Rendering Structures

Analyzing modern rendering on mobile

- Hans-Kristian Arntzen
- 2018-08-16 – SIGGRAPH 2018

Content

1

Motivation

2

Scene and lights

3

Rendering structures overview

4

Benchmark results

5

Post-AA benchmarking

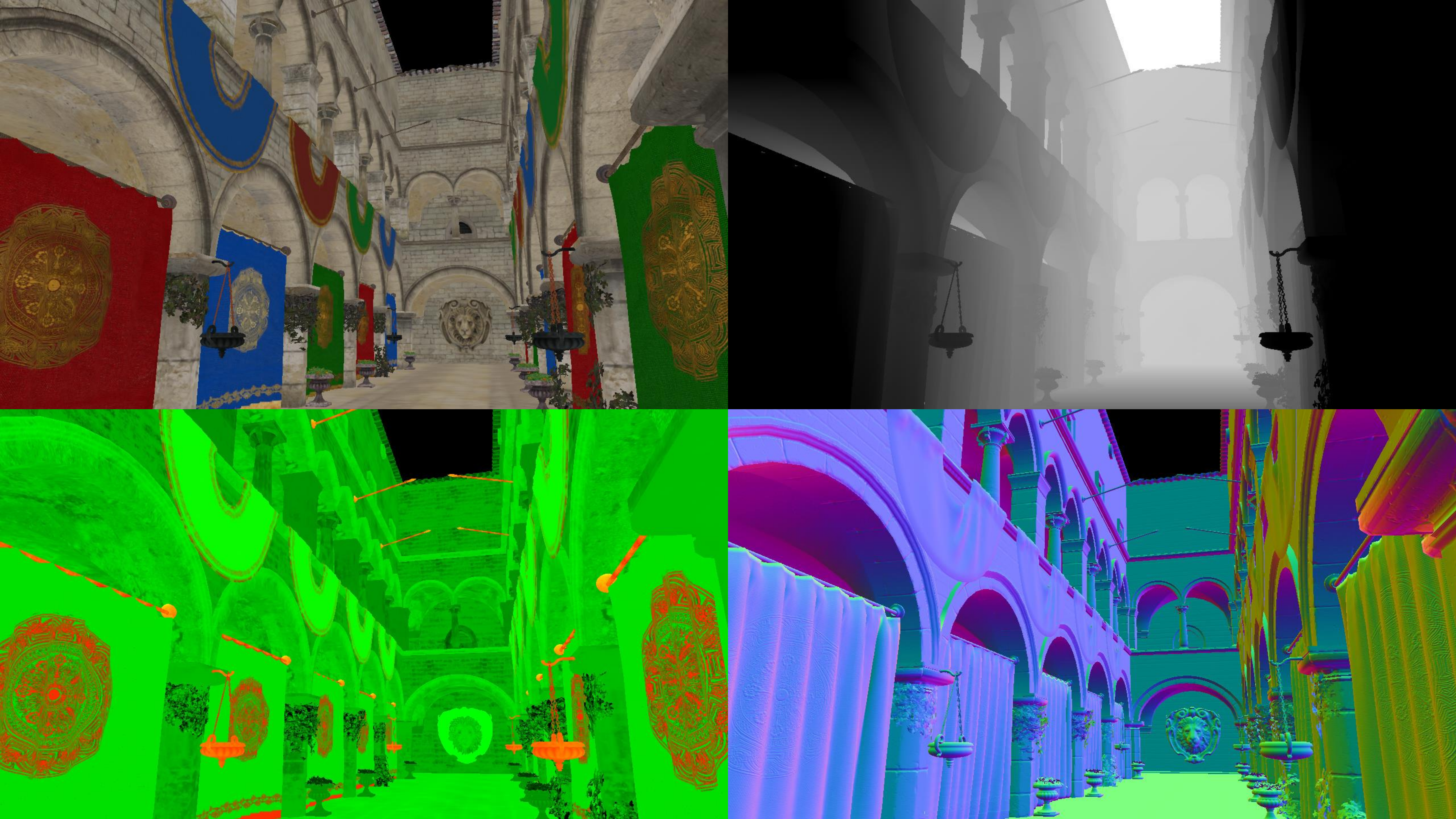
Motivation

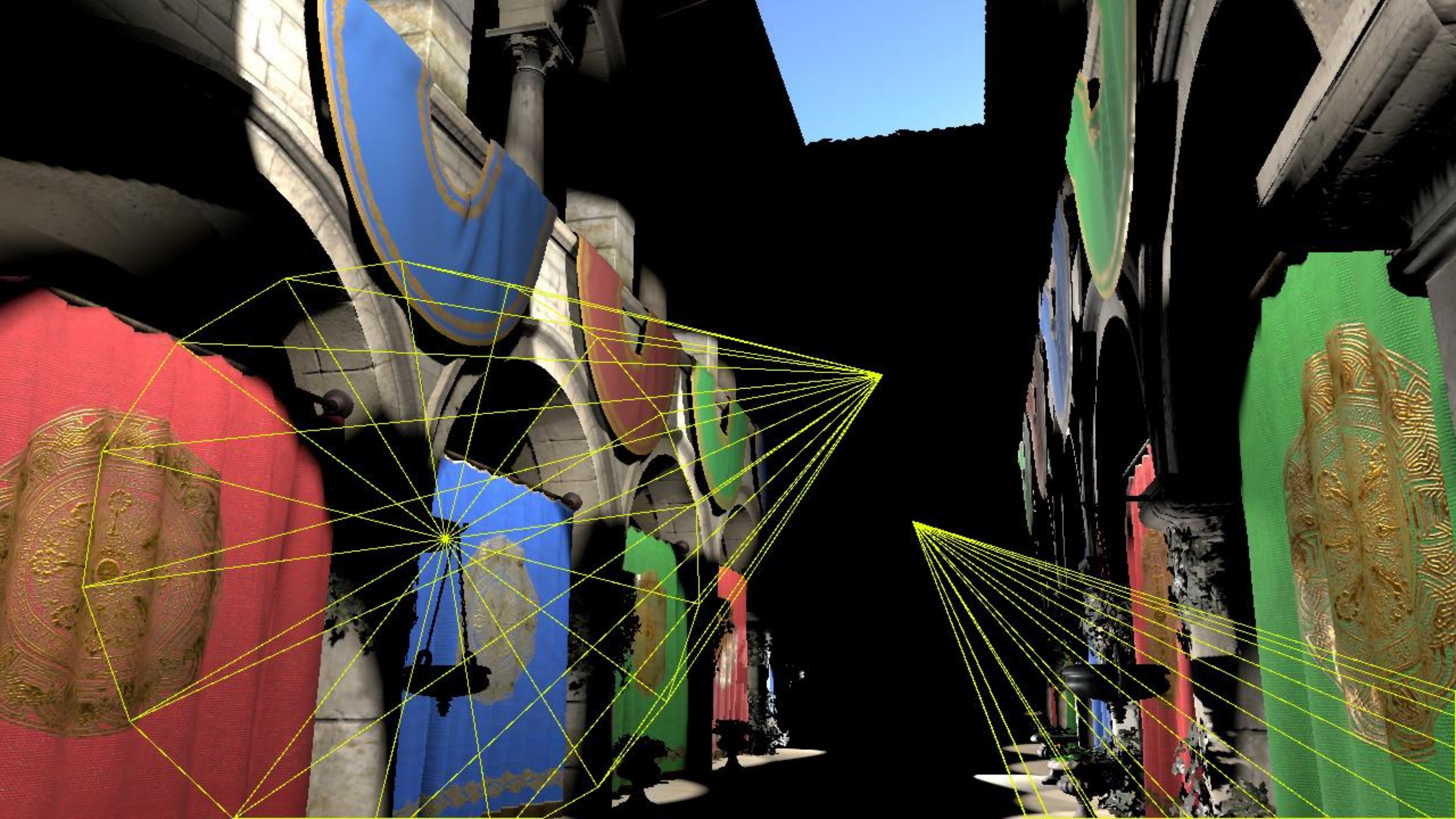
- Performance characteristics for mobile architectures differ from desktop
- Very little comparative data on rendering many lights on mobile
- Explore the most promising rendering structures for mobile
- Focus on Mali
- Midgard GPU family is very different to Bifrost

The scene

Sponza, duh! [3]

