

# Rendering Roblox

Vulkan Optimisations on  
PowerVR

**Presented By:**  
Arseny Kapoulkine  
David Hamill

# What is Roblox?

- Online multiplayer game creation platform
- All content is user generated
- Windows, macOS, iOS, Android, Xbox One
- 100M+ MAU, 2.5M+ CCU





ENTER SEASON 3



You can disguise your vehicle in the garage.



Player	Money
Player 1	3,500
Player 2	40,100
Player 3	5,375
Player 4	5,300
Player 5	425,000
Player 6	175,000
Player 7	100,000
Player 8	14,880
Player 9	9,800
Player 10	9,800
Player 11	9,800
Player 12	9,800
Player 13	2,775
Player 14	750
Player 15	175,000
Player 16	86,750
Player 17	18,450
Player 18	13,400
Player 19	9,200
Player 20	1,425
Player 21	1,540
Player 22	1,000
Player 23	800





zeuxcg

HP 542/542

EXP 2044/2277

28

63620



Show Friends

Show Players

VIP

28

zeuxcg

542/542

Pirate Defender

Block Destroyer

Let's Yank That!

67

NEW!



Cosmetics

Inventory

Play











# Graphics scaling

- Substantial differences between devices
  - PowerVR SGX543: Apple iPad 2 (2011), Samsung Galaxy S4 (2013)
  - PowerVR GM 9446: Oppo Reno Z (2019)
- Developers make the content once
- It's our job to run it well\*

# Full spectrum design

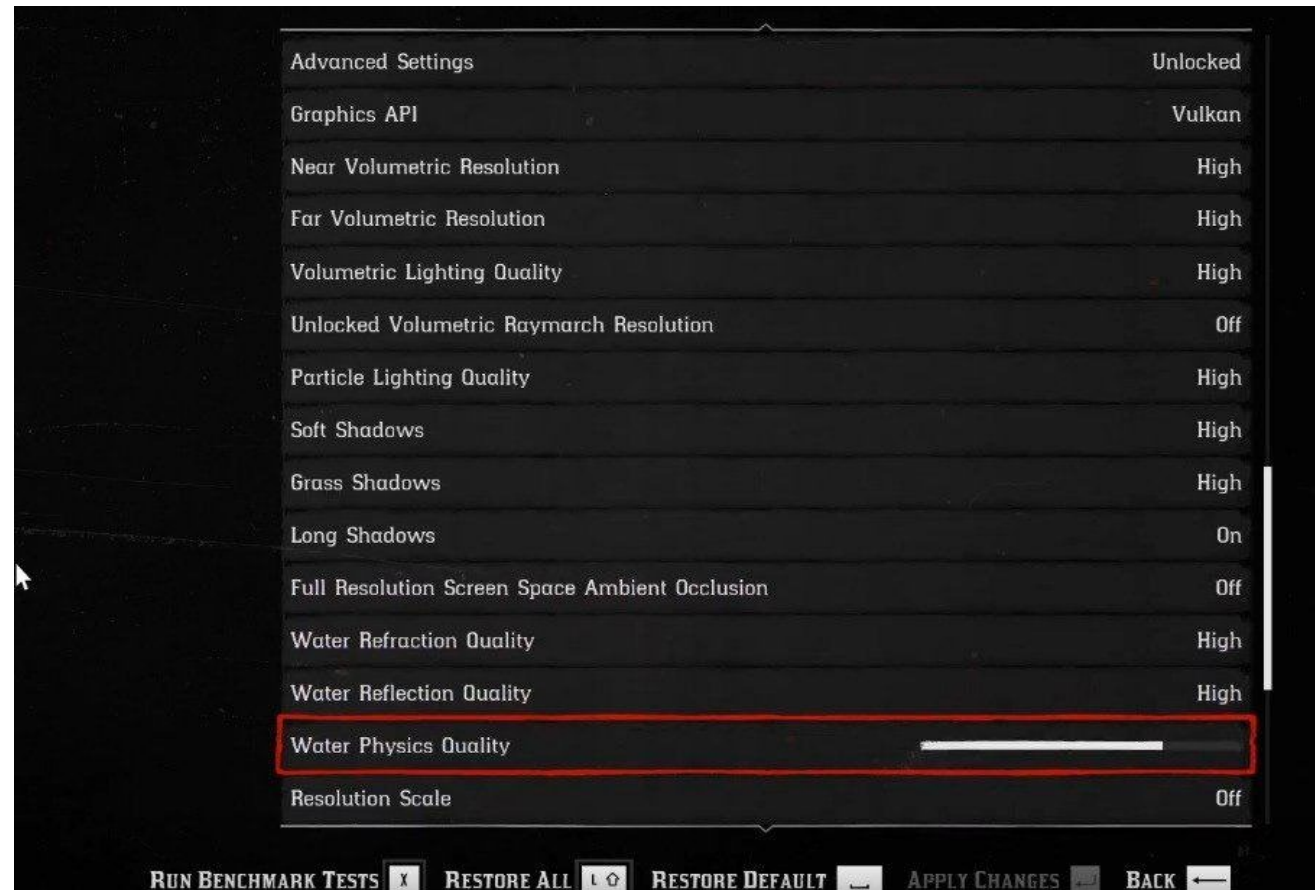
- How do you implement a graphics feature?
  - Build for the lowest common denominator and upscale?
  - Build for the highest end GPU and optimize?
- Build a feature for the lowest common denominator and upscale?
  - Pro: Consistent behavior
  - Con: Poor quality
- Build a feature for the highest end GPU and optimize?
  - Pro: Great quality on high-end
  - Con: Poor performance on low-end, inconsistent behavior, battery/thermal limits



# Full spectrum design

- How do you implement a graphics feature?
  - Build for the lowest common denominator and upscale?
  - Build for the highest end GPU and optimize?
- Sometimes we pick one and iterate
- Prefer to design for the entire spectrum from the beginning

# Quality vs performance



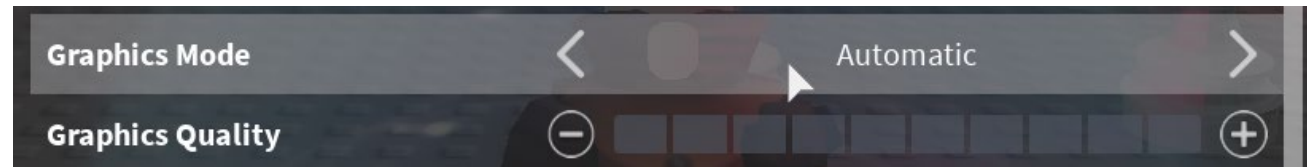


# Quality vs performance

- Advanced settings give users lots of control! However...
- Suboptimal performance for users who can't tune these well
  - ... or know about them! Especially on mobile.
- Performance is often counter-intuitive
  - CPU vs GPU bottlenecks
  - Example: denser grass is faster
- Different games have different performance characteristics

# Quality vs performance

- We have a single quality slider
  - ... arguably we made it too simple and need a separate “draw distance” slider
- By default, we automatically tune it based on observed performance
  - Do we have headroom on CPU vs GPU?
- All graphics effects are using this to balance quality vs performance
  - NB: Internally the slider is more granular than it is in UI
- Less control for the user, but easier for developers and graphics engineers





# Quality vs performance

- Depending on the effect, scaling strategy can be different!
- Disable effect completely
  - Possible if not gameplay critical
  - Example: SSAO
- Geometry / shading LOD
  - Reduce computational cost while reducing quality
  - Important to keep the overall color balance!
- Temporal throttling
  - Visual pops due to late updates
  - Examples: lighting, terrain LOD

# Lighting system: goals

- No baking. Every object can move at any point in time.
- Can't disable lighting altogether - large visual and gameplay implications
- Want minimal "performance" annotations from content
- Want to run on low-end laptop/mobile (D3D9 / GL2 / GLES2 class)
- Want good looking lighting on high-end laptop/mobile

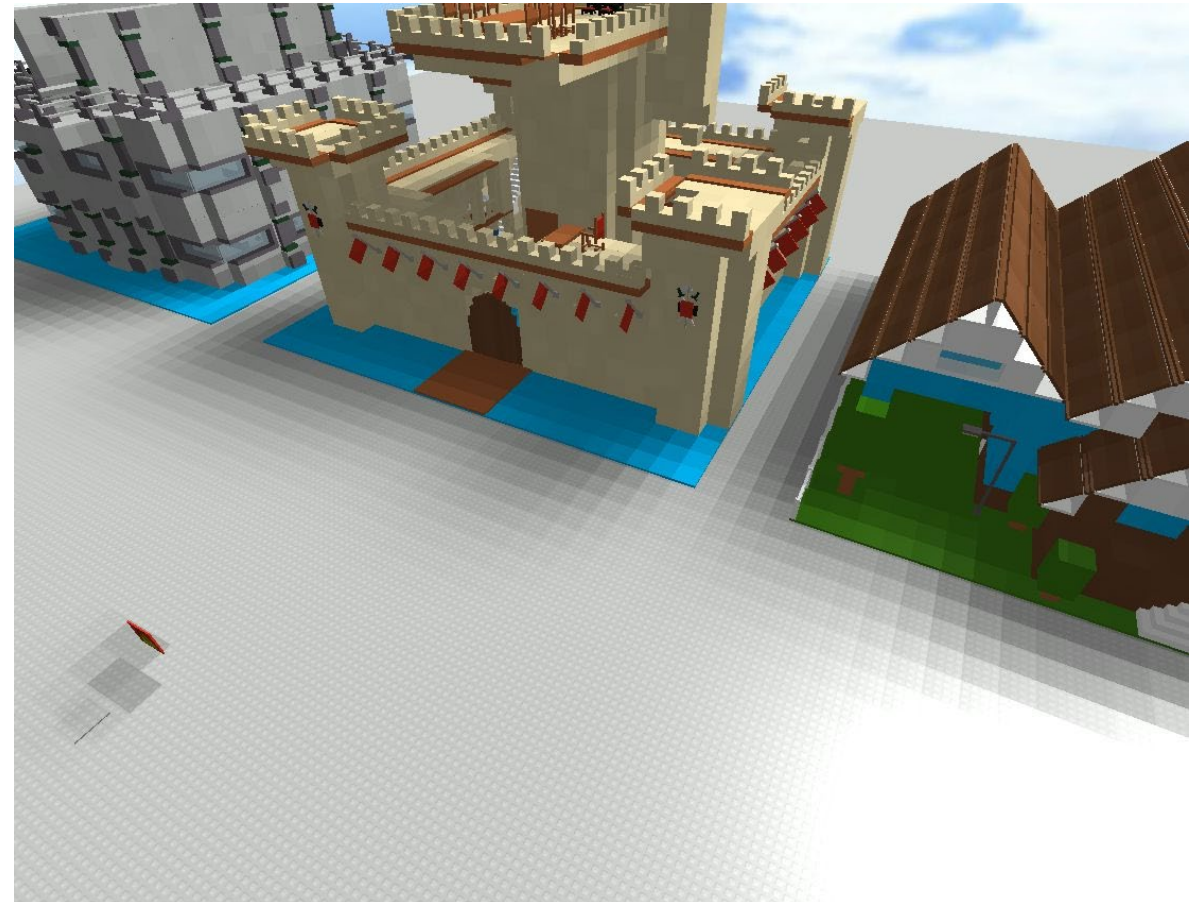


# Lighting system: high level overview

- Features
  - Light sources: sun/moon, sky, local lights (point, spot, surface)
  - Geometry and light sources are dynamic
  - All light sources can cast shadows
- Today
  - Coarse voxel lighting: runs everywhere; efficient, soft lighting
  - Shadow maps: runs on mid-end to high-end; high quality sun shadows
- Future
  - Forward+: high-end, high quality local lights
  - Better voxel lighting: better skylight

# Lighting system: phase 0, voxels

- Most of the work is done on CPU
- Voxelize all geometry dynamically
- Compute light impact per voxel
  - Sunlight
  - Skylight
  - Accumulated local light RGB
- Upload the information to GPU
  - RGB:  $\text{light} + \text{skylight} * \text{skycolor}$
  - A: sunlight
- Final lighting in fragment shader



# Lighting system: phase 0, voxels

- Managing CPU performance
  - Voxel grid is split into chunks
  - Only a handful of chunks update each frame
  - Fixed quality, but updates can be “stale”
  - Update kernels use hand-optimized SSE2 / NEON
- Managing GPU performance
  - A single 3D texture lookup with bilinear\* filtering
  - GLES2: emulate 3D texture lookup with 2 2D texture atlas lookups
- Fully decoupled geometry vs light complexity



# Lighting system: phase 1, better voxels

- Keep the overall system design
- HDR light color
  - Encoded using RGBM to save space
- Store skylight separately
  - Better integration of sky into BRDF
- Anisotropic occupancy
  - Voxelizer keeps 3 axial values per voxel
  - Critical for content compatibility
- Results are closer to shadow maps/Forward+



# Lighting system: phase 1, performance

- CPU cost is larger but manageable
  - Anisotropic occupancy rasterization is more expensive
  - RGBM encoding is almost free
- GPU cost is still relatively small
  - Fetch two textures
  - RGBM decode (future: RGB9E5 / R11G11B10F?)
- CPU-GPU upload
  - We upload 1-4 chunks each frame
  - Each chunk is 32x32x16 volume (16K voxels) = 128KB per chunk
  - vkCmdCopyBufferToImage copies 128KB-512KB of data from a VkBuffer each frame
  - ... is this fast?

# Analysing Roblox on PowerVR GPUs

# What is Imagination Technologies?

PowerVR mobile GPUs, *Rogue* architecture and new *A-Series* architecture

VideoLogic discrete graphics

Arcade machines,

Sega Dreamcast,

PS Vita

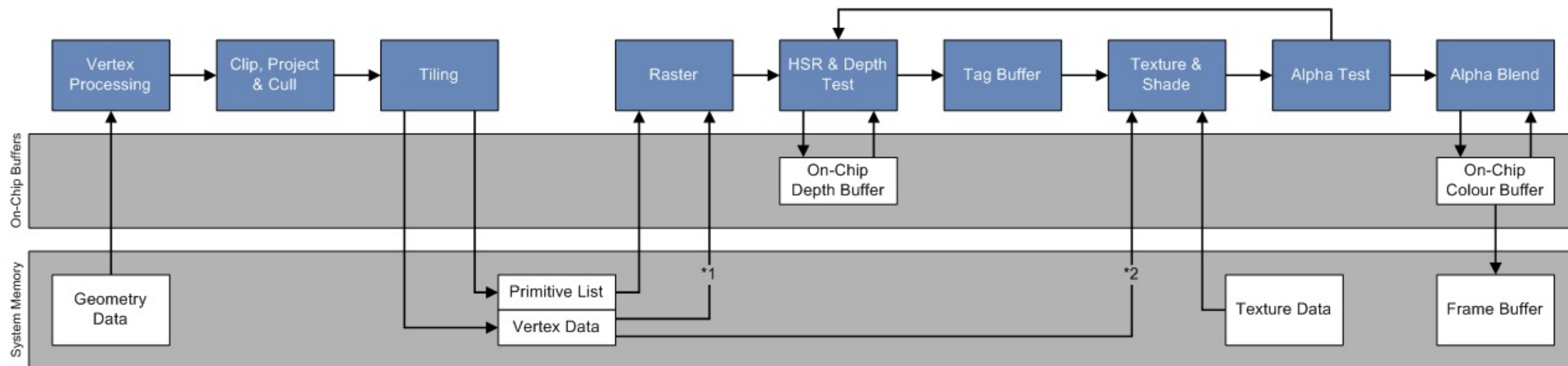
iPhones and iPads

Now, neural network accelerators,  
networking chips, and in the near future *hardware ray-tracing*...

