Attaining Post-Processing Effects on Mid-Range Smartphones

Jose Emilio Muñoz Lopez, Software Engineer, Arm

Đorđe Đurđević, Software Engineer, Nordeus

Unite Berlin

June 20th 2018

Agenda

- Spellsouls
  - Overview
  - Lighting
  - Bloom
- Optimizations
  - Bloom optimization
  - General optimizations
- Arm Tools

About Nordeus

Technical Pillars

AAA Quality

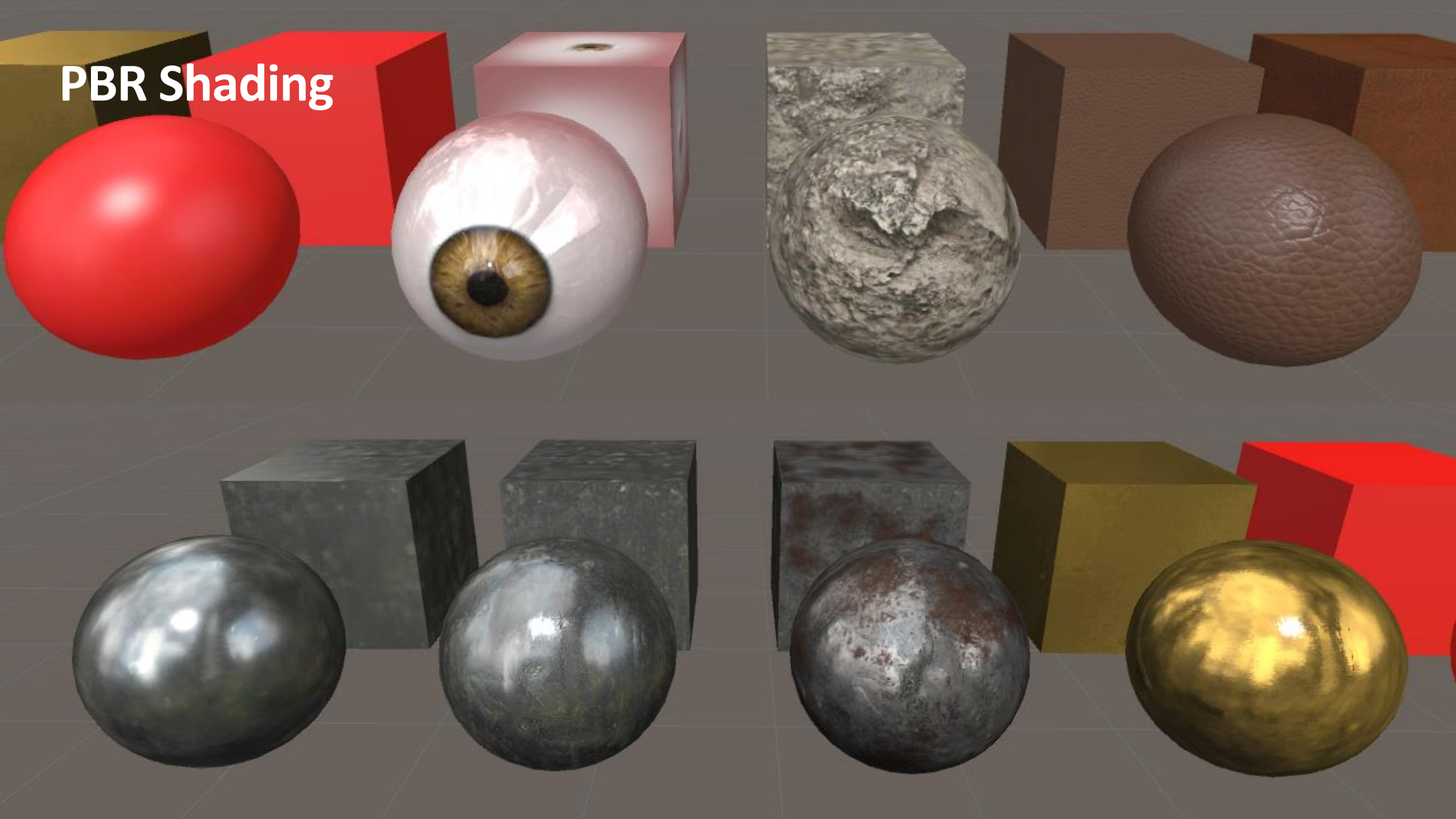
User experience is the most important thing