Classic deferred shading

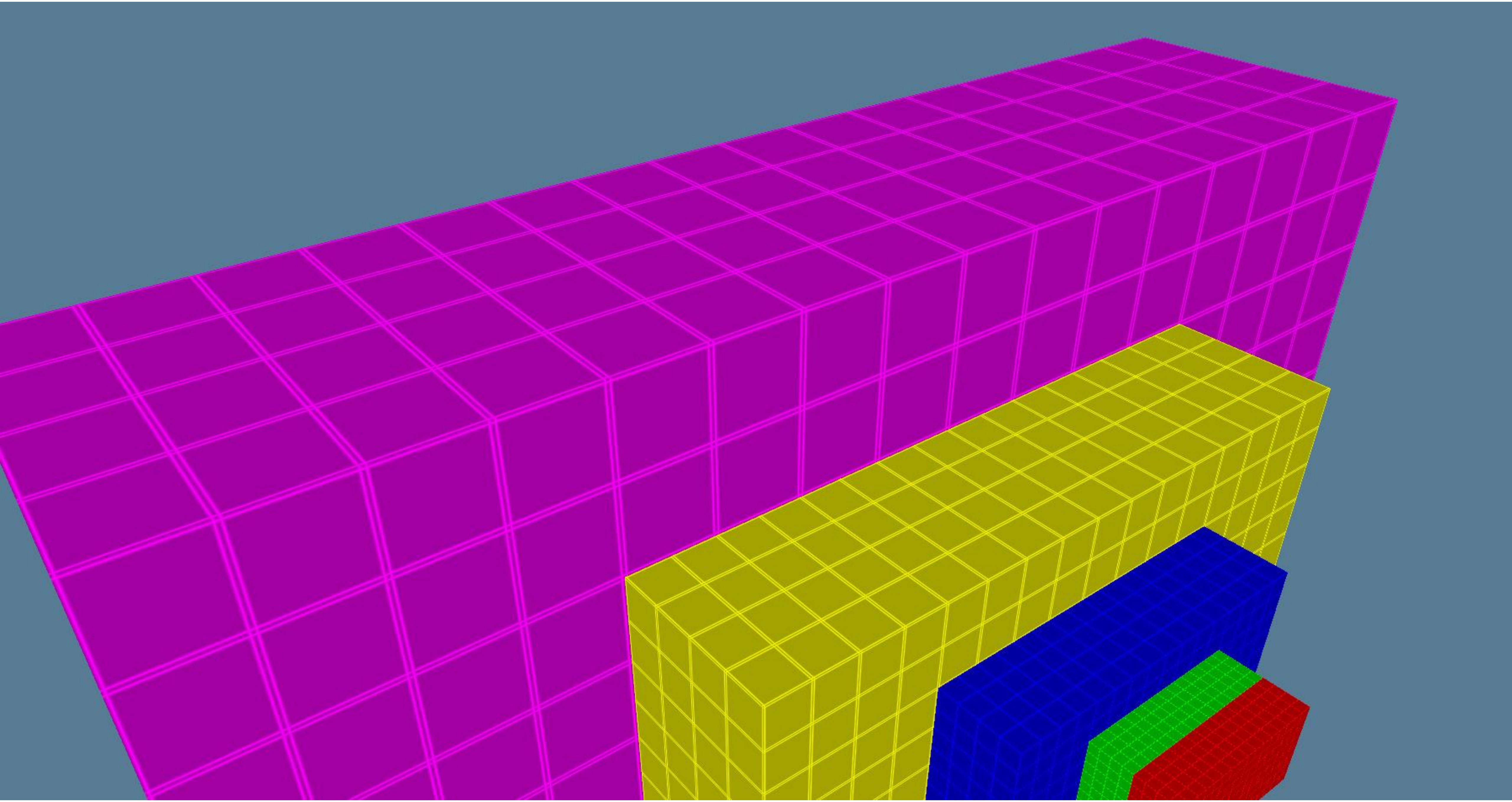


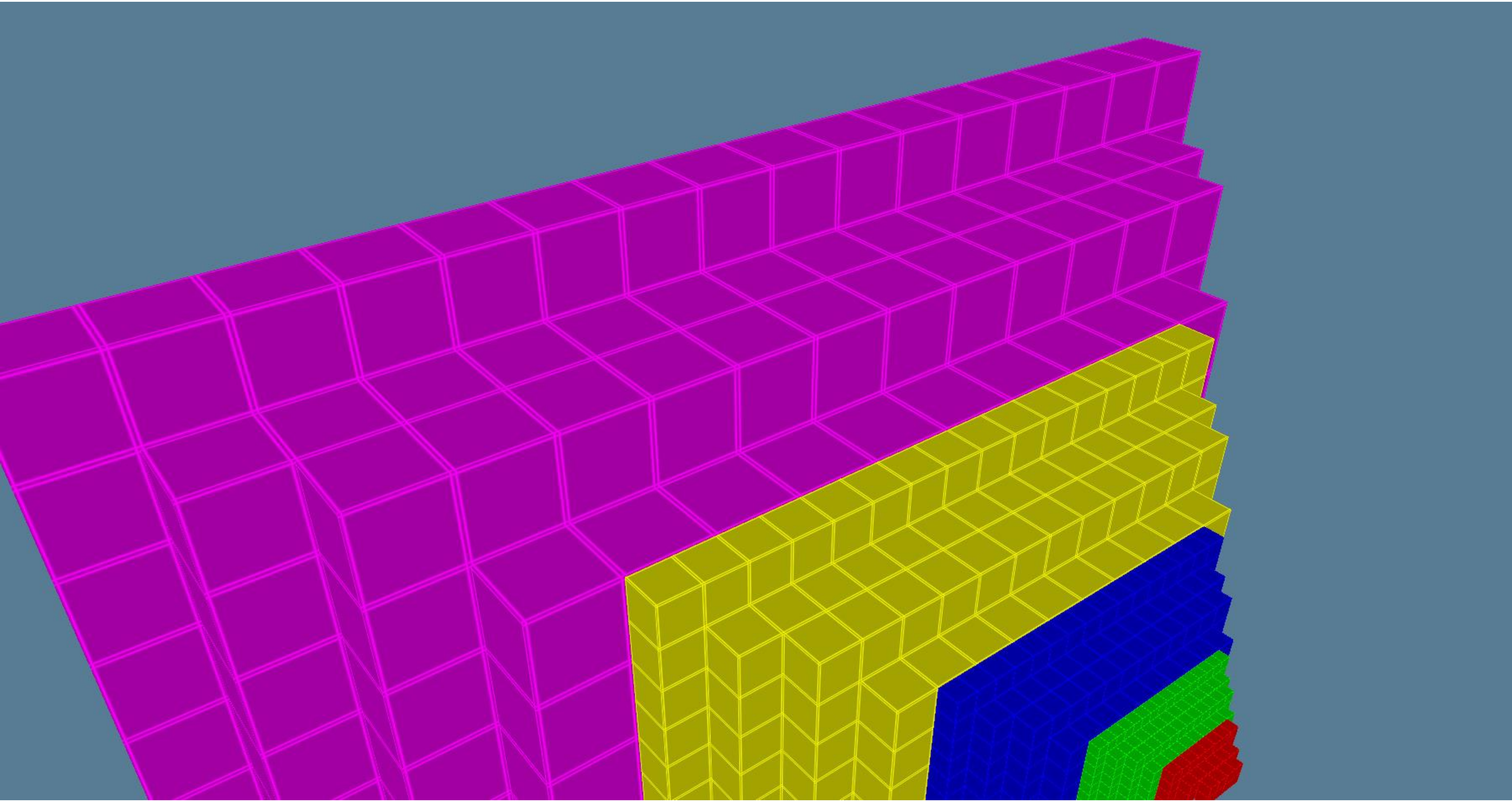


Pros and Cons

- Pros
  - Arbitrary number of lights
  - Arbitrary different light types
  - Separate shadow maps per light
  - Small, decoupled shaders
  - Robust against geometry overdraw
- Cons
  - No (easy) MSAA
  - No (easy) transparency
  - False positives in shading (single-sided depth test)

Clustered shading





Forward and deferred clustered shading

- Forward
  - Look up cluster and shade all lights when rendering a mesh directly
- Deferred
  - Render a full-screen quad in lighting pass and shade all positional lights in one go based on reconstructed position

Pros and Cons

- Pros
  - Very flexible
    - Deferred, Forward, Transparency, MSAA, Volumetrics
  - Can be computed before knowing depth buffer
  - Can be expanded to support more than just lights
- Cons
  - All resources (e.g. shadow maps) need to be bound
  - Some fixed overhead to sample cluster
  - Very heavy shader

Forward Z pre-pass

The forward depth prepass

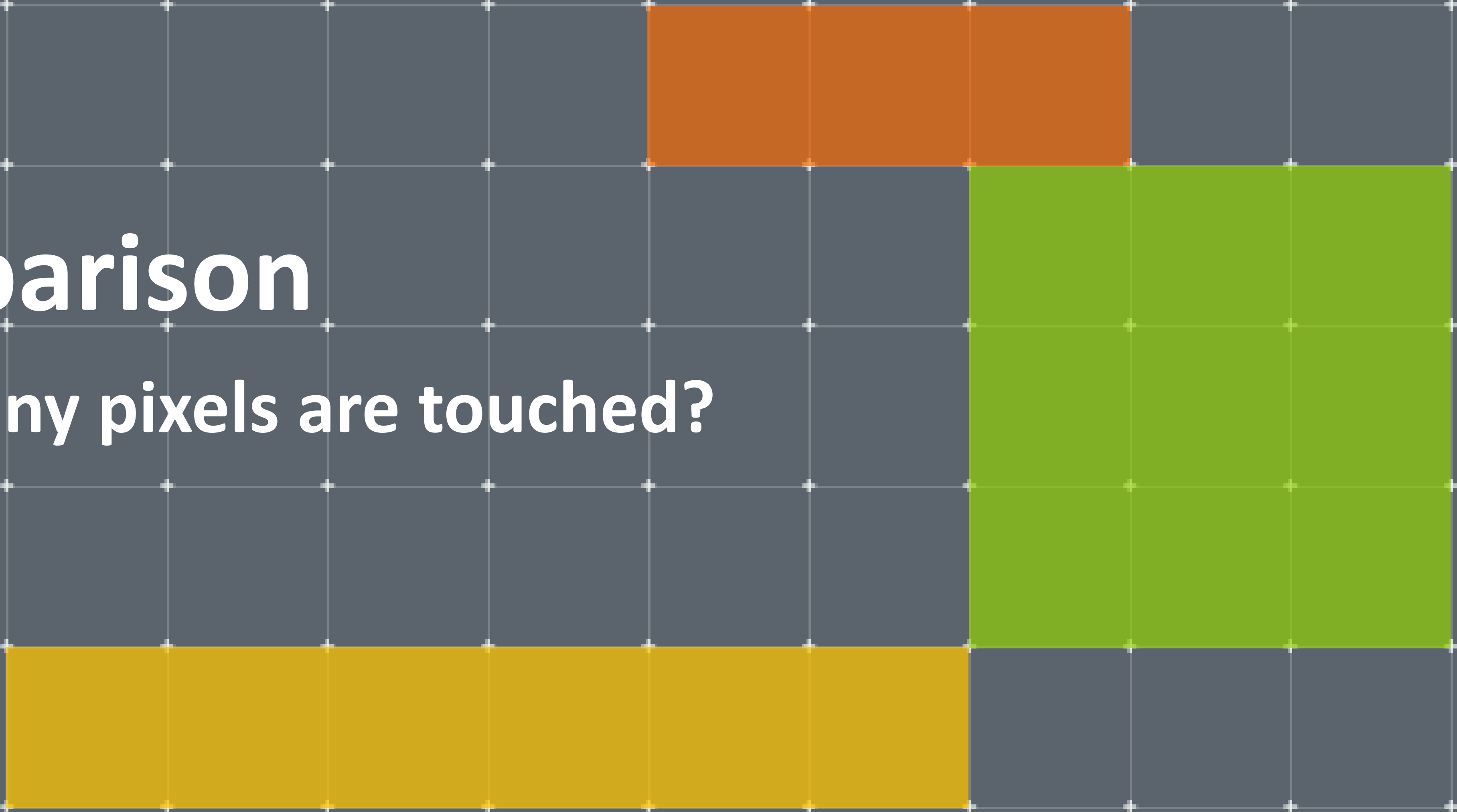
- Forward clustering shaded pixels are heavy
- Want to avoid over-shading
- Perfect front-to-back sorting is impractical
- Can potentially be done on-chip
- Sometimes unavoidable for certain effects
  - Screen-space AO techniques

