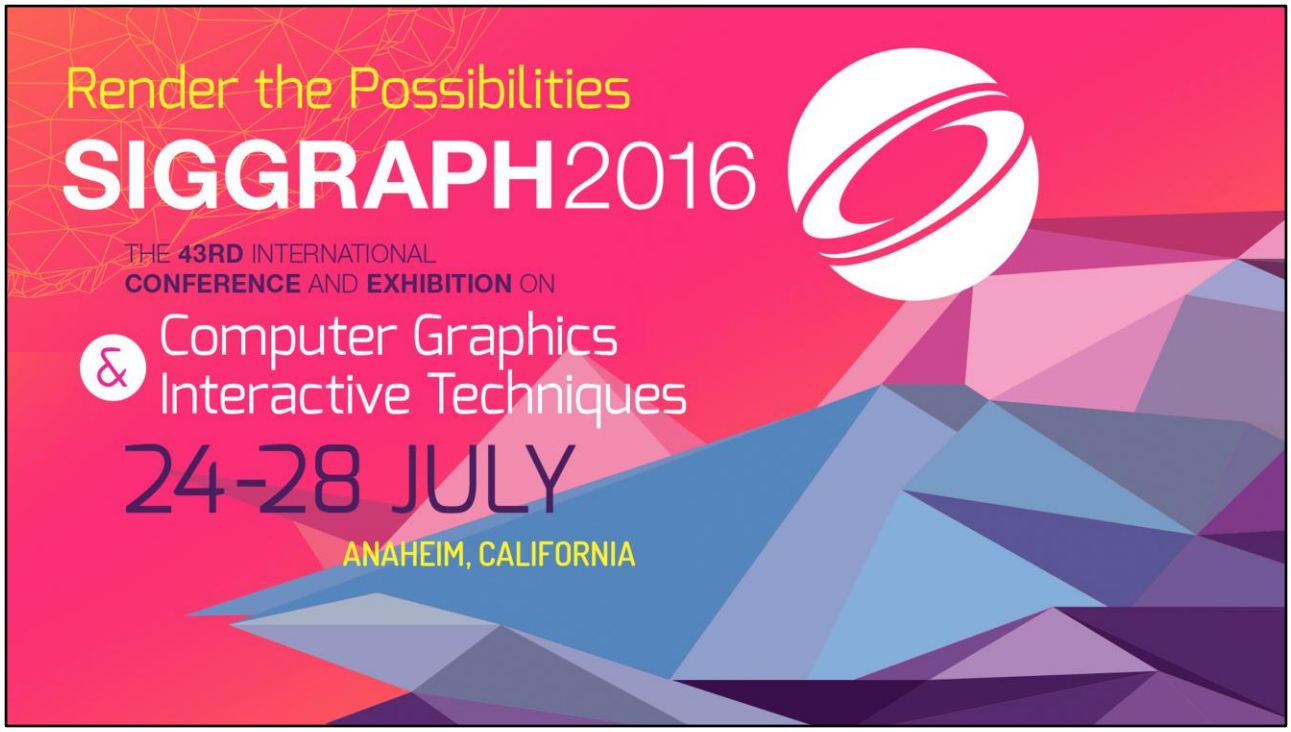

Render the Possibilities
SIGGRAPH2016
THE 43RD INTERNATIONAL
CONFERENCE AND EXHIBITION ON
&
Computer Graphics
Interactive Techniques
24-28 JULY
ANAHEIM, CALIFORNIA

My name is Marius Bjorge, and I'm from the ARM Trondheim office.

One of the main goals I set out for this talk was to explore modern shadow mapping techniques in mobile. I wanted to see how performance and quality scaled on the very latest mobile GPU hardware.

Agenda

- Shadow algorithms
- Shadow filters
- Results

So in this talk I will talk a bit about the challenges with shadows in mobile graphics. I'll recap some of the existing shadow algorithms and how they can be efficiently implemented to run on mobile GPUs. After that I'll talk a bit about shadowmap filtering before showing some of the results I got out of this.

SHADOW ALGORITHMS

Shadow Algorithms

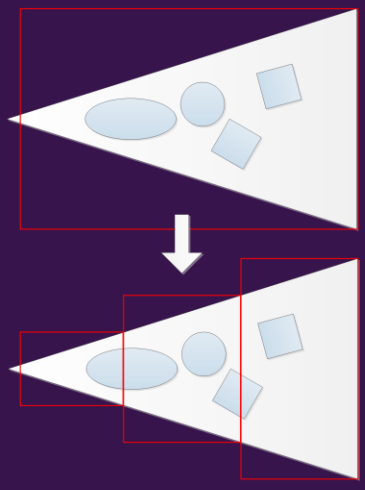
- Shadow mapping
- Depends on light type
  - Directional, spot light, point light, area light, etc…

In this talk I'll mainly focus on shadow mapping. Which particular shadow mapping algorithm you choose depends heavily on the light type you're associating it with. For instance, it really doesn't make sense to use cascaded shadow maps for something like a point light – you would much rather use a cubemap for that.

In this talk I've mainly focused on cascade shadow maps – which map very well to directional light sources.

Cascaded shadow maps

- Builds on regular shadow mapping
- Split visible depth range into N partitions
- Slice distribution
  - Logarithmic?
  - Uniform?



Cascaded shadow maps extend the concept of regular shadow maps. Instead of having a single shadowmap covering the entire camera frustum, we split it up into N number of partitions – each with a separate shadowmap associated with it.

One of the problems with regular shadow maps is that the resolution of the shadow map is poorly utilized. This can results in severe aliasing artifacts. So cascaded shadow maps is a way to combat that problem.