Wide range of devices

From Galaxy S3 all the way to the latest flagships

PBR Shading

PBR Shading



Lights

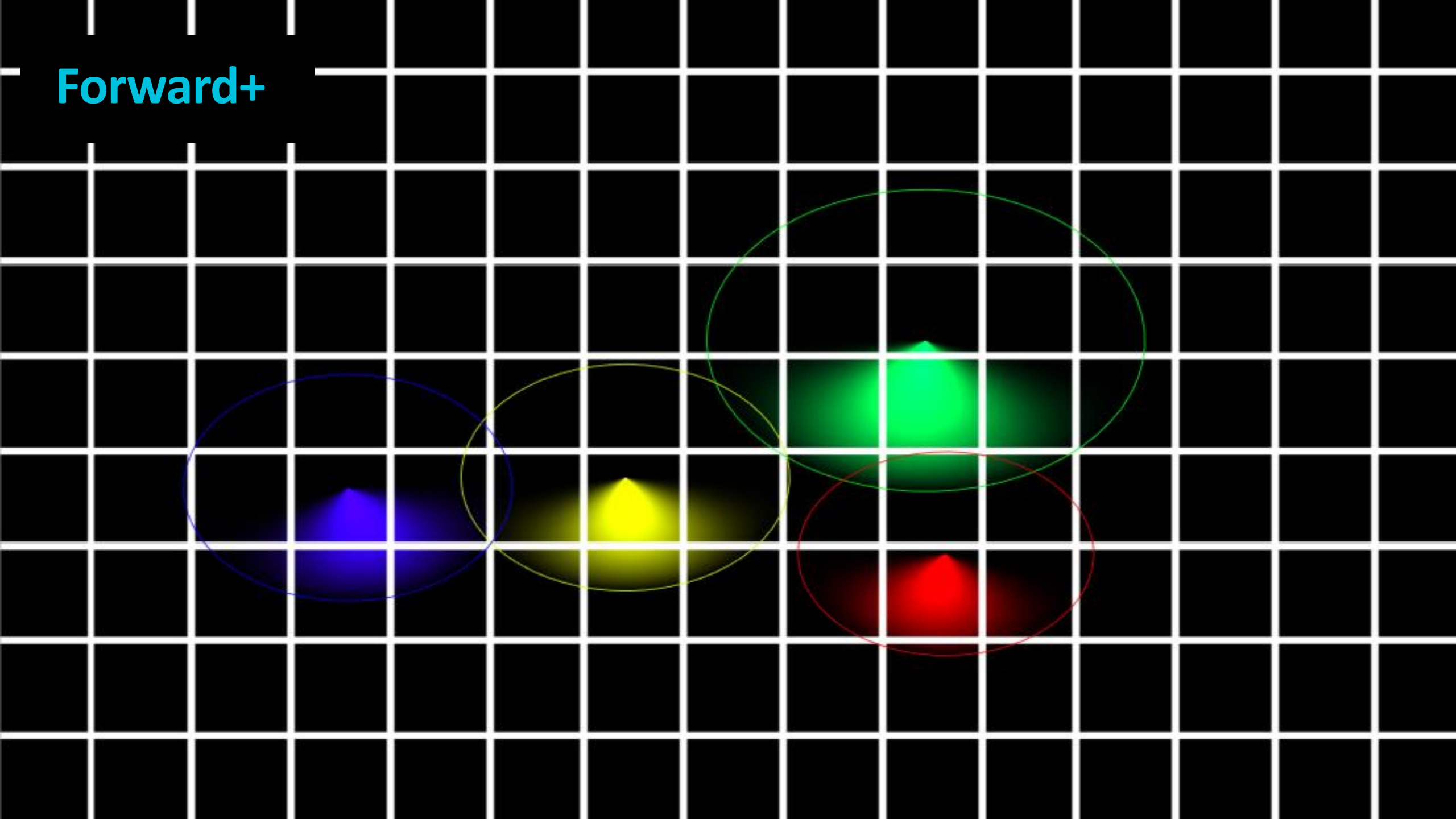
arm

Lights

Specular lights

- Blinn-Phong
- Point lights
- Per pixel

Forward+



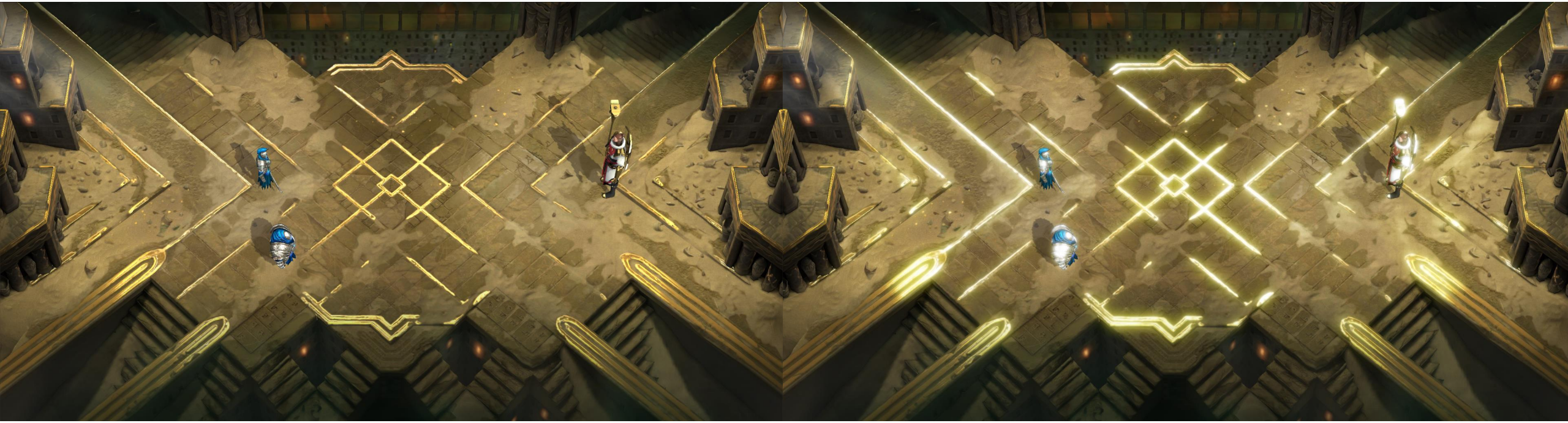
Bloom

arm

Bloom

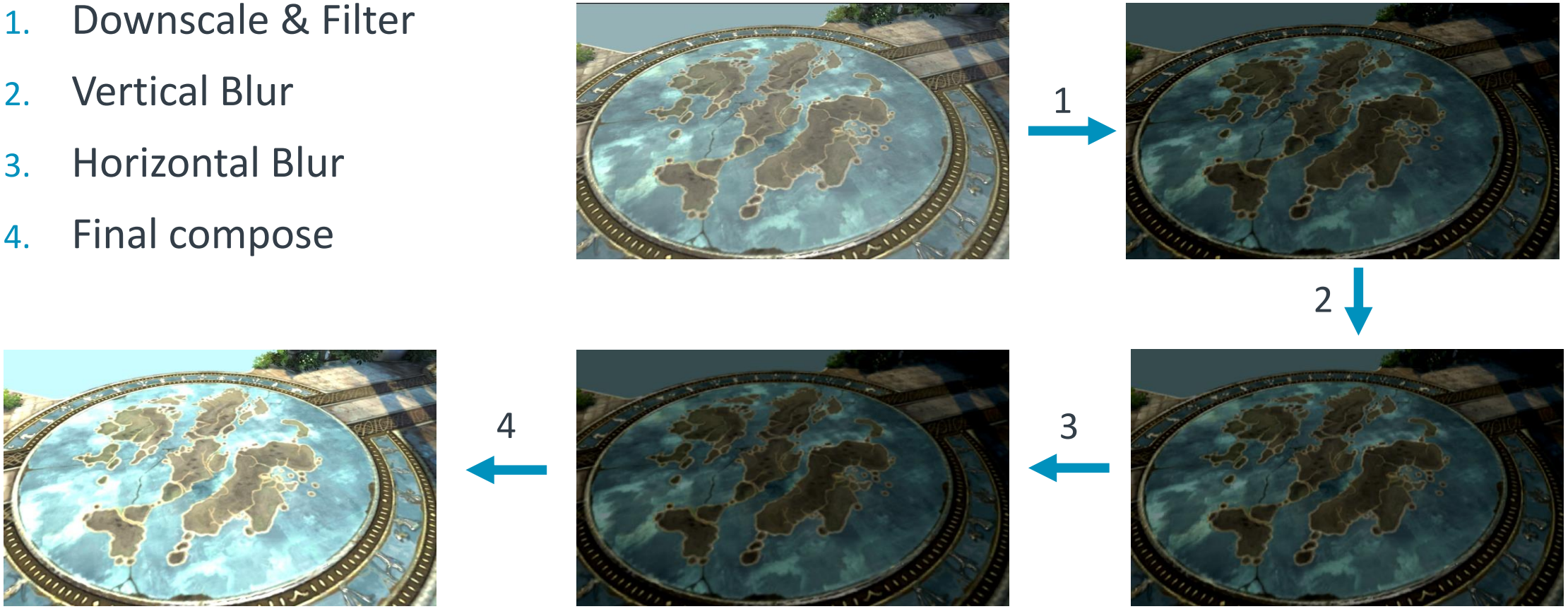
Bloom

Bloom



Postprocess bloom

- 4 pass Bloom
  1. Downscale & Filter
  2. Vertical Blur
  3. Horizontal Blur
  4. Final compose