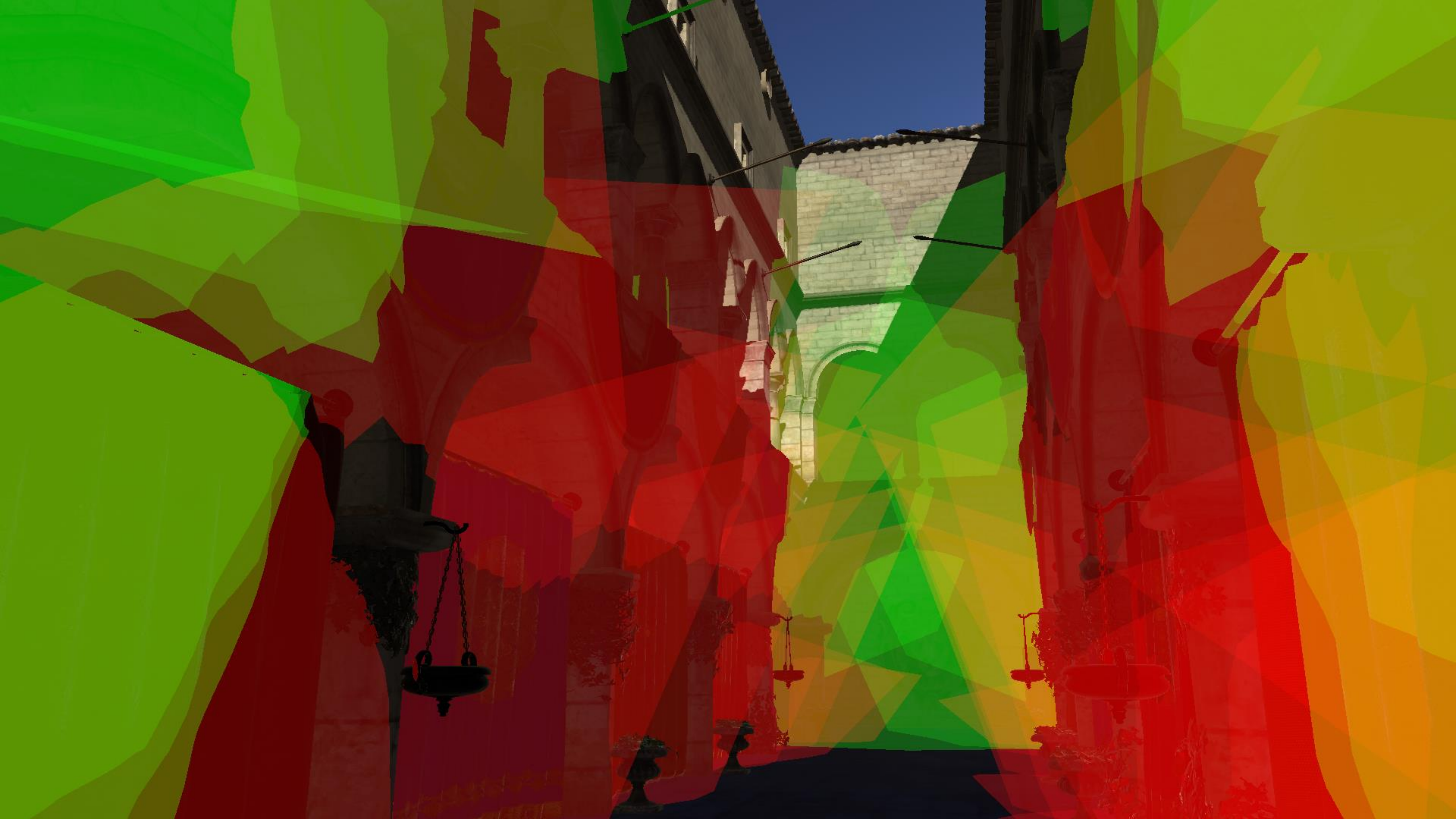
The forward depth prepass

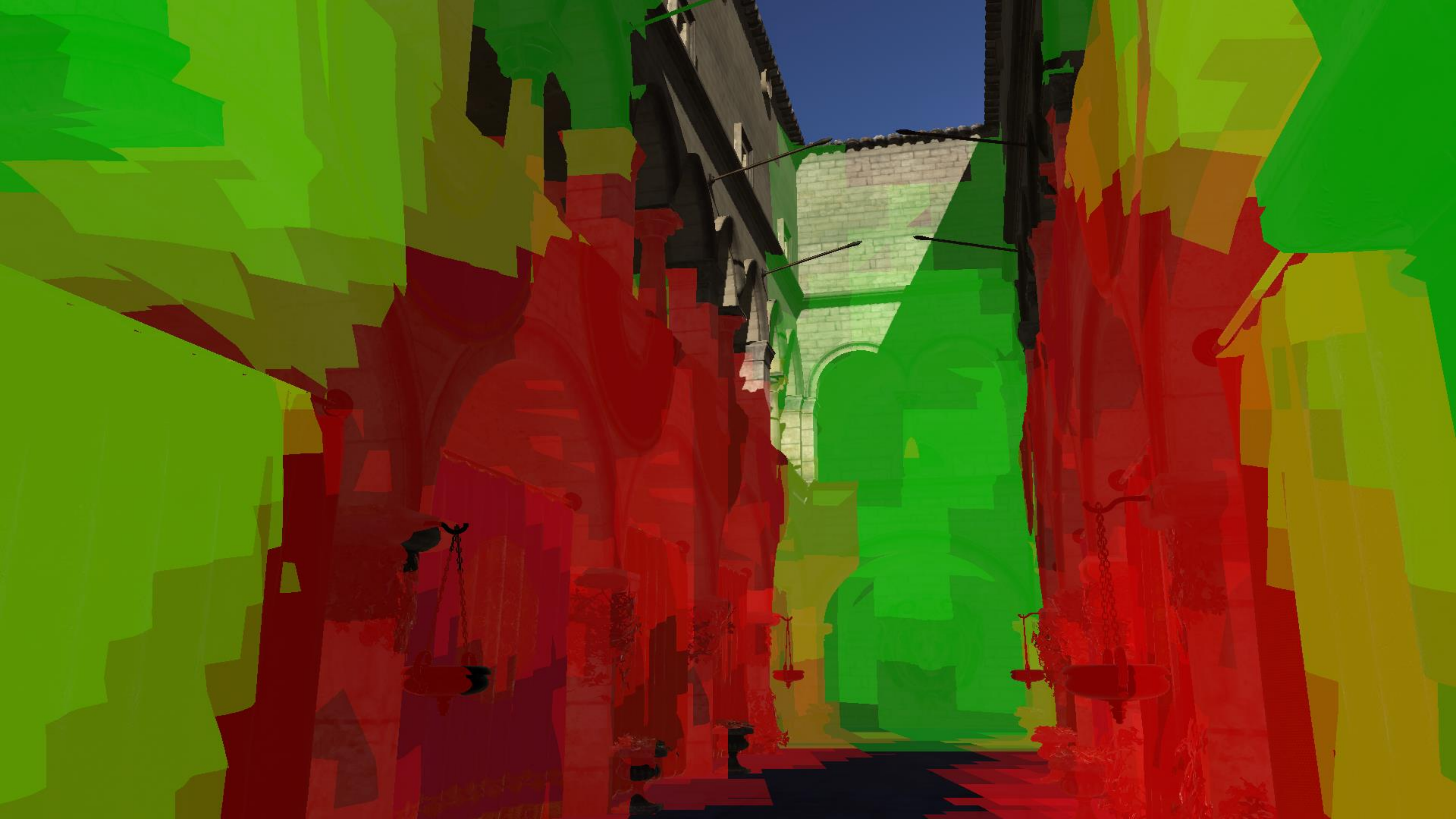
- Pros
  - No over-shading
  - More flexible drawing order
- Cons
  - Double the geometry load
  - More bandwidth required
  - More CPU overhead

Comparison

How many pixels are touched?







Results

Using the Granite renderer [2]

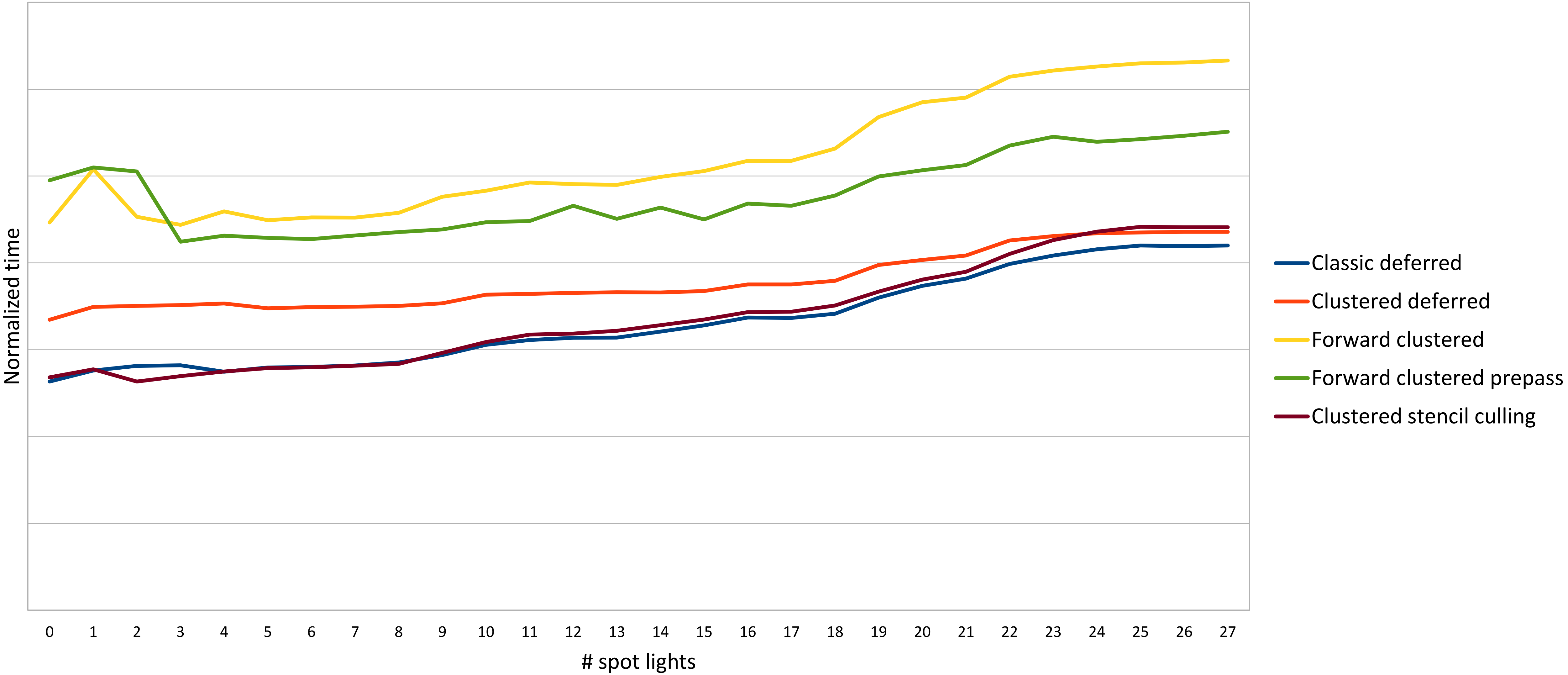
The grand benchmark

- Create a massive benchmark sweep
- Compare the rendering structures head-to-head
- Turn knobs on and off, and see how it affects performance