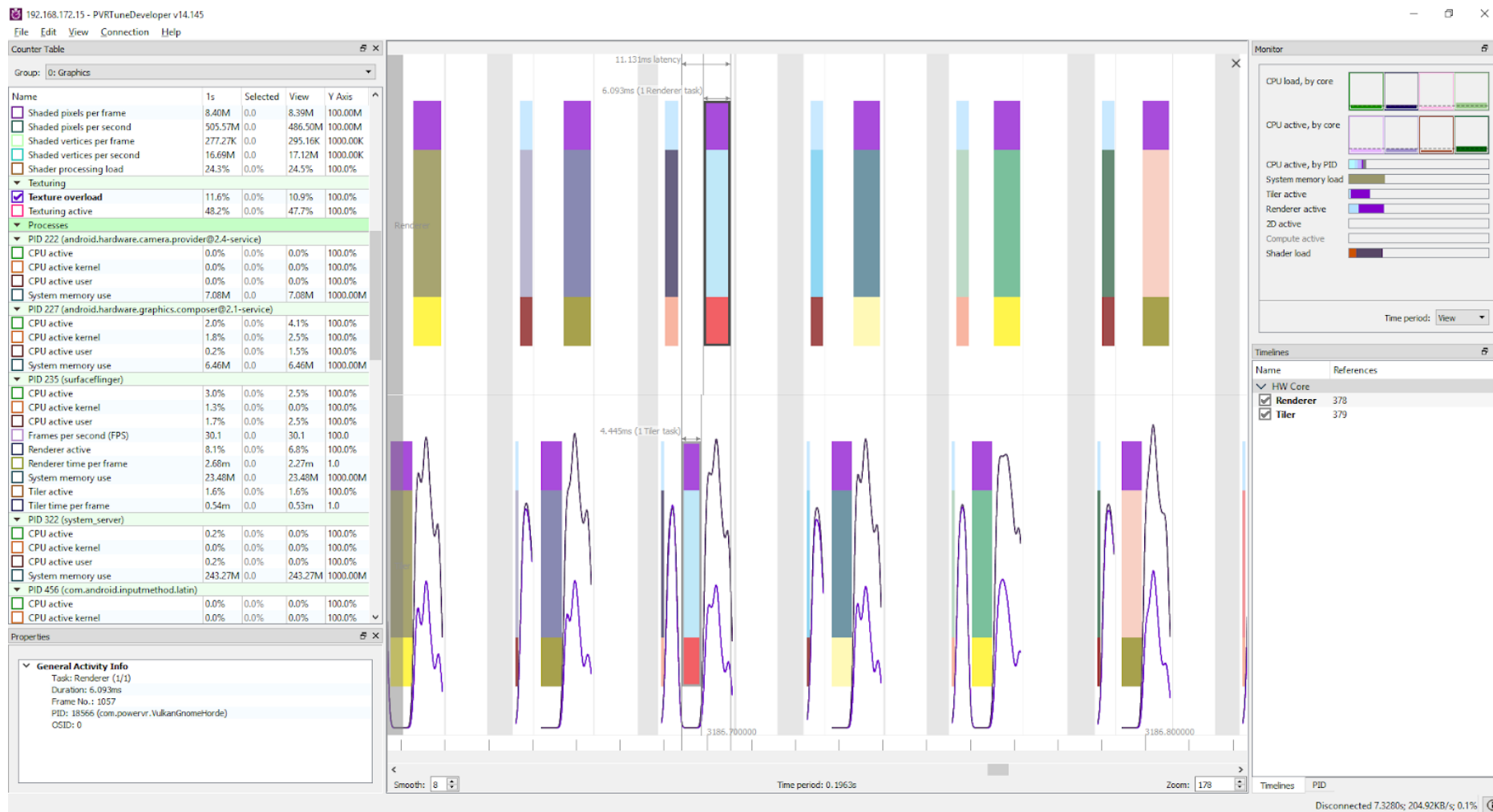




# PowerVR Tile based deferred rendering



# PVRTune



# PVRCarbon

D:/Documents/current/roblox/roblox-cb-170220.pvrcbn - PVRCarbon v0.5

File Edit View Help

Threads

ID	UID	Error	Result	Name	Arguments
7548	109206	0x70ef13168	glGetString	(name=GL_VERSION)	
7550	109207		glUseProgram	(program=0)	
7596	109208		glActiveTexture	(texture=GL_TEXTURE0)	
7609	109209		glBindTexture	(target=GL_TEXTURE_2D, texture=9)	
7706	109210		glPixelStorei	(pname=GL_UNPACK_ALIGNMENT, param=4)	
7714	109211		glTexSubImage2D	(target=GL_TEXTURE_2D, level=0, xoffset=32, yoffset=544, width=32, height=32, format=GL_	
	109212		glBindTexture	(target=GL_TEXTURE_2D, texture=0)	
	109213		glActiveTexture	(texture=GL_TEXTURE0)	
	109214		glBindTexture	(target=GL_TEXTURE_2D, texture=10)	
	109215		glPixelStorei	(pname=GL_UNPACK_ALIGNMENT, param=4)	
	109216		glTexSubImage2D	(target=GL_TEXTURE_2D, level=0, xoffset=32, yoffset=544, width=32, height=32, format=GL_	
	109217		glBindTexture	(target=GL_TEXTURE_2D, texture=0)	
	109218		glActiveTexture	(texture=GL_TEXTURE0)	
	109219		glBindTexture	(target=GL_TEXTURE_2D, texture=9)	
	109220		glPixelStorei	(pname=GL_UNPACK_ALIGNMENT, param=4)	
	109221		glTexSubImage2D	(target=GL_TEXTURE_2D, level=0, xoffset=160, yoffset=544, width=32, height=32, format=GL_	
	109222		glBindTexture	(target=GL_TEXTURE_2D, texture=0)	
	109223		glActiveTexture	(texture=GL_TEXTURE0)	
	109224		glBindTexture	(target=GL_TEXTURE_2D, texture=10)	
	109225		glPixelStorei	(pname=GL_UNPACK_ALIGNMENT, param=4)	
	109226		glTexSubImage2D	(target=GL_TEXTURE_2D, level=0, xoffset=160, yoffset=544, width=32, height=32, format=GL_	
	109227		glBindTexture	(target=GL_TEXTURE_2D, texture=0)	
	109228		glActiveTexture	(texture=GL_TEXTURE0)	
	109229		glBindTexture	(target=GL_TEXTURE_2D, texture=9)	
	109230		glPixelStorei	(pname=GL_UNPACK_ALIGNMENT, param=4)	
	109231		glTexSubImage2D	(target=GL_TEXTURE_2D, level=0, xoffset=288, yoffset=544, width=32, height=32, format=GL_	
	109232		glBindTexture	(target=GL_TEXTURE_2D, texture=0)	
	109233		glActiveTexture	(texture=GL_TEXTURE0)	
	109234		glBindTexture	(target=GL_TEXTURE_2D, texture=10)	
	109235		glPixelStorei	(pname=GL_UNPACK_ALIGNMENT, param=4)	
	109236		glTexSubImage2D	(target=GL_TEXTURE_2D, level=0, xoffset=288, yoffset=544, width=32, height=32, format=GL_	
	109237		glBindTexture	(target=GL_TEXTURE_2D, texture=0)	

Log

Opening File 'D:/Documents/current/roblox/roblox-cb-170220.pvrcbn'  
Connected to renderer  
File loading has finished (0 errors, 0 warnings)

Static Analysis

Level	Frame	UID	Call	Description
Warning	1	328	glDrawRangeElements	No appropriate framebuffer clears have been used between glBindFramebuffer and the first glDraw* command. This c
Warning	1	372	glDrawArrays	No appropriate framebuffer clears have been used between glBindFramebuffer and the first glDraw* command. This c
Warning	1	419	glDrawRangeElements	No appropriate framebuffer clears have been used between glBindFramebuffer and the first glDraw* command. This c

Render

111208: egISwapBuffers

1920 x 1080 px

Recorded Framebuffer

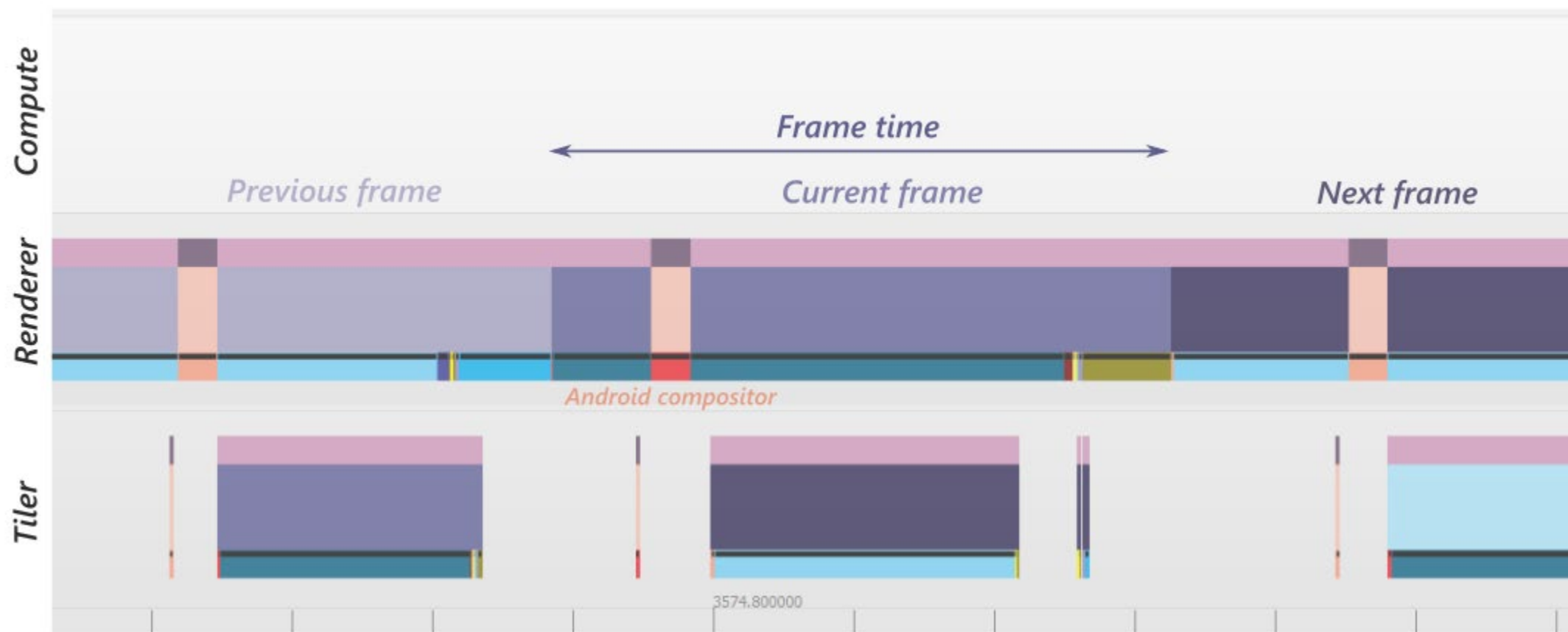
111208: egISwapBuffers

1920 x 1080 px

Current Frame: 80/100

Renderer: Host

# PVRTune timeline



Vulkan – use double/triple buffering please



# Analysing Roblox on PowerVR



# PVRCarbon analysis


File Edit View Help

Threads

ID	UID	Error	Result	Name	Arguments
20005	64071		VK_SUCCESS	vkWaitForFences	(device=0xc2660c00, fenceCount=1, pFences=0xbfb7be8, waitAll=VK_TRUE, timeout=184467440737)
20007	64072		VK_SUCCESS	vkResetFences	(device=0xc2660c00, fenceCount=1, pFences=0xbfb7be8)
20052	64073			vkDestroyBuffer	(device=0xc2660c00, buffer=0xc3098280, pAllocator=NULL)
20084	64074			vkDestroyBuffer	(device=0xc2660c00, buffer=0xb29de540, pAllocator=NULL)
20164	64075			vkDestroyBuffer	(device=0xc2660c00, buffer=0xb29de620, pAllocator=NULL)
20170	64076			vkDestroyBuffer	(device=0xc2660c00, buffer=0xb29de770, pAllocator=NULL)
	64077			vkDestroyBuffer	(device=0xc2660c00, buffer=0xb119d080, pAllocator=NULL)
	64078			vkDestroyBuffer	(device=0xc2660c00, buffer=0xb119cd00, pAllocator=NULL)
	64079			vkDestroyBuffer	(device=0xc2660c00, buffer=0xb29e4f20, pAllocator=NULL)
	64080			vkDestroyBuffer	(device=0xc2660c00, buffer=0xb29e4e40, pAllocator=NULL)
	64081			vkDestroyBuffer	(device=0xc2660c00, buffer=0xb29e4d60, pAllocator=NULL)
	64082			vkDestroyBuffer	(device=0xc2660c00, buffer=0xb29e4c80, pAllocator=NULL)

Render

64723: vkQueuePresentKHR



97371 vkCmdCopyBufferToImage (commandBuffer=0xc2e01f80, srcBuffer=0xb92870d0, dstImage=0xc0b10140, dstImageLayout=VK\_IMAGE\_LAYOUT\_TRANSFER\_DST\_OPTIMAL, regionCount=1, pRegi

97372 vkCmdPipelineBarrier (commandBuffer=0xc2e01f80, srcStageMask=VK\_PIPELINE\_STAGE\_TRANSFER\_BIT, dstStageMask=VK\_PIPELINE\_STAGE\_ALL\_COMMANDS\_BIT, dependencyFlags=0x000

