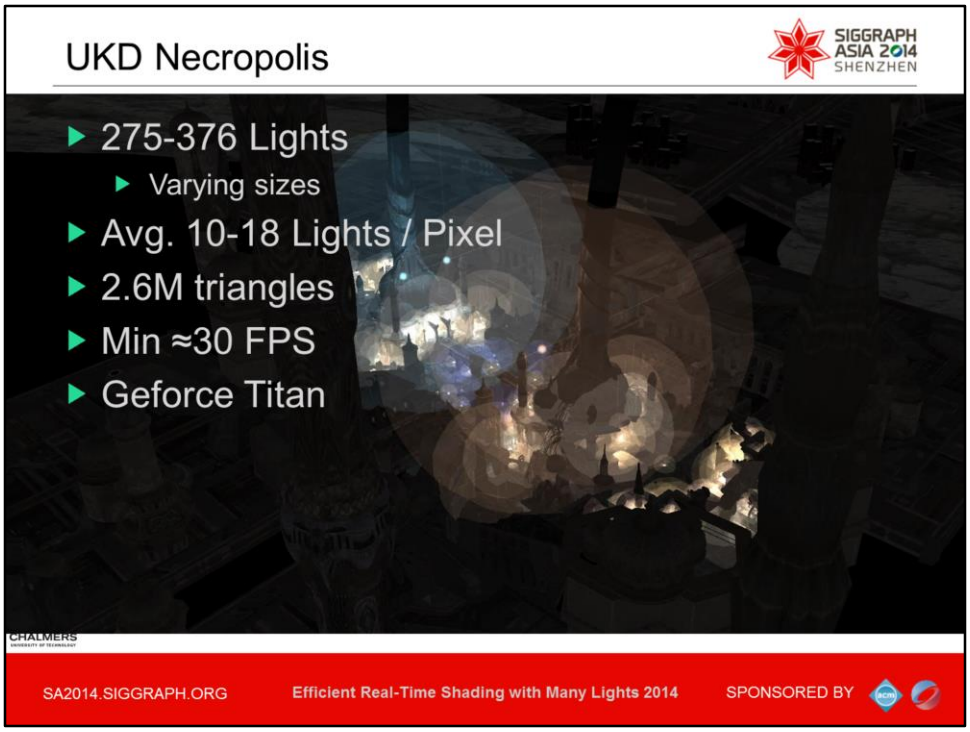
Shadows for Many Lights

- Part 3 – 40 Minutes
- Presenter: Ola Olsson

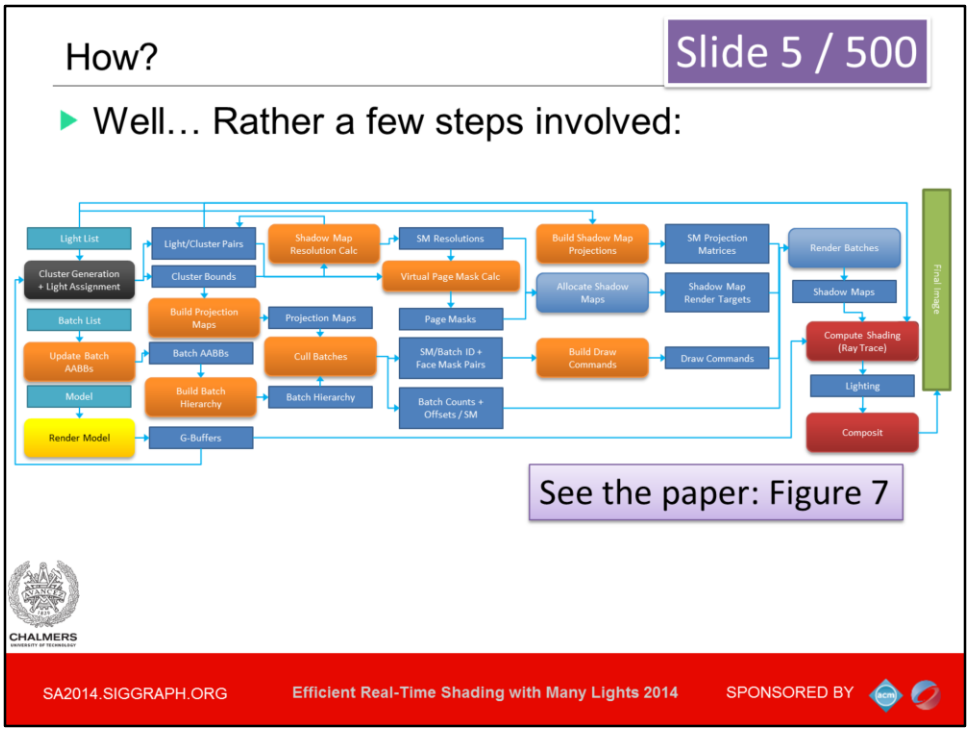
Shadows for many lights

- Paper from I3D 2014
  - "Efficient Virtual Shadow Maps for Many Lights".
- Builds on Clustered Shading
  - Efficient real-time many-light algorithm
- Adds support for shadows
  - Hundreds in, real-time
    - Well, research-real-time at least.
  - High & Consistent quality
  - Efficient culling
  - Virtual/Sparse texture use case
- Novel results (Unpublished)
  - Improved performance.

- This talk is about the techniques presented in my paper at I3D earlier this year…
- …titled Efficient Virtual Shadow Maps for Many Lights.
- This paper in turn builds on clustered shading,
- which is an efficient and robust real-time algorithm for many lights,
- and adds support for shadows from hundreds of **omnidirectional** lights in real time.
- What is novel about this apart from the speed is that we achieve a high and uniform quality across all lights.
- And the scheme we introduce for culling the geometry to reduce the triangle rate.

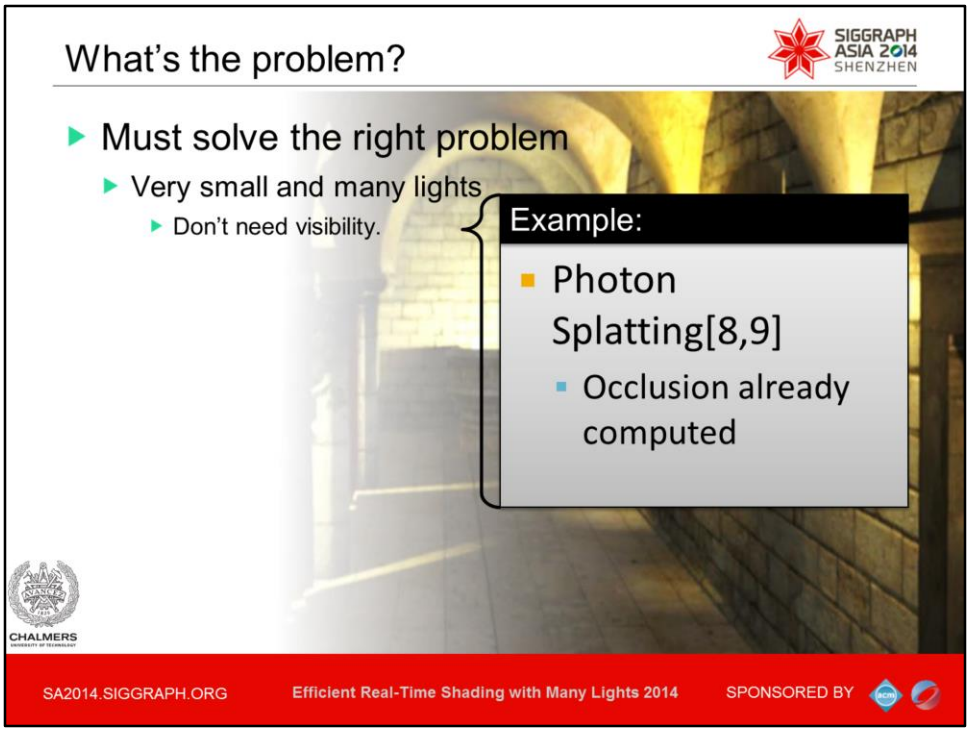


- To show what we achieve, here are some results from the paper
- This scene contains almost 400 shadow casting, omnidirectional, point lights,
- Some very large, most smaller.
- All lights and geometry is treated as dynamic.
- Even with almost 20 lights per pixel on average, we achieve…
- …a minimum of around 30 fps on a Titan GPU.

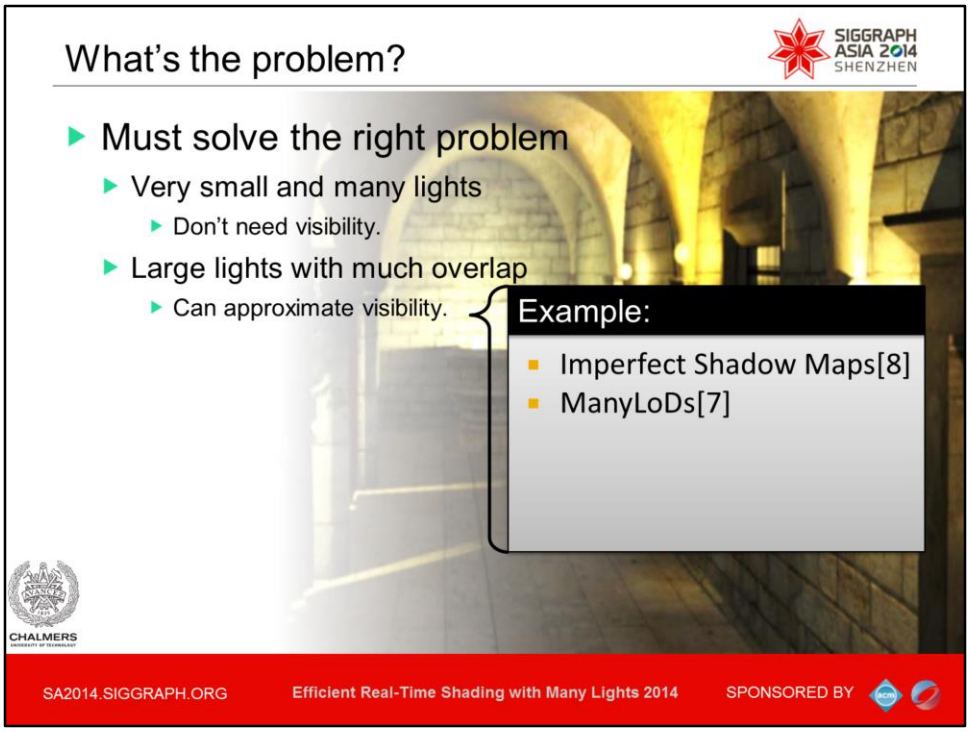


- So how do we achieve this?
- This chart is in the paper, and shows the, rather many, stages and data flow in our system.
- Even in 40 minutes, I cannot hope to cover this in detail, but I will cover the main points.

- When approaching this problem it is necessary to limit the problem to be able to design an efficient solution.
- As there exists a limitless number of possible variations that is more or less suitable depending on the target application.



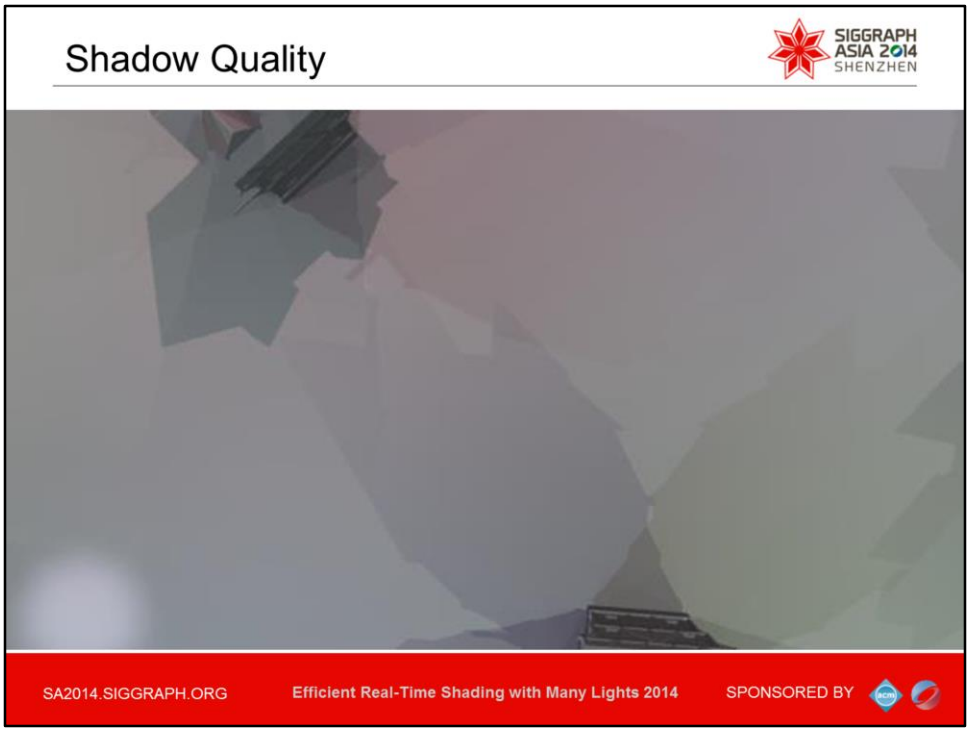
- For example, very small and numerous lights may not require visibility calculations at all.
- An example of this is photon splatting, where visibility is part of the photon tracing.



- On the other hand, with many very large lights we can use approximate visibility
- As errors average out, as it is called.
- An example of where this is done is Virtual point lights.

- We aim for something in between these,
- targeting the numbers of lights and degree of overlap we might expect in modern games.
- We also target fully dynamic environments, so nothing is precomputed.
- The resulting numbers of lights per pixel implies that we must calculate high-quality shadows.
- As there will not be enough overlap to hide errors.

- This scene has with about 13 lights per pixel, which is quite a lot…

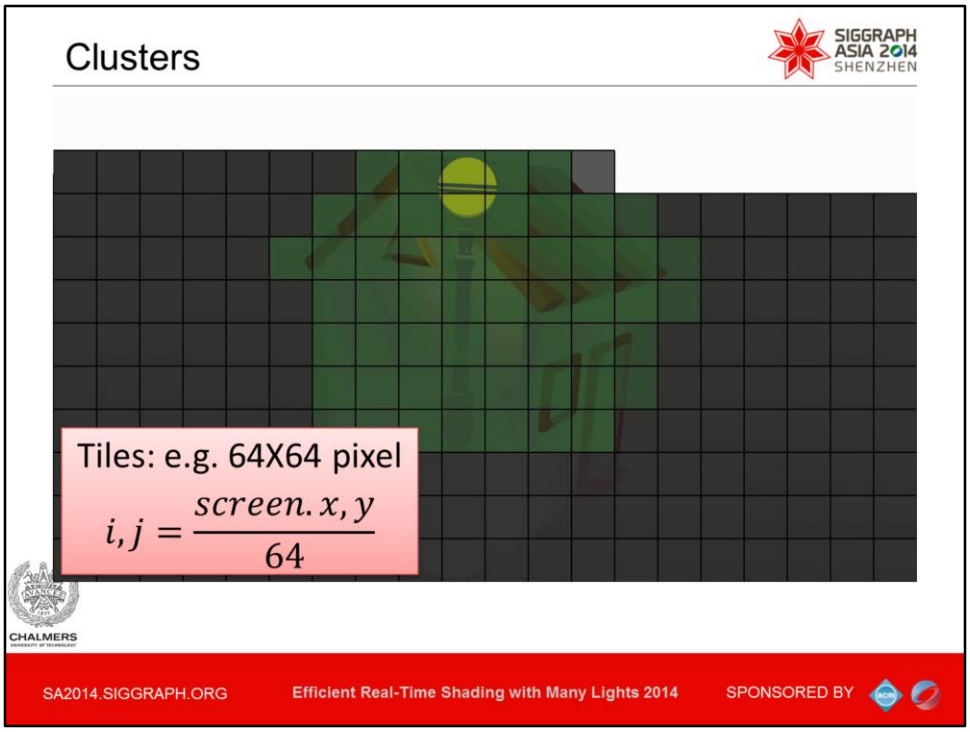


- Removing the lights spheres, leaving just the shadows.
- As you can see, each shadow cast is quite sharp, and so we need a high quality.

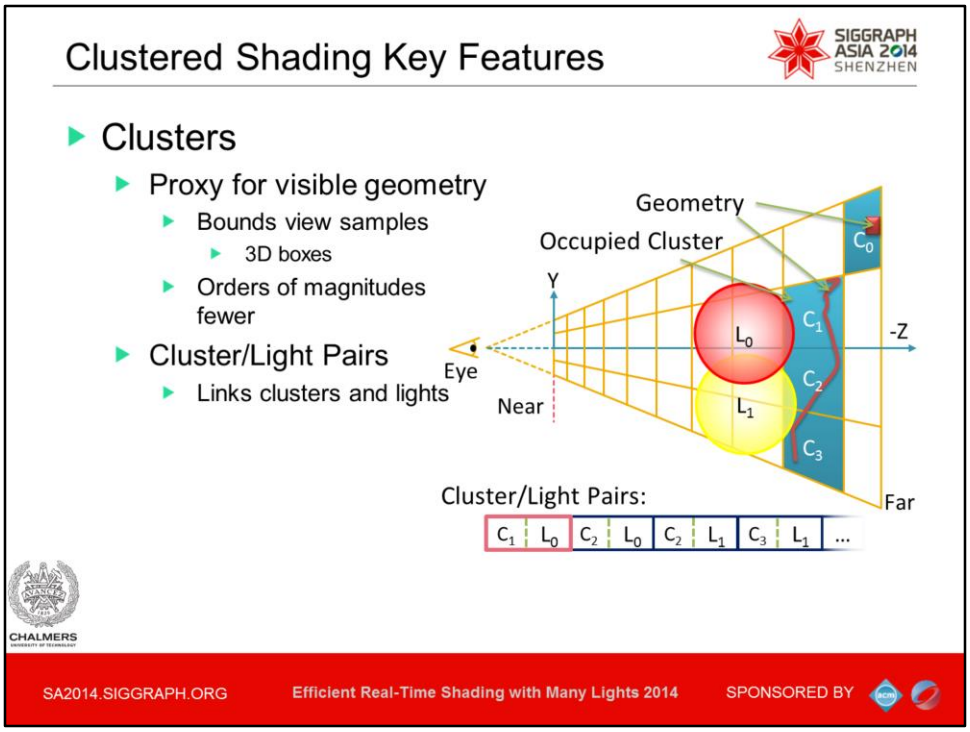
- Now some might wonder why we went with shadow maps?
- Although I'm guessing a lot of you are not wondering :)
- When we started this project, I had lots of more exciting ideas for how to do this.
- and it has even been stated that shadows maps are unsuited for many lights.
- However, when limiting the problem as I have just done, there really is no fundamental reason against shadow maps.
- And given that they are the de facto standard in the real-time industry, they must clearly be the first stop,
- if nothing else to provide a benchmark for more clever ideas.

- These are the steps we need to perform each frame, using the current camera view, to calculate a shaded scene.

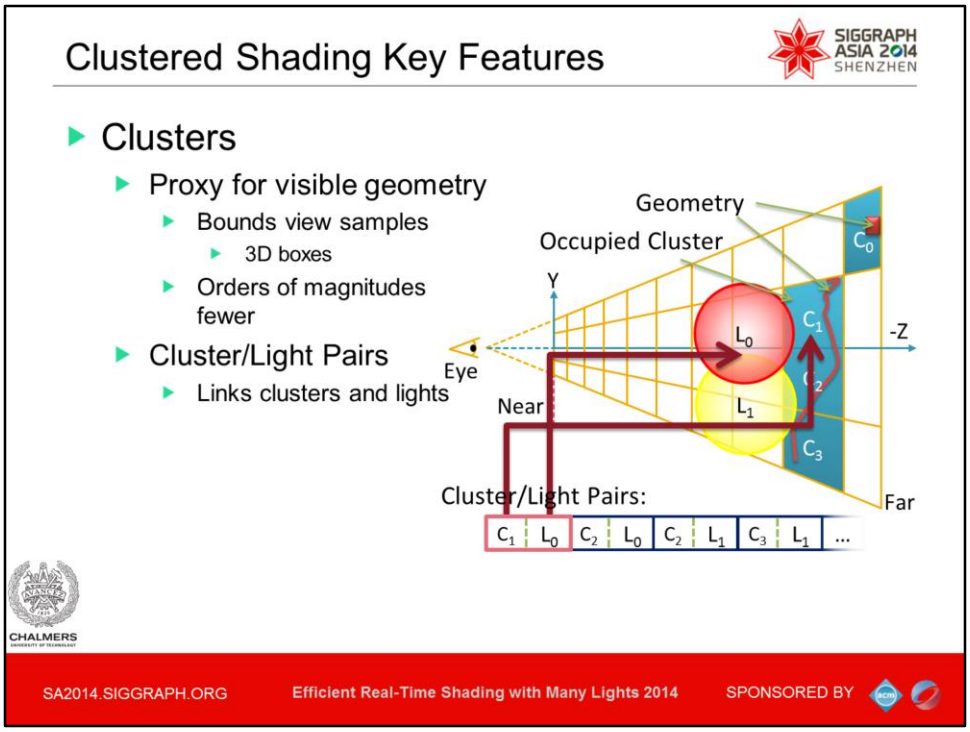
- The first step is the same as determining what lights are needed for shading
- And we have already seen how this can be achieved using clustered shading and other methods.



- We use clustered shading as the starting point.
- And recall that this algorithm is very efficient, coming close to the minimal set of lights for shading,
- and so provides a good starting point for adding shadows.
- To recap, in its simplest form, it is the extension of screen space tiles, shown here…(click)
- into 3D, by placing regular subdivisions along the depth direction.



- Note that the clustered shading method we build upon here is the one described in the paper, not Emils eminently practical method.
- The reason for this is that we need some information about the shadow receiving geometry to be able to get good shadow performance.
- Therefore, we cannot just use the up-front full grid approach.
- Two things about clusters are worth repeating in this context:
- First, the visible geometry is approximated reasonably well by the clusters, but there are very few in comparison. Usually around a thousand times more pixels than clusters, though this is of course tuneable with cluster size.
- Secondly, clustered shading provides an association between clusters and lights. For shading it allows us to know which lights
- affect each cluster, but also the reverse mapping, which we will make heavy use of to compute shadows.



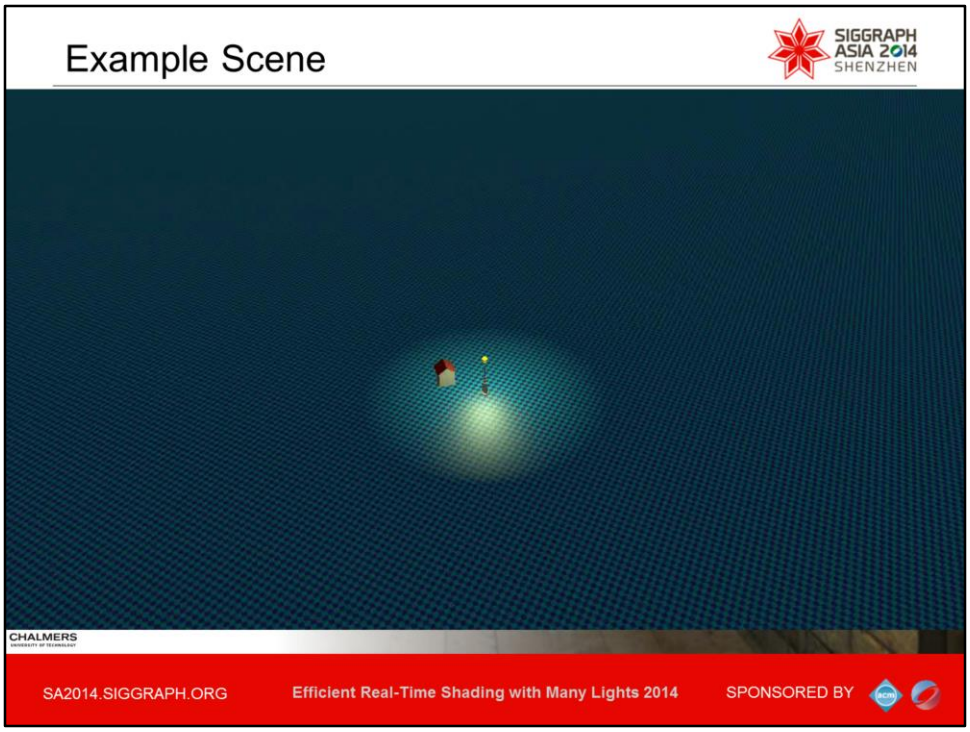
- The key data is the cluster/light pairs that enable parallel operations on these pairs.
- We use this for a lot of the processing later on.
- In the illustration, the pair of integers link the L0 light and the C1 cluster, the cluster is shown here as containing geometry, and overlapping a light, and it is these that we will process.

- So, the clustered shading algorithm provides the lights affecting each cluster, and a light that does not affect any cluster need not be processed further.

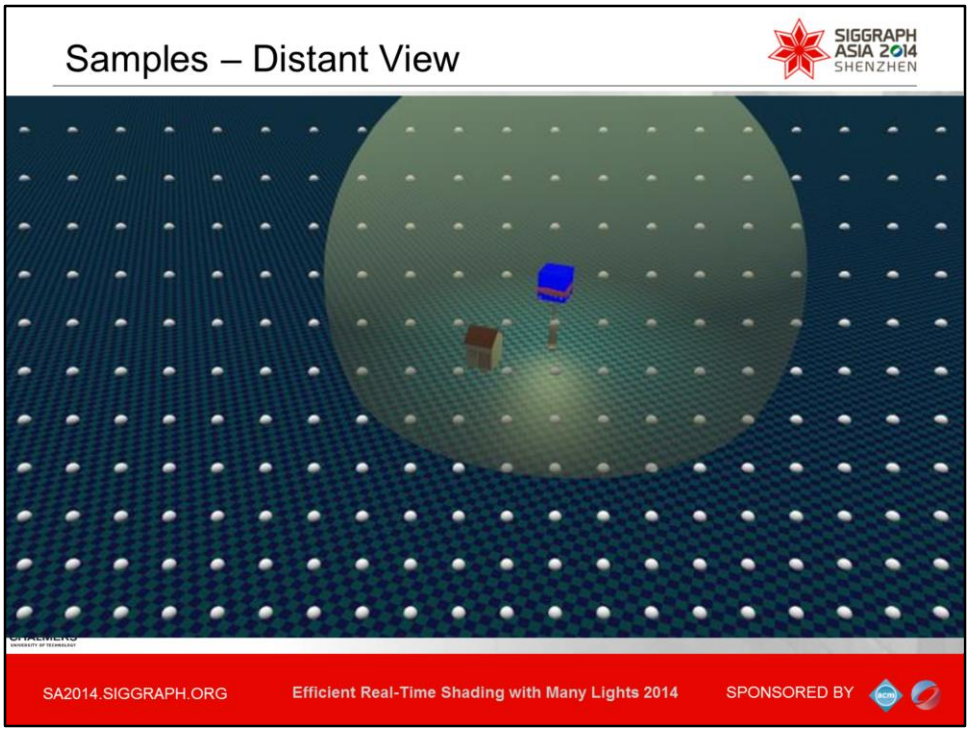
- Next we must work out the resolution of each shadow map

- We could of course just pick a constant resolution…
- but then we'd not be talking about high quality shadows any more, with over and undersampling both being the norm.
- Using oversampled shadow maps is not just wasteful in terms of memory, it is also slow and may yield aliasing.

