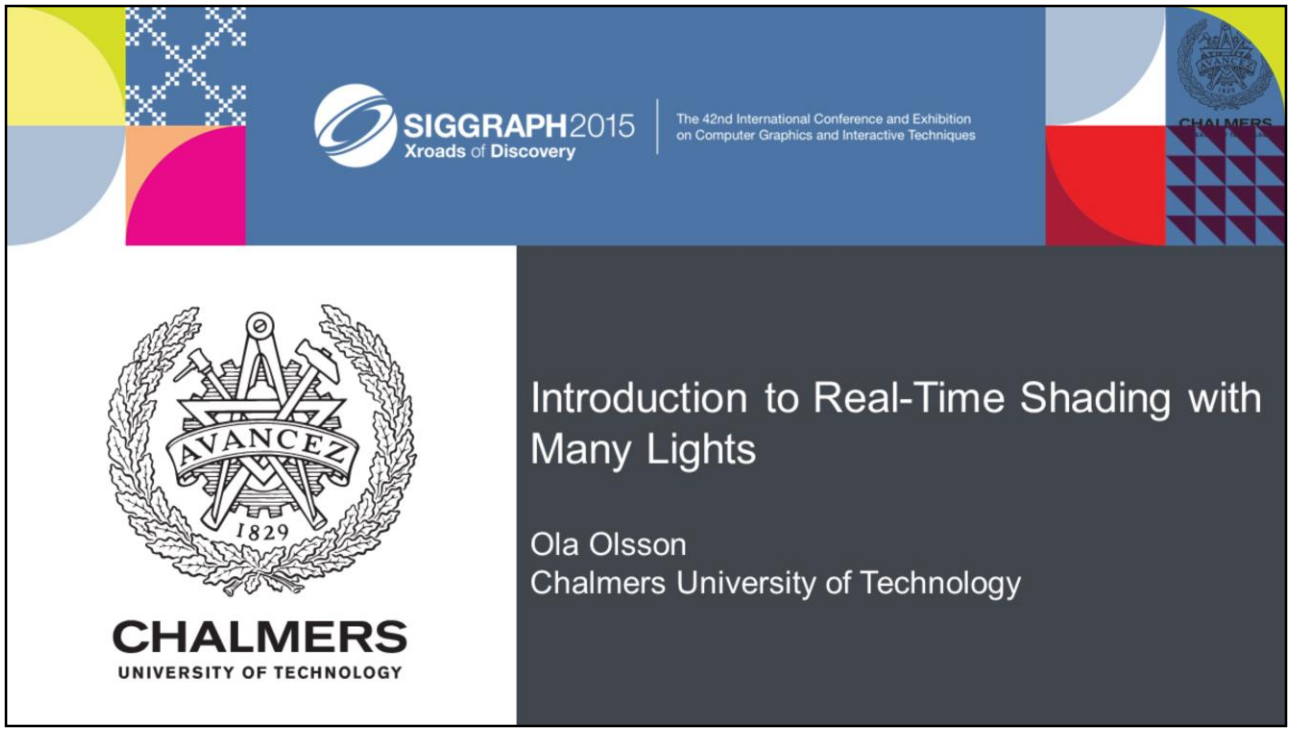
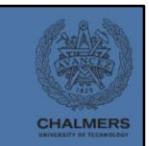

SIGGRAPH2015
Xroads of Discovery



- 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...
- …and is what the remaining talks in this course build on.

Talk Outline

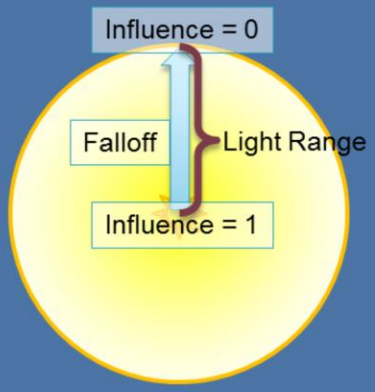


- Problem Definition
  - What, exactly, are you talking about!?!
- Part 1: 'Ancient' Shading Techniques
  - Forward Shading
  - Deferred Shading
- Part 2: Modern Shading Techniques
  - Tiled Deferred/Forward Shading
  - Clustered Deferred/Forward Shading

Real-Time Many-Light Management and Shadows with Clustered Shading – 2015

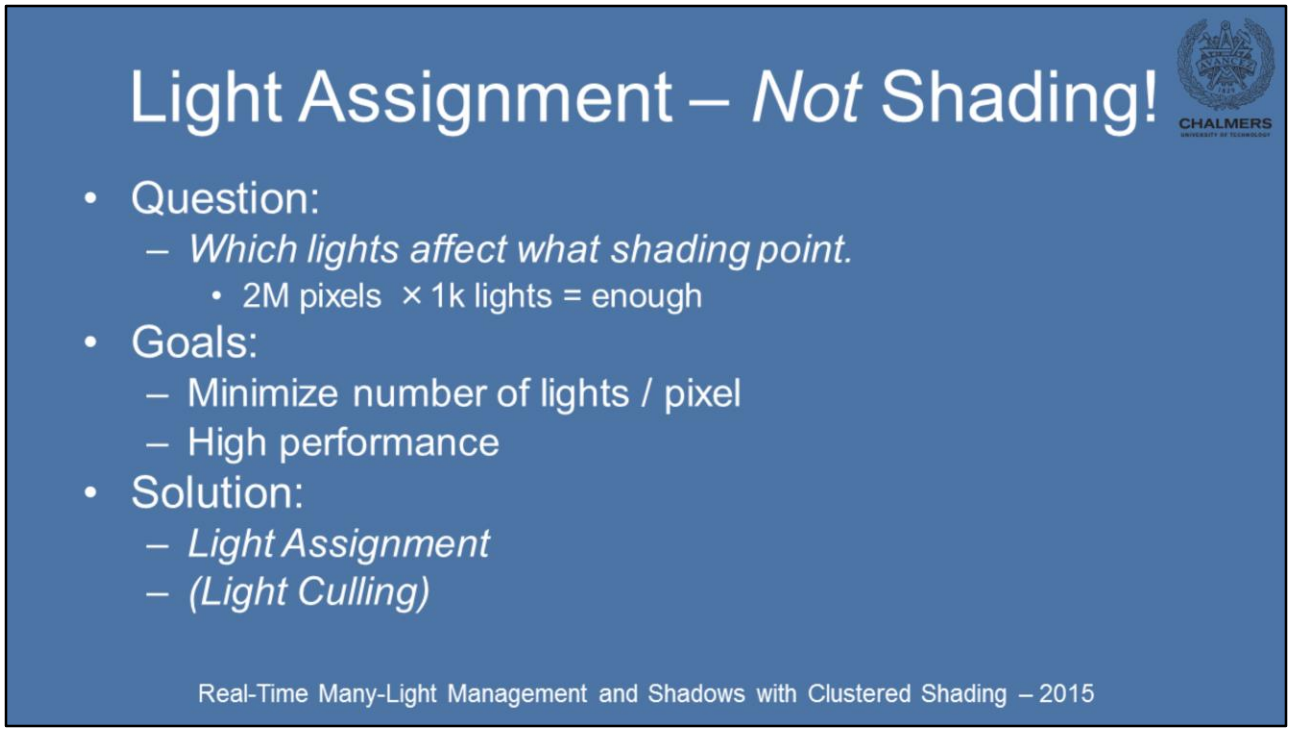
Problem Definition

- Real-time shading algorithms
  - Thousands of lights
  - Limited range light
  - Fully dynamic
    - Lights and Geometry



Real-Time Many-Light Management and Shadows with Clustered Shading – 2015

- The algorithms we are going to talk about are relevant when we target hundreds or thousands of lights in real time,
- And in a way you could say that the end goal is to make the number a non-issue. <click>
- The lights we consider have a limited range, with some falloff which goes to 0 at the boundary.
- This means that lights are not physical, but this is (still) the normal procedure in games. <click>
- 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.

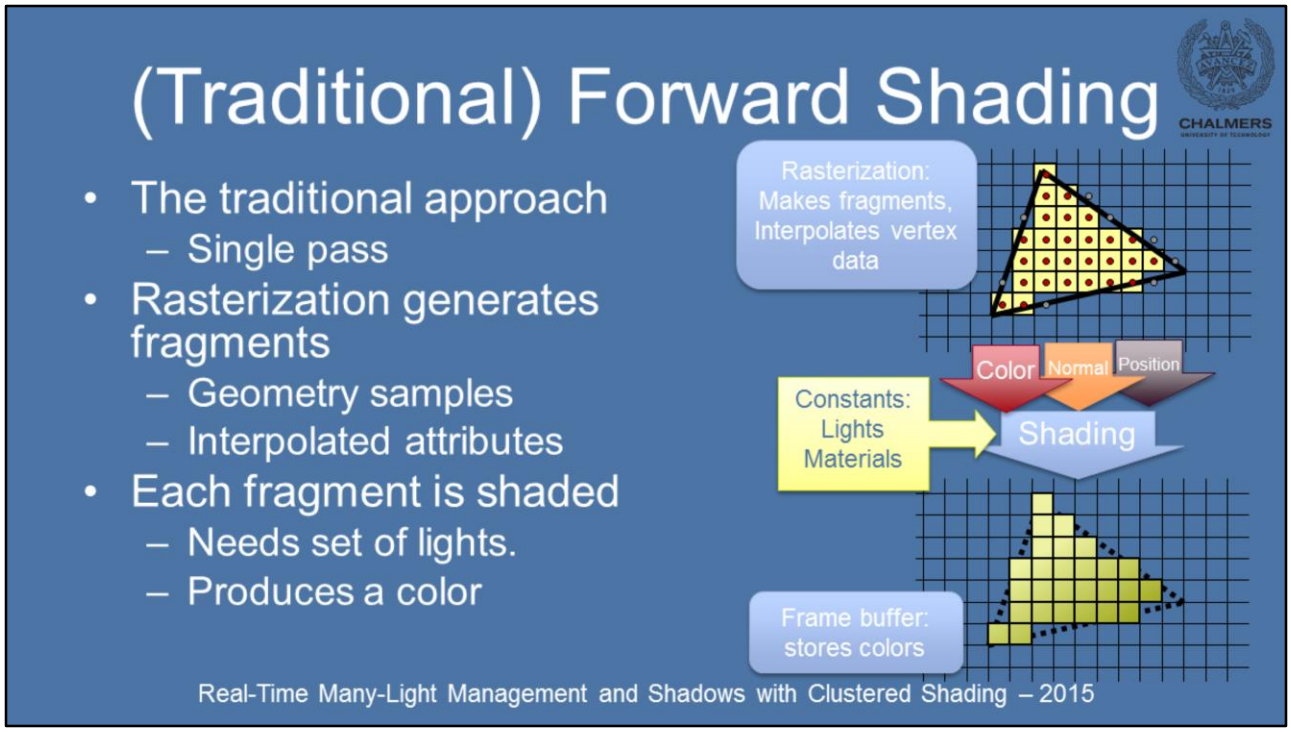


- Despite the name of all the techniques in this course, it is not actually about shading at all!
- This is a name we have inherited from forward shading / deferred shading etc.
- In reality the course is about working out which lights are relevant for shading each point of interest.
- With millions of sample points (pixels) and thousands of lights
- Doing this with high performance becomes sufficiently complex to be an interesting problem.
- We call this process ‘light assignment’, but is also known as ‘light culling’.

Part 1

TRADITIONAL FORWARD SHADING

Real-Time Many-Light Management and Shadows with Clustered Shading – 2015



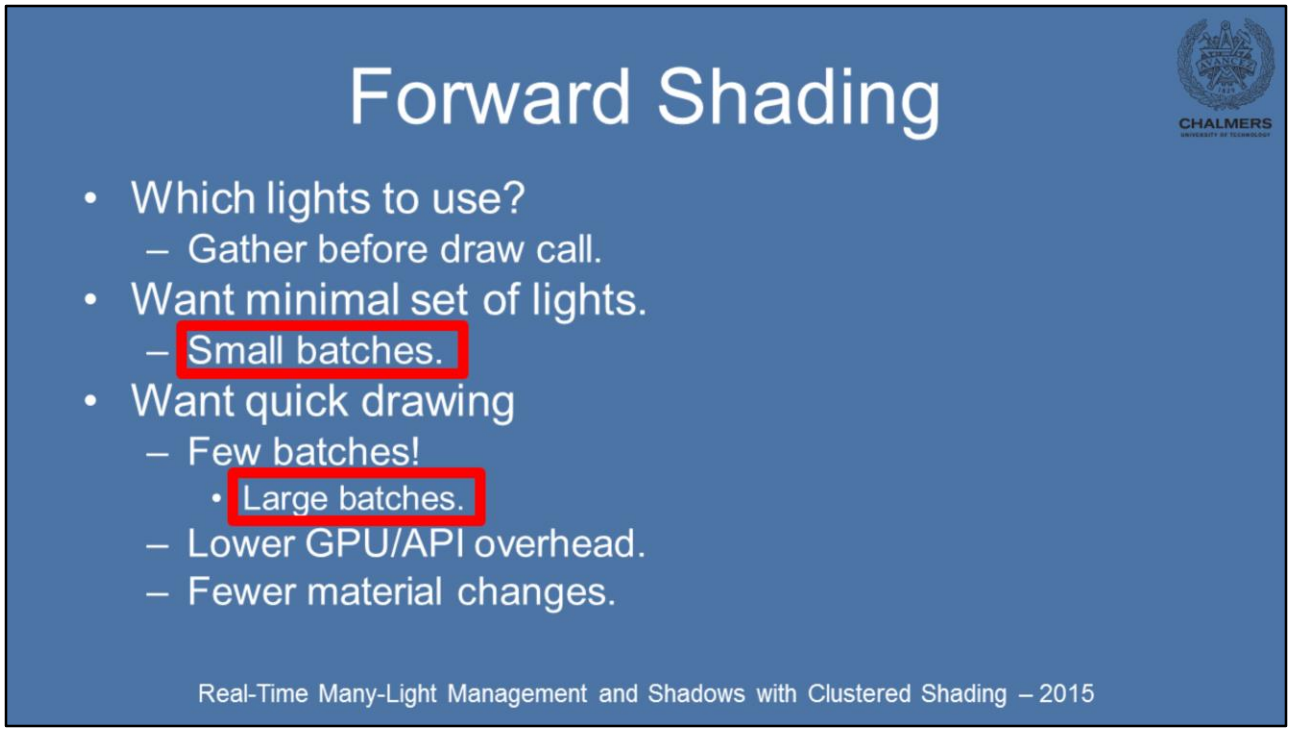
• The traditional method for real-time shading is called forward shading and used to be the only method during the first decade, or so, of consumer GPUs.

• 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.

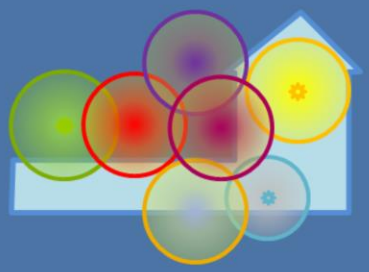
• Shading is performed in the fragment shaders. (Or indeed, in vertex shaders…)



- 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 Problems

- Few large objects
- Many small lights

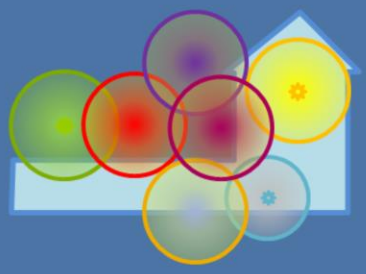


Real-Time Many-Light Management and Shadows with Clustered Shading – 2015

- 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 - Problems

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



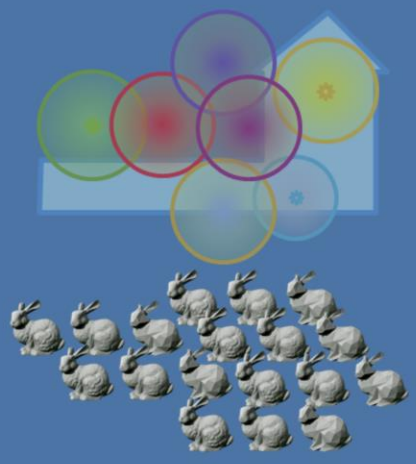
Real-Time Many-Light Management and Shadows with Clustered Shading – 2015

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

Forward Shading - Problems

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

- Many small objects



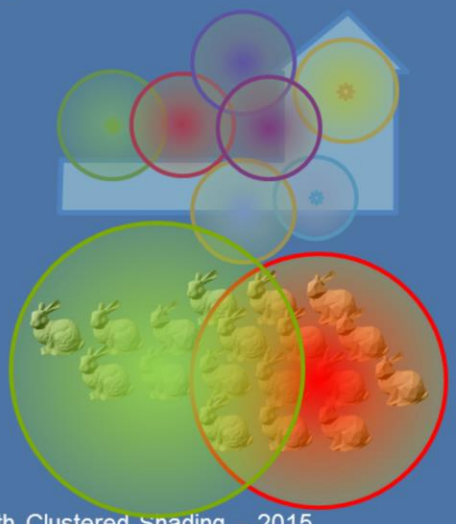
Real-Time Many-Light Management and Shadows with Clustered Shading – 2015

• 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



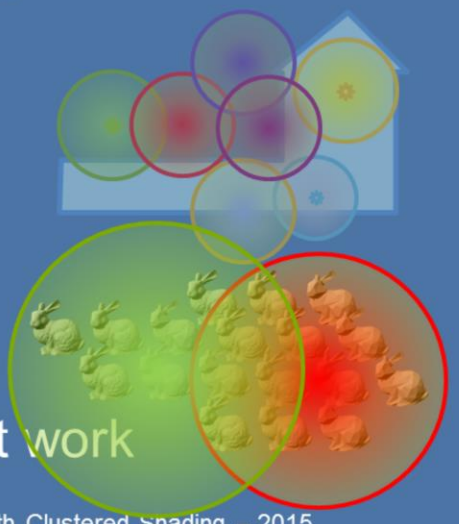
Real-Time Many-Light Management and Shadows with Clustered Shading – 2015

- 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



Real-Time Many-Light Management and Shadows with Clustered Shading – 2015

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

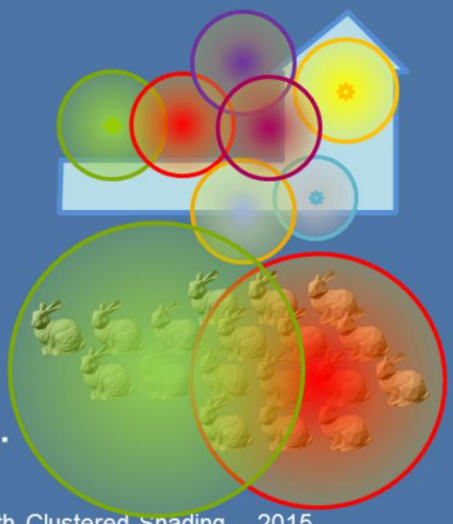
Forward Shading - Problem Examples

- Few large objects
- Many small lights

AND

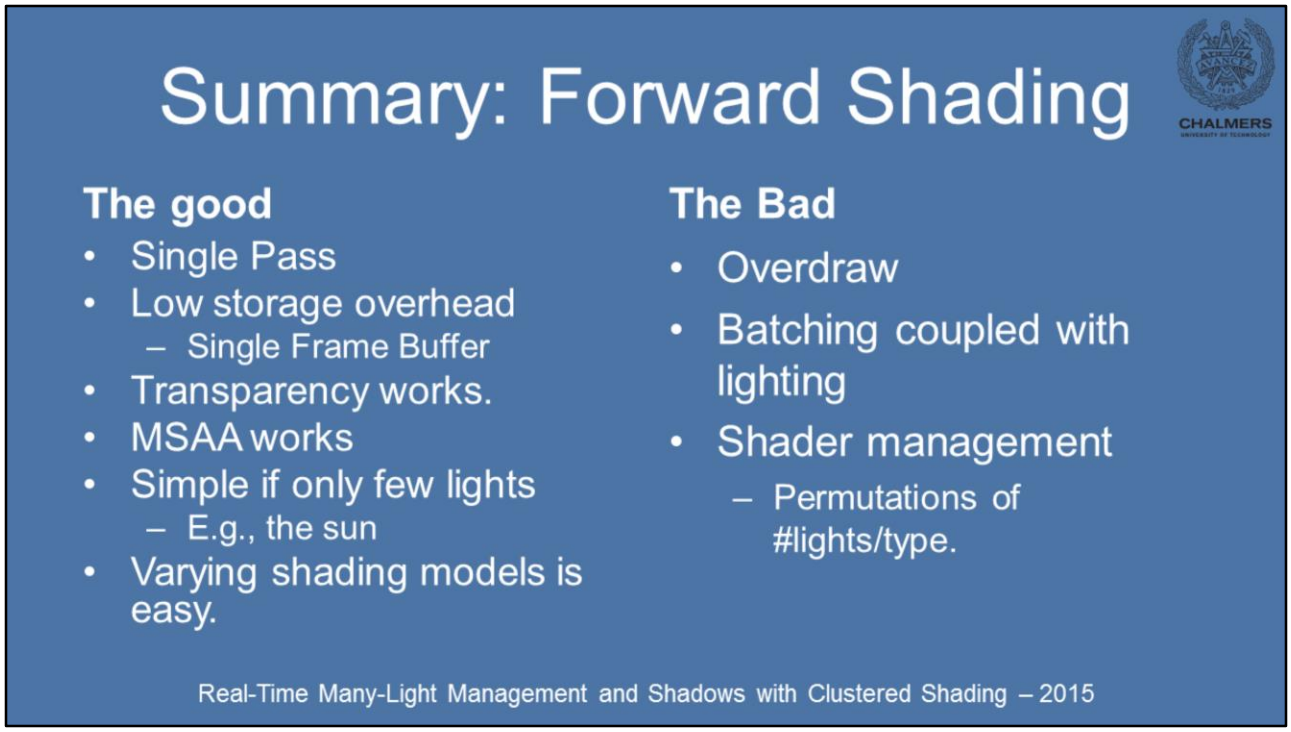
- Many small objects
- Few large lights

- Large triangles.
- Varying light density (dynamic).

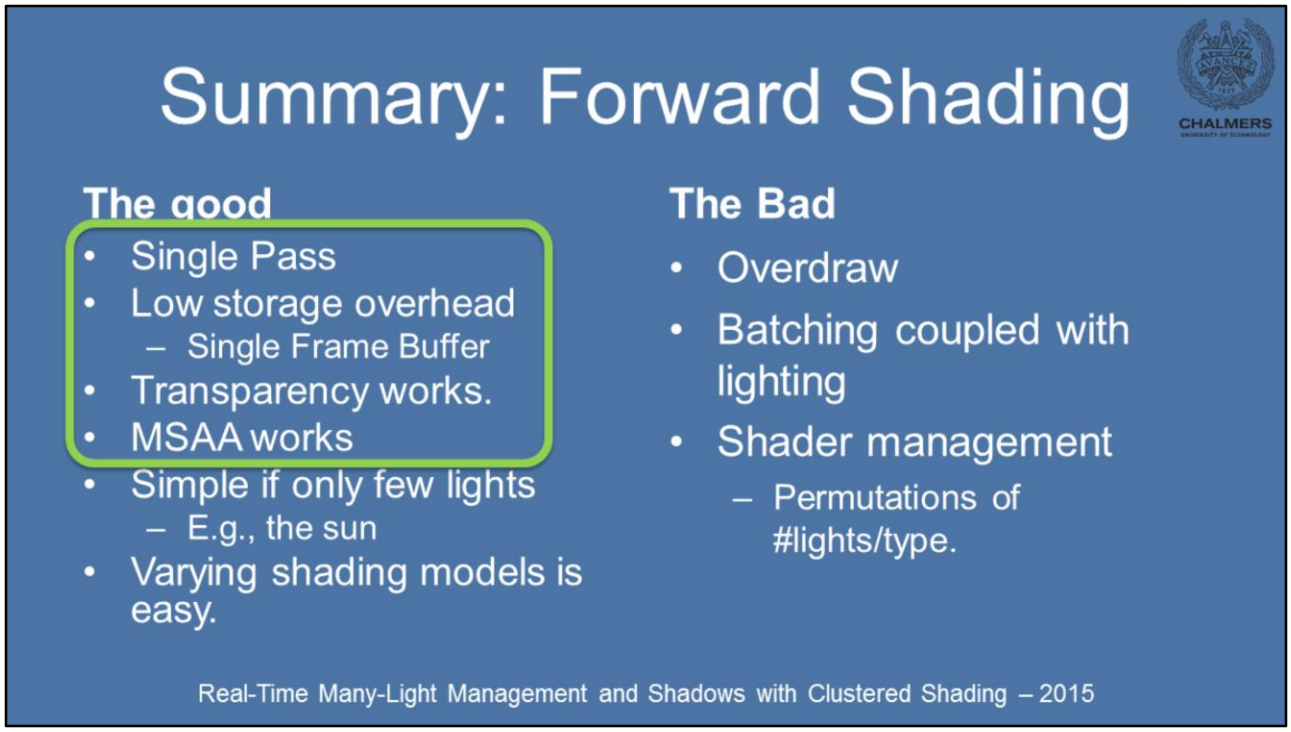


Real-Time Many-Light Management and Shadows with Clustered Shading – 2015

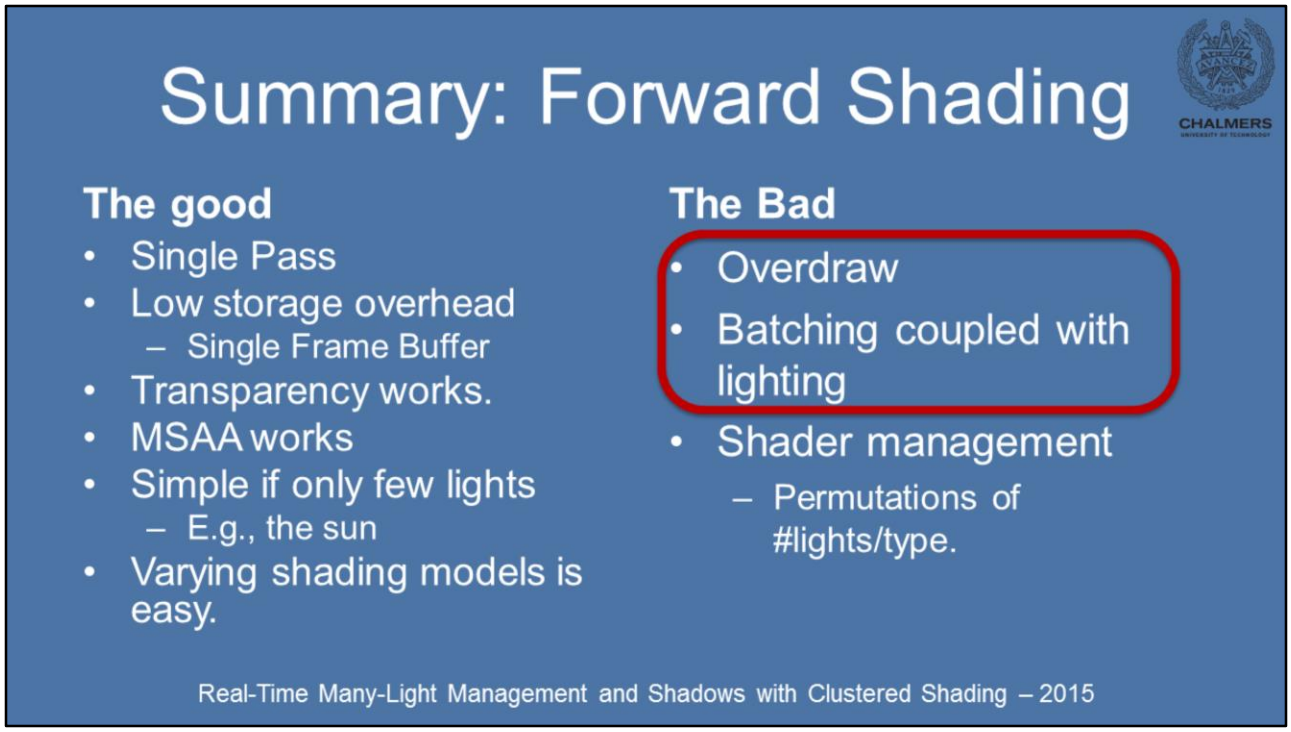
- So the conflict runs deeper than just batch size…<click>
- And of course, in the same scene we might have both houses and rabbits, which means it is difficult or impossible to ensure a good performance balance.
- Note that the large object could be a single triangle, and in general triangle size and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