Log

Opening File 'D:/Documents/current/roblox/pvr碳/dasco597\_csm\_soft0.pvr碳'

Connected to renderer

File loading has finished (0 errors, 0 warnings)

File closed

Opening File 'D:/Documents/current/roblox/pvr碳/com.roblox.client\_2019\_10\_18\_11\_33\_18.pvr碳'

Disconnected from renderer

File loading has finished (0 errors, 0 warnings)

Connected to renderer

File closed

Opening File 'D:/Documents/current/roblox/pvr碳/pvr碳/dasco597\_csm\_soft0.pvr碳'

Disconnected from renderer

Connected to renderer

Warning: Substituting recorded surface extension 'VK\_KHR\_android\_surface' with 'VK\_KHR\_win32\_surface'

Warning: Substituting recorded surface extension 'VK\_KHR\_android\_surface' with 'VK\_KHR\_win32\_surface'

File loading has finished (0 errors, 0 warnings)

Level	Frame	UID	Call	Description
Warning	10	328	glDrawRangeElements	No appropriate framebuffer clears have been used between glBindFramebuffer and the first glDraw* command. Thi
Warning	10	372	glDrawArrays	No appropriate framebuffer clears have been used between glBindFramebuffer and the first glDraw* command. Thi
Warning	10	419	glDrawRangeElements	No appropriate framebuffer clears have been used between glBindFramebuffer and the first glDraw* command. Thi

Static Analysis

Filter:

Level Frame UID Call Description

64088 vkDestroyBuffer (device=0xc2660c00, buffer=0xb29e1100, pAllocator=NULL)

64089 vkDestroyImageView (device=0xc2660c00, imageView=0xb943f300, pAllocator=NULL)

64090 vkDestroyImage (device=0xc2660c00, image=0xb4b0f440, pAllocator=NULL)

64091 vkDestroyImageView (device=0xc2660c00, imageView=0xb5167a00, pAllocator=NULL)

64092 vkDestroyImage (device=0xc2660c00, image=0xb4b0eccc, pAllocator=NULL)

64093 vkDestroyBuffer (device=0xc2660c00, buffer=0xb119e3c0, pAllocator=NULL)

64094 VK\_SUCCESS vkResetCommandPool (device=0xc2660c00, commandPool=0xc316d400, flags=0x00000000)

64095 VK\_SUCCESS vkGetPhysicalDeviceSurfaceC (physicalDevice=0xc2e96260, surface=0xc33a7550, pSurfaceCapabilities=0xbfb8004)

64096 VK\_SUCCESS vkBeginCommandBuffer (commandBuffer=0xb91f30c0, pBeginInfo=0xbfb7bf8)

64097 VK\_SUCCESS vkCreateBuffer (device=0xc2660c00, pCreateInfo=0xbfb7908, pAllocator=NULL, pBuffer=0xb1160c68)

64099 vkGetBufferMemoryRequirem (device=0xc2660c00, buffer=0xb119e3c0, pMemoryRequirements=0xbfb78f0)

64100 VK\_SUCCESS vkBindBufferMemory (device=0xc2660c00, buffer=0xb119e3c0, memory=0xc0c217c0, memoryOffset=1105920)

64101 VK\_SUCCESS vkCreateBuffer (device=0xc2660c00, pCreateInfo=0xbfb7908, pAllocator=NULL, pBuffer=0xb1160b28)

64103 vkGetBufferMemoryRequirem (device=0xc2660c00, buffer=0xb95e61d0, pMemoryRequirements=0xbfb78f0)

64104 VK\_SUCCESS vkBindBufferMemory (device=0xc2660c00, buffer=0xb95e61d0, memory=0xc302f740, memoryOffset=69632)

Static Analysis

Filter:

Level Frame UID Call Description

Warning 10 328 glDrawRangeElements No appropriate framebuffer clears have been used between glBindFramebuffer and the first glDraw\* command. Thi

Warning 10 372 glDrawArrays No appropriate framebuffer clears have been used between glBindFramebuffer and the first glDraw\* command. Thi

Warning 10 419 glDrawRangeElements No appropriate framebuffer clears have been used between glBindFramebuffer and the first glDraw\* command. Thi

Static Analysis

Current Call Find Results

Current Frame: 137/248

Render

64722: vkQueueSubmit

1920 x 1080 px

Recorded Framebuffer

64722: vkQueueSubmit


1920 x 1080 px

0.073, 0.53

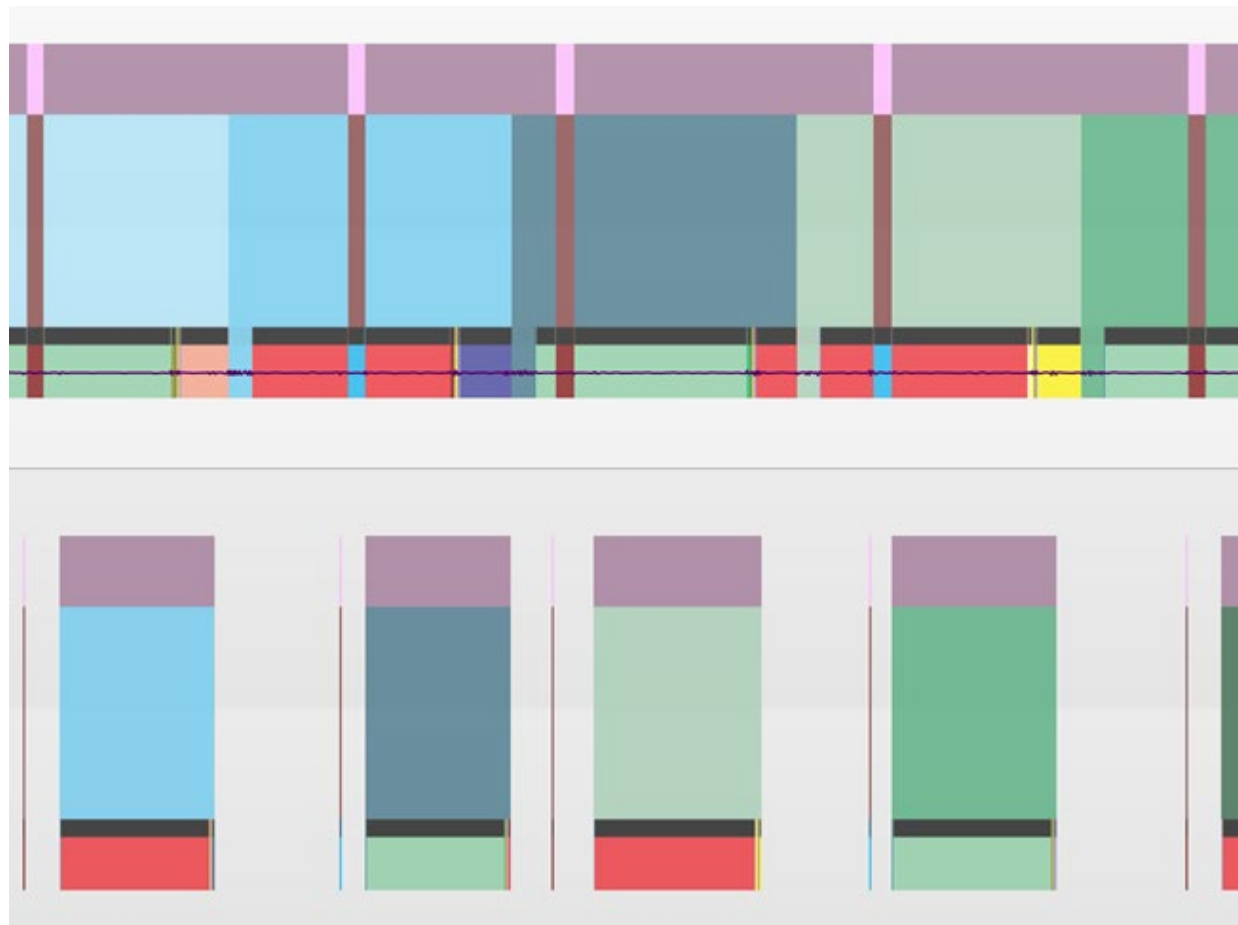
RGBA(30, 19, 13, 255)

141, 573 px

Renderer: Host



# New tune recording



Glorious task packing!

# GLSL optimisations

`vec_1 = (float_1 * vec_2) * float_2;`       $\longrightarrow$       `vec_1 = (float_1 * float_2) * vec_2;`  
 Group scalar arithmetic

`vec_1 = sqrt(vec_2);`       $\longrightarrow$       `vec_1 = vec_2 * inversesqrt(vec_2);`

`float_0 = max(float_1, 0.0);`       $\longrightarrow$       `float_0 = clamp(float_1, 0.0, 1.0);`  
 abs() neg() and clamp(..., 0.0, 1.0) can be free

`vec_0 = vec_1 * vec_2 + vec_3;`       $\longleftrightarrow$       `vec_0 = clamp(vec_1 * vec_2 + vec_3, 0.0, 1.0);`

PowerVR developer documents: <https://docs.imgtec.com/>



# GLSL optimisations - utilisation

Too many registers being used

-> less threads can be processed in a cluster at a time

-> *reduced utilisation*

Mediump allows PowerVR to use 16 bit floats

-> reduce register pressure, increase occupancy (up to 100%)

# GLSL optimisations

PBR shader:

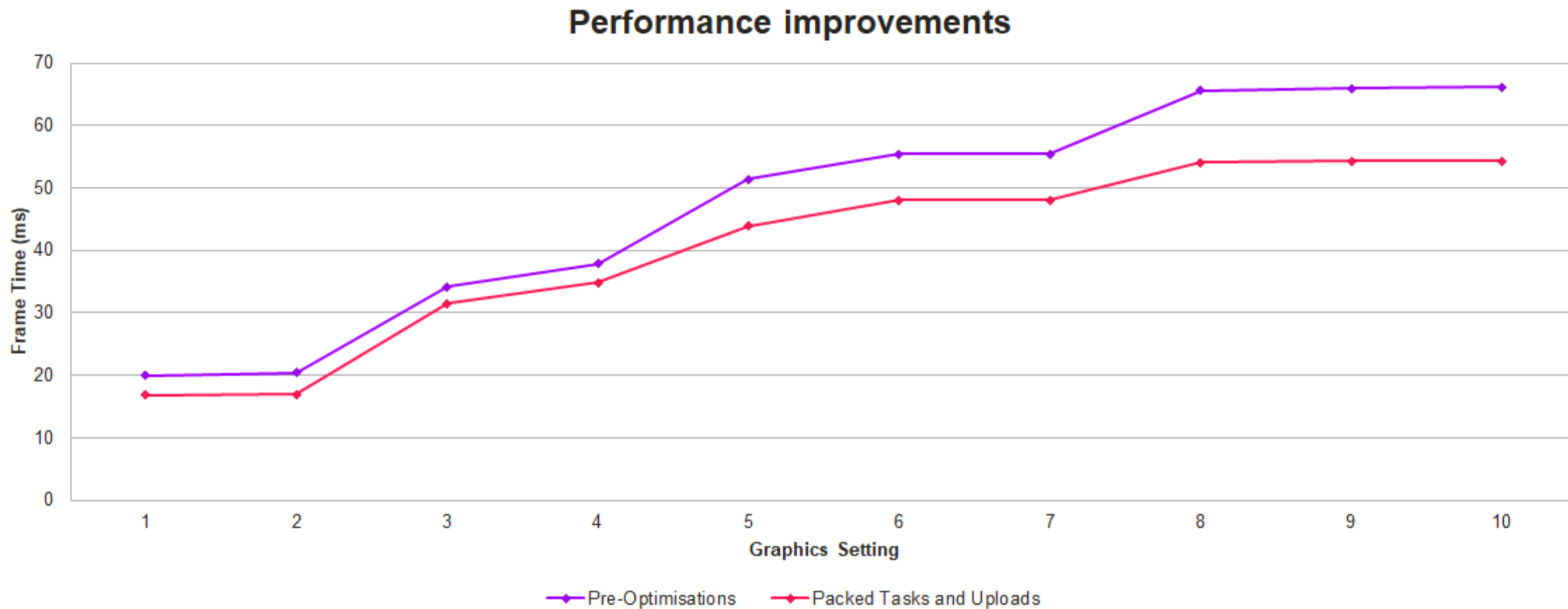
- A-Series: cycle count decreased by 9%
- Rogue (Oppo Reno): cycle count decreased by 12% and utilisation improved by 33%

PBR+IBL shader:

- A-Series: cycle count decreased by 9%
- Rogue (Oppo Reno): cycle count decreased by 20% and utilisation improved by 36%

Use PVRShaderEditor to analyse cycle count and disassembly

# Analysing Roblox on PowerVR



# Lighting Shadows Roblox

and  
in



# Lighting system: phase 2, shadow maps

- Voxel shadows are too coarse
- Shadow maps to the rescue!
- Excellent quality
- Challenge: “soft” shadows
- Challenge: high cost to re-render
- Challenge: many shadow casting lights
- Only released sun shadows to simplify