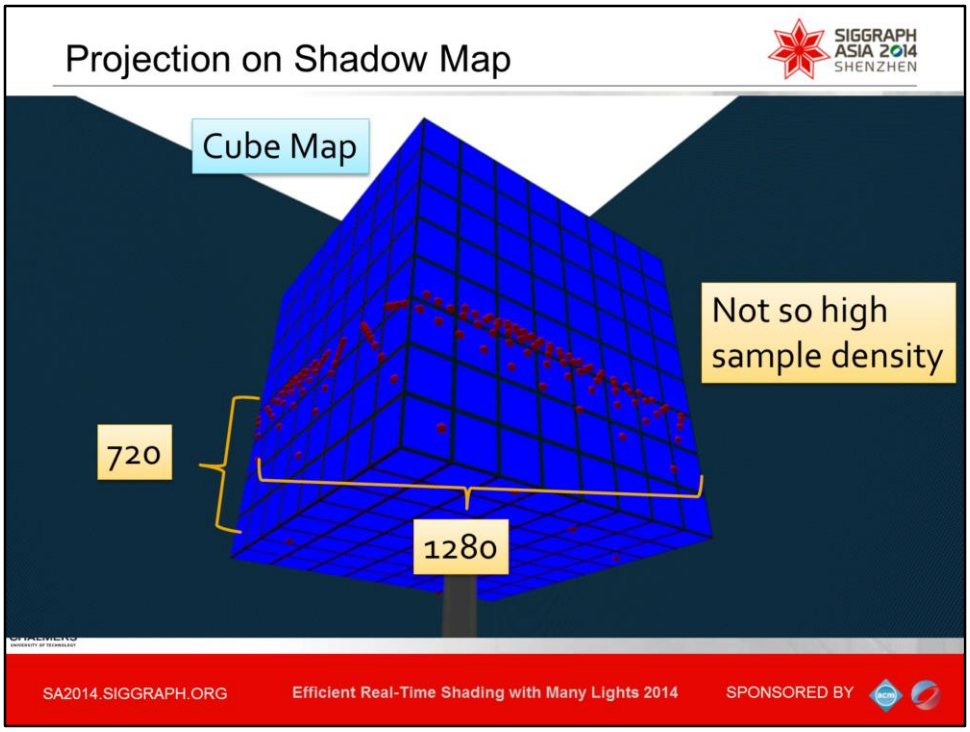
- So what is a good resolution for a shadow map?
- Well, the easiest way to think about it is that to avoid sampling artifacts (as far as possible)
- Is to have one sample in the shadow map for each sample on screen…



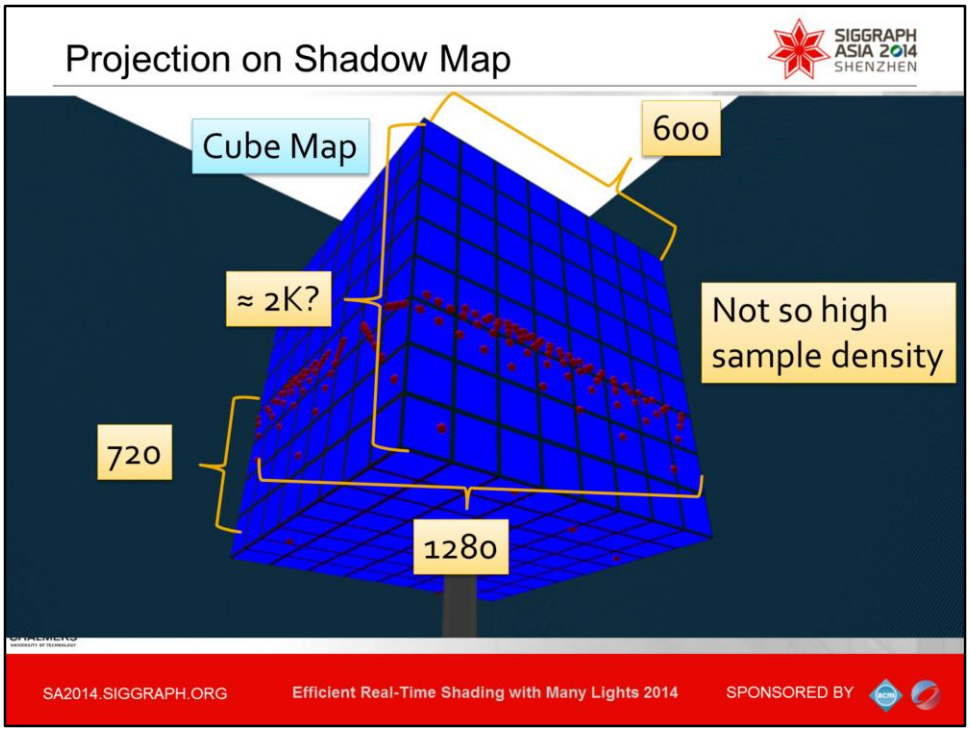
- to illustrate the idea, here's a simple scene,
- With a house, lights sphere, shadow cube map



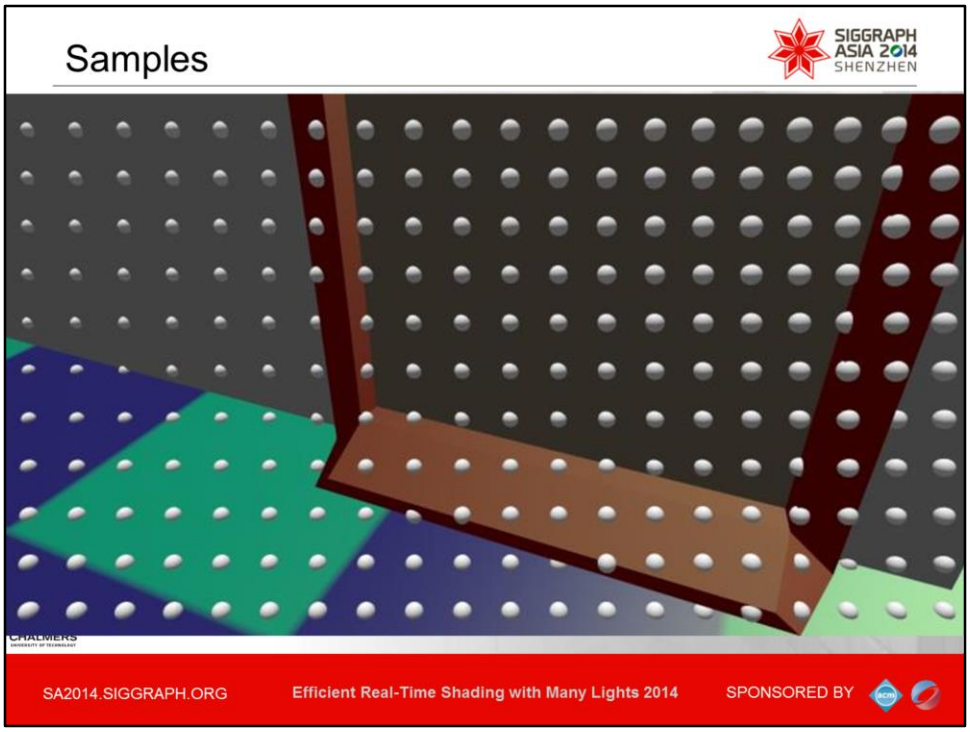
- View sample representatives, from a relatively distant view



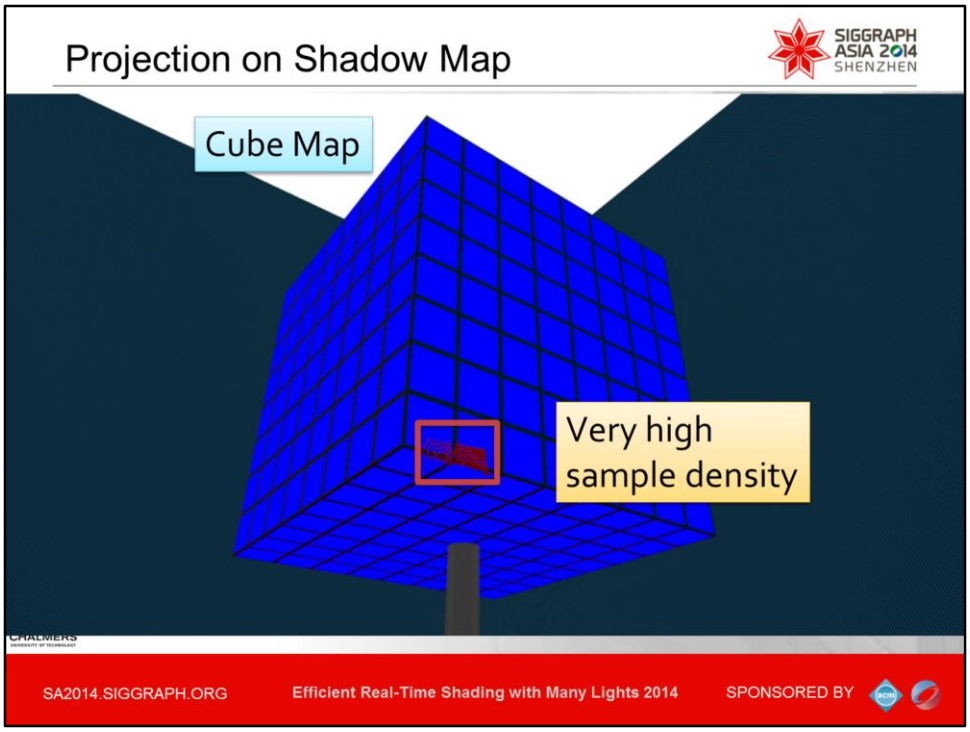
- And their projeciton on the cube map.,
- The density, assuming this was a standard definition render, is not extreme…



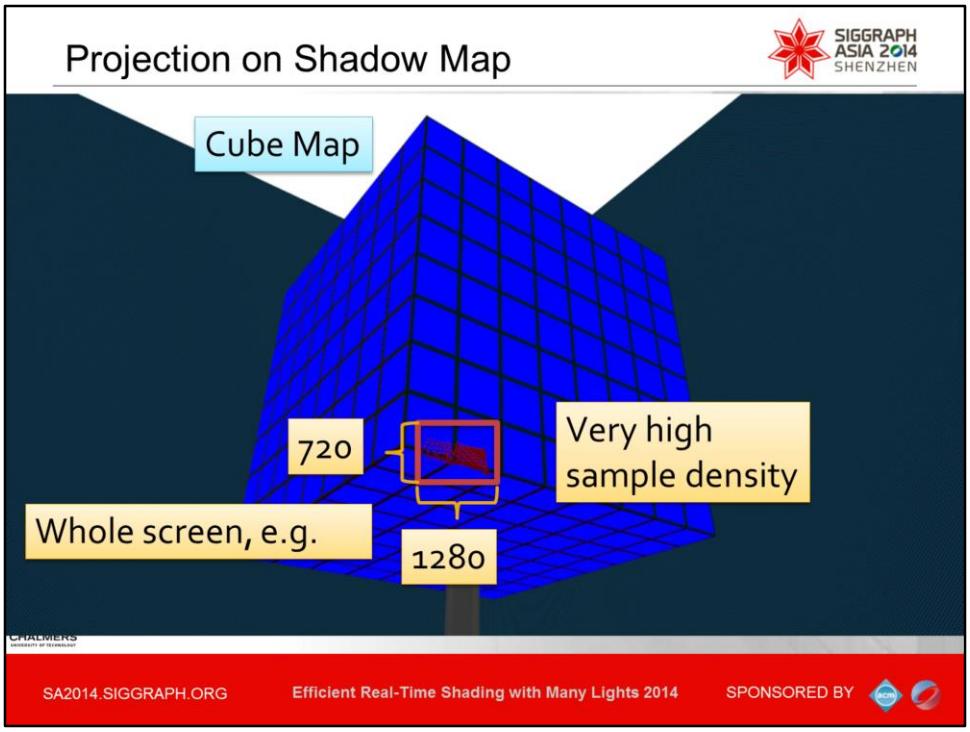
- with this sample distribution, the shadow map might need a 2k by 600 resolution.



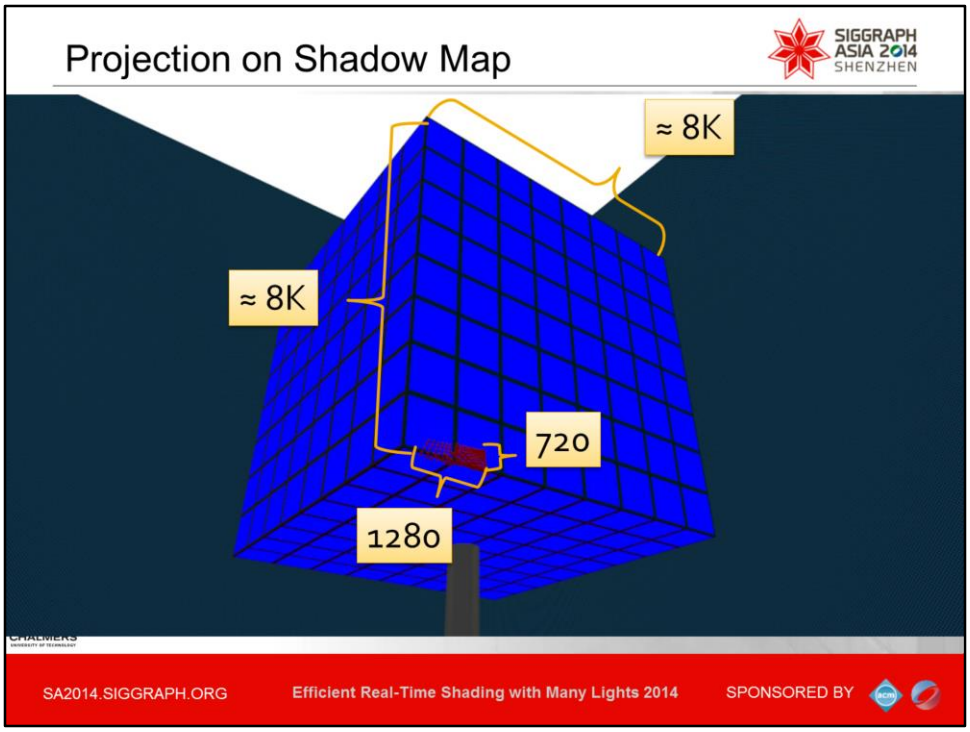
- Now we zoom the camera to a small part of the house,
- And thus concentrating the samples in world space…



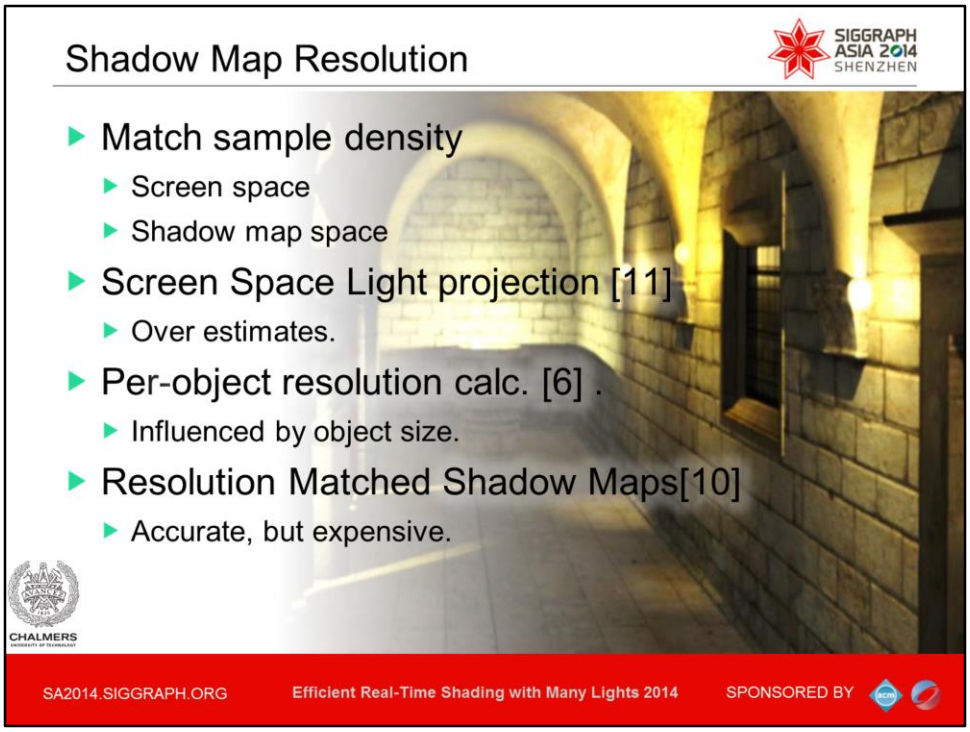
- Then we get this very dense projection onto the cube map



- Again with a waving the hands calculation…

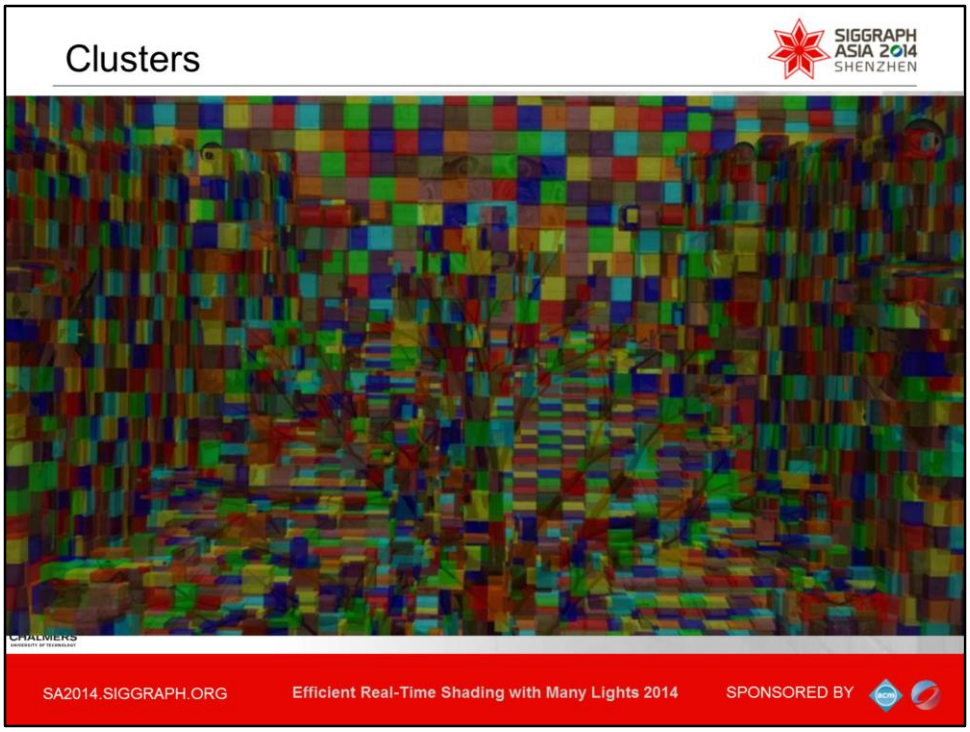


- Now, we could perhaps calculate the shadow map space derivatives for each sample individually, and use this to find the required resolution, by using the highest value.
- However, this can be quite expensive, and is

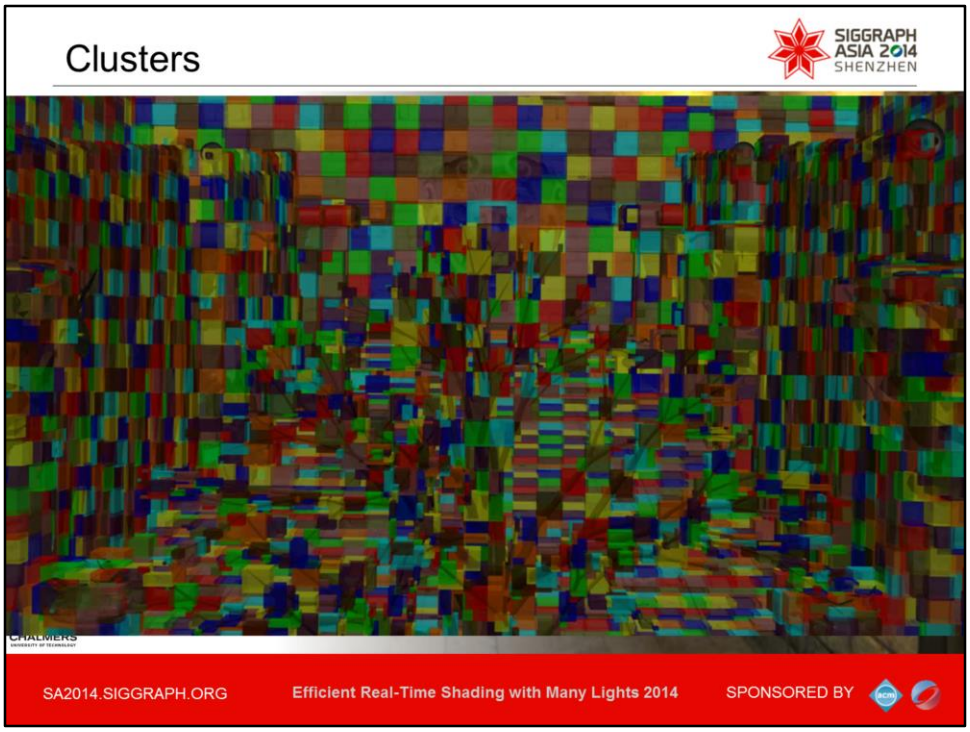


- This has of course been explored many times before,
- One common approach is that of King and Newhall, which is to simply estimate the required resolution of the shadow map from the projection of the light bounding sphere.
- This works well enough when the light is far away, and thus small on-screen.
- But produces completely useless results when the camera is near or within the light sphere.
- Another approach, is that of Forsyth, calculating the required resolution from the projection of each object that is to be drawn into the shadow map.
- This suffers from problems similar to those of traditional forward shading,
- where large objects causes problems.
- In Resolution Matched Shadow Maps, the shadow map space derivatives is calculated for all samples in a full-screen pass.
- This is obviously pretty accurate, but quite expensive, especially for many lights.
- we'll take this idea and apply the knowledge about clusters of view space samples available in clustered shading.

- Recall that the clusters for a given view like this <click>

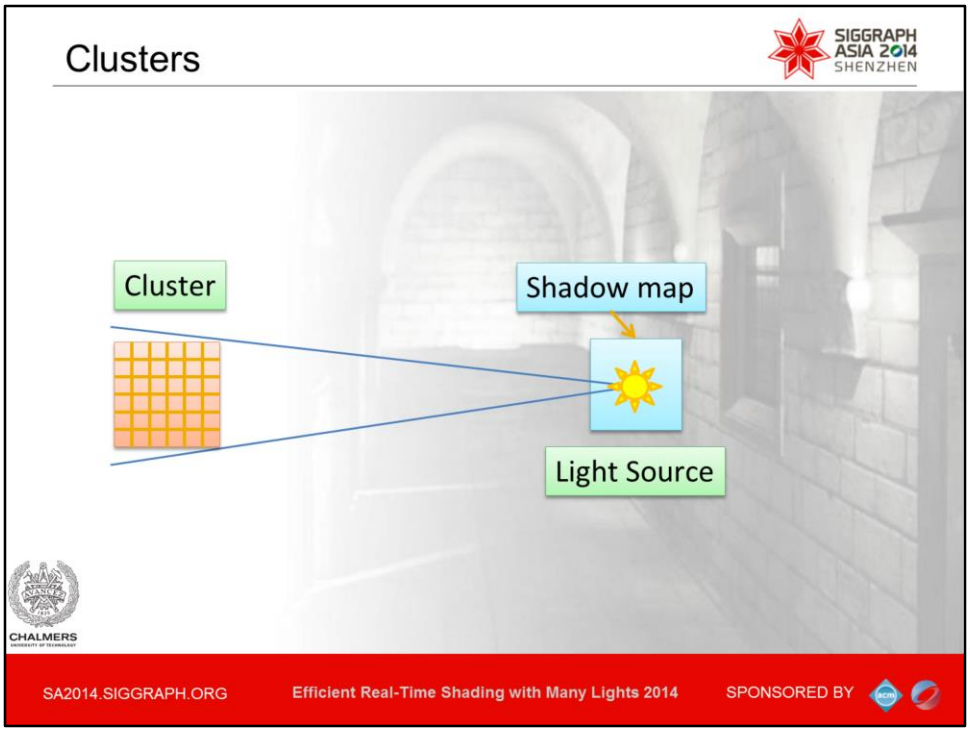

Clusters
SIGGRAPH ASIA 2014 SHENZHEN
CHALMERS
SA2014.SIGGRAPH.ORG
Efficient Real-Time Shading with Many Lights 2014
SPONSORED BY

- ...