The concept is simple – the visible depth range, near to far, is split into N partitions. There are many ways of doing this, for my results I've used the logarithmic split distance. Often this might require tweaking in order to get just right for your scene.

Cascaded shadow maps

- Requires rendering geometry N times
  - Same geometry might be rendered multiple times into different shadow slices
- Interleaved updates?
- Geometry shader could help, but is not efficient (on mobile)
  - Multiview extension!

One of the problems with cascaded shadow maps is that for every frame you need to actually update and render N slices and the same geometry might cover multiple slices at the same time, so you end up having to draw the same geometry multiple times.
There are schemes that could be applied here such as interleave updating cascades – such that the nearest cascade is updated every frame, while cascades further away are updated in an interleaved fashion.

Geometry shaders could help here, but that approach is not particularly efficient on mobile.

Luckily we have an extension for this!

Multiview extension

- GL_OVR_multiview
  - Originally conceived in order to make VR rendering more efficient
  - Bind multiple slices of a 2D texture array as framebuffer output
- Submit draw calls for all cascades once

So the multiview extension was originially conceived in order to make VR rendering more efficient. You can basically bind multiple slices of a 2D texture array as framebuffer output. All draw calls going into this framebuffer is duplicated to the bound texture slices.

So you end up with a single command stream for all cascades – which can be both good and bad. Bad in the cases where geometry is only visible in a single cascade. In this case we end up wasting bandwidth storing primitives for the cascades where it's not visible – luckily this won't affect the rasterizer since these primitives will be effectively culled away when tiling the geometry. So it might not be too bad still..

Multiview extension

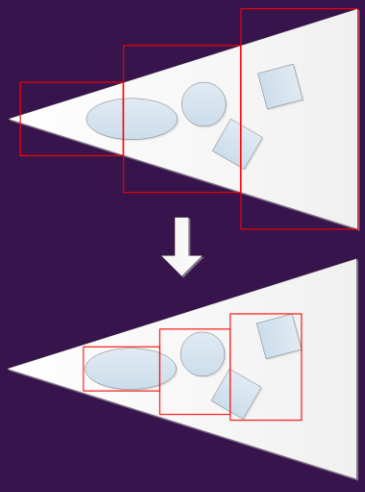
```
#extension GL_OVR_multiview: require

layout (num_views = 4) in;
uniform mat4 uViewProj[4];
in vec3 aPos;

void main()
{
  gl_Position = vec4(aPos, 1.0) * uViewProj[gl_ViewID_OVR];
}
```

This is roughly what the multiview vertex shader looks like.

The num_views layout qualifier sets number of views to output to. gl_ViewID_OVR is a new built-in that identifies the slice you're rendering to – so you can quite easily apply different transform to different slices. I'll return to the results I get when enabling this extension later.

Sample distribution shadow maps

- Analyze the current view to determine best fit for shadow projection



To improve on cascaded shadow maps, Andrew Lauritzen, Marco Salvi and Aaron Lefohn introduced the concept of sample distribution shadow maps. The idea is quite simple: analyze the current view to determine the best fit for the shadow projection.

So for normal cascaded shadow maps you would compute the split distances using a uniform or logarithmic distance formula. This would be based on the assumption that the shadowmap extends from near plane all the way to the far plane. By analyzing the current view we can extrude information such as nearest and furthest pixel and use that to compute cascade split distances that fits much tighter to the scene you're rendering.

We can also further trim the split frustum to only cover actually visible pixels – which in return will give us much better utilization of shadowmap resolution.

Sample distribution shadow maps

- Compute hierarchical min-max depth buffer from scene depth buffer
- To avoid GPU stalls, use depth buffer from previous frame
  - Re-project depth values and add some rubber band region

The basic idea of the approach is this.
Using the scene depth buffer we compute a hierarchical min-max depth buffer. Here we also ensure that values corresponding to the far plane is ignored.

To avoid GPU stalls we used the depth buffer from the previous frame and re-project the depth values to the current frame.

Sample distribution shadow maps

- Compute shader used to generate cascade split distances
- Min-max depth buffer used to further trim size of splits
  - Read-back to CPU to allow further per split culling
  - ...or store to buffer and draw “everything”

Next we have a compute shader that computes the cascade split distances and the min-max depth buffer is used to further trim the size of the cascade splits.

There are multiple ways of using the resulting values. The values can be read back to the CPU to allow further per split culling. Another possibility is to store per split transforms to a buffer and just draw “everything” – this might be okay if you’re content is not too complex, and you save the roundtrip back to the CPU.

Finally you could also do culling on the GPU and issue indirect draw calls.

Sample distribution shadow maps

- Balanced primitive count per cascade
- Better ratio between shadow map resolution and output resolution
- Can lower resolution to get similar result as higher resolution non-SDSM approach

Some of the advantages of sample distribution shadow maps is that you often end up with a more balanced primitive count per cascade split. There's also a better ratio between the shadowmap resolution and the frame output resolution.

Also, since sample distribution does a tighter fit to the actual scene, there's less wasted space in the shadow map and you end up rendering less geometry. You can even lower the resolution to get similar results as a fixed uniform or logarithmic split approach.

SHADOW FILTERING

Now I'll move onto shadow filtering. Shadow filtering is orthogonal to the shadowmap algorithms I've talked about.

Percentage Closer Filtering

- Depth textures cannot be filtered the same way as color textures
- Compare depth around current pixel
  - Returned value is average of the comparison results
  - Modern GPUs have HW support for this

Depth textures cannot be filtered directly. So in order to filter shadowmaps you have to apply filtering on the shadowmap comparison value – this approach is called percentage closer filtering.