# Lighting system: rendering shadow maps

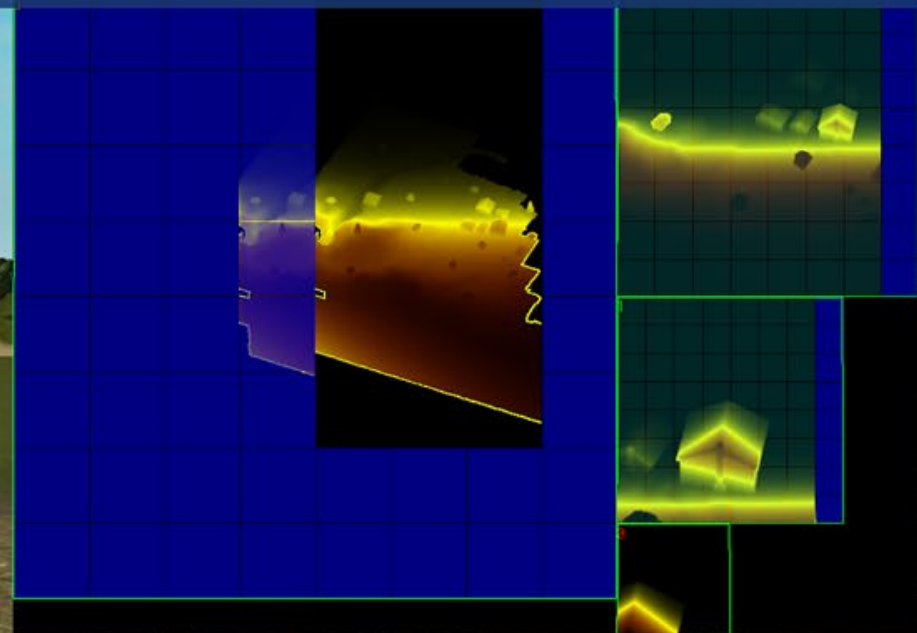
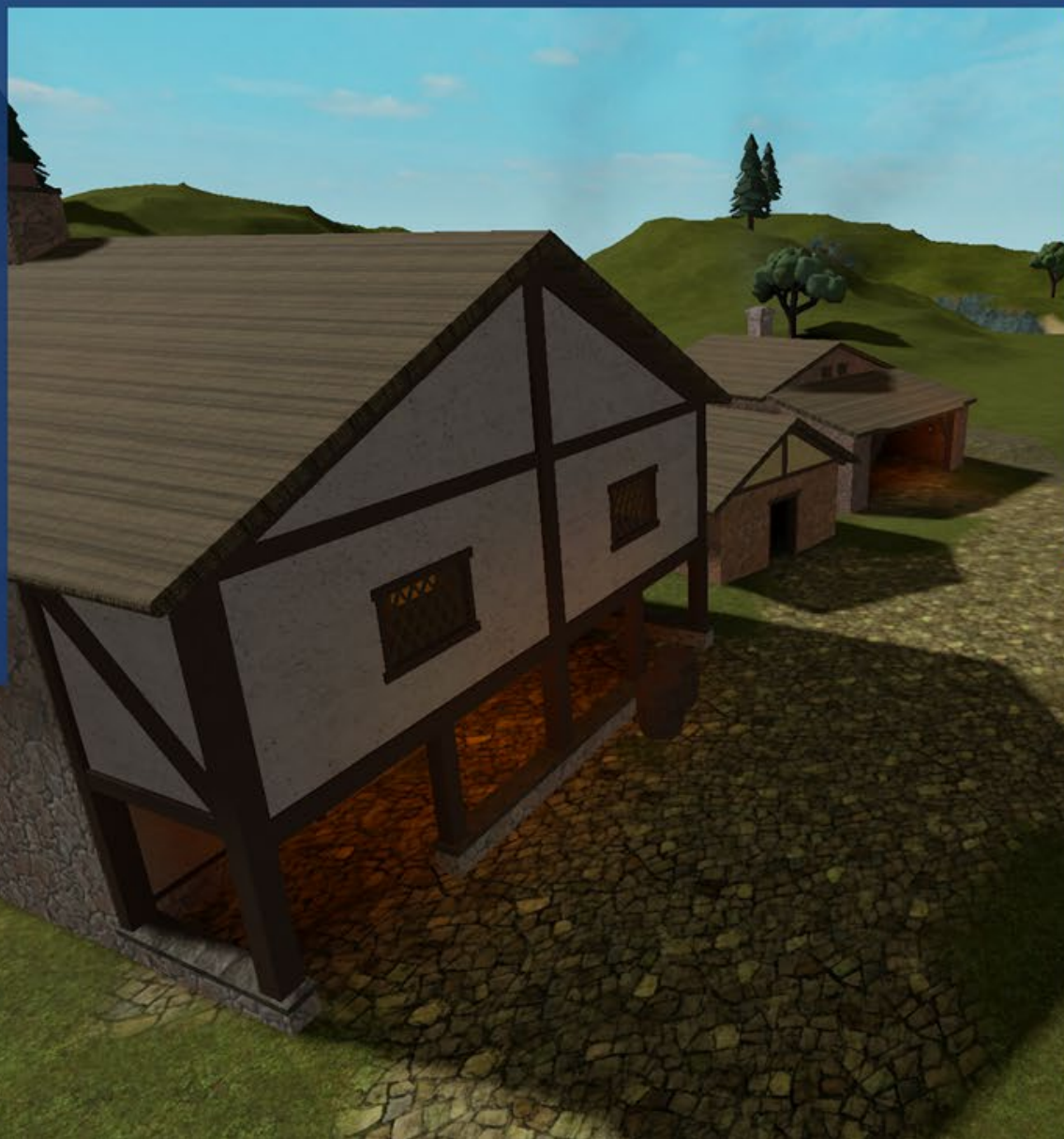
- Cascaded tiled shadow maps
  - 1-4 cascades depending on the quality levels
  - Far cascades are split into tiles to be able to throttle shadow updates
- CSM Scrolling (Insomniac @ SIGGRAPH 2012)
  - CSM Scrolling: An acceleration technique for the rendering of cascaded shadow maps
- When we need to update a cascade, it's done in single pass
  - If multiple tiles are marked as dirty, we re-render geometry in these tiles
  - CPU does per-tile frustum culling
- Light direction changes invalidate caching :(
  - Haven't found a good solution to this



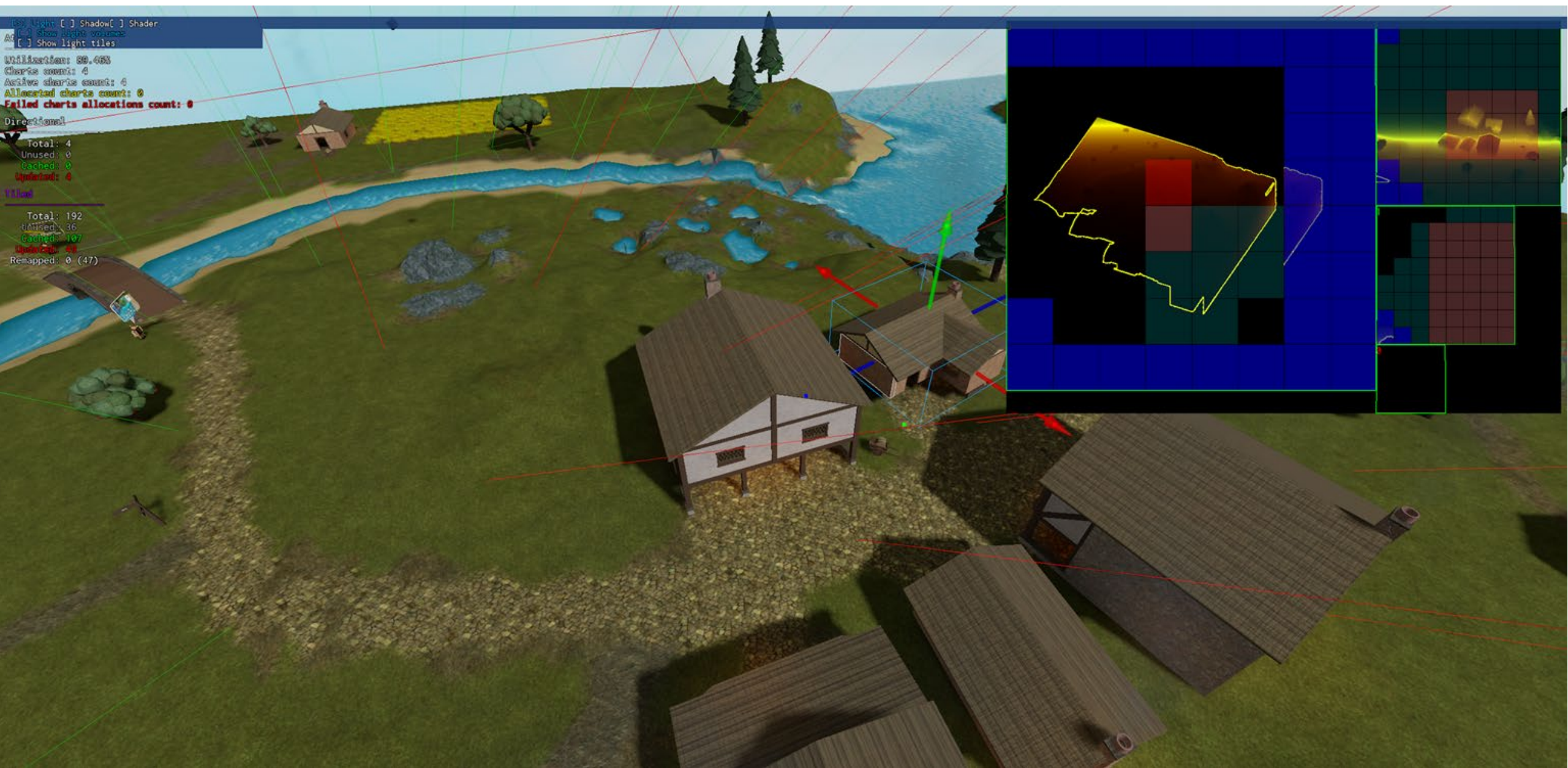
```

[ ] Light [x] Shader
Atlas [x] Freeze debug render
[x] Render stats
[x] Invalidate each frame
[x] Throttle each frame
-----[ Shadow Atlas ]-----
Active char [x] Show atlas
Allocated char [x] Dump bitmask as image
Failed char [x] Dump atlas as image
[x] Show atlas charts lifetime
[x] Show atlas updates
[x] Show unused sub frustums
[x] Show pancaked sub frustums
-----[ Shadow Casters ]-----
[x] Show shadow casters (bound boxes)
[x] Precise shadow casters culling (sun)
-----[ Shadow Volumes ]-----
[x] Cascade #0
[x] Cascade #1
[x] Cascade #2 1 by remapping (4:12)
[x] Cascade #3
[x] Point lights
[x] Spot lights
[x] Global influence
-----[ Shadow Frustums ]-----
[x] Cascade #0
[x] Cascade #1
[x] Cascade #2
[x] Cascade #3
[x] Tiled frustums
[x] Point lights
[x] Spot lights
-----[ Features and Optimizations ]-----
[x] EVSM blur
[x] Precise shadow casters culling (sun)
[x] Casters per cascade filtering (sun)
[x] Pancaking
[x] CSM Scroll
-----[ Debug Options ]-----
[x] Render depth as a checkerboard
[x] Force clear atlas before update
[x] CSM Scroll only mode
-----[ Debug Shader Options ]-----
[x] Show colored sun cascades
[x] Show sun shadowmap texel density
[x] Switch to simple shadows (ESM)

```



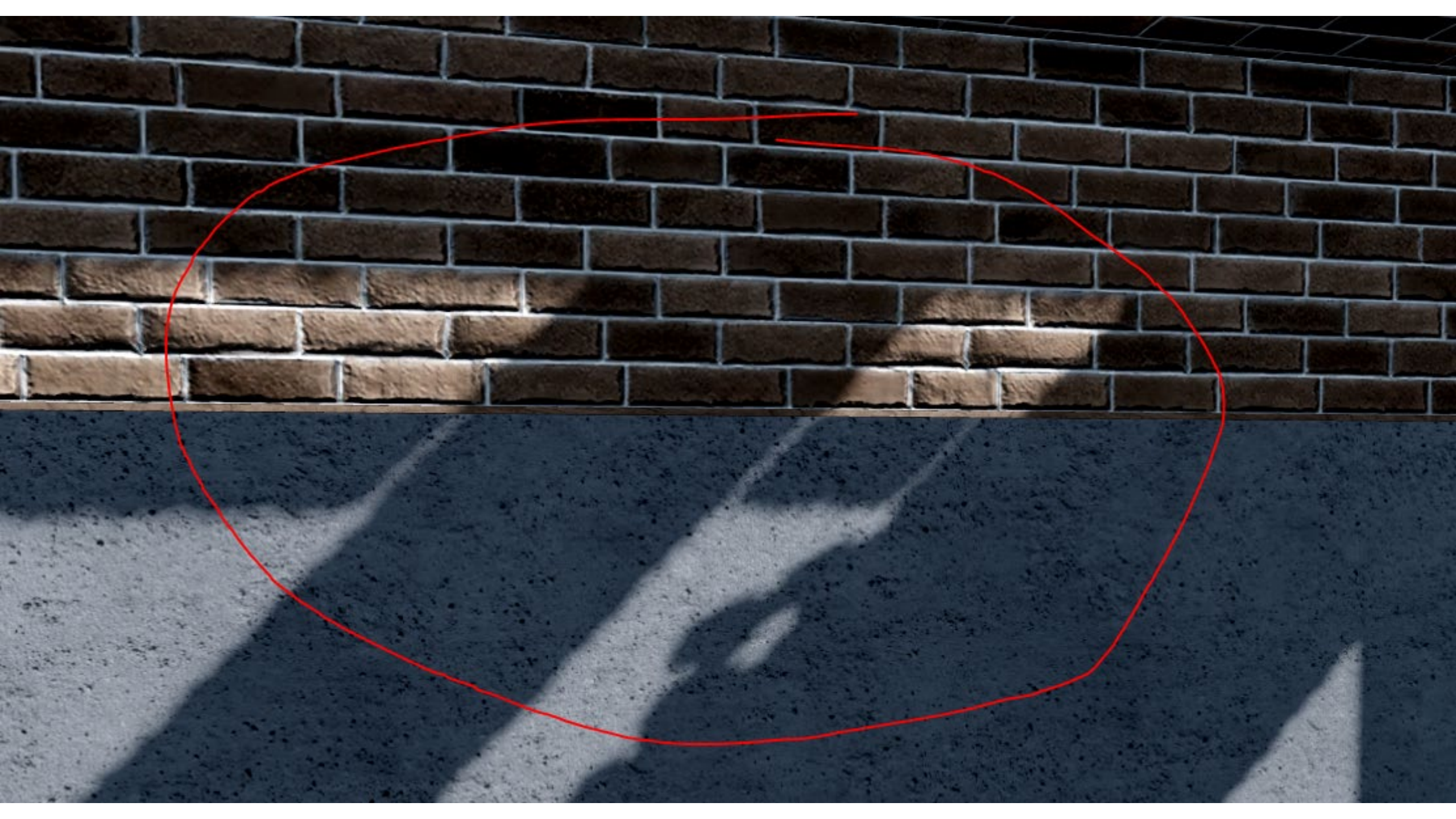




# Lighting system: shadow softness

- Want a wide range of shadow map “softness”
  - Wide PCF kernel expensive at high resolutions
  - Screen-space filtering impractical due to transparent geometry and tiler performance
- EVSM to the rescue!
  - Fully decouples shadow rendering from sampling
  - Light leaks...