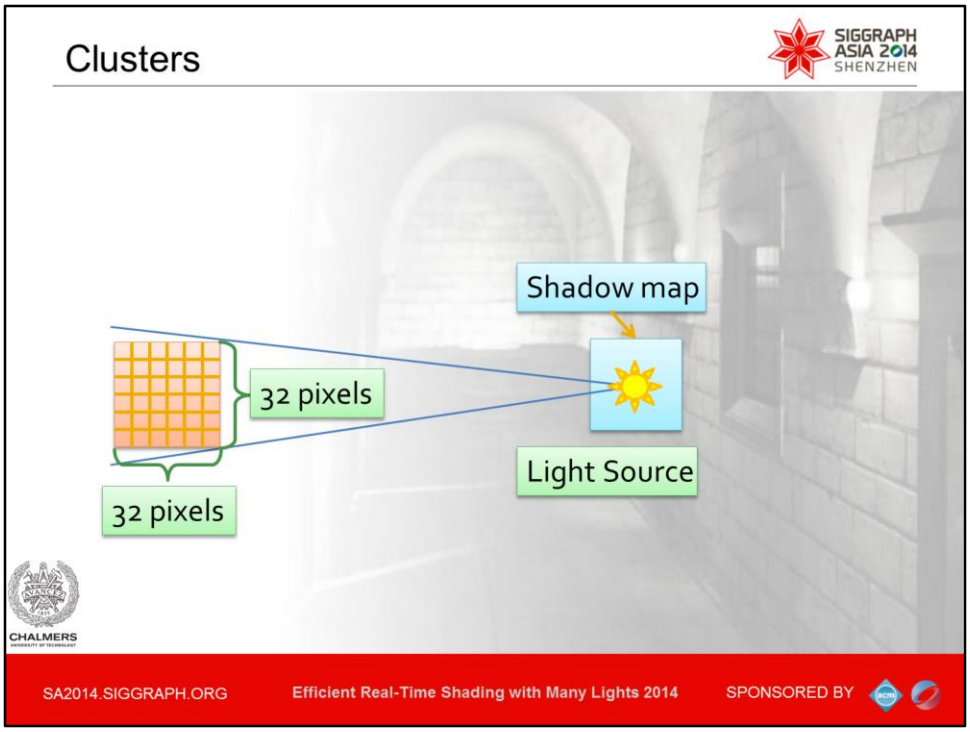


- represent the visible geometry <click>

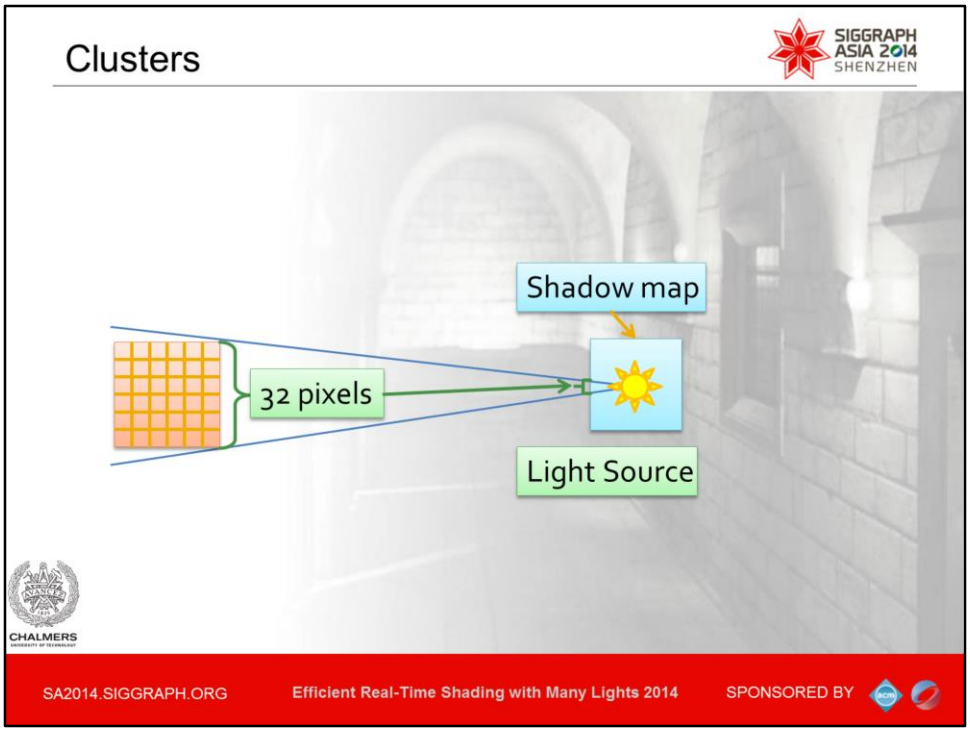
- …quite well.
- So we use them instead of directly using the pixels.
- Reducing complexity significantly.



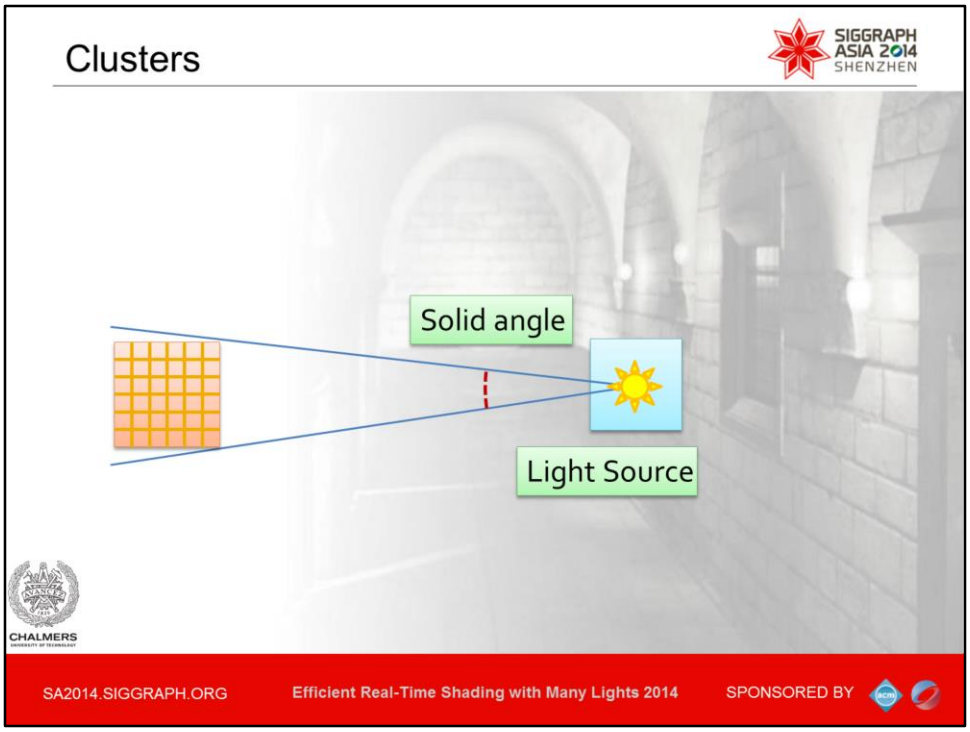
- Our approach works like this
- A cluster, here shown in screen space…<click>



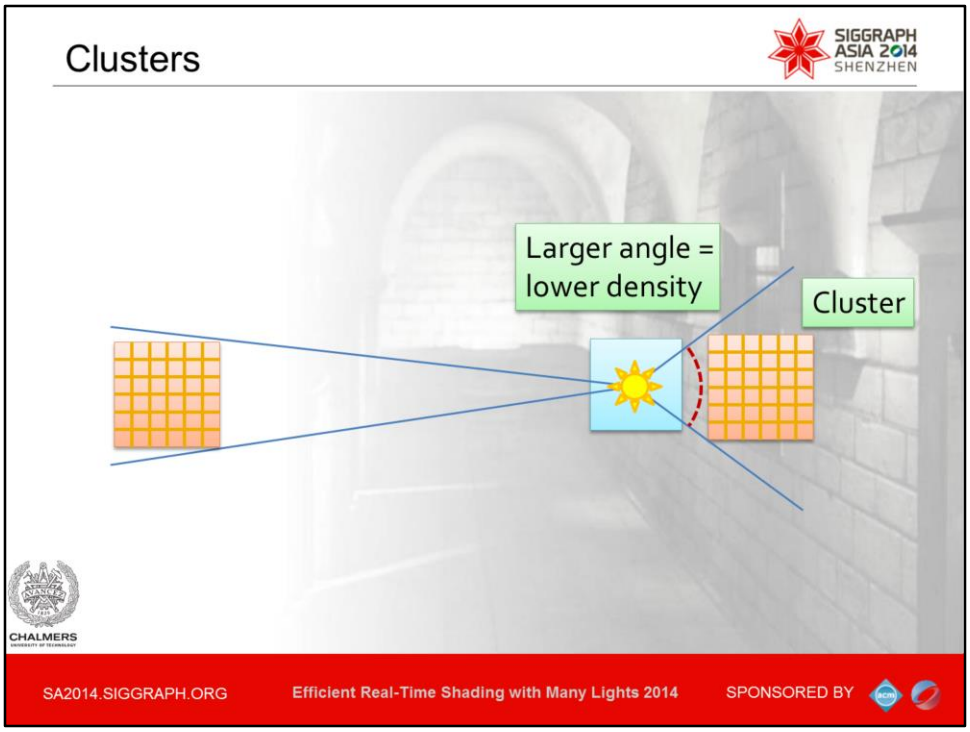
- Has a fixed, pre-determined, screen-space footprint (i.e., the tile size in 2D).



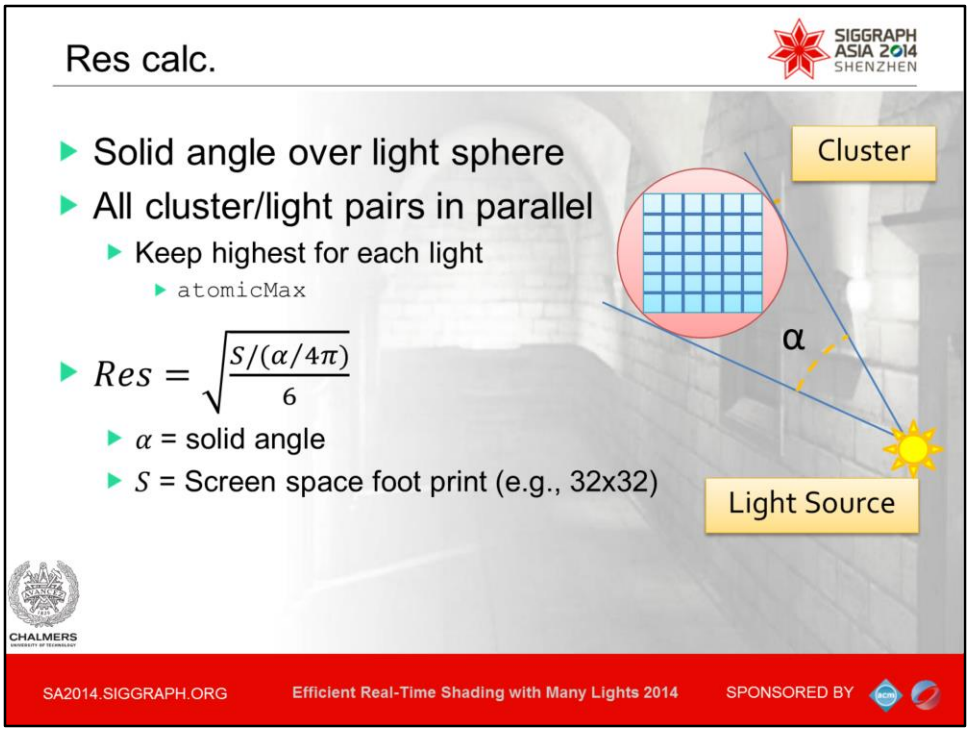
- This means that it's projection on the shadow map ought to have a corresponding footprint (in texels)
- to achieve that one-to-one mapping.
- And, yes, this is a fairly gross approximation, as it doesn't care about the orientation of the surface.
- But that's how we do it.



- The projection corresponds to the solid angle subtended by the cluster.



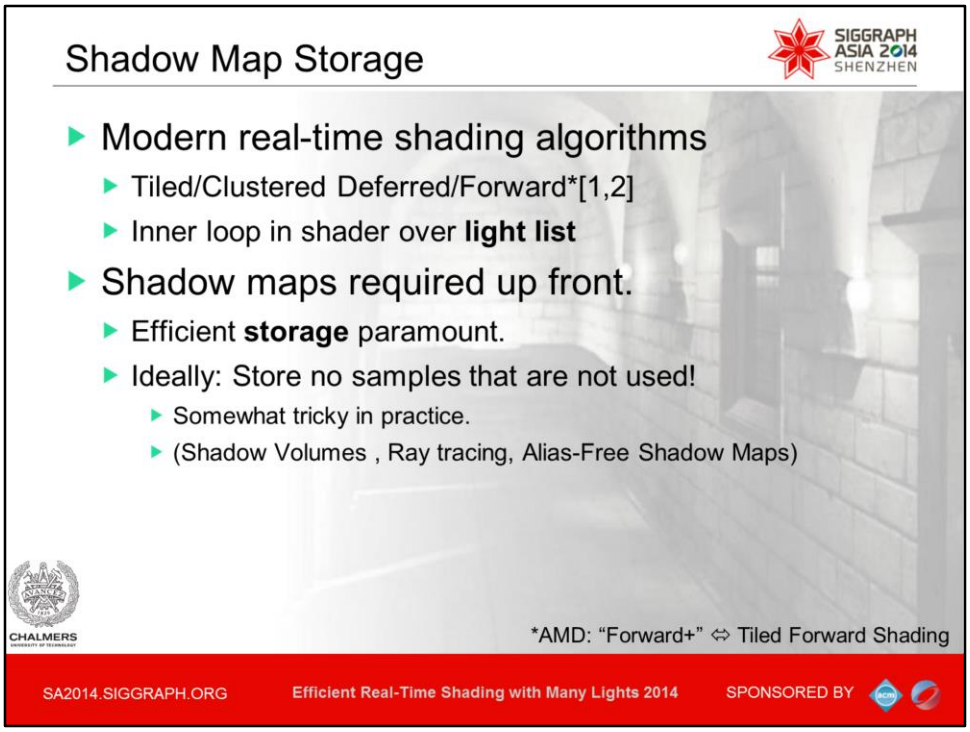
- In this example, we see that a cluster closer to the light requires lower resolution.
- As the same footprint spans a larger solid angle.



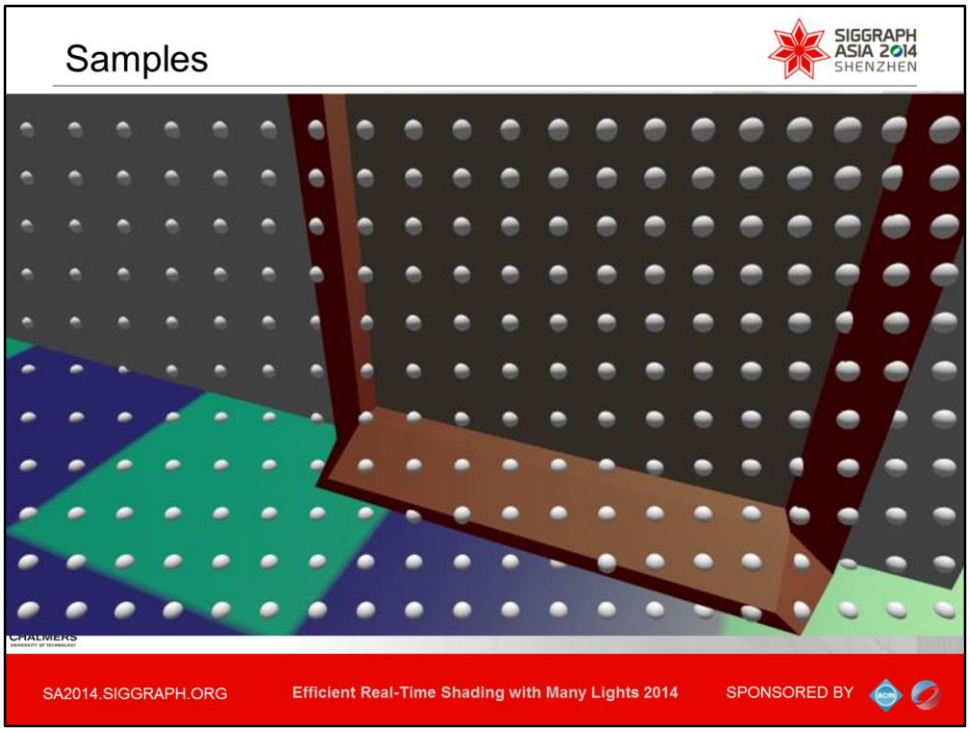
- We perform this calculation in parallel for all cluster light pairs.
- And reduce the final result for each light using atomic operations.

- This results in a list with a resolution for each light, or shadow map.
- Some will have zero, if they are not referenced by any cluster/light pairs, and need not be considered further.

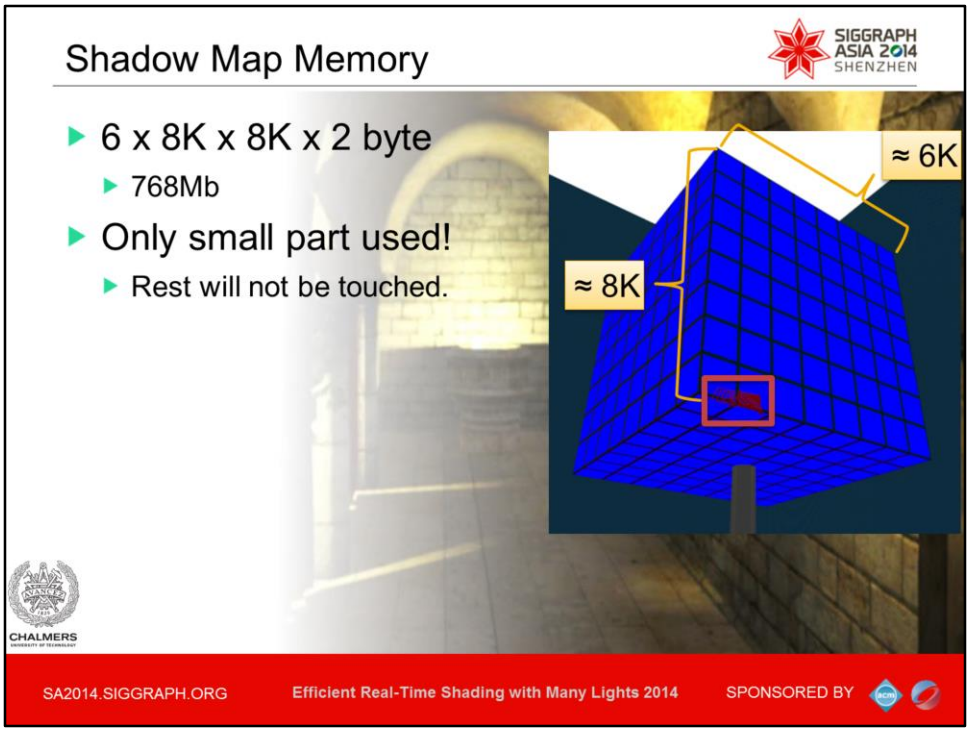
- Now, we'll need to provide some shadow maps to match said resolutions.



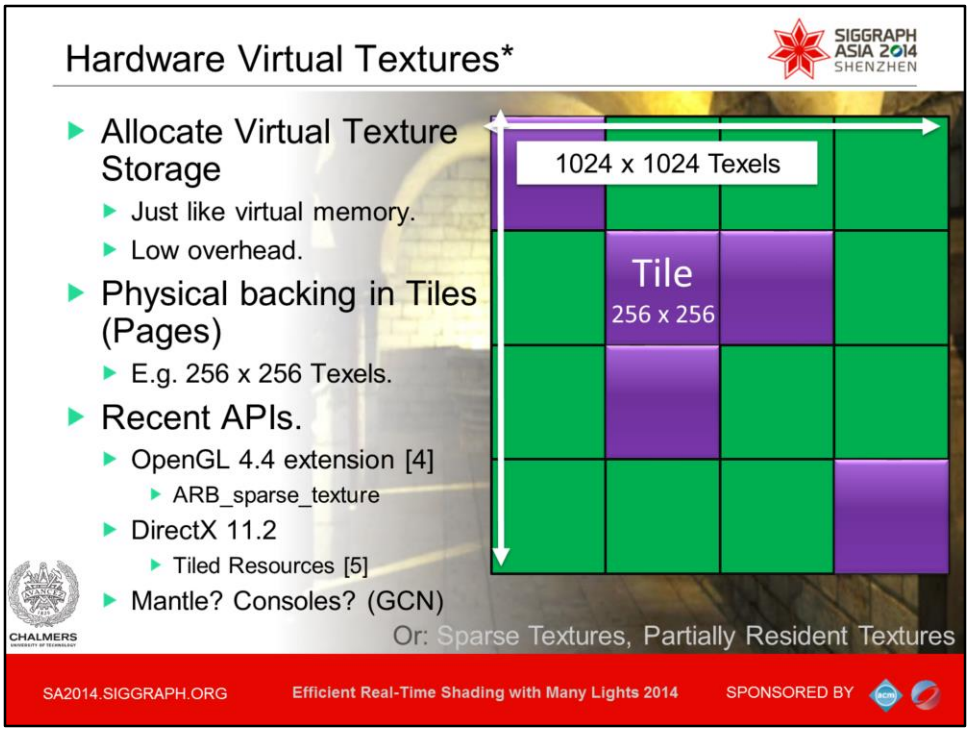
- Recall that clustered shading, like other modern modern real-time shading algorithms,
- have the inner loop, in the fragment shader, iterates over a list of lights.
- This means all shadow maps must exist up-front before the shading pass.
- Consequently, efficiently managing shadow map storage has become a very important problem.
- And this does not just mean parceling out shadow maps as needed,…
- but ideally we should be able to store only those samples that matter.
- Several shadow algorithms do just that, but fail to achieve consistent real-time performance,
- For different reasons.



- Recall the zoomed in view from before,

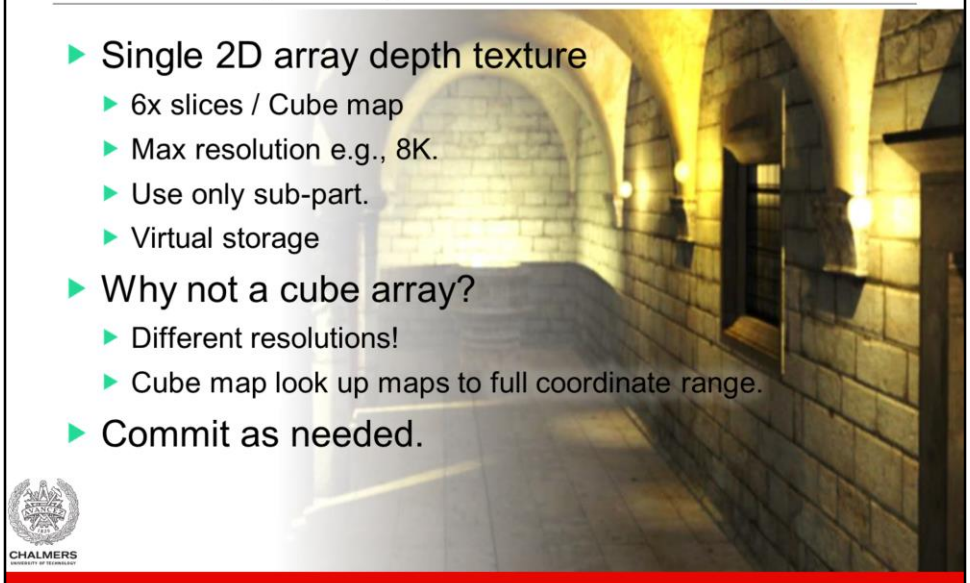


- And the corresponding projection on the shadow map.
- This requires a very high shadow-map resolution,
- But note how this happens when most of the shadow map would be unused!
- This is pretty much how it has to be, as the high density comes from looking at something very near the camera,
- and then we're guaranteed to not see so very much of the scene.
- This is highly wasteful and a fantastic opportunity, for...

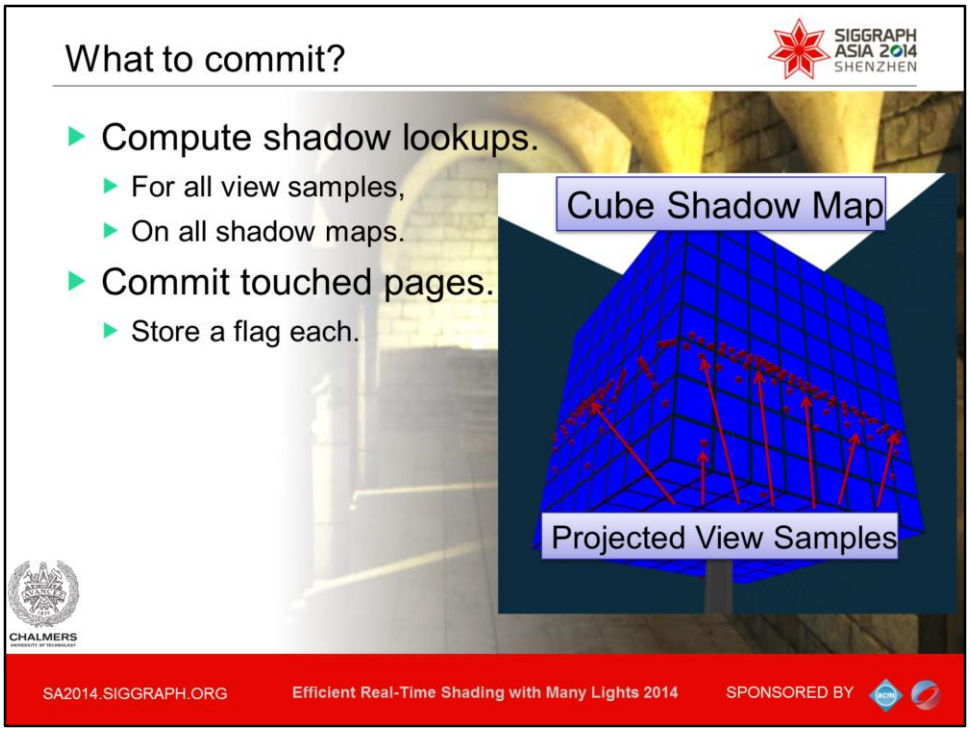


- …hardware virtual, or sparse, textures
- These have become available in mainstream graphics APIs recently,
- and makes it possible to allocate large virtual textures,
- …but commit physical storage in tiles only where needed.
- This fits our ideal pretty well, especially given that shadow map samples tend to clump together because of coherency [10].