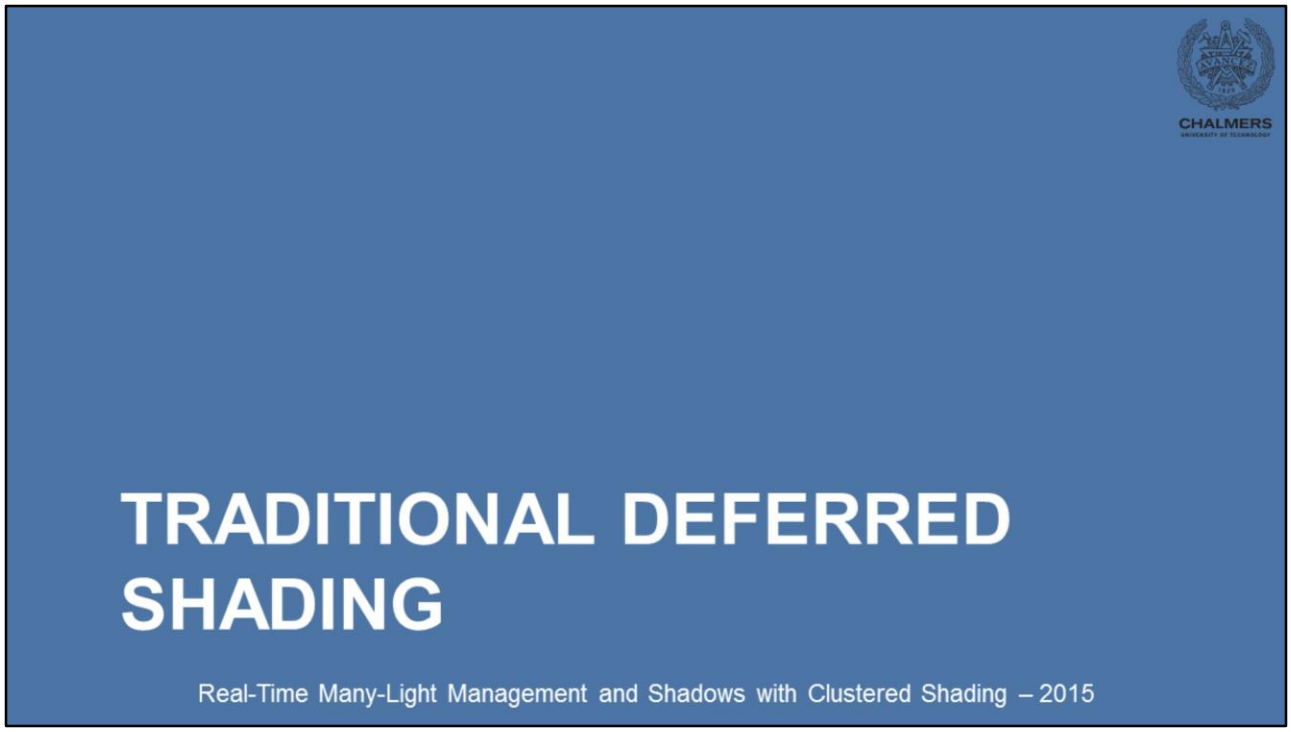
- To summarize, the main good thing about traditional forward shading is that

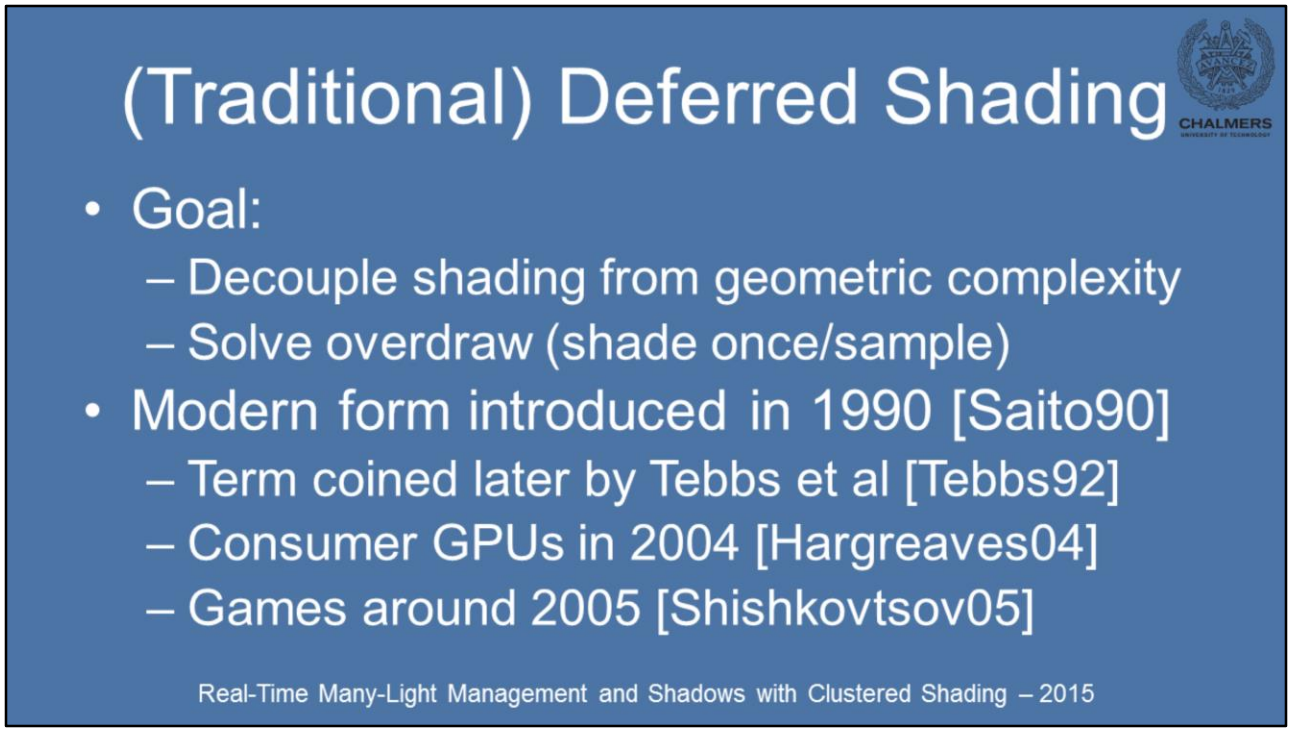


- Everything can be done in a single pass with a single frame buffer, just storing the final color
- This enables supporting transparency and MSAA without much ado.



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

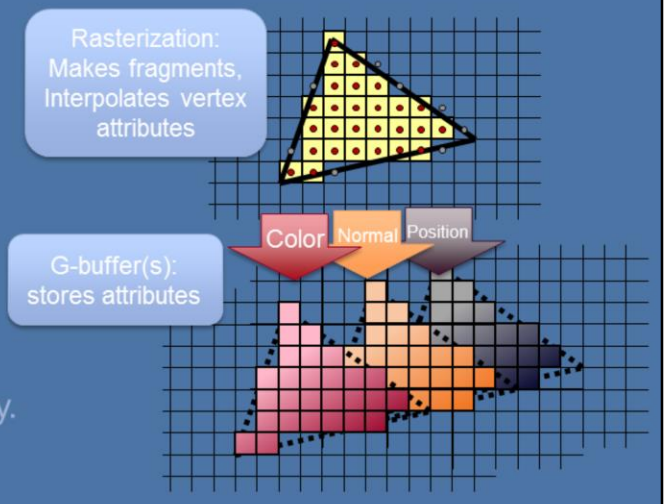

CHALMERS
TRADITIONAL DEFERRED SHADING
Real-Time Many-Light Management and Shadows with Clustered Shading – 2015



- The basic goal is to solve both the overdraw problem and light assignment problem by decoupling geometry from shading.<click>
- Deferred shading as we know it today with G-Buffers was introduced in 1990.
- Although the term ‘deferred shading’ actually was coined later.
- Started appearing in games around 2005, with the introduction of MRT.

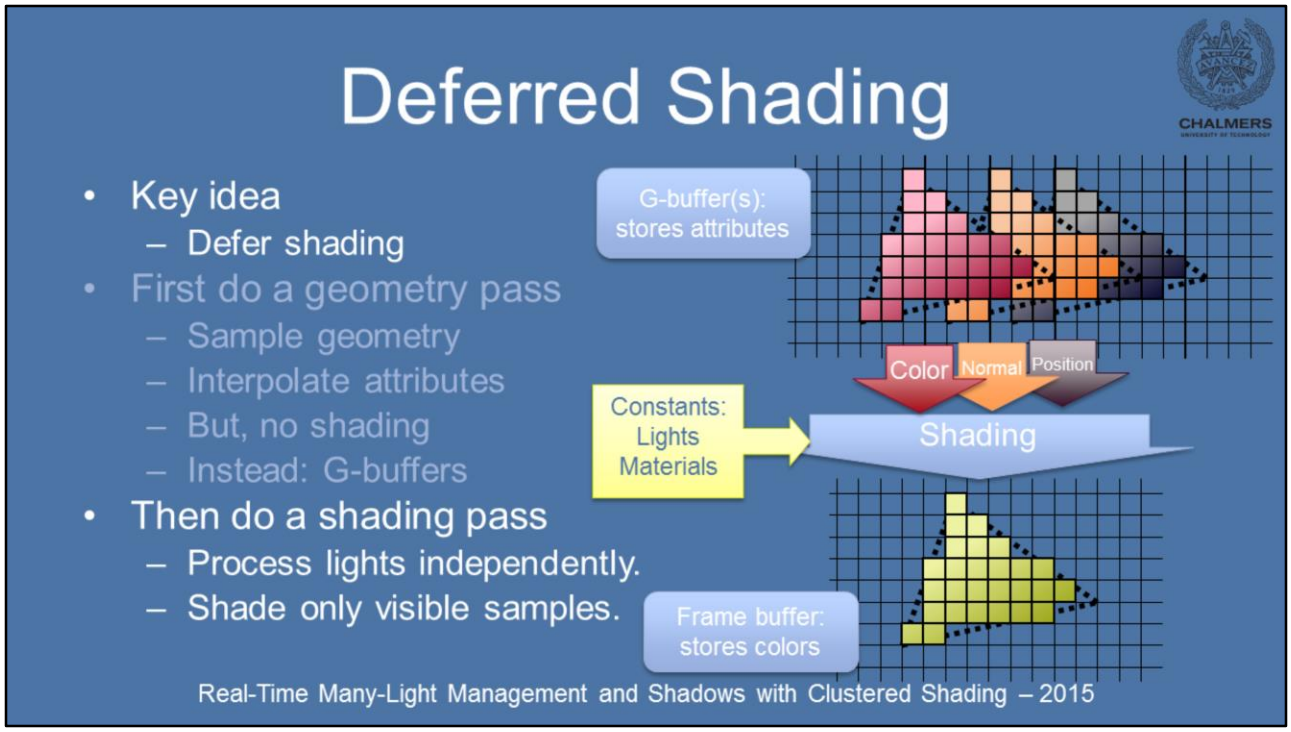
Deferred Shading

- Key idea
  - Defer shading
- First do a geometry pass
  - Sample geometry
  - Interpolate attributes
  - But, no shading
  - Instead: G-buffers
- Then do a shading pass
  - Process lights independently.
  - Shade only visible samples.



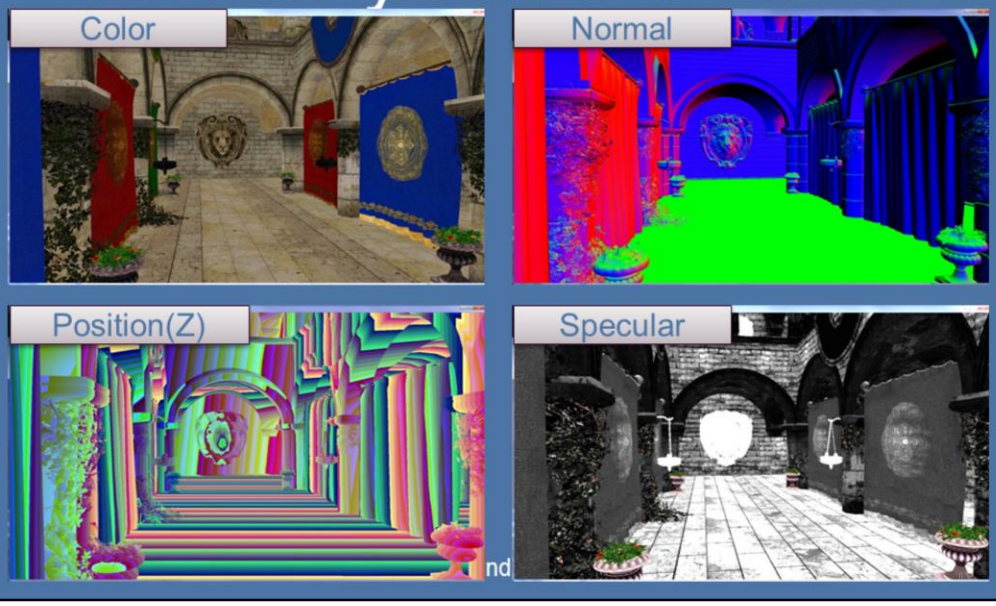
Real-Time Many-Light Management and Shadows with Clustered Shading – 2015

- The key idea is to defer, that is wait with, the shading to its own pass.
- So in a first pass, triangle geometry is transformed and sampled into fragments.
- But instead of shading, the per-pixel geometry attributes are stored in what is called G-buffers.
- The important thing is that they store all the attributes that can change per pixel.<click>



- When all geometry has been processed a separate shading pass is performed.
- And this pass is then independent of the scene geometry representation, and only works on the actually visible samples.

Geometry Pass: G-Buffers



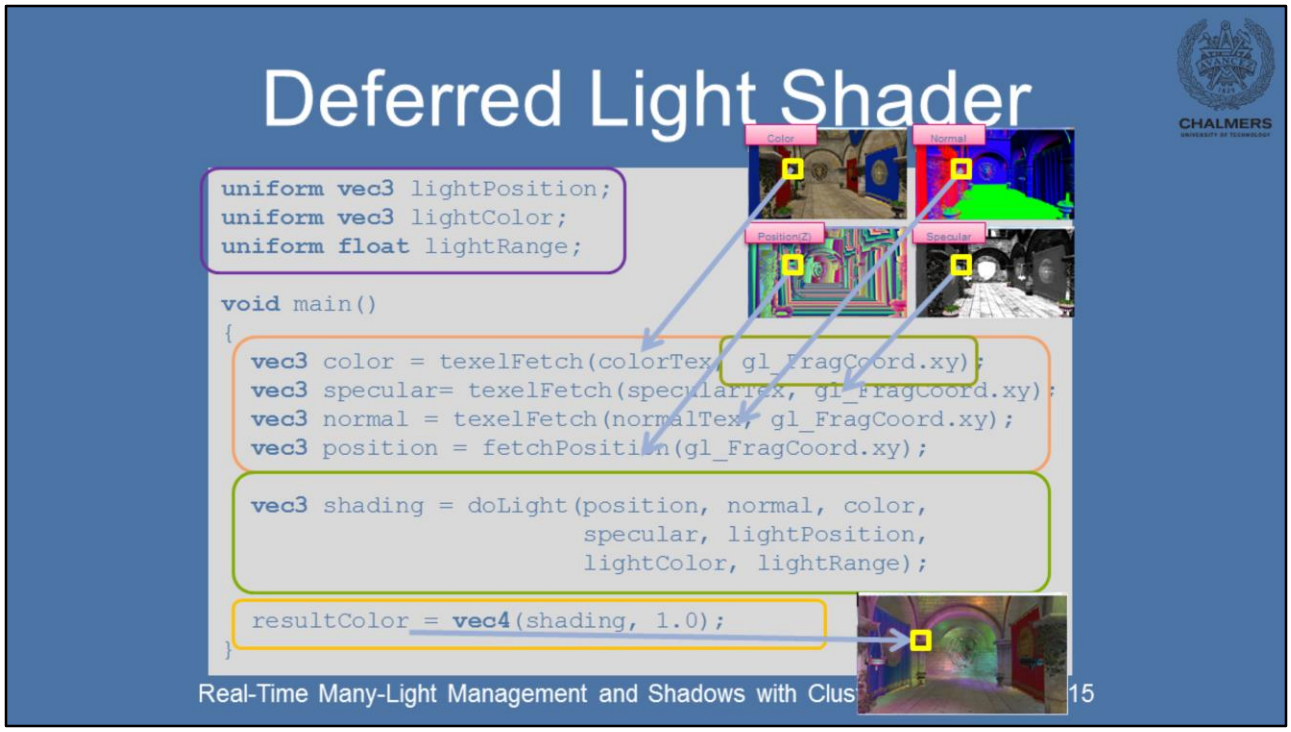
- So a set of G-Buffers might look like this.

Shading Pass

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

Real-Time Many-Light Management and Shadows with Clustered Shading – 2015

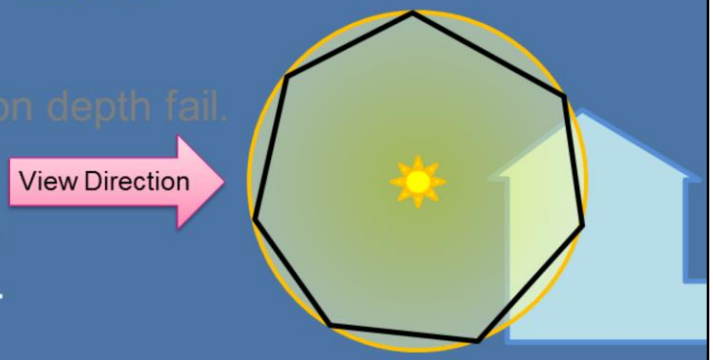
- The way we solve the light assignment is to use the rasterizer to draw the bounding volumes of each light <click> as geometry to the screen<click>
- In the fragment shader of these we compute the shading.
- Which is then blended into the frame buffer.
- This is done independently for each light.



- 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]

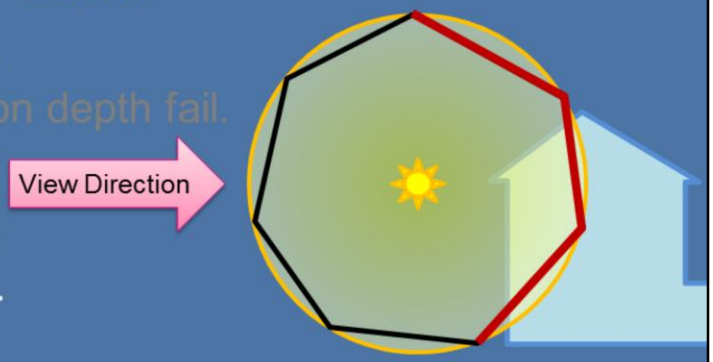
- Analogous to stencil shadows.
- Draw light volume twice.
  - First back faces.
    - Set stencil mask on depth fail.
  - Then front faces.
    - Test stencil mask.
    - Compute shading.



Real-Time Many-Light Management and Shadows with Clustered Shading – 2015

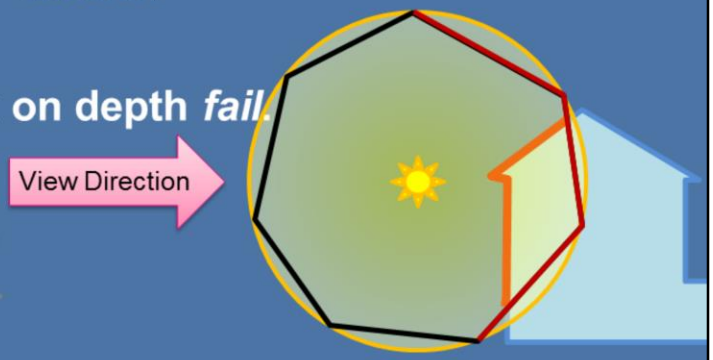
Stencil Buffer Optimization [Arvo03]

- 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]

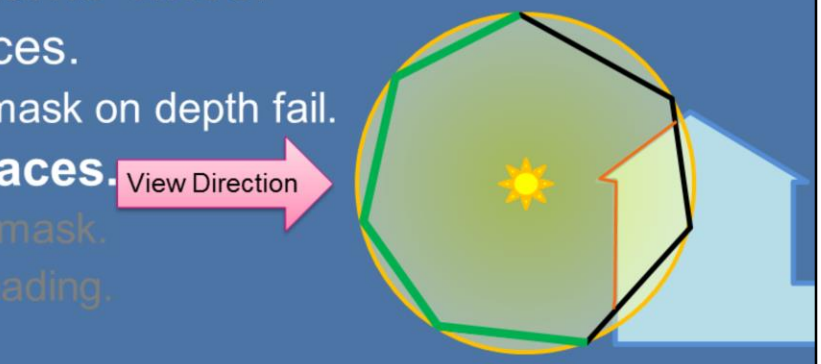
- Analogous to stencil shadows.
- Draw light volume twice.
  - First back faces.
    - **Set stencil mask on depth *fail*.**
  - Then front faces.
    - Test stencil mask.
    - Compute shading.



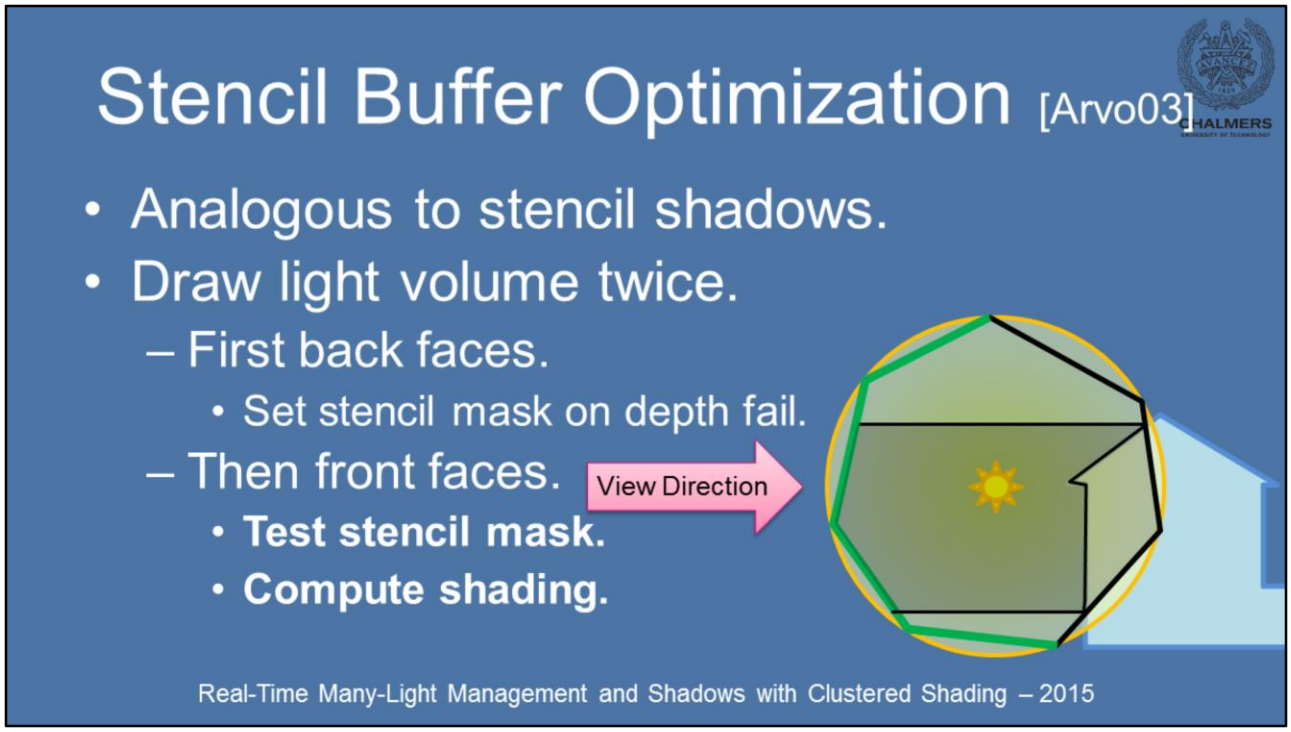
Real-Time Many-Light Management and Shadows with Clustered Shading – 2015

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.



Real-Time Many-Light Management and Shadows with Clustered Shading – 2015

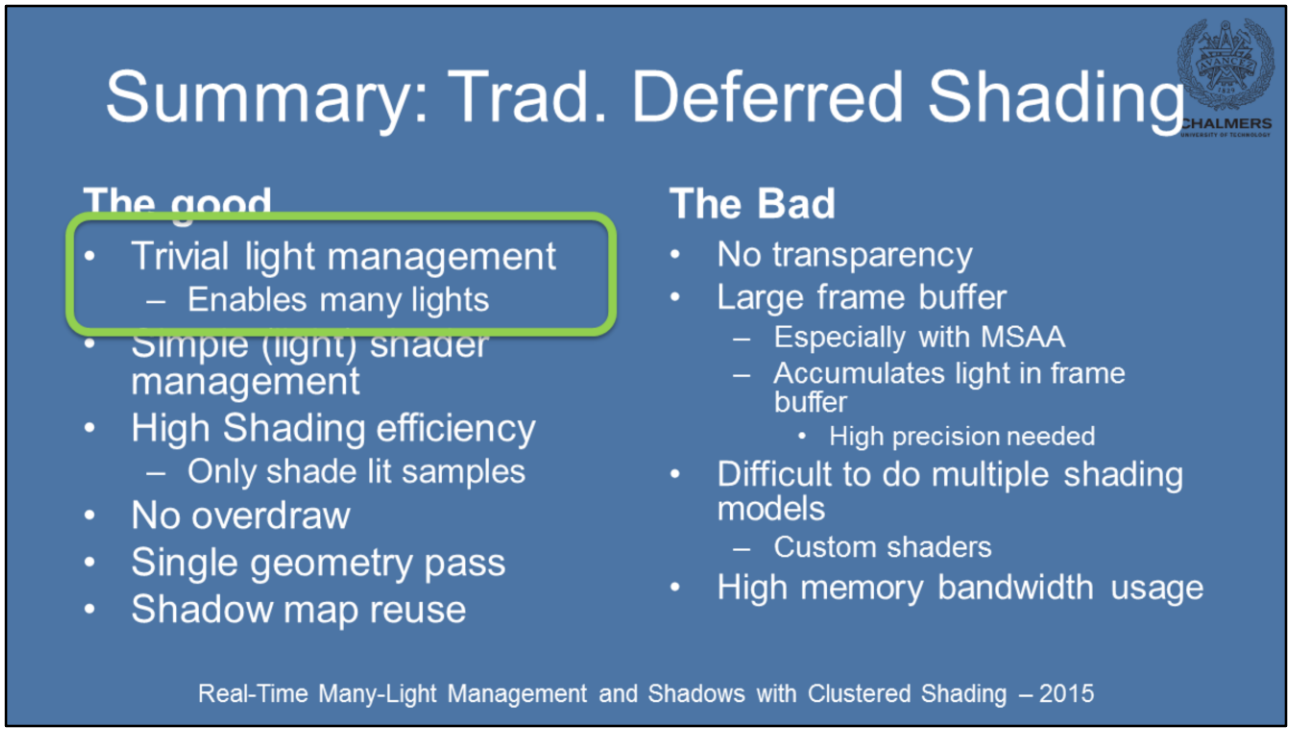


- A very important optimization is using the stencil buffer to exclude any pixels that are outside the light volume.
- This improves efficiency by only shading exactly those pixels that are needed.
- It is however unclear if it can be efficiently implemented on modern hardware for many lights.