Percentage closer filtering will be the baseline shadowmap filtering for this talk, and modern GPUs have hardware support for doing 2x2 PCF.

Shadow aliasing

- Shadow resolution doesn't map 1:1 with rendered image
- 2x2 PCF is not enough
  - … and going higher is expensive at run-time

The reason why shadow filtering is important is to hide aliasing artifacts and often the hardware 2x2 PCF is not enough to hide these filtering artifacts. Going higher than this can also severely hurt performance.

Variance Shadow Maps

- Approximates a distribution of depth values
  - Variance moments
- Filterable to some extent

- http://www.punkuser.net/vsm/

Variance shadow maps was introduced by William Donnelly and Andrew Lauritzen back in 2006. It basically reformulates shadow filtering to support pre-filtering. Instead of storing depth, we store variance moments. These are filterable to some extent – bluring too much for instance will increase the variance and make the shadows look worse.

Variance Shadow Maps

- Multi-sampled render to texture
  - Output variance moments in fragment shader
  - Result is resolved to single sample
- Apply separable blur (optional)
- Apply shadow map using Chebychev's inequality

Variance terms can be rendered using multi-sampled render to texture. The variance terms are output in the fragment shader, and resolved to a single sample using coverage averaging before being stored to texture memory. This is a good approach since you get this averaging for free.

After this you can further blur the texture before applying it to your scene using chebychev's inequality.

I won't go into a lot of detail here, but I will recommend you to read up on the publications for more details.

Variance Shadow Maps

- Requires two components rather than a single depth value
- Sensitive to floating point precision
  - Use RG32_FLOAT
- Light bleeding issues
- Careful when filtering with too large kernels

Now, variance shadow maps do come with some cost. It requires two components rather than a single depth value, it's also very sensitive to floating point precision so the recommendation is to use RG32_FLOAT textures. This means that it will cost an extra bit of bandwidth.

The biggest problem, however, with variance shadow maps is the light bleeding issues. It can be worked around to some extent but you cannot completely get rid of it.

Finally, as mentioned, you need to be careful not to apply too large filtering kernels.

Exponential Variance Shadow Maps

- Builds on variance shadow maps and exponential shadow maps*
- Fixes most of the light bleeding issues
- Apply exponential warp to shadow map depth
  - Positive and negative bounds
- Best result with RGBA32_FLOAT textures

Next is exponential variance shadow maps. This builds on variance shadow maps and exponential shadow maps and fixes most of the light bleeding issues. It works by applying an exponential warp to the depth values.

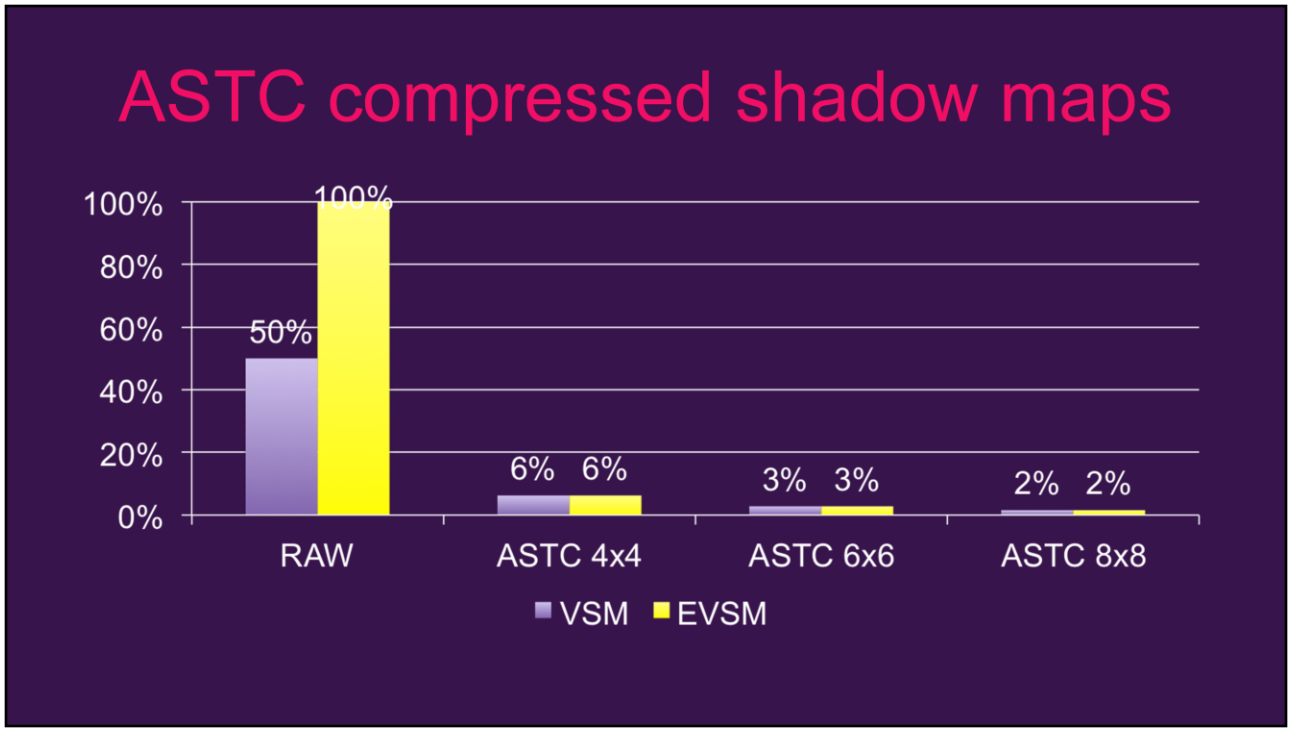
The biggest disadvantage is that it requires 4 floating point components so the bandwidth requirement becomes very high.