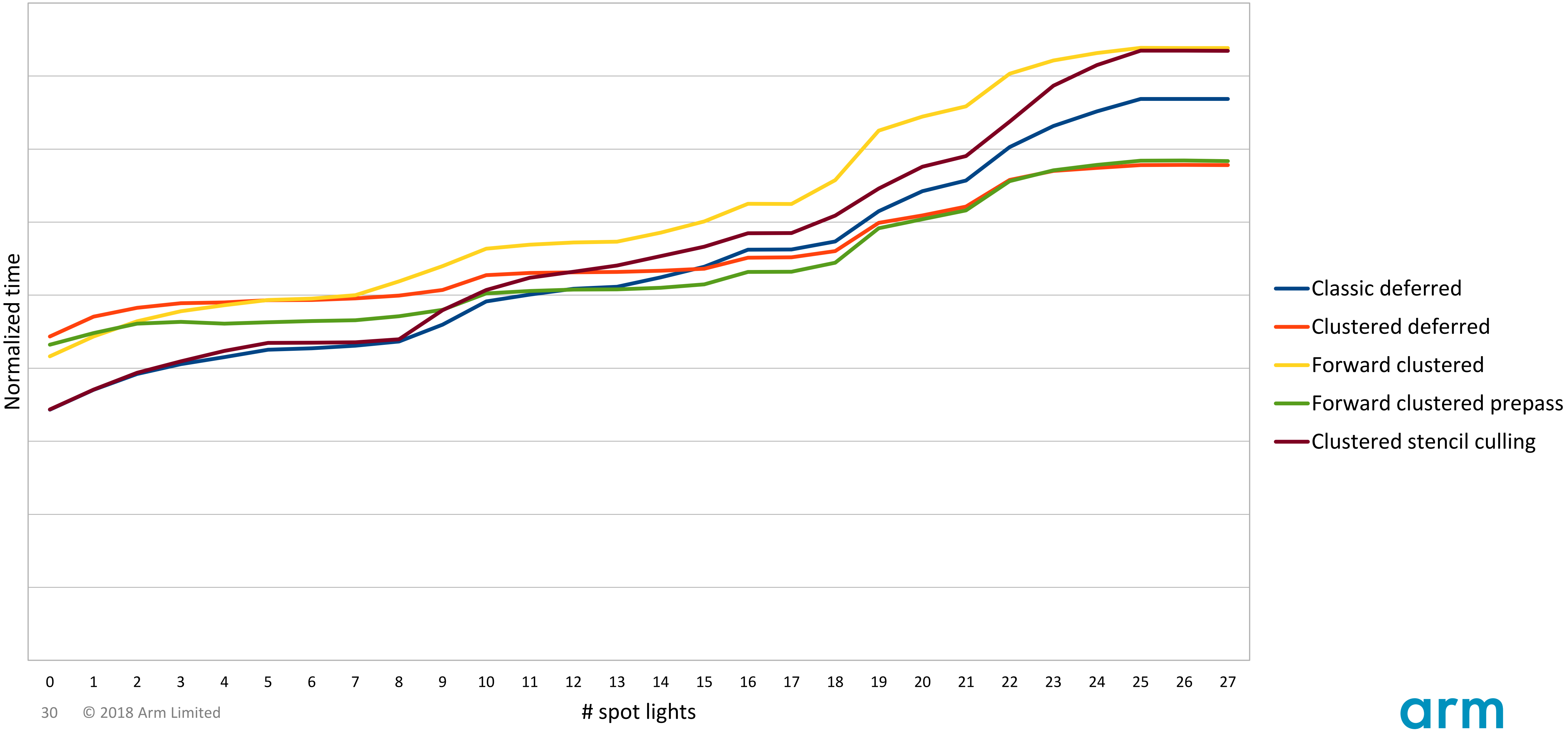
Test hardware and renderer

- Mobile
- Desktop reference points
- Vulkan
- PBR
- Multipass techniques used for deferred

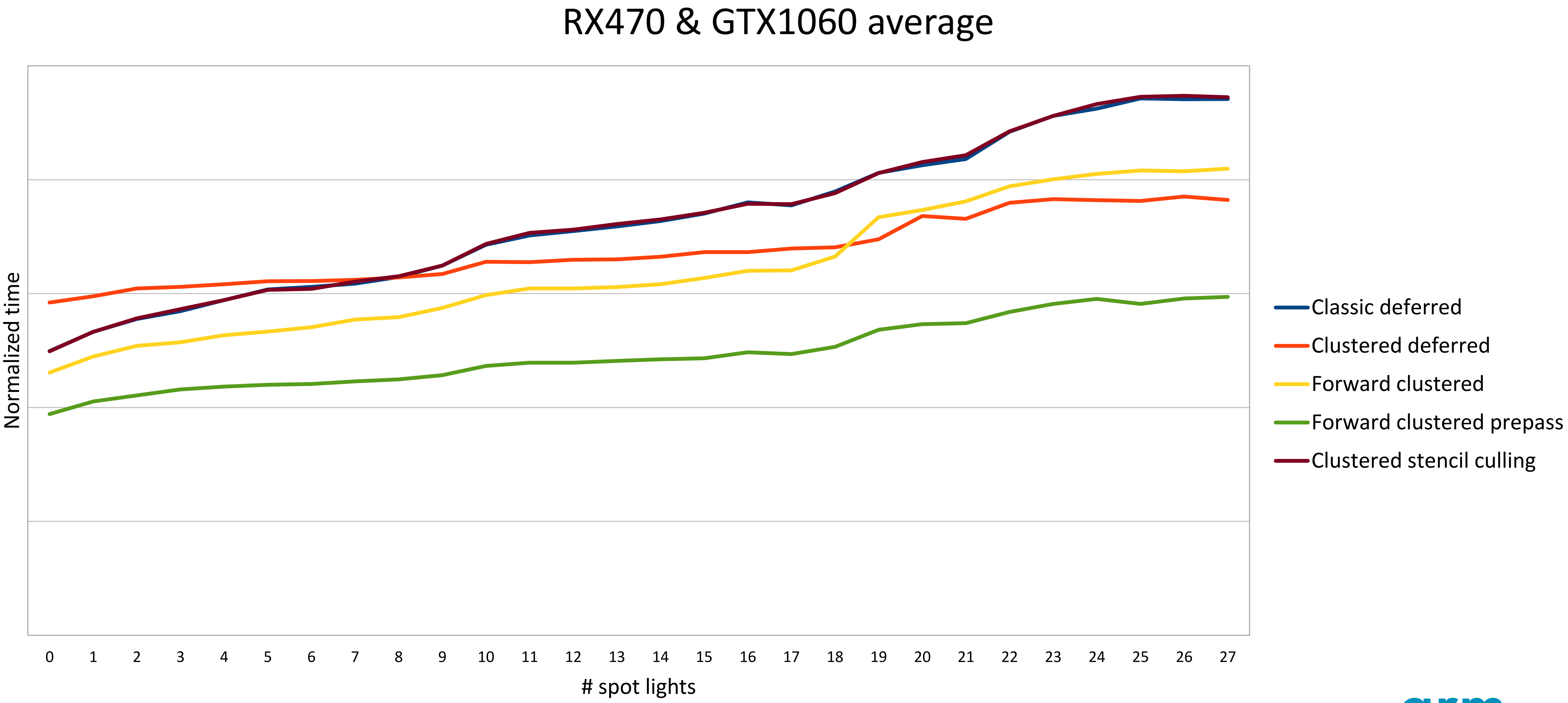
Midgard (T-880) greatly prefers deferred techniques



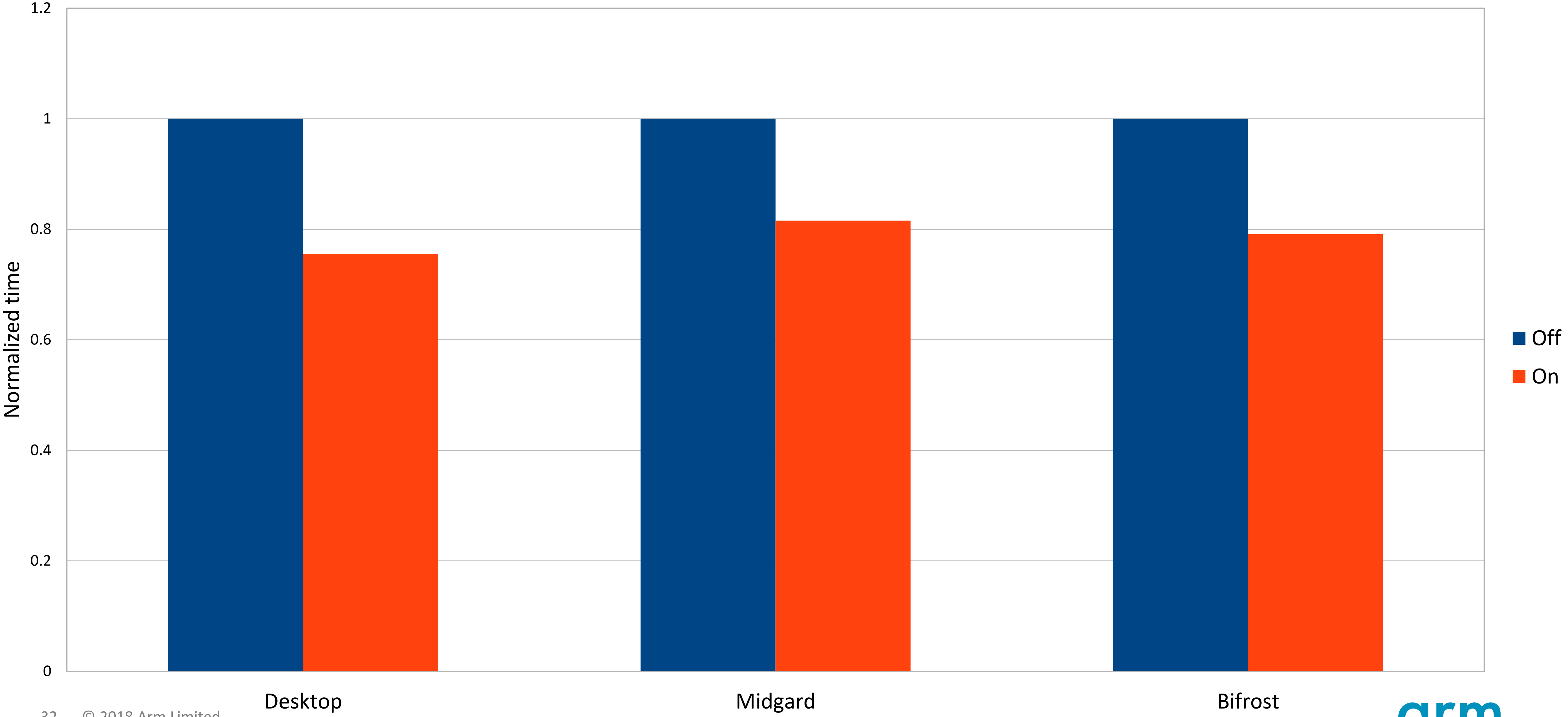
Clustered shading scales very well on Bifrost (G71 & G72)