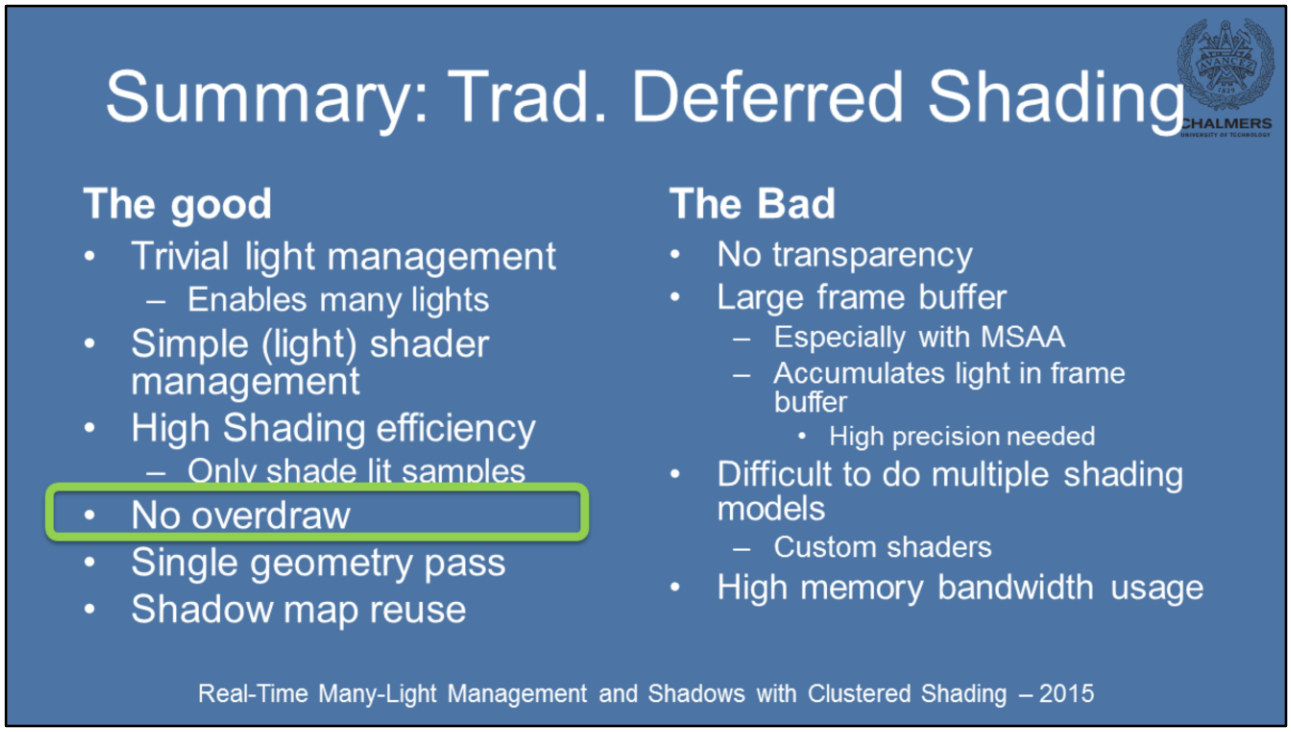
Other Variants

- 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

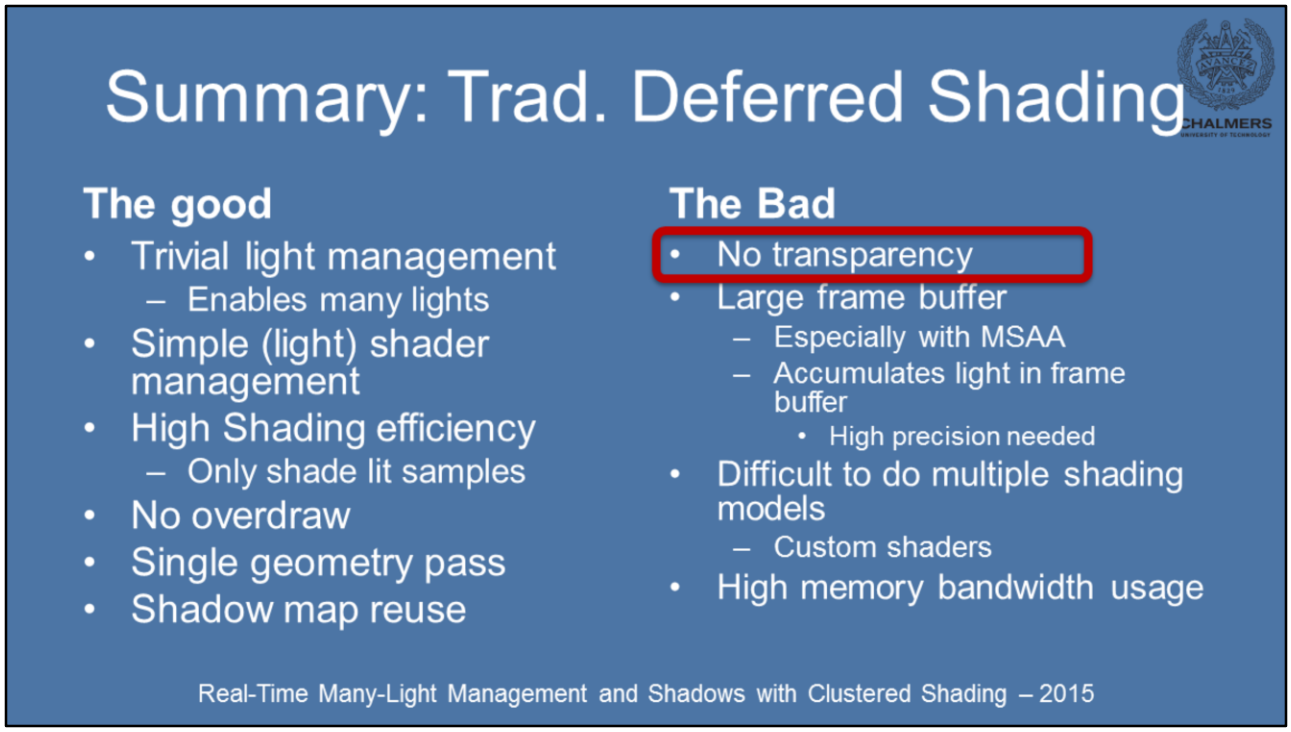
Real-Time Many-Light Management and Shadows with Clustered Shading – 2015



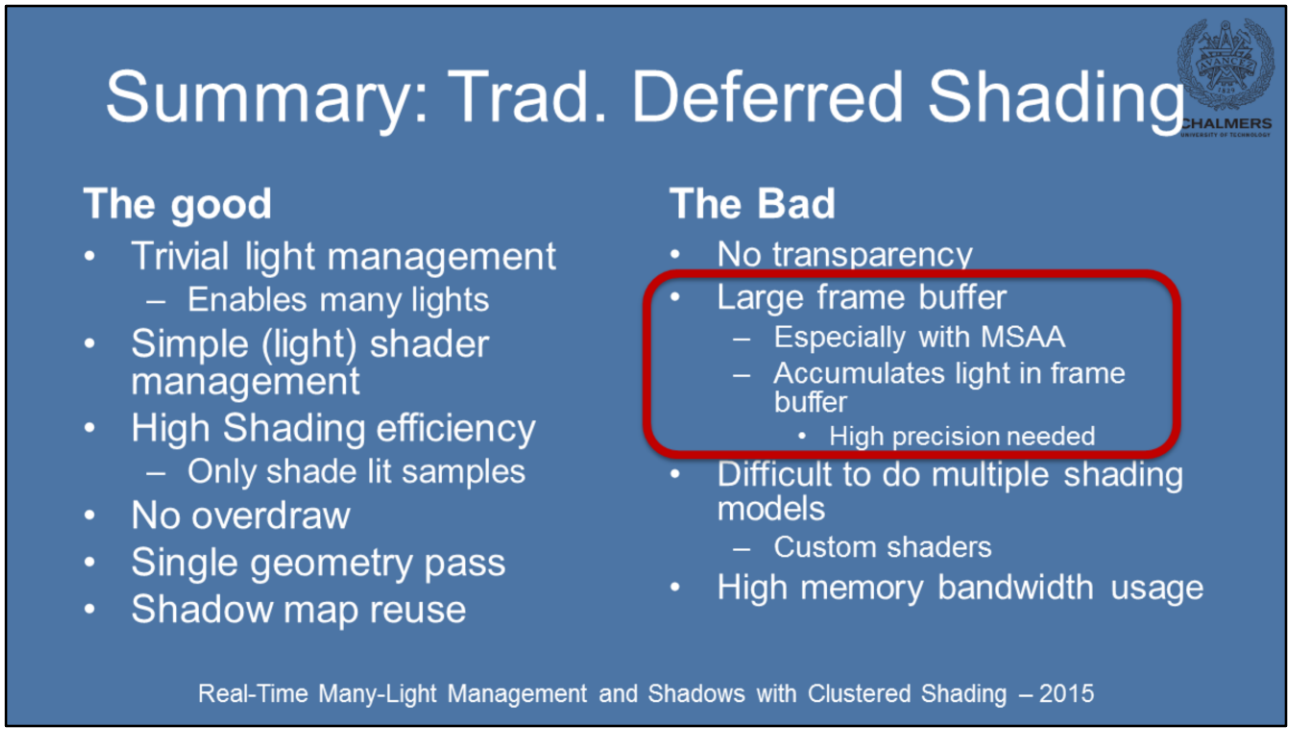
- The large advantages are, especially when the technique was introduced,
- Trivial light management, you just cull and draw the light bounds,
- And light shaders only need to support one light type.



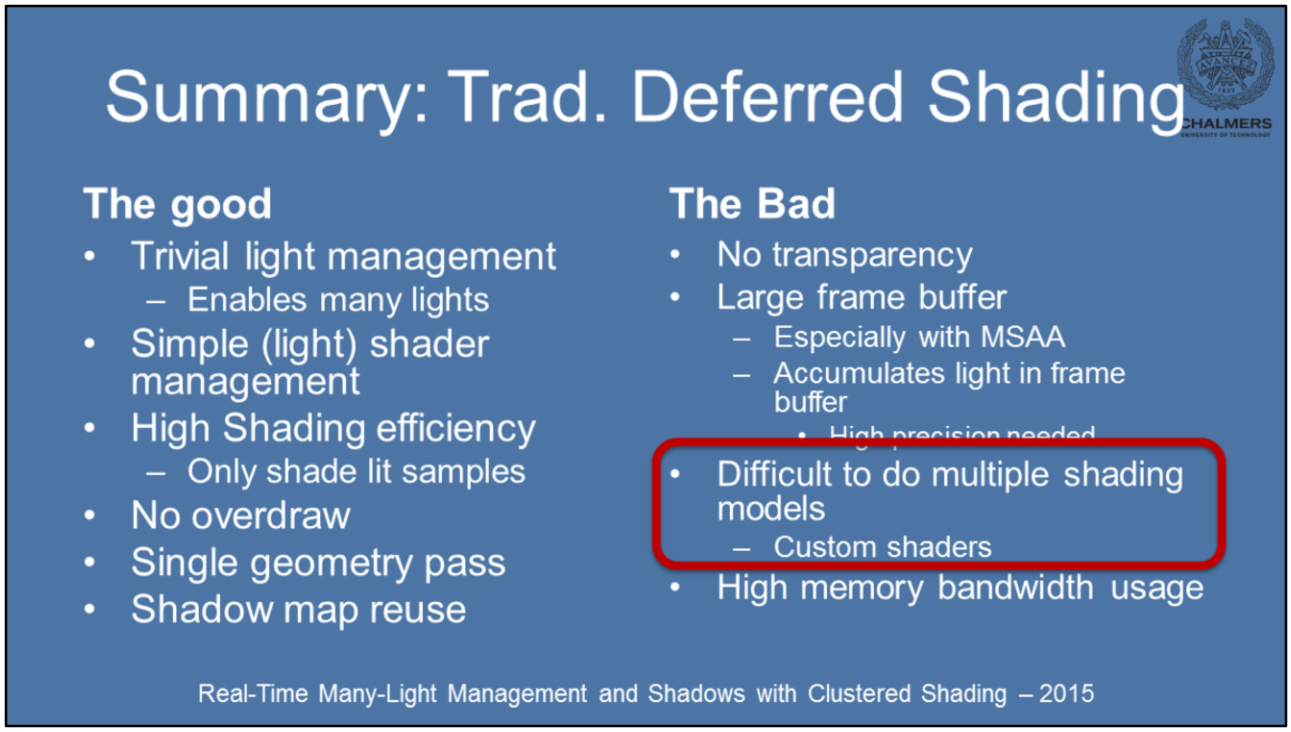
- The large advantages are
- Efficiency is very high, in that only samples that are actually affected by lights need to be shaded.



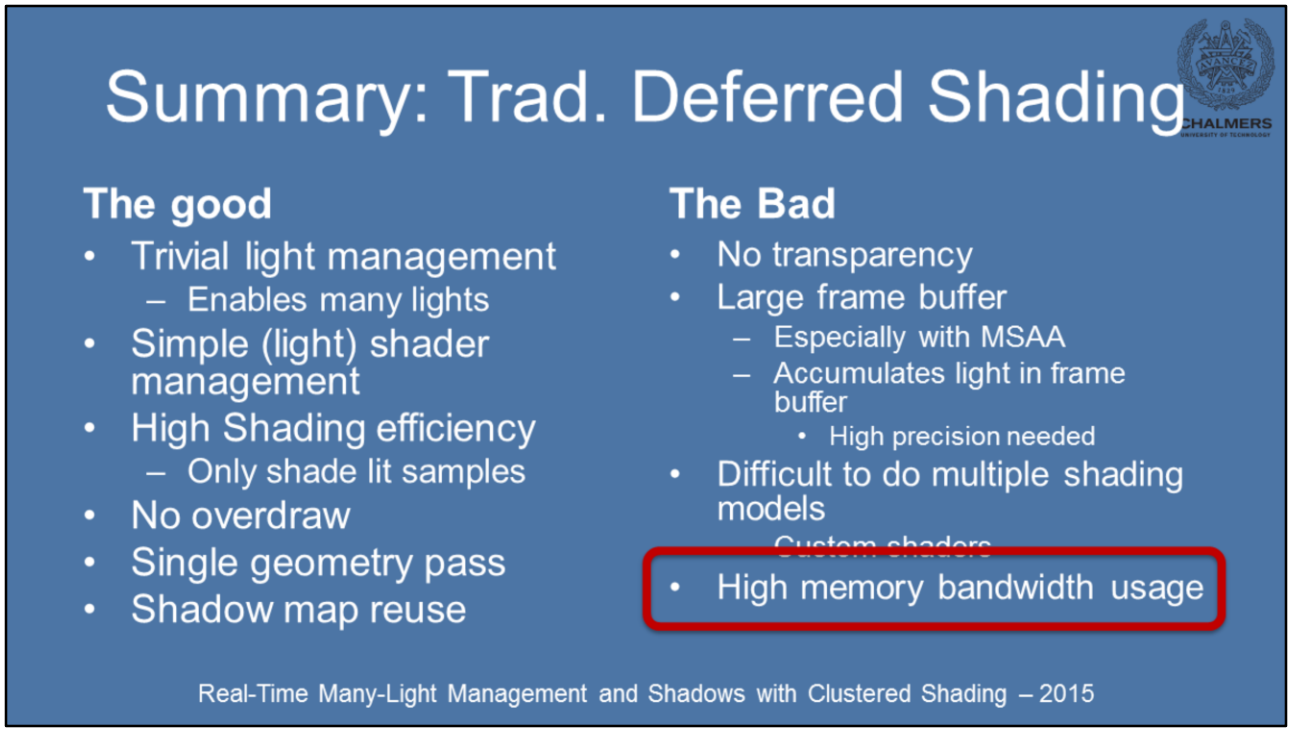
- On the down side,
- Transparency is not readily supported



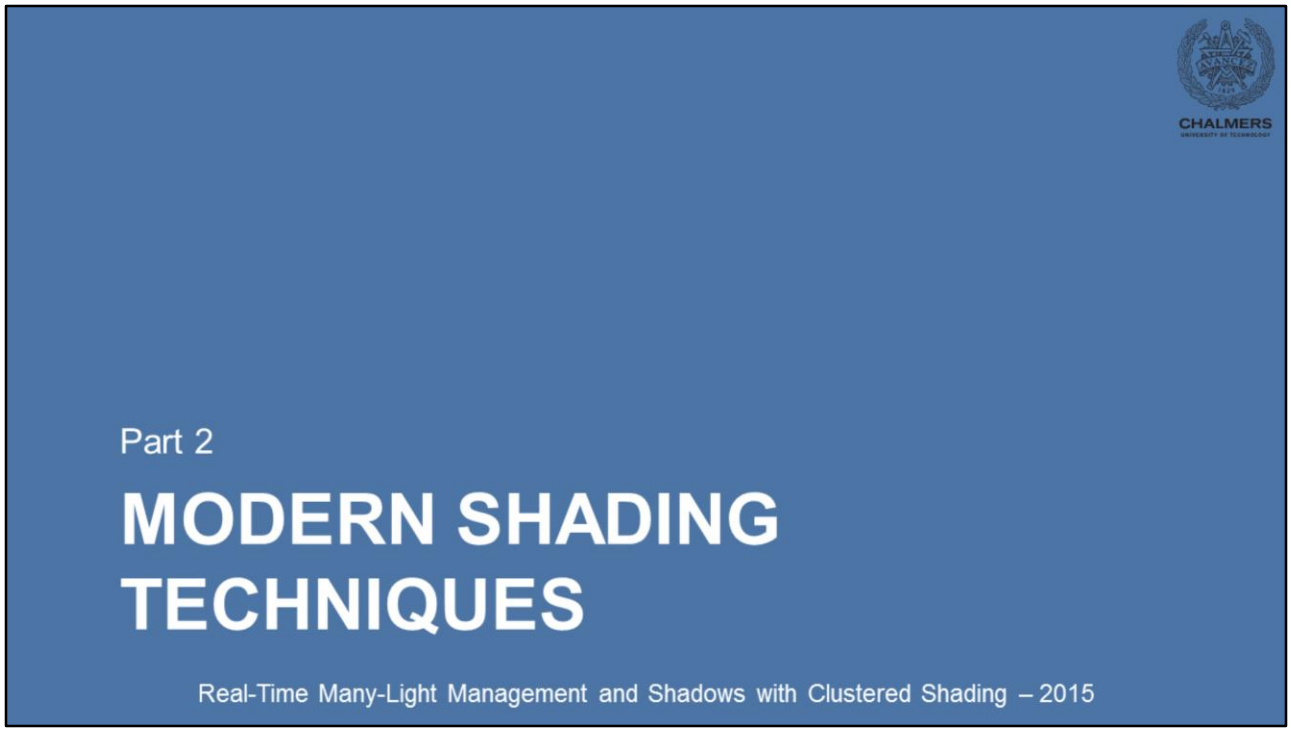
- On the down side,
- We get very large frame buffers, especially with MSAA
- And because we need higher precision for light accumulation.



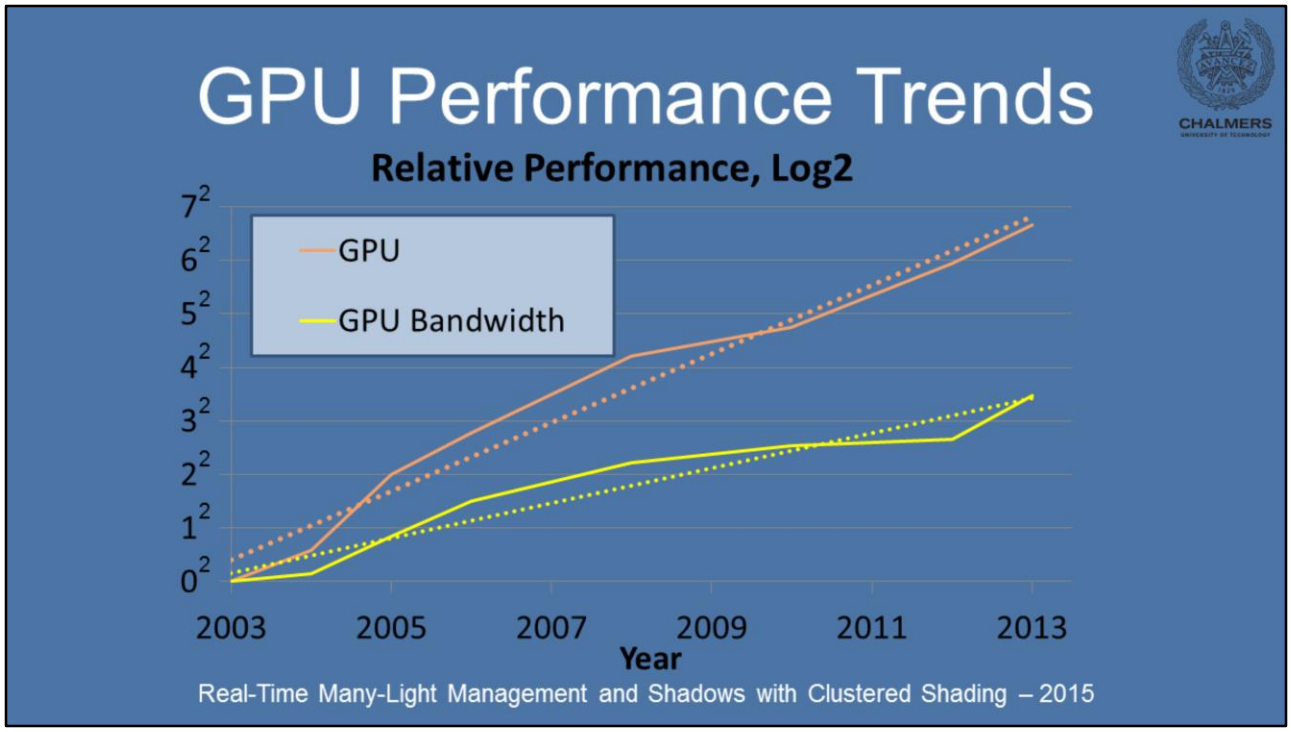
- On the down side,
- It is more difficult to do custom shaders for certain geometry.



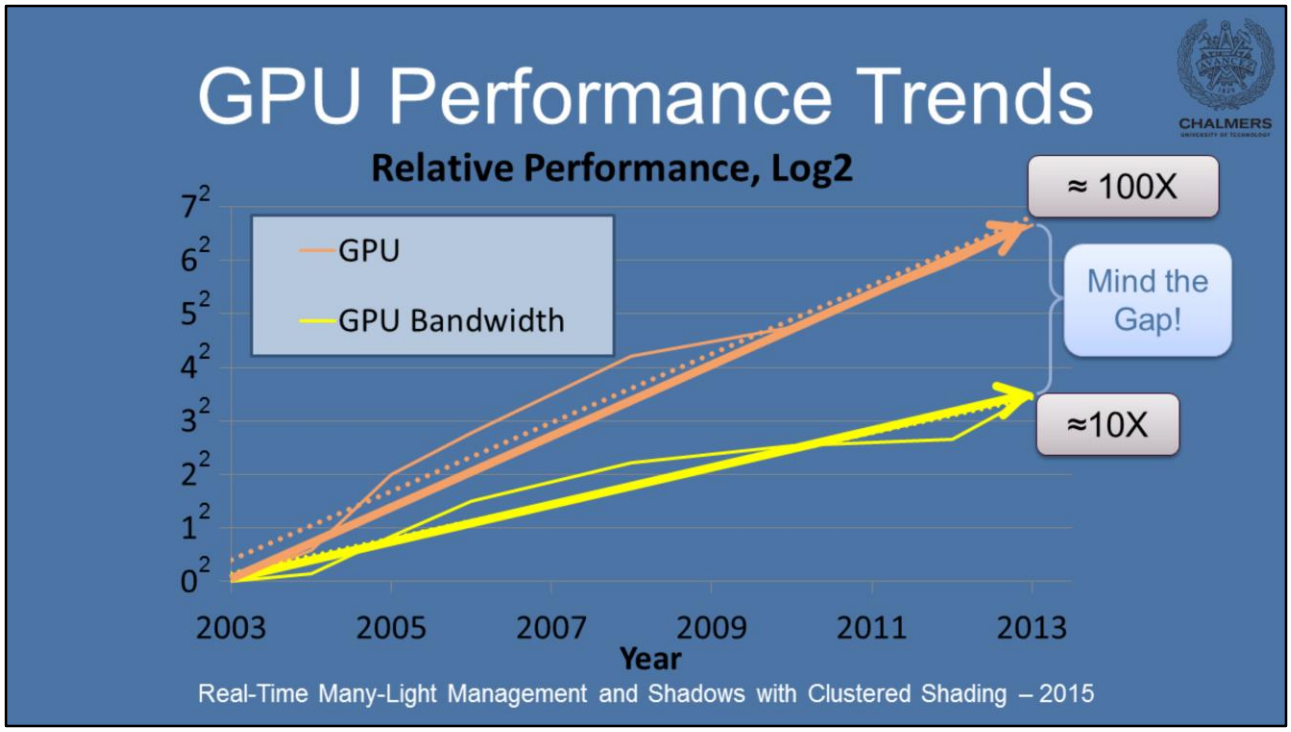
- There are many other pros and cons, but we'll focus on the most problematic:
  - High memory bandwidth requirements,



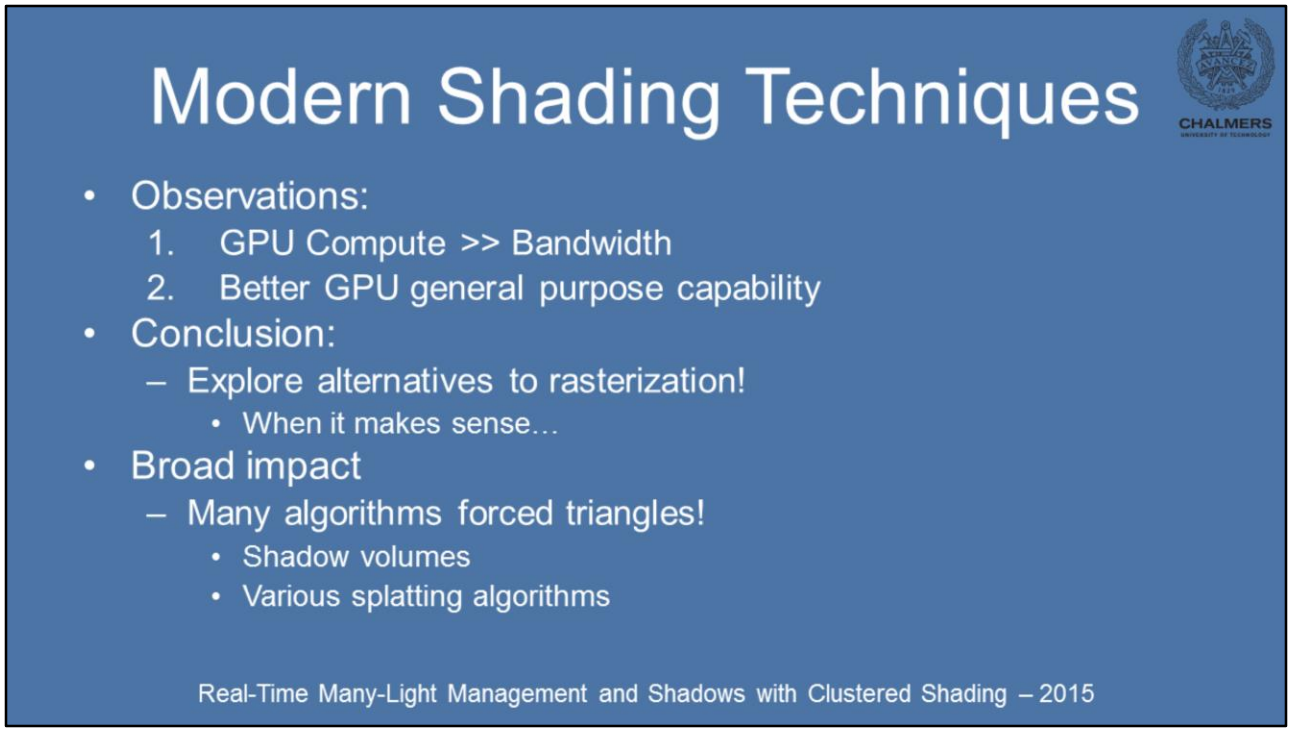
- This last problem brings us to the topic of modern shading techniques, so we'll look into this in some more detail.