ASTC compressed shadow maps

- Idea: Compress VSM/EVSM to HDR ASTC textures
- Useful for static scenes with static lighting
  - Can still have dynamic geometry
- Massive storage savings
- Requires tweaking with ASTC encoder to get good results

One idea I tried out was to compress VSM/EVSM textures down to ASTC HDR textures. This can be useful for static scenes with static lighting – but you can still have dynamic geometry in there that receives shadow.

It does require some tweaking ASTC encoder parameters to good results, but the end result is a massive storage saving.



Just to give you an idea of the storage saving you could get, here's different ASTC block sizes compared to the raw VSM/EVSM.

RESULTS

Now onto the results.

Fixed distribution

512x512 w/4 splits

First screenshot is of a fixed logarithmic distribution using a 512x512 shadowmap and 4 splits.

Sample distribution

512x512 w/4 splits

With sample distribution you can see that there's more shadow details near the camera, while preserving the shadow detail in the back.

Fixed distribution

1024x1024 w/4 splits

Same comparison using a 1024x1024 shadowmap.

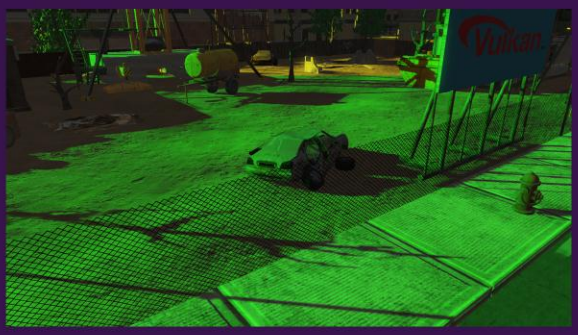
Sample distribution

1024x1024 w/4 splits

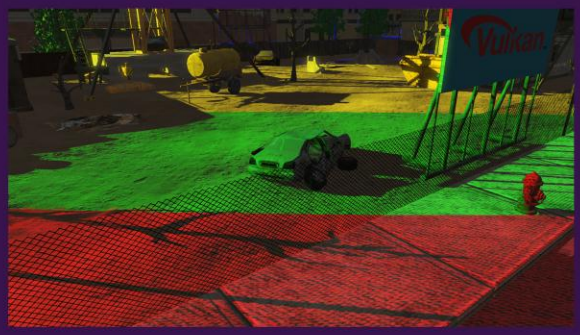
Here you can really see the advantage of using sample distribution shadow maps. A lot more shadow detail near the camera.

Split comparison

Fixed distribution

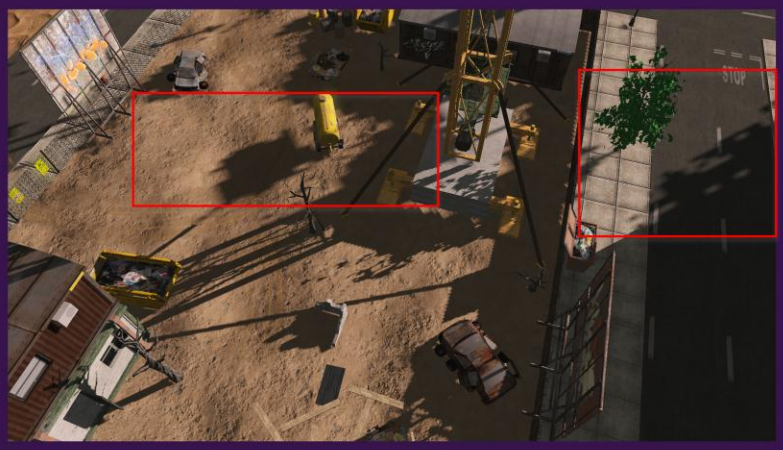


Sample distribution



For comparison this is how the distribution differs between the approaches. So red is nearest, then green, yellow and blue.

Notice that the nearest split isn't even used when rendering using the fixed distribution.



Fixed distribution

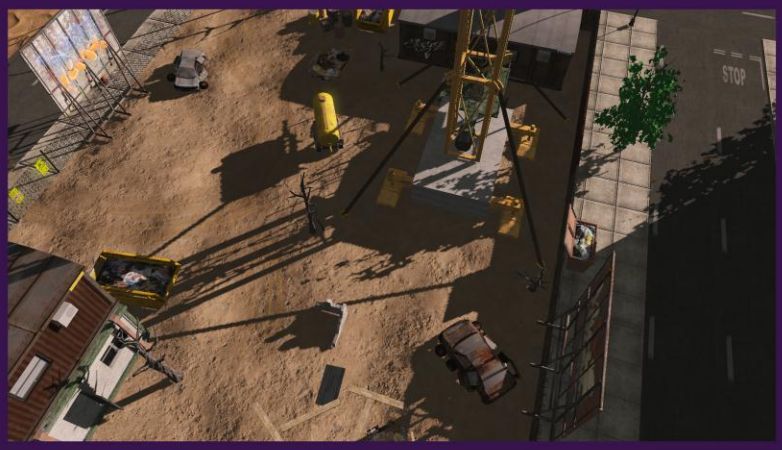
1024x1024 w/4 splits

Here’s another camera angle. Notice how little resolution is available in shadow inside the rectangle. If I turn on sample distribution, this changes dramatically.

Sample distribution

1024x1024 w/4 splits

Much better. This is with 4 cascades – if I change this to 2 cascades.