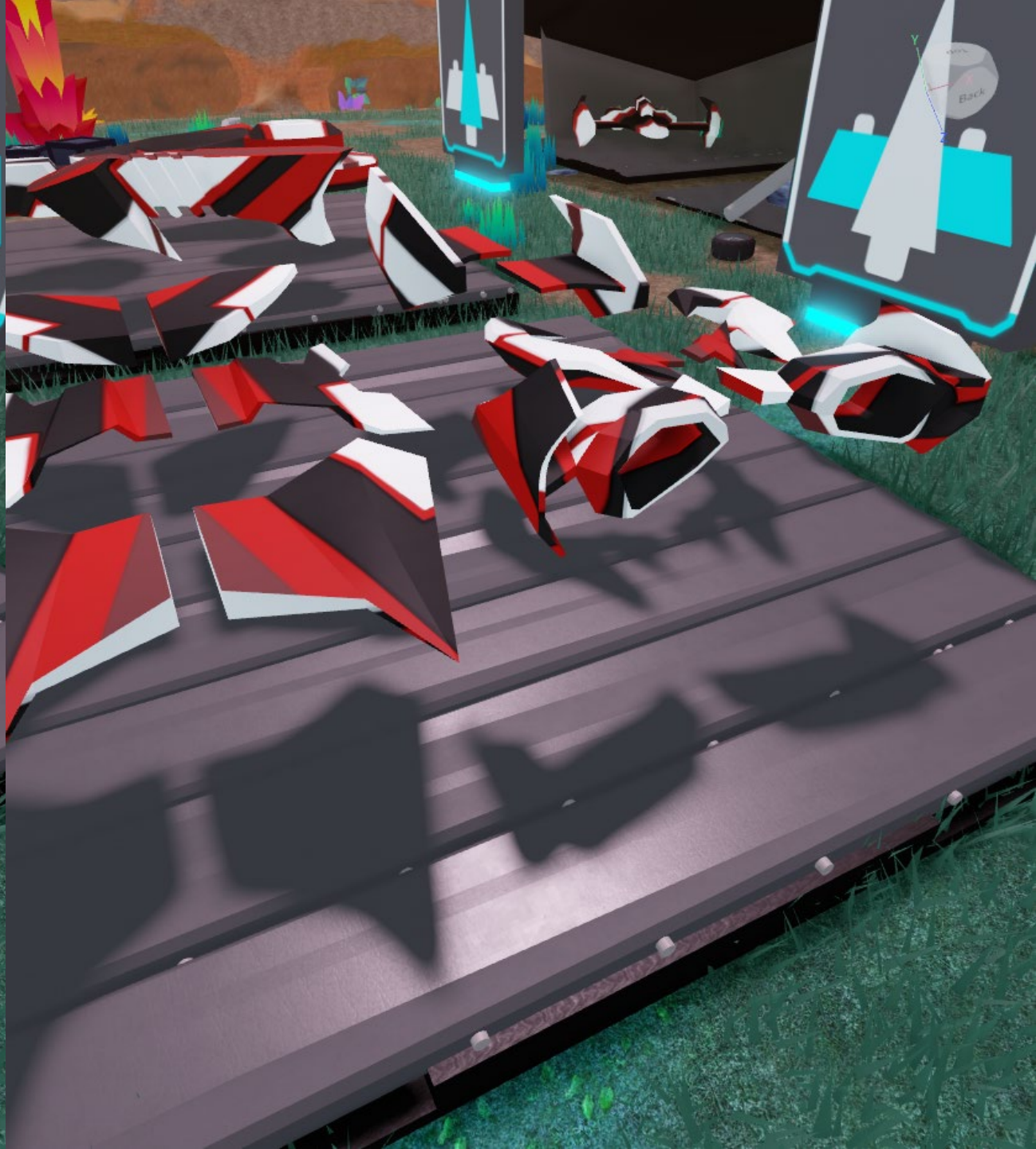
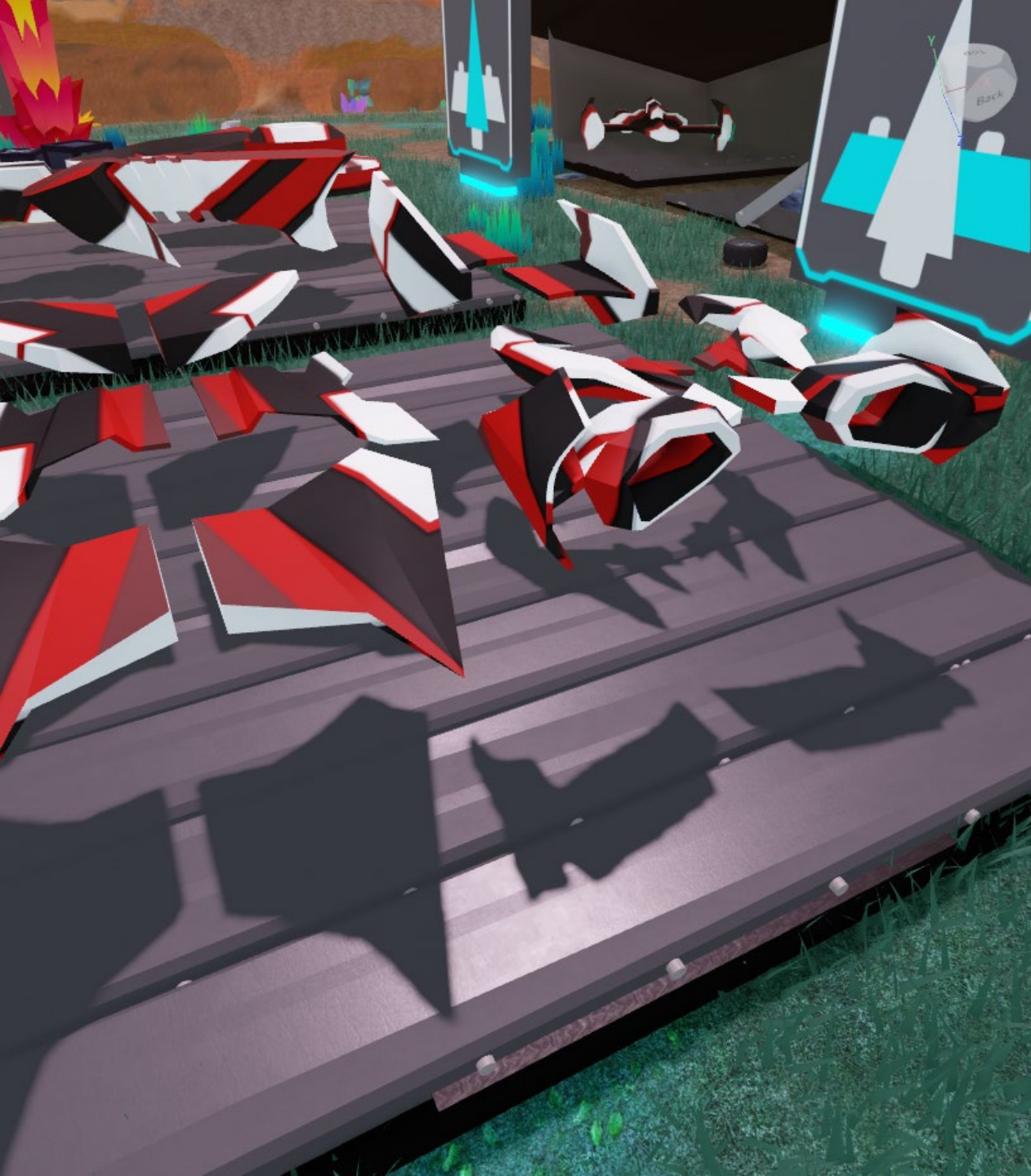
# Lighting system: making EVSM practical

- Light leaks
  - Reduce Z range!
    - Tightly fit shadow frustum to receiving geometry
    - “Pancaking” when rendering caster geometry
    - `VkPipelineRasterizationStateCreateInfo::depthClampEnable`
  - Over-darkening when sampling EVSM
- Performance
  - Use two moments (RG16F) on mobile, four (RGBA16F) on desktop
  - Render depth only, reconstruct exponentially warped moments during first blur pass
- Integrating with tiling...
  - Blurring is per tile to make it fully incremental, we cache blurred EVSM contents
  - CSM scrolling might require adjusting the cascade depth
    - $\exp(z + \text{offset}) = \exp(z) * \exp(\text{offset})$

# Lighting system: blurring shadow maps

- Standard two-pass blur (horizontal & vertical)
- First pass reads directly from depth texture (D16/D24)
  - Need to reconstruct EVSM moments from depth
  - Can't use bilinear filter optimizations :(
- Second pass reads from EVSM texture
  - Fewer taps!
  - See “Efficient Gaussian blur with linear sampling”
- Filter width varies from 3x3 to 17x17
  - Depends on developer-configured parameter, “ShadowSoftness”





# Lighting system: shadow MSAA

- Wide blurs are pretty expensive
  - Render depth at full resolution, two pass blur
  - Three full stores of tile contents to memory
- Since shadow moments are filterable, we can use MSAA!
  - Use 4x MSAA to reduce shadow resolution by 2x2
  - Fragment shader directly outputs shadow moments (render pass isn't depth only anymore)
  - Subsequent filters are smaller (2x) and can use bilinear optimizations
- On-chip resolve is crucial
  - \*Don't\* use `vkCmdResolveImage`!
  - Instead, use `pResolveAttachments` in render pass
- MSAA becomes an almost free way to increase shadow quality



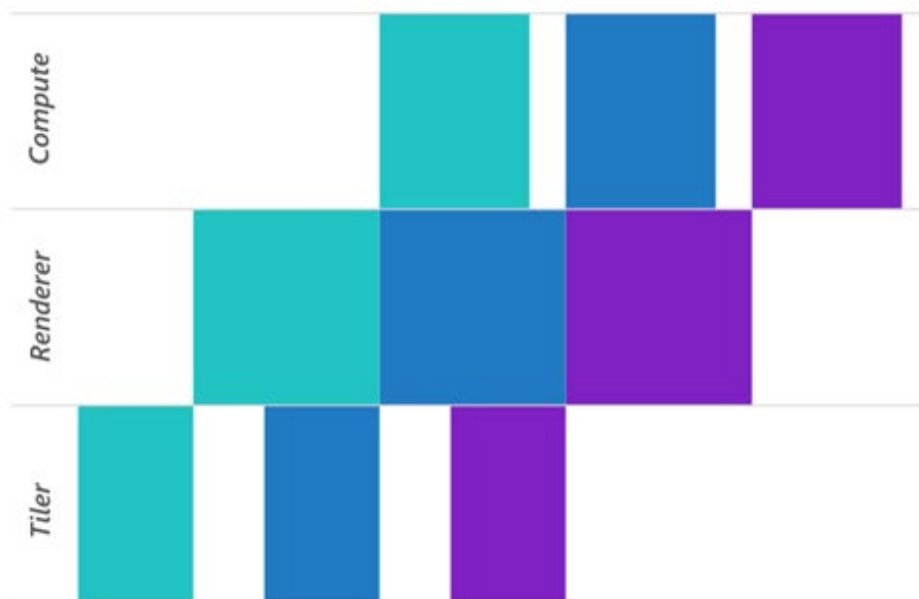
# Lighting system: sampling shadow maps

- Find a single cascade per pixel
  - Some ALU to do point-sphere tests per cascade
  - Because of throttling & quality concerns can't use pixel depth
- Sample EVSM texture
  - Requires adjusting UV coordinates
  - All shadow maps are stored in a single shadow atlas
- Compute shadow factor
  - Exponential depth warping
  - 2 or 4 moment VSM
  - Apply over-darkening to reduce leaks

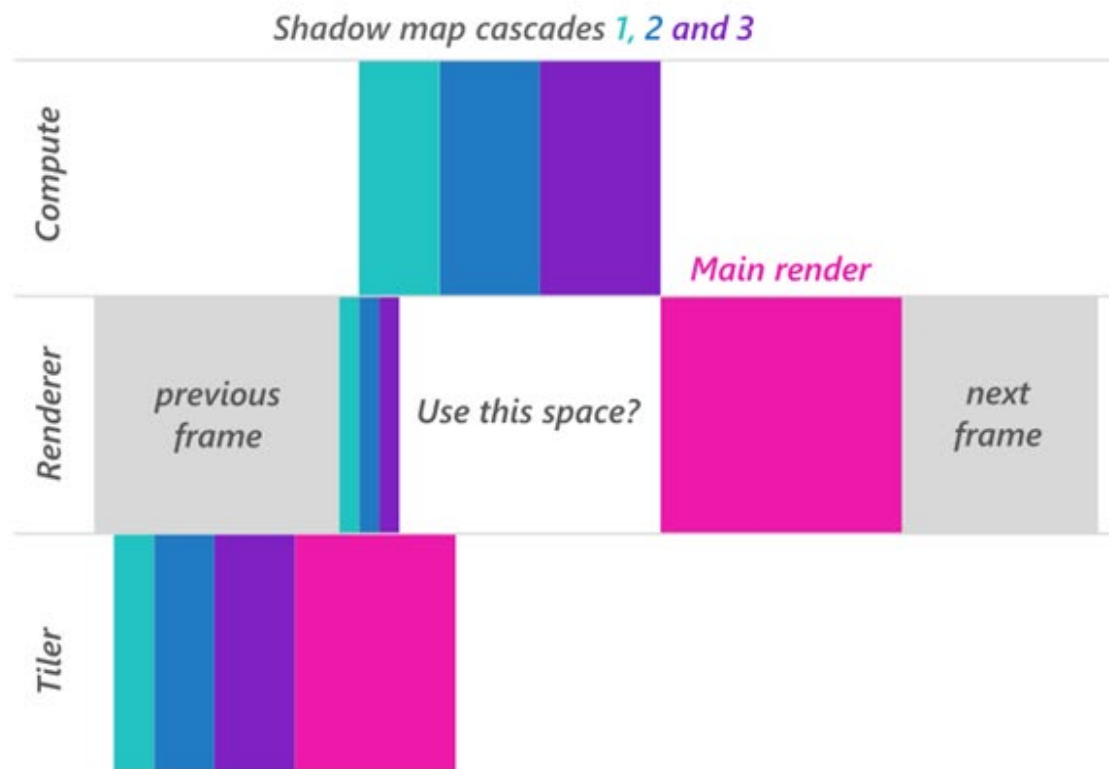
# Compute Optimisations for PowerVR

# Move gaussian blur to compute?

Compute post-processing allows more scheduling flexibility

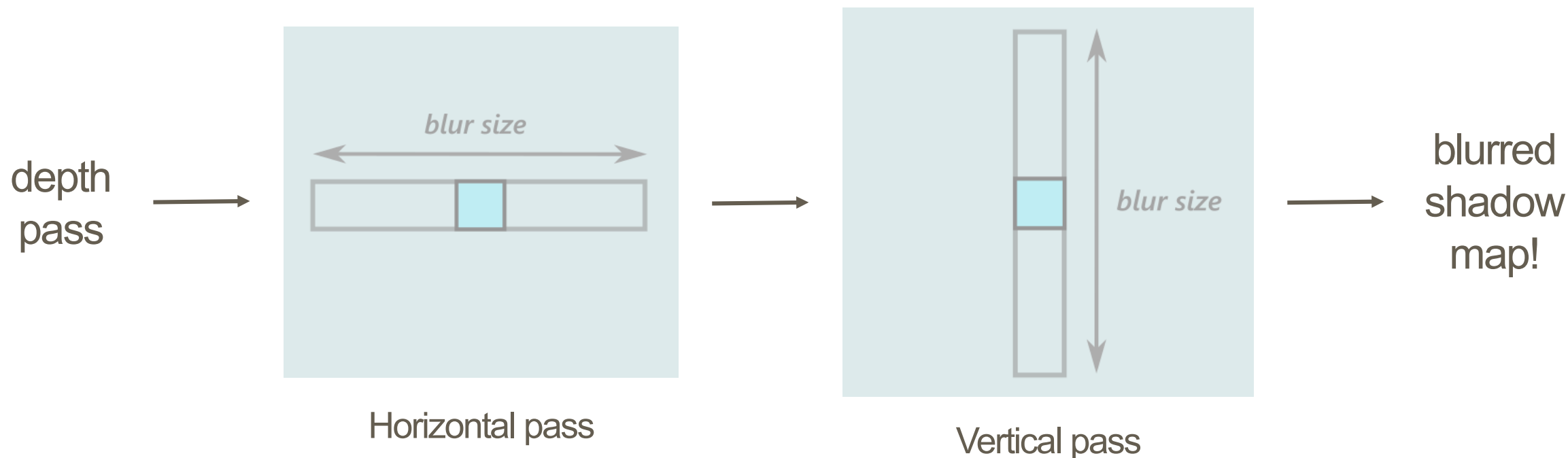


What about work in the middle of the frame?