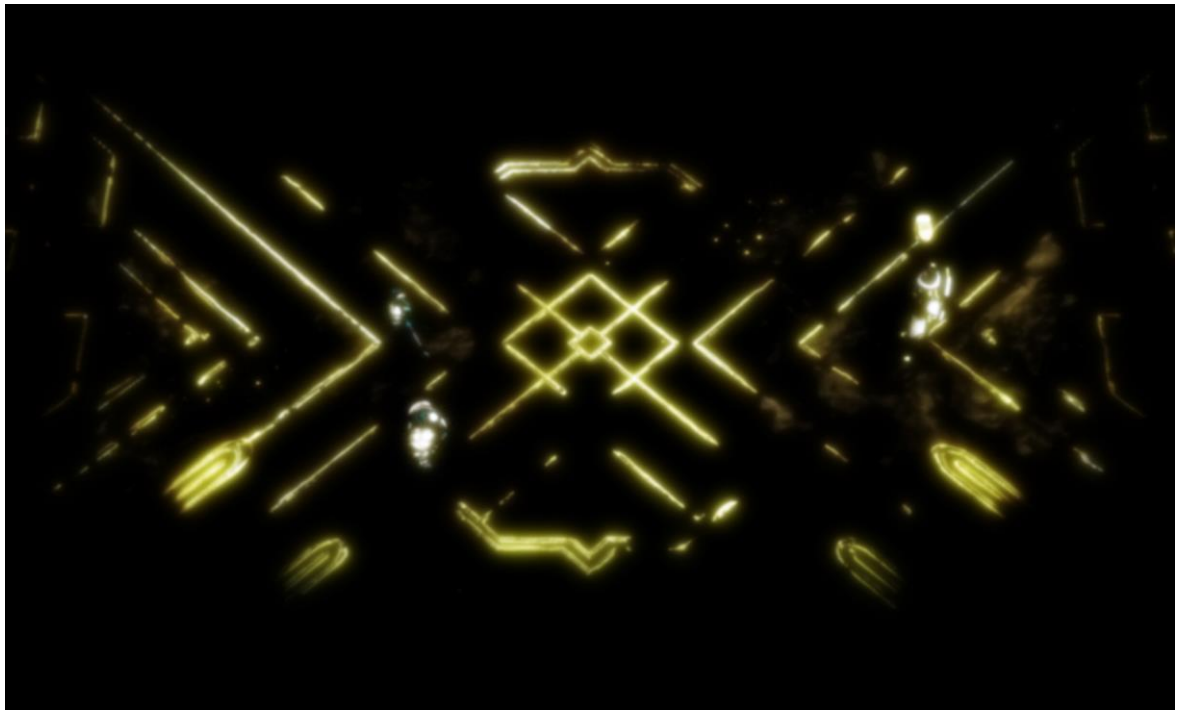


Postprocess HDR? bloom

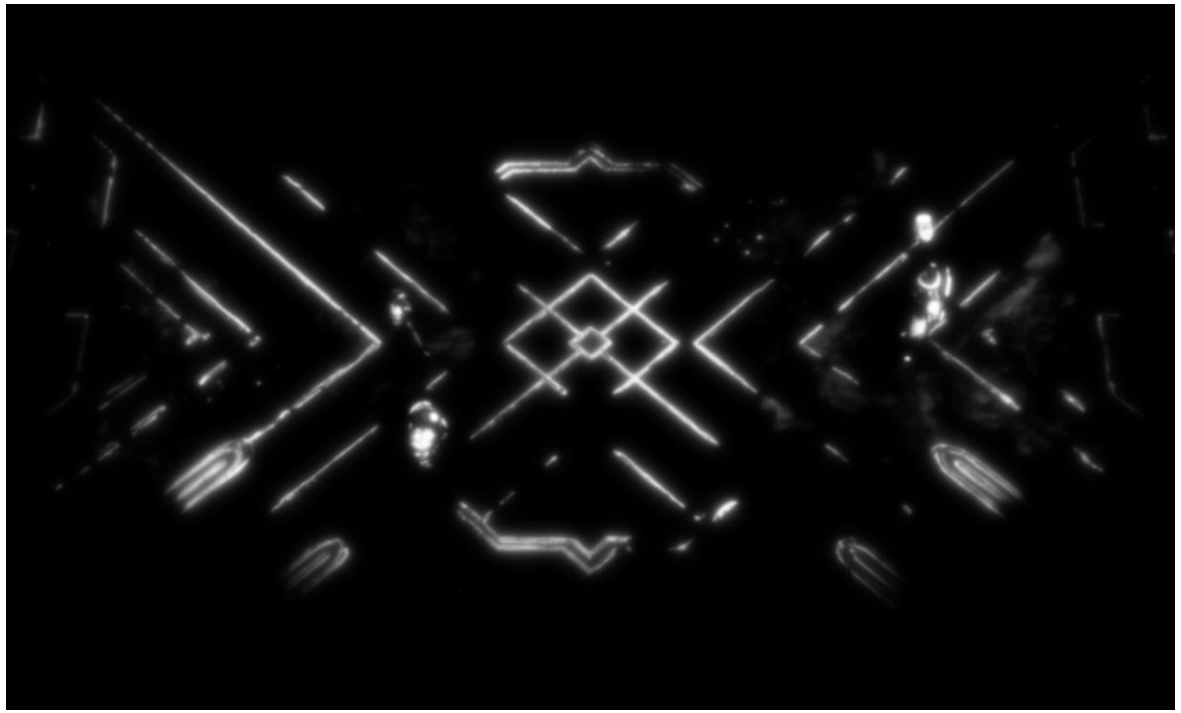
- R11G11B10 framebuffer format?
- Expensive
- No alpha

Multiple Render Target Bloom

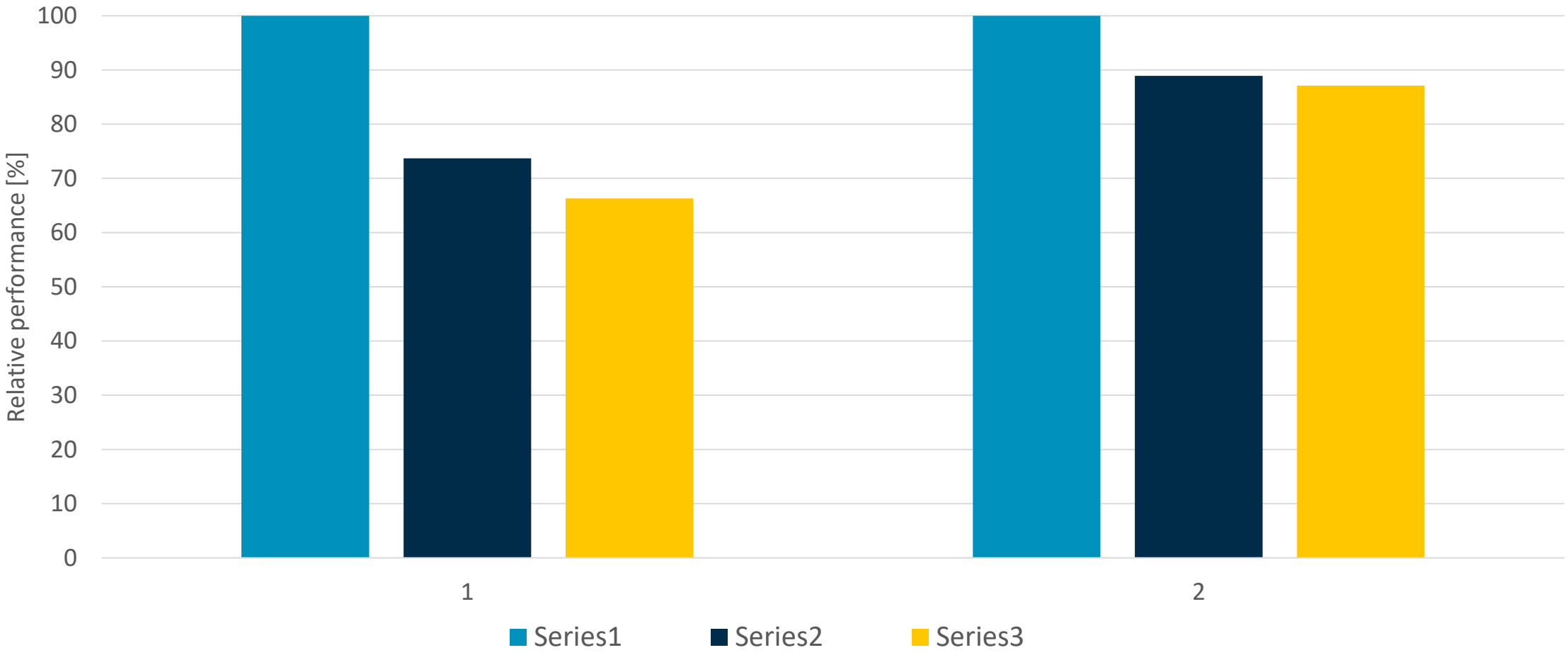
RGB24



R8



Performance



Bloom optimizations

arm

The big picture

- Our target is rendering the whole scene in 16 ms (60 FPS)