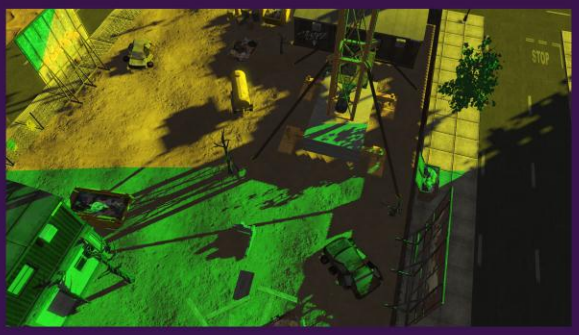
Sample distribution

1024x1024 w/2 splits

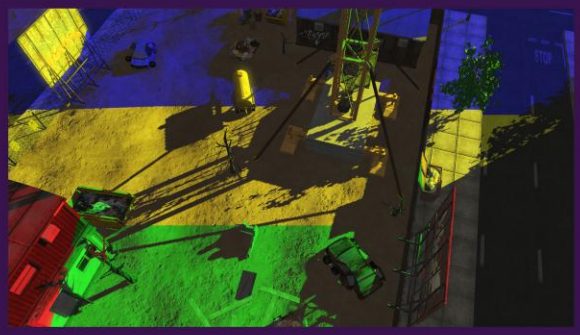
There’s not much of a difference.

Split comparison

Fixed distribution



Sample distribution

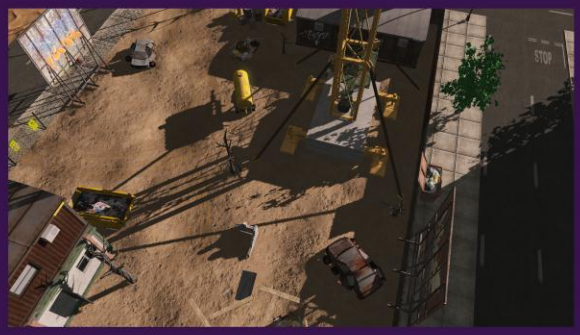


Again comparing the split distribution of the two approaches – again much better utilization with sample distribution.

Filter comparisons

256x256 PCF

1024x1024 PCF



Now moving on to some filtering comparisons.

Here we have two screenshots rendered using PCF filtering with 256x256 and 1024x1024 shadow maps. Notice in particular how little resolution is available in the 256x256 PCF version.

Filter comparisons

256x256 VSM

1024x1024 VSM

With VSM, even the 256x256 shadowmap looks OK. The shadows aren't crisp but they aren't aliased either.

But as mentioned earlier, the problem with variance shadow maps is light bleeding. Enclosed in the red rectangle you can see this issue quite clearly.

Filter comparisons

256x256 EVSM

1024x1024 EVSM

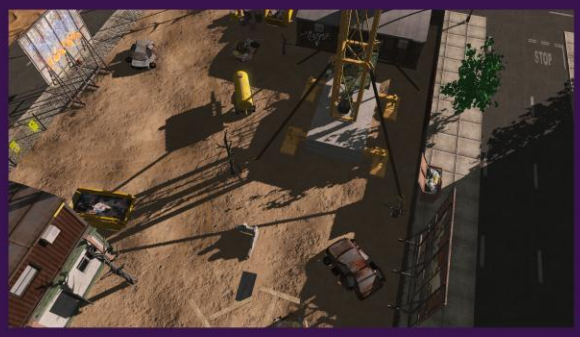
Next up is exponential variance shadow maps. Filtering quality is very similar to variance shadow maps, but now most of the light bleeding issues are gone.

There are still some light bleeding – this is probably due to the complexity of the scene. The scene has multiple areas where there are large depth discontinuities in the shadow map.

In order to improve this further we would have to implement multi layer version of variance shadow maps.