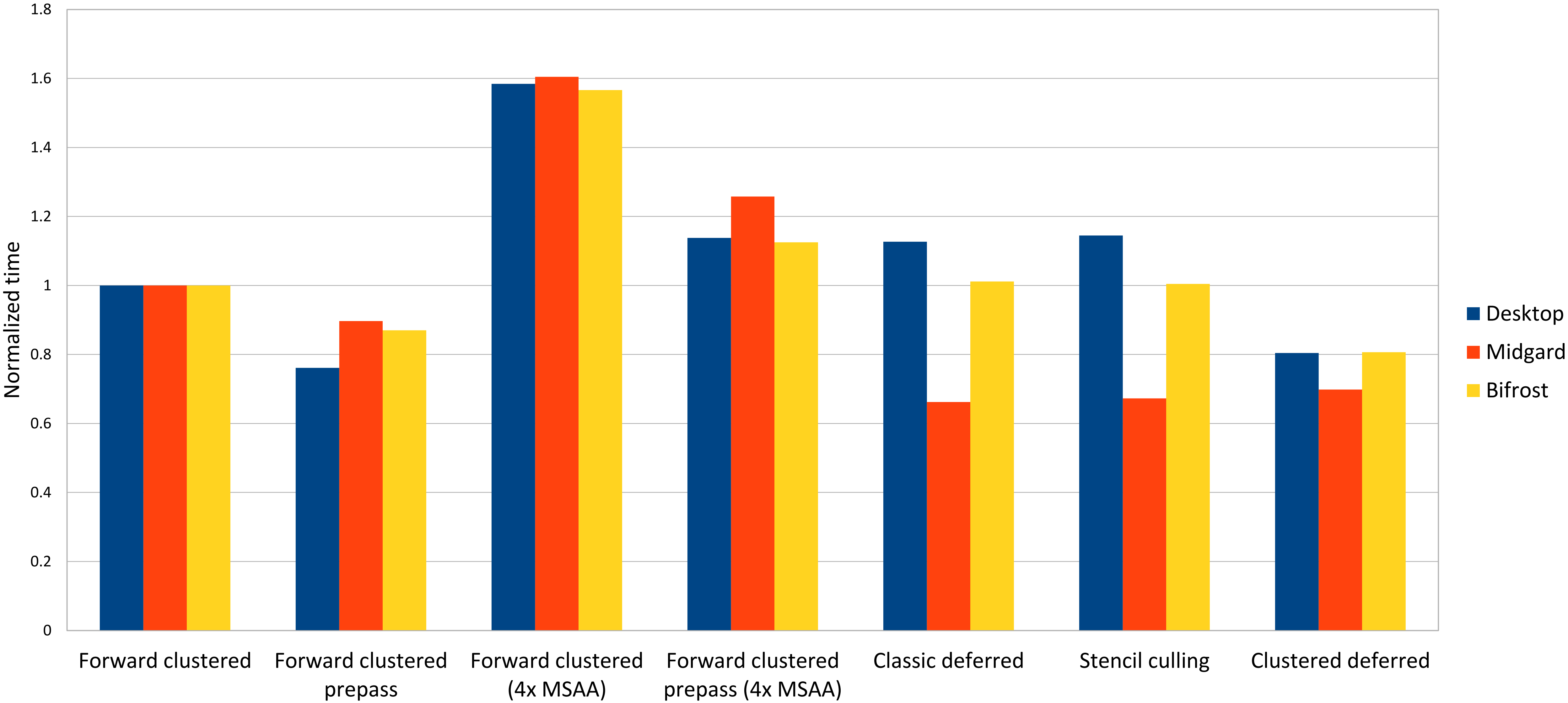
Gap between deferred and forward is greater on desktop



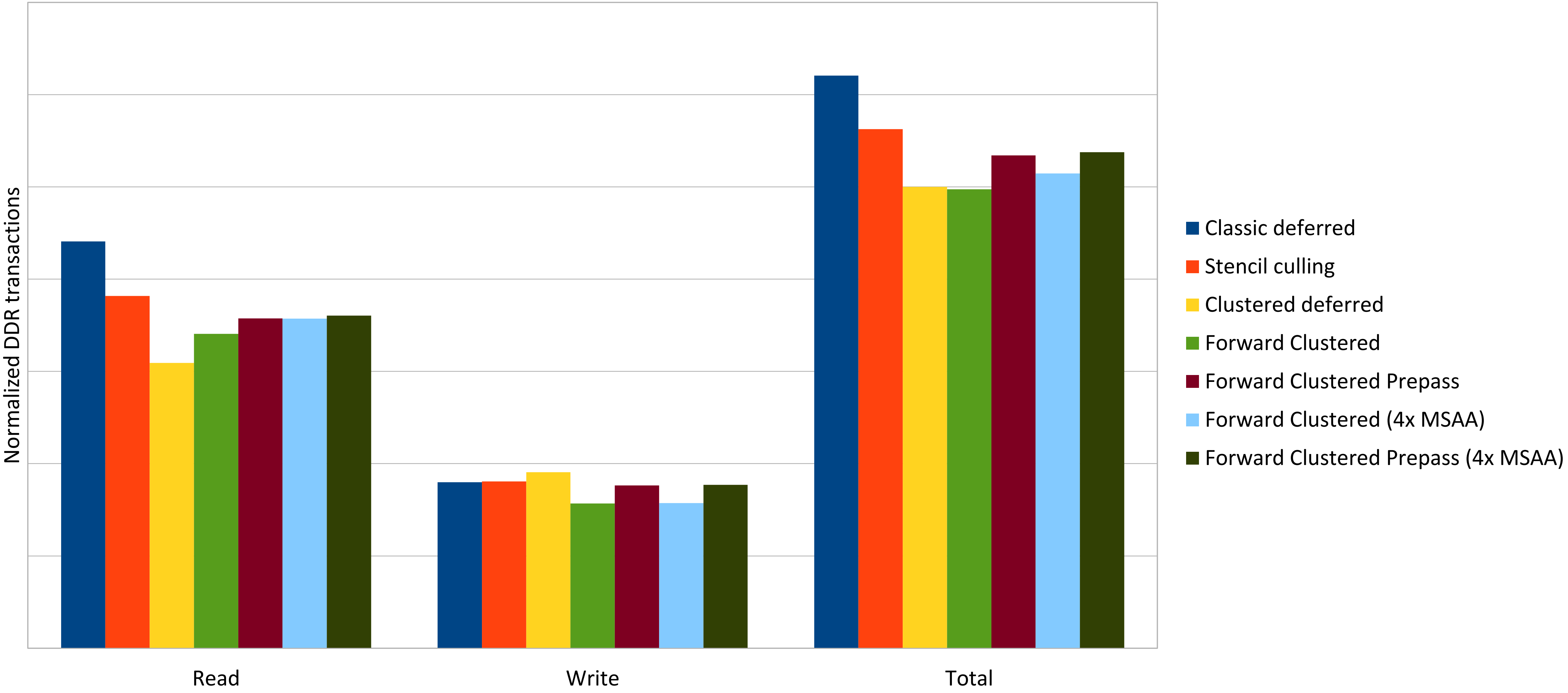
Forward depth prepass might be a good idea



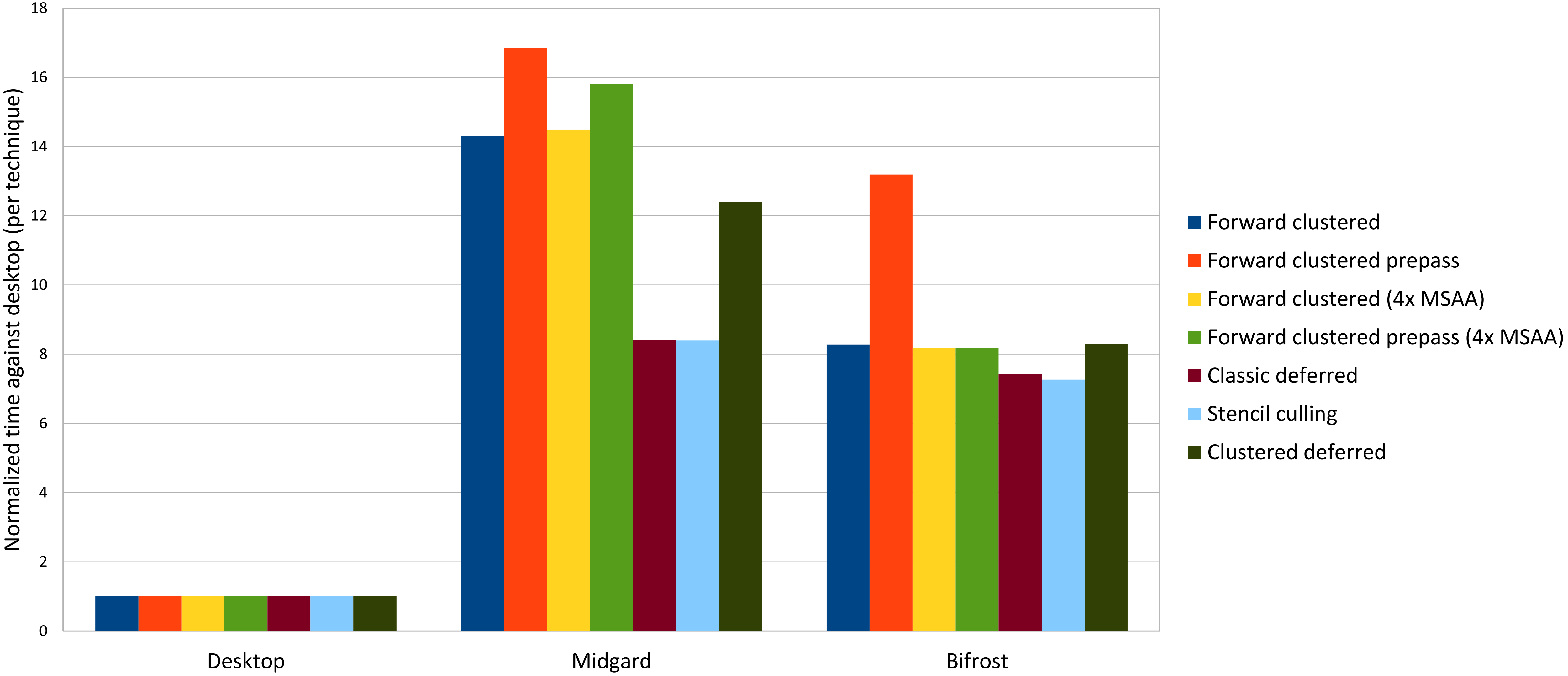
MSAA is super expensive with microgeometry