Base

+

Bloom

=

Complete frame

16 ms

(target)

The big picture

- We want to reduce the contribution from bloom

Base

+

Bloom

3 ms

(Multiple Render Target)

=

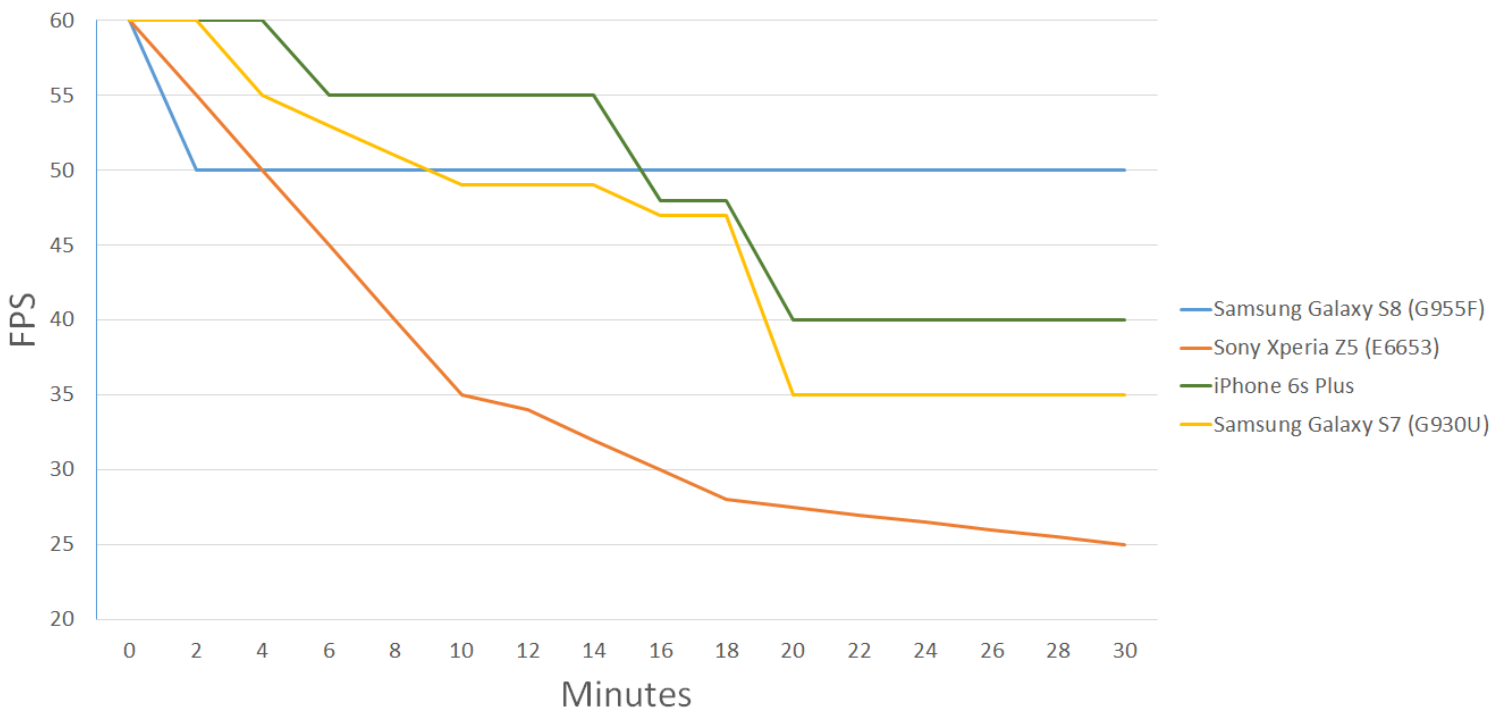
Complete frame

16 ms

(target)

The challenge

- Power budget, energy budget
- Areas explored:
  - Optimize post-processing pipeline
  - Simulate bloom in other ways



Optimizing MRT bloom

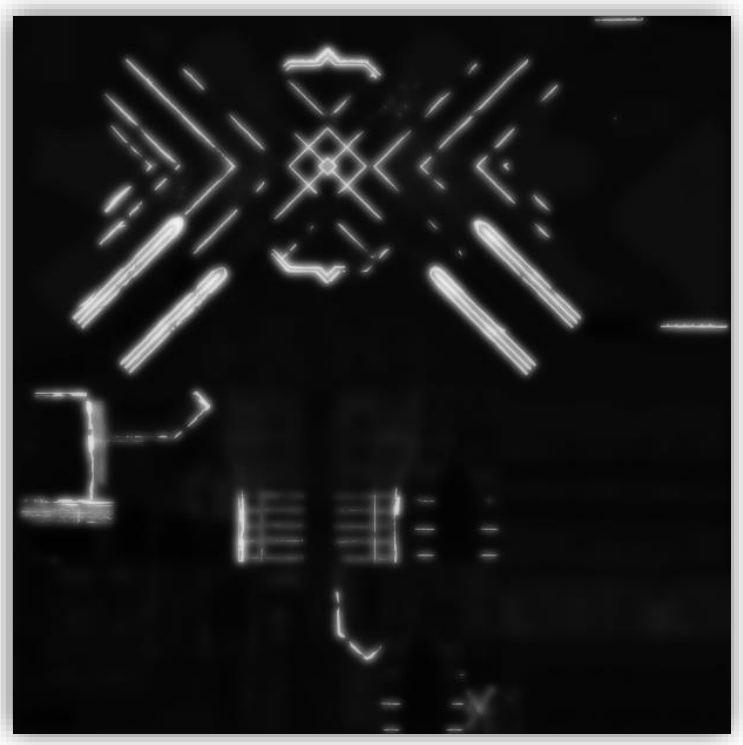
- Blur is the most expensive step in the bloom pipeline
- Shaders need to sample more than one pixel to blur
- Dual filtering approach
  - Downscale and blur several times, then upscale
  - Much more efficient than Gaussian at full resolution (14x)
  - Comparable performances with downscaled Gaussian, but nicer effect

"Bandwidth Efficient Rendering", Marius Bjørge (Arm), Siggraph 2015

Texture-based bloom

- Generate a bloom intensity map, based on the PBR glossiness map
- Save it in the alpha component of the glossiness map itself