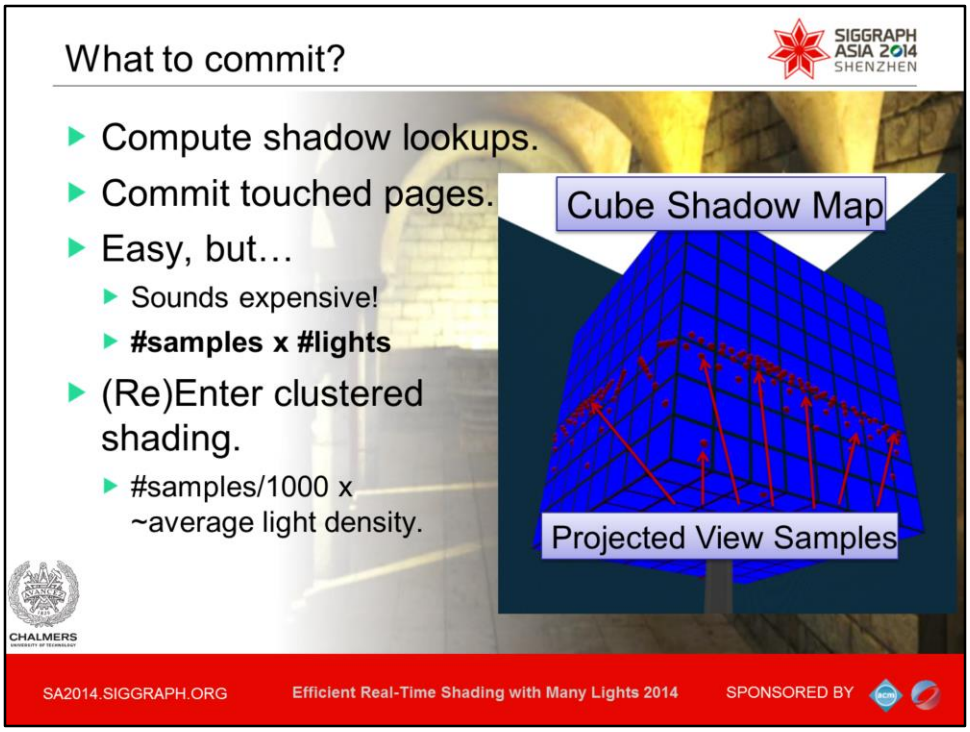
Shadow Map Storage



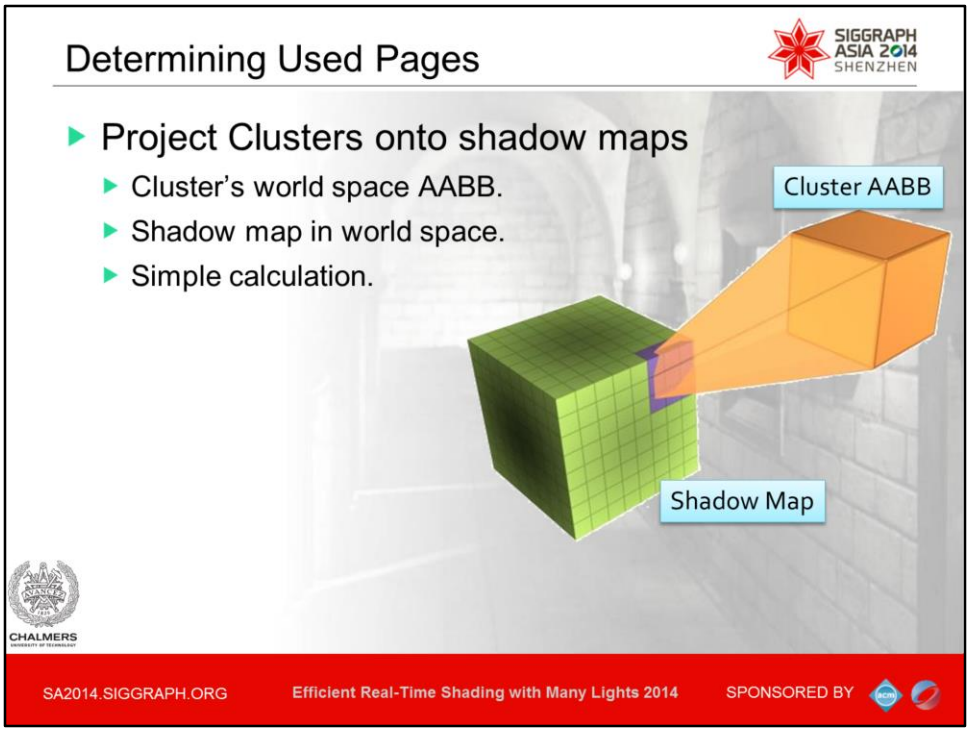
- Single 2D array depth texture
  - 6x slices / Cube map
  - Max resolution e.g., 8K.
  - Use only sub-part.
  - Virtual storage
- Why not a cube array?
  - Different resolutions!
  - Cube map look up maps to full coordinate range.
- Commit as needed.



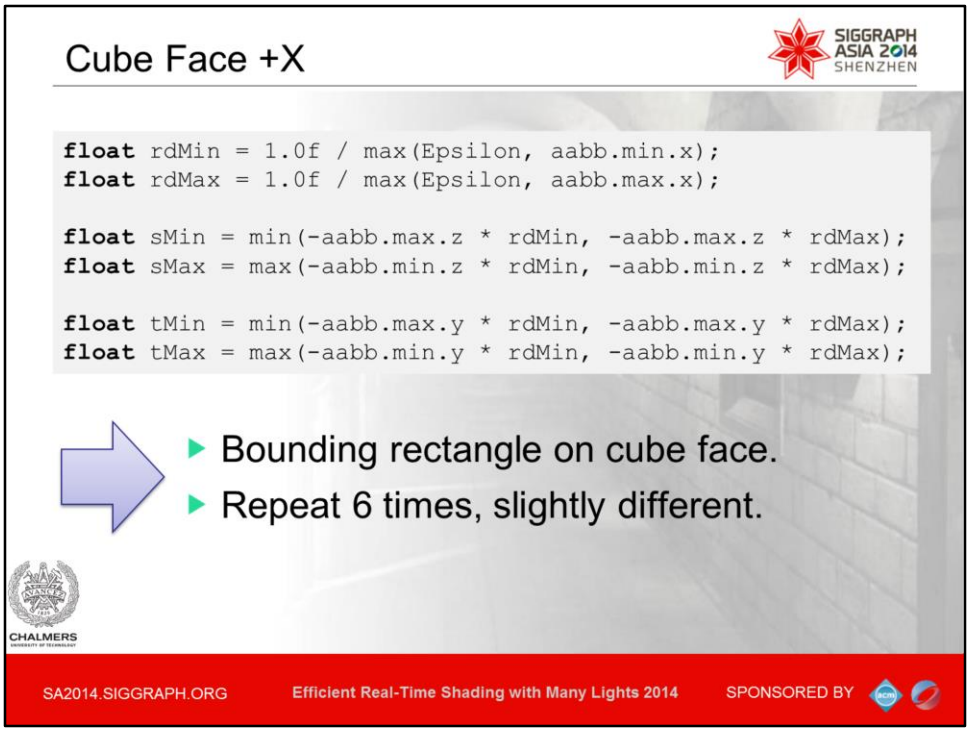
- So we must work out which physical tiles to commit.
- Essentially, we can solve this by performing all the shadow lookup once **before** drawing the shadow maps.
- and commit the touched pages.



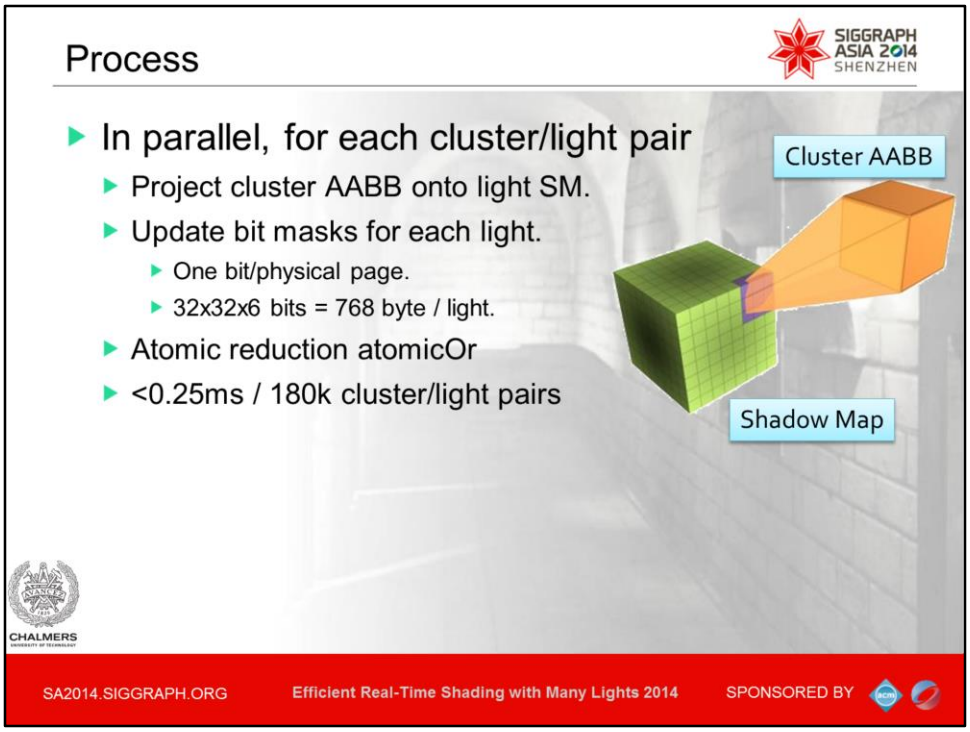
- So that is pretty simple in principle, but, just like with the resolution calculation…
- this is going to be a bit inefficient!
- So naturally, we dust off our trusty clusters and the cluster/light pairs to cut down both of these numbers significantly.



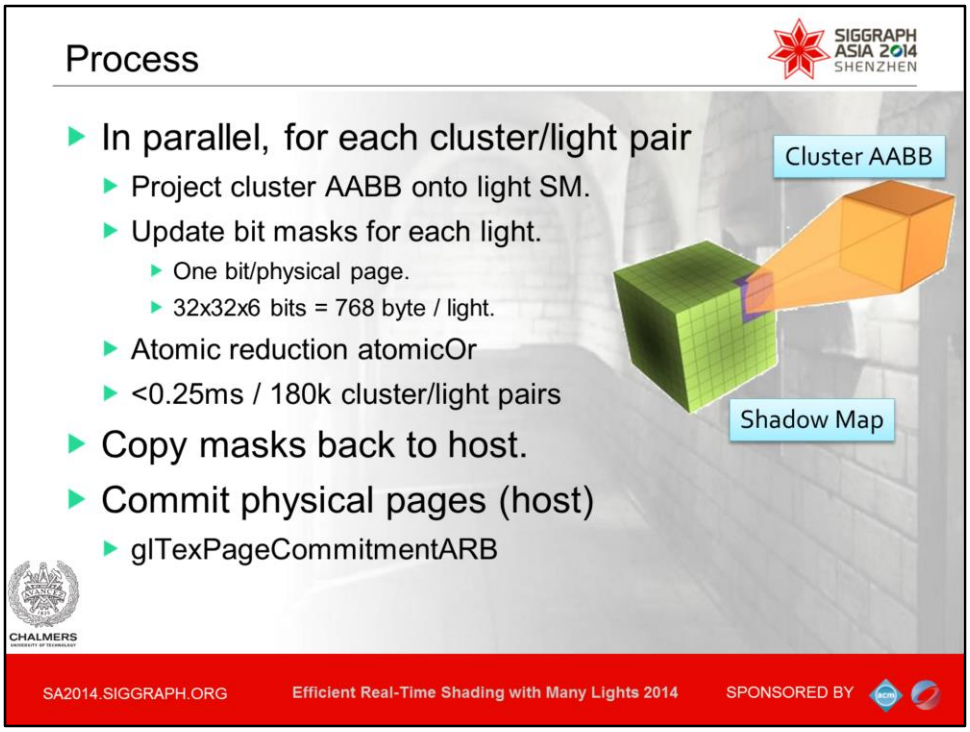
- Now however, we must project bounding boxes onto the cube maps instead of points (AKA shadow lookups)
- Projecting bounding boxes is a bit more complex.
- But by transforming the cluster AABB to world space
- And also keeping cube shadow maps in world space.
- Calculating the projection becomes very simple.



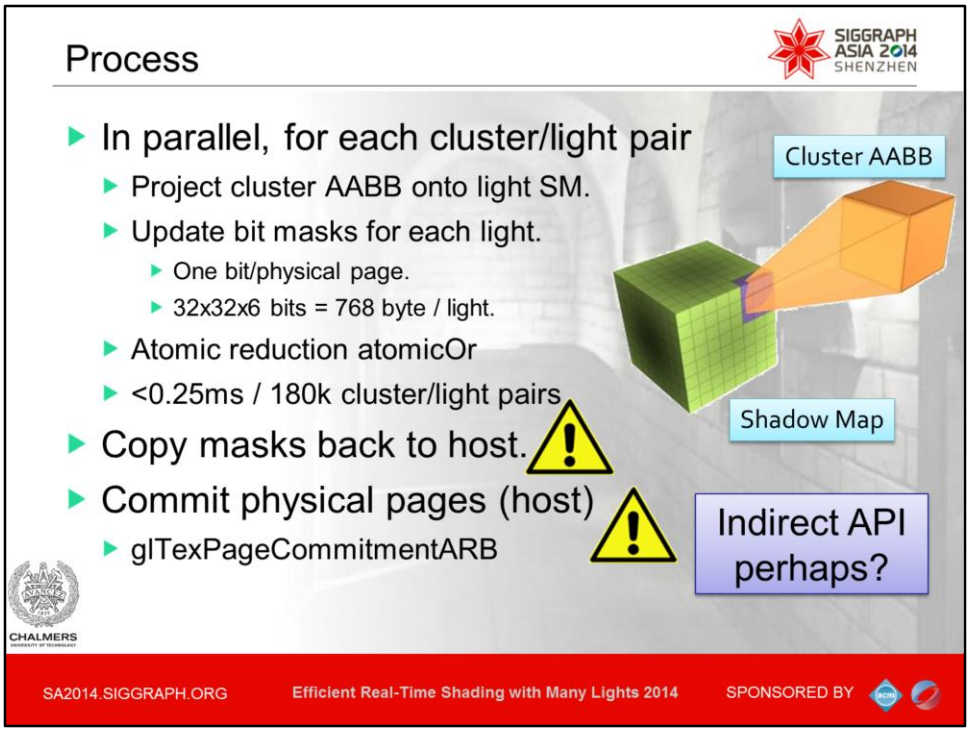
- This code is what we use to compute the projection on one cube face.
- And as you can see, it is not extremely complex.
- We repeat this, slightly differently for each face and get a rectangle in texture coordinates for each.
- This is then converted to a bit mask, with a single bit for each tile or page in the virtual cube map.



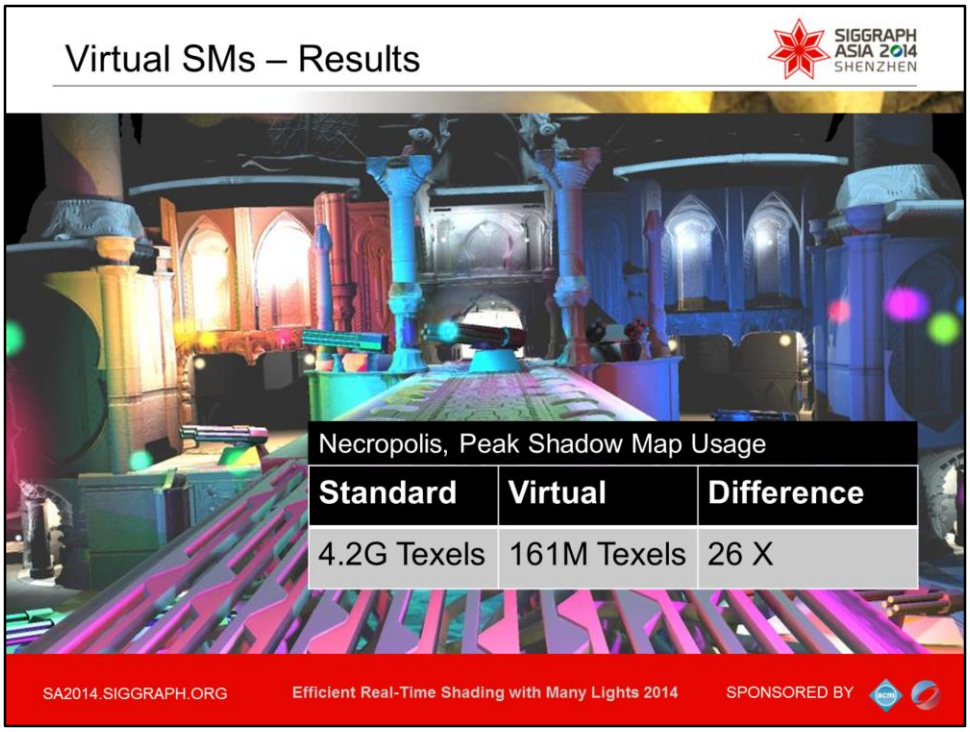
- We compute this mask, by running a cuda thread for each cluster/light pair.
- In the kernel we calculate the projection just shown
- As the mask is stored for each light and thus referenced by many cluster light pairs,
- It is updated using atomicOr.
- This is a pretty quick pass, <0.25ms, 180k light/cluster pairs.



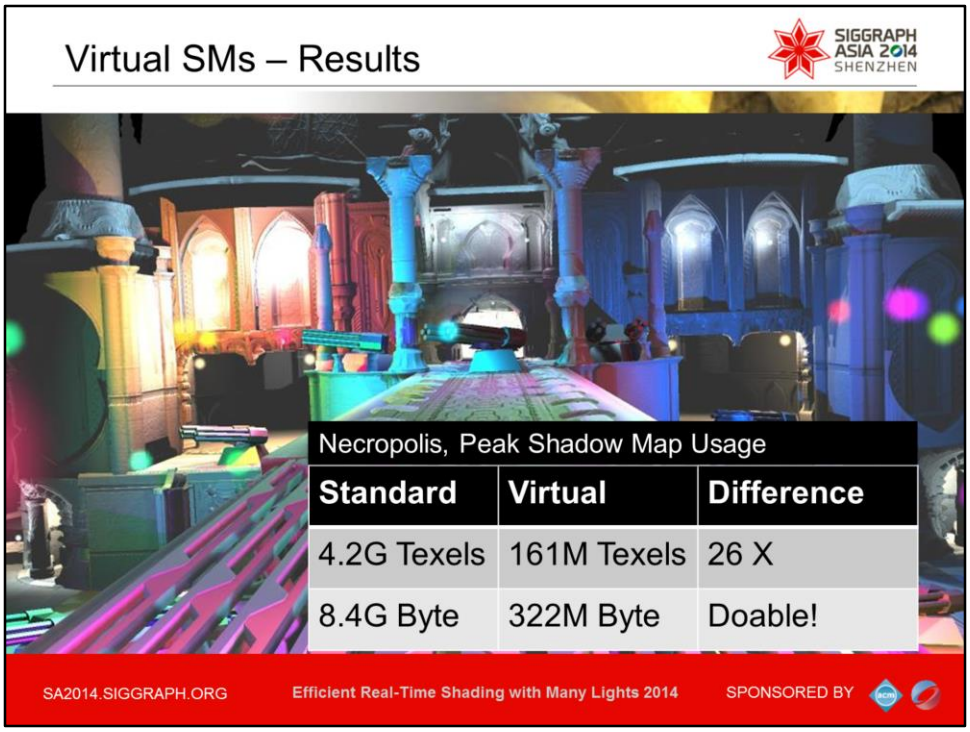
- The masks are copied back to the host, and used to commit pages for the shadow map textures.



- Warning bells
- This is potential stall waiting to happen, and also forces an API call per page commit.
- We'll come back to this problem later, but the spoiler is that it has gotten better!

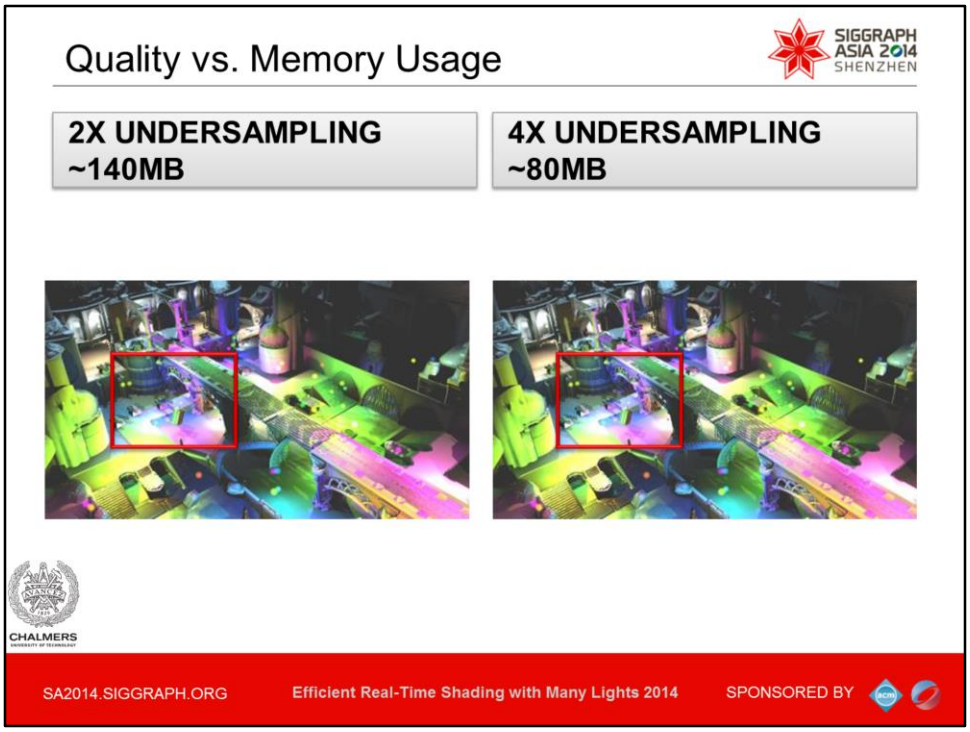


- So we might ask, is it worthwhile doing virtual shadow maps?
- Well, for the necropolis scene, there is a factor 26 improvement, compared to physical shadow maps of the same quality
- And <click>

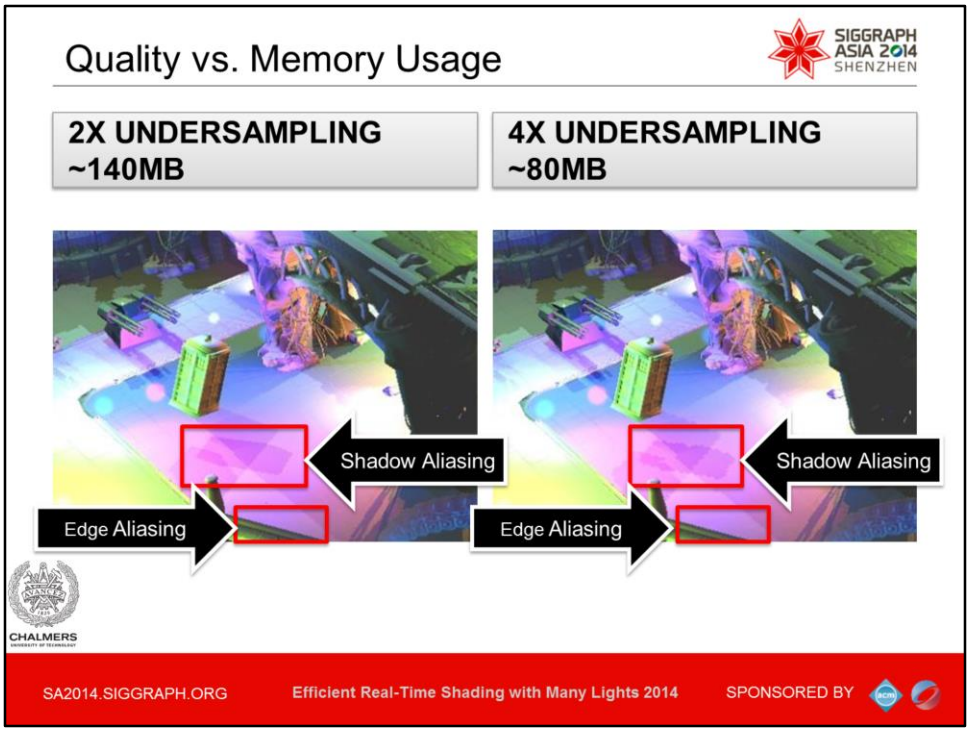


- …in other terms, this means the difference between impossible on a current console
- And something we might consider.

- Our method achieves quite uniform shadow quality,
- This means we can control quality and thus memory usage with a global parameter.
- This allows more flexibility in memory use while maintaining uniform quality.
- Rather than say, some lights getting very poor shadows and others good.
- One idea is to do this dynamically, to ensure a certain memory budget.
- Should be possible as it is very quick to work out memory usage from the used pages and resolutions.

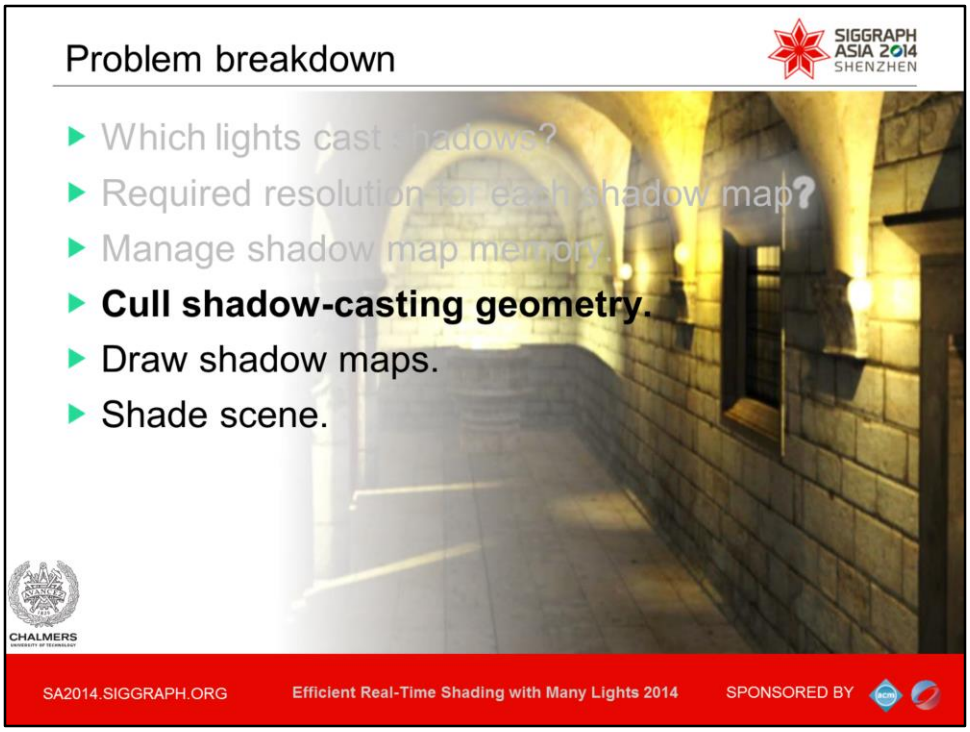


- Here is a shot showing this, with 2x or 4x global reduction in shadow map resolution.
- Shown are the corresponding peaks in shadow map memory usage.
- Recall that the peak is 322MB without reducing quality.

