

Forward Shading - Problem Examples



- ▶ Few large objects
- ▶ Many small lights
- ▶ Redundant shading work
- ▶ Many small objects
- ▶ Few large lights
- ▶ Redundant light assignment work



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY

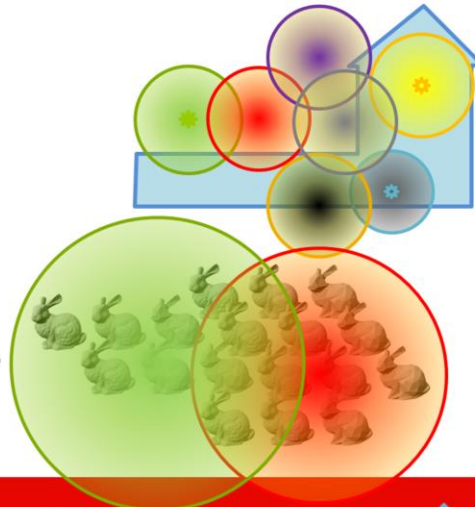


- in which case we will spend a lot of effort to individually discover that they are affected by the same lights.

Forward Shading - Problem Examples



- ▶ Few large objects
 - ▶ Many small lights
- AND
- ▶ Many small objects
 - ▶ Few large lights
-
- ▶ Large triangles.
 - ▶ Varying light density.



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- So the conflict runs deeper than just batch size...<click>
- And of course, in the same scene we might have both houses and rabbits, which means it is difficult or impossible to ensure a good performance balance.
- Note that the large object could be a single triangle, and in general triangle size and varying light density makes this a very difficult problem to solve well.

Summary: Forward Shading



THE GOOD

- ▶ Single Pass
- ▶ Low storage overhead
 - ▶ Single Frame Buffer
- ▶ Transparency works.
- ▶ MSAA works
- ▶ Simple if only few lights
 - ▶ E.g., the sun
- ▶ Varying shading models is easy.



THE BAD

- ▶ Overdraw
- ▶ Shader management
 - ▶ Permutations of #lights/type.
- ▶ Batching coupled with lighting

SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- To summarize, the main good thing about traditional forward shading is that

Summary: Forward Shading



THE GOOD

- ▶ Single Pass
- ▶ Low storage overhead
 - ▶ Single Frame Buffer
- ▶ Transparency works.
- ▶ MSAA works
- ▶ Simple if only few lights
 - ▶ E.g., the sun
- ▶ Varying shading models is easy.



THE BAD

- ▶ Overdraw
- ▶ Shader management
 - ▶ Permutations of #lights/type.
- ▶ Batching coupled with lighting

SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- Everything can be done in a single pass with a single frame buffer, just storing the final color
- This enables supporting transparency and MSAA without much ado.

Summary: Forward Shading



THE GOOD

- ▶ Single Pass
- ▶ Low storage overhead
 - ▶ Single Frame Buffer
- ▶ Transparency works.
- ▶ MSAA works
- ▶ Simple if only few lights
 - ▶ E.g., the sun
- ▶ Varying shading models is easy.



THE BAD

- ▶ Overdraw
- ▶ Shader management
 - ▶ Permutations of #lights/type.
- ▶ Batching coupled with lighting

SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- If there are *not* many lights, this is a simple and high-performing approach.

Summary: Forward Shading



THE GOOD

- ▶ Single Pass
- ▶ Low storage overhead
 - ▶ Single Frame Buffer
- ▶ Transparency works.
- ▶ MSAA works
- ▶ Simple if only few lights
 - ▶ E.g., the sun
- ▶ Varying shading models is easy.



THE BAD

- ▶ Overdraw
- ▶ Shader management
 - ▶ Permutations of #lights/type.
- ▶ Batching coupled with lighting

SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- And it is trivial to change shader for each draw call, so custom shaders is easy.

Summary: Forward Shading



THE GOOD

- ▶ Single Pass
- ▶ Low storage overhead
 - ▶ Single Frame Buffer
- ▶ Transparency works.
- ▶ MSAA works
- ▶ Simple if only few lights
 - ▶ E.g., the sun
- ▶ Varying shading models is easy.



THE BAD

- ▶ Overdraw
- ▶ Shader management
 - ▶ Permutations of #lights/type.
- ▶ Batching coupled with lighting

SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- On the other hand, the technique suffers from overdraw which means that parts of the scene may be shaded only to be drawn over later.

Summary: Forward Shading



THE GOOD

- ▶ Single Pass
- ▶ Low storage overhead
 - ▶ Single Frame Buffer
- ▶ Transparency works.
- ▶ MSAA works
- ▶ Simple if only few lights
 - ▶ E.g., the sun
- ▶ Varying shading models is easy.



THE BAD

- ▶ Overdraw
- ▶ Shader management
 - ▶ Permutations of #lights/type.
- ▶ Batching coupled with lighting

SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- Managing shaders can be a challenge, at least it used to be, as each shader must support all permutations of lights.
- This can push up register use, at the very least.

Summary: Forward Shading



THE GOOD

- ▶ Single Pass
- ▶ Low storage overhead
 - ▶ Single Frame Buffer
- ▶ Transparency works.
- ▶ MSAA works
- ▶ Simple if only few lights
 - ▶ E.g., the sun
- ▶ Varying shading models is easy.



THE BAD

- ▶ Overdraw
- ▶ Shader management
 - ▶ Permutations of #lights/type.
- ▶ Batching coupled with lighting

SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- Finally, we have that problem of assigning lights at the right granularity which makes an efficient implementation difficult.

Summary: Forward Shading



THE GOOD

- ▶ Single Pass
- ▶ Low storage overhead
 - ▶ Single Frame Buffer
- ▶ Transparency works.
- ▶ MSAA works
- ▶ Simple if only few lights
 - ▶ E.g., the sun
- ▶ Varying shading models is easy.



THE BAD

- ▶ Overdraw
- ▶ Shader management
 - ▶ Permutations of #lights/type.
- ▶ Batching coupled with lighting

SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- I want to point out that none of the problems are necessarily fundamental,
- But they are important considerations,
- and have, at least historically, been important obstacles.



Traditional Deferred Shading

SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



(Traditional) Deferred Shading



- ▶ Modern form introduced in 1990 [Saito90]
 - ▶ Term coined later by Tebbs et al [Tebbs92]
 - ▶ Consumer GPUs in 2004 [Hargreaves04]
 - ▶ Games around 2005 [Shishkovtsov05]
- ▶ Goal:
 - ▶ Decouple shading from geometric complexity



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- Traditional deferred shading was introduced in its modern form with G-Buffers in 1990.
- Although the term 'deferred shading' was coined later.
- Started becoming mainstream in games around 2005.
- The basic goal is to solve both the overdraw problem and light assignment problem by decoupling geometry from shading.

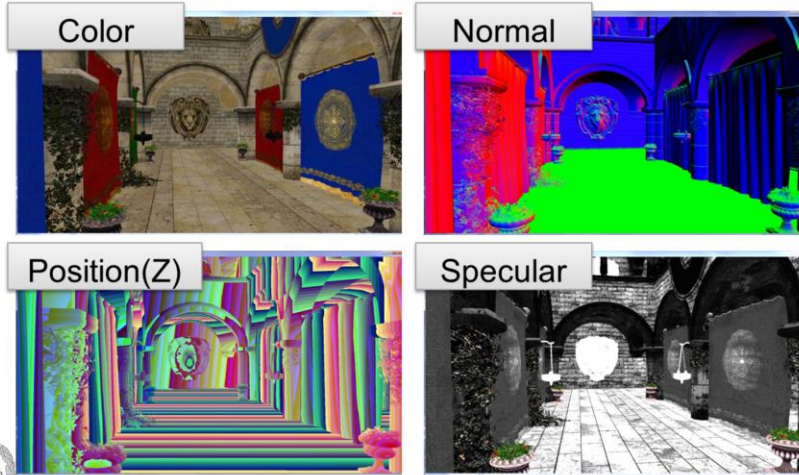
(Traditional) Deferred Shading

- ▶ Key idea
 - ▶ Defer shading computations
- ▶ First do a geometry pass
 - ▶ Sample geometry as per normal
 - ▶ But, no shading
 - ▶ Store geometry attributes per pixel (position, normal, albedo)
- ▶ Then do a shading pass
 - ▶ Use attributes from first pass.
 - ▶ Process lights one at a time.
 - ▶ Treat lights as geometry!



- The key idea is to defer the shading to its own pass.
- Such that in the first pass, geometry attributes are just sampled, and stored in G-Buffers
- And then a separate shading pass is performed, which is thus independent of geometric complexity, and indeed representation.

Render G-Buffers



- So the geometry pass populates the G-Buffers with sampled geometry attributes.
- These are all the attributes that can change per pixel, and in addition a material ID can be used.


Draw Light Bounds

- ▶ For each light
 - ▶ Draw Light Bounds
 - ▶ For each fragment
- Read G-Buffers
- Compute Shading
- Add to frame buffer



- Next we draw the bounding volumes of the lights, as polygon geometry to the screen.
- In the fragment shader of these we compute the shading.
- Which is then blended into the frame buffer.
- This is done independently for each light.

Deferred Light Shader



```


uniform vec3 lightPosition;
uniform vec3 lightColor;
uniform float lightRange;


void main()
{
    vec3 color = texelFetch(colorTex, gl_FragCoord.xy);
    vec3 specular = texelFetch(specularTex, gl_FragCoord.xy);
    vec3 normal = texelFetch(normalTex, gl_FragCoord.xy);
    vec3 position = fetchPosition(gl_FragCoord.xy);



    vec3 shading = doLight(position, normal, color,
                           specular, lightPosition,
                           lightColor, lightRange);

    resultColor = vec4(shading, 1.0);
}

```





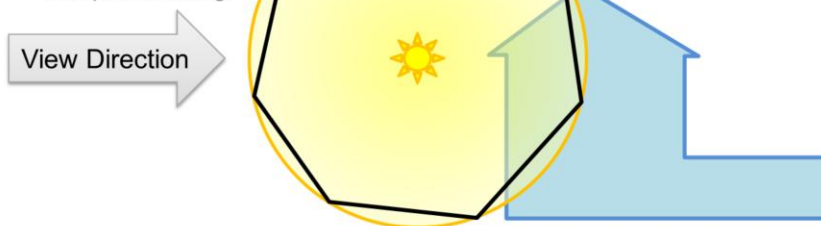
SA2014.SIGGRAPH.ORG Efficient Real-Time Shading with Many Lights 2014 SPONSORED BY  

- Here is a somewhat simplified light shader.
- So this is a fragment shader that is used when drawing a *light*.
- It is thusly executed once for each pixel covered by the light bounding sphere.
- The shader has as uniform parameters the light attributes, position etc, these are the same for each fragment.
- First all attributes are fetched from the G-Buffers.
 - Note that the screen space coordinate of the fragment is in `gl_FragCoord.xy`
- Then shading is computed using these and the uniform attributes of the light.
- And output to the color of the pixel.

Stencil Buffer Optimization [Arvo03]



- ▶ Analogous to stencil shadows.
- ▶ Draw light volume twice.
 - ▶ First back faces.
 - ▶ Set stencil mask on depth fail.
 - ▶ Then front faces.
 - ▶ Test stencil mask.
 - ▶ Compute shading.



Stencil Buffer Optimization [Arvo03]



- ▶ Analogous to stencil shadows.

- ▶ Draw light volume twice.

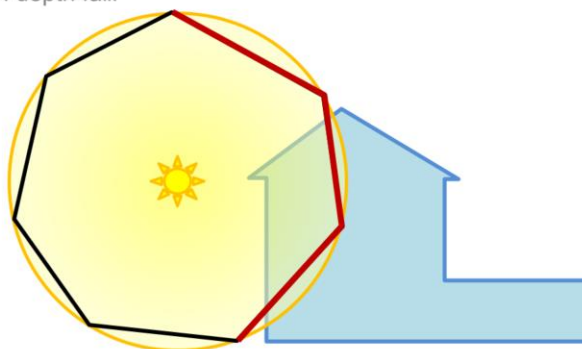
- ▶ **First back faces.**

- ▶ Set stencil mask on depth fail.