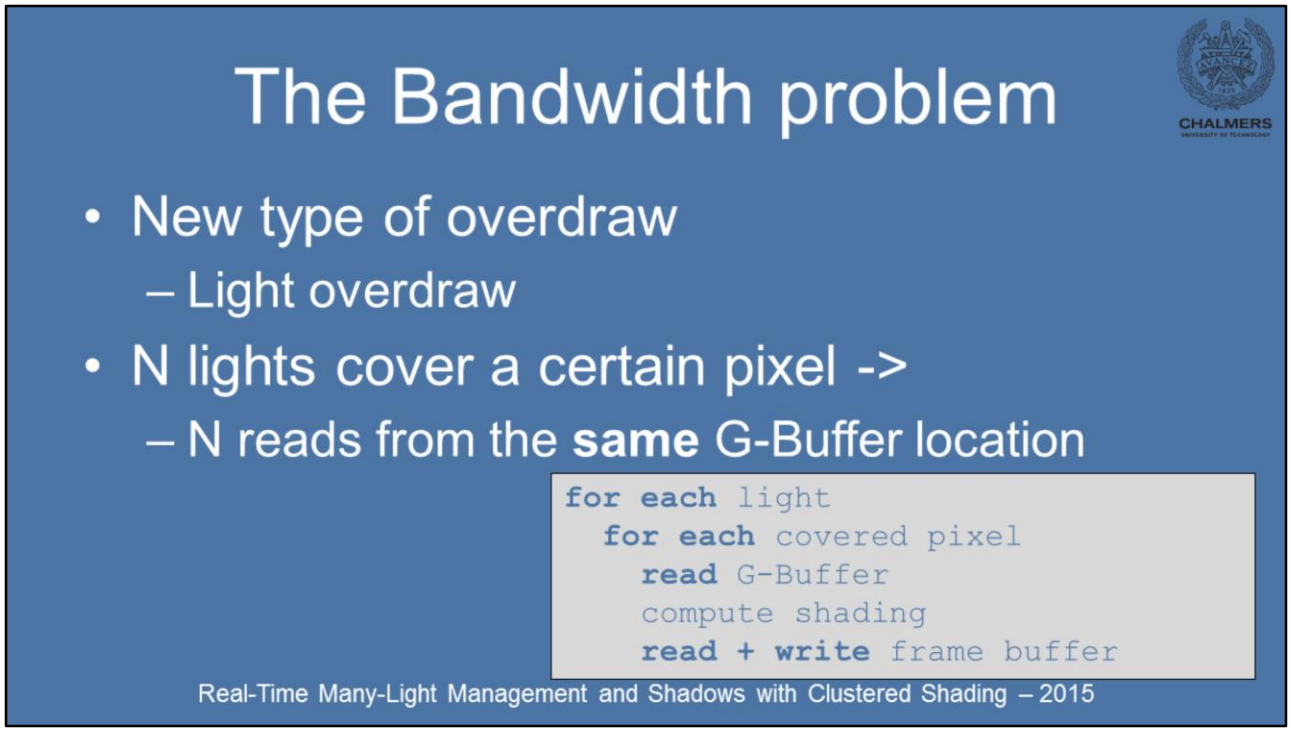
- The graph shows the relative performance trends of top NVIDIA GPUs over 10 years
- Showing clear exponential growth in both compute and bandwidth…



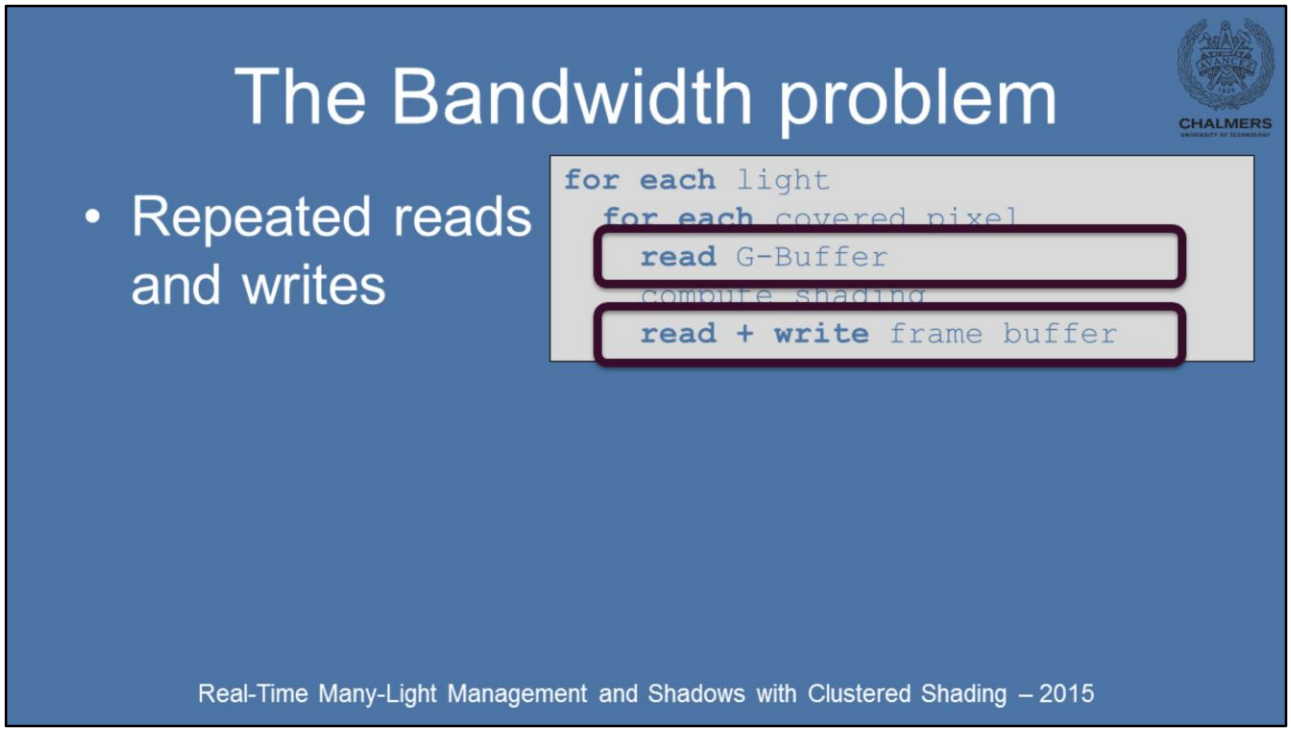
- Note, however, the difference in exponent!<click>
- Over the 10 years plotted that amounts to a difference of a factor 10.<click>…
- …for a compute versus memory bound algorithm.
- So, aiming for compute bound is key in future proofing your algorithms.



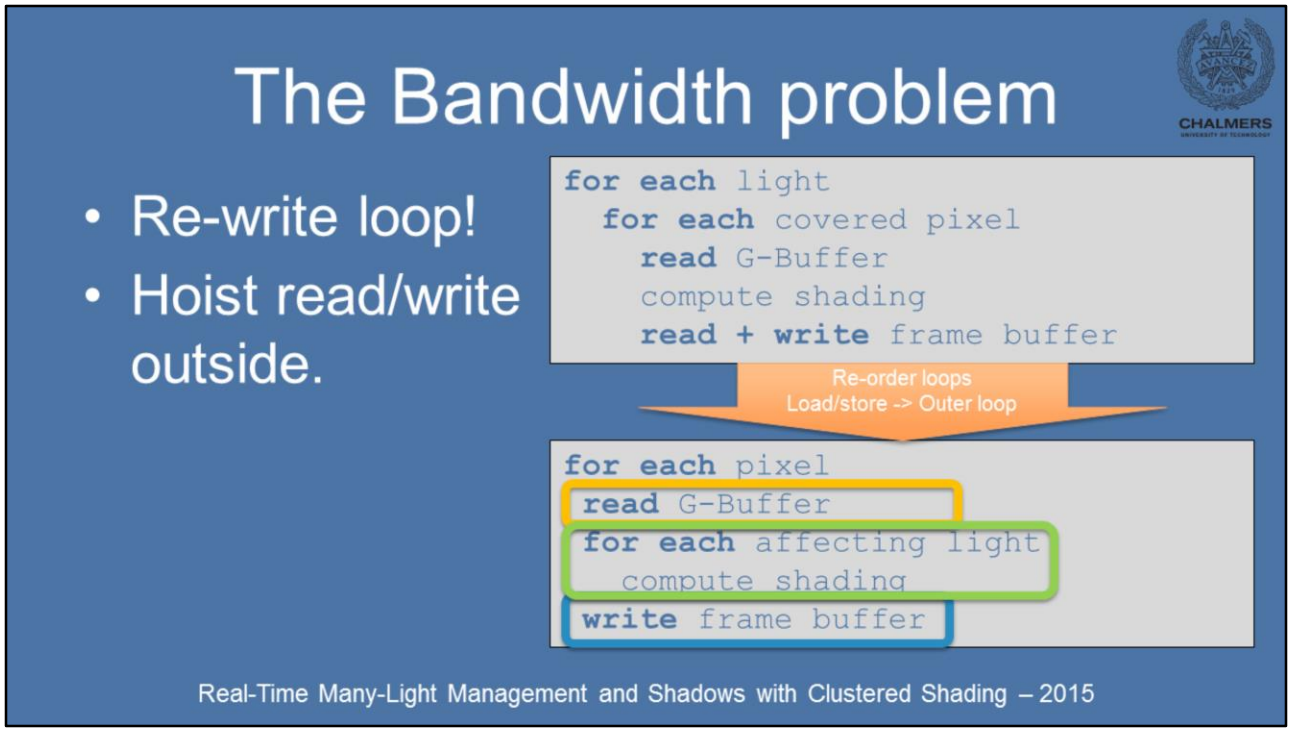
- With this, modern approaches to real-time shading with many lights all take their in the following 2 observations
    - First, as we just saw compute is king.
    - Second, the GPU general purpose programming models and power of these cores has improved tremendously over the last years.
- These observations have broad impact because not long ago triangle rasterization was the only game in town if you wanted real-time performance.
- This means that we are now sometimes stuck with a way of doing things that is no longer suitable.
- Thus I think the field is open to revisit many real-time algorithms with this frame of mind.



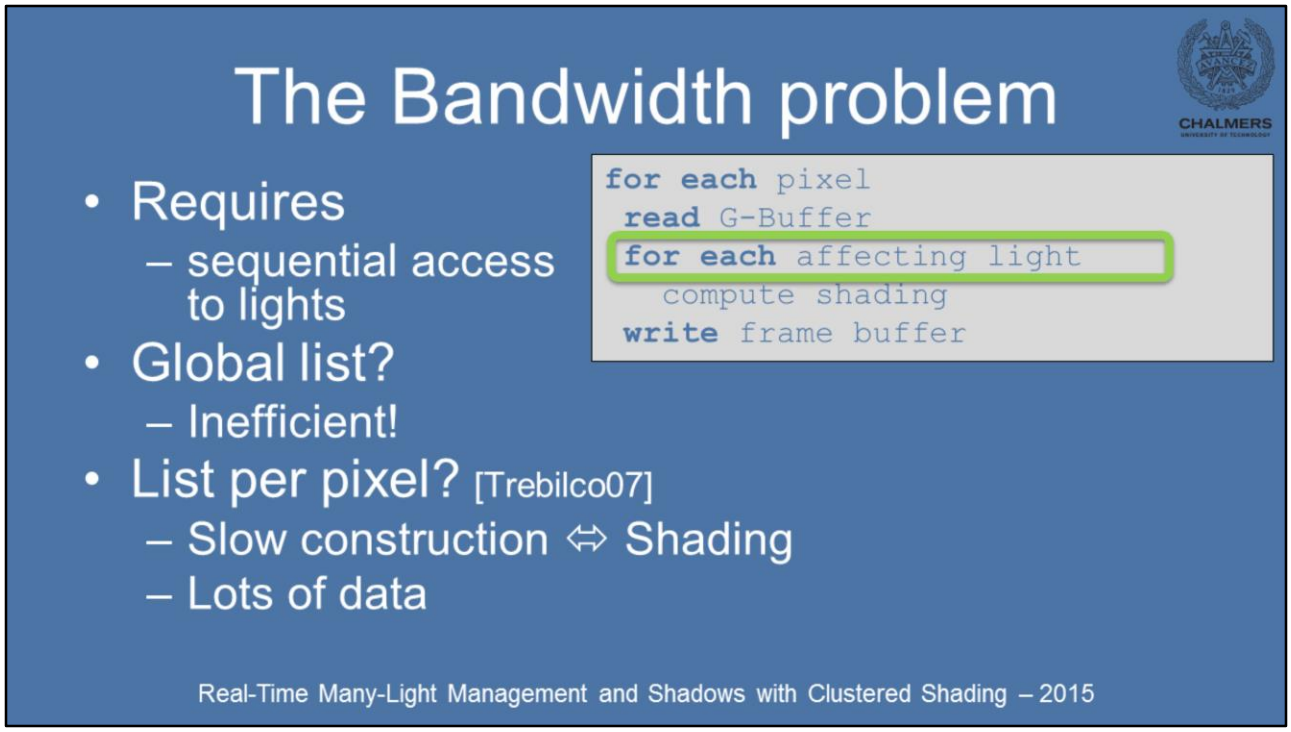
- Tiled and Clustered shading are examples of this, and it brings us back to the bandwidth problem of traditional deferred.
- As we are now drawing the lights, in the shading pass, we have overdraw when many lights overlap the same pixel.
- Schematically deferred shading looks like this, we iterate over each light,
- and then in parallel, by drawing the bounding volume, over all the fragments.



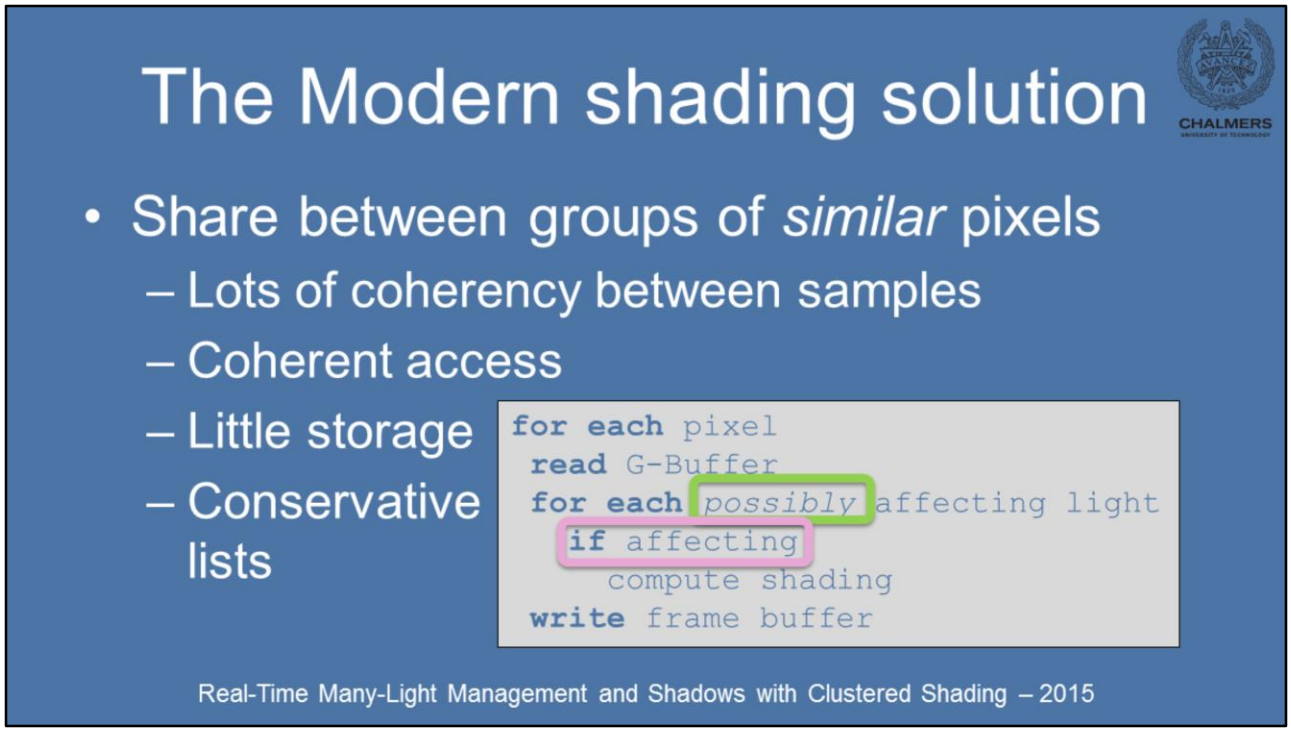
- The problem is from the fact that the innermost loop is over the pixels,
- This which requires repeated reading, and writing of the G-Buffers and frame buffer.



- So we want to re-arrange this loop to make the innermost loop iterate over the lights instead. .<click>
- Then we get this nice compute oriented loop.
- With a single read and write which would solve the bandwidth problem.



- However, 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.



•The modern solution is instead to build list for groups of similar samples.

•The motivation is of course that many pixels will use the same lights in a typical scenario

•Allowing us to share the list over many samples.

•Then 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.

•But at the same time significantly closer to the optimum compared to brute force.

Modern Deferred Shading

- 1. Render Scene to G-Buffers
- 2. Light assignment
  - Build Light Acceleration Structure (Grid)
- 3. Full Screen Pass
  - Quad (or CUDA, or Compute Shaders, or SPUs)
  - For each pixel
    - Fetch G-Buffer Data
    - Look up light list in acceleration structure.
    - Loop over lights and accumulate shading
    - Write shading

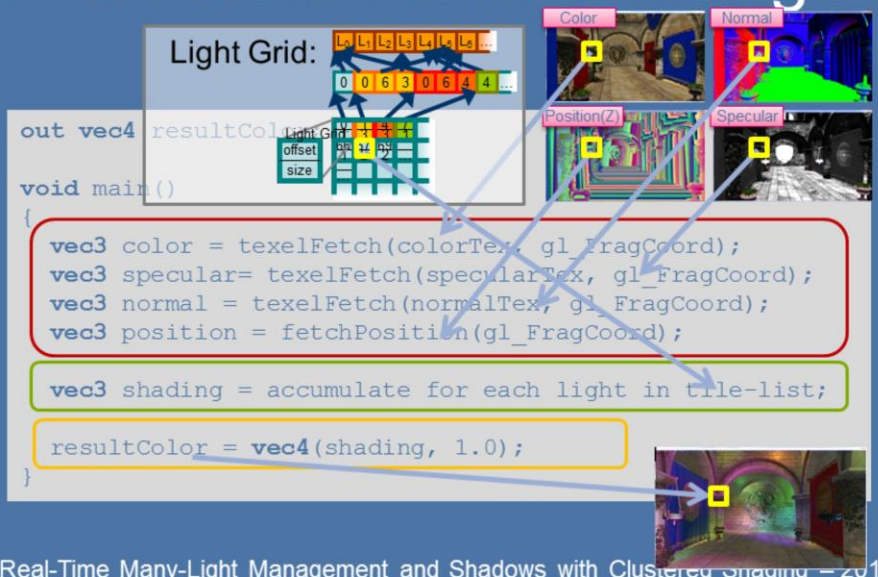
Real-Time Many-Light Management and Shadows with Clustered Shading – 2015

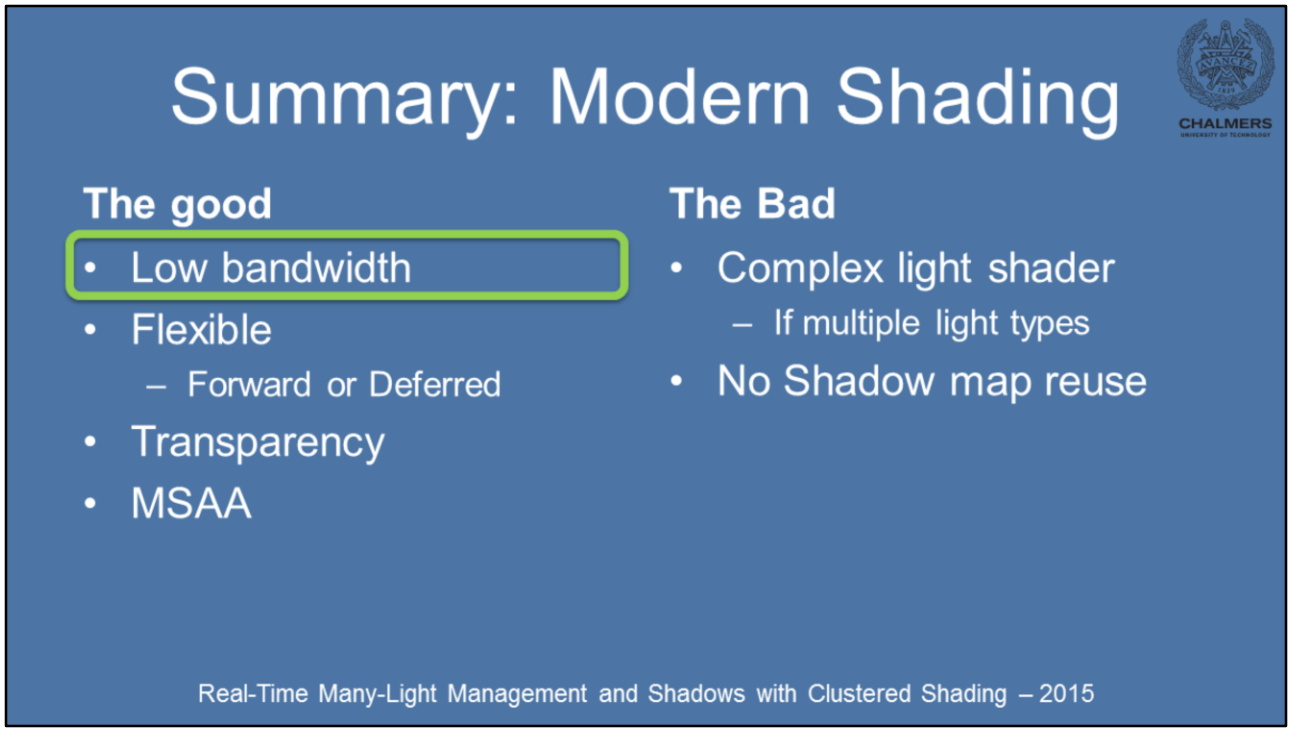
Modern Forward Shading

- (1. Optional: Pre-Z/Geometry Pass)
- 2. Light assignment
  - Build Light Acceleration Structure (Grid)
- 3. Geometry Pass
  - Just your normal shading pass.
  - For each fragment
    - Look up light list in acceleration structure.
    - Loop over lights and accumulate shading
    - Write shading to frame buffer

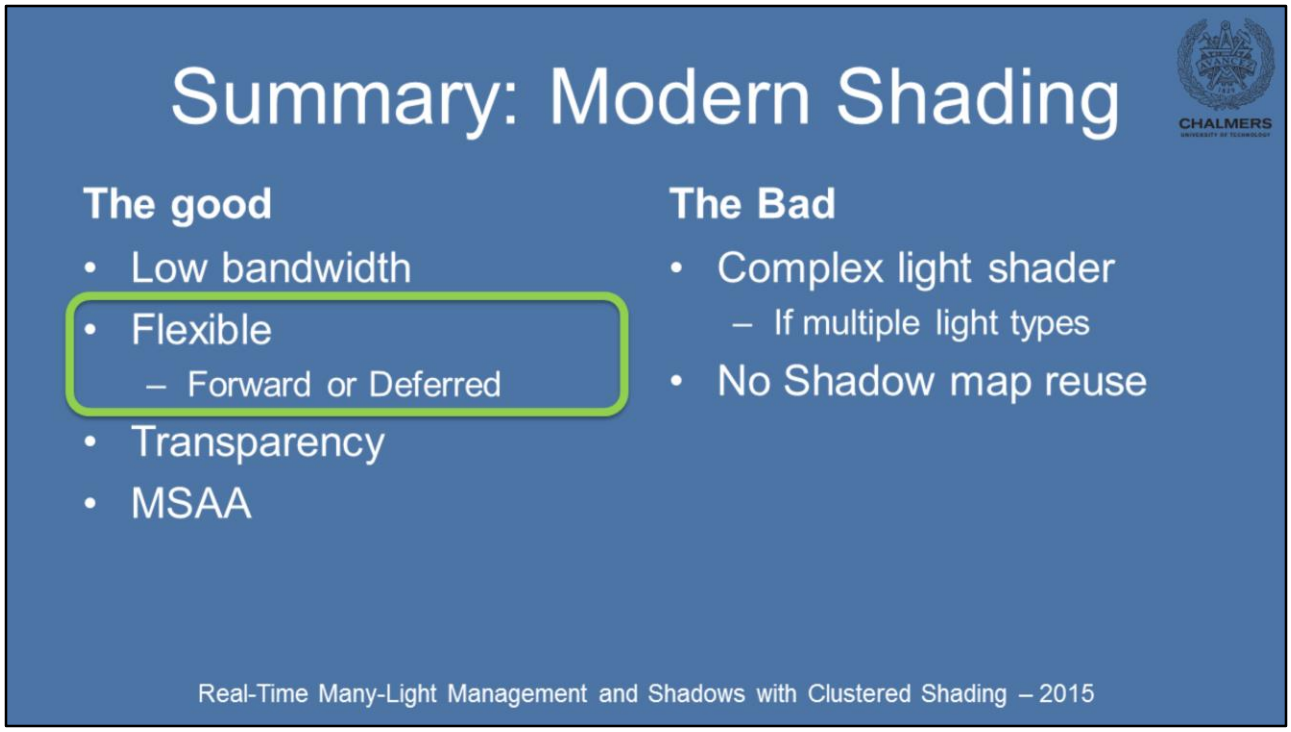
Real-Time Many-Light Management and Shadows with Clustered Shading – 2015