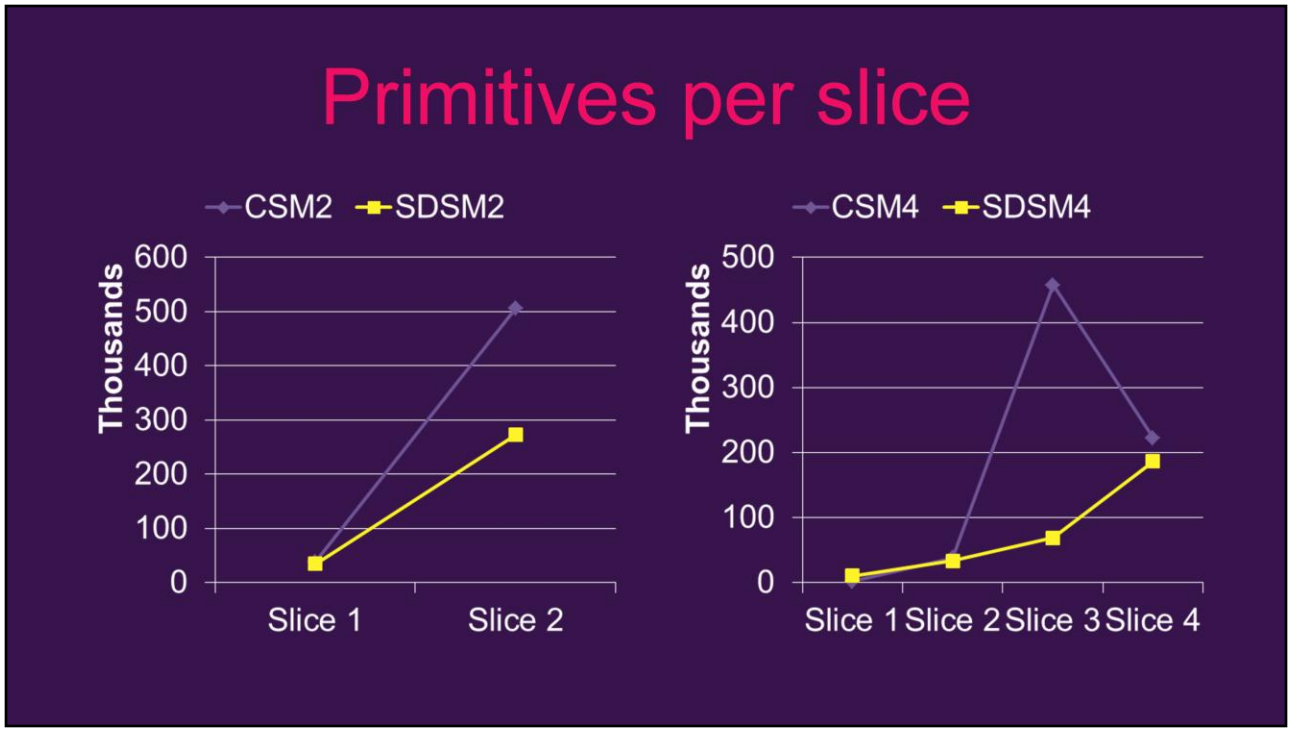
Performance

- Test scene contains ~2.5 million primitives
- Tested on a Mali-T880 MP12 running at 720p resolution



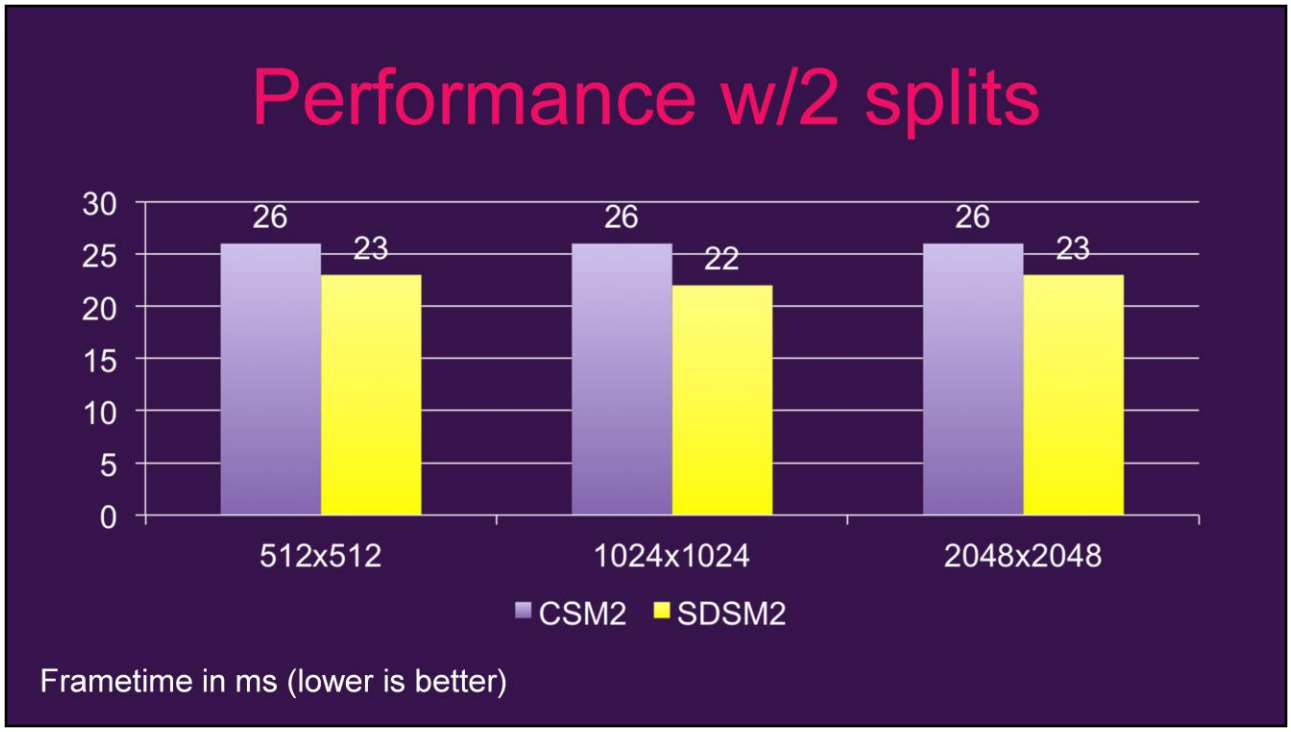
Performance was measured by rendering a scene that contains a total of around 2.5 million primitives. The test was conducted on a Mali-T880 MP12 running at 720p resolution.

Note that performance measurements was done on a single device – time constraints prevented me from doing this on more devices.