- ▶ **Then front faces.**

- ▶ Test stencil mask.

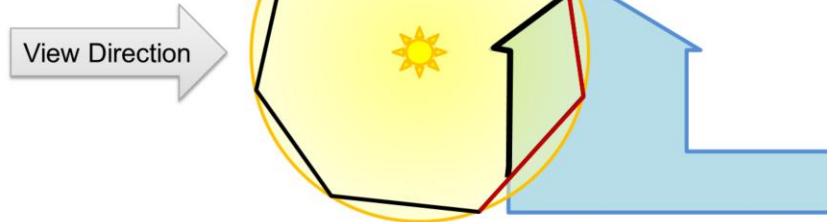
- ▶ Compute shading.



Stencil Buffer Optimization [Arvo03]



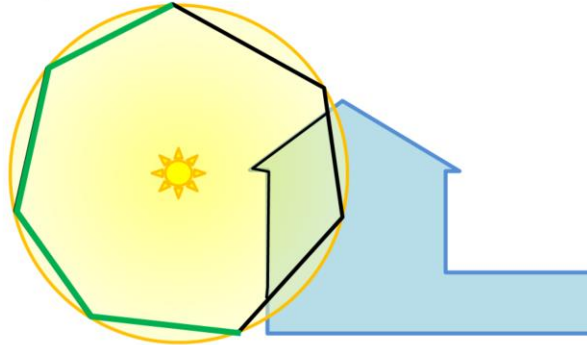
- ▶ Analogous to stencil shadows.
- ▶ Draw light volume twice.
 - ▶ First back faces.
 - ▶ **Set stencil mask on depth fail.**
 - ▶ Then front faces.
 - ▶ Test stencil mask.
 - ▶ Compute shading.



Stencil Buffer Optimization [Arvo03]



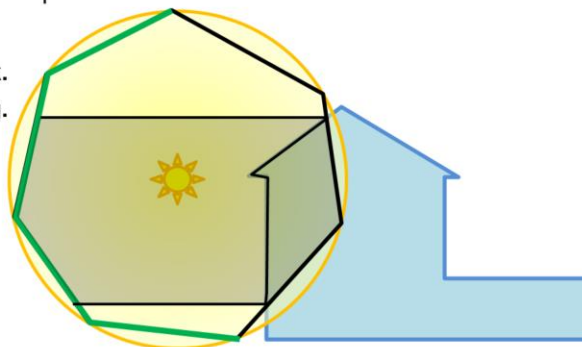
- ▶ Analogous to stencil shadows.
- ▶ Draw light volume twice.
 - ▶ First back faces.
 - ▶ Set stencil mask on depth fail.
 - ▶ **Then front faces.**
 - ▶ Test stencil mask.
 - ▶ Compute shading.



Stencil Buffer Optimization [Arvo03]



- ▶ Analogous to stencil shadows.
- ▶ Draw light volume twice.
 - ▶ First back faces.
 - ▶ Set stencil mask on depth fail.
 - ▶ Then front faces.
 - ▶ Test stencil mask.
 - ▶ Compute shading.



Other Variants



- ▶ **Deferred Lighting**
 - ▶ Factor out specular and diffuse color
 - ▶ G-Buffer only store normal and shininess
 - ▶ Output diffuse and specular shading
 - ▶ Second geometry pass which multiplies colors
- ▶ **Light PrePass**
 - ▶ Much like the above
 - ▶ But with monochromatic specular highlight
- ▶ **Similar performance as deferred**
 - ▶ Only improves constant factors.



Limits shading model even further



Summary: Trad. Deferred Shading



THE GOOD

- ▶ Trivial light management
 - ▶ Enables many lights
- ▶ Simple (light) shader management
- ▶ High Shading efficiency
 - ▶ Only shade lit samples
- ▶ No overdraw
- ▶ Single geometry pass
- ▶ Shadow map reuse



THE BAD

- ▶ No transparency
- ▶ Large frame buffer
 - ▶ Especially with MSAA
 - ▶ Accumulates light in frame buffer
 - ▶ High precision needed
- ▶ Difficult to do multiple shading models
 - ▶ Custom shaders
- ▶ High memory bandwidth usage

SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- The large advantages are
- Trivial light management, you just cull and draw the light bounds,
- And light shaders only need to support one light type.

Summary: Trad. Deferred Shading



THE GOOD

- ▶ Trivial light management
 - ▶ Enables many lights
- ▶ Simple (light) shader management
- ▶ High Shading efficiency
 - ▶ Only shade lit samples
- ▶ No overdraw
- ▶ Single geometry pass
- ▶ Shadow map reuse



THE BAD

- ▶ No transparency
- ▶ Large frame buffer
 - ▶ Especially with MSAA
 - ▶ Accumulates light in frame buffer
 - ▶ High precision needed
- ▶ Difficult to do multiple shading models
 - ▶ Custom shaders
- ▶ High memory bandwidth usage

SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- The large advantages are
- Efficiency is very high, in that only samples that are actually affected by lights need to be shaded.

Summary: Trad. Deferred Shading



THE GOOD

- ▶ Trivial light management
 - ▶ Enables many lights
- ▶ Simple (light) shader management
- ▶ High Shading efficiency
 - ▶ Only shade lit samples
- ▶ No overdraw
- ▶ Single geometry pass



Shadow map reuse

THE BAD

- ▶ No transparency
- ▶ Large frame buffer
 - ▶ Especially with MSAA
 - ▶ Accumulates light in frame buffer
 - ▶ High precision needed
- ▶ Difficult to do multiple shading models
 - ▶ Custom shaders
- ▶ High memory bandwidth usage

SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- There is no overdraw, and only a single geometry pass.

Summary: Trad. Deferred Shading



THE GOOD

- ▶ Trivial light management
 - ▶ Enables many lights
- ▶ Simple (light) shader management
- ▶ High Shading efficiency
 - ▶ Only shade lit samples
- ▶ No overdraw
- ▶ Single geometry pass
- ▶ Shadow map reuse



THE BAD

- ▶ No transparency
- ▶ Large frame buffer
 - ▶ Especially with MSAA
 - ▶ Accumulates light in frame buffer
 - ▶ High precision needed
- ▶ Difficult to do multiple shading models
 - ▶ Custom shaders
- ▶ High memory bandwidth usage

SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- Also, as lights are processed sequentially shadow maps storage can be reused between lights

Summary: Trad. Deferred Shading



THE GOOD

- ▶ Trivial light management
 - ▶ Enables many lights
- ▶ Simple (light) shader management
- ▶ High Shading efficiency
 - ▶ Only shade lit samples
- ▶ No overdraw
- ▶ Single geometry pass
- ▶ Shadow map reuse



THE BAD

- ▶ No transparency
- ▶ Large frame buffer
 - ▶ Especially with MSAA
 - ▶ Accumulates light in frame buffer
 - ▶ High precision needed
- ▶ Difficult to do multiple shading models
 - ▶ Custom shaders
- ▶ High memory bandwidth usage

SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- On the down side,
- Transparency is not readily supported

Summary: Trad. Deferred Shading



THE GOOD

- ▶ Trivial light management
 - ▶ Enables many lights
- ▶ Simple (light) shader management
- ▶ High Shading efficiency
 - ▶ Only shade lit samples
- ▶ No overdraw
- ▶ Single geometry pass
- ▶ Shadow map reuse



THE BAD

- ▶ No transparency
- ▶ Large frame buffer
 - ▶ Especially with MSAA
 - ▶ Accumulates light in frame buffer
 - ▶ High precision needed
- ▶ Difficult to do multiple shading models
 - ▶ Custom shaders
- ▶ High memory bandwidth usage

SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- On the down side,
- We get very large frame buffers, especially with MSAA
- And because we need higher precision for light accumulation.

Summary: Trad. Deferred Shading



THE GOOD

- ▶ Trivial light management
 - ▶ Enables many lights
- ▶ Simple (light) shader management
- ▶ High Shading efficiency
 - ▶ Only shade lit samples
- ▶ No overdraw
- ▶ Single geometry pass
- ▶ Shadow map reuse



THE BAD

- ▶ No transparency
- ▶ Large frame buffer
 - ▶ Especially with MSAA
 - ▶ Accumulates light in frame buffer
 - ▶ High precision needed
- ▶ Difficult to do multiple shading models
 - ▶ Custom shaders
- ▶ High memory bandwidth usage

SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- On the down side,
- It is more difficult to do custom shaders for certain geometry.

Summary: Trad. Deferred Shading



THE GOOD

- ▶ Trivial light management
 - ▶ Enables many lights
- ▶ Simple (light) shader management
- ▶ High Shading efficiency
 - ▶ Only shade lit samples
- ▶ No overdraw
- ▶ Single geometry pass
- ▶ Shadow map reuse



THE BAD

- ▶ No transparency
- ▶ Large frame buffer
 - ▶ Especially with MSAA
 - ▶ Accumulates light in frame buffer
 - ▶ High precision needed
- ▶ Difficult to do multiple shading models
 - ▶ Custom shaders
- ▶ High memory bandwidth usage

SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- And most problematically, high memory bandwidth requirements,



Modern Shading Techniques

SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- This last problem brings us to the topic of modern shading techniques, so we'll look into this in some more detail.

Modern Shading Techniques



► Observations:

1. GPU Compute >> Bandwidth
2. Better GPU general purpose capability

► Conclusion:

- Explore alternatives to rasterization!

► Broad impact

- Many algorithms forced triangles!
 - Accurate shadows
 - Culling
 - Various splatting algorithms



SA2014.SIGGRAPH.ORG

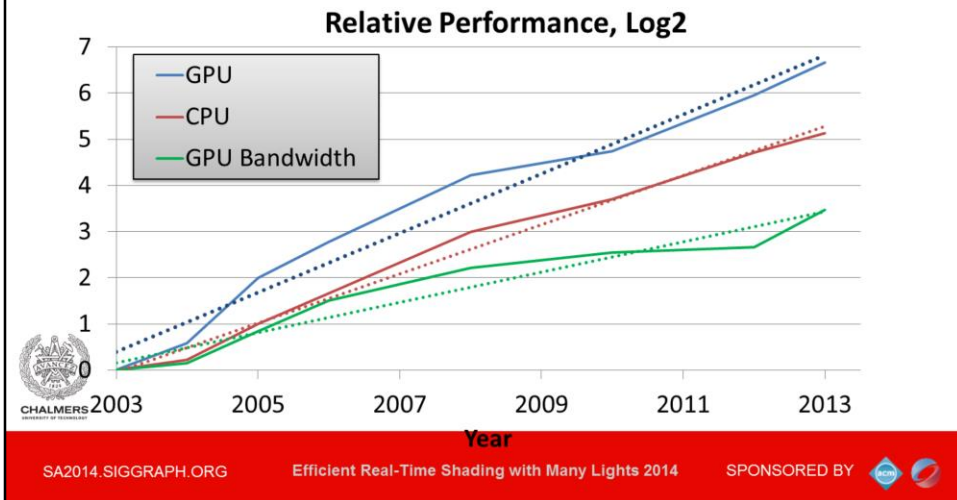
Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