# Separable-kernel gaussian blur

- Perform 2D gaussian blur with two 1D passes
- Mathematically equivalent (rank 1 matrix)
- $2n$  texture fetches instead of  $n^2$

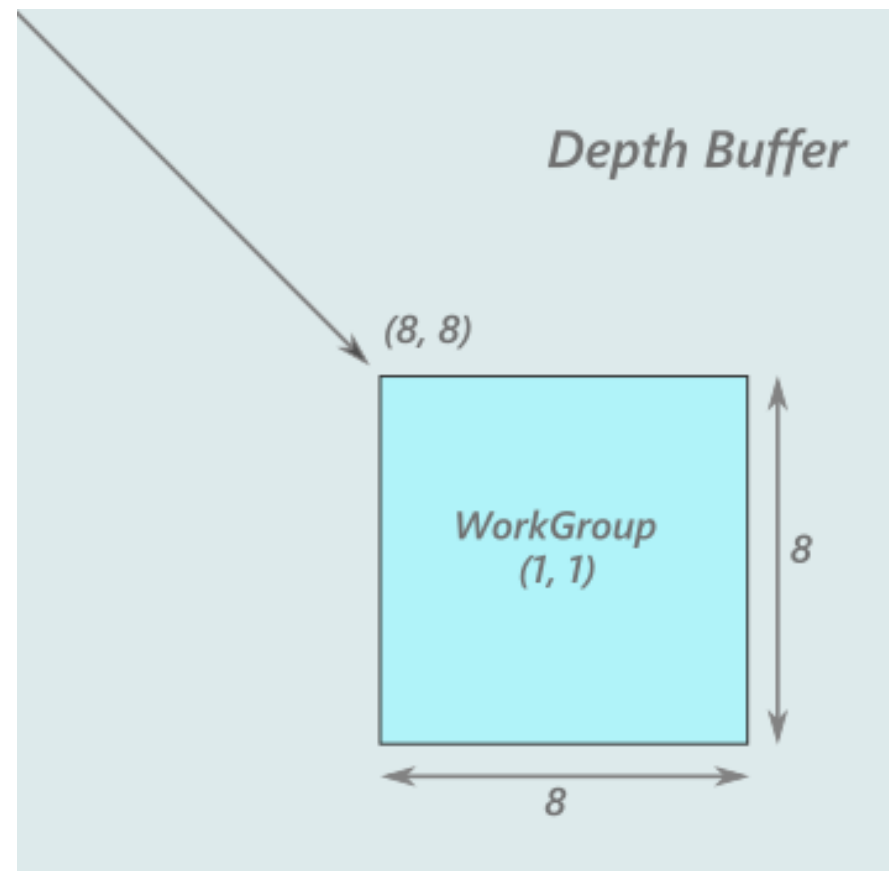


# Compute gaussian blur gather algorithm

Optimal work-group size on PowerVR is 32

8x4 work group size processing an 8x8 area

Experiment with looping shader multiple times (2 texels per thread here)

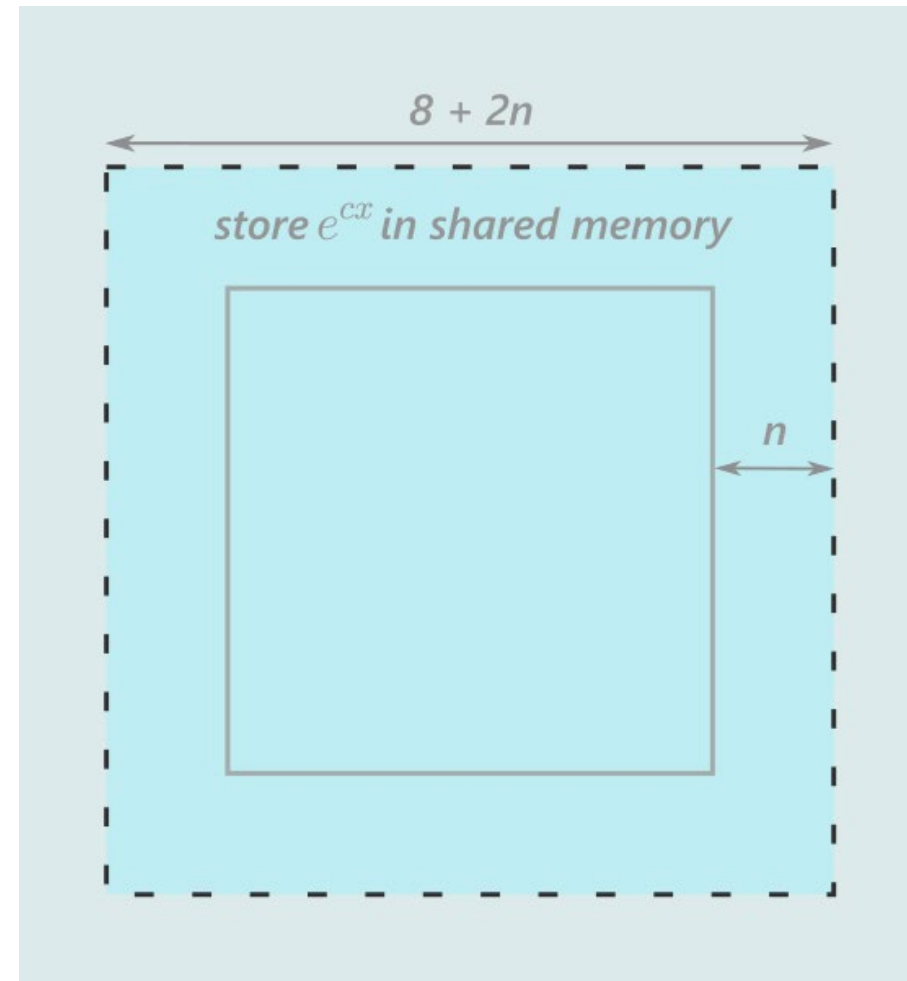
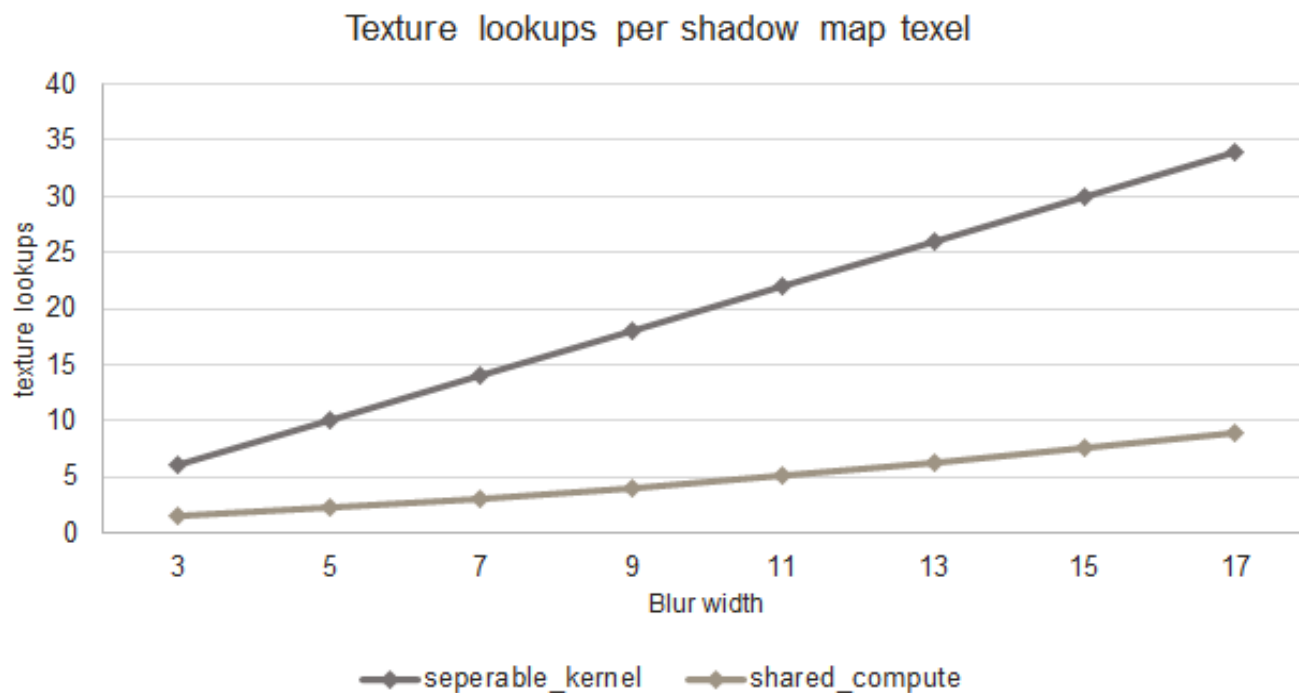




# Compute gaussian blur gather algorithm

Read depth values including surrounding area into shared memory

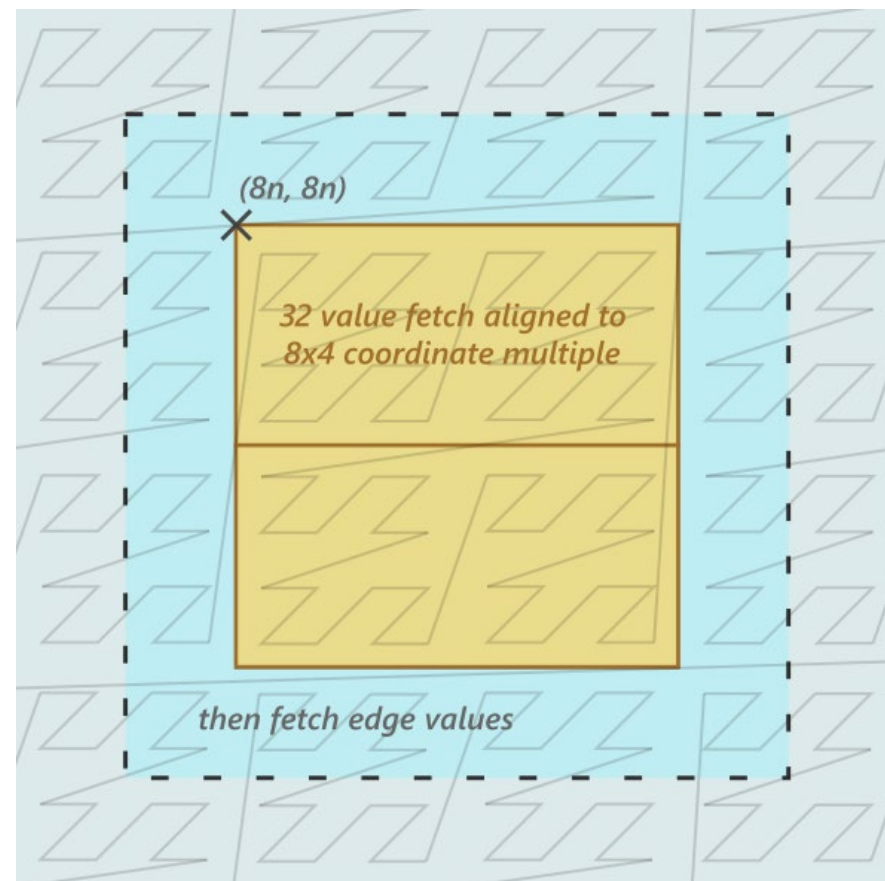
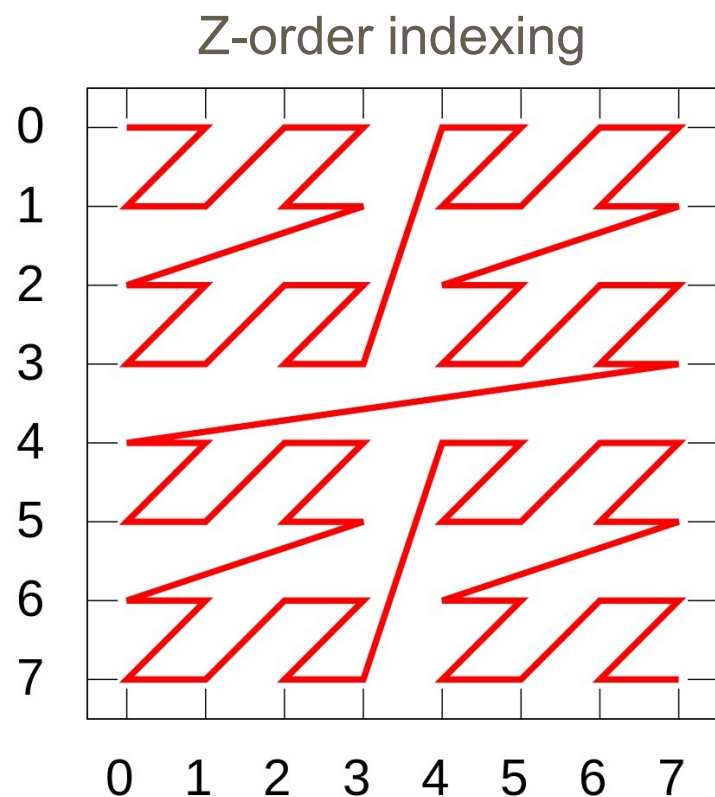
Reduces the number of texture fetches