Clustered shading actually improves bandwidth



One tenth performance compared to mid-range desktop



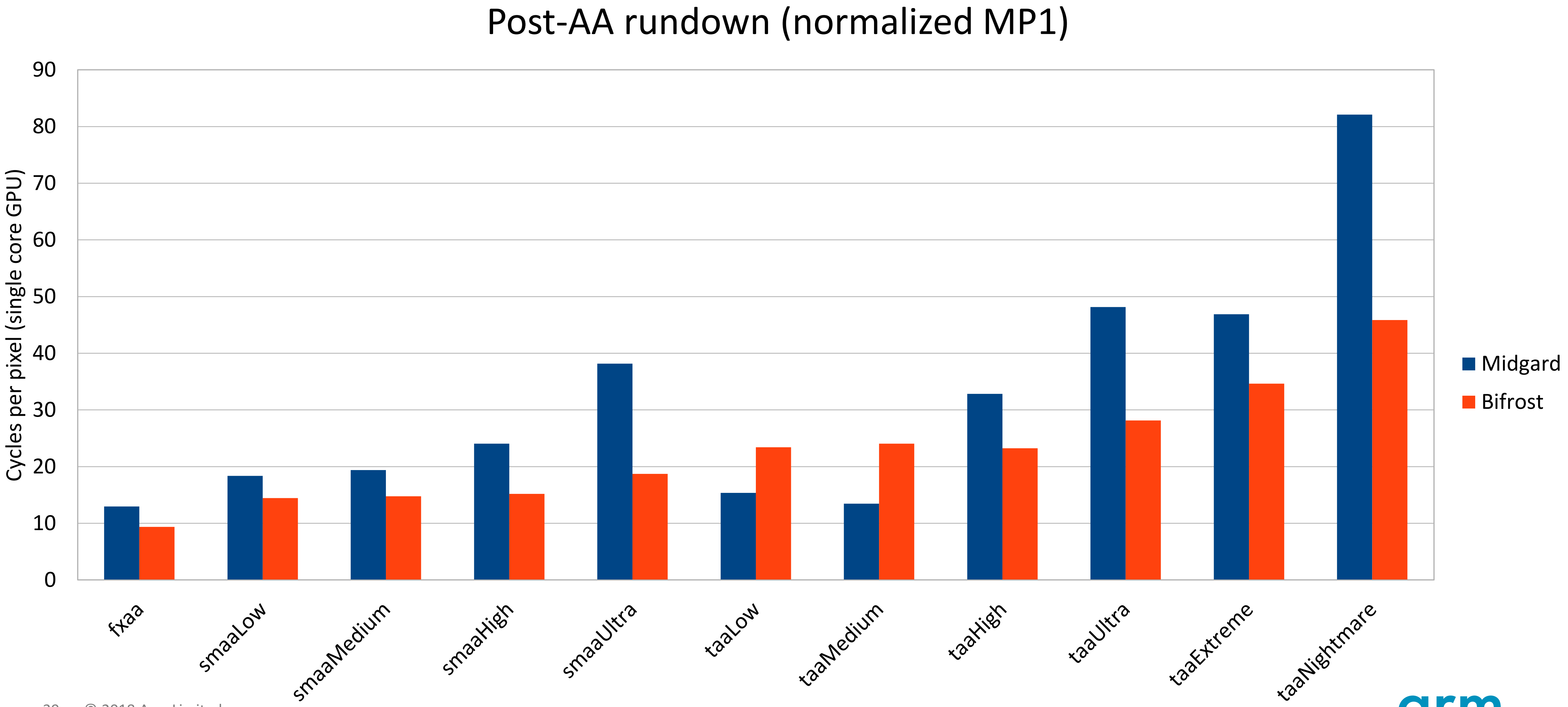
Post-AA rundown

The usual suspects

- FXAA
  - 9-tap
- SMAA
  - Low, Medium, High, Ultra
- TAA

Reconstruction filter	Color neighborhood	Pre/Post tonemapping	Max velocity	Color space	Clamping method	
Bilinear	5-tap cross	None	5-tap cross	RGB	Clamp	Low
		MAX3				Medium
	Rounded Corner				AABB clip	High
				YCgCo		Ultra
	Variance Clipping		3x3			Extreme
Bicubic						Nightmare

Results


Post-AA rundown (normalized MP1)
Cycles per pixel (single core GPU)
90
80
70
60
50
40
30
20
10
0
fxaa
smaaLow
smaaMedium
smaaHigh
smaaUltra
taaLow
taaMedium
taaHigh
taaUltra
taaExtreme
taaNightmare
Midgard
Bifrost

Links

[1] - http://efficientshading.com/wp-content/uploads/s2015_mobile.pptx

[2] - https://github.com/Themaister/Granite

[3] - https://github.com/KhronosGroup/glTF-Sample-Models/tree/master/2.0/Sponza

[4] - http://advances.realtimerendering.com/s2016/Siggraph2016_idTech6.pdf

[5] - https://www.khronos.org/assets/uploads/developers/library/2017-gdc/GDC_Vulkan-on-Mobile_Vulkan-Multipass-ARM_Mar17.pdf

Thank You

Danke

Merci

谢谢

ありがとう

Gracias

Kiitos

감사합니다

धन्यवाद

תודה

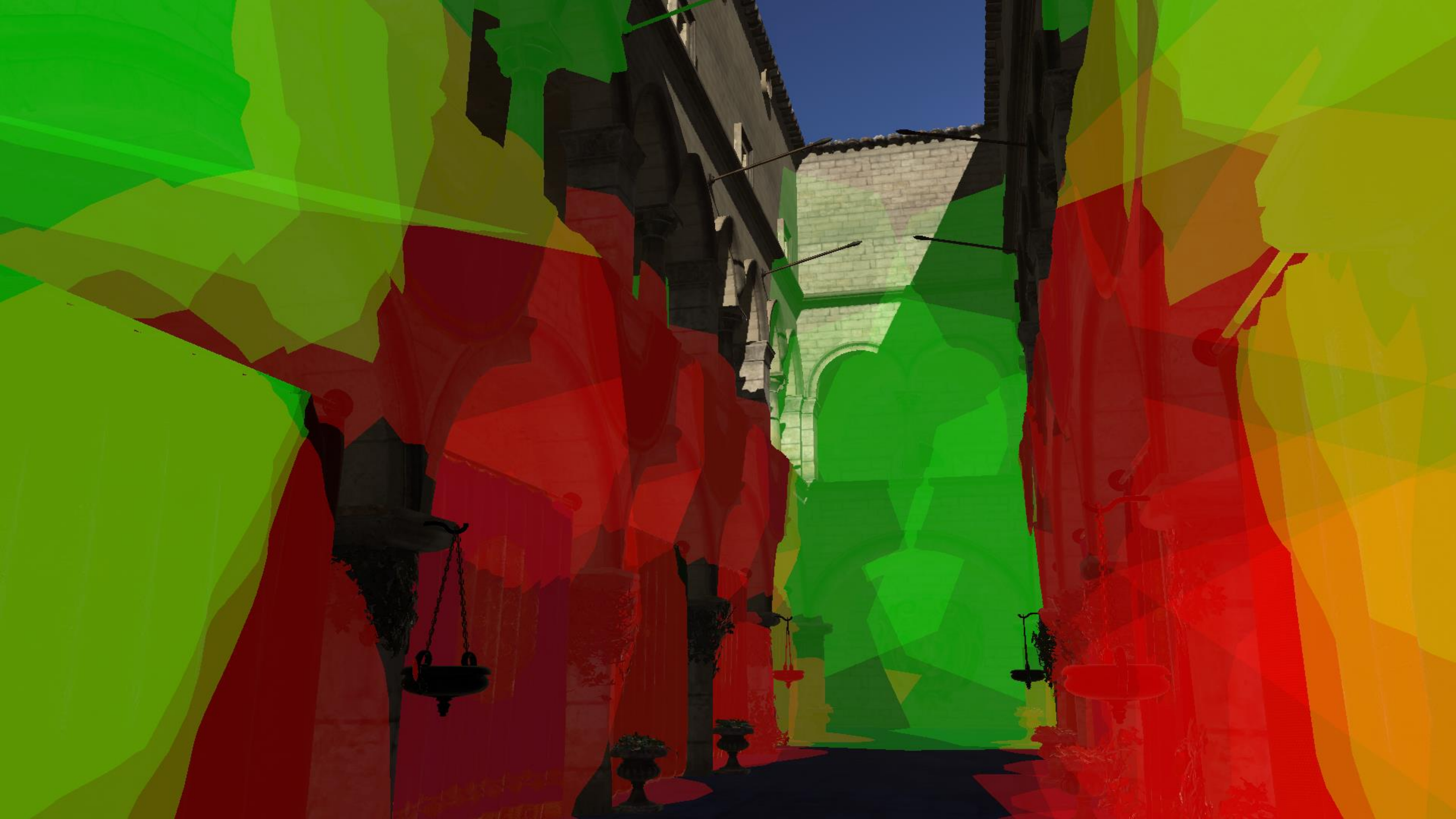
arm

Bonus slides

Clustered stencil culling

Clustered stencil culling

- Basically, bucket the positional lights into N buckets
- Each bucket gets their own stencil bit
  - I used 7, 1 for masking background, YMMV
- Render backfaces with greater test at end of G-buffer pass
  - Each bucket sets their own stencil bit if depth passes
  - Instance all lights in a bucket
- In lighting pass
  - Less-than test, also test stencil read-only if the bucket bit is set.
  - Effectively a conservative double-sided test is achieved.
  - Cuts out a lot of overdraw against background.
  - Lights which clip near plane do not participate in cluster, just use back-face test as-is.