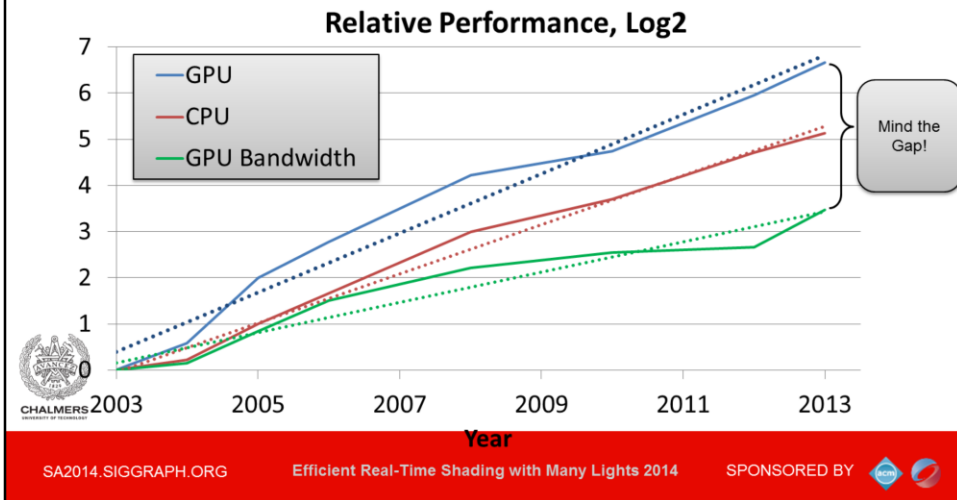
- The modern approaches to real-time shading with many lights all take their in the following 2 observations
 - First, GPU Compute Capacity is greater than the memory bandwidth, and grows faster .
 - Second, the GPU general purpose programming models and power of these cores has improved tremendously over the last years.
- This leads to the conclusion that we ought to explore more clever alternatives to rasterization based techniques, such as deferred shading, because they are bandwidth intensive.
- These observations have much broader impact as because of how GPUs were designed not long ago our community has spent a lot of effort developing algorithms that map well to triangle rasterization, even when this has been less than intuitive. To do so was simply the only path to real-time performance, as evidenced by the many examples, and I'm sure you can think of several of your own.
- Today this is not the case, with indirect drawing and programmable shader cores, we can, and must, revisit the same problems with a fresh approach.

GPU Performance Trends



- To illustrate this process, this graph shows the relative performance trends of intel enthusiast level CPUs and NVIDIA GPUs.
- The green line also plots the memory bandwidth of GPUs,
- As this is a logarithmic plot, we see a fairly clear exponential growth in all three.

GPU Performance Trends



- However, note the great difference in exponent!
- This means that compute capacity is continually outpacing memory bandwidth,
- Any algorithm that is bottle necked by bandwidth will scale along this line.
- Whereas a compute bound algorithms will scale much, much, better.
- So we need to be mindful of this widening gap.

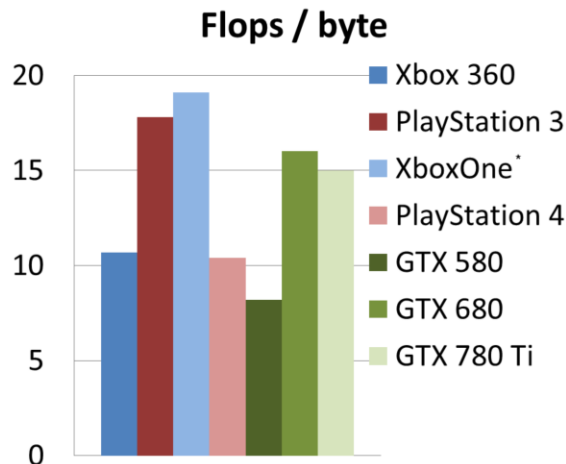
GPU Compute / Bandwidth ratio



- ▶ 1 byte / 10-20 FLOPs
- ▶ Lots of ops/byte
- ▶ Bias towards compute.



* XBoxOne ESRAM
Cache not included.



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- Currently the gap is a span of some 10 to 20 floating point operations that can, or must be, performed for each byte of data loaded to reach peak performance.
- So if we load a single float, we need to do about 40 to 80 operations locally before fetching another...
- Of course texture caches and constant registers help a lot, so it is never quite that simple...
- ...but the trend is clear and shows no sign of slacking off.



► Tiled Deferred, Tiled Forward (Forward+)

Tiled Shading

SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- This brings us to the first of the modern techniques that has been developed in this new, bandwidth constrained and compute oriented, landscape.
- Collectively called Tiled Shading, it covers both deferred and forward variants,
- The forward variant has been re-branded Forward+ by AMD, which obscures its nature.

The Bandwidth problem



- ▶ New type of overdraw
 - ▶ Light overdraw
- ▶ N lights cover a certain pixel ->
 - ▶ N reads from the **same** G-Buffer location
- ▶ Deferred Shading

```
for each light
  for each covered pixel
    read G-Buffer
    compute shading
    read + write frame buffer
```



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- To motivate tiled shading, we will look at why traditional deferred shading is bandwidth bound (or will be...)
- As we are now drawing the lights, in the shading pass, we have overdraw when many lights overlap the same pixel.
- Schematically deferred shading looks like this, we iterate over each light,
- and then in parallel, by drawing the bounding volume, over all the fragments.

The Bandwidth problem



```
for each light
  for each covered pixel
    read G-Buffer
    compute shading
    read + write frame buffer
```



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- The problem is from the fact that the innermost loop is over the pixels,
- This which requires repeated reading, and writing of the G-Buffers and frame buffer.
- So it is pretty clear that we need to get this out of the inner loop somehow, especially since G-buffers contain lots of data.

The Bandwidth problem



```
for each light
  for each covered pixel
    read G-Buffer
    compute shading
    read + write frame buffer
```

Re-order loops
Load/store -> Outer loop

```
for each pixel
  read G-Buffer
  for each affecting light
    compute shading
  write frame buffer
```



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- So we want to re-arrange this loop to make the innermost loop iterate over the lights instead.
- Then we can hoist the G-Buffer read to the outer loop, only reading a single time.<click>
- Then we get this nice compute oriented loop.
- And finally a single write.
- This would effectively eliminate the bandwidth problem.
- So how do we go about this in practice?

The Bandwidth problem



- ▶ Requires
 - ▶ sequential access to lights for each pixel
- ▶ Global list
 - ▶ Inefficient
- ▶ List per pixel
 - ▶ Lots of data
 - ▶ Slow construction
- ▶ Share list in screen space tiles
 - ▶ E.g. 32 x 32 pixels
 - ▶ Simple construction
 - ▶ Little storage
 - ▶ 1 list / 1024 pixels
 - ▶ Coherent access



SA2014.SIGGRAPH.ORG

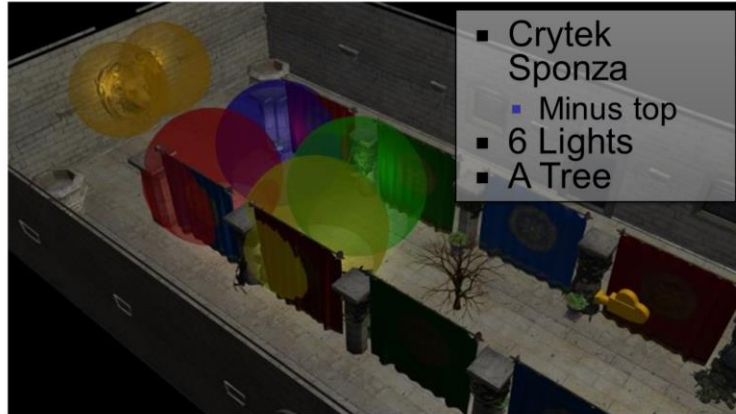
Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- We now need to access all the relevant lights for each pixel sequentially.
- Just using a global list of lights is of course terribly inefficient.
- At the other end of the spectrum, creating lists of lights for each pixel individually is both slow and requires lots of storage.
- Tiled shading strikes a balance, where we create lists for tiles of pixels.
- The list must be conservative, storing all lights that *may* affect any sample within the tile.
- So we trade some compute performance for bandwidth,
- which as we have seen is a pretty good gambit on modern GPUs.

Example Scene



- Crytek Sponza
 - Minus top
- 6 Lights
- A Tree



SA2014.SIGGRAPH.ORG

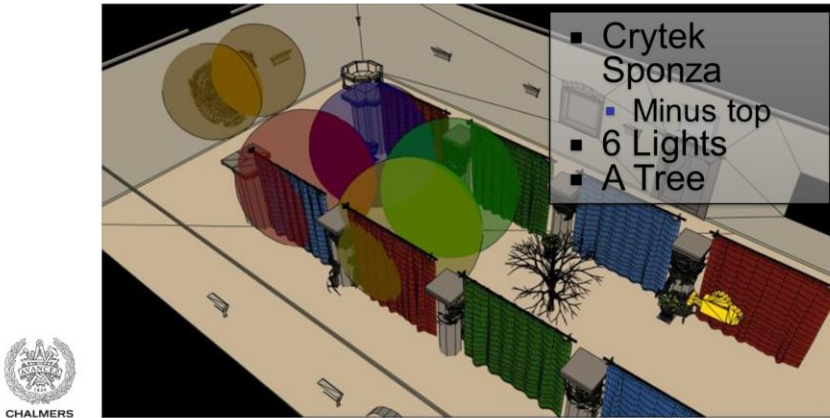
Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



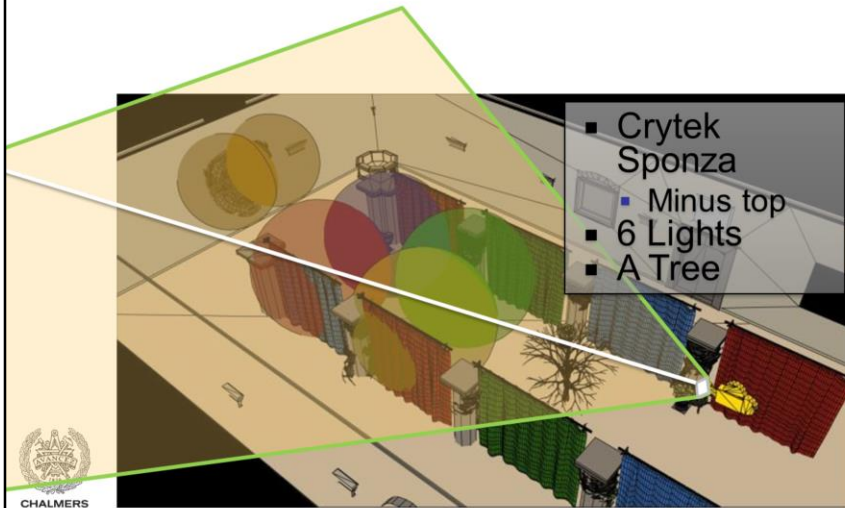
- We will be using this example scene to illustrate how tiled and clustered shading works.
- The scene is the usual Crytek Sponza scene, but with the top three quarters lifted off, to let us get a better view.
- I've added six lights of different colours, represented as spheres.
- To make the view more interesting, I've also added a tree.

Example Scene



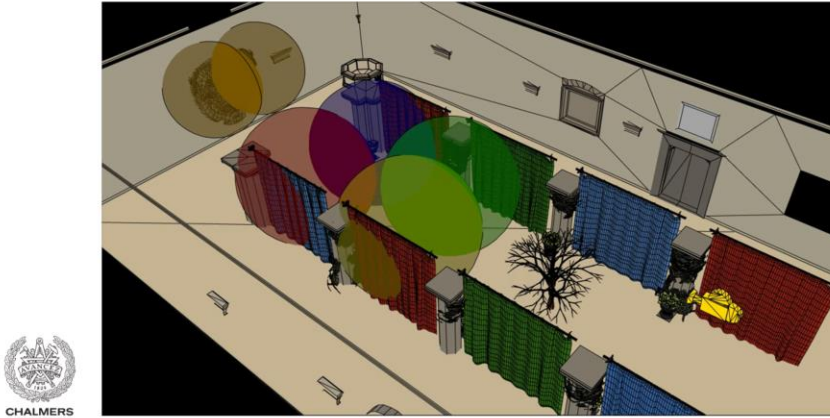
- The same view in outline mode, to make it more clear.