Glossiness map



Bloom map

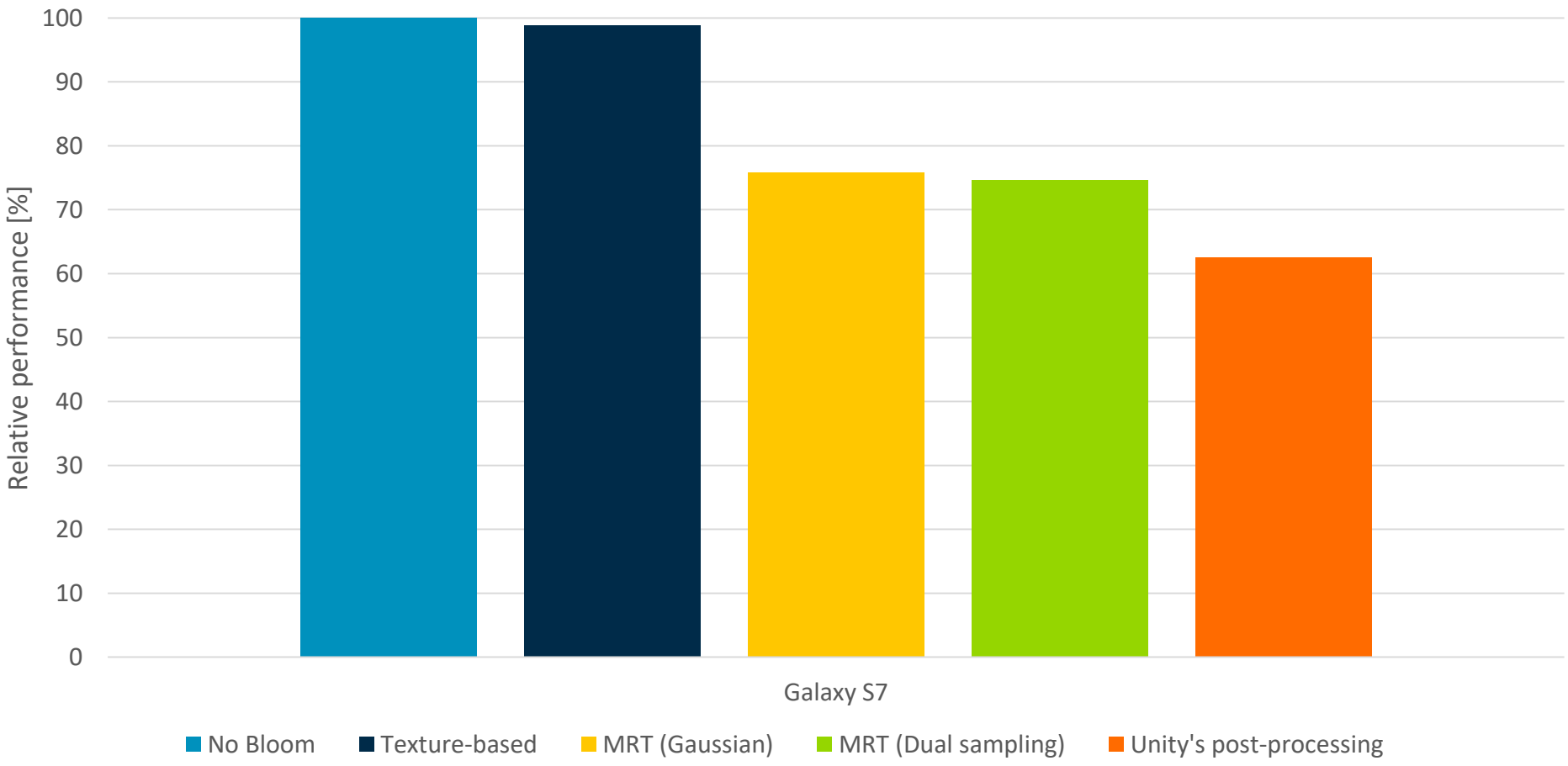
Texture-based bloom

- When rendering, make a pixel brighter based on:
  - the bloom map we previously baked
  - the angle between the eye vector and the light
- Minimal cost since we are already fetching the RGB components of the glossiness map

```
// Vertex shader
float lightObjCameraAlignment = dot(objToCam, reftLightDir);
half alignmentFactor = clamp(lightObjCameraAlignment, 0.0, 1.0);

// Fragment shader
half bloom = rawGlossMap.a;
finalColor += finalColor * bloom * i.alignmentFactor * _BloomStrength;
```

Terrain performances




No bloom

Unity’s post-processing

MRT (Gaussian)

MRT (Dual filtering)

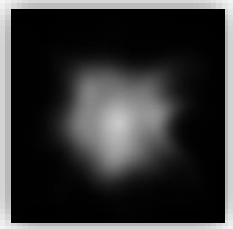
Texture-based

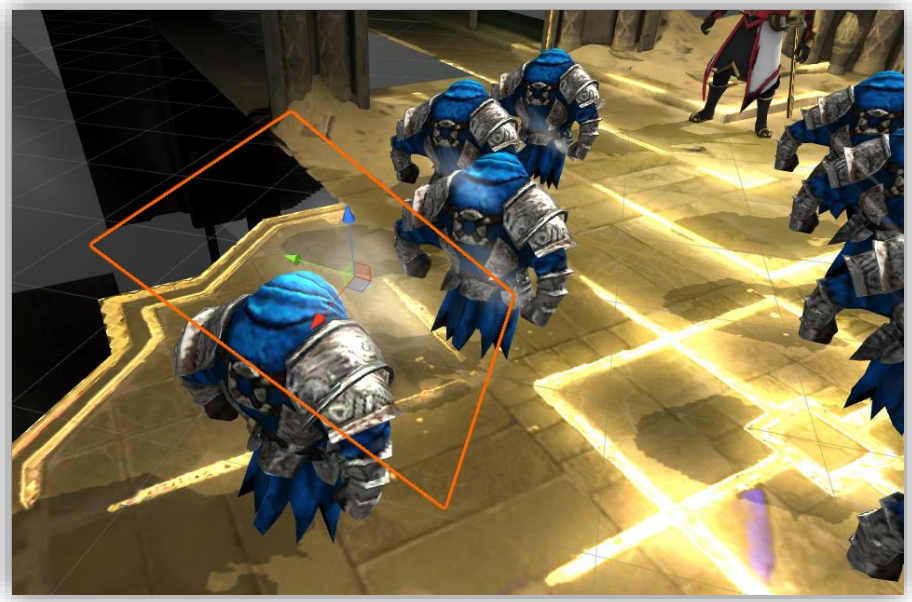
Texture-based bloom

- Does not work so well on the characters
- Light stays within the character, does not bloom outside