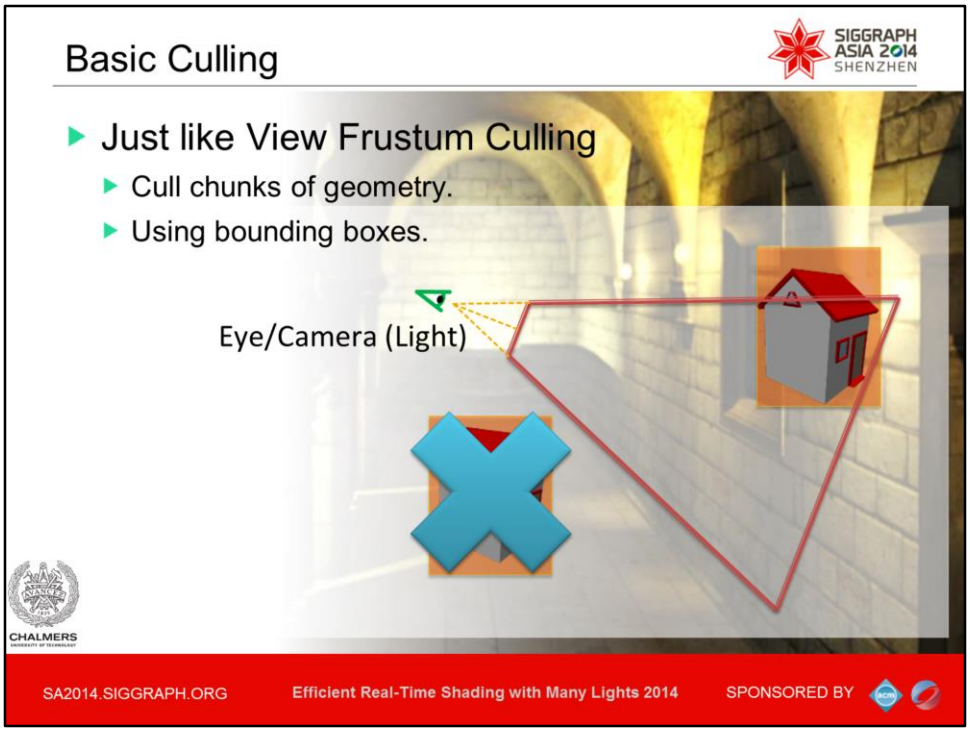


- Zooming in, we see that the shadow aliasing is about the expected level when compared to edge aliasing in the image.
- This indicates that our simple scheme for calculating the required resolution yields reasonable results.
- With filtering, this can be acceptable, especially as it allows *uniform* quality rather than unevenly allocating shadow maps.
- Memory usage is, reduced significantly.

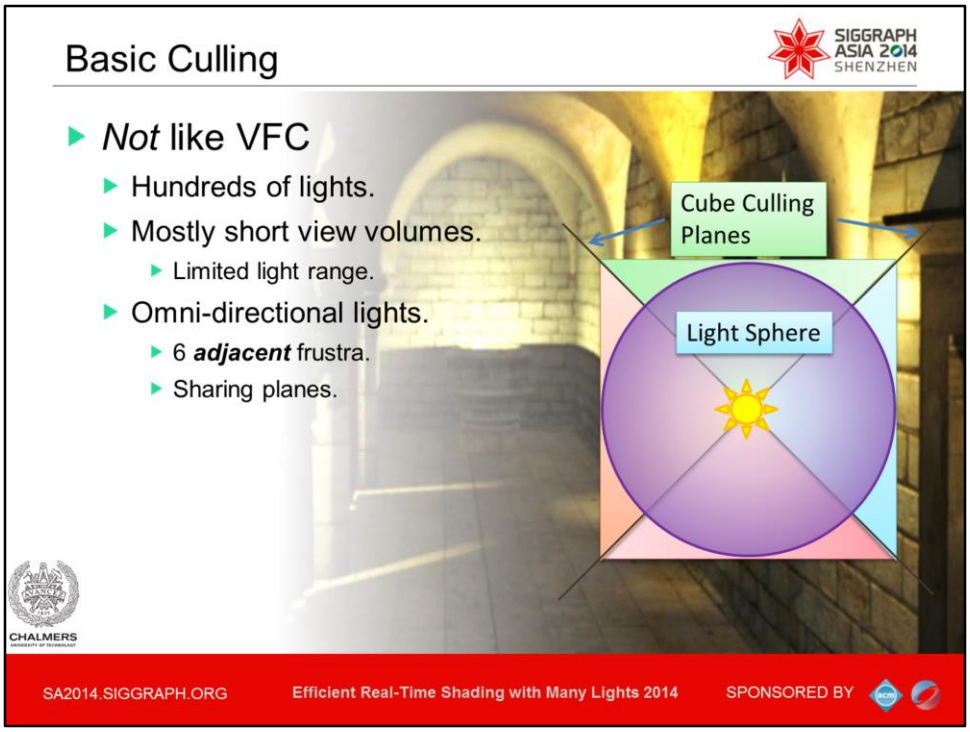
- So that takes care of memory management, and we can now allocate shadow maps to draw into.



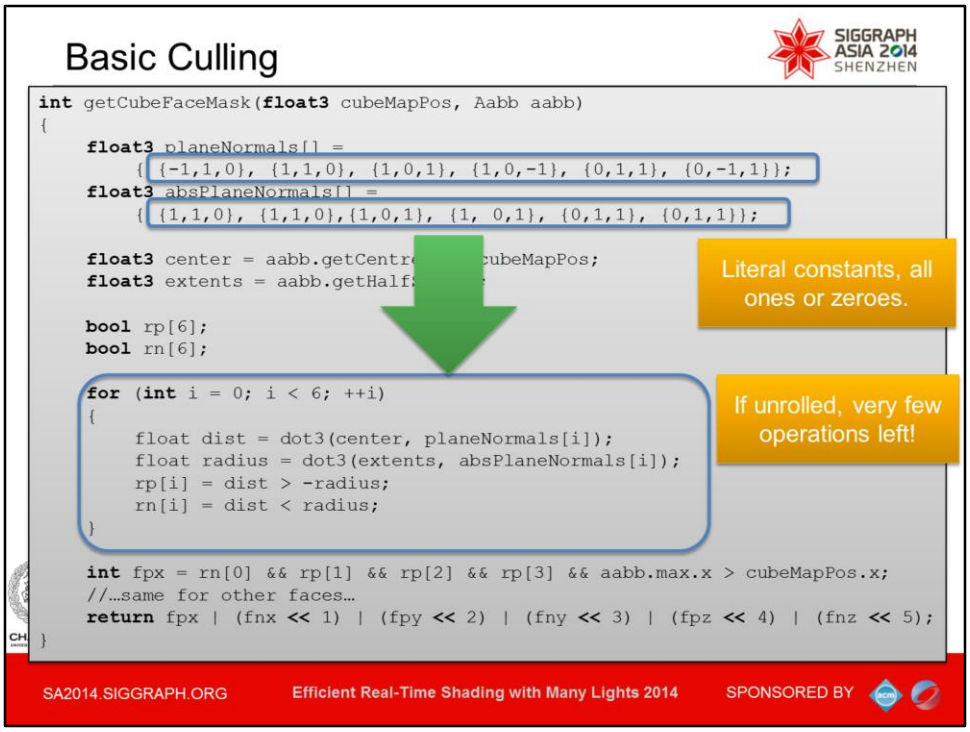
- Next must rasterize triangles into the shadow maps.
- To do this efficiently requires culling, as with any rasterization, only more so.



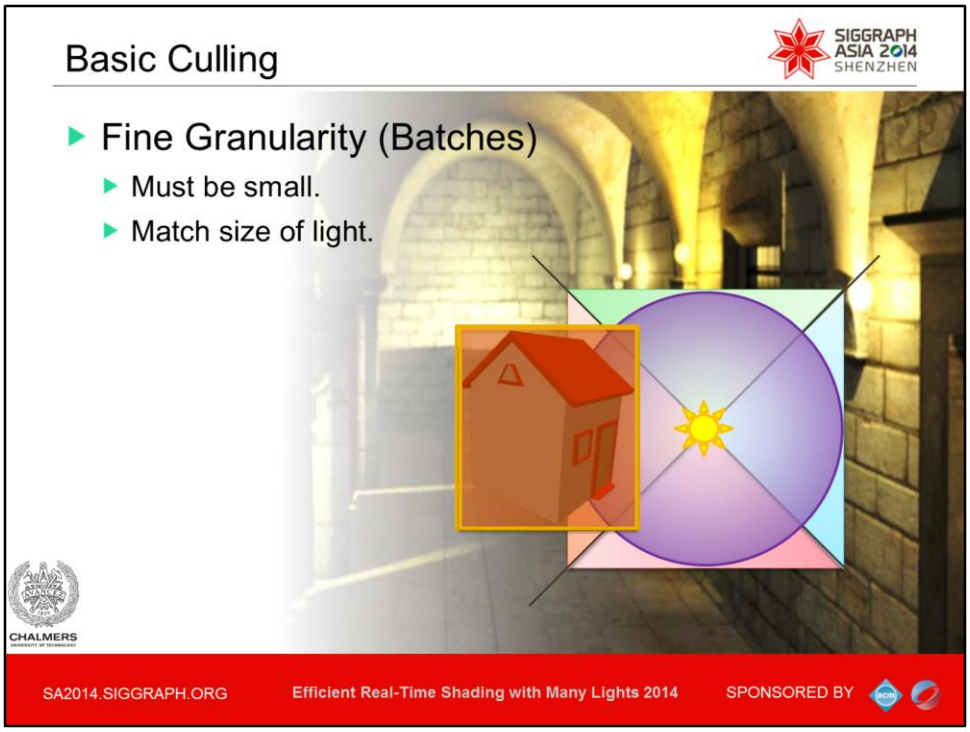
- This process is just like good old view frustum culling
- In that we are trying to get rid of geometry that is not visible, or within the view-volume.
- And that we do this by testing bounding volumes representing batches of triangles.



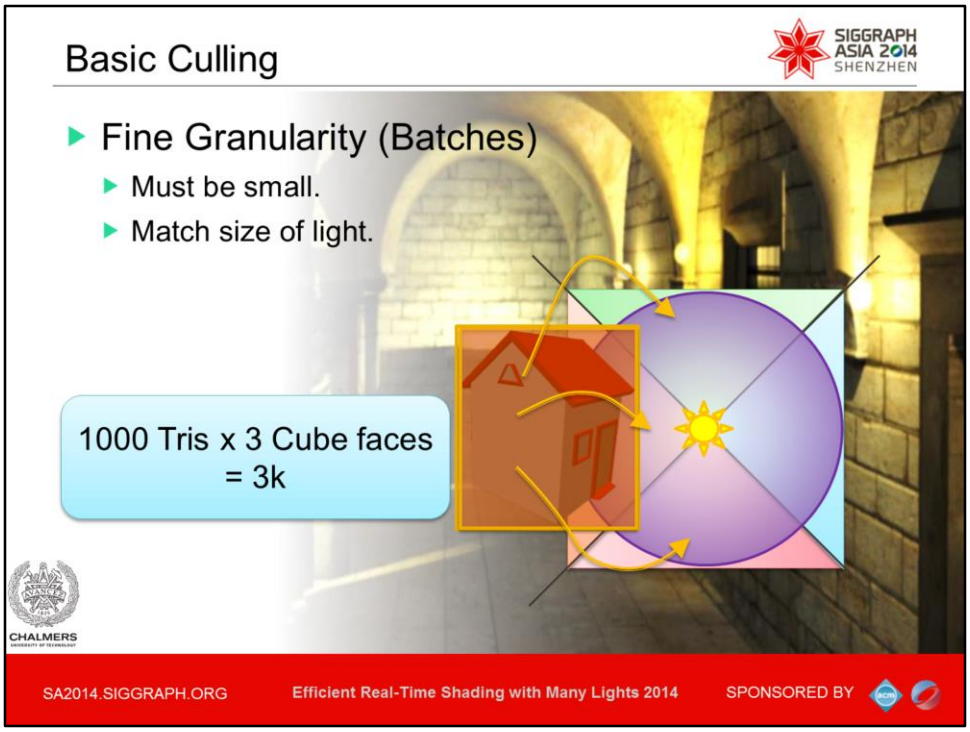
- What is not like view frustum culling is that we need to perform hundreds of these tests.
- The view volumes are quite small, or short, given the limited range of the lights
- There are 6 *adjacent* frustra sharing planes.
- And this adjacency is an opportunity to share calculations.



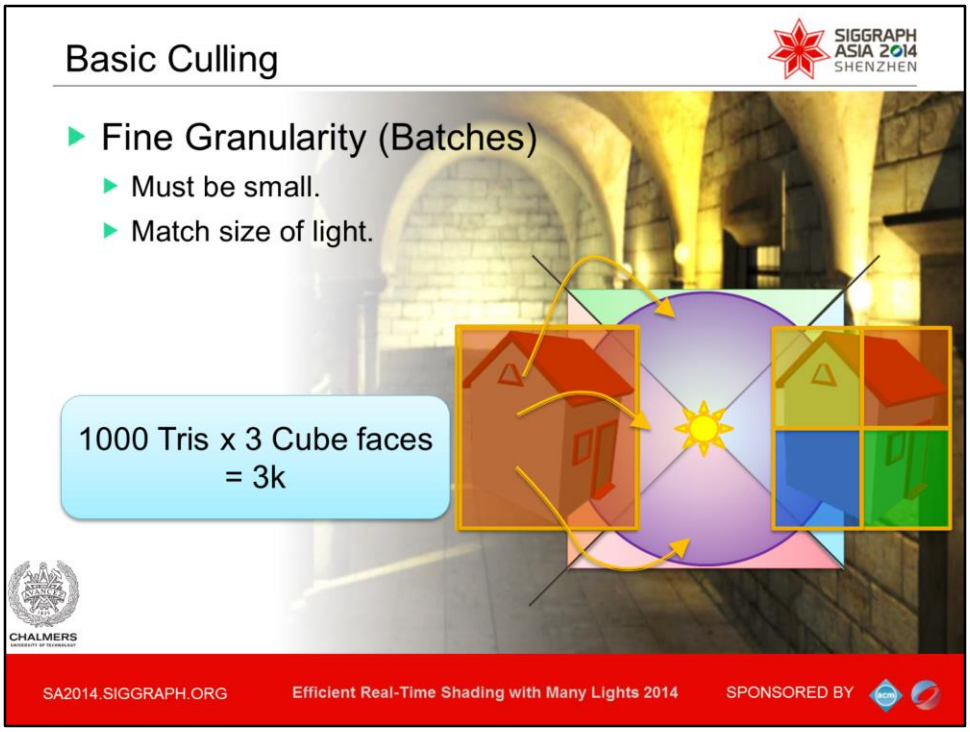
- Here is the somewhat condensed code we use to calculates the culling mask, with a bit for each cube face.
- This a very efficient, testing only 6 planes for six frustums. (click)
- Especially as the plane equations are all ones and zeroes, (click)
- Which means that if the loop is unrolled, most of this code just goes away!
- I think this is a rather big advantage with cube maps, over using separate frustums (As done in [6]).
- This efficiency is especially important given that we will be culling a lot more objects than normal culling!
- How so, I hear you say?
- Glad you asked!



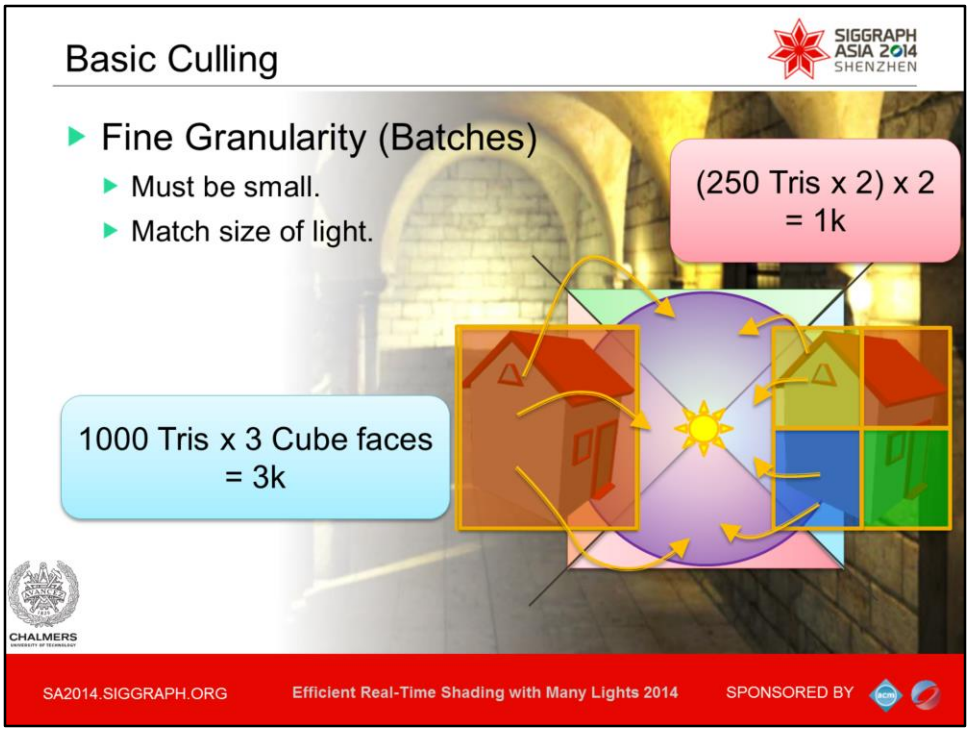
- You see, to enable effective culling, as in getting rid of a large proportion of the triangles, batches must be small,
- Intuitively, for any culling operation, the optimal size of batches correlates to the size of the frustums.



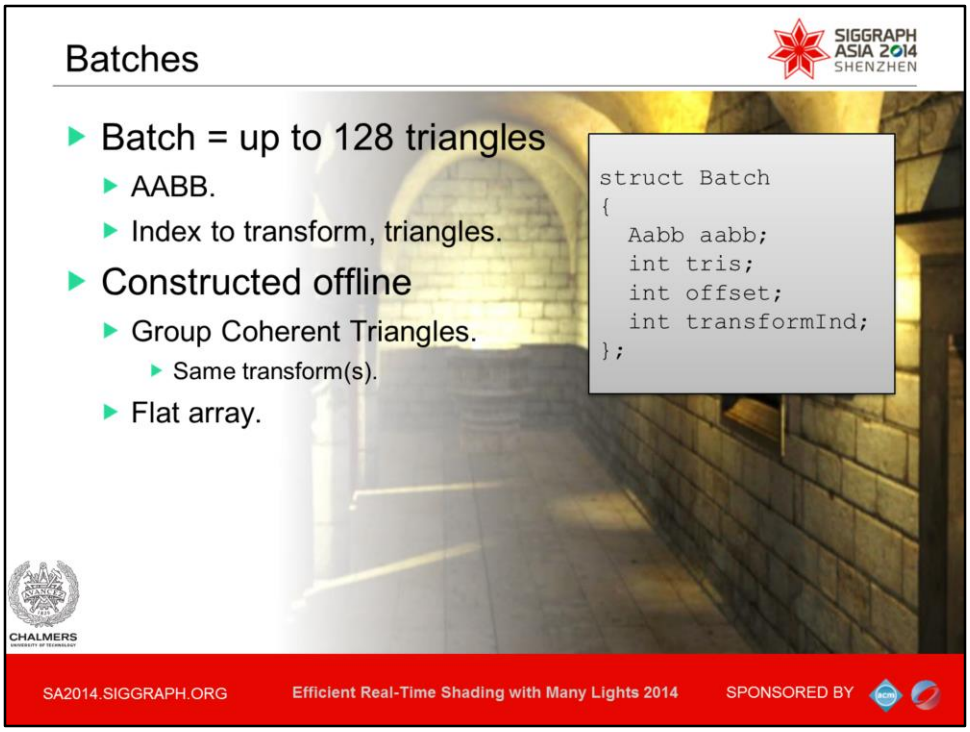
- If batches are too large, the triangles get replicated into most cube faces replicated.



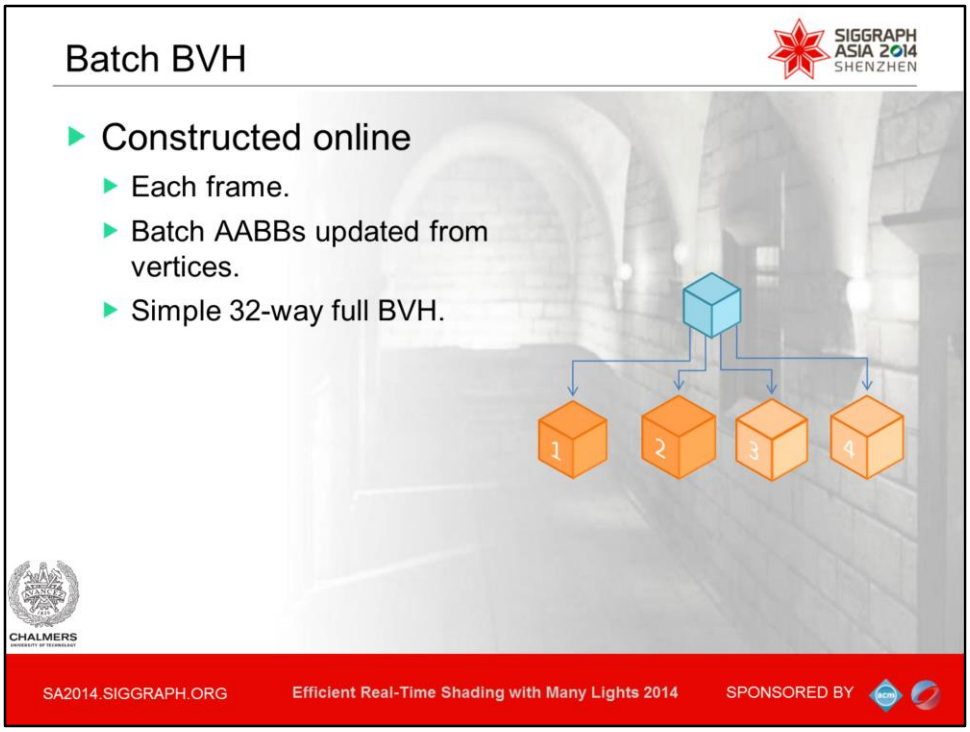
- Smaller batches, enables



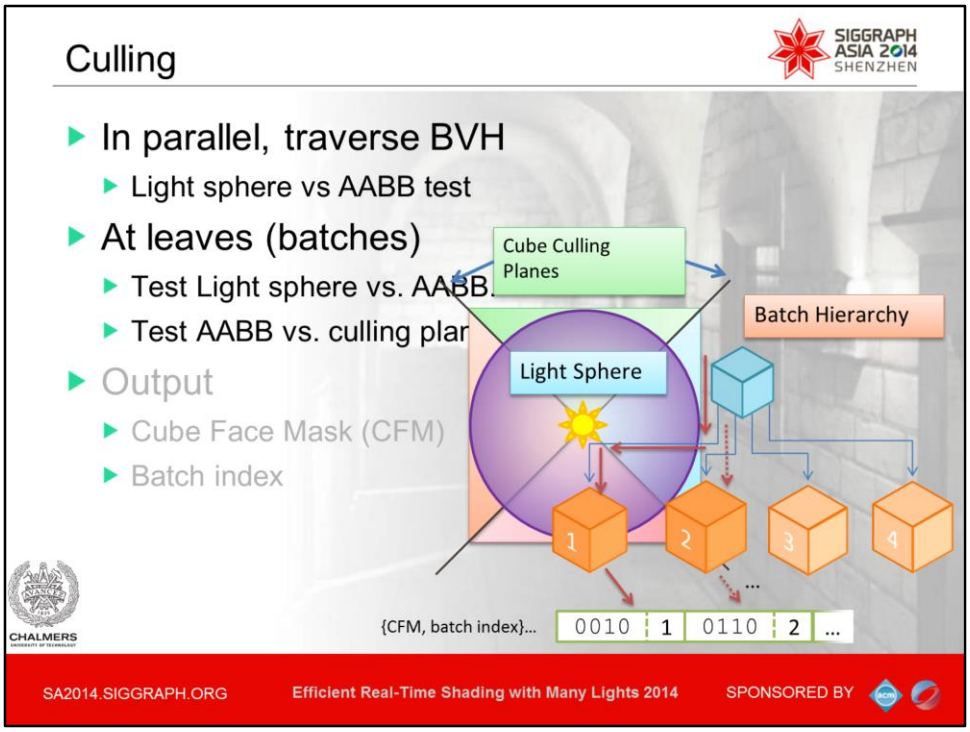
- More precise culling, and thus fewer triangles drawn.
- So we are trading increased culling work for fewer triangles drawn.
- Triangle drawing is the biggest performance bottleneck so this is important to be able to tune.