Example Scene



This is the viewpoint that will be used to demonstrate the algorithms, the yellow camera is looking through the tree towards the lion head on the wall.

Example Scene



- Shifting to that point of view.
- We see that there is a tree near the camera and then we're looking through the 6 lights to the wall in the background.

Tiled Shading



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



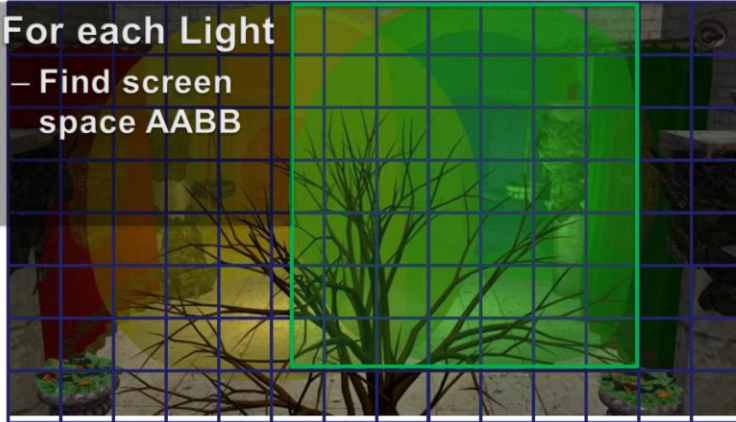
- With this I will summarize the tiled shading algorithm.
- Tiled shading can be implemented using either deferred or forward shading techniques, or a mix.
 - Note that AMD insists on calling “Tiled Forward Shading”, “Forward+”, it is however identical.
- The algorithm is conceptually very simple, and also quite easy to implement, at least in the simplest form.
- Performance can be very good, given the right circumstances.
- And fundamentally it is a 2D algorithm (we’ll come back to this).

- Screen space tiles
 - E.g. 32x32 pixels
 - Each contains list of lights



- The screen is divided into tiles, each covering say 32 by 32 pixels.
- Each tile contains a single list of all the lights that might influence any of the pixels inside.
- Note that this list is shared between the pixels, so overhead for list maintenance and fetching is low.

- For each Light
 - Find screen space AABB

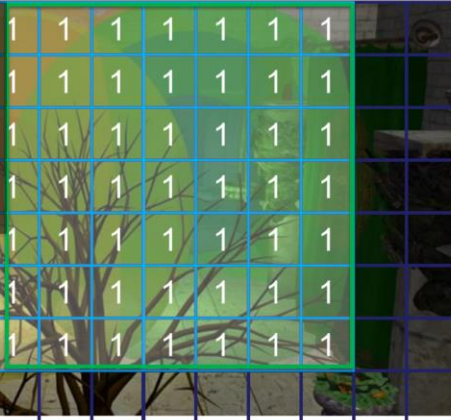


- To construct the lists, we might do as follows.
- For each light, establish the screen space bounding box, illustrated for the green light.

Tiled Shading

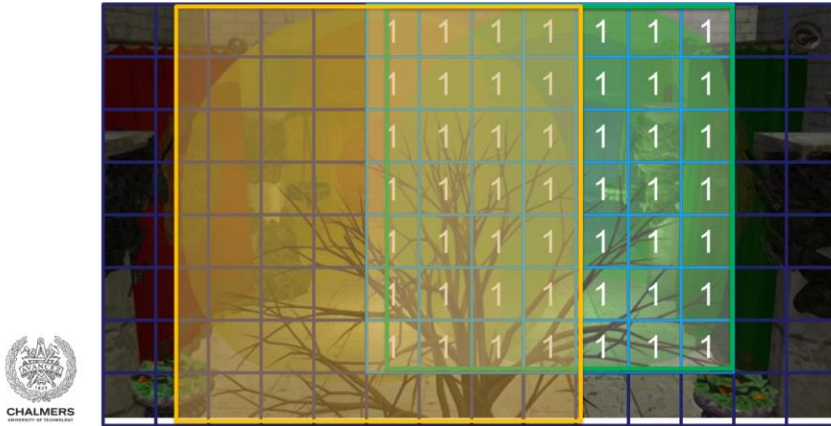
- For each Light

- Find screen space AABB
- Add light to affected tiles



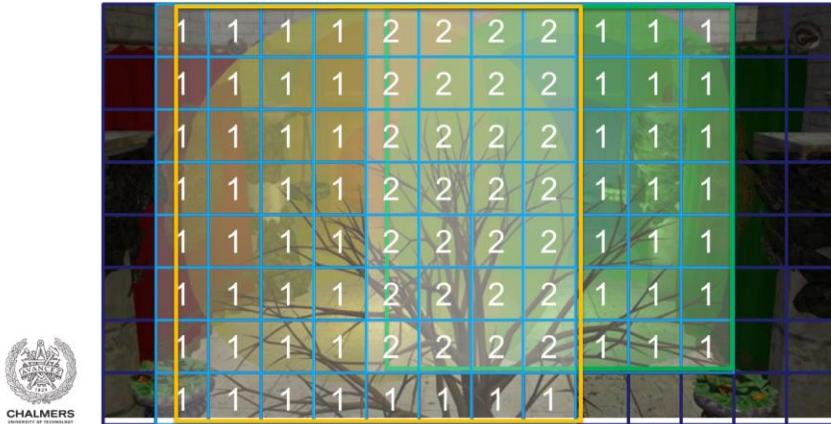
Then add the index of the light to all of the overlapped tiles.

Tiled Shading



Then repeat this process for all remaining lights.

Tiled Shading



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- The illustration only shows the counts, so you need to imagine the lists being built as well.
- In practice we'd also do a conservative per-tile min/max depth test, to cull away lights occupying empty space.

Depth Range Optimization

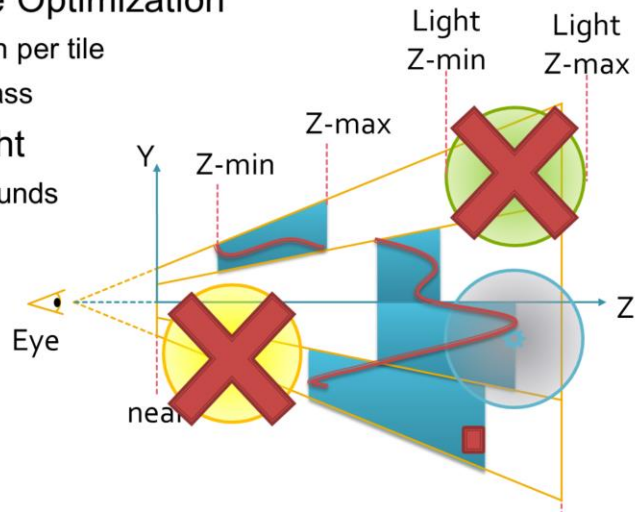


► Depth Range Optimization

- Min/Max depth per tile
- From Pre-z pass

► For each Light

- Test depth bounds



SA2014.SIGGRAPH.ORG

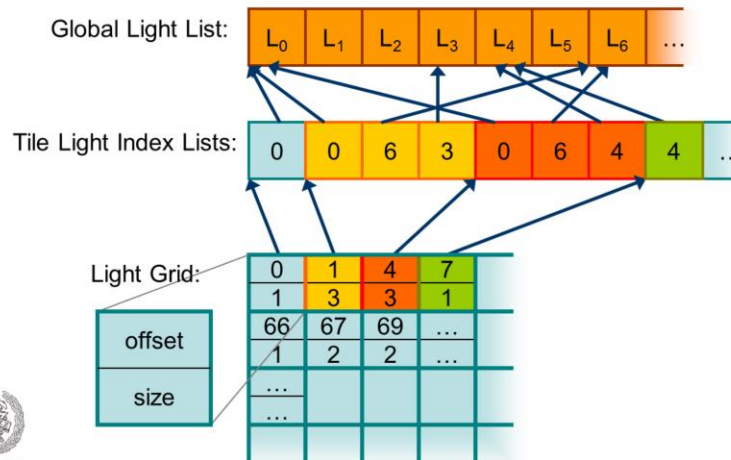
Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- Tiles in 1D, from side
- View Frustum
- 4 subdivisions
- Redline is geometry
- Min and max depth per tile
- Light range, rejected, completely hidden
- Another rejected, completely in front
- Rejected in one tile, not others

Tiled Shading Data



- After we have processed all the lights, we end up with a 2D grid such as this
- Each cell stores an offset and count that represent a range in a global buffer.
- This range contains a list of light indices indicating all the lights that the may affect the samples in the tile.

Tiled Shading



- ▶ Light Grid Provides
 - ▶ Access to light list for each pixel
 - ▶ List shared within tile
 - ▶ Low memory usage
 - ▶ Good access coherency
 - ▶ Not pixel exact
- ▶ Light Grid building
 - ▶ Is pretty quick
 - ▶ CPU for hundreds of lights



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



Tiled Deferred Shading



- ▶ 1. Render Scene to G-Buffers
 - ▶ Store geometry attributes per pixel
 - ▶ G-Buffers
- ▶ 2. Build Light Grid
- ▶ 3. Full Screen Quad (or CUDA, or Compute Shaders, or SPUs)
 - ▶ For each pixel
 - ▶ Fetch G-Buffer Data
 - ▶ Find Tile
 - ▶ Loop over lights and accumulate shading
 - ▶ Write shading



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



Full-Screen Tiled Shading Pass

Light Grid:

L ₀	L ₁	L ₂	L ₃	L ₄	L ₅	L ₆	L ₇	L ₈	L ₉	L ₁₀	L ₁₁	L ₁₂	L ₁₃	L ₁₄	L ₁₅
0	0	6	3	0	6	4	4

offset: 2
size: 2

Color

Normal

Position(Z)

Specular

```
out vec4 resultColor;  
  
void main()  
{  
    vec3 color = texelFetch(colorTex, gl_FragCoord);  
    vec3 specular = texelFetch(specularTex, gl_FragCoord);  
    vec3 normal = texelFetch(normalTex, gl_FragCoord);  
    vec3 position = fetchPosition(gl_FragCoord);  
  
    vec3 shading = accumulate for each light in tile-list;  
  
    resultColor = vec4(shading, 1.0);  
}
```

CHALMERS

SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY

Summary: Tiled Shading



THE GOOD

- ▶ Low bandwidth
- ▶ Simple light assignment
- ▶ High performance
- ▶ Flexible
 - ▶ Forward or Deferred
- ▶ Transparency
- ▶ MSAA

THE BAD

- ▶ Complex light shader
- ▶ No Shadow map reuse
- ▶ 2D light assignment
 - ▶ Depth discontinuities



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- The good things then is that we have solved the bandwidth problem

Summary: Tiled Shading



THE GOOD

- ▶ Low bandwidth
- ▶ Simple light assignment
- ▶ High performance
- ▶ Flexible
 - ▶ Forward or Deferred
- ▶ Transparency
- ▶ MSAA

THE BAD

- ▶ Complex light shader
- ▶ No Shadow map reuse
- ▶ 2D light assignment
 - ▶ Depth discontinuities



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- Light assignment is still quite simple, just a 2D operation

Summary: Tiled Shading



THE GOOD

- ▶ Low bandwidth
- ▶ Simple light assignment
- ▶ High performance
- ▶ Flexible
 - ▶ Forward or Deferred
- ▶ Transparency
- ▶ MSAA

THE BAD

- ▶ Complex light shader
- ▶ No Shadow map reuse
- ▶ 2D light assignment
 - ▶ Depth discontinuities



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- Performance can be very high

Summary: Tiled Shading



THE GOOD

- ▶ Low bandwidth
- ▶ Simple light assignment
- ▶ High performance
- ▶ Flexible
 - ▶ Forward or Deferred
- ▶ Transparency
- ▶ MSAA

THE BAD

- ▶ Complex light shader
- ▶ No Shadow map reuse
- ▶ 2D light assignment
 - ▶ Depth discontinuities



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- The technique is also very flexible, which means that it is very easy to switch between deferred and forward shading,
- Or indeed combine, using deferred for opaque and forward for transparent.
- MSAA is thus trivial if forward shading is used.