Compared to classic deferred

- Pros
  - Can reduce false positives in shading
- Cons
  - Need to free up some bits in the stencil buffer
  - Some extra early-ZS fill-rate required
  - Some work required to bin lights to stencil bits

Clustered shading

- Has been presented at SIGGRAPH 2015 in the past [1]
  - Unfortunately, no real performance data
- Extremely flexible
  - **Does not need to know depth buffer up-front**
  - Supports marching-like techniques for volumetrics
  - Can be computed on CPU or GPU (I use GPU)
- Fully supports
  - Forward
  - Deferred
  - MSAA
  - Transparency
- To shade
  - Look up (offset, count) or equivalent from a 3D texture based on rasterized position.
  - Iterate and shade lights, lights can be stored in a large buffer.
  - Needs atlasing techniques (or bindless) for shadowmaps.

Implementation

- Async compute shader computes per cell in cluster
  - Low-resolution pre-pass in compute
  - Prunes lights in 4x4x4 blocks before testing at full res.
- Accurate intersection tests are easy
  - Because we have perfect small cubes, we can approximate well treating cube as a sphere
  - No need for conservative raster which is popular for the common «froxel» layout
- Bitmasks limit number of maximum lights
  - Can also use classic «list» of lights for arbitrary amounts, but more expensive to compute list on GPU
  - Shading performance seems the same (within margin of error), so going to leave it at that

Light sweep test

Test how the different rendering algorithms react to increasing number of spot lights:

- Classic deferred (blend light volumes over frame buffer)
- Stencil culling (same as classic deferred, but tries to reduce false positives)
- Forward clustered
- Deferred clustered (all positional lights rendered as a single full-screen quad)
- Forward clustered prepass (On-tile, run prepass depth in same render pass)

Barebones rendering outside positional lights:

- LDR is used to avoid constant overhead of HDR bloom.
- Shadows for directional light is turned off to avoid overhead of rendering shadows.
- No MSAA.

Numbers on mobile are presented with normalized runtime based on:

- GPU cycle counts
- Peak clock speed

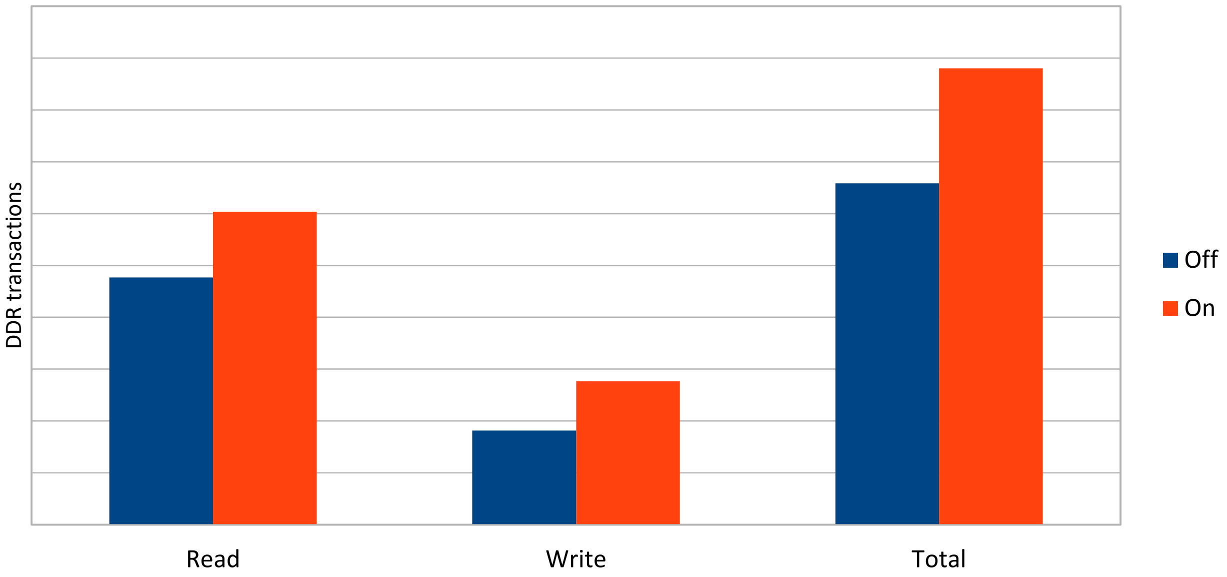
Method sweep test

Tries to look at aggregate results to answer questions like:

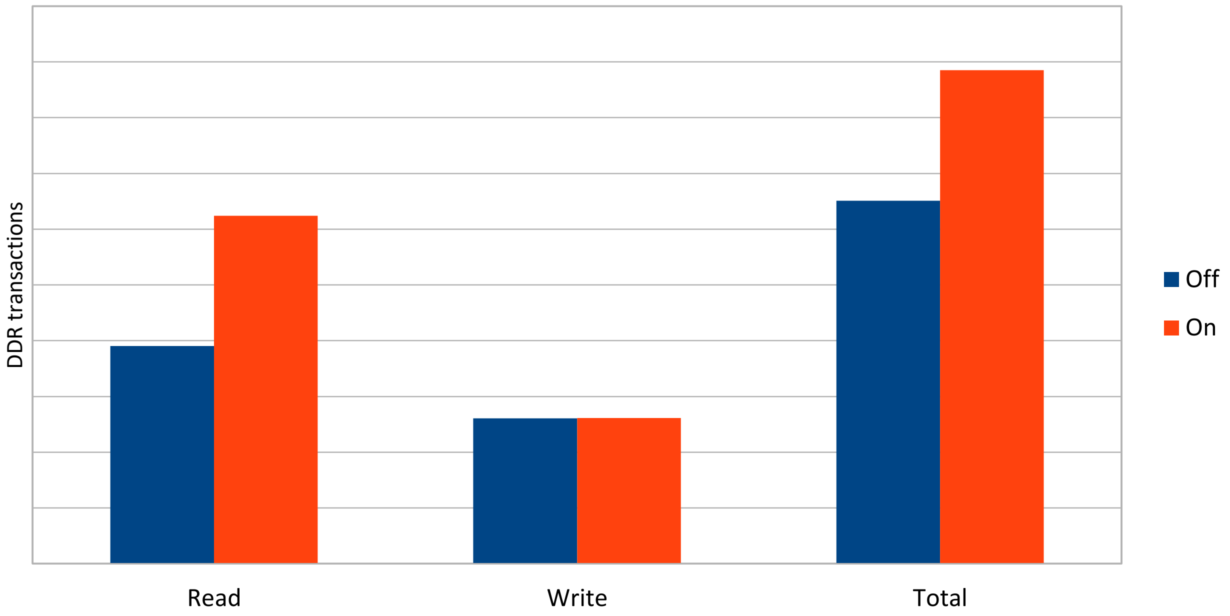
- How expensive are shadows?
- How expensive is VSM vs PCF 1x1?
- Prepass vs no prepass?
- Does anything stick out compared to desktop?

Also have bandwidth numbers captured on the S9+.