Full-Screen Modern Shading Pass

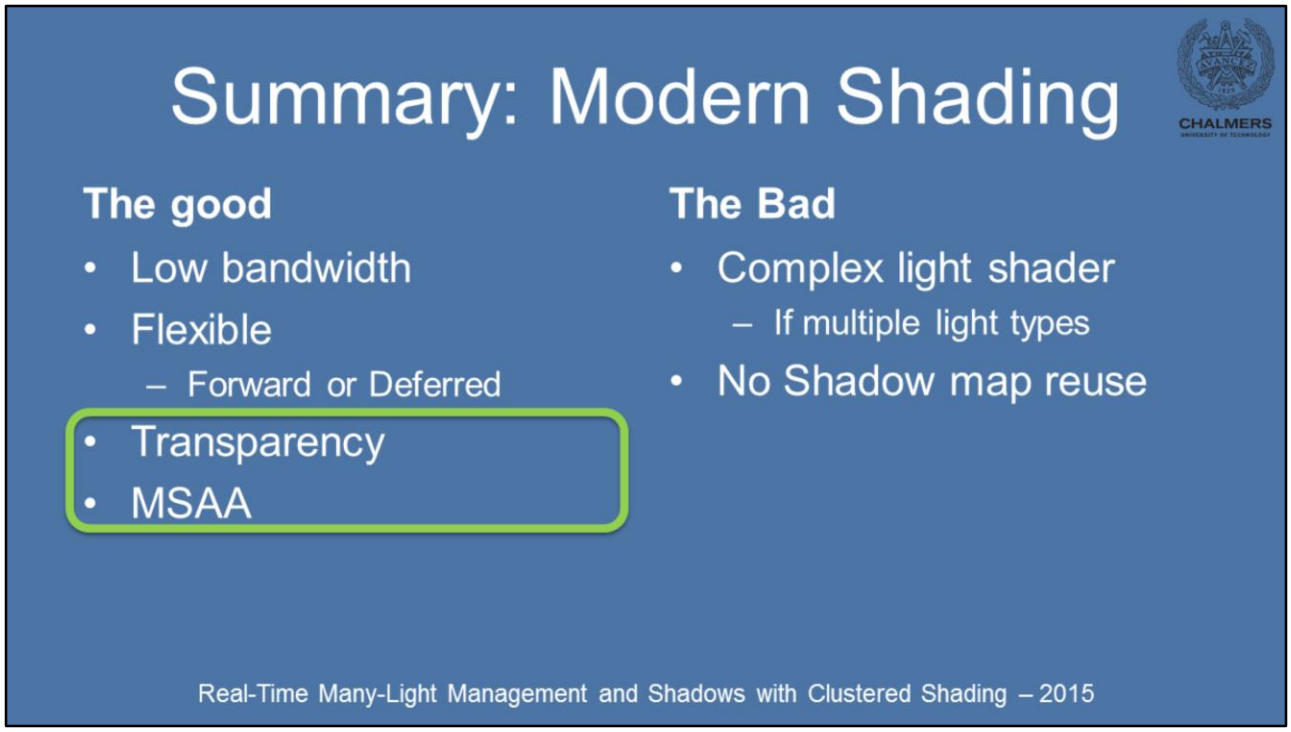




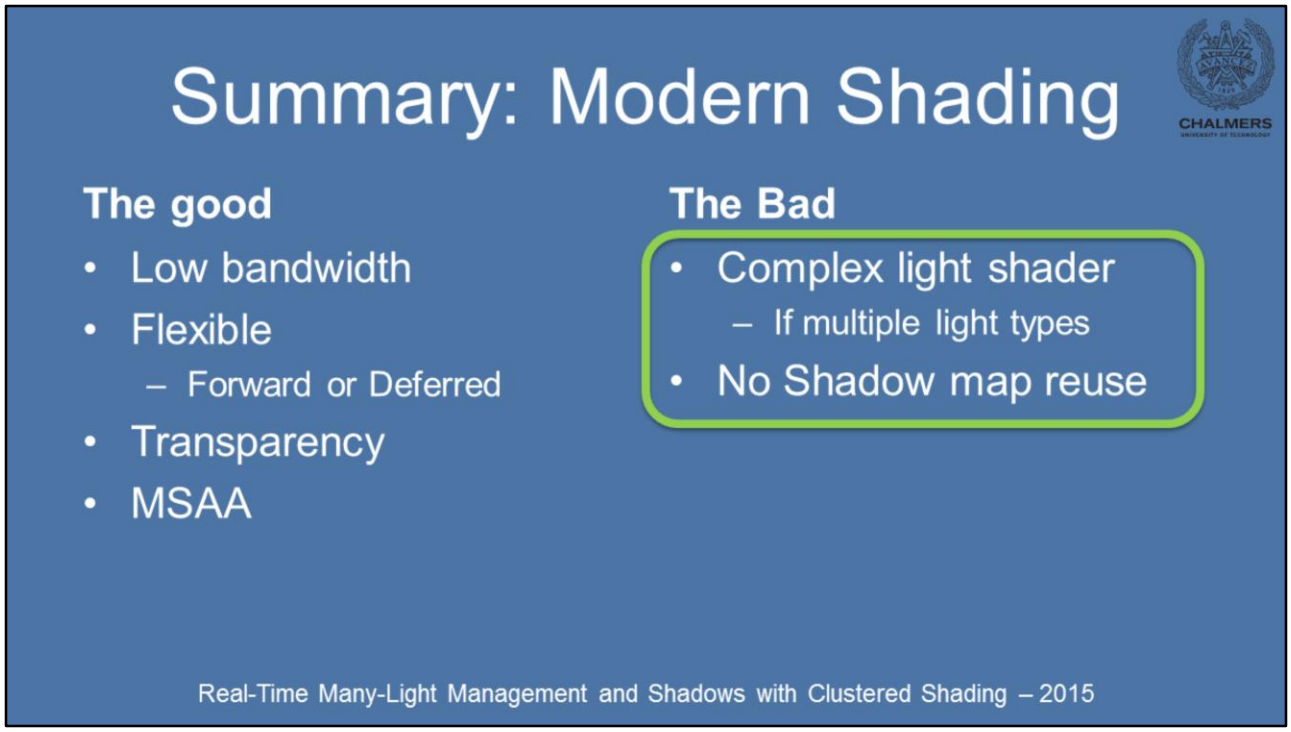
- So, we have pretty much solved the bandwidth problem which is great,
- It matters a lot less suddenly if the G-Buffers are fat as chips



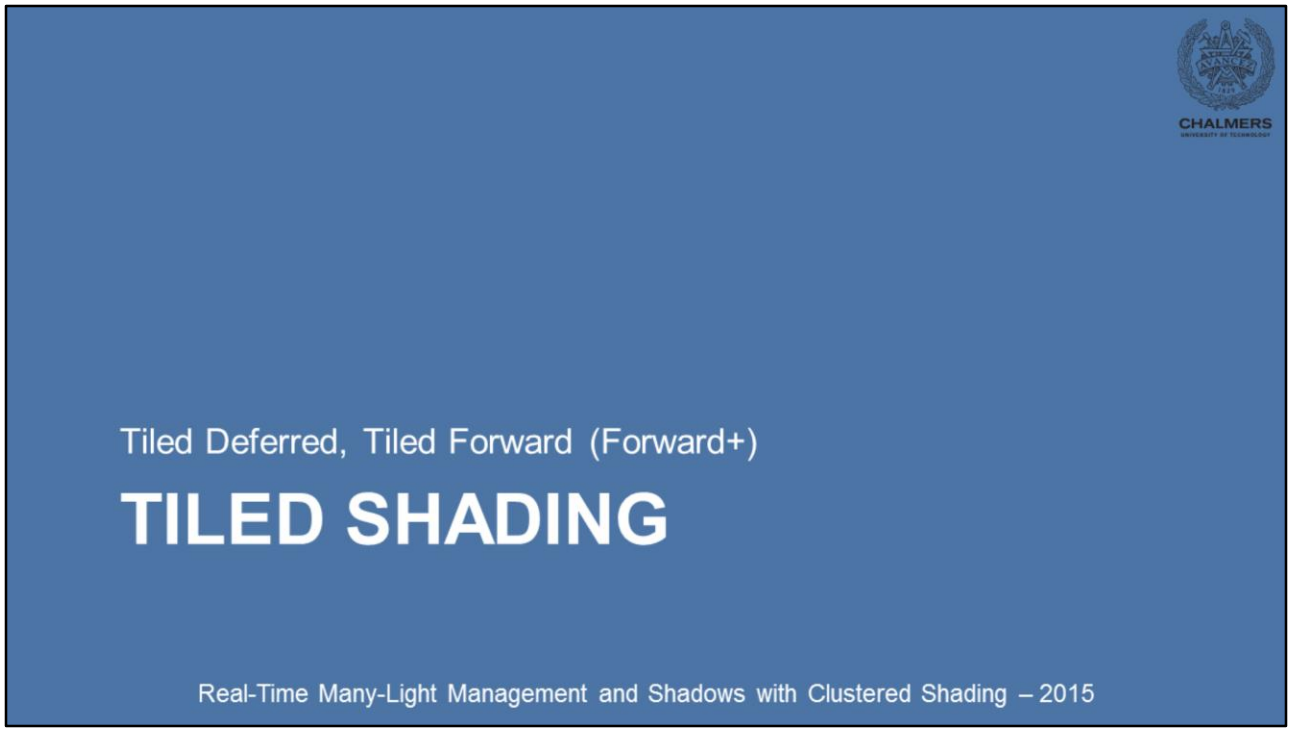
- The techniques are flexible, in that it is fairly trivial to support forward or deferred shading, or both.



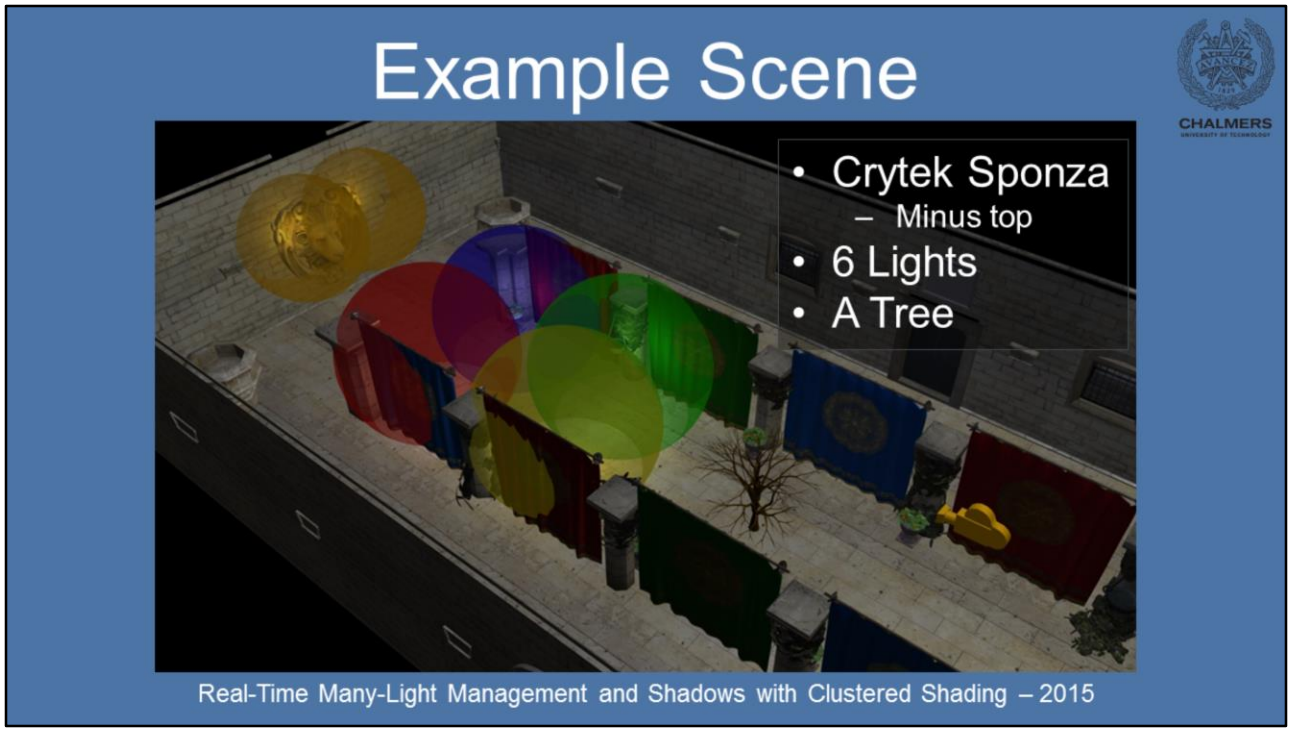
- Using forward, in turn, makes it possible to support transparent and MSAA.



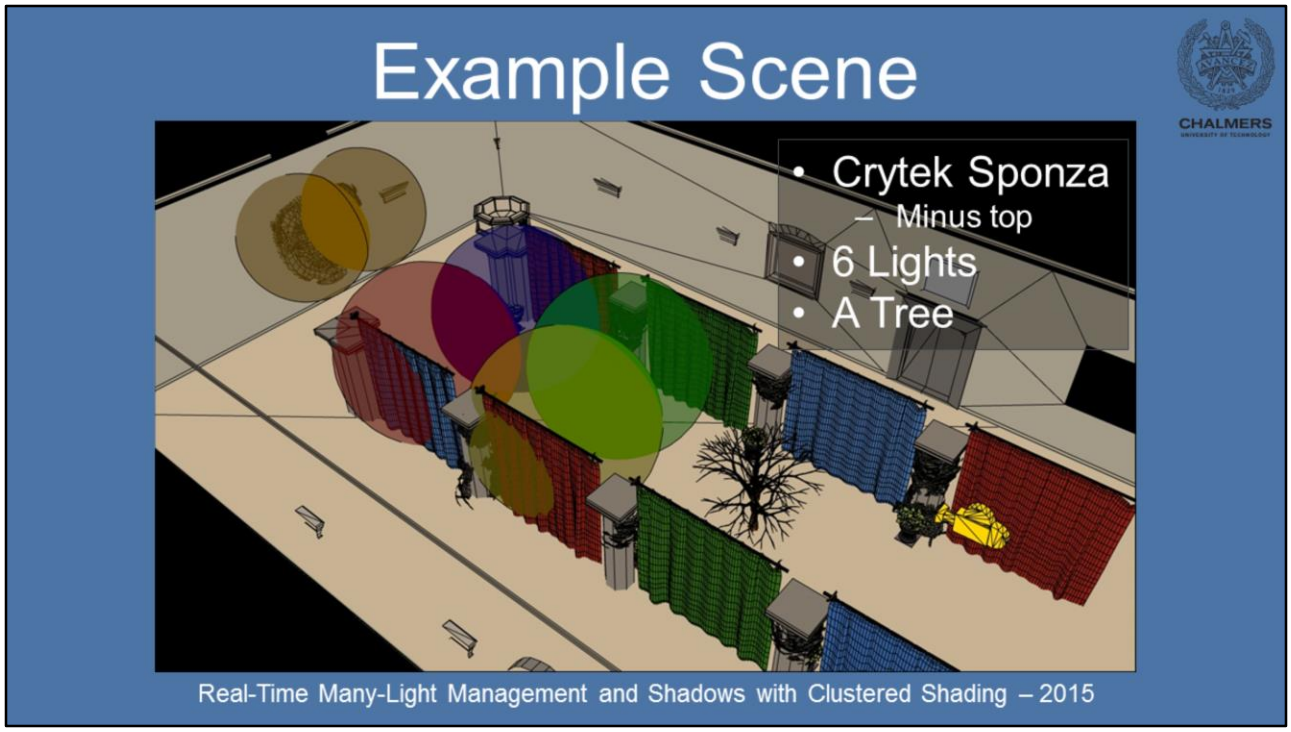
- On the down side, the light shader now must support all light types, which makes it more complex.
- And since lights are not sequentially processed we cannot re-use shadow map storage space between lights.
    - More on this in my second talk!



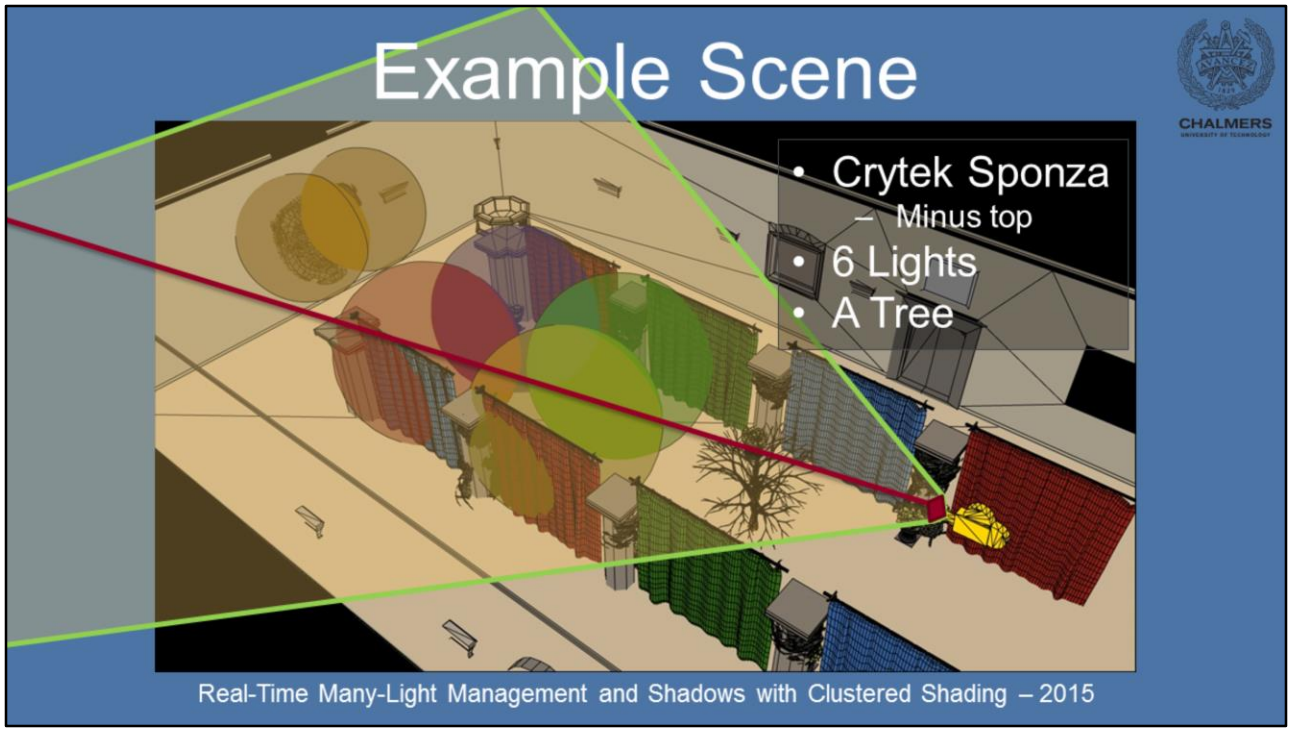
- This brings us to the first of the modern techniques that has been developed in this new, bandwidth constrained and compute oriented, landscape.
- Collectively called Tiled Shading, it covers both deferred and forward variants,
- The forward variant is also referred to as “Forward+” by AMD. I highly recommend using Tiled Forward Shading as this name really communicates something, for roughly the same reason that we don’t name our functions ‘foo’ and ‘bar’… right?



- 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.



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

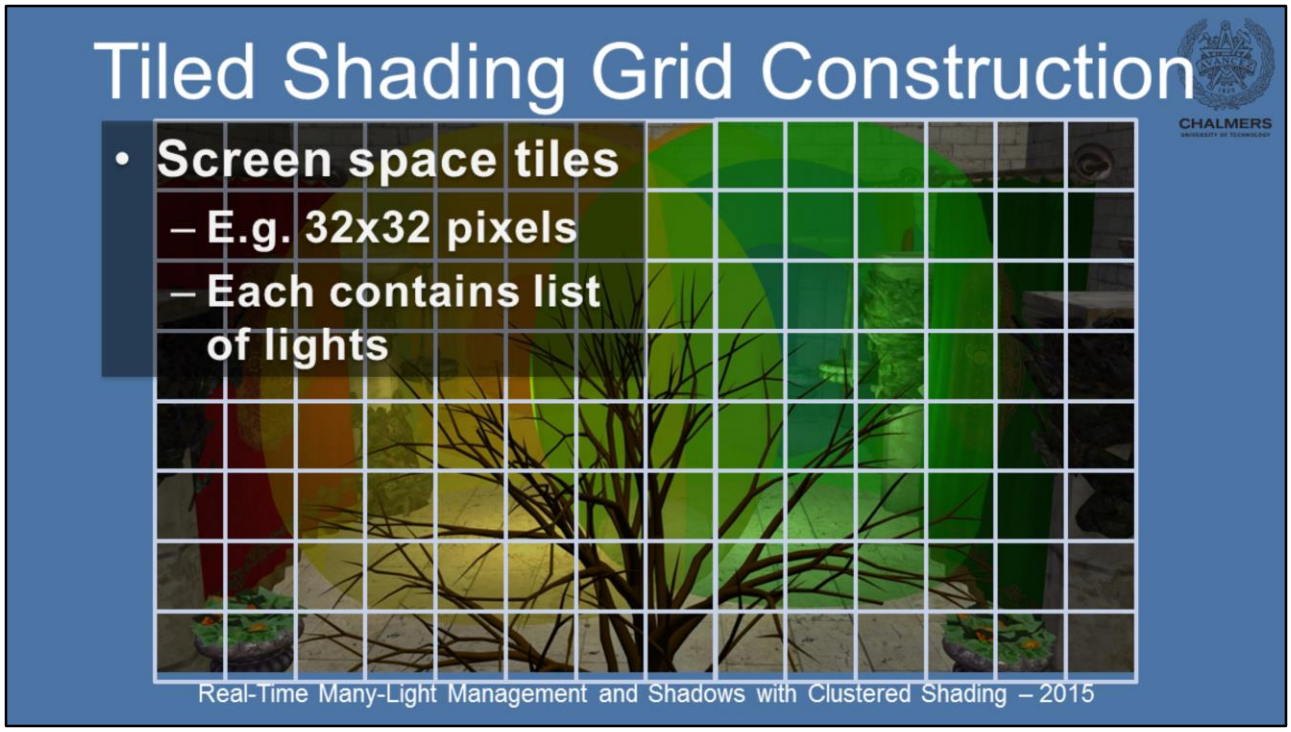


- This is the viewpoint that will be used to demonstrate the algorithms, the camera is looking through the tree towards the lion head on the wall.

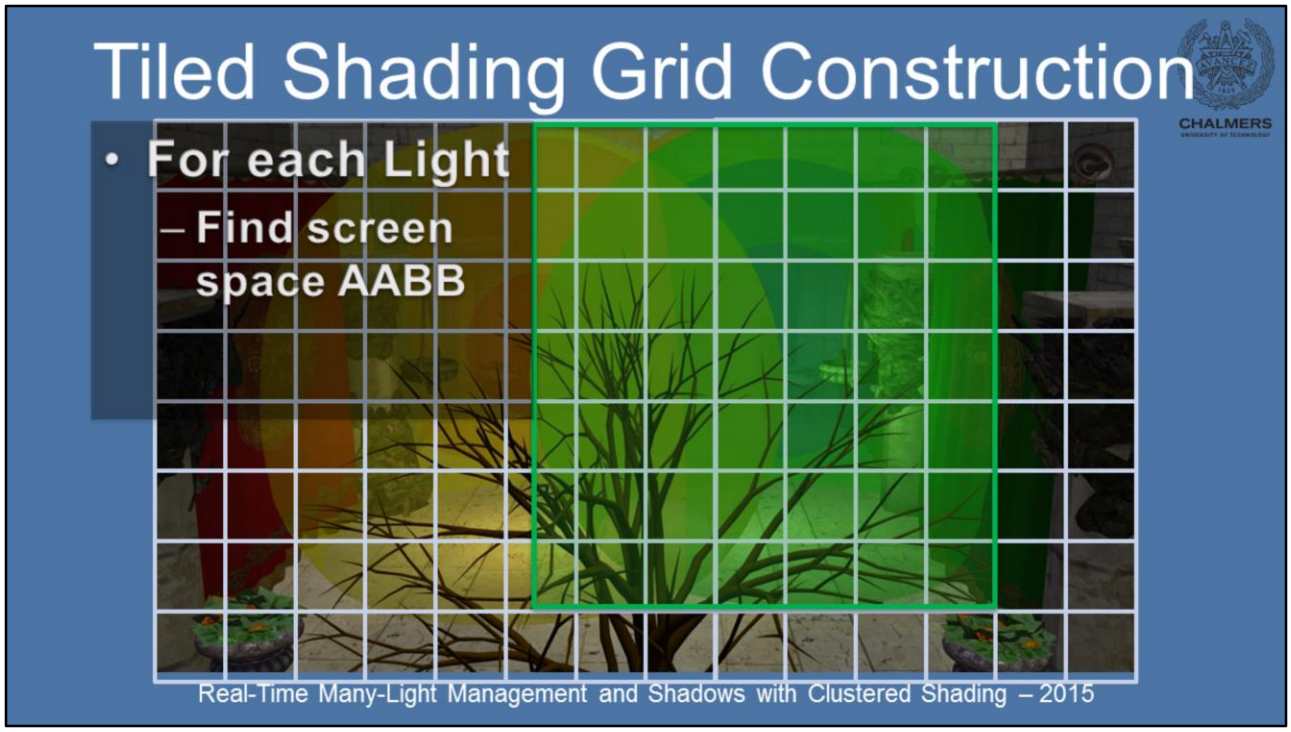
Example Scene

Real-Time Many-Light Management and Shadows with Clustered Shading – 2015

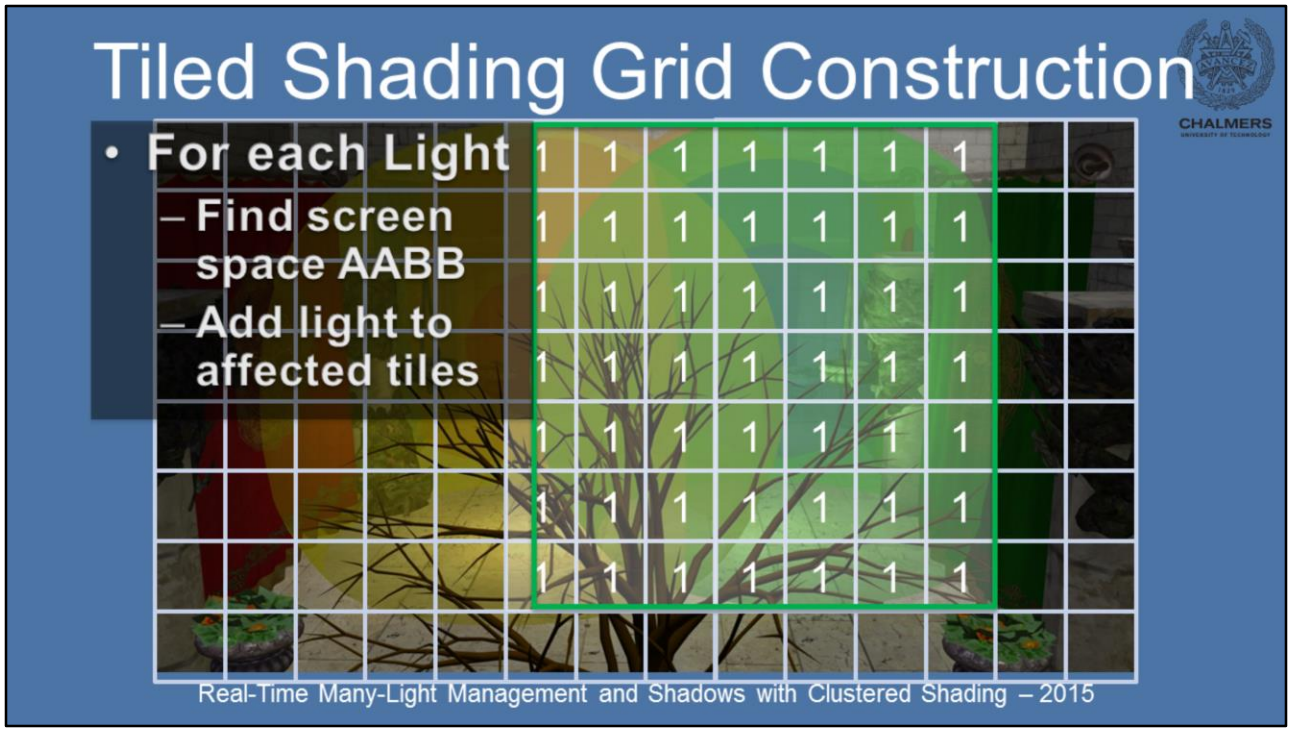
- Shifting to the point of view of the yellow camera.