Summary: Tiled Shading



THE GOOD

- ▶ Low bandwidth
- ▶ Simple light assignment
- ▶ High performance
- ▶ Flexible
 - ▶ Forward or Deferred
- ▶ Transparency
- ▶ MSAA

THE BAD

- ▶ Complex light shader
- ▶ No Shadow map reuse
- ▶ 2D light assignment
 - ▶ Depth discontinuities



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- On the down side, we do get back the issue of having all light types in one shader
- Although this is less of an issue on modern hardware it does impact register use

Summary: Tiled Shading



THE GOOD

- ▶ Low bandwidth
- ▶ Simple light assignment
- ▶ High performance
- ▶ Flexible
 - ▶ Forward or Deferred
- ▶ Transparency
- ▶ MSAA

THE BAD

- ▶ Complex light shader
- ▶ No Shadow map reuse
- ▶ 2D light assignment
 - ▶ Depth discontinuities



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- We are also now randomly accessing shadow maps, and cannot reuse storage.

Summary: Tiled Shading



THE GOOD

- ▶ Low bandwidth
- ▶ Simple light assignment
- ▶ High performance
- ▶ Flexible
 - ▶ Forward or Deferred
- ▶ Transparency
- ▶ MSAA

THE BAD

- ▶ Complex light shader
- ▶ No Shadow map reuse
- ▶ View dependence
 - ▶ 2D light assignment
 - ▶ Depth discontinuities



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- Finally, the performance of the technique is strongly view dependent
- Which means run time performance is not readily predictable from scene parameters.
- We'll look into this problem in some detail next...

Tiled Shading Problems



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- Here's that view again...

Tiled Shading Problems



SA2014.SIGGRAPH.ORG

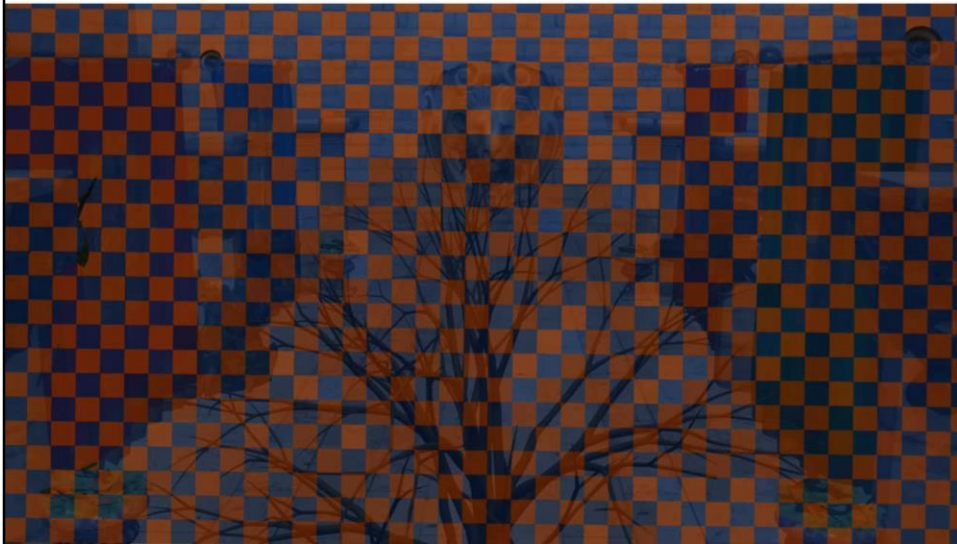
Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- And here are the tiles, pulled from our implementation...

Tiled Shading



SA2014.SIGGRAPH.ORG

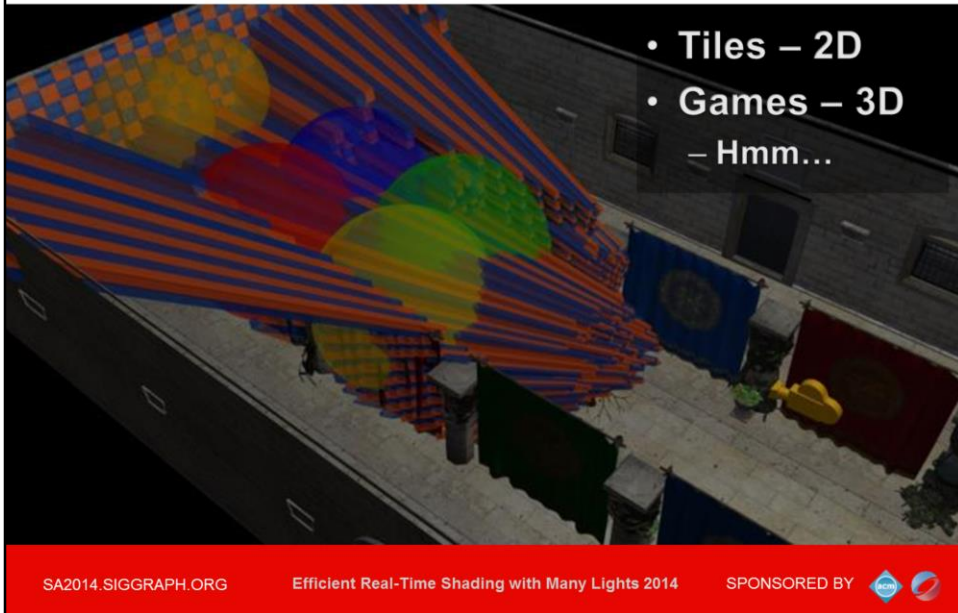
Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- Switching to the overhead view, we see how they extend from the tree over to the background geometry.

Tiled Shading Problems

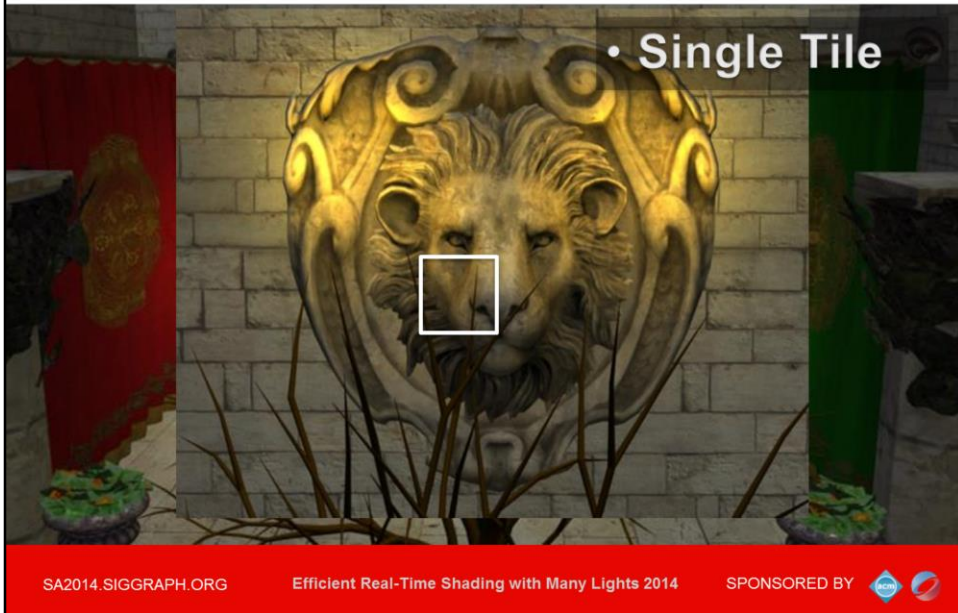


- Toggling on the light geometry, we see that there is a lot of overlap...
- ...even in the empty space behind the tree.
- We now should be able to start seeing the shape of the problem with 2D tiles in a 3D world.



We'll now take a closer look at a single tile, in order to highlight the problem.





- As we can see, there is a small number of samples from the tree
- And the rest, the lion share of the pixels are in the background.

Tiled Shading - The Discontinuity Dysfunction



CHALMERS

SA2014.SIGGRAPH.ORG

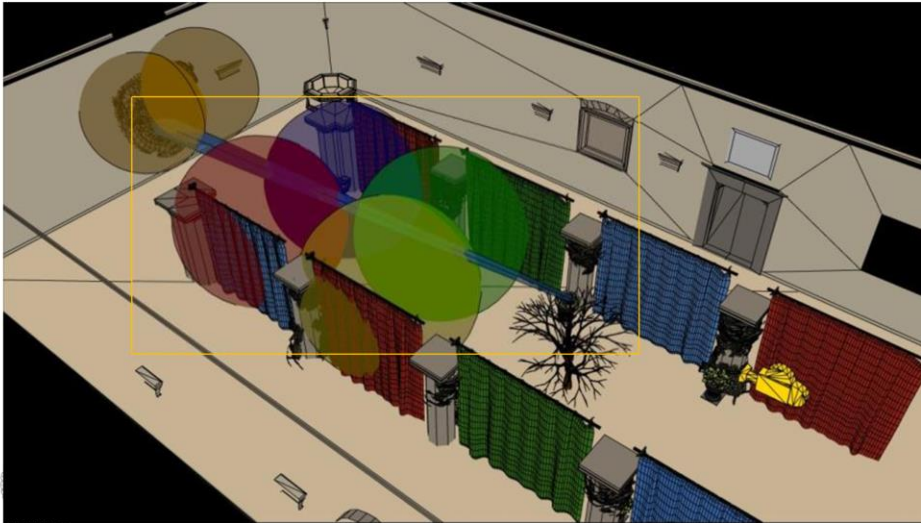
Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



* In 3D the tile looks like this...

Tiled Shading - The Discontinuity Dysfunction



CHALMERS

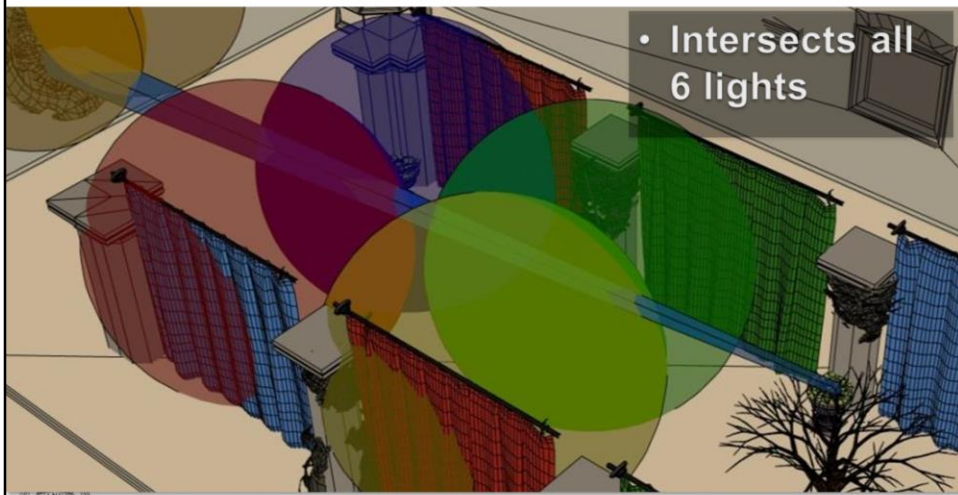
SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



Tiled Shading - The Discontinuity Dysfunction



SA2014.SIGGRAPH.ORG

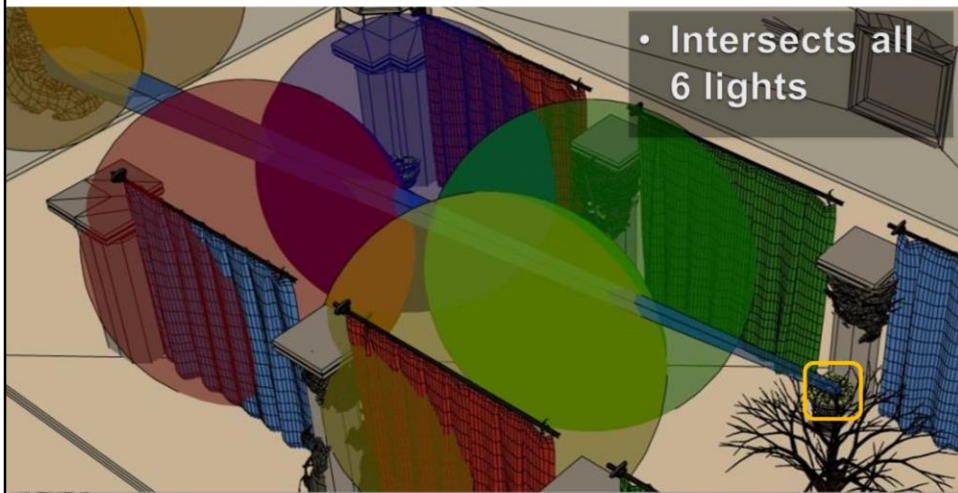
Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- The tile extends from the tree through the visible scene to the wall.
- This one troublesome tile intersects all 6 lights in our simple test scene.