# Morton ordering

Images created with VK\_IMAGE\_TILING\_OPTIMAL addressed with Morton ordering

Improve cache efficiency by loading an aligned area of texels (512 bit cache line)



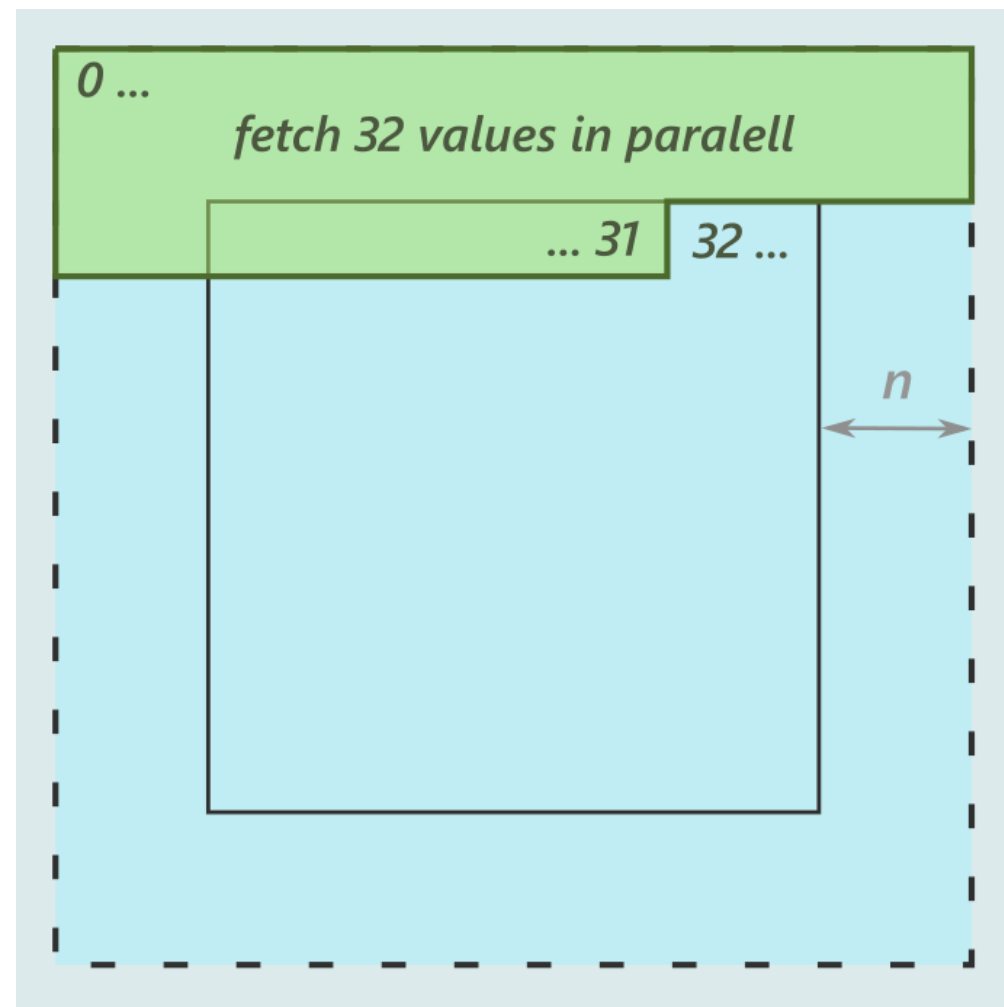
# Linear indexing

	Bank 1				Bank 2				Bank 3				Bank 4			
dword	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0																
16																
32																
...																
4080																

Shared memory:

- Read up to 16 adjacent floats 1 cycle
- Read multiple rows in a bank in  $n$  cycles
- Read = 1 cycle, write = 4 cycles

```
// our 32 threads write to adjacent memory simultaneously
float warped_depth = warp(texelFetch(depth_buffer,
    gl_LocalInvocationID.xy + offset).x);
shared_storage[gl_LocalInvocationIndex + n*32].x =
    warped_depth;
```



# Horizontal pass

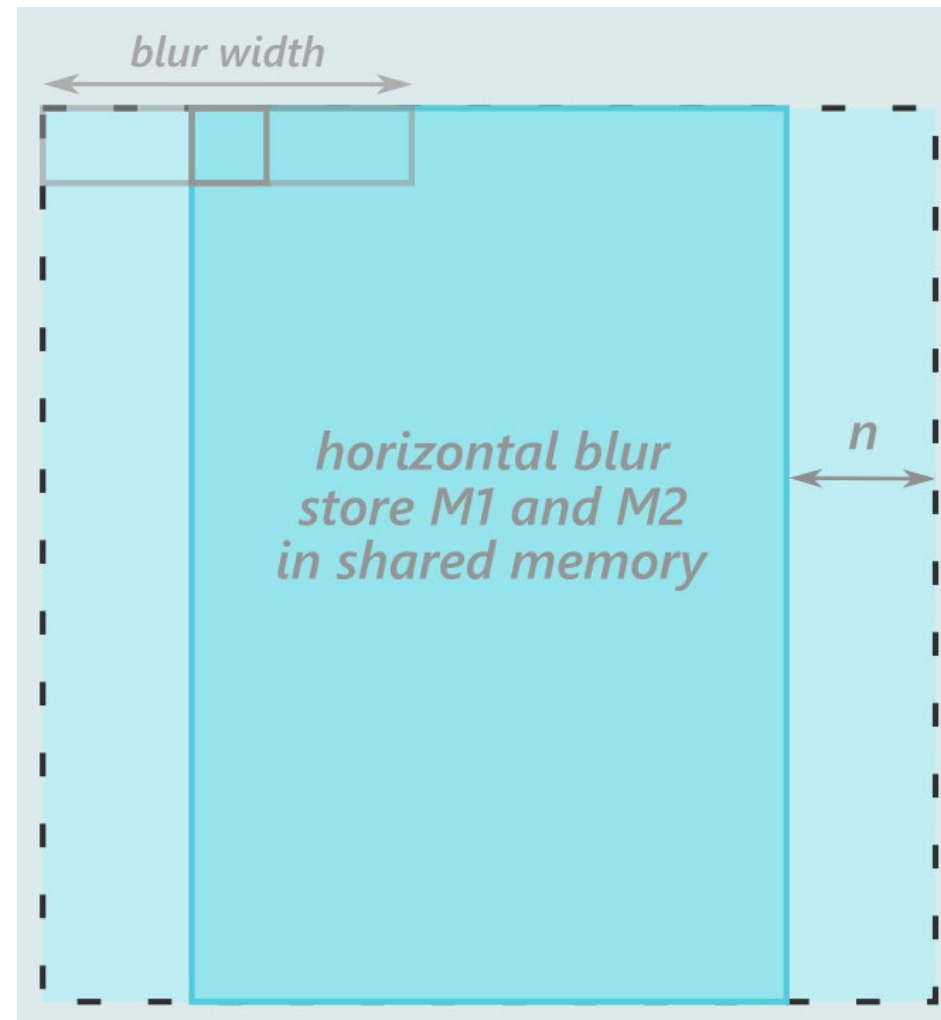
Write moments M1 and M2 to shared memory

$$M_1 = E(x) = \int_{-\infty}^{\infty} xp(x)dx$$

$$M_2 = E(x^2) = \int_{-\infty}^{\infty} x^2p(x)dx$$

VSM algorithm:

[http://www.punkuser.net/vsm/vsm\\_paper.pdf](http://www.punkuser.net/vsm/vsm_paper.pdf)



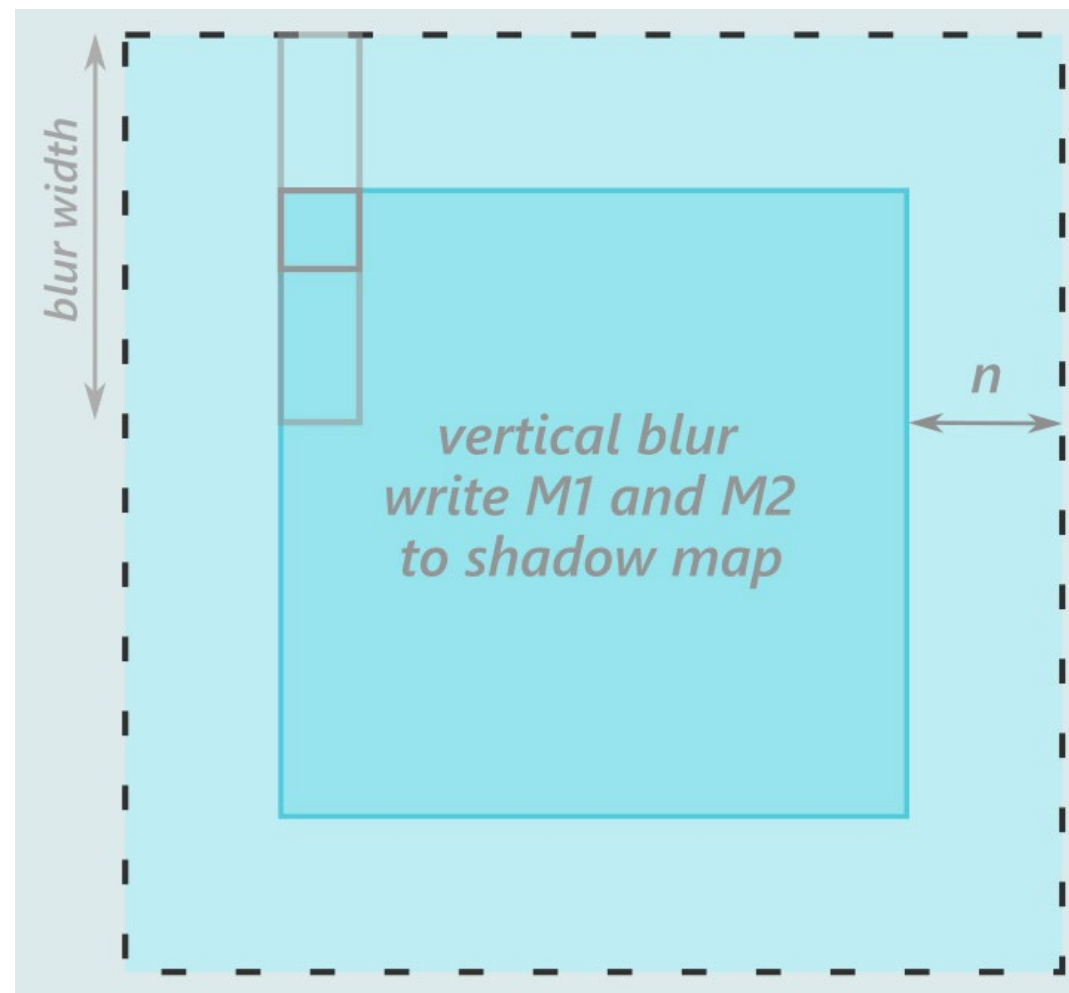
# Vertical pass

Write moments M1 and M2 to the shadow map

```
vec2 M1M2 = vec2(0.0);
memoryBarrierShared();

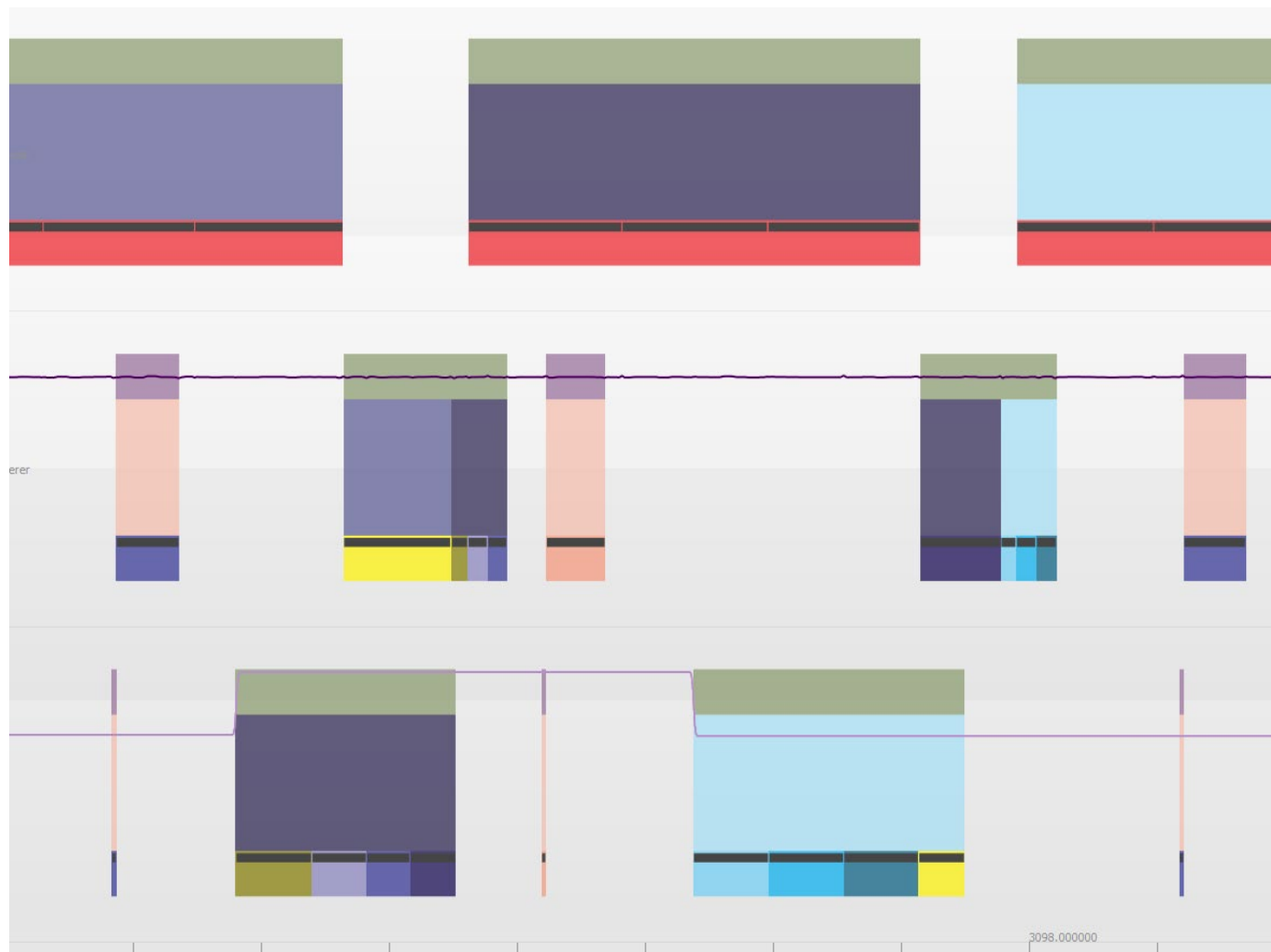
for (uint y = 0u; y < blur_width; y++) {
    uint shared_index = get_shared_index(y,
        gl_LocalInvocationID);
    vec2 M1M2_val = shared_storage[shared_index];
    M1M2 += M1M2_val * get_gaussian_factor(y);
}

imageStore(shadow_map, global_coord, vec4(M1M2,
    0.0f, 0.0f));
```