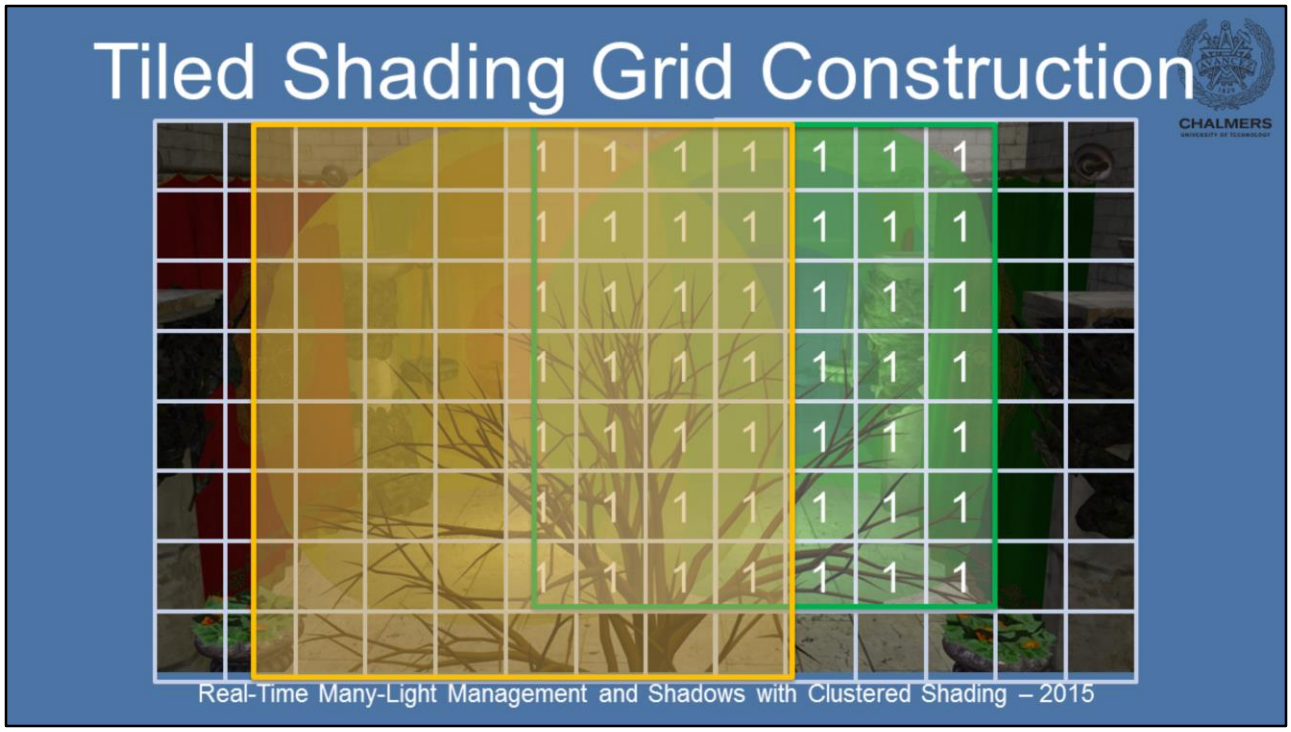
- The screen is divided into tiles, each covering say 32 by 32 pixels.
- Each tile contains a single list of all the lights that might influence any of the pixels inside.
- Note that this list is shared between the pixels, so overhead for list maintenance and fetching is low.



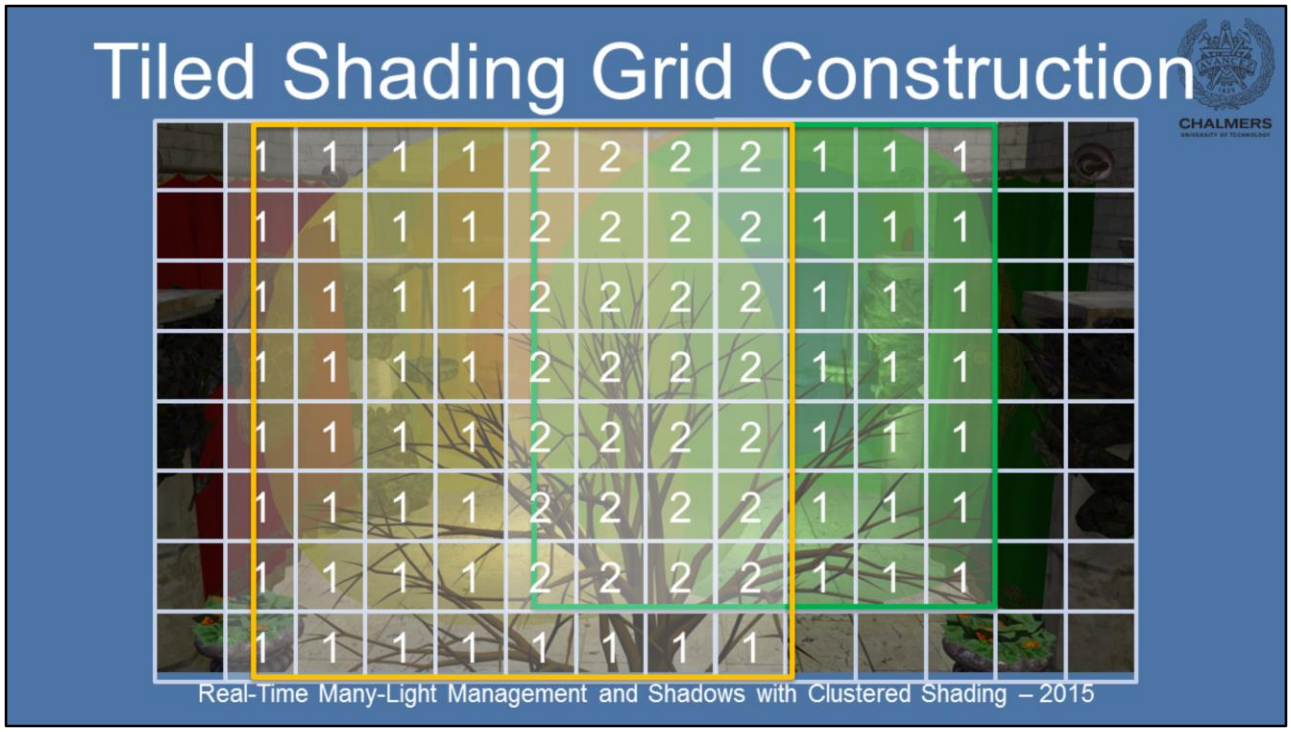
- To construct the lists, we might do as follows.
- For each light, establish the screen space bounding box, illustrated for the green light.



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



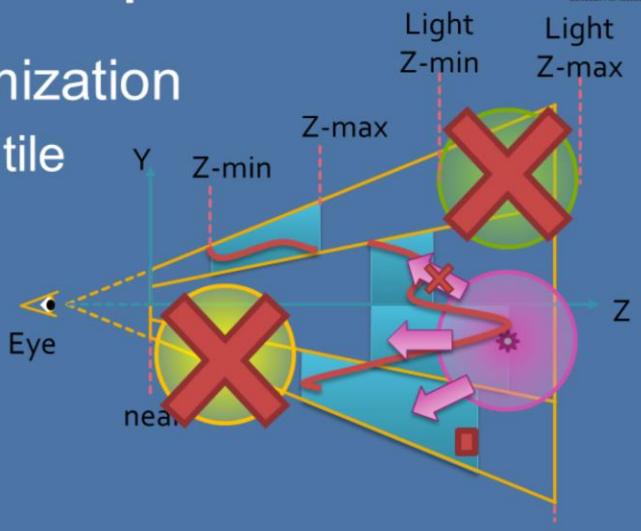
- Then repeat this process for all remaining lights.



- The illustration only shows the counts, so you need to imagine the lists being built as well.

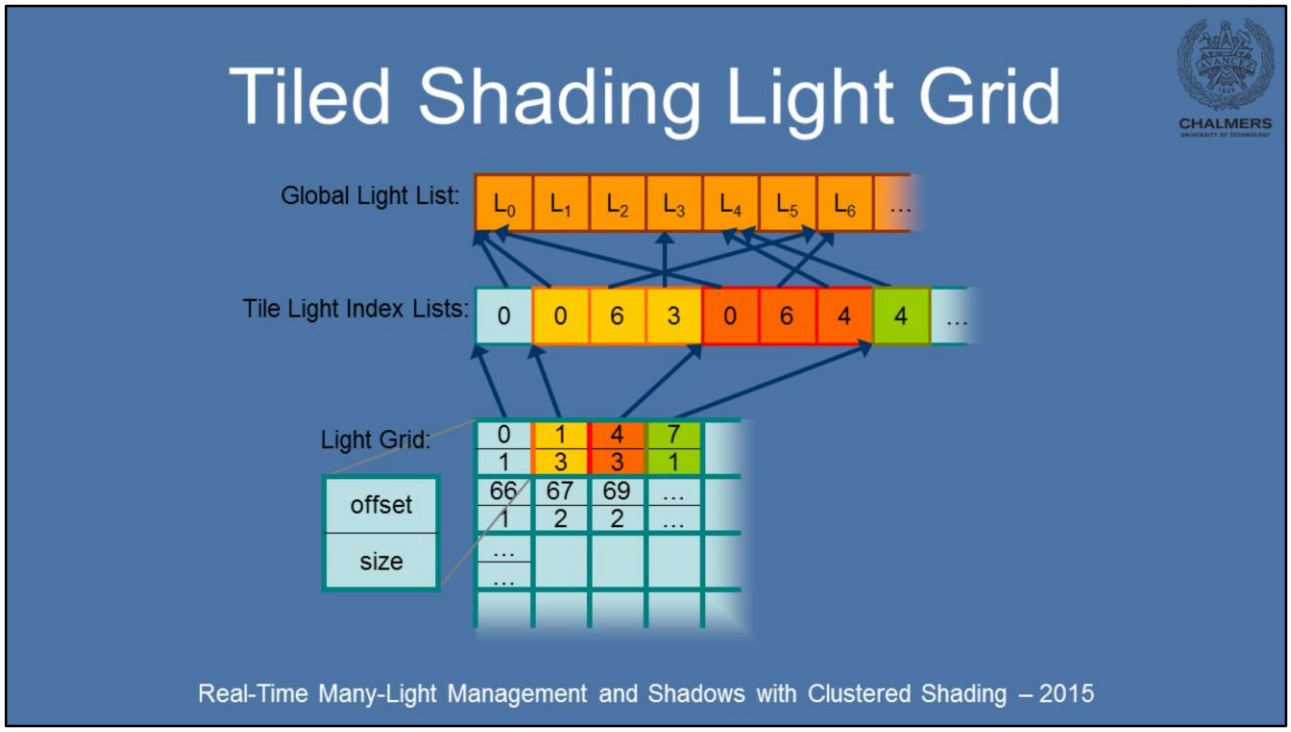
Depth Range Optimization

- Depth Range Optimization
  - Min/Max depth per tile
  - From Pre-z pass
- For each Light
  - Test depth bounds

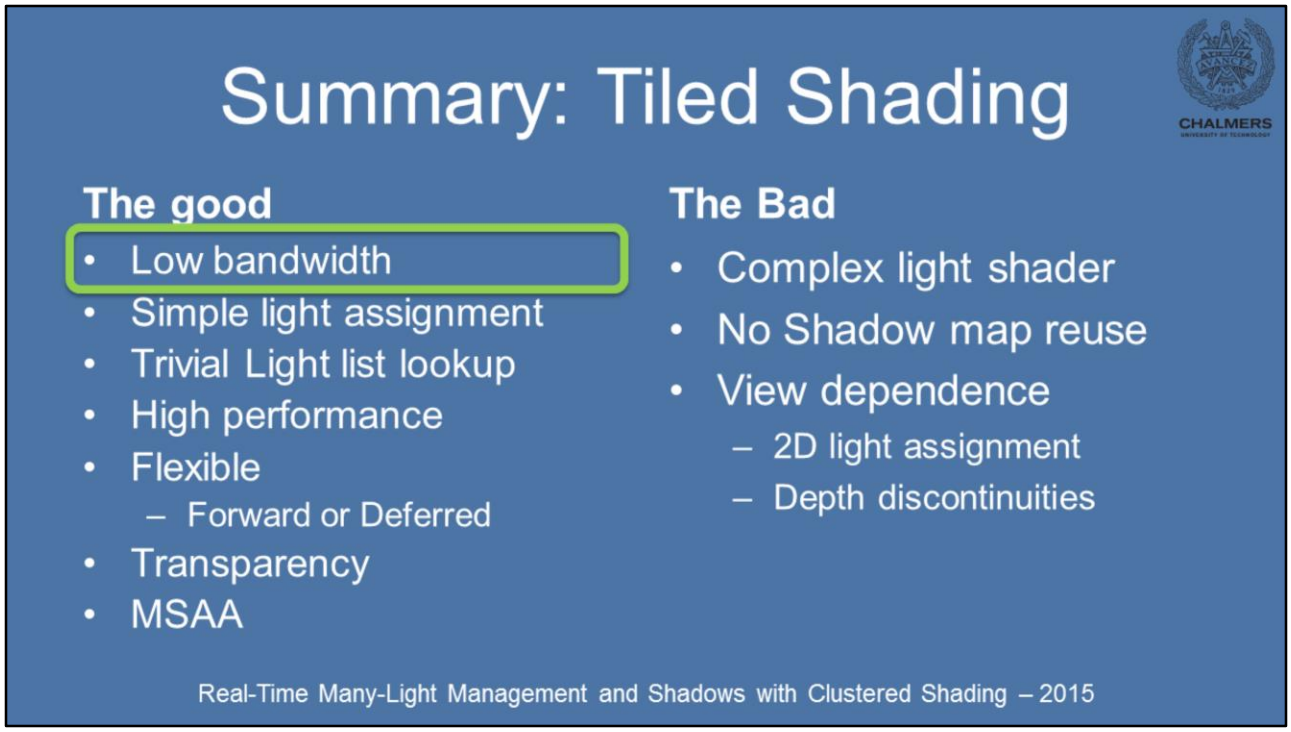


Real-Time Many-Light Management and Shadows with Clustered Shading – 2015

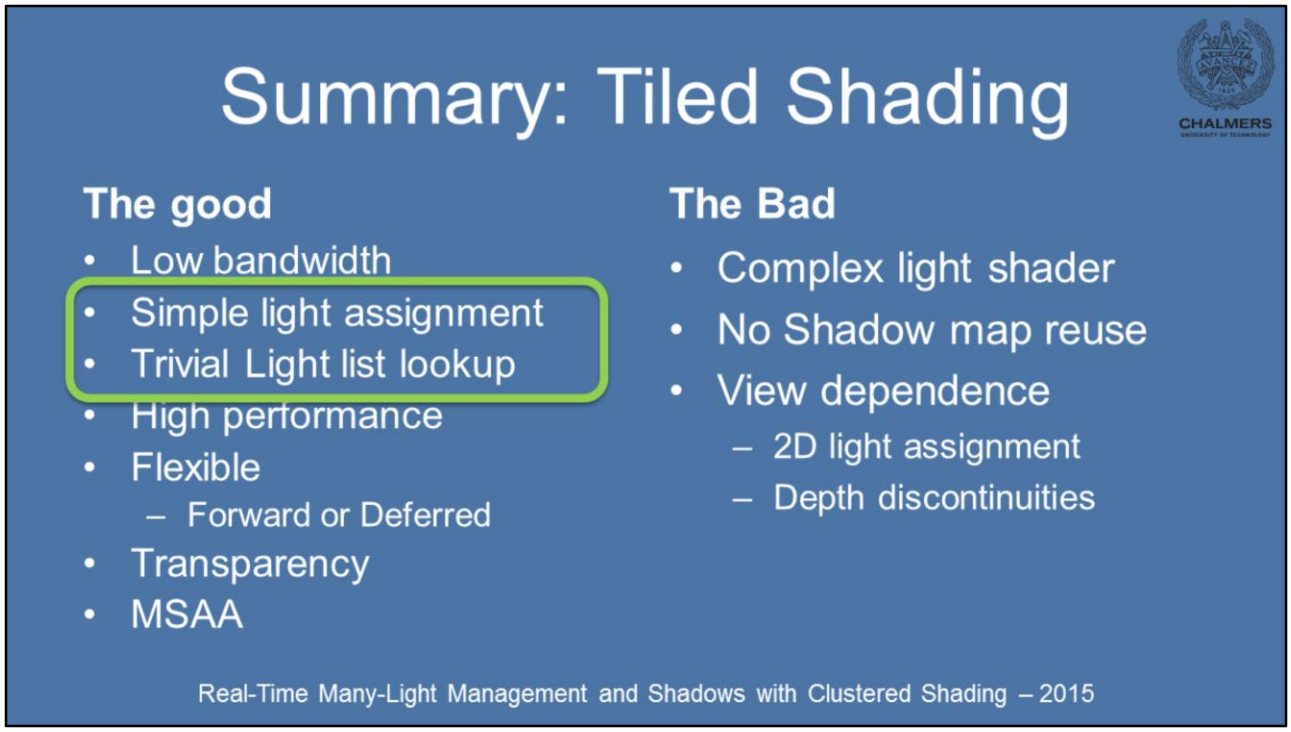
- In practice we'd also do a conservative per-tile min/max depth test, to cull away lights occupying empty space.
  - Just like the stencil optimization for traditional deferred.
- If we don't do this performance generally goes down the toilet…
- and this is what makes a pre-z pass necessary for tiled forward shading.



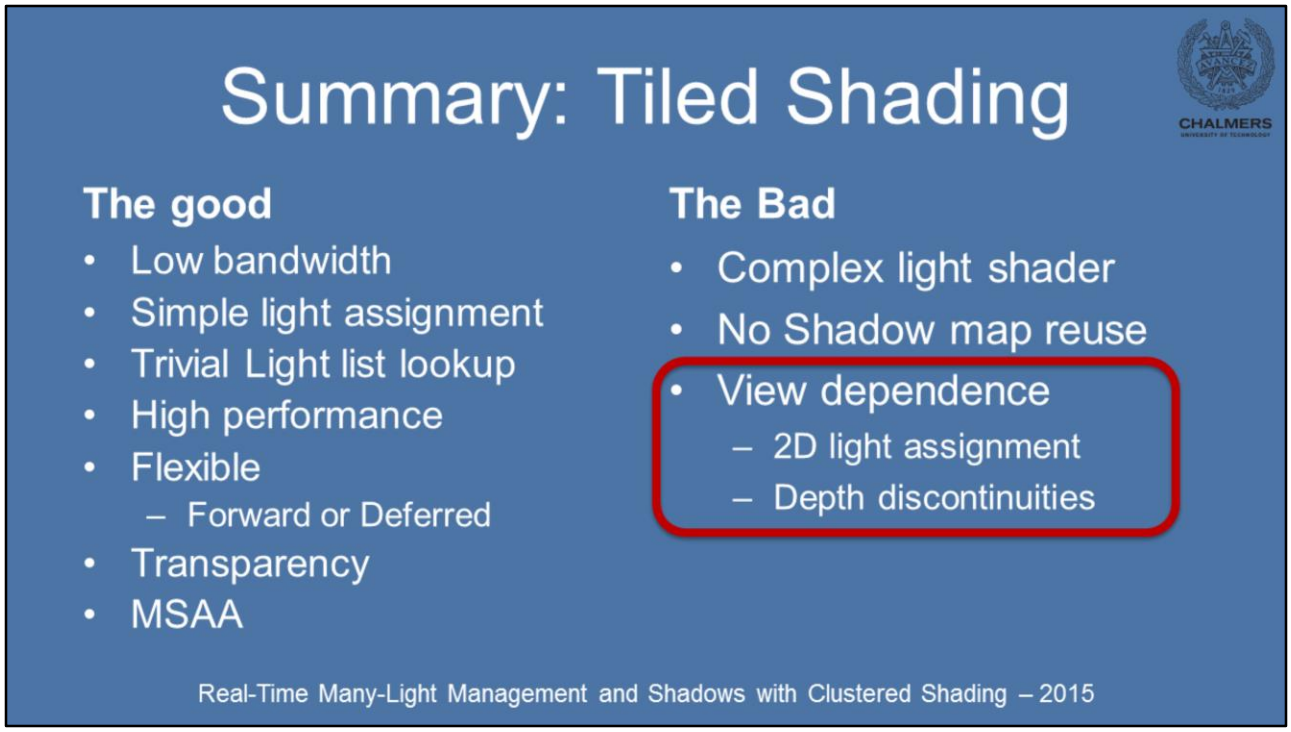
- After we have processed all the lights, we end up with a 2D grid such as this
- This can then be used in the shading pass, as we saw before.



- To reiterate, we've solved the bandwidth problem



- Light assignment is quite simple, just a 2D operation and lookup just depends on screen space location!

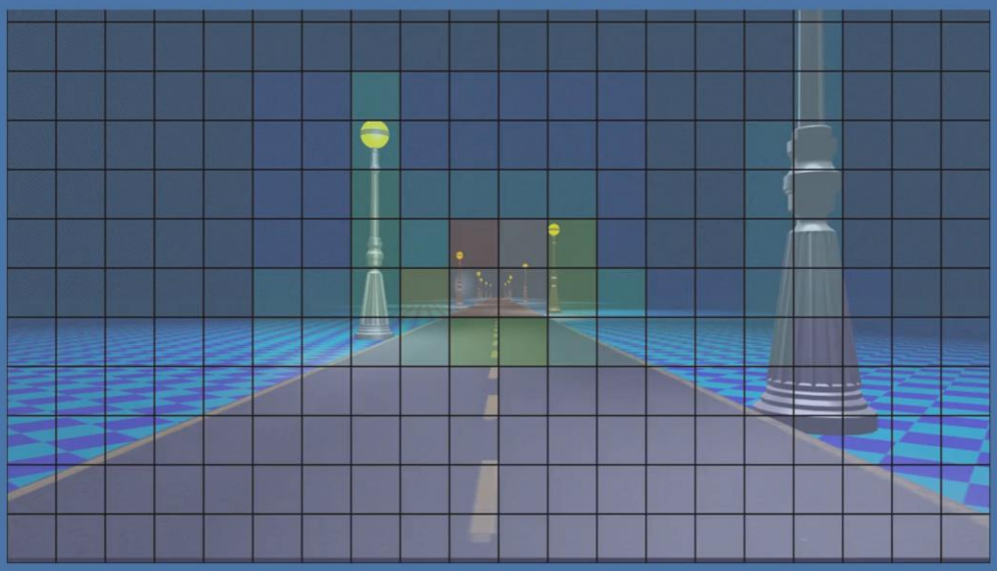


- However, performance is strongly view dependent.
- Which means run time performance is not readily predictable from scene parameters such as light distribution.
- We'll look into this problem in some detail next…

Troublesome Tiles

Real-Time Many-Light Management and Shadows with Clustered Shading – 2015

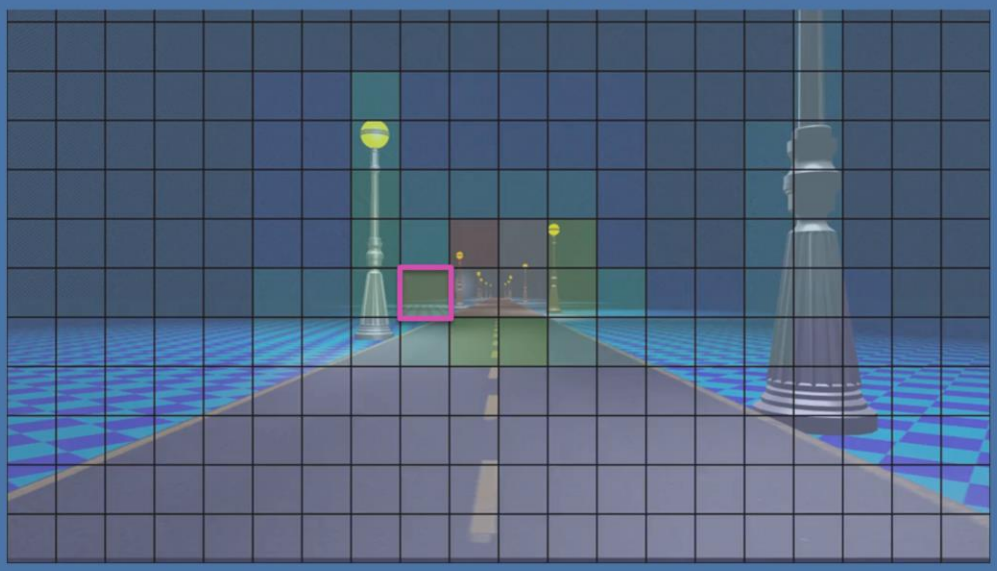
Troublesome Tiles



Real-Time Many-Light Management and Shadows with Clustered Shading – 2015

- So as we can see most of the trouble is around the light posts, where there are depth discontinuities.
- And roads aren't that uncommon in games I think.

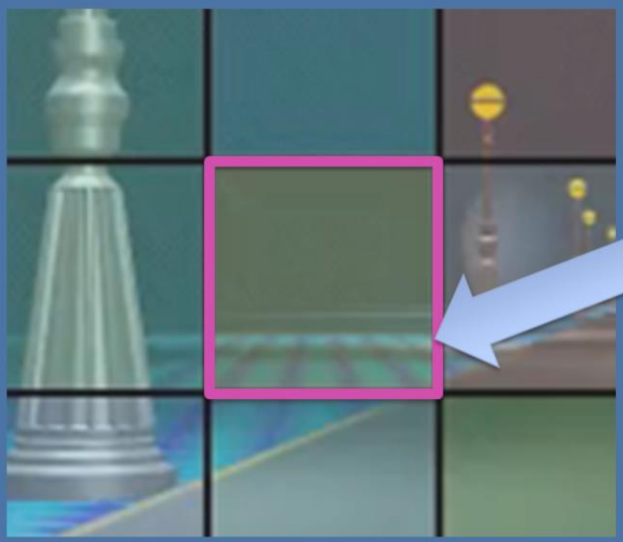
Troublesome Tiles



Real-Time Many-Light Management and Shadows with Clustered Shading – 2015

* But, as if that is not bad enough, lest focus on this tile.