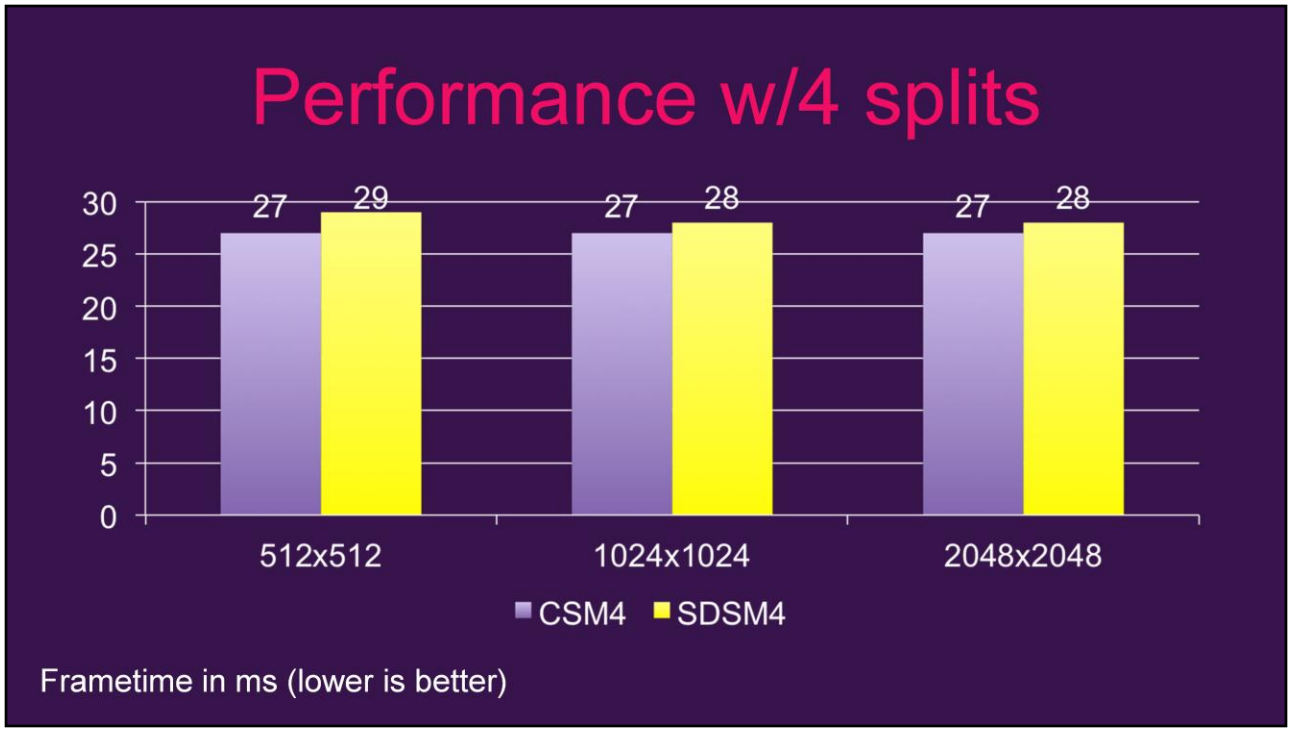
This slide shows the primitive count distribution of the different shadowmapping algorithms.

The important take away here is that for sample distribution the total primitive count is much lower than using a fixed distribution – also the primitives are much more uniformly spread among the splits.

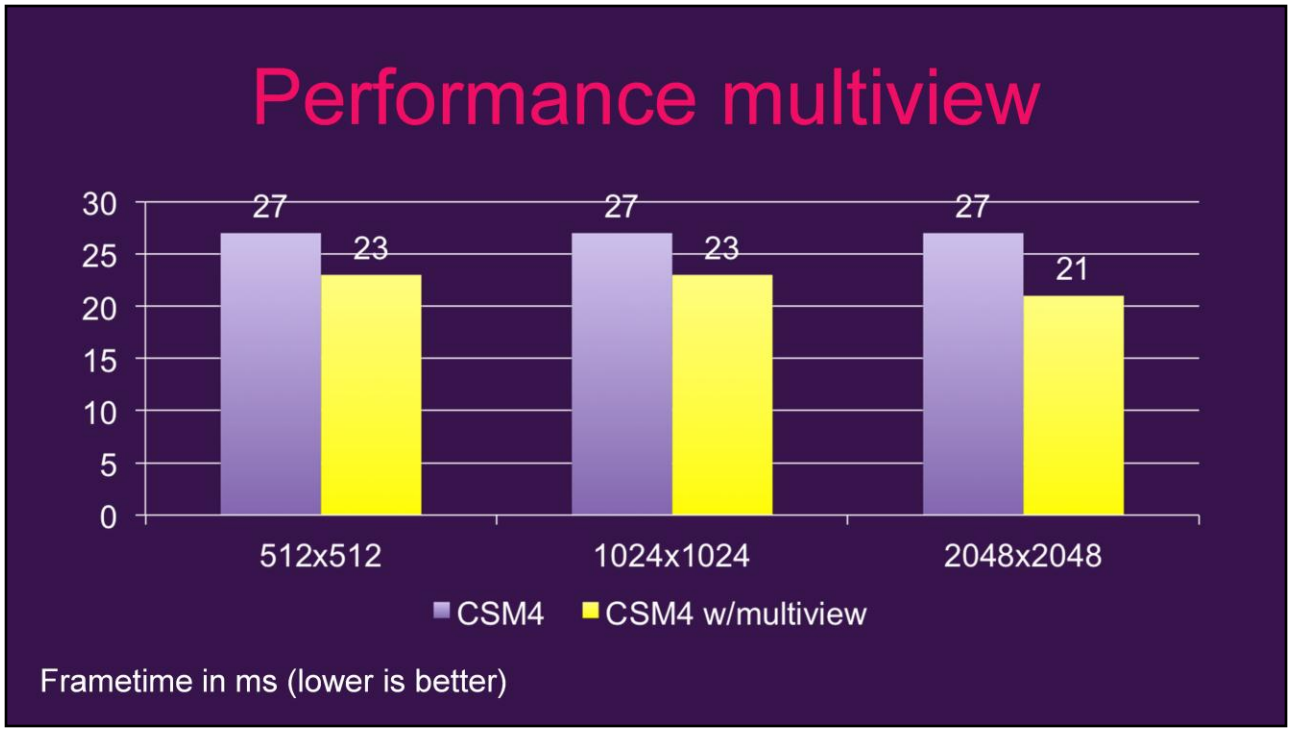


The first performance numbers is a comparison between cascaded shadow maps and sample distribution shadow maps using 2 splits.