# Task packing



Vulkan spec section 6.2:

*“The initial order is determined by the order in which `vkQueueSubmit` commands are executed on the host, for a **single queue**, from first to last.”*

# Task packing

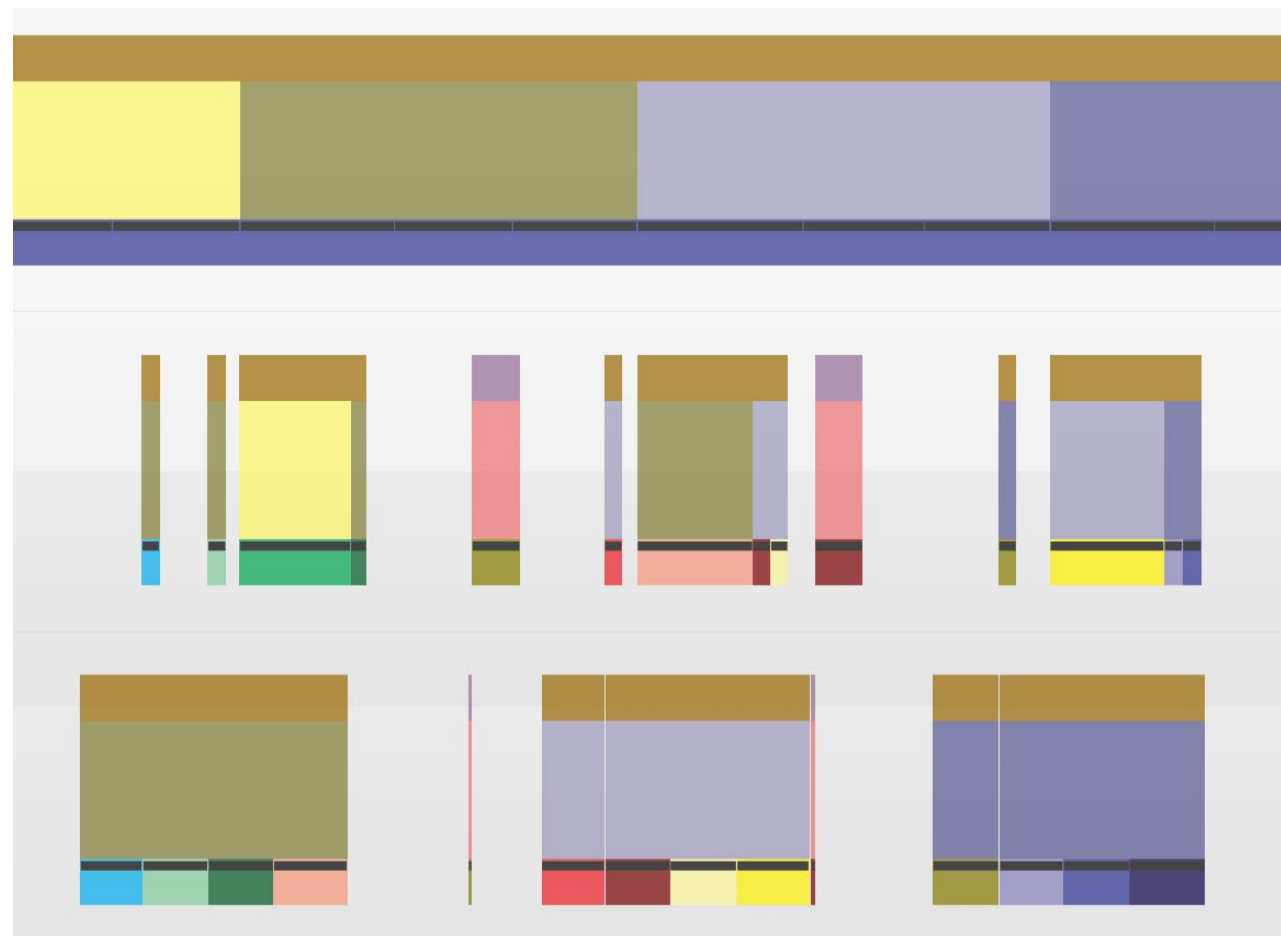
**Alternate between two queues** each frame for submitting commands.

Allows frame independent scheduling and increases task packing!

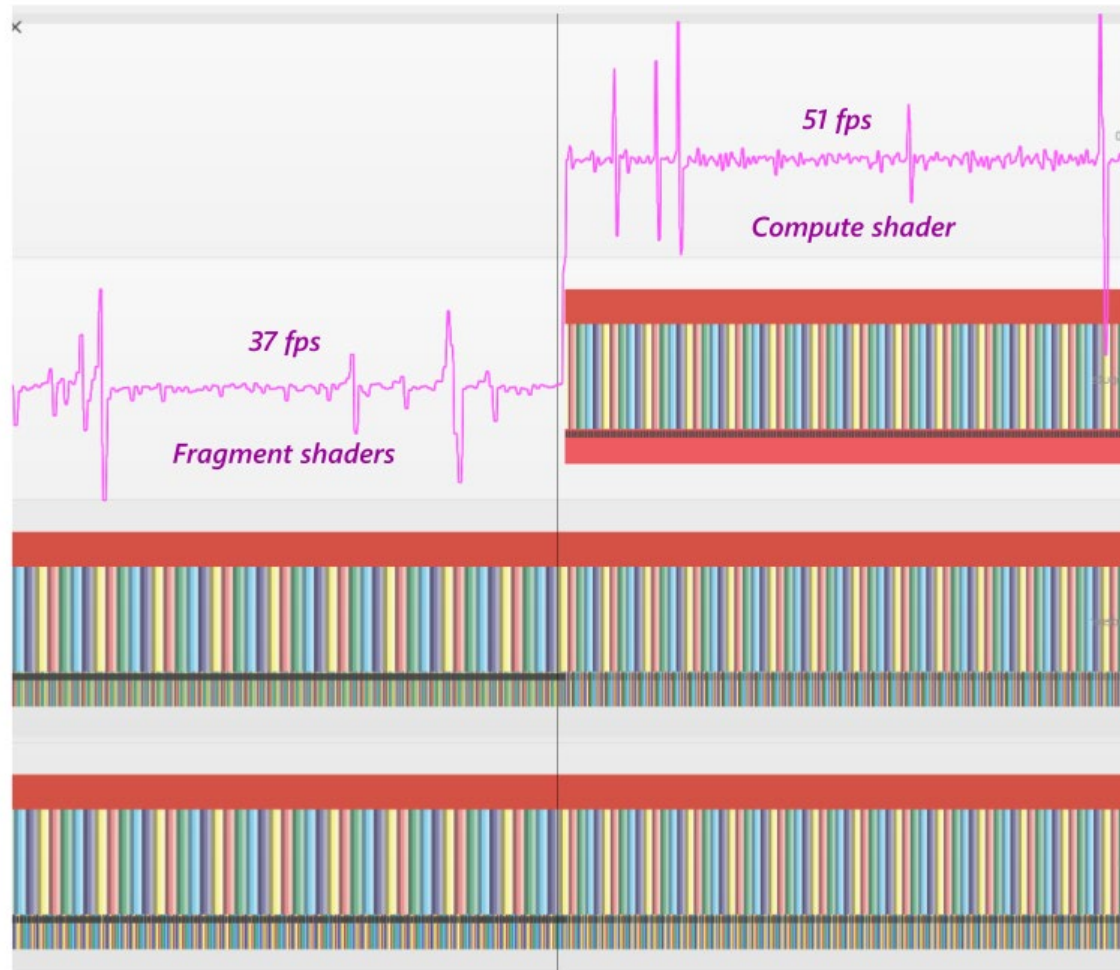
May require alternating between two sets of resources.

More developer recommendations!

<https://docs.imgtec.com/>



# Compute improvements

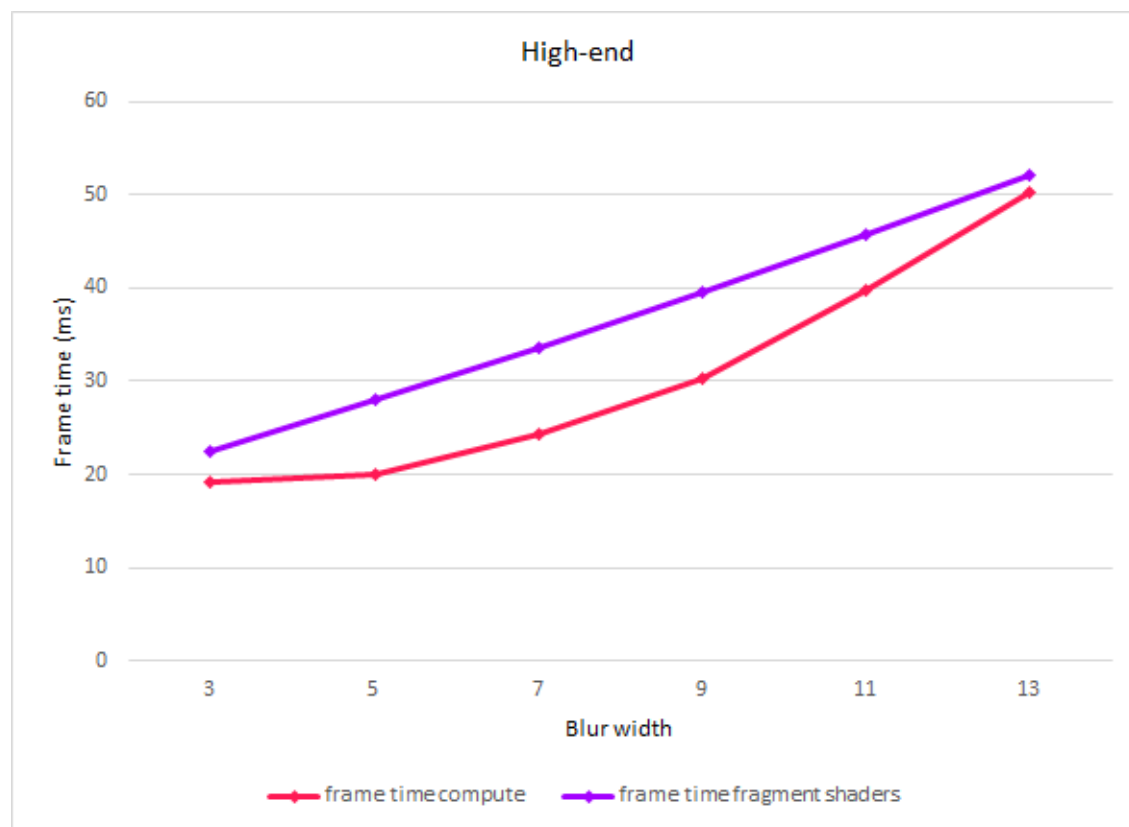


40% fps increase for 5x5 blur!  
**Meizu Pro 7 Plus - PowerVR 7XTP**

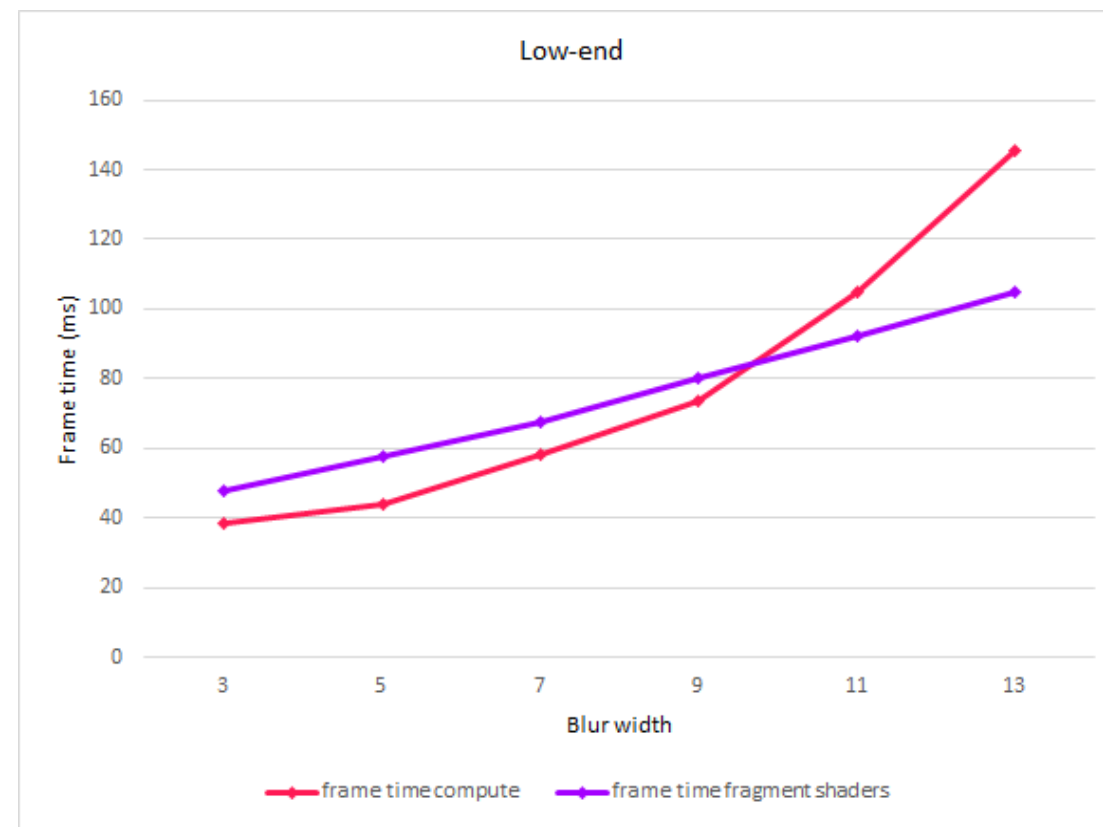


# Compute improvements

Frame times for high-end PowerVR GPUs



Low-end PowerVR GPUs



Thank you!