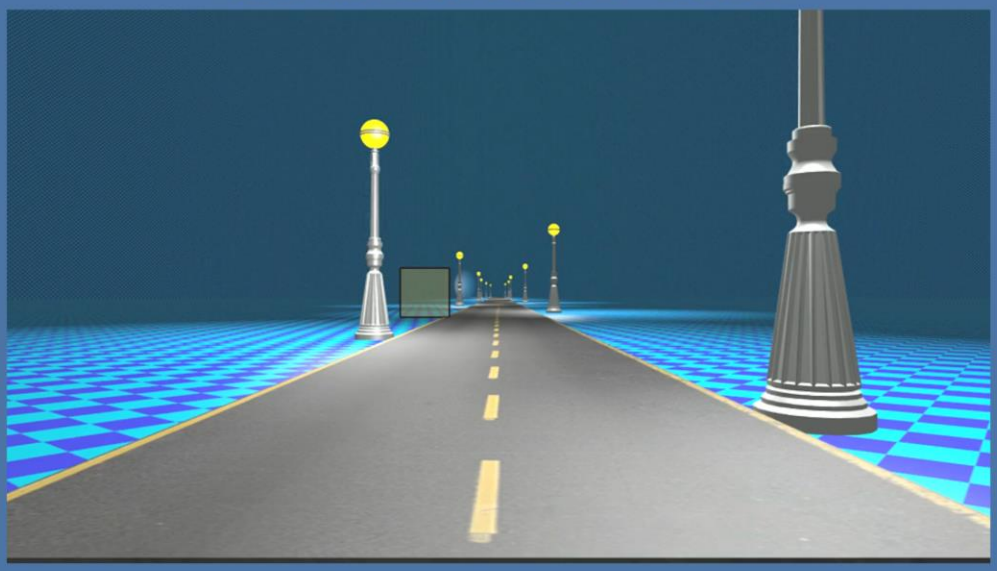
Troublesome Tiles



Real-Time Many-Light Management and Shadows with Clustered Shading – 2015

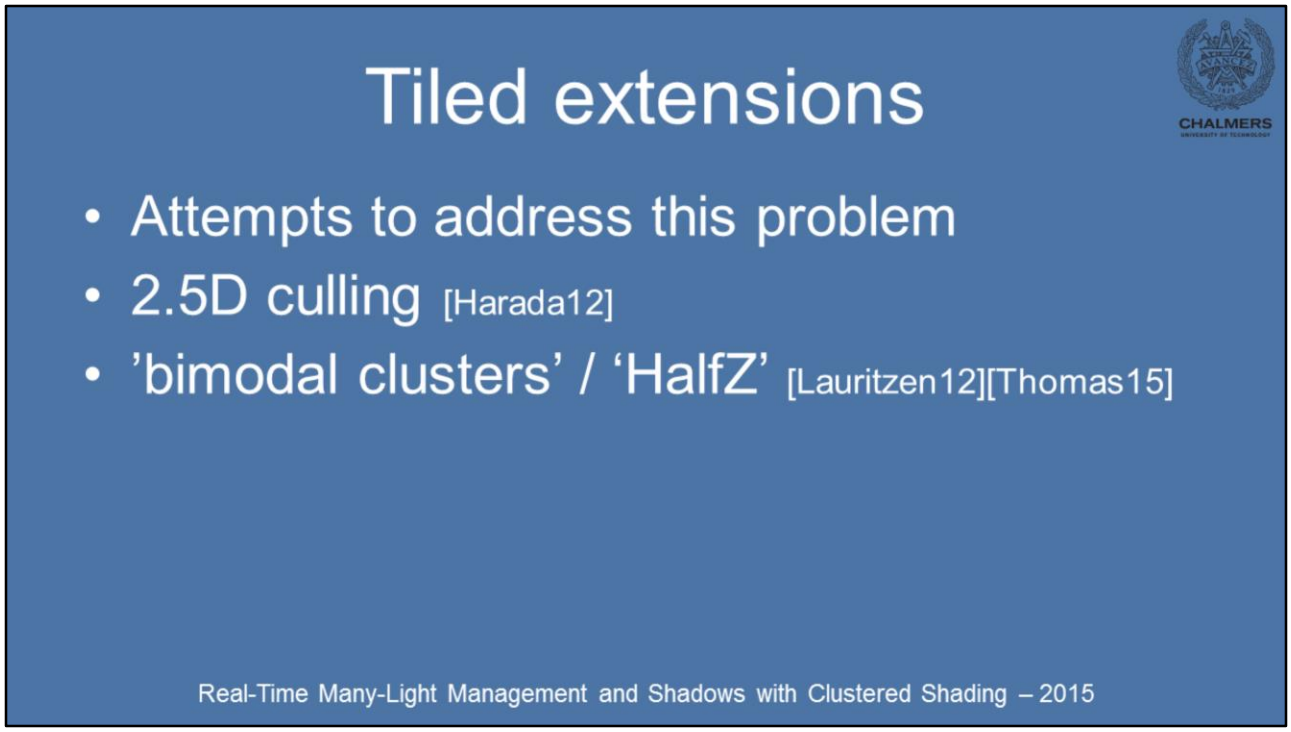
- Zoomed in we see there is only flat ground, and no discontinuities.
- So that should be a nice and good tile, right?

Troublesome Tiles

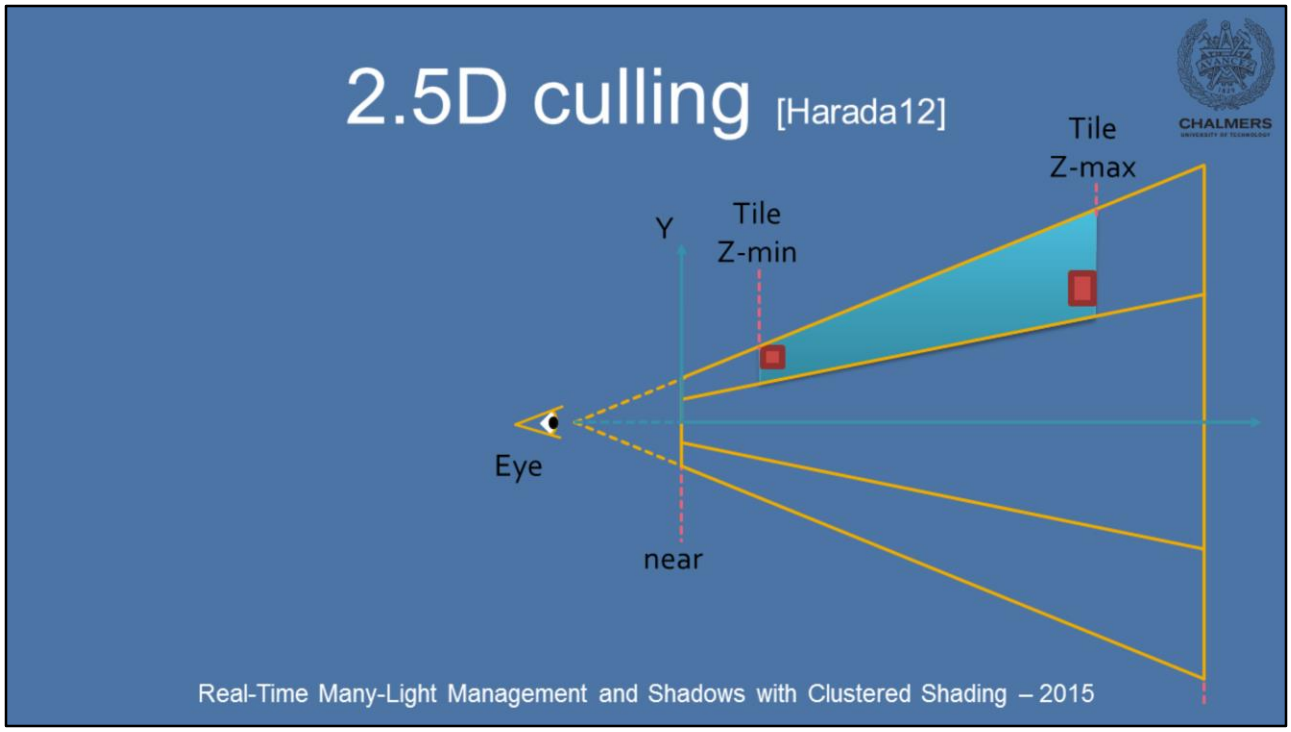


Real-Time Many-Light Management and Shadows with Clustered Shading – 2015

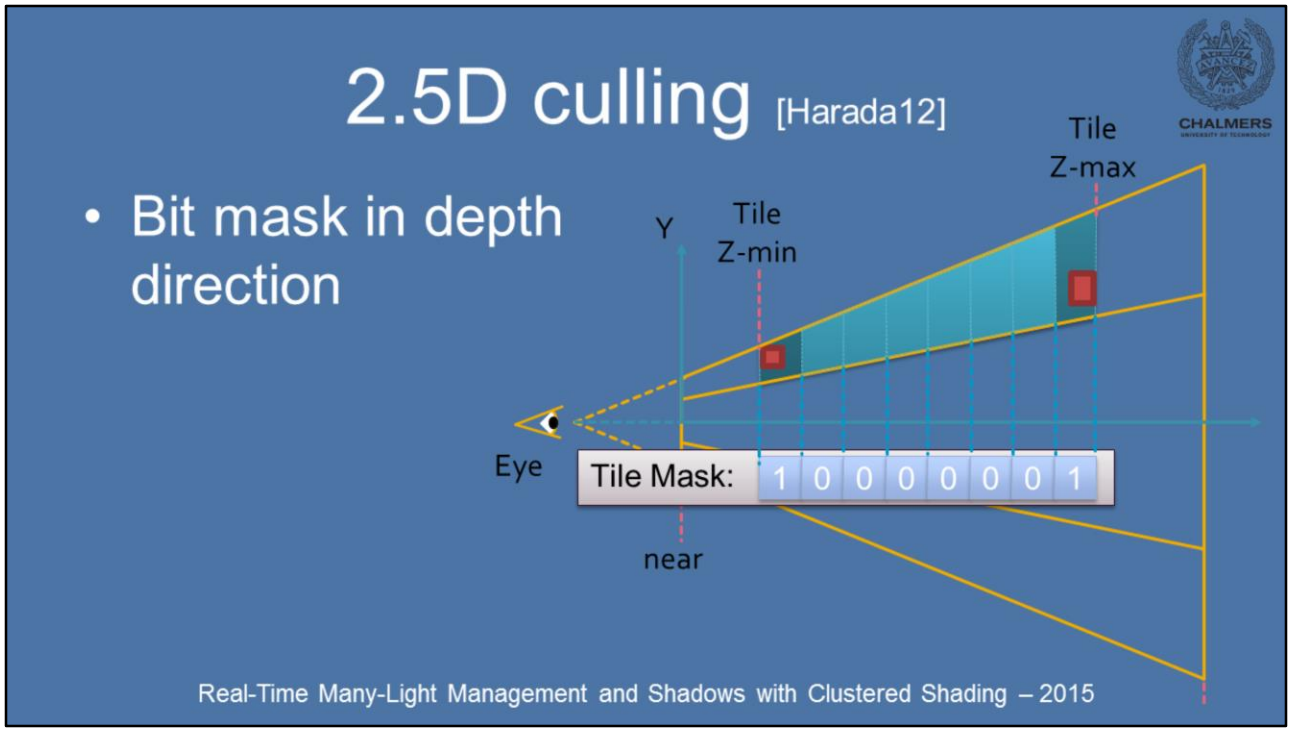
- Well, no! It is pretty darn long.
- And I believe ground is something many games contain, so this might crop up here or there.



- Some techniques that mitigate this problem exists
- I had to cull the slides describing how they work, but you can check out the GDC presentation by Gareth Thomas, where they are explained very well.



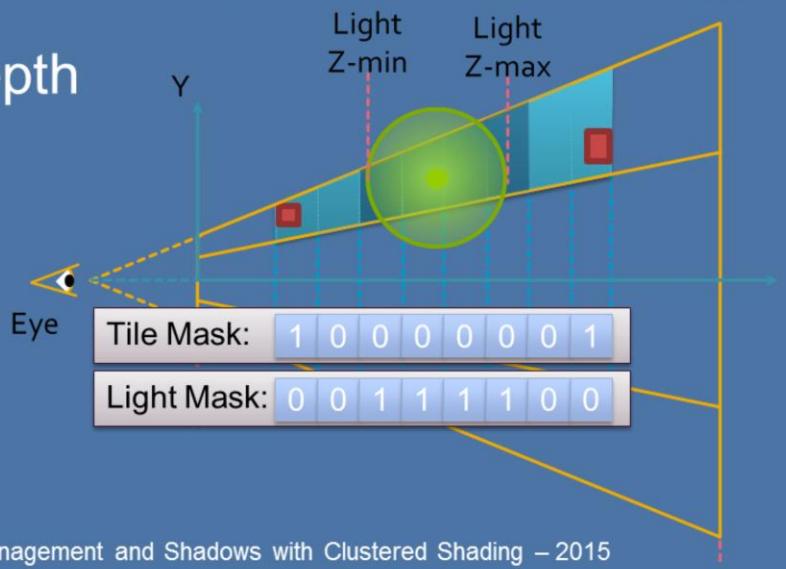
- Some techniques that attempts to address this exists
- One is 2.5D culling, which



- Some techniques that attempts to address this exists
- One is 2.5D culling, which

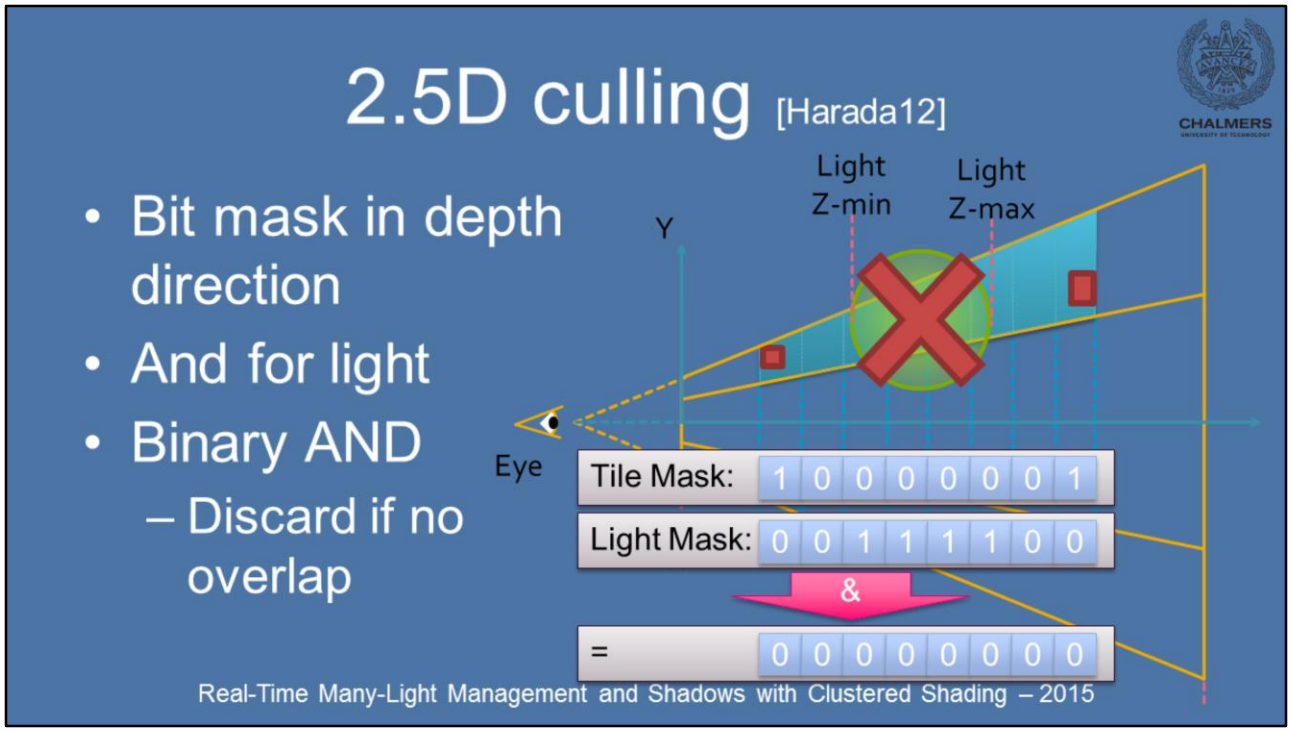
2.5D culling [Harada12]

- Bit mask in depth direction
- And for light

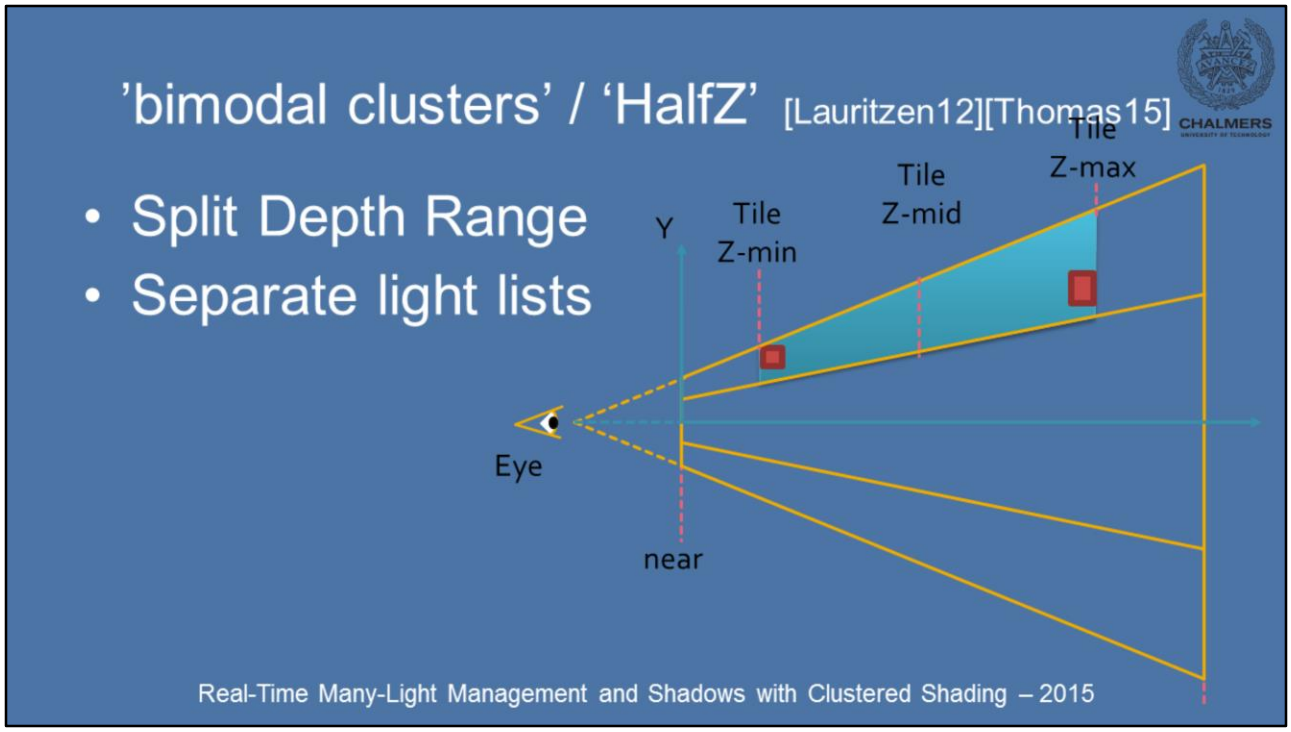


Real-Time Many-Light Management and Shadows with Clustered Shading – 2015

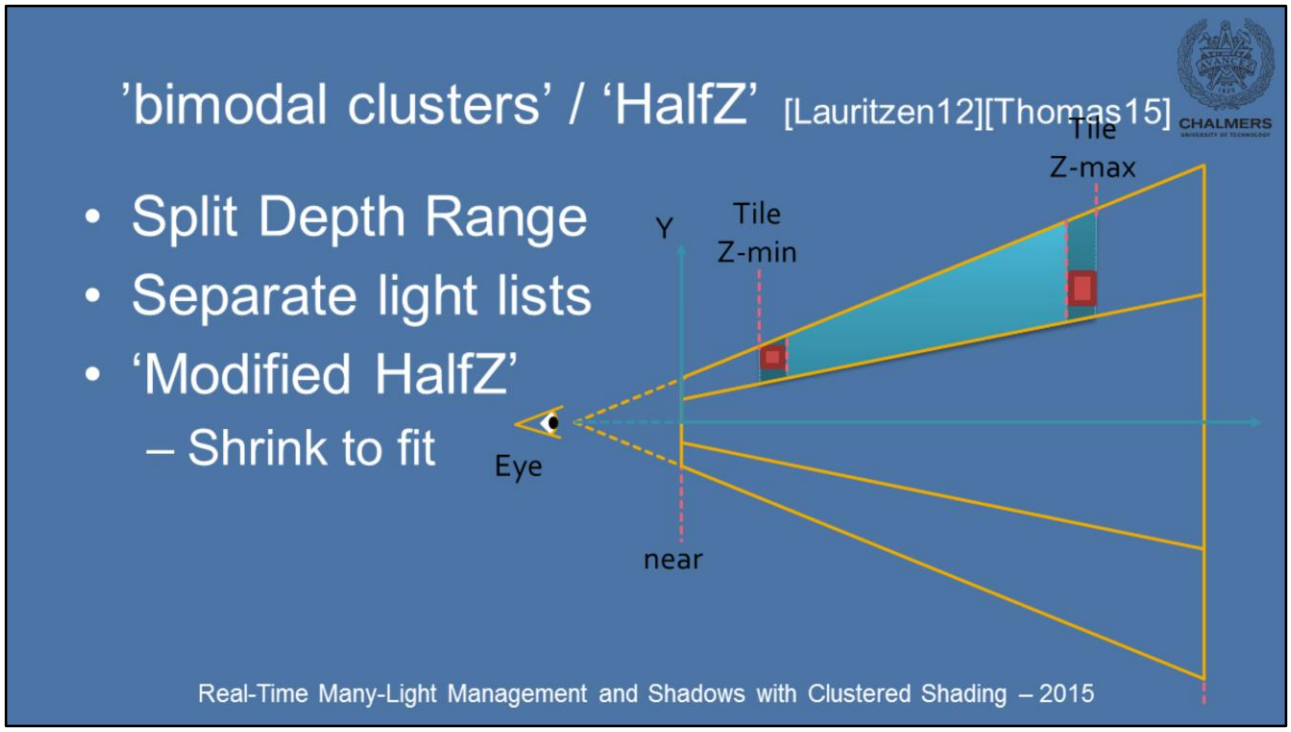
- Some techniques that attempts to address this exists
- One is 2.5D culling, which



- This takes care of some lights overlapping empty space.



- Some techniques that attempts to address this exists
- One is 2.5D culling, which



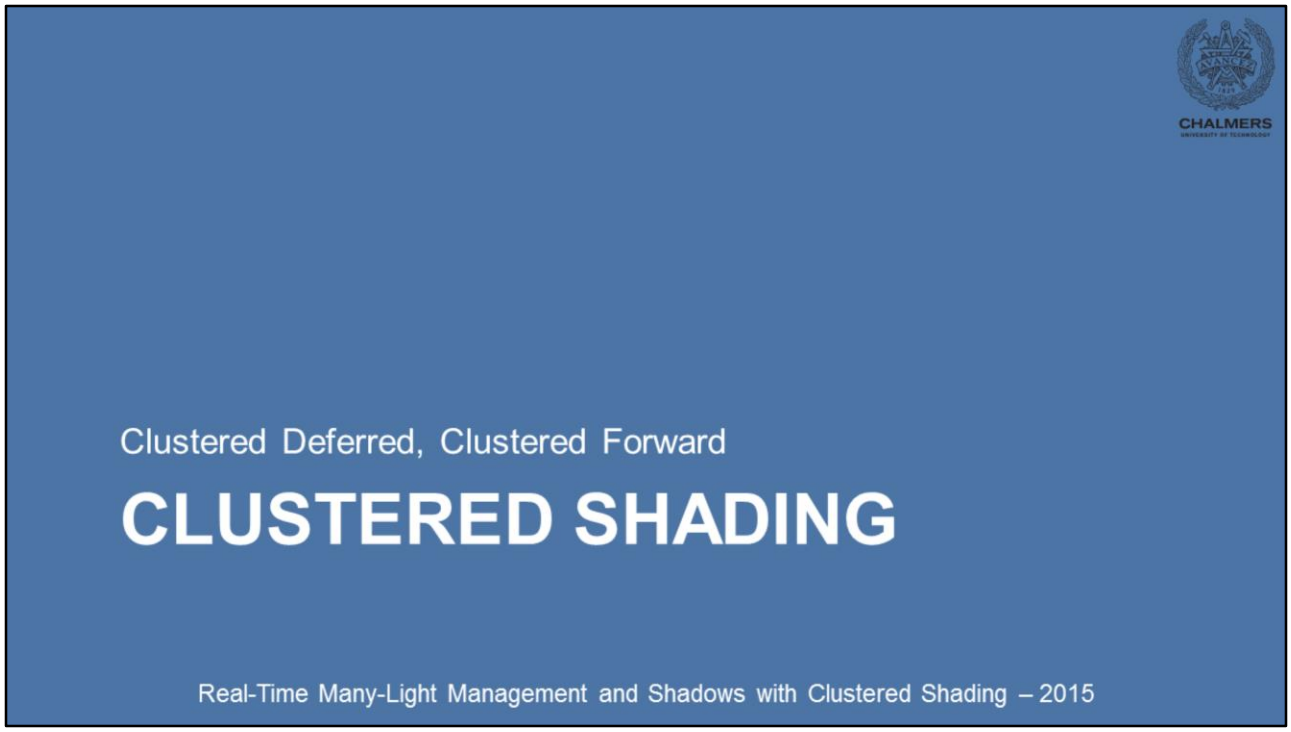
- Some techniques that attempts to address this exists
- One is 2.5D culling, which

Tiled extensions

- Attempts to address this problem
- 2.5D culling [Harada12]
- 'bimodal clusters' / 'HalfZ' [Lauritzen12][Thomas15]
- Problems
  - Lack of generality: Slopes / multiple layers
  - No solution for transparency

Real-Time Many-Light Management and Shadows with Clustered Shading – 2015

- In summary, these techniques lack generality, and don't really handle slopes.
- Also they both require a pre-existing depth buffer, and thus do not support transparency.

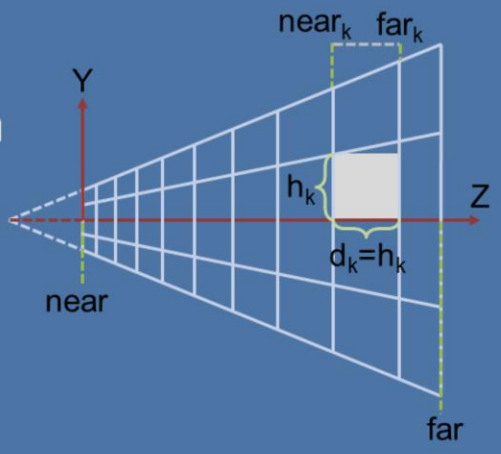


- This brings us to the second of the modern techniques, clustered shading.
- It is squarely aimed at addressing this problem, and making sure it stays solved.