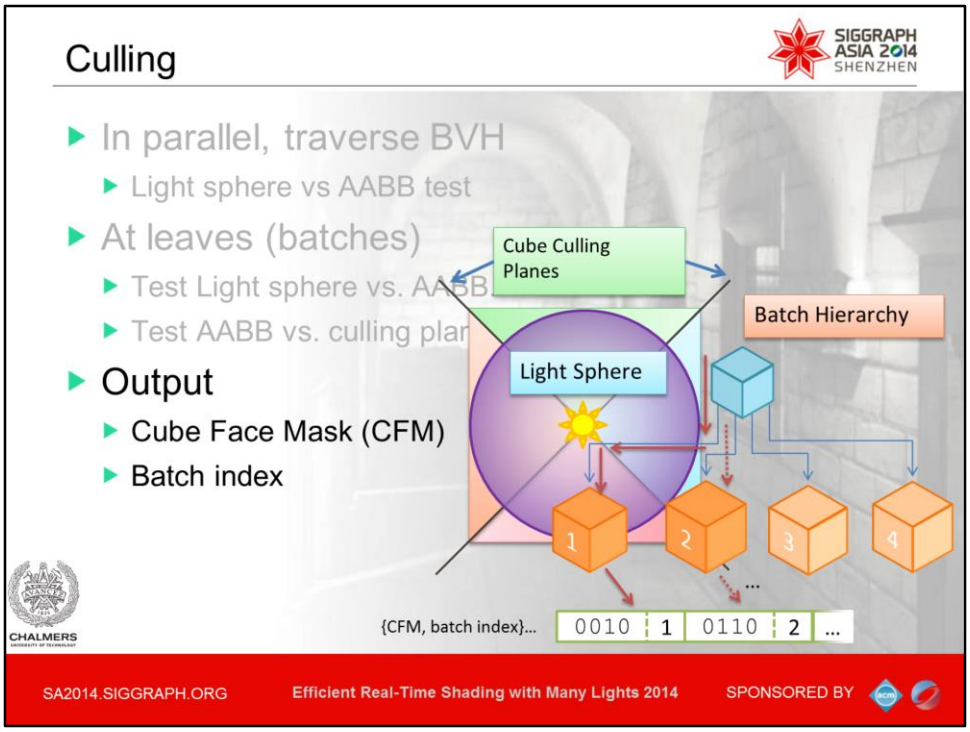
- We used batches of up to 128 triangles,
- in practice they average around 68 triangles.
- A batch is represented by an AABB and a list of triangles.
- The batches are constructed in a pre-process, that builds a tree using agglomerative clustering (see the paper section 6.2.1).
- Note that the quality of the batches is fairly important for good performance.
- The batches are stored in a flat array that is loaded into the runtime.



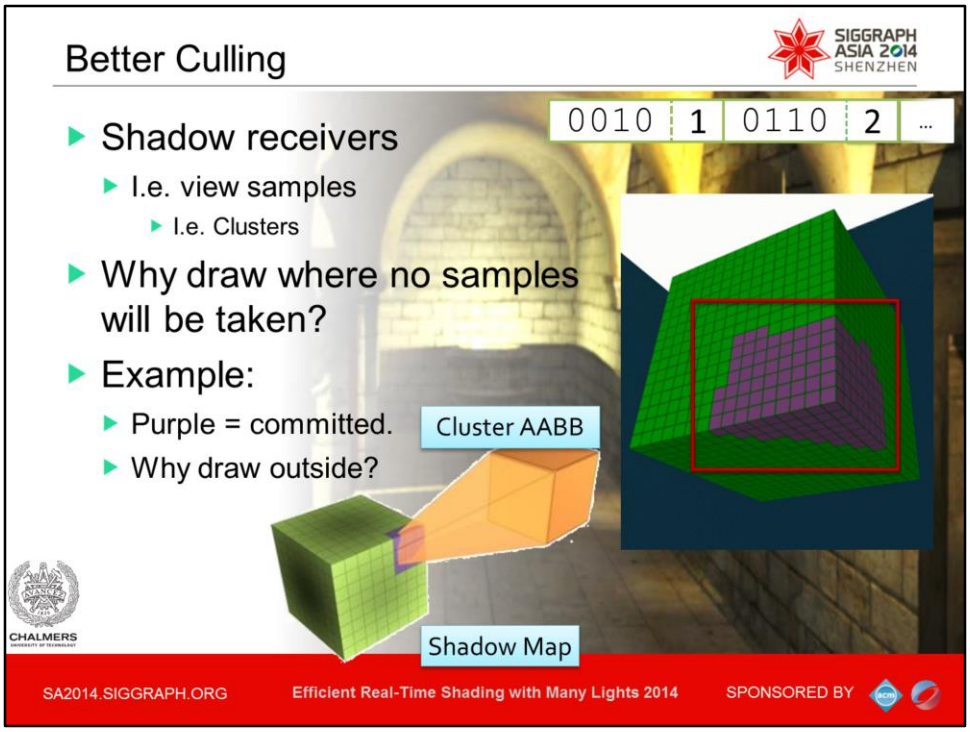
- For efficient culling, we obviously need a hierarchy.
- To this end, we used a very simple, full, 32-way BVH which is completely rebuilt each frame.
- There are more details on this structure in our papers, and even CUDA code online in the clustered forward demo.
- This is in no way the best possible structure, or even the fastest to build, but it has served us well enough.



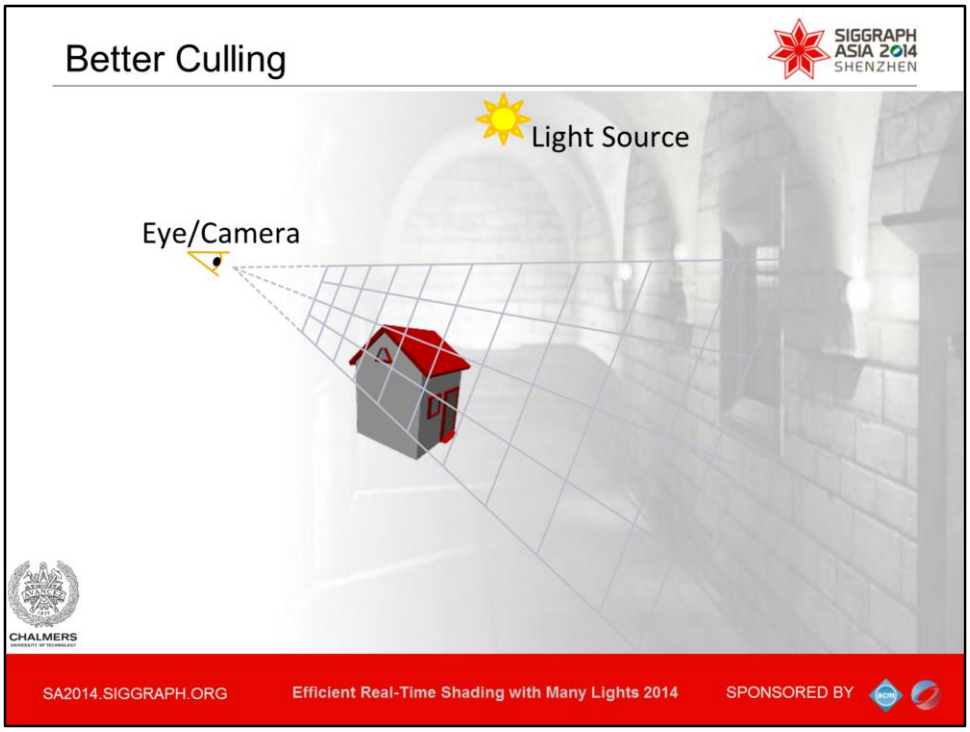
- To find the batches to draw each light traverses the hierarchy to find the batches that overlap the light sphere
- These batches are those that may produce a shadow if drawn into the shadow map.



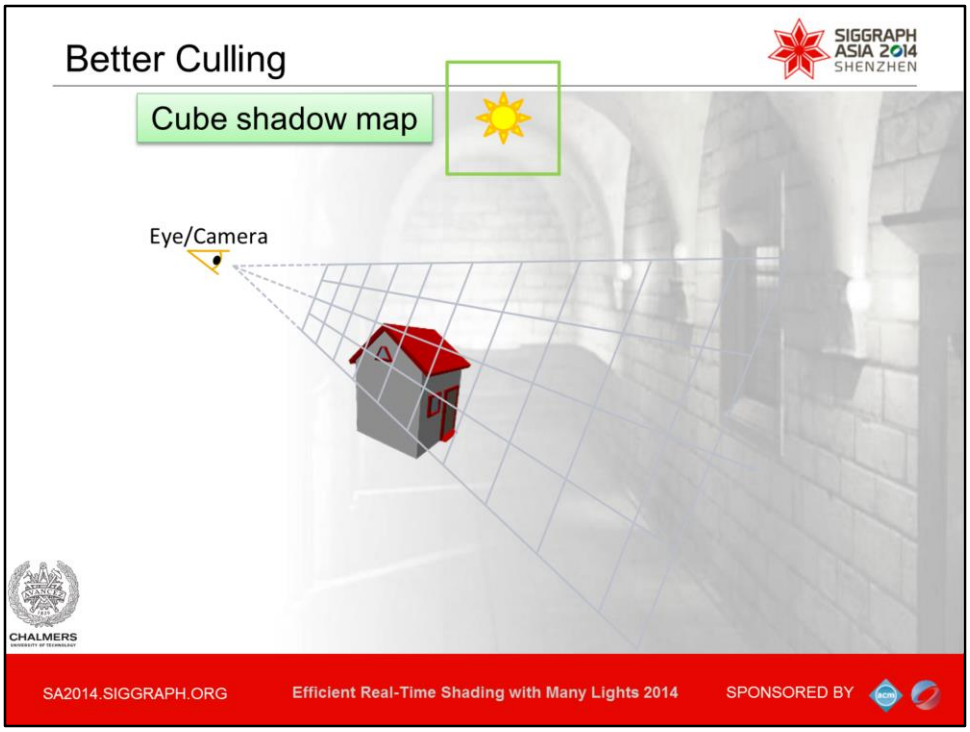
- This pass produces a list for each light of pairs of cube face masks and batch indexes
- The Cube Face Mask is a bit mask where each bit indicates if it overlaps a certain cube face that we saw earlier how to compute.
- This tells us what batches to draw to which cube faces of each light.



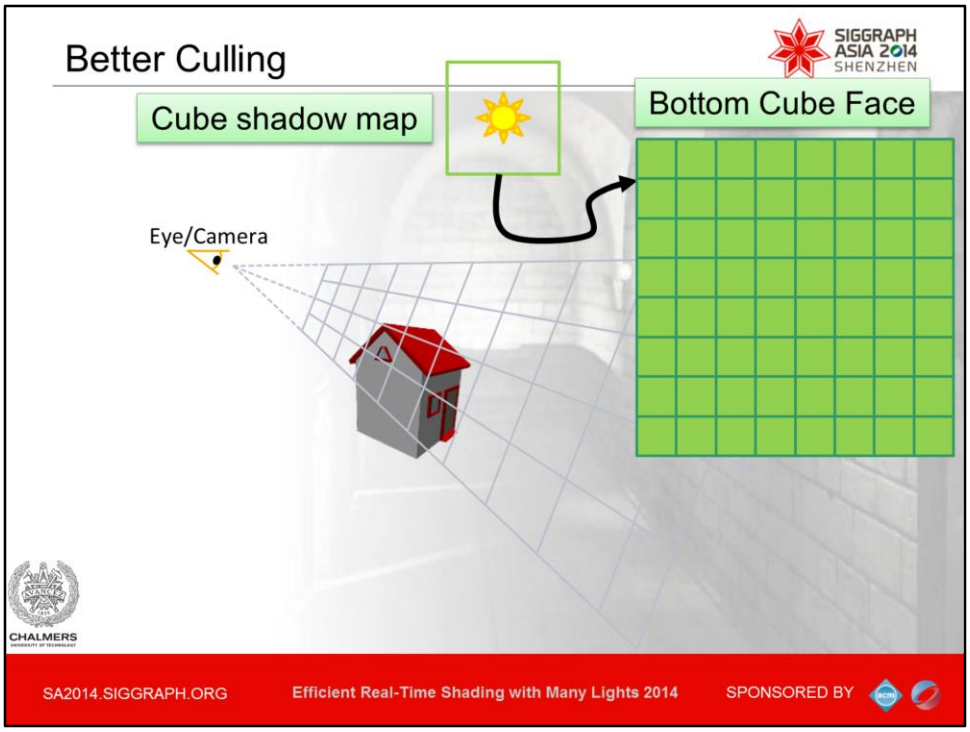
- Now, we also wish to take advantage of the sparsity of the shadow maps
- …to avoid drawing geometry to parts of the shadow map which will not be sampled, or indeed stored!
- So we re-use the technique for projecting the clusters to a bit grid for finding the needed pages,
- and do this for batches also, lets illustrate.



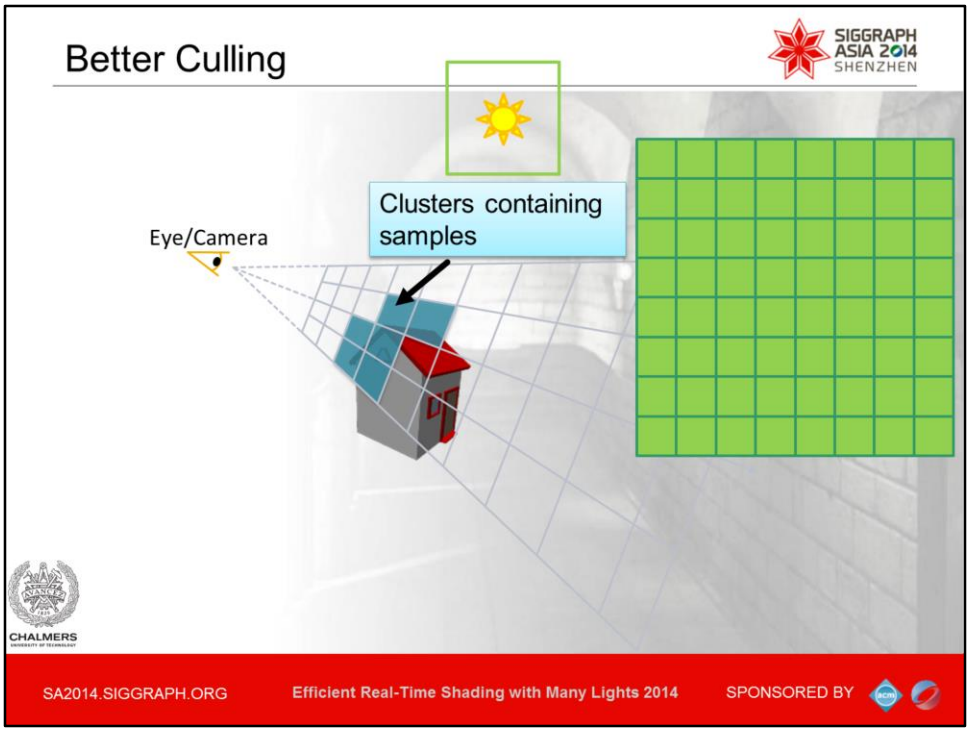
- In this example we have a camera, looking onto the little house, and an overhead light source



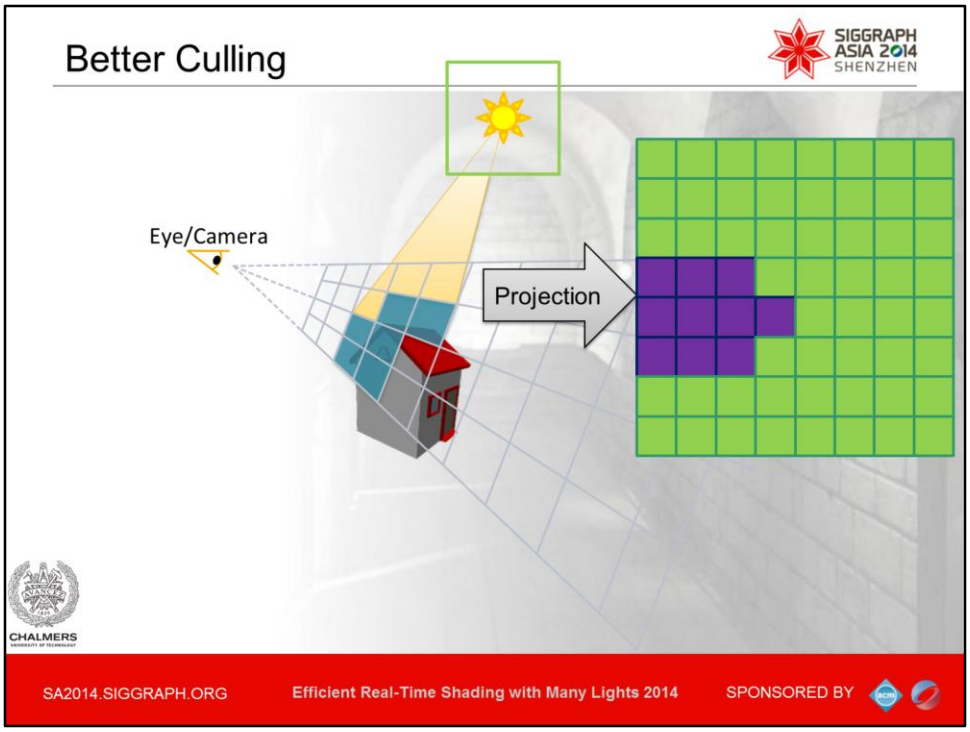
- This box represents the cube shadow map for the light.



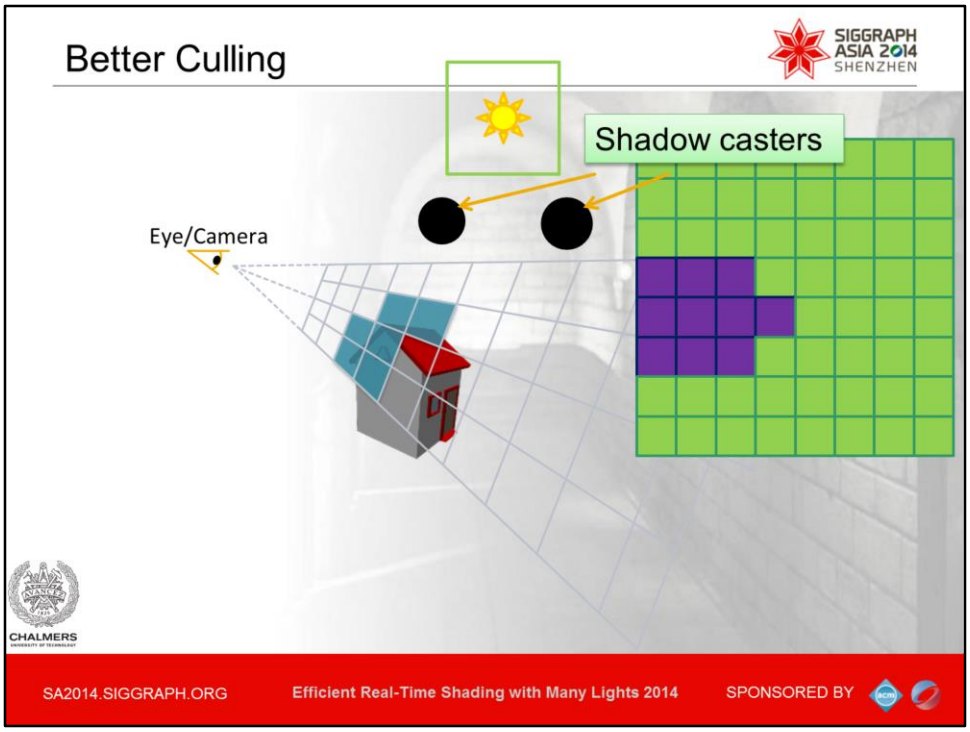
- This box represents the cube shadow map for the light.



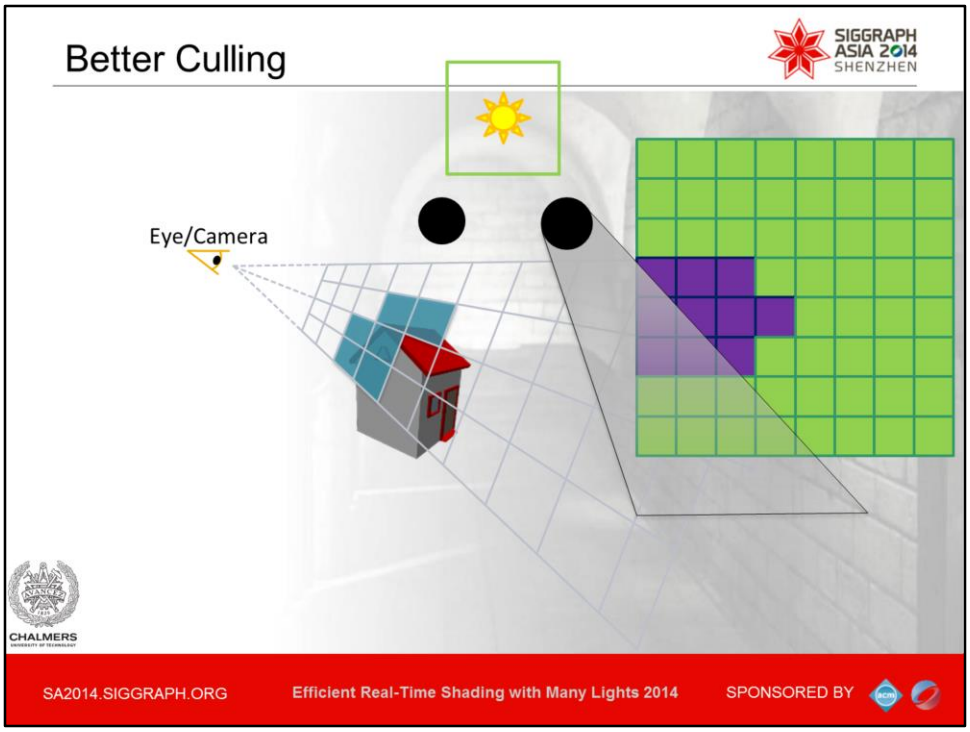
- Clusters that contain visible samples are shown here,
- They represent the sample points that shadow may be queried for, or the shadow recievers



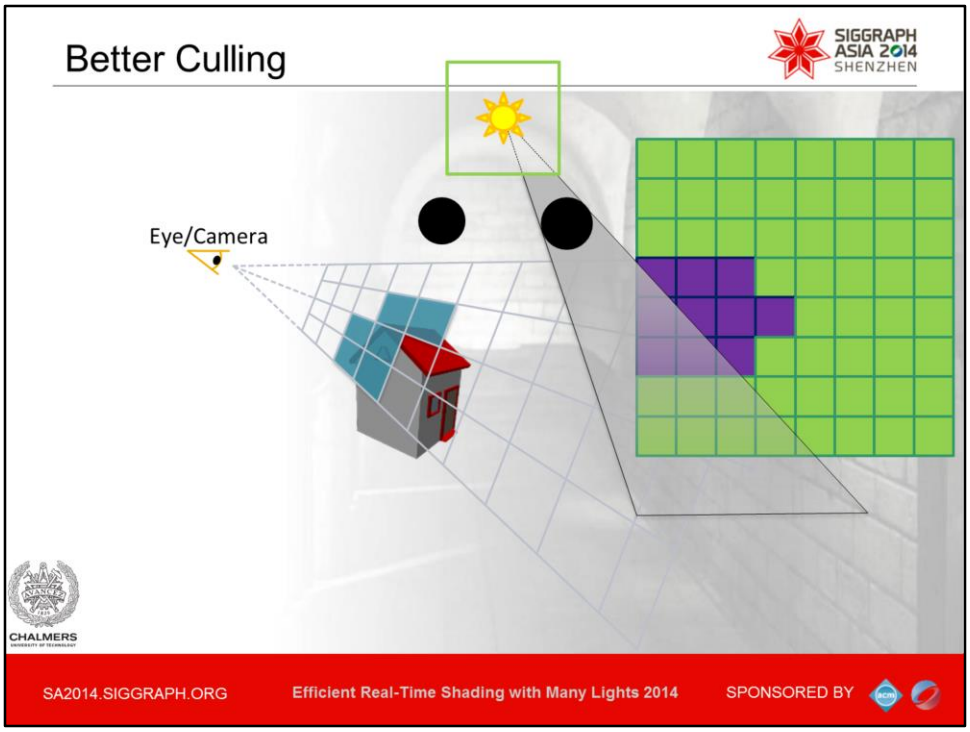
- And this is their projection onto the cube face



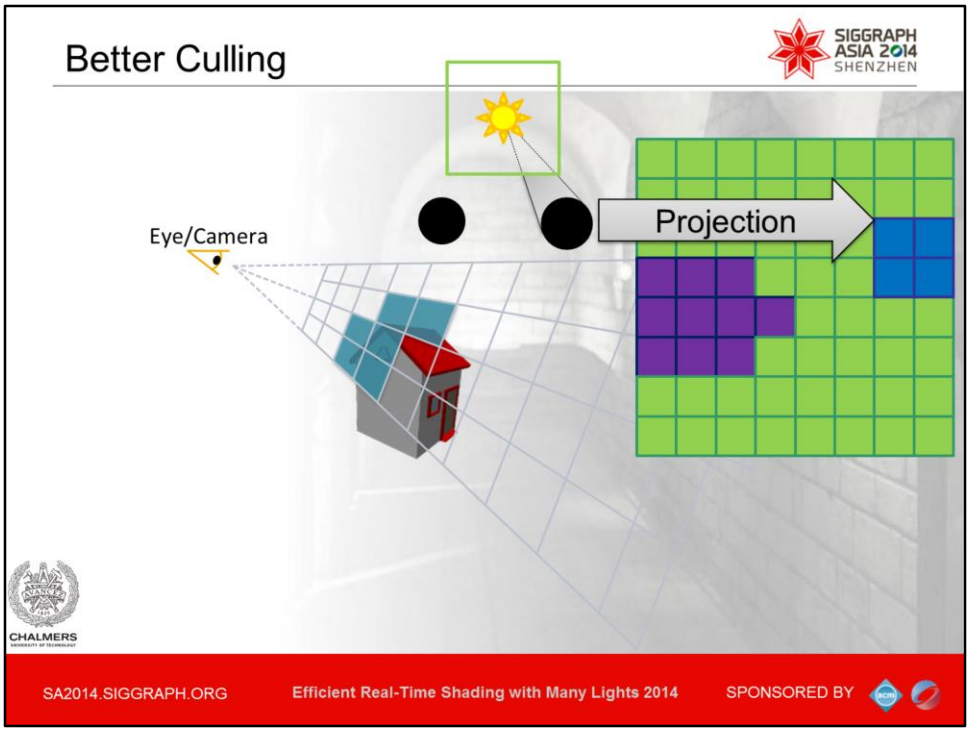
- Here are two shadow casters.
- Note that they are outside of the view volume, and so have no clusters associated…



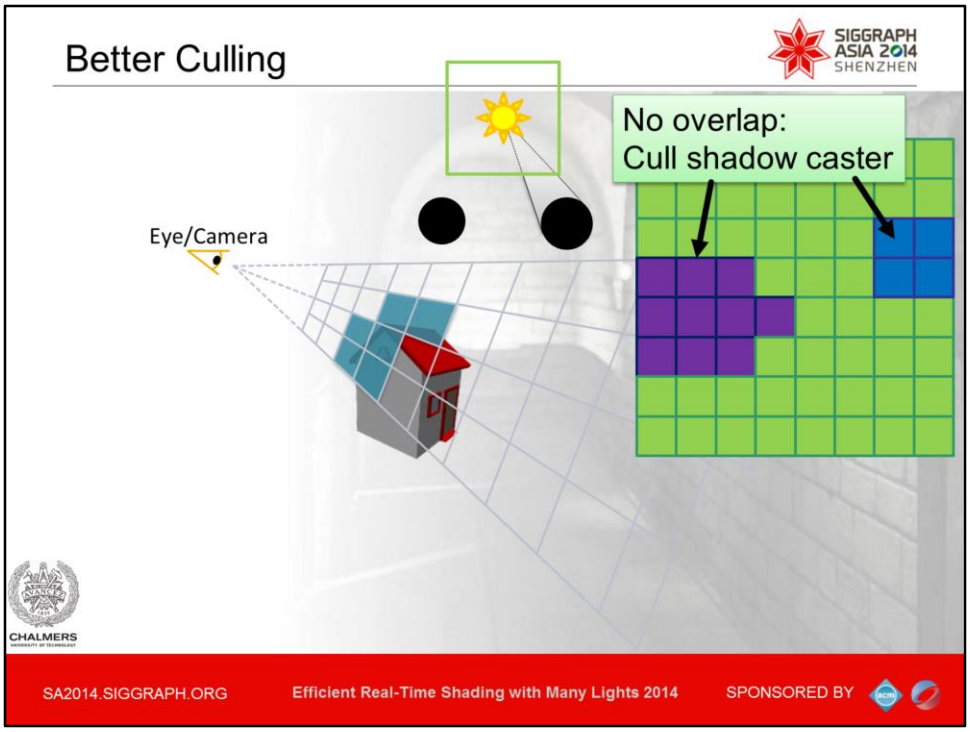
- But cast their shadow through the view volume.
- So we're interested in finding out what shadows affect the visible samples,
- and thus determine if the shadow caster need to be drawn.