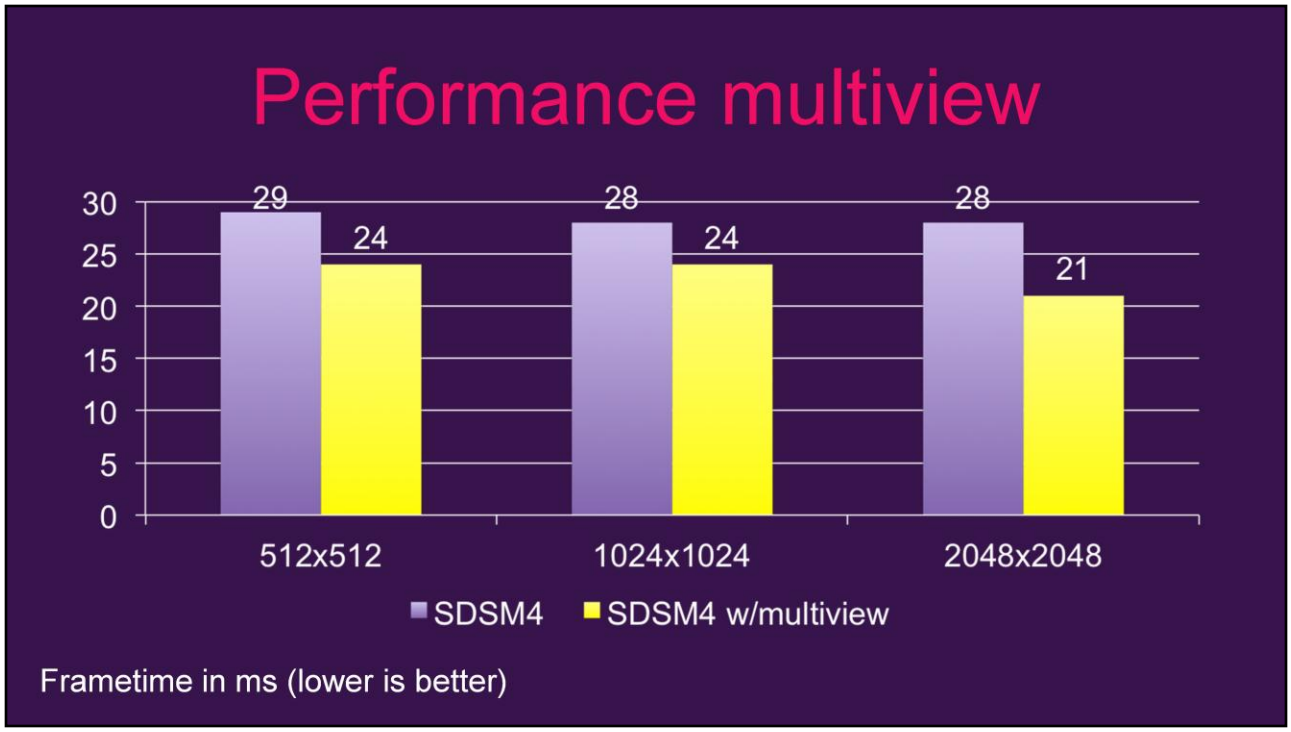
Interestingly, even though the sample distribution code path does more work before rendering shadows, that work pays off in terms of total frametime. This is due to sample distribution allowing much more aggressive culling of geometry which does not contribute to the shadow map.



Next up is the same performance comparison, but now with 4 splits. In this case sample distribution shadow maps does slightly worse – this is due to having more geometry that covers multiple splits. So even if you're able to cull away geometry that does not contribute to the shadowmap – that gain is lost since you have to re-submit and draw the same geometry into multiple slices.



Next is a comparison between cascaded shadow maps with and without multiview rendering enabled. As you can see, multiview allows much more efficient rendering of shadow depth compared to the regular approach.



Similar to the previous slide, but now a comparison with sample distribution shadow maps with and without multiview enabled. Again multiview shows a great performance improvement over the non-multiview enabled codepath.

Regarding why 2k resolution shadow maps perform much better – I'm afraid I don't have a good explanation yet. After Siggraph I will do more detailed profiling to figure out what exactly is going on.

Conclusion

- Sample distribution shadow maps is practical on mobile
- Multiview is great for shadow rendering
- Future work
  - ASTC encoding and quality trade-off
  - Vulkan
  - Broader performance analysis

So, my goal was to try out different shadow mapping algorithms on mobile, and I think I've shown that performance and quality of shadowmapping in mobile is very good. By using extensions such as multiview we can improve shadowmap rendering by a quite large margin. I was actually surprised myself at the performance and quality level I was able to achieve when implementing these various techniques.

Thank you!

Marius.Bjorge@arm.com

References

1. Sample Distribution Shadow Maps
   - Lauritzen, Salvi, Lefohn
2. Variance Shadow Maps
   - Donnelly, Lauritzen
3. Exponential Shadow Maps
   - Annen, Mertens, Seidel, Flerackers, Kautz
4. GL_OVR_multiview
   - https://www.opengl.org/registry/specs/OVR/multiview.txt
5. Lighting Research at Bungie
   - Chen, Tatarchuk