~25 - 28

Clustered Shading – Key Idea

- Add the 3rd dimension
  - Tile also in depth direction
- Cluster
  - Groups *related* samples

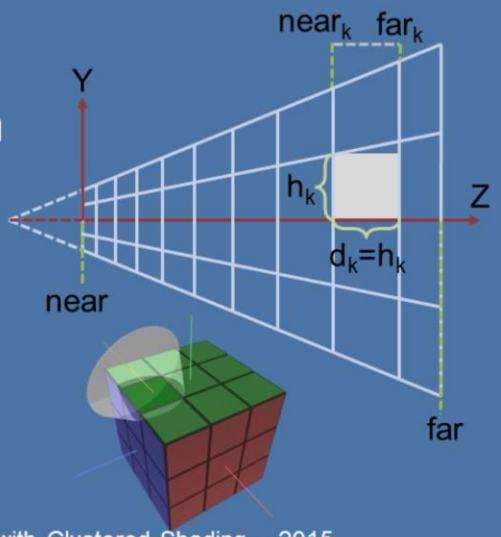


Real-Time Many-Light Management and Shadows with Clustered Shading – 2015

- Clustered shading can be seen as a generalization of tiled shading, and can be said to cover pretty much any way of
- grouping of related sample points.
- For what we're interested in, this relation pretty much boils down to 3D Euclidean distance.
- which means that they are nearby in space.<click>

Clustered Shading – Key Idea

- Add the 3rd dimension
  - Tile also in depth direction
- Cluster
  - Groups *related* samples
  - Also > 3 dimensions (e.g. normals) [Olsson12a]
  - Or other attributes…

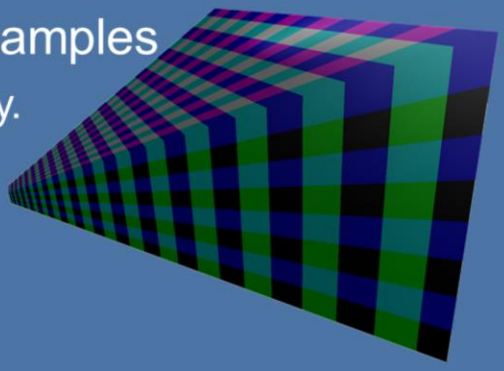


Real-Time Many-Light Management and Shadows with Clustered Shading – 2015

- But we could cluster in yet higher dimensions, for example based on normal direction…<click>
- …or on completely different attributes, like reflectance.
- But here we'll focus on 3D position, and we don't use any fancy k-means clustering algorithm (though that would be interesting)
- But simply bin the view frustum into grid cells – you may have heard the term froxels, this is pretty much the same.

Clustered Shading Properties

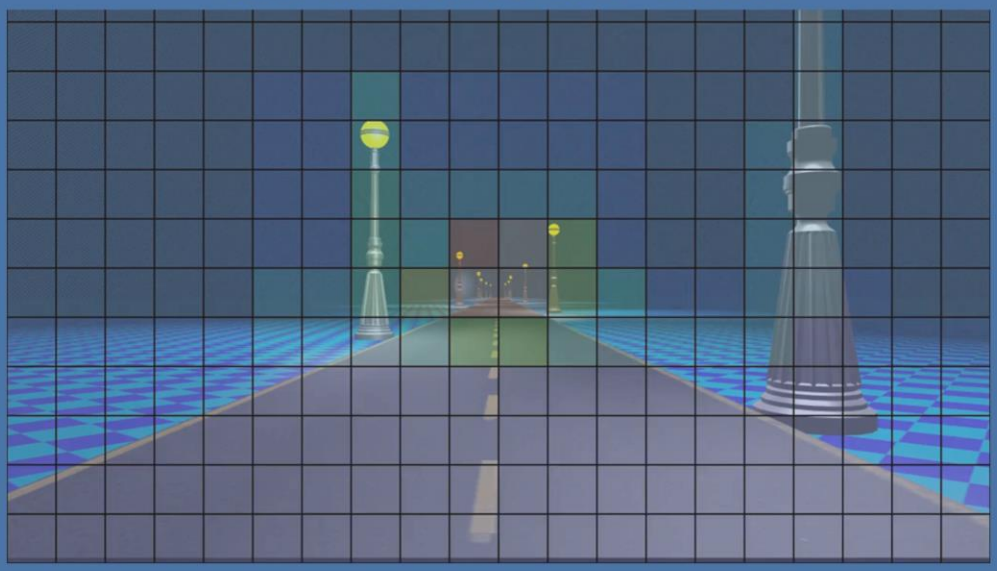
- 3D subdivisions
  - Bounded volume around samples
    - Shading cost ~ Light density.



Real-Time Many-Light Management and Shadows with Clustered Shading – 2015

- So by doing 3D subdivisions we get a much more localized and consistent volume around each group of samples.
- Which enables a correlation between light density in the scene and the run time shading cost.
- This is important for control and predictability of performance.
- Lets take a look at how it works out using the road example from before.

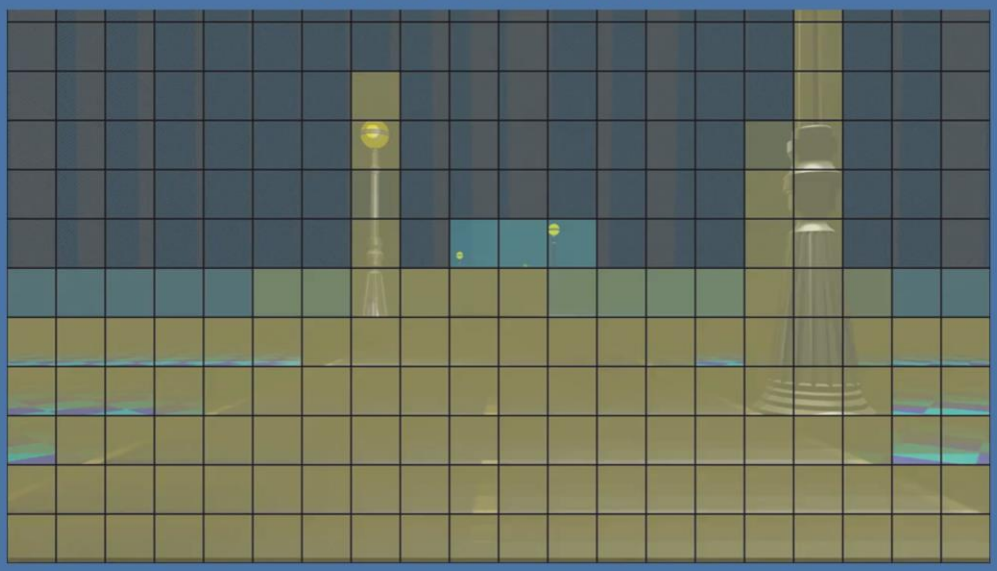
Troublesome Tiles



Real-Time Many-Light Management and Shadows with Clustered Shading – 2015

- Recall the tiles.

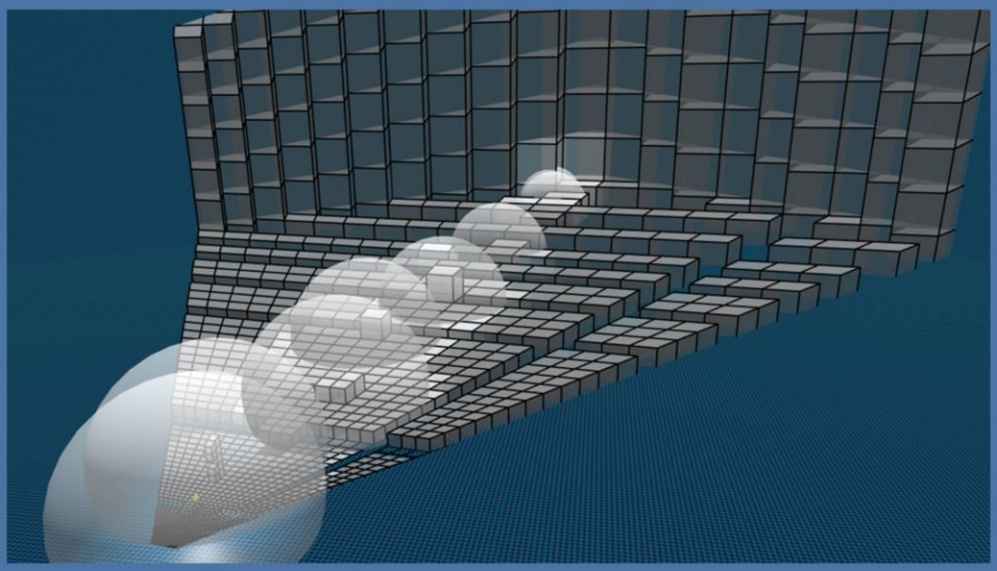
Nice Clusters



Real-Time Many-Light Management and Shadows with Clustered Shading – 2015

- And here the same view, but with clusters.
- As we can see, the clusters will overlap only a few lights each
- And follow the scene geometry well.

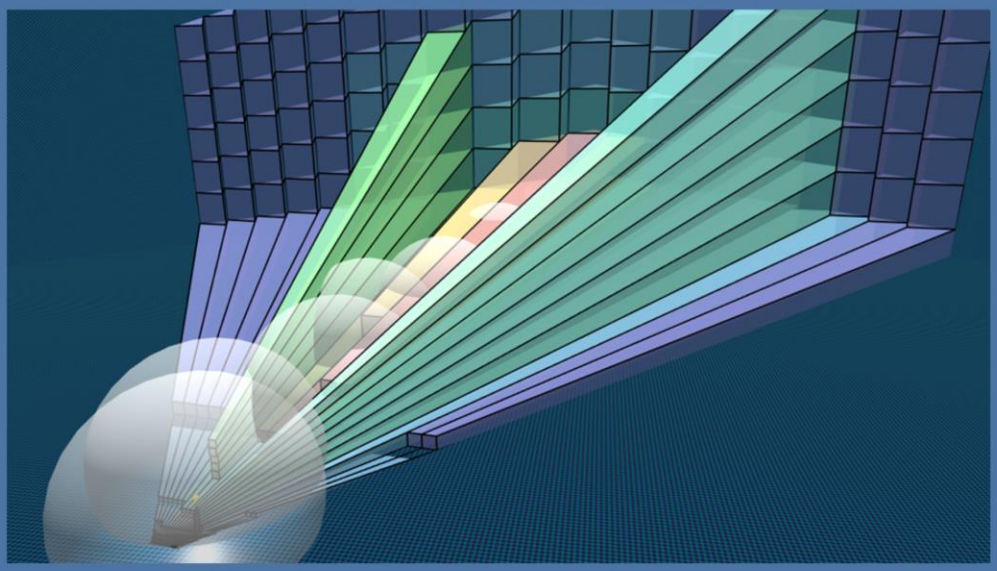
Nice Clusters



Real-Time Many-Light Management and Shadows with Clustered Shading – 2015

- And here the same view, but with clusters.
- As we can see, the clusters will overlap only a few lights each
- And follow the scene geometry well.

Troublesome Tiles

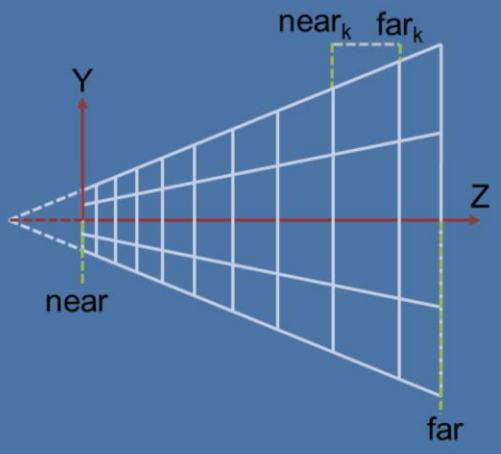


Real-Time Many-Light Management and Shadows with Clustered Shading – 2015

- Going back to the tiles, it looks like this…

Cluster key / index

- Cluster key:
  - ck = (i, j, k) integer tuple
- i, j = 2D tile id
  - gl_FragCoord.xy
- k = ≈log(view space z)

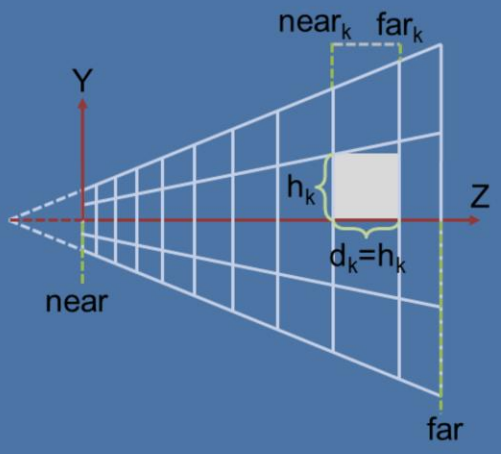


Real-Time Many-Light Management and Shadows with Clustered Shading – 2015

- So, a bit more detail on how we actually build our clusters?
- The original scheme from the paper, though you will see a couple of modifications later, is to use regular subdivision starting from 2D tiles,
- which provides the i and j coordinates.
- While k is a logarithmic function of the view space Z of the sample, not simply the logarithm, for the exact equation see the paper.
- This defines a 3D grid within the view volume. <click>

Cluster key / index

- Cluster key:
  - ck = (i, j, k) integer tuple
- i, j = 2D tile id
  - gl_FragCoord.xy
- k = ≈log(view space z)

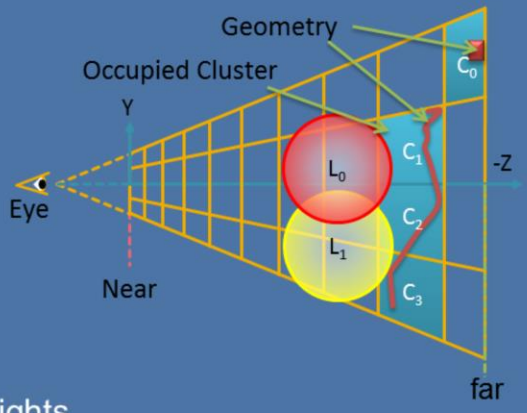


Real-Time Many-Light Management and Shadows with Clustered Shading – 2015

- The logarithmic subdivision creates self similar clusters that are as cube like as possible.
- This means that as clusters become larger further away, we get a kind of LOD behavior and do not end up with insane numbers of clusters far away, for a wide range of view parameters.

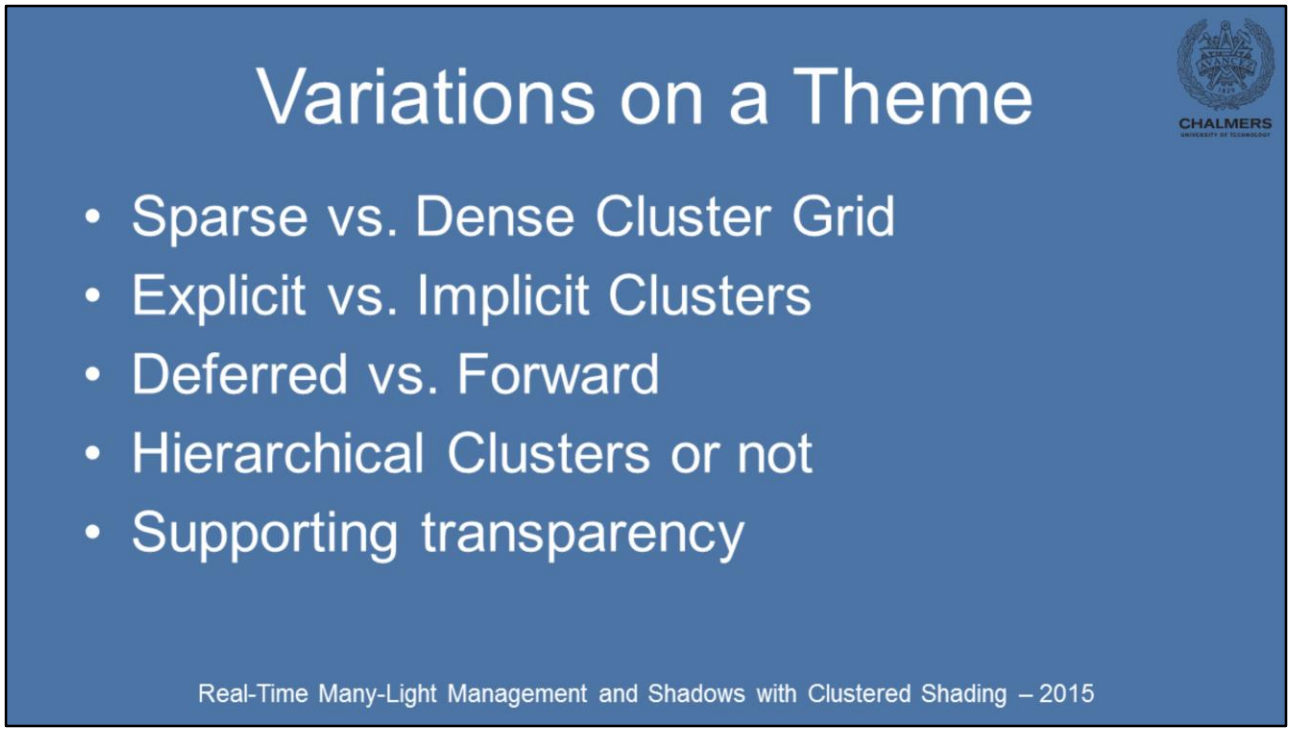
Light Assignment

- Clusters
  - Proxy for visible geometry
    - Bounds view samples
      - 3D boxes
    - Orders of magnitudes fewer
- Lights
  - Bounding spheres/cones
  - Cluster/Light Pairs
    - Links overlapping clusters and lights



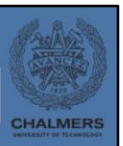
Real-Time Many-Light Management and Shadows with Clustered Shading – 2015

- Light assignment, just like for tiled shading, is finding the overlap between lights and clusters…
- and recording this in a list per cluster.
- We will cover some concrete approaches to this later in the course.



- So that was the super high level overview of clustered shading.
- Now lets try to look at some of the important design decisions you can pick when implementing it.

Sparse vs. Dense Cluster Grid



- Sparse Cluster Grid
  - Only store cells that contain samples
  - Requires Pre(-Z) pass / Deferred
  - No redundant light assignment
  - Geometry info useful for other things!
- Dense Cluster Grid
  - Must assign lights to all clusters
  - Can be done on CPU / Asynchronously
  - Can access any point in view volume
  - SAME shading cost as for sparse!

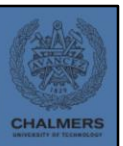
Real-Time Many-Light Management and Shadows with Clustered Shading – 2015

Explicit vs. Implicit Clusters

- Explicit cluster bounds
  - Actual bounding box of samples
  - Some storage
  - Some cost to build
  - Tight bounds
  - Extra geometry pass for forward shading
- Implicit cluster bounds
  - Implied by grid coordinate
  - No storage
  - Can have large empty space.

Real-Time Many-Light Management and Shadows with Clustered Shading – 2015

Deferred vs. Forward Shading



- No fundamental decision
  - Easy to change
- Deferred
  - Easier to support sparse grid
- Forward
  - MSAA, Transparency
  - Vary shading model
- Both
  - Deferred for opaque
  - Forward for transparent
  - Same or separate grids

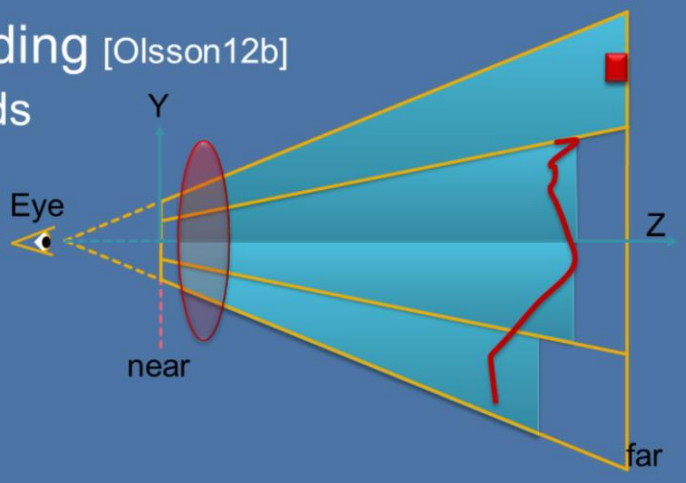
Real-Time Many-Light Management and Shadows with Clustered Shading – 2015

Cluster Hierarchy

- Cheap to build [Sintorn14]
- Hierarchical representation of visible samples.
- Enables novel techniques
  - Efficient Shadow Volumes
  - Shadow maps (Talk #3)
  - Splatting in general.

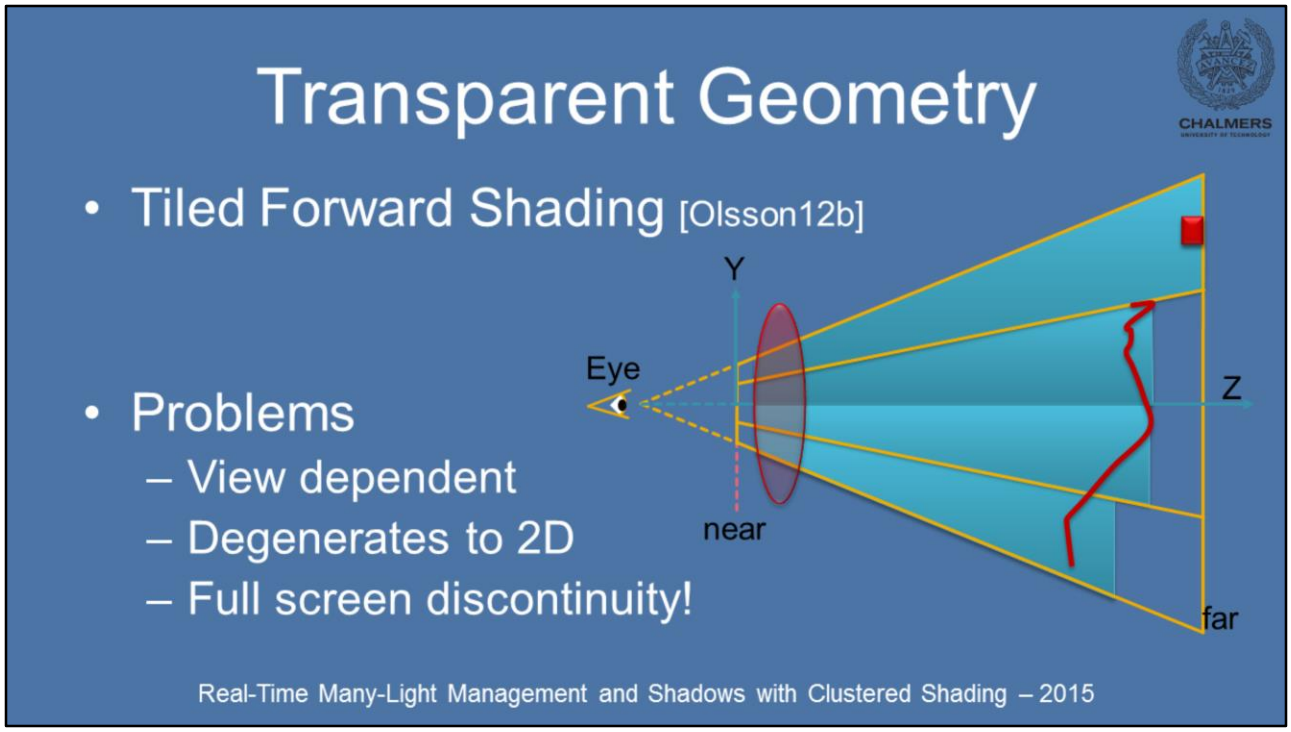
Transparent Geometry

- Tiled Forward Shading [Olsson12b]
  - Extend tile Z bounds
    - To Min Z
    - Or Near plane



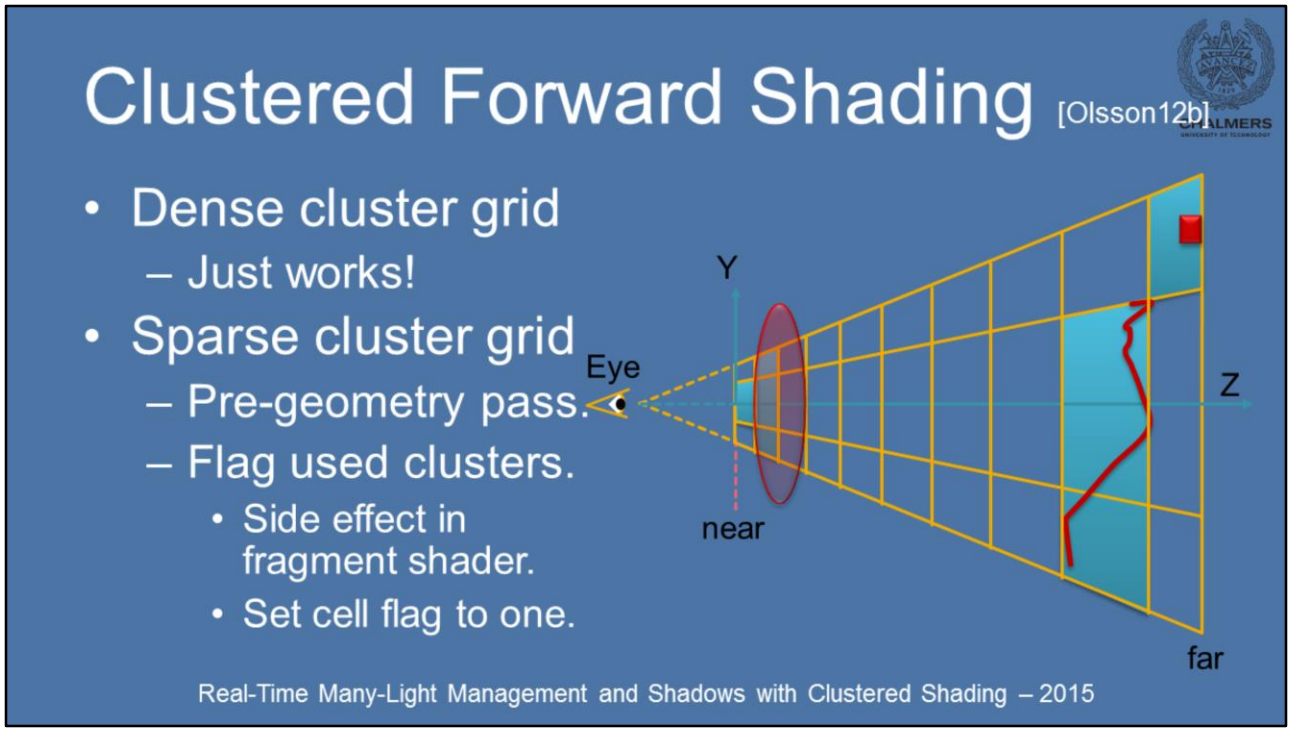
Real-Time Many-Light Management and Shadows with Clustered Shading – 2015

- 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…



- 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 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 need 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.

End!

Real-Time Many-Light Management and Shadows with Clustered Shading – 2015

Technique Comparison

	Trad. Deferred	Tiled Deferred	Tiled Forward	Clustered Deferred	Clustered Forward
Bandwidth Use	High	Low	Low	Low	Low
Transparency	No	With Fwd →	Yes	With Fwd →	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

Footnotes:

- *: Depends on whether pre-z pass is used, which is pretty much require if the scene has any depth complexity
- +: 1 Pass is generally fine if a dense grid is used. Then a pre-z pass is only used to prime depth buffer for overdraw handling.
- #: If the stencil buffer optimization is NOT used (and perhaps it is not practical today) then view dependence might be quite high because of light overdraw!

References

[Persson13] Practical Clustered Deferred and Forward Shading, Persson & Olsson 2013, http://advances.realtimerendering.com/s2013

[Olsson12a] Clustered Deferred and Forward Shading, Olsson et.al. 2012

[Olsson12b] 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. JGT, 8(3):23–32, 2003.

[Harada12] A 2.5D culling for forward+. In SIGGRAPH Asia 2012 Technical Briefs, ACM, 18:1–18:4.

[Lauritzen12] Andrew Lauritzen, Intersecting Lights with Pixels, In SIGGRAPH 2012 Courses, Beyond Programmable Shading,

[Ferrier11] Alex Ferrier and Christina Coffin, Deferred shading techniques using frostbite in "Battlefield 3" and "Need for Speed the Run", SIGGRAPH 2011 Talks

[Sintorn14] Erik Sintorn , Viktor Kämpe , Ola Olsson and Ulf Assarsson, Per-Triangle Shadow Volumes Using a View-Sample Cluster Hierarchy, I3D 2014

[Trebilco07] Damian Trebilco, 2007 (Shader X7 2009), GitHub: https://github.com/dtrebilco/lightindexed-deferredrender

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