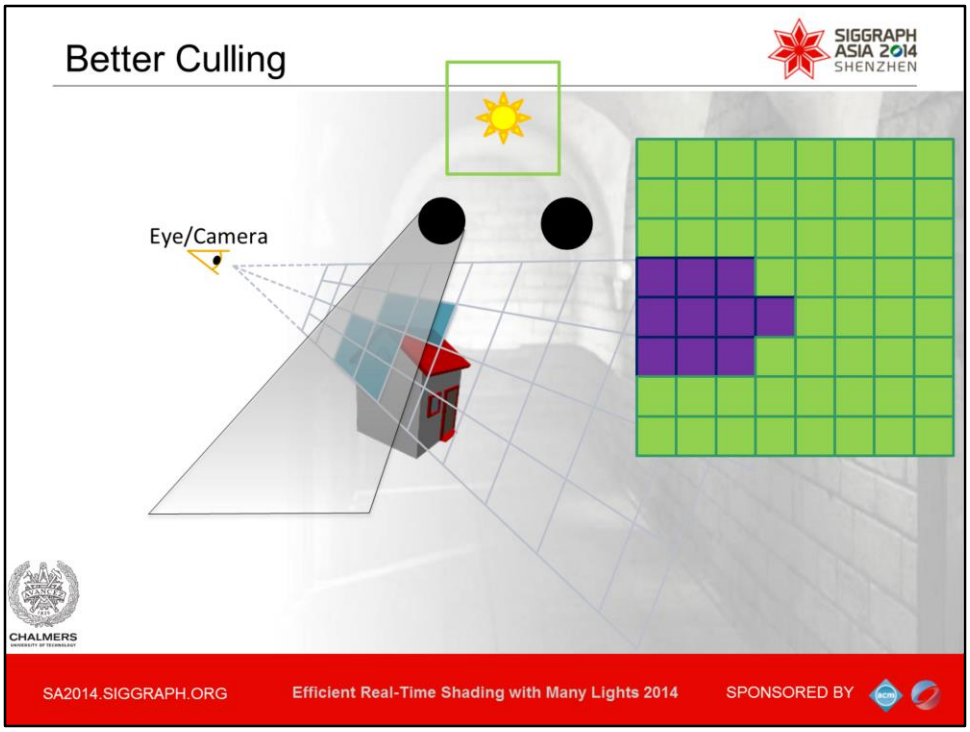
- We can figure this out by projecting the shadow caster onto the light source


Better Culling
SIGGRAPH ASIA 2014 SHENZHEN
Eye/Camera
Projection
CHALMERS
SA2014.SIGGRAPH.ORG
Efficient Real-Time Shading with Many Lights 2014
SPONSORED BY

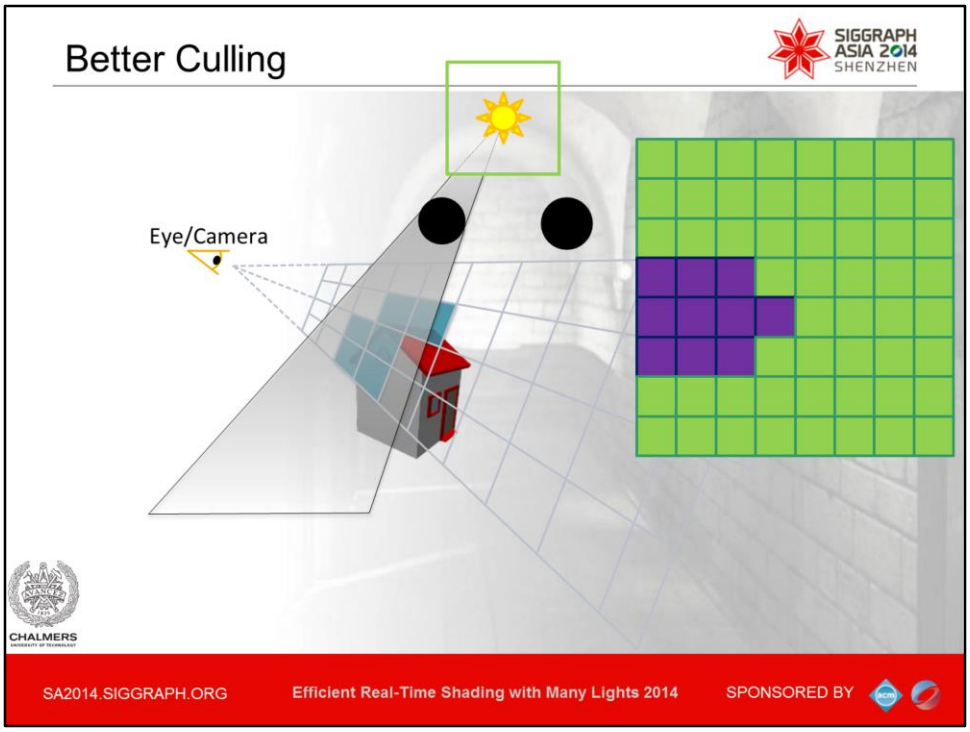
- Like so.



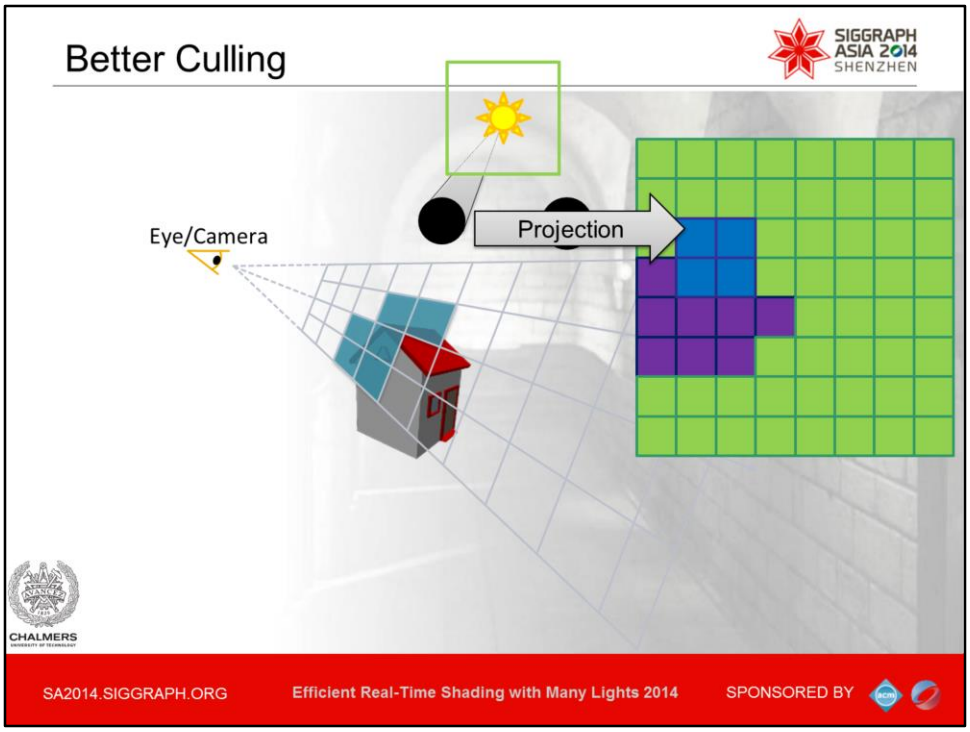
- We then test the projection of the batch against the projection map
- And since there is no overlap, we can discard the shadow caster.



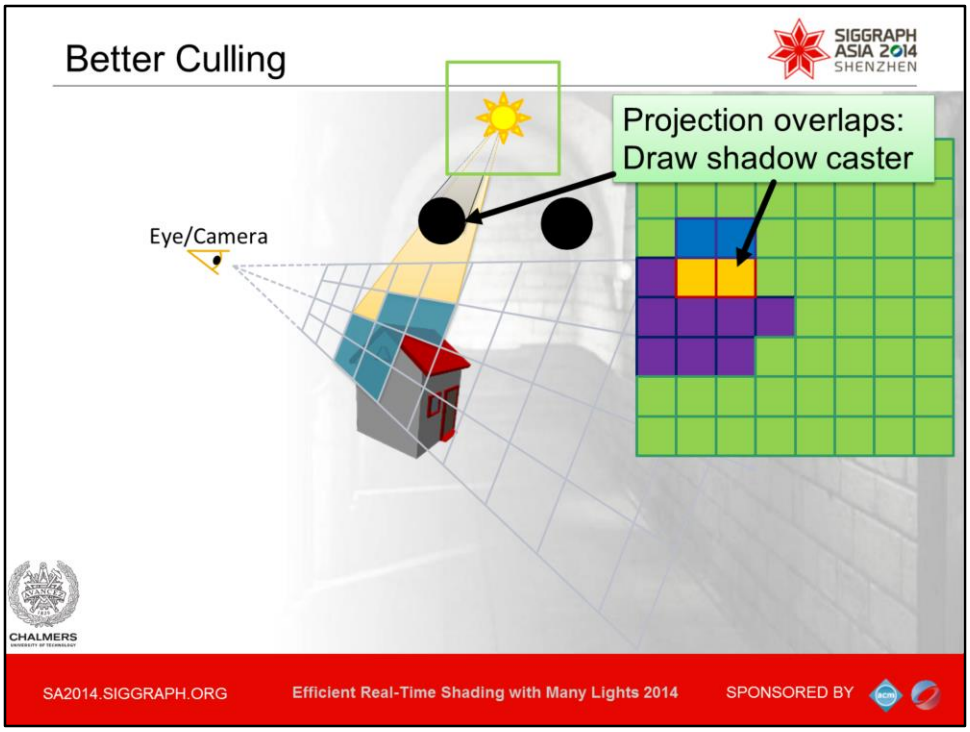
- The other shadow caster



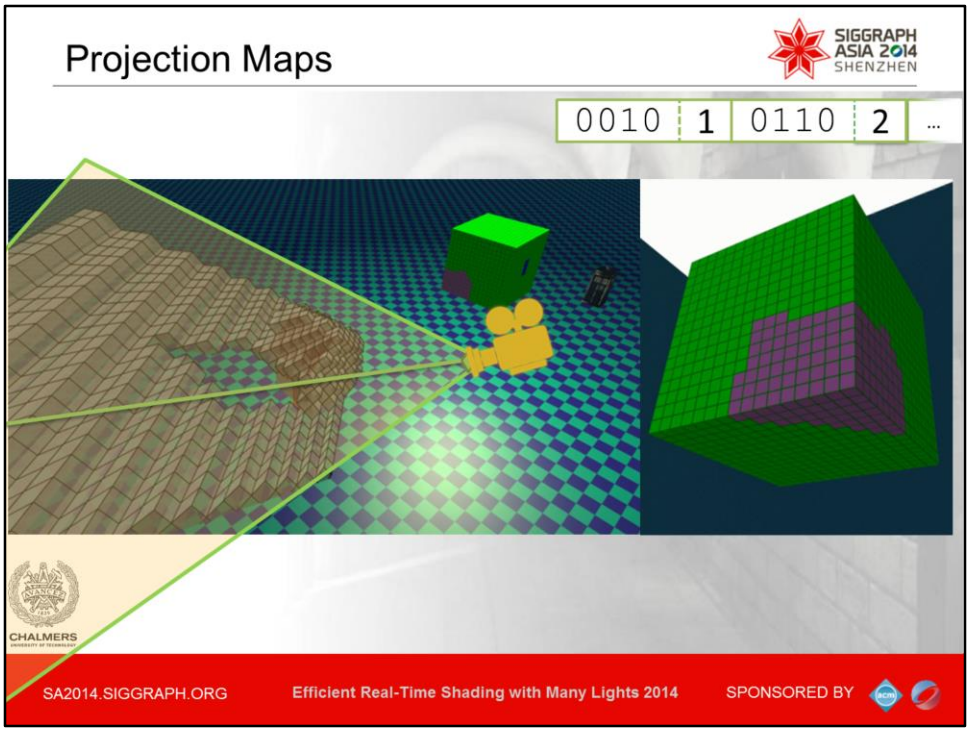
- Projected likewise…



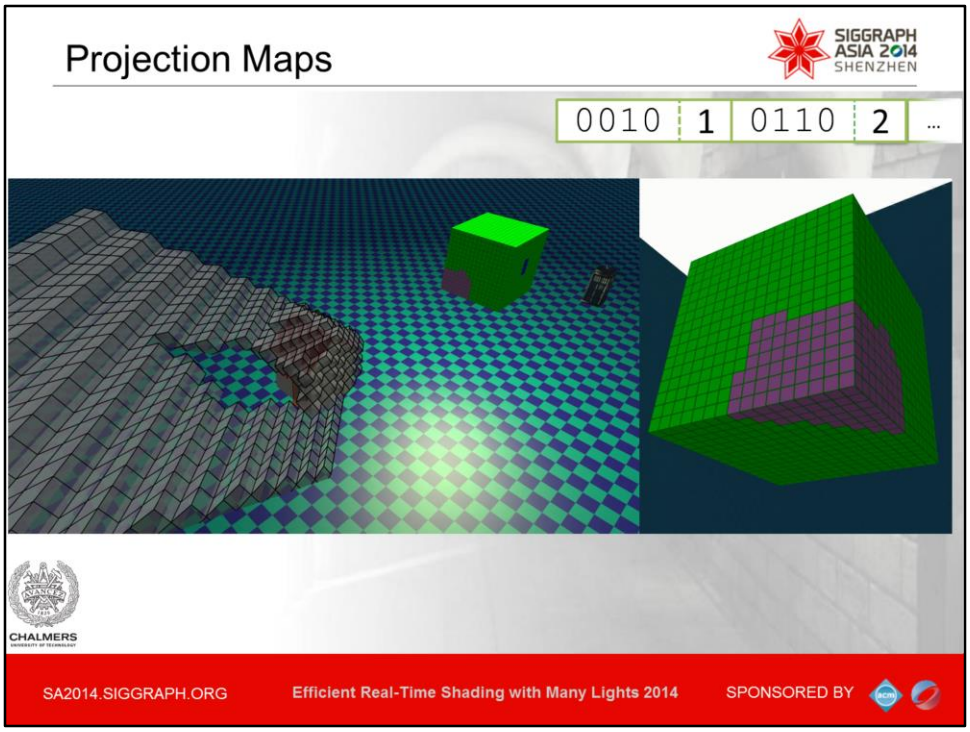
- We get this projection,



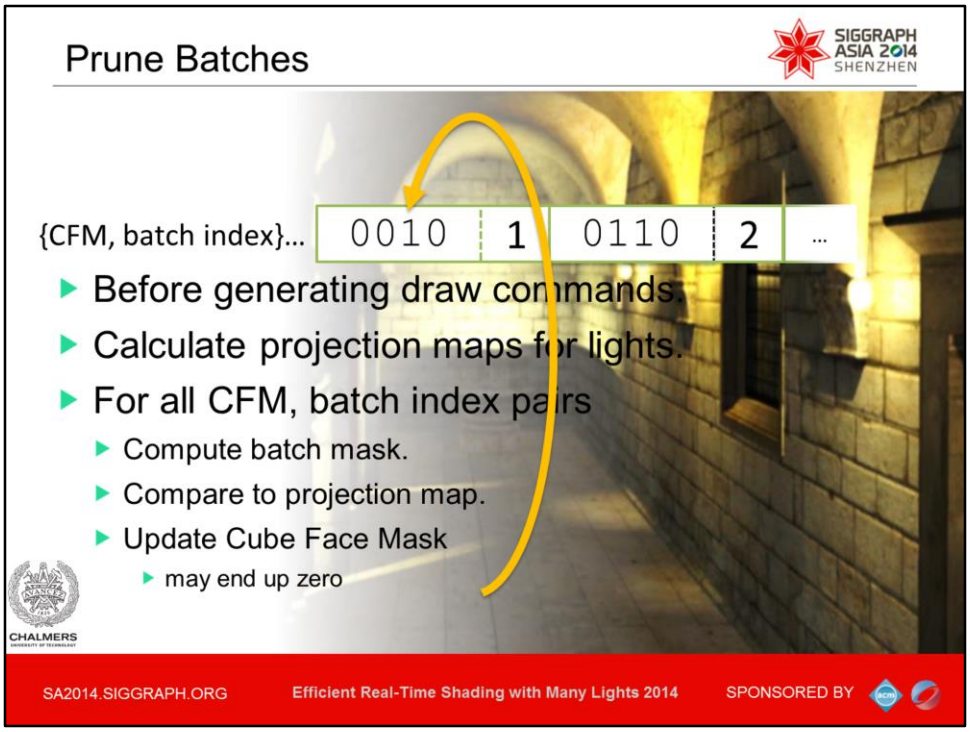
- This time there is overlap, so the shadow caster must be drawn into this cube face.



- Here we show this in action, where the tardis is the only batch, for illustration.
- The clusters from the house are represented in the projection map in purple.



- When there is yellow there is overlap, and this is the only time the batch need to be drawn.
- So as we see, most of the time the tardis can be culled, despite always being in the field of view of at least one cube face.

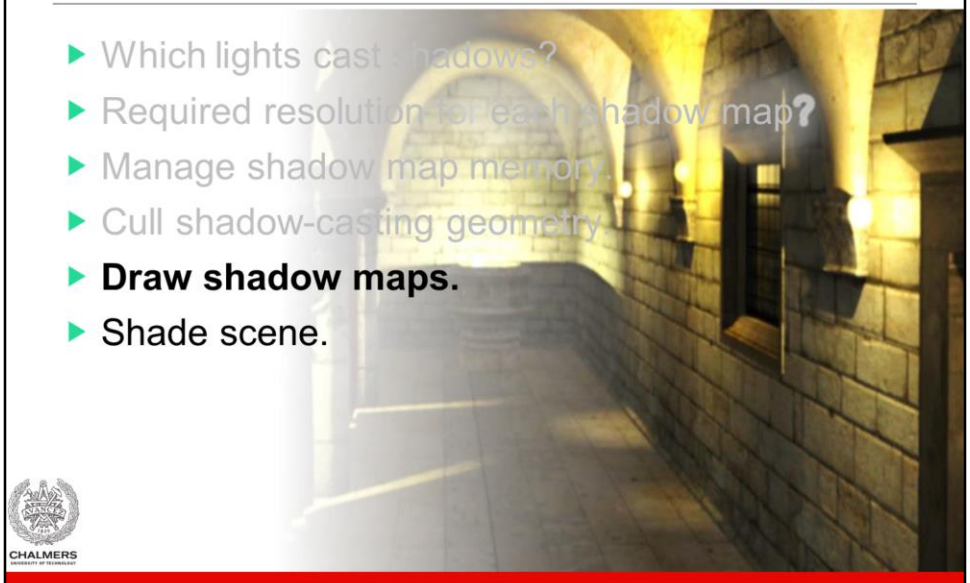


- We implement this with a pass over all the results from the previous step.
- The kernel just loads the light and cluster, and computes the projection for the cluster
- For any face where there is no overlap, the cube face mask bit is cleared, meaning it will not be drawn.
- Finally the updated CFMs are stored back, now usually with fewer bits set, and sometimes zero.

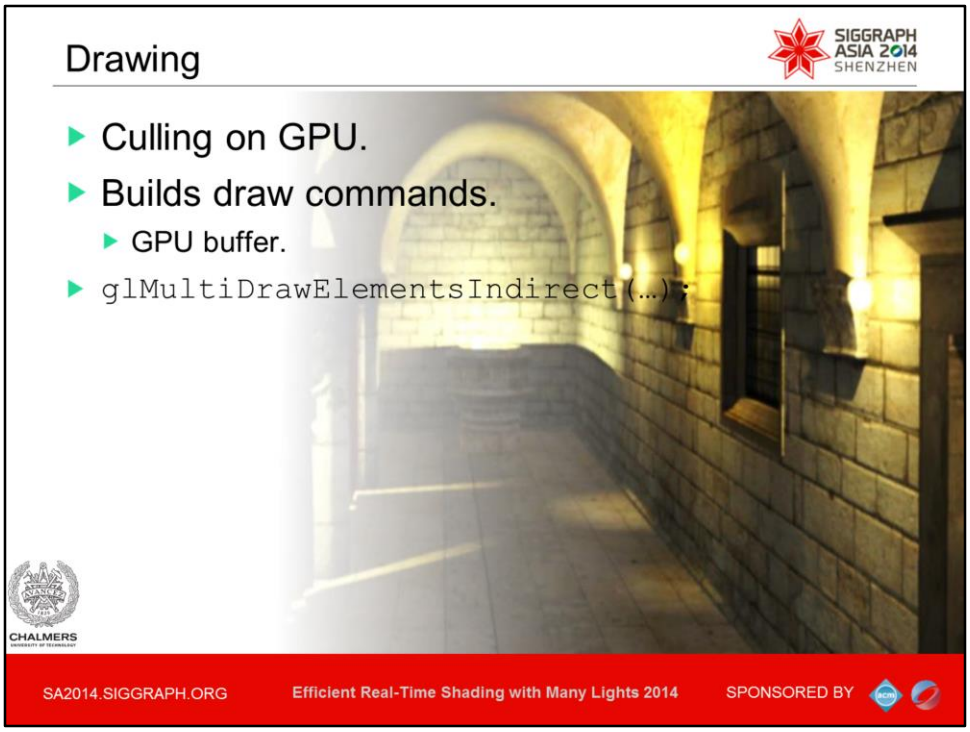
Problem breakdown

- Which lights cast shadows?
- Required resolution for each shadow map?
- Manage shadow map memory.
- **Cull shadow-casting geometry.**
- Draw shadow maps.
- Shade scene.

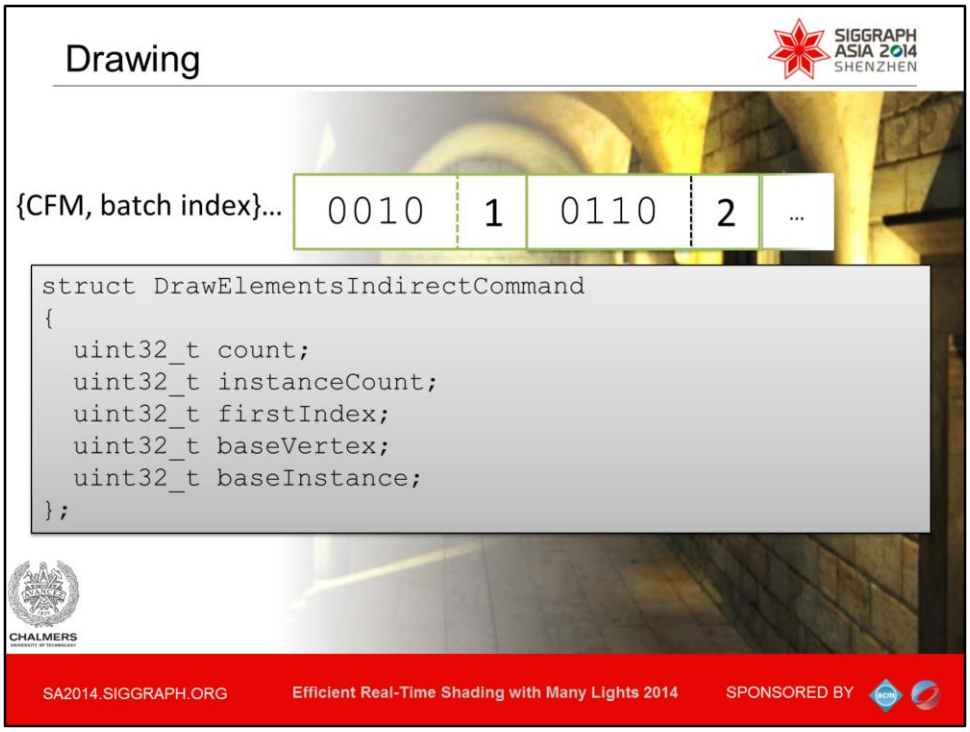
Problem breakdown



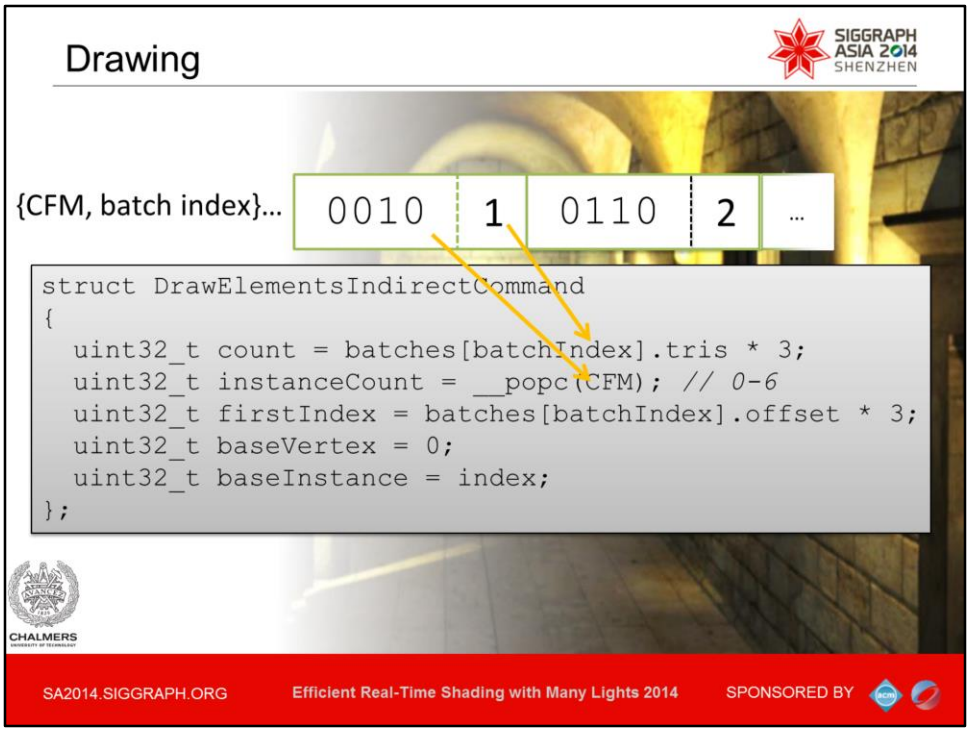
- Which lights cast shadows?
- Required resolution for each shadow map?
- Manage shadow map memory.
- Cull shadow-casting geometry.
- **Draw shadow maps.**
- Shade scene.