Tiled Shading - The Discontinuity Dysfunction



SA2014.SIGGRAPH.ORG

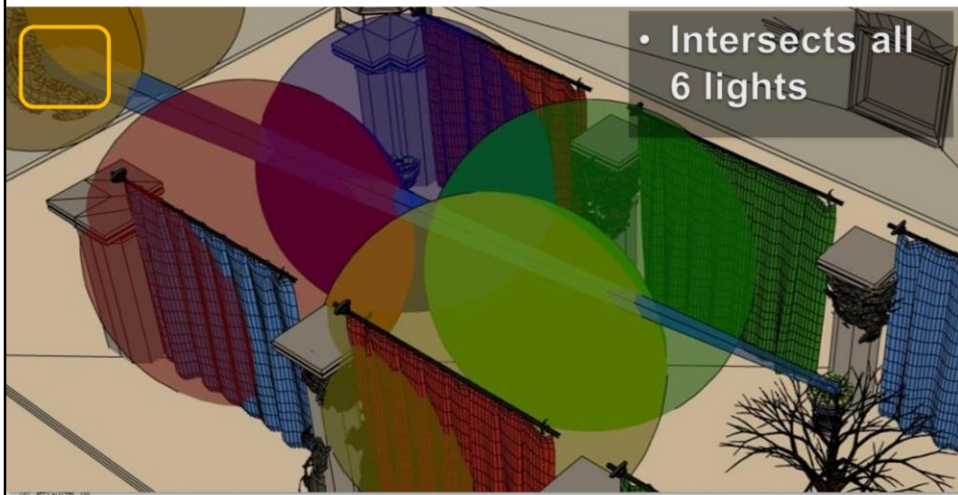
Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- While actually some of the samples, from the tree, are affected zero of these lights.

Tiled Shading - The Discontinuity Dysfunction



SA2014.SIGGRAPH.ORG

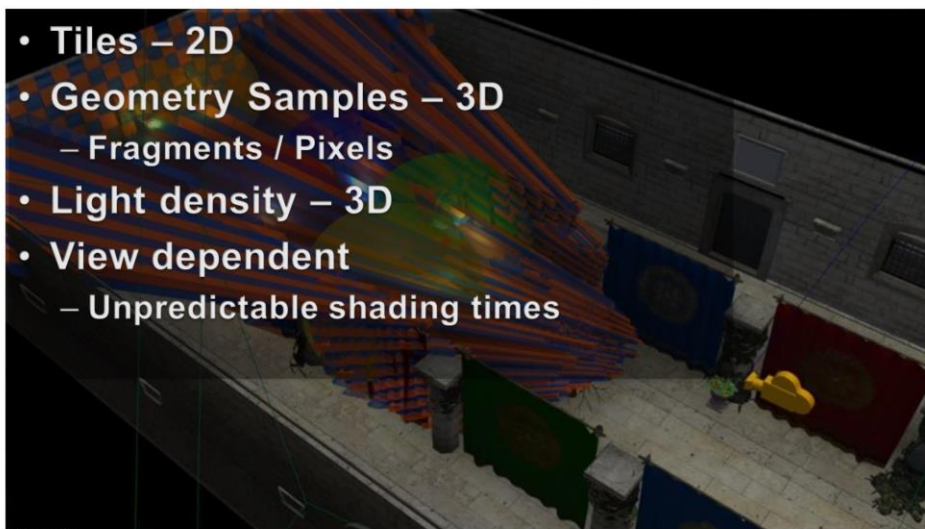
Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- And the rest, would only need two of the lights.

Tiled Shading



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY

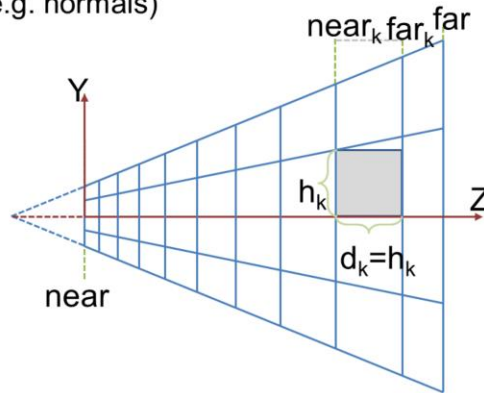


- So as we have shown, there is a fairly fundamental problem with tiled shading.
- The basic problem stems from that we are making the intersection between lights and geometry samples, both of which are 3D entities, in a 2D screen space.
- The main practical issue with this is that the resulting light assignment is highly view dependent. This means that we cannot author scenes with any strong guarantee on performance, as a given view of the scene may have a significantly higher *screen space* light density than average.
- For example, we'd like to be able to construct a scene with, say, maximum 4 lights affecting any part of the scene. In this case, we would like shading cost to be proportional to this, and stable, given different view points. Unfortunately, no such correlation exists for tiled shading.
- In other words shading times are unpredictable, which is a major problem for a real time application.

Clustered Shading – Key Idea



- ▶ Add the 3rd dimension
 - ▶ Tile also in depth direction = cluster
 - ▶ Also > 3 dimensions (e.g. normals)



SA2014.SIGGRAPH.ORG

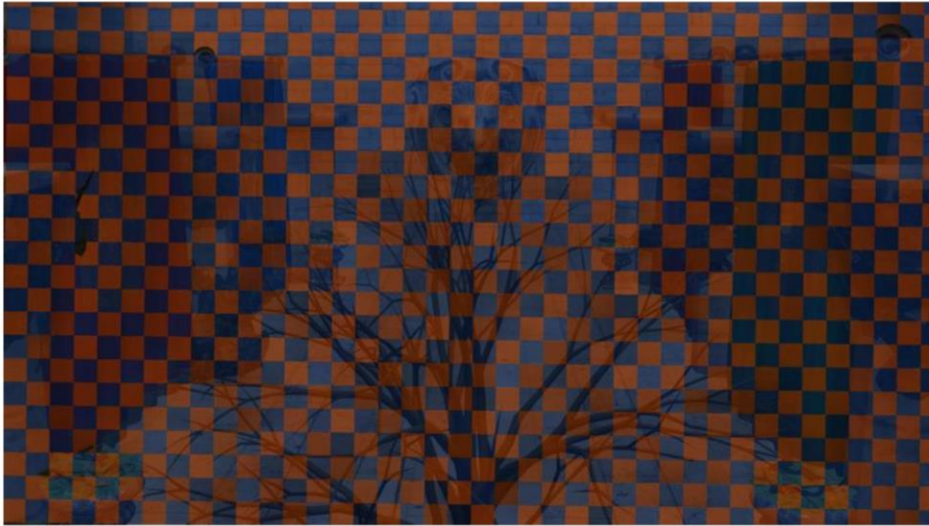
Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- So how do we solve this?
- This is the basic question for our clustered shading paper.
- A fairly obvious solution, given what has been said so far, is to use 3D tiles of some sort.
- It is less obvious what particular kind of subdivisions to use, and whether this will actually improve efficiency.
- Another question we explore in the paper is to tile in yet higher dimensions, based on normal direction as well.

Clustered Shading: The Solution



SA2014.SIGGRAPH.ORG

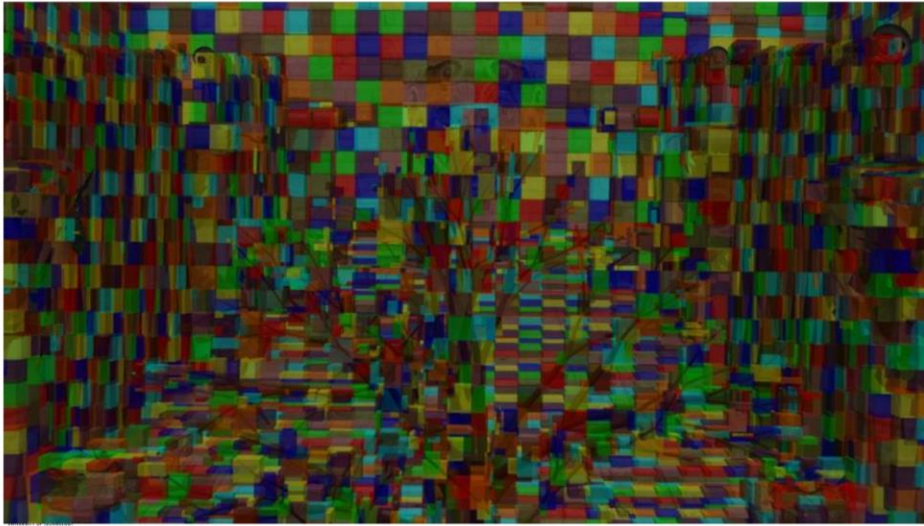
Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



Compare the tiles shown here...

Clustered Shading: The Solution



SA2014.SIGGRAPH.ORG

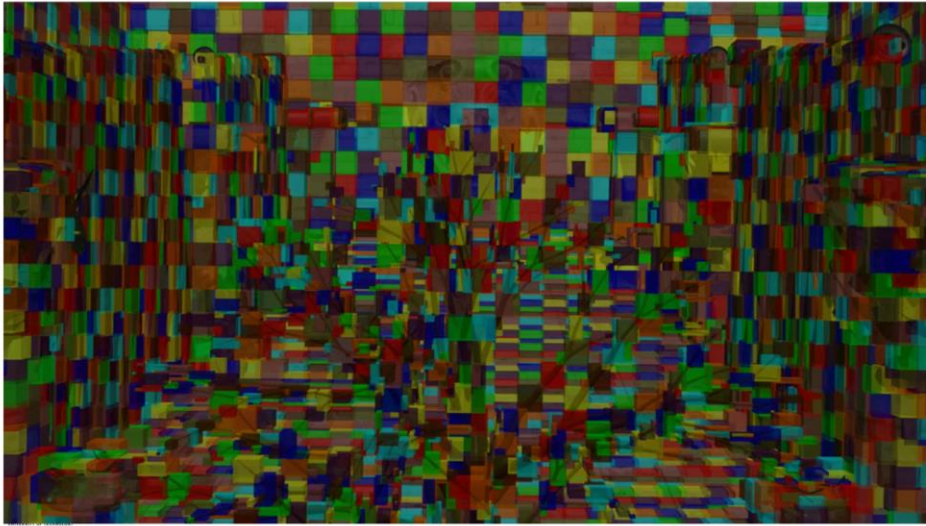
Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- ...with the view space cluster AABBs shown here.
- Along the top edge of the screen, it is easy to see that the clusters and tiles are very similar, where the depth within each tile is shallow.
- In the middle, where the tree is, things are more interesting, as several layers of depth are visible.