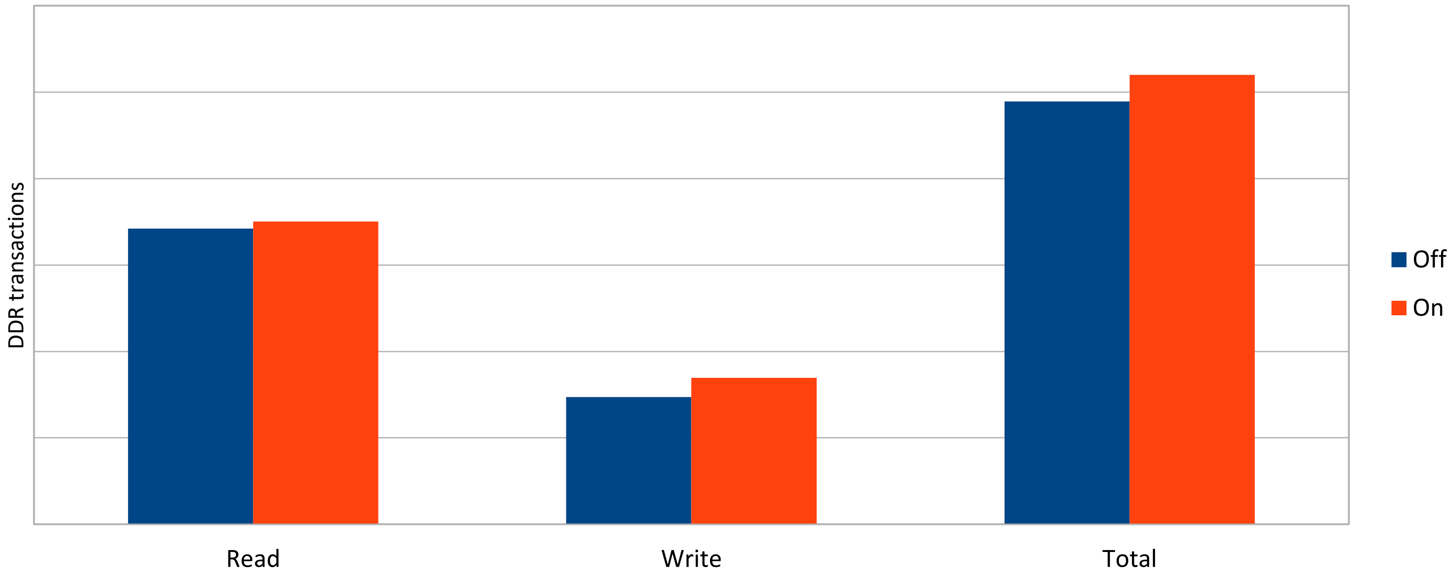
Directional light shadows bandwidth



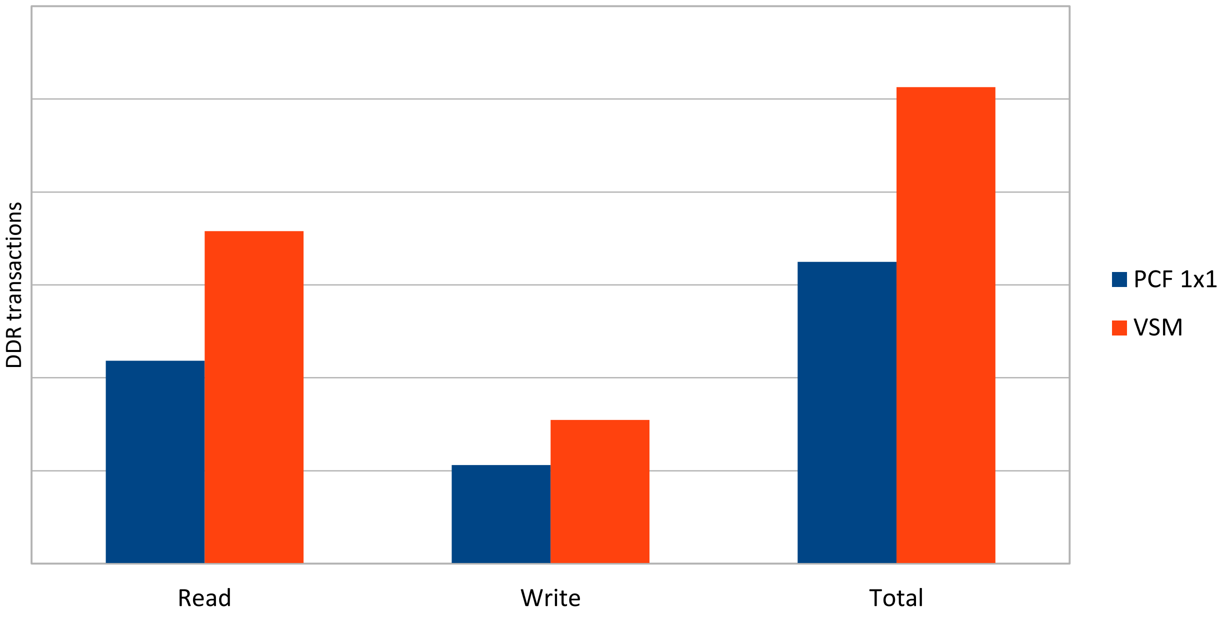
Positional light shadows bandwidth



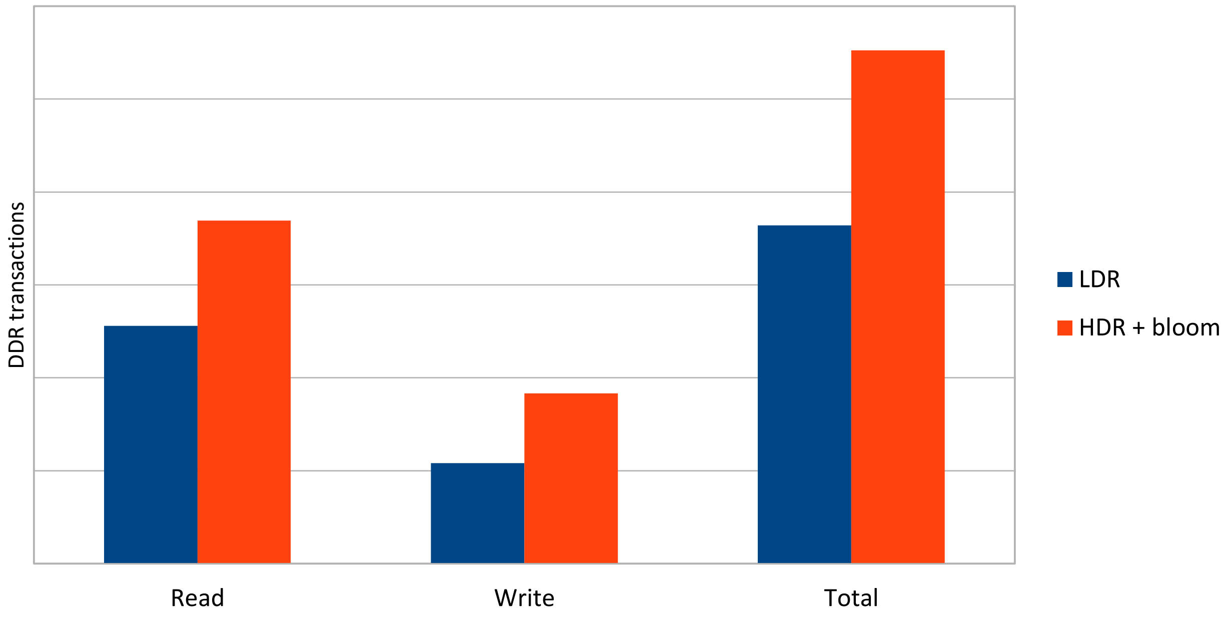
Forward prepass bandwidth



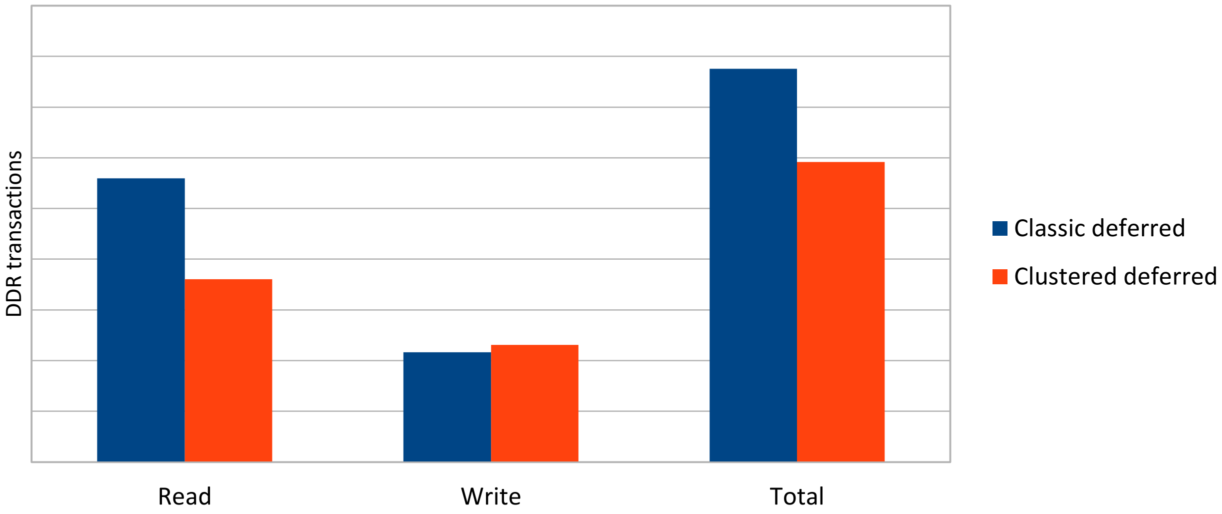
Shadow mapping filter bandwidth



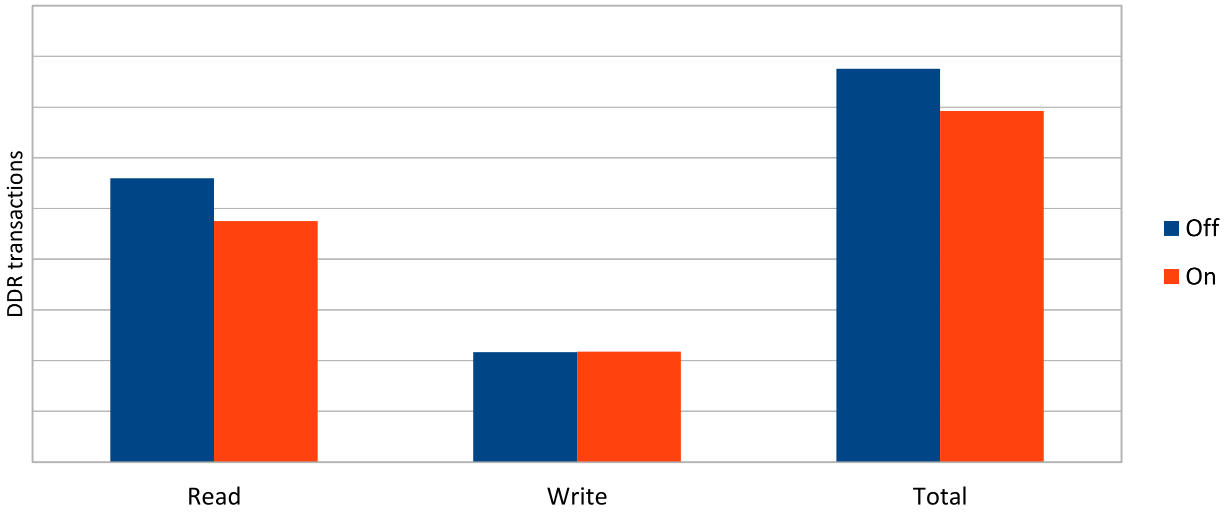
HDR vs LDR bandwidth



Deferred methods bandwidth



Stencil culling bandwidth



Observations

Bifrost deals way better with forward shading than Midgard

- Far more registers available, and scalar instead of vector

Clustered shading is great on Bifrost

- Deferred with multipass or forward, either is good

Forward prepass is surprisingly good

- Gets better with more complex content
- Bandwidth hit isn’t as extreme as I expected
- Expect there to be a cutoff point

MSAA is expensive with denser geometry

- We avoid all the bandwidth hit, but still need to shade a lot more partial quads

~10x gap to mid-range desktop

- At least with this renderer ☺

The usual suspects

- FXAA
  - The 9-tap version
  - It’s light on arithmetic, so it’s basically a 9-tap filtering benchmark ☺
- SMAA
  - Low
  - Medium
  - High
  - Ultra
- TAA
  - There is no de-facto reference TAA implementation
  - I implemented various well-known refinements as building blocks
  - Made some «presets»

TAA variants

- Low
  - RGB input used for clamping
  - No HDR luminance adjustment
  - Neighbor rejection method based on clamping to 5-tap cross
  - Nearest depth / max velocity found from 5-tap cross
- Medium
  - Turns on HDR luminance adjustment so we blend in tonemapped space (reduces flicker)
- High
  - Uses AABB clipping for neighbors (less color squashing in the AABB corner)
  - Rounded corner method to find neighbor color AABB
- Ultra
  - Converts RGB to YCgCo for neighbor clipping purposes (retains hue better)

More intense variants

- Extreme
  - Neighbor clamping method changed to rounded corner with variance clipping
  - Nearest depth / max velocity method bumped to a 3x3 grid
- Nightmare
  - When sampling the history buffer, use a 9-tap bicubic filter
    - Trades a lot of arithmetic to avoid full 16-tap bicubic
    - Massive blur reduction in motion
    - This final shader is about 27 texel fetches per pixel

Observations

- Mobile is 10-20x slower than mid-range desktop
  - The gap is larger than for regular rendering
- Post-AA is still hard to fit into a budget
  - At 720p, FXAA seems reasonable (~1 ms)
- Mali-G71 and G72 are neck and neck on some post-AA
  - Texture pipe throughput is theoretically the same per clock
  - µ-arch improvements show up nicely on SMAA
  - Otherwise, uplift seems to be eaten by texture throughput/bandwidth.
- T-880 (and other Midgard GPUs) don’t scale with large, complex shaders