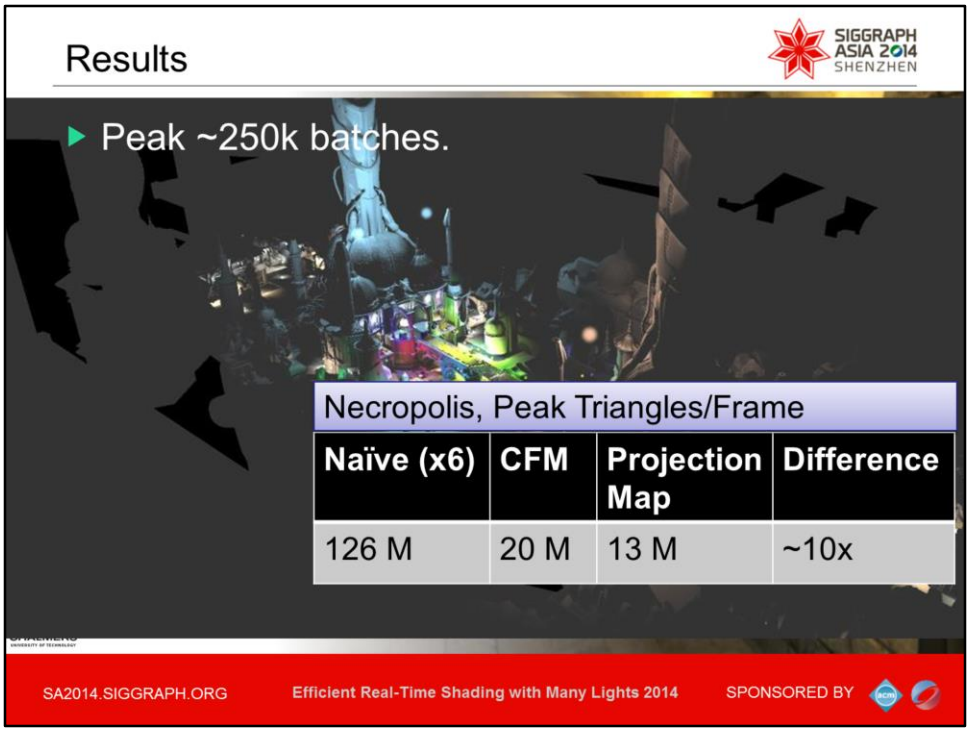
- As all the culling is performed on the GPU, using CUDA, make use of modern OpenGL and build draw commands also on the GPU
- Then we just call multi draw indirect once for each cube map, though it could be done once for all in principle.



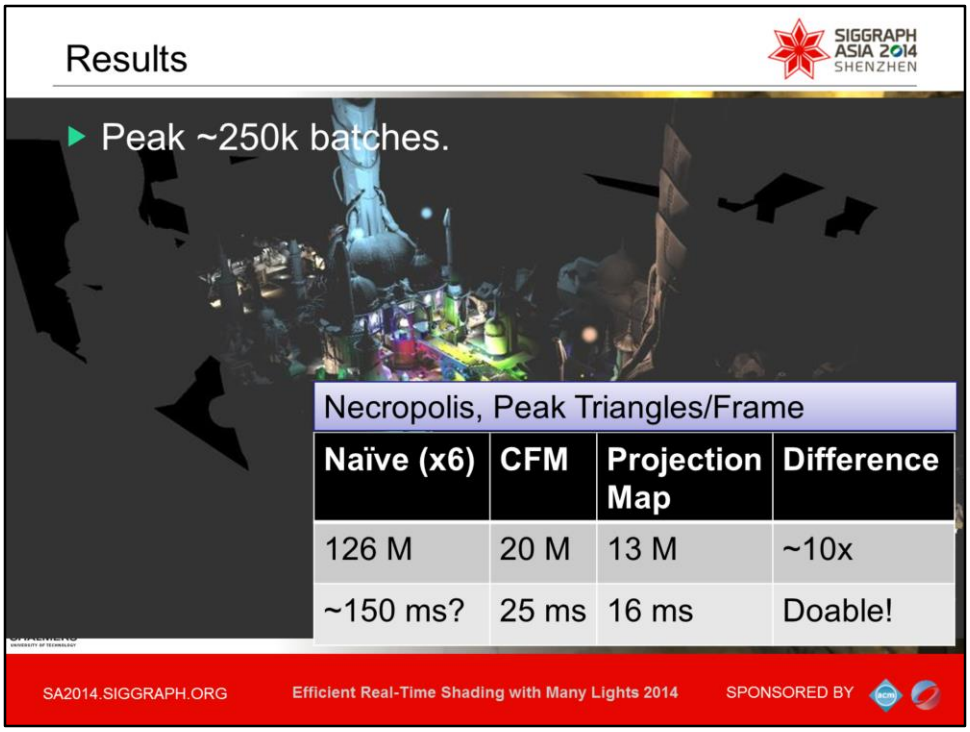
- The draw commands look like this,
- and are built from the Cube Face Mask and the Batch index



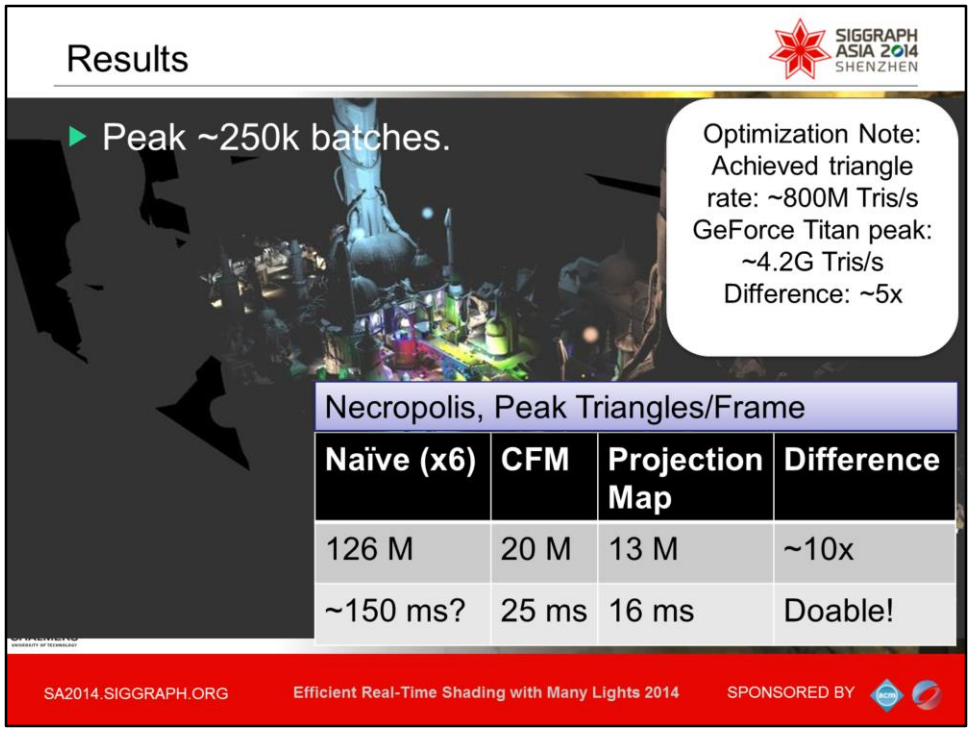
- Note that the number of bits set in the cube face mask correspond to the number of instances drawn
- We use instancing instead of duplication in a geometry shader to draw such batches that overlap multiple cube faces.
- A geometry shader uses the bit mask (through a per-instance attribute) and the instance index to route the batch to the correct cube face.
- The result is very low CPU overhead for the drawing, and good GPU performance.
- When the instance count is zero, nothing is drawn for that command, this is legal OpenGL and we found it to not impact performance much, so we did not find it worthwhile to insert a compaction step to get rid of them.



- At the peak, we draw around 250 thousand batches into the shadow maps, with instancing on top.
- Using the culling techniques just outlined, this results in some 13 million triangles
- which is almost 10 times better than the naïve approach of replicating the triangles to all cube faces.
- And some 65% of performing culling just using the cube face mask.
- The cube face mask path actually already detects completely empty cube faces, and so is already quite effective.

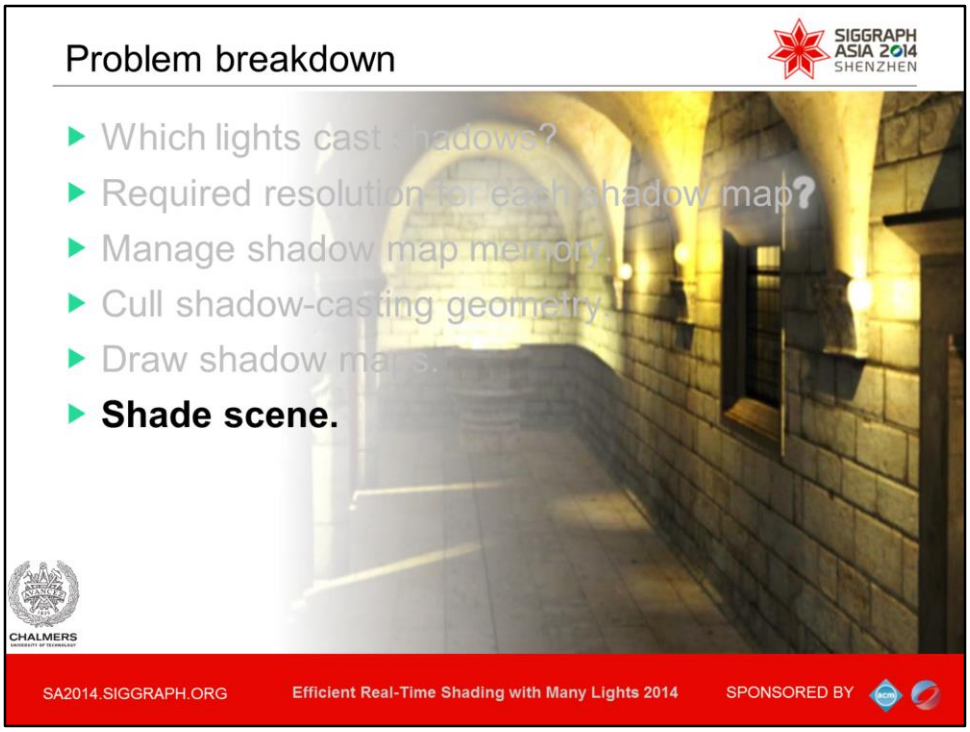


- Again, put differently, this is the difference between infeasible and something we can do in real time.



- As a side note, our implementation achieved a fifth of the theoretical peak triangle rate
- So there could be some pretty good improvements for the handy optimizer out there.

- We now have all the components needed to shade the scene.

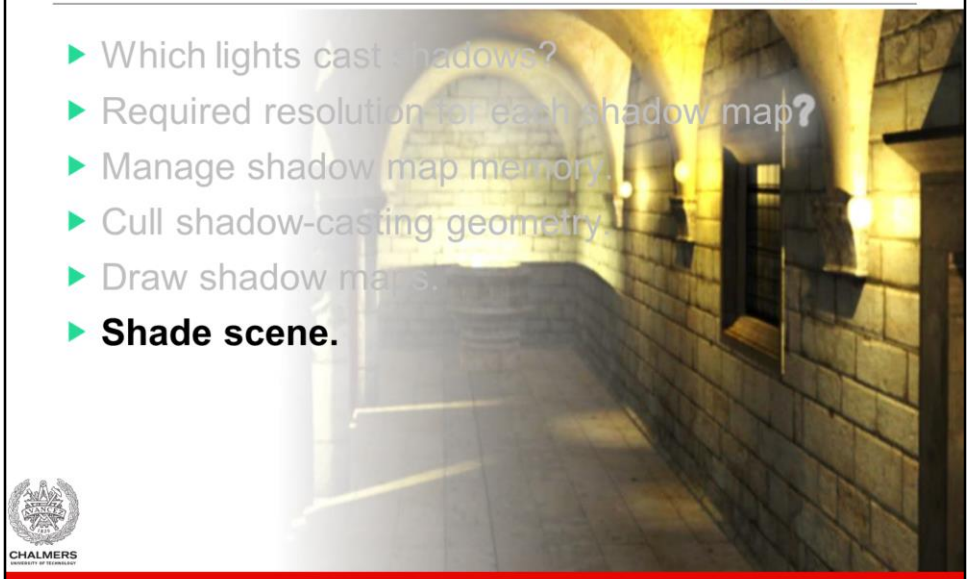


- The shader, be it forward or deferred, just loops over the lights as usual,
- Using the light index it can look up the shadow map (using bindless textures or an array texture or something) and sample it.
- With that, we come to the boring part of the show…

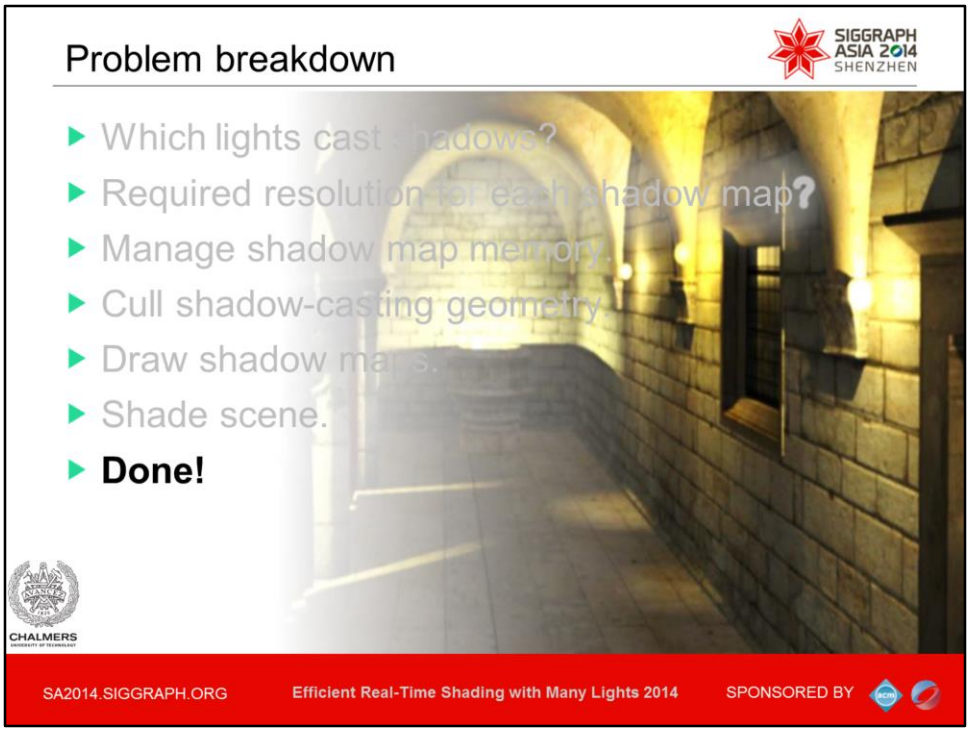
Shading (2014 edition)

- Loop over lights
  - Light index <-> shadow map index
- Calculate cube face index from direction
  - GCN: cubeFaceIndexAMD[4].
- Calculate cube face coords. from direction
- Sample 2D layer in shadow map
- Note:
  - Not using bindless textures any more (As in I3D paper)
  - Reason:
    - 2X performance increase in shading.
    - Conformant to ARB Spec.

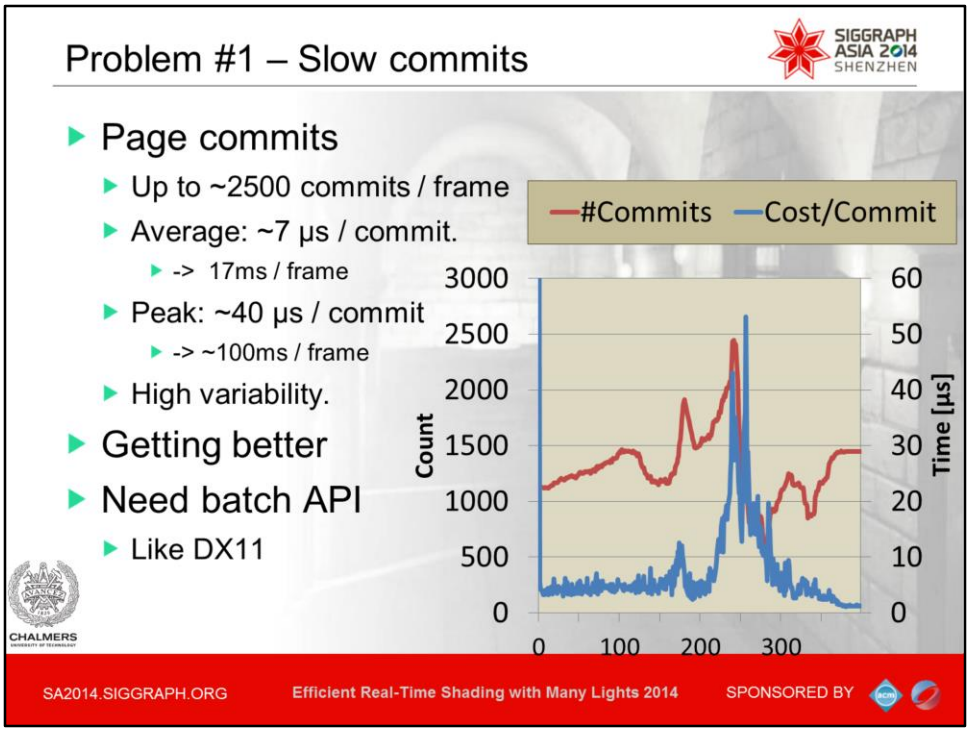
Problem breakdown



- Which lights cast shadows?
- Required resolution for each shadow map?
- Manage shadow map memory.
- Cull shadow-casting geometry.
- Draw shadow maps.
- **Shade scene.**

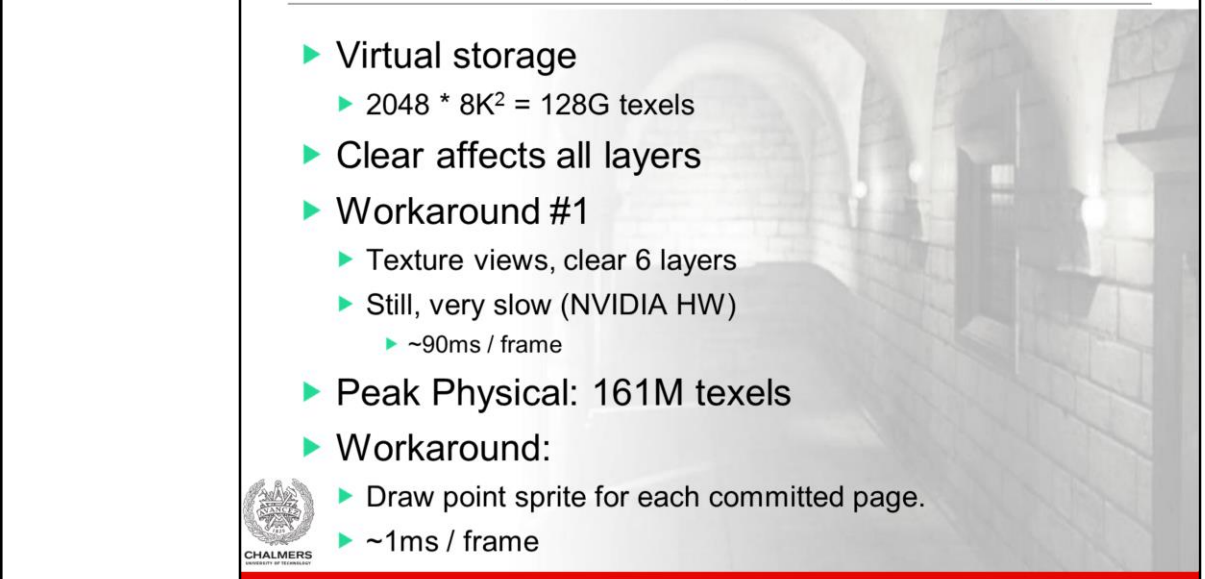


- So we're done… well almost.

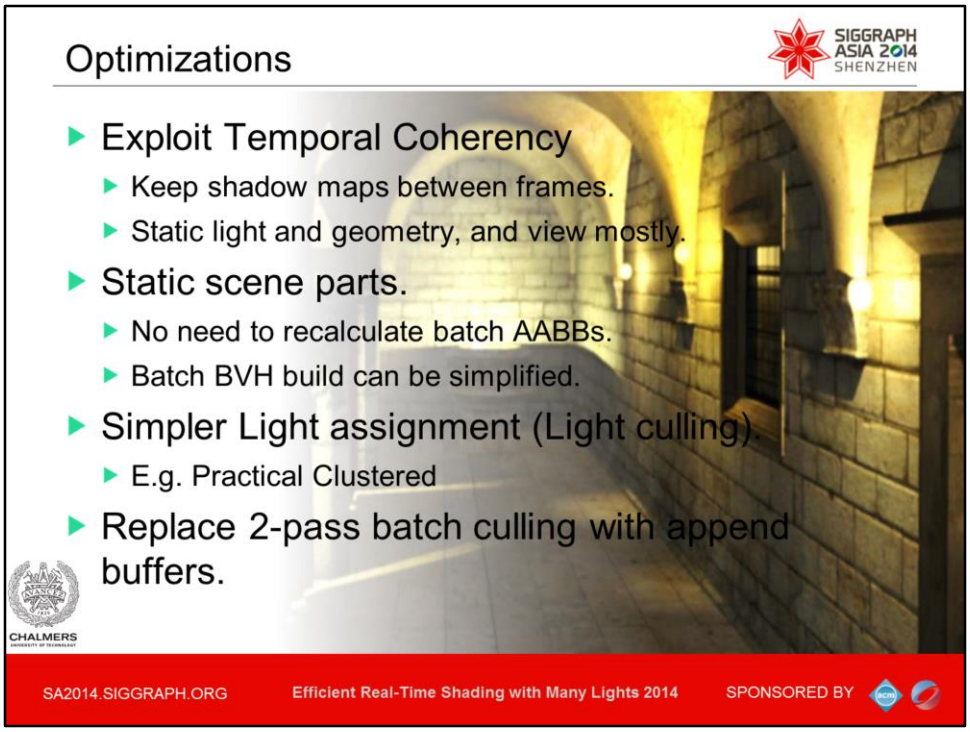


- I have to bring a couple of issues to your attention.
- The first, that page commits are slow, has gotten a lot better.
- It used to be a couple of milliseconds per commit.
- However, while it can be usable for real-time performance today, there is still a high variability in the cost.

Problem #2 – Clearing large array textures



- Virtual storage
  - 2048 * $8K^2$ = 128G texels
- Clear affects all layers
- Workaround #1
  - Texture views, clear 6 layers
  - Still, very slow (NVIDIA HW)
    - ~90ms / frame
- Peak Physical: 161M texels
- Workaround:
  - Draw point sprite for each committed page.
  - ~1ms / frame



- There are quite a few fairly obvious ways we can improve performance, but which we did not do for the paper.
- The main thing is that we assumed all geometry and all lights were moving, every frame.
- We did this to ensure we were measuring performance for the paper on the most difficult case, as opposed to report at best-case scenario.
- In practice a lot of stuff is going to be static, which means that there is a lot of room to exploit static scene elements, in different ways.
- In the upcoming update of the paper, we do explore this, but this did not make it into the presentation in time. But it does show promise!
- We reused our hierarchical light assignment code from previous papers, but it is really dimensioned for huge numbers of lights and much simpler code would be much faster (e.g., Emils design).
- There are also more low level opportunities.

Further reading.

[1] My own publications and demos. http://www.cse.chalmers.se/~olaolss

[2] Forward+ related info, Harada et al.
http://scholar.google.com/citations?user=4R7xOcsAAAAJ&hl=en

[3] Karras et Aila, HPG 2013, *Fast Parallel Construction of High-Quality Bounding Volume Hierarchies*

[4] OpenGL extension registry, `ARB_sparse_texture`, `ARB_bindless_texture`, `AMD_gcn_shader`

[5] Coombes, GDC 2014, *Taking Advantage of DirectX11.2 Tiled Resources*

[6] Forsyth, GDC 2004, Multiple Shadow frustra
http://home.comcast.net/~tom_forsyth/papers/shadowbuffer_pseudocode.html

[7] HOLLÄNDER, M., RITSCHEL, T., EISEMANN, E., AND BOUBEKEUR, T. 2011. ManyLoDs: parallel many-view levelof- detail selection for real-time global illumination. Computer Graphics Forum 30, 4, 1233–1240.

[8] Wolfgang Stürzlinger and Rui Bastos. Interactive rendering of globally illuminated glossy scenes. In Proc. of the Eurographics Workshop on Rendering Techniques '97, pages 93–102. Springer-Verlag, 1997.

[9] Carsten Dachsbacher and Marc Stamminger. Splatting indirect illumination. In Proc. I3D '06, page 93–100. ACM, 2006.

[10] LEFOHN, A. E., SENGUPTA, S., AND OWENS, J. D. 2007. Resolution-matched shadow maps. ACM Trans. Graph. 26, 4 (Oct.).

[11] G. King and W. Newhall, "Efficient omnidirectional shadow maps," in ShaderX3: Advanced Rendering with DirectX and OpenGL