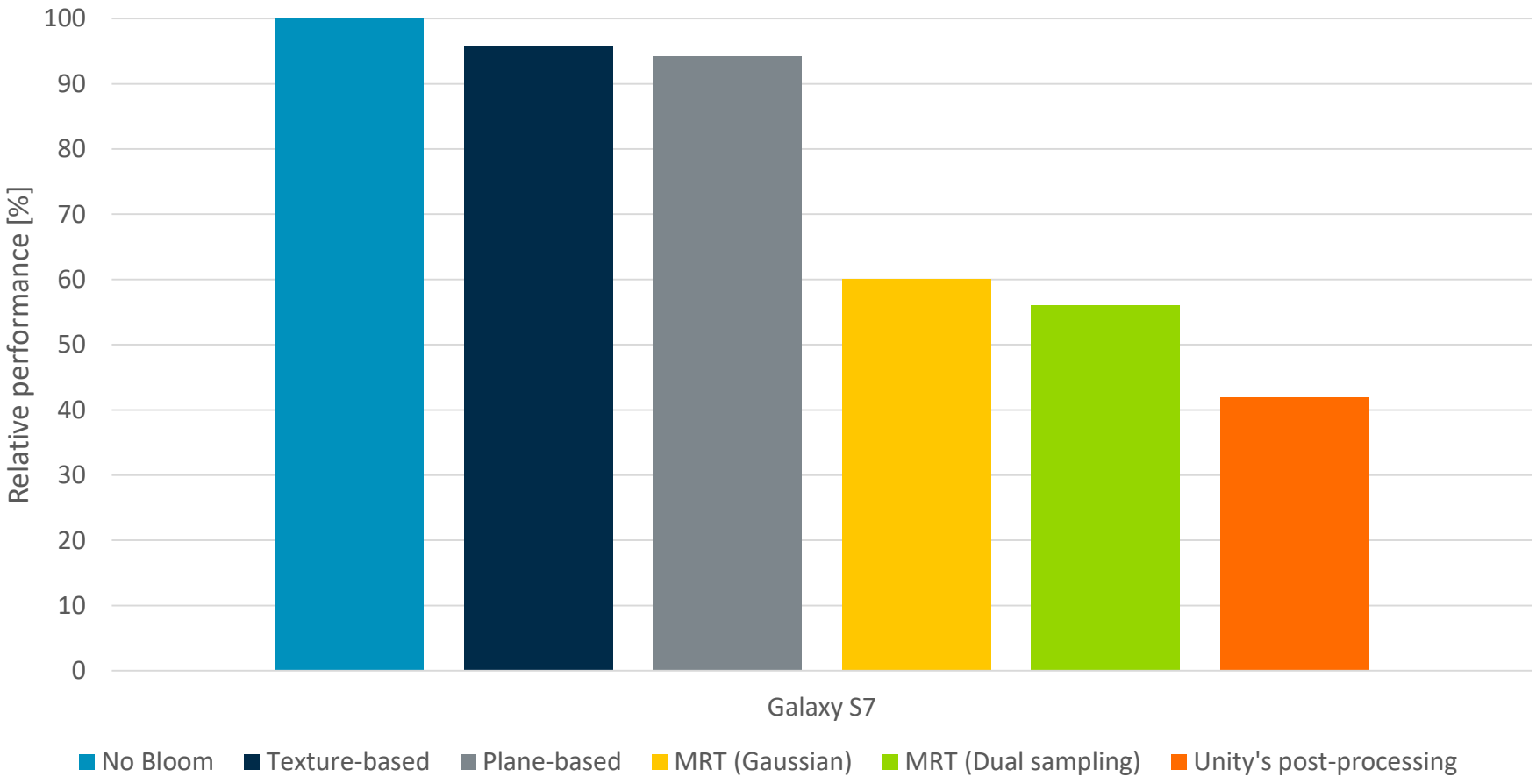
Bloom using a plane overlay

- A billboard on each character overlays a ‘bloom texture’
- Always facing the camera, intensity based on light angle
- Light blooms out





Characters performances



The big picture

- With texture-based and plane-based approaches we brought bloom down from 3 ms to less than 1 ms

Base

+

Bloom

< 1 ms

(texture/plane based)

=

Complete frame

16 ms

(target)

The big picture

- What's our budget for bloom?

Base

?

+

Bloom

< 1 ms

(texture/plane based)

=

Complete frame

16 ms

(target)

The big picture

- …

Base

28 ms

+

Bloom

< 1 ms

(texture/plane based)

=

Complete frame

16 ms

(target)

The big picture

- No budget for bloom, no matter how optimized
- We actually optimized the rest of the game first!
  - Need to make space for bloom
  - Easier to achieve significant savings