Clustered Shading: The Solution



SA2014.SIGGRAPH.ORG

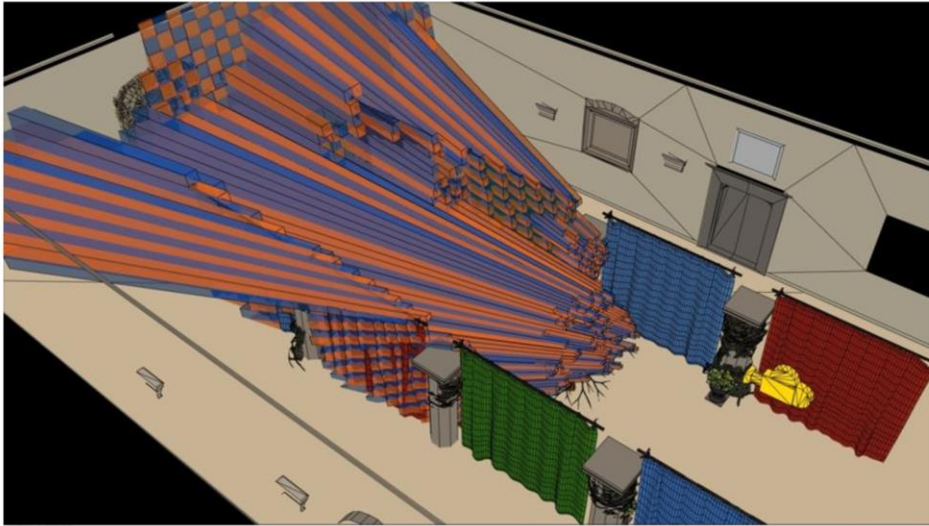
Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



Going to the overhead view again.

Clustered Shading: The Solution



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



Comparing the tiles...

Clustered Shading: The Solution



SA2014.SIGGRAPH.ORG

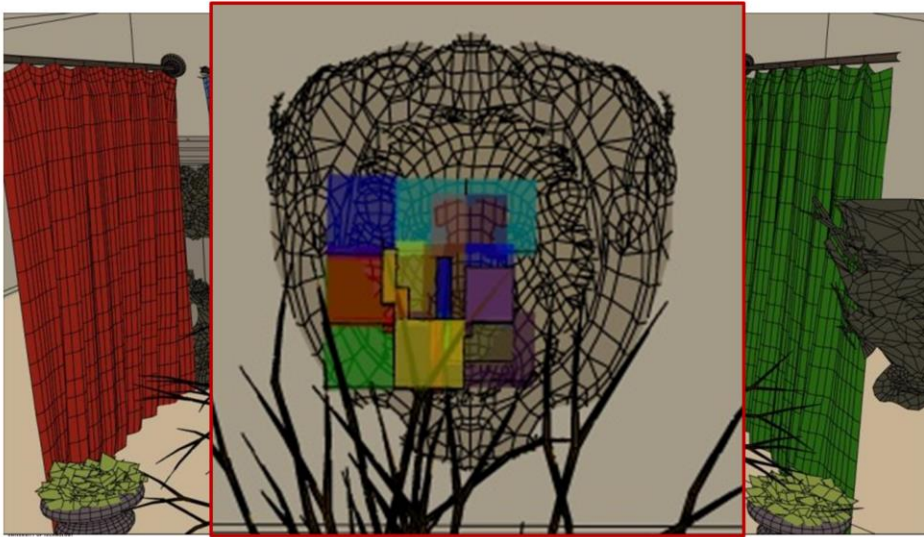
Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- ...to the clusters, shows that the clusters approximate the visible geometry a lot better.
- This means that the intersection with the light volumes ought to also be more precise.

Clustered Shading: The Solution



SA2014.SIGGRAPH.ORG

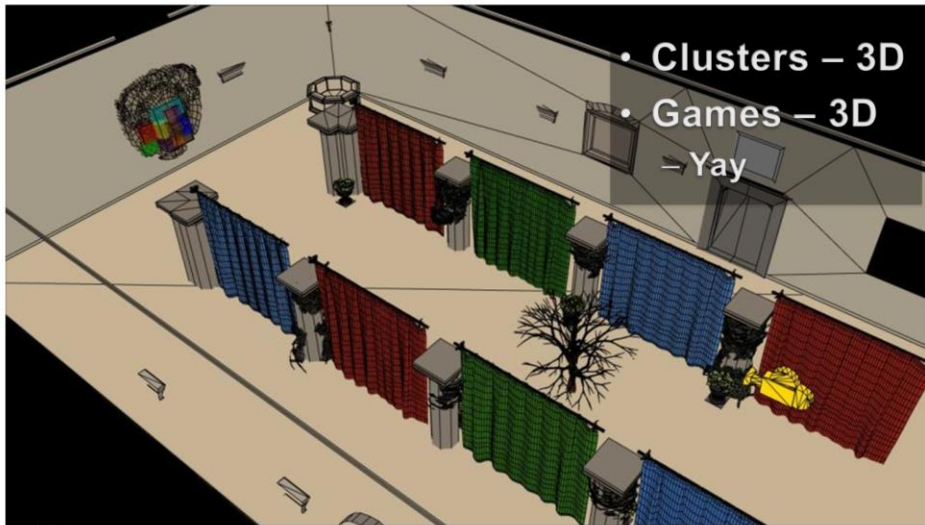
Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- Going back to the single tile example, but with clusters.
- We see that now there is a little flock of them instead of just one tile.

Clustered Shading: The Solution



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



* From the overhead view again...

Clustered Shading: The Solution



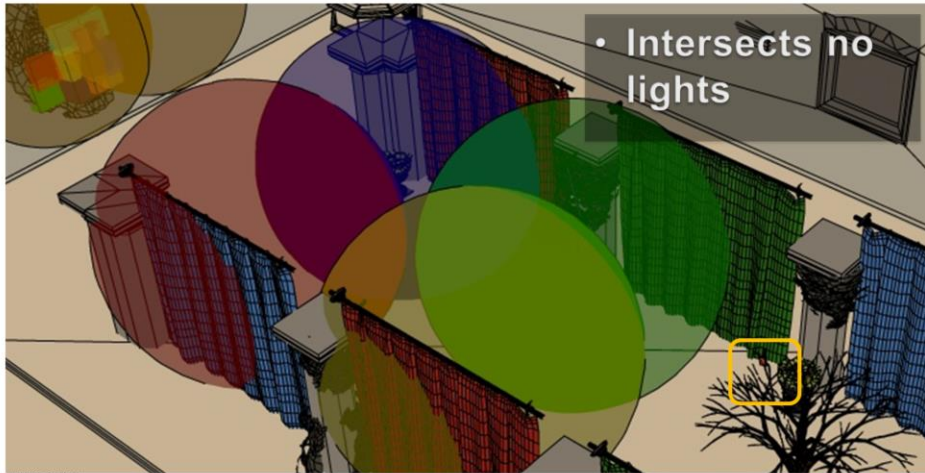
SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



Clustered Shading: The Solution



SA2014.SIGGRAPH.ORG

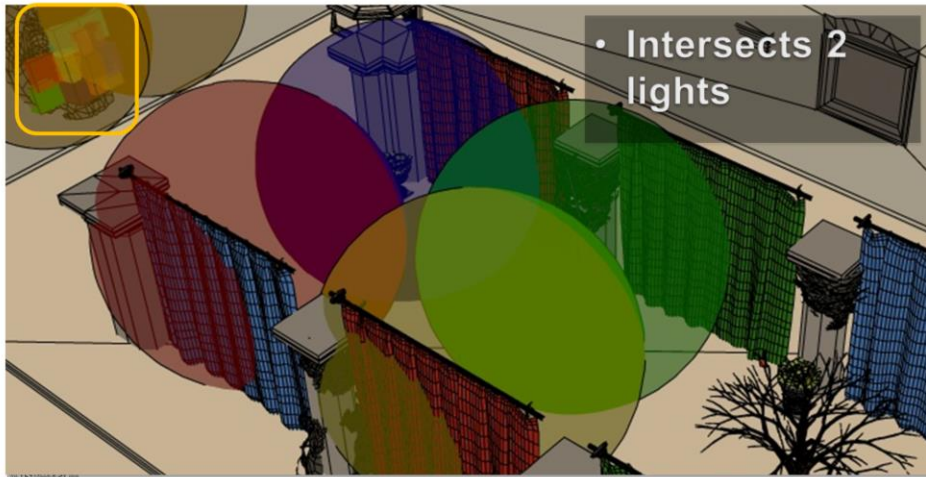
Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



We see that none of the lights overlap the clusters that are on the tree in the foreground.

Clustered Shading: The Solution



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- And in the background, only the volumes of the two required lights intersect.
- Clearly clusters has a good potential for improving efficiency,
- ...we'll now need to talk about how they can be implemented.
- And whether this translates to improved performance.

Clustered Shading – Idea



- ▶ Add the 3rd dimension
 - ▶ Tile also in depth direction = cluster
 - ▶ Also > 3 dimensions (e.g. normals)
- ▶ Bounded volume around samples
 - ▶ Shading cost ~ Light density.
- ▶ New Challenges
 - ▶ Many more (potential) clusters
 - ▶ Must find those actually used
 - ▶ Adding lights no longer screen space



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



Clustered Shading – Algorithm



- ▶ Rasterize G-Buffers
 - ▶ Forward: pre-z pass
- ▶ Cluster assignment
- ▶ Find unique clusters
- ▶ Assign lights to clusters
- ▶ Shade view samples
 - ▶ Deferred: Full screen pass
 - ▶ Forward: Geometry pass



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY

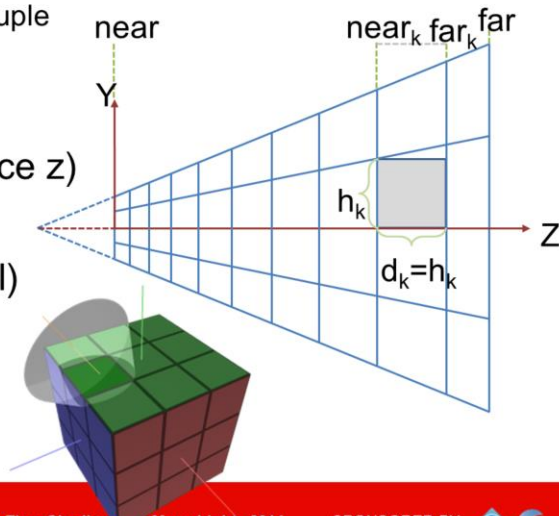


- This is a high level version of the clustered shading algorithm
- I've grayed out parts that are identical to tiled shading.
- Cluster assignment means to identify what cluster each sample belongs to, this is a simple mapping.
- Then we need to work out which of the clusters are represented by these samples, and
- Finally we assign lights to these clusters, ensuring we are not wasting storage and work assigning lights to empty clusters.
- Note that Emil will be presenting a rather different approach to implementing the algorithm, which leaves out and replaces the first two steps.

Cluster Assignment



- ▶ Cluster key:
 - ▶ $ck = (i, j, k)$ integer tuple
 - ▶ i, j = 2D tile id
 - ▶ `gl_FragCoord.xy`
 - ▶ $k = \approx \log(\text{view space } z)$
- ▶ (quantized normal)



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading in Many Lights 2014

SPONSORED BY

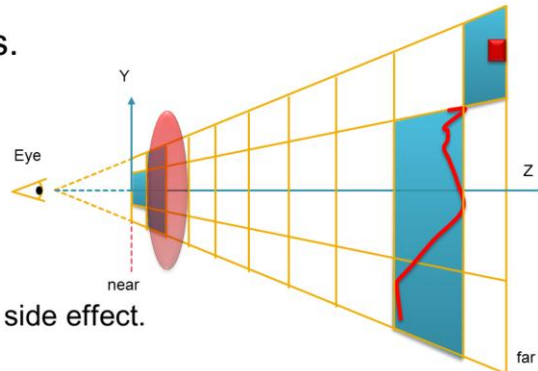


- Cluster assignments is a simple mapping from sample coordinate, to an integer tuple i, j, k
- i and j are simply the tile coordinates, which can be derived by dividing `gl_FragCoord.xy` by the tile size.
- k is a logarithmic function of the view space Z of the sample, not simply the logarithm, for the exact equation see the paper.
- We use this subdivision as it creates self similar clusters that are as cube like as possible. This makes them better suited for culling. The logarithmic subdivisions also means that as clusters become larger further away, we get a kind of LOD behaviour and do not end up with insane numbers of clusters, for a wide range of view parameters.