Optimization best practices

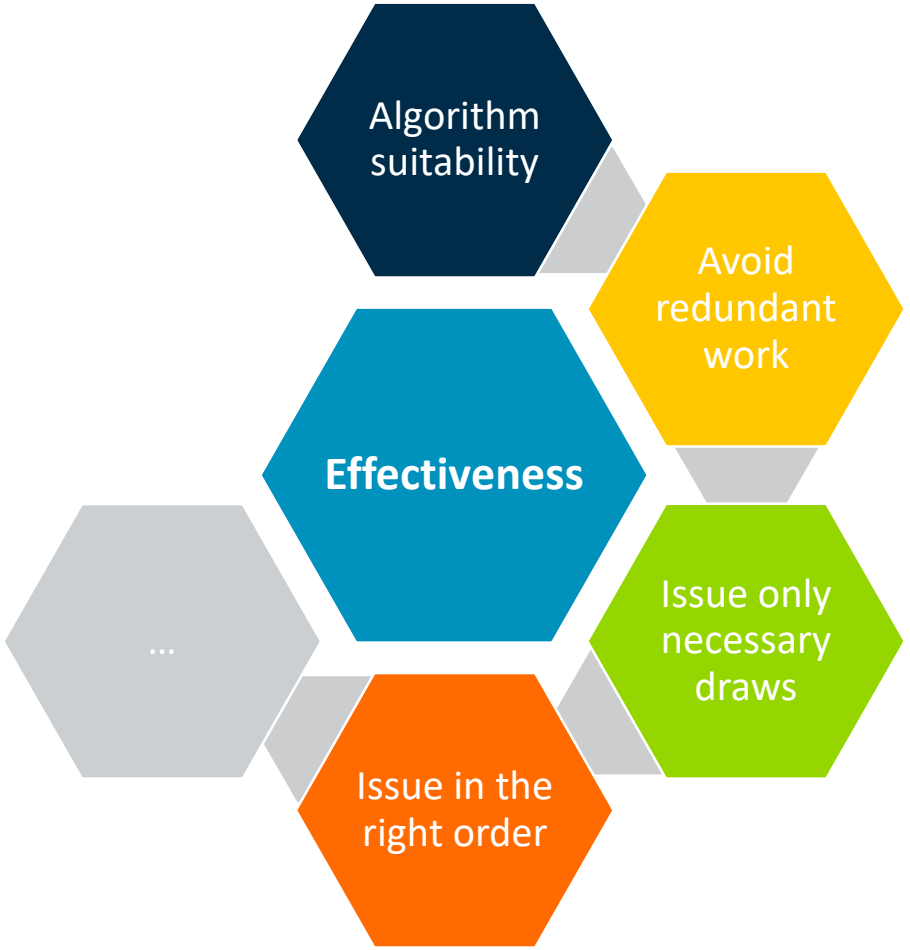
arm

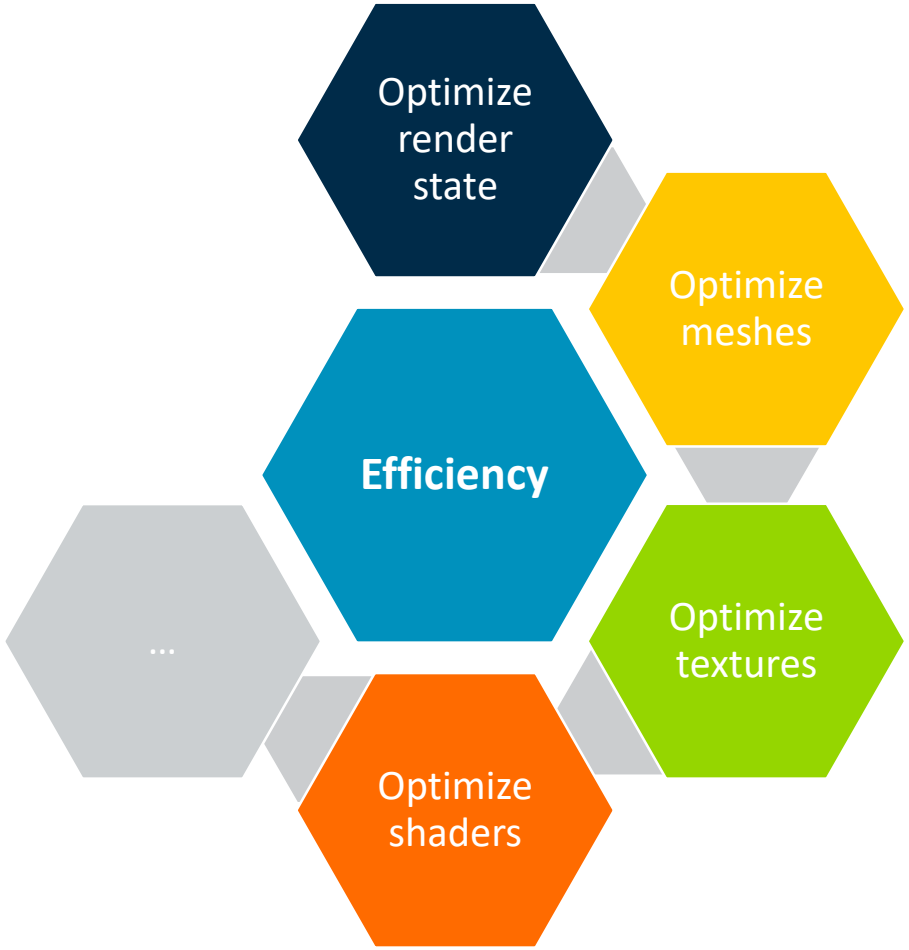
“Efficiency is doing things right

Effectiveness is doing the right things”

- Peter Drucker

Effectiveness and efficiency





Optimizing Spellsouls

- The game is GPU-bound
- Fragment shaders: what’s the heaviest part?

Optimizing Spellsouls

Optimizing Spellsouls

- Correct answer: profile the game and figure it out
- Turns out the terrain was the heaviest part!
- It covers most of the screen, any improvement will have a large impact