Finding Unique Clusters



- ▶ Full screen pass.
- ▶ Flag used clusters.
 - ▶ Read depth
 - ▶ Compute ck
 - ▶ Set cell in grid to one.
- ▶ Forward:
 - ▶ Geometry pass with side effect.



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY

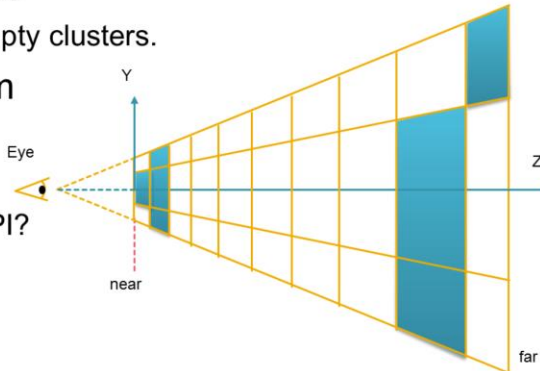


- Finding the unique clusters is a full screen pass, which simply constructs the cluster key for each sample.
- And sets the corresponding cell in a 3D grid to 1. This grid is non-regular subdivision of the view volume.

Finding Unique Clusters



- ▶ **Compact Non-Zero**
 - ▶ Yields list of non-empty clusters.
- ▶ **Parallel prefix sum**
 - ▶ chag::pp (CUDA)
 - ▶ thrust (CUDA)
 - ▶ Compute Shader API?



1 2 4 5 6 7 32 33 34 39

SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- We then compact the grid into the list of non-zero elements.
- This can be implemented through a parallel prefix sum.
- Which is a very quick process, for the million elements or so needed.
 - Taking a fraction of a millisecond.
- This leaves us with a list of clusters which needs lights assigned to them.

Light Assignment



- ▶ Many clusters
- ▶ Many lights
- ▶ Hierarchical approach
 - ▶ Hierarchy over lights
 - ▶ 32-way tree (matches SIMD of GPU)
 - ▶ Dynamically rebuilt on GPU
 - ▶ Each cluster traverses light tree
 - ▶ BV tests



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- When pushing the limit on number of lights, a hierarchy over lights makes sense.
- We use a BVH with a branching factor of 32, which is rebuilt each frame.
- When not so many lights are used, there are many other approaches which may be better. Again Emil will show a rather different approach later on.

Full-Screen Clustered Shading Pass



Light Grid:

	L ₀	L ₁	L ₂	L ₃	L ₄	L ₅	L ₆	L ₇	L ₈	L ₉	...
	0	0	6	3	0	6	4	4	...		

offset: 2, 2

size: 2, 2

```

out vec4 resultColor;

void main()
{
    vec3 color = texelFetch(colorTex, gl_FragCoord);
    vec3 specular = texelFetch(specularTex, gl_FragCoord);
    vec3 normal = texelFetch(normalTex, gl_FragCoord);
    vec3 position = fetchPosition(gl_FragCoord);

    vec3 shading = accumulate for each light in cluster;

    resultColor = vec4(shading, 1.0);
}
    
```

- The shading pass then is virtually identical to tiled shading.
- The only difference is how to calculate the index into the light grid.
- So we clearly have the same benefits in memory bandwidth.



Transparency

SA2014.SIGGRAPH.ORG

SPONSORED BY



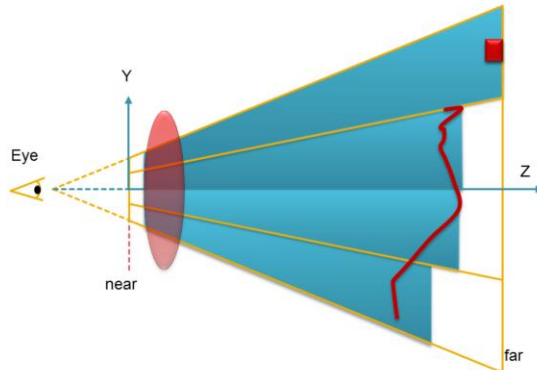
Transparent Geometry



► Tiled Forward Shading [4]

► Extend tile Z bounds

- To Min Z
- Or Near plane



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- It's simple to support transparent geometry with tiled forward shading.
 - For example by finding the min and max Z values, or just extending tiles to the near plane.
 - However...

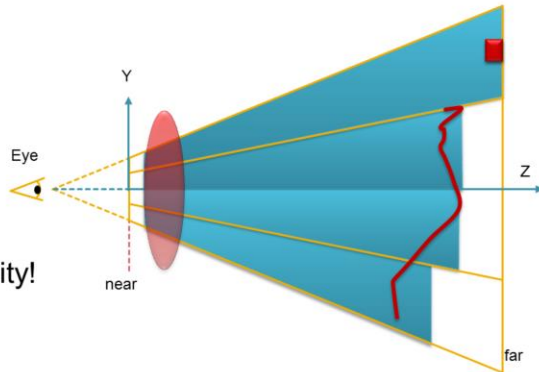
Transparent Geometry



► Tiled Forward Shading [4]

► Problems

- View dependent
- Degenerates to 2D
- Full screen discontinuity!



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY

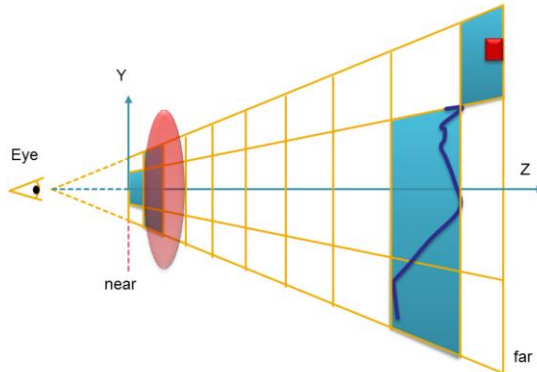


- The result of transparent geometry covering the view is effectively the loss of the depth range optimization
 - which can be a very bad thing™.
- Again, the biggest issue here is that it is view dependent, and so will only be possible to determine at run time
 - and, as we all know, this kind of problem usually starts showing up five minutes after we shipped a build to the publisher.

Clustered Forward Shading [4]



- ▶ Pre-geometry pass.
- ▶ Flag used clusters.
 - ▶ Side effect in fragment shader.
 - ▶ Set cell in grid to one.



SA2014.SIGGRAPH.ORG

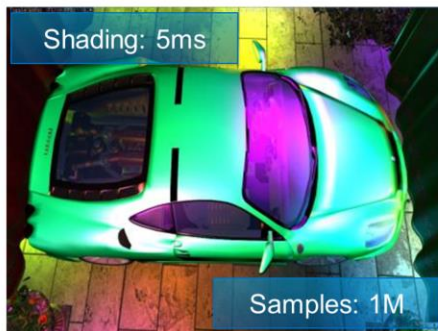
Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- Clustered Shading does not suffer from this problem, as each cluster represents a fixed section of 3D space.
- The only problem we face is trying to determine what clusters are needed such that we can assign lights to them.
- This can be done by rendering a pre-pass with the transparent geometry.
- This can be performed after a regular G-buffer or pre-z pass, with color and depth writes turned off, and sets the used clusters to one as a side effect in the fragment shader.
- The algorithm is otherwise as before.
- Note that clustered shading provides an efficient way to solve the light assignment problem for any transparency technique, not just good old order dependent transparency.

Tiled Forward Shading



► Ok



- Depth ranges fairly short
- Overdraw fairly limited

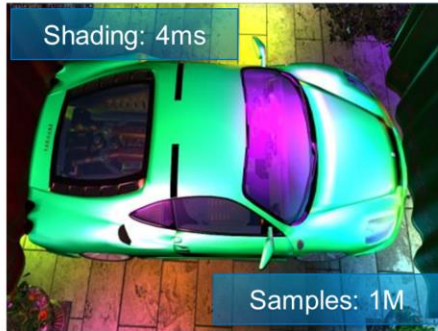


► Disaster

- Depth ranges long
- Extra screen of overdraw

- Here's an example where the view dependence is quite evident,
- Going from an overhead view of the car, to looking through the window.
- We see that the cost for tiled forward increases significantly.

Clustered Forward Shading



► Ok



► Also ok

► Extra screen of overdraw



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



- The performance for the top down view is pretty similar.
- But for clustered shading the increase is only about 50%
- Despite shading 1.8 times the number of samples .

Clustered Shading - Summary



- ▶ High performance
- ▶ Low view dependence
- ▶ Good worst-case performance
- ▶ Fully Dynamic
- ▶ Supports transparency
- ▶ Forward / Deferred, or both
- ▶ No Pre-Z required



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



Technique Comparison



	Trad. Deferred	Tiled Deferred	Tiled Forward	Clustered Deferred	Clustered Forward
Bandwidth Use	High	Low	Low	Low	Low
Transparency	No	Maybe	Yes	Maybe	Yes
MSAA	Maybe	Maybe	Yes	Maybe	Yes
Shadow Map reuse	Yes	No	No	No	No
Geometry Passes	1	1	1-2 ⁺	1	1-2 ⁺
Register Pressure	Low	High	High	High	High
Innermost loop	Pixels	Lights	Lights	Lights	Lights
FB Precision	High	Low	Low	Low	Low
View dependence	Low	High	High	Low	Low
Vary Shading Models	Hard	Hard	Simple	Hard	Simple



SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY



References



- ▶ [1] Practical Clustered Deferred and Forward Shading, Persson & Olsson 2013, <http://advances.realtimerendering.com/s2013>
- ▶ [3] Clustered Deferred and Forward Shading, Olsson et.al. 2012
- ▶ [4] Tiled and Clustered Forward Shading, Olsson et.al. 2012
- ▶ [Valient14] Reflections and volumetrics of Killzone Shadow Fall , SIGGRAPH'14, <http://advances.realtimerendering.com/s2014>
- ▶ [Andersson14] Rendering Battlefield 4 with Mantle, GDC 2014
- ▶ [Shulz14] Moving to the Next Generation - The Rendering Technology of Ryse, GDC 2014.
- ▶ [Tebbs92] Brice Tebbs, Ulrich Neumann, John Eyles, Greg Turk, and David Ellsworth. Parallel architectures and algorithms for real-time synthesis of high quality images using deferred shading. 1992.
- ▶ [Saito90] Takafumi Saito and Tokiichiro Takahashi. Comprehensible rendering of 3-d shapes. SIGGRAPH Comput. Graph., 24(4):197–206, 1990.
- ▶ [Hargreaves04] Shawn Hargreaves and Mark Harris. Deferred shading. NVIDIA Developer Conference: 6800 Leagues Under the Sea, 2004.
- ▶ [Shishkovtsov05] Oles Shishkovtsov. Deferred shading in S.T.A.L.K.E.R. In GPU Gems 2. Addison-Wesley, 2005.
- ▶ [Arvo03] Jukka Arvo and Timo Aila. Optimized shadow mapping using the stencil buffer. journal of graphics, gpu, and game tools, 8(3):23–32, 2003.



Papers / Slides / Demo implementation with source: <http://www.cse.chalmers.se/~olaolss>

SA2014.SIGGRAPH.ORG

Efficient Real-Time Shading with Many Lights 2014

SPONSORED BY