Terrain optimizations

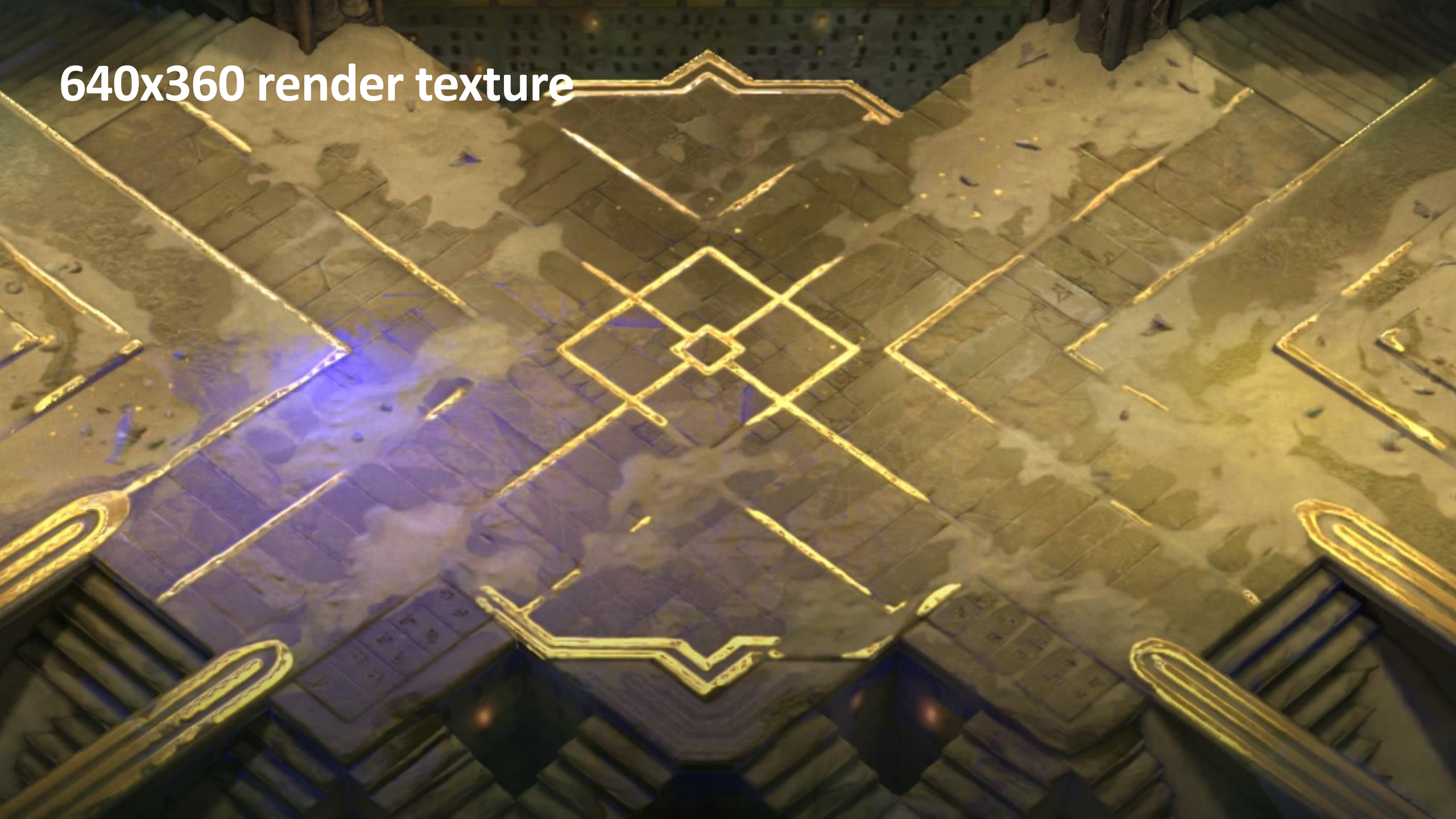
arm

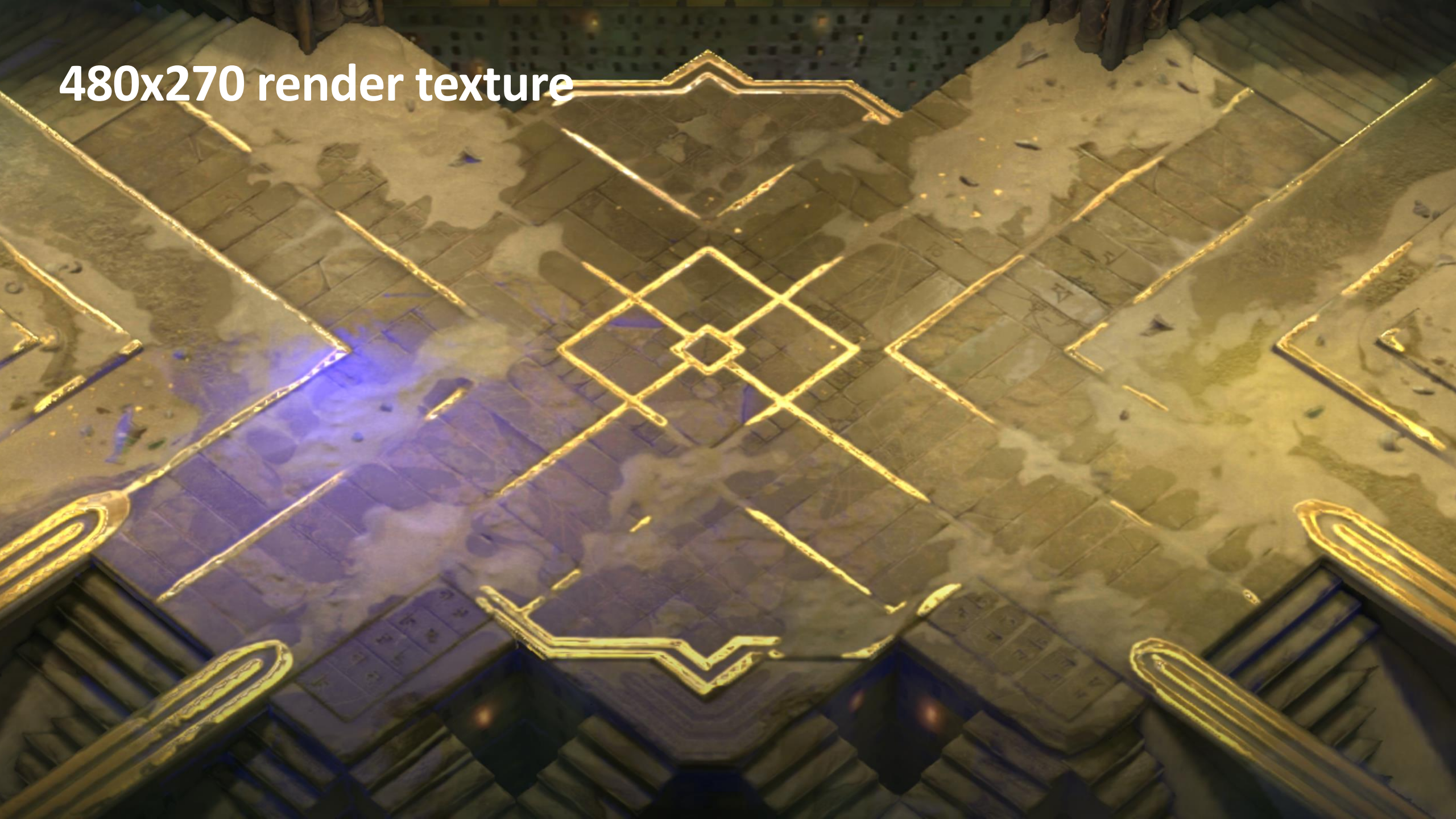
Fragment time contributions

Tangent-space normal maps	Reflections	Lighting
1.2 ms	4.4 ms	6.1 ms
Moving to world-space normals	Don’t want to reduce quality – crisp reflections	Good space for optimization

Lightmap resolution

- Lighting can be moved to a lower resolution
- We used a 640x360 render texture
- Lights are rendered once and then sampled from the PBR shader
- Rendering time from 14.3 ms to 10 ms


640x360 render texture


480x270 render texture


Extra lights!

The big picture

- Lower resolution lightmaps bring us closer to our goal

Base

21 ms

+

Bloom

< 1 ms

(texture/plane based)

=

Complete frame

16 ms

(target)

Terrain resolution

- Terrain rendering can be moved to a lower resolution
- We used 720p instead of 1080p (55% less pixels)
- Don’t downscale the whole game, just the terrain
- Rendering time from 10 ms to 5 ms

Original scene

Optimized scene

The big picture

- We can now reach 60 fps at a sustainable frequency on S7

Base		Bloom		Complete frame
	+		=	
16 ms		< 1 ms (texture/plane based)		16 ms (target)

Arm Developer Tools

arm

Mali GPU Tools

Performance Analysis, Debug, and Software Development

Mali Graphics Debugger

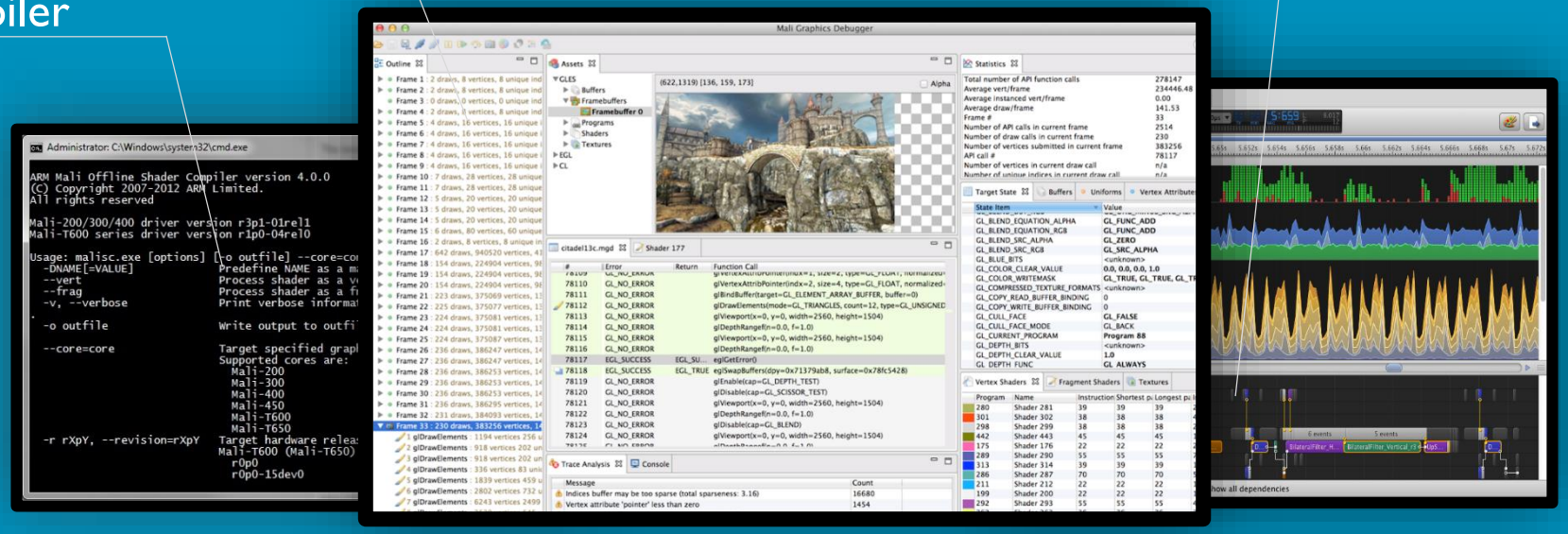
- API Trace & Debug
- OpenGL ES, OpenCL
- Debug and improve performance at frame level

Mali Offline Compiler

- Analyze shader performance
- Command line tool
- Number of cycles
- Registers utilization

ARM DS-5 Streamline

Profile CPUs and Mali GPUs
Timeline
HW Counters
OpenCL visualizer



OpenGL ES Emulator

- Emulate OpenGL ES 2.0, 3.1
- Supports Android Extension Pack
- Windows and Linux
- Benchmarked against Khronos Conformance Suite

Texture Compression Tool

- Command line and GUI
- ETC, ETC2, ASTC, 3D textures

ASTC encoder

- Available on GitHub

Key takeaways

- Specialize techniques for your specific use case
- Worth exploring alternative effects that don’t involve post-processing
- Always look at the game as a whole when optimizing it
- Let tools help you optimize your game

Want to know more?

Arm Mali Developer Guides & Tools

Developer.arm.com/graphics/

Thank You!
Danke!
Merci!
谢谢!
ありがとう!
¡Gracias!
Kiitos!
감사합니다
धन्यवाद

arm