World War Z - Using Vulkan® to tame the zombie swarm

Nikolai Petrov, Saber Interactive

Jordan Logan, AMD

Nikolai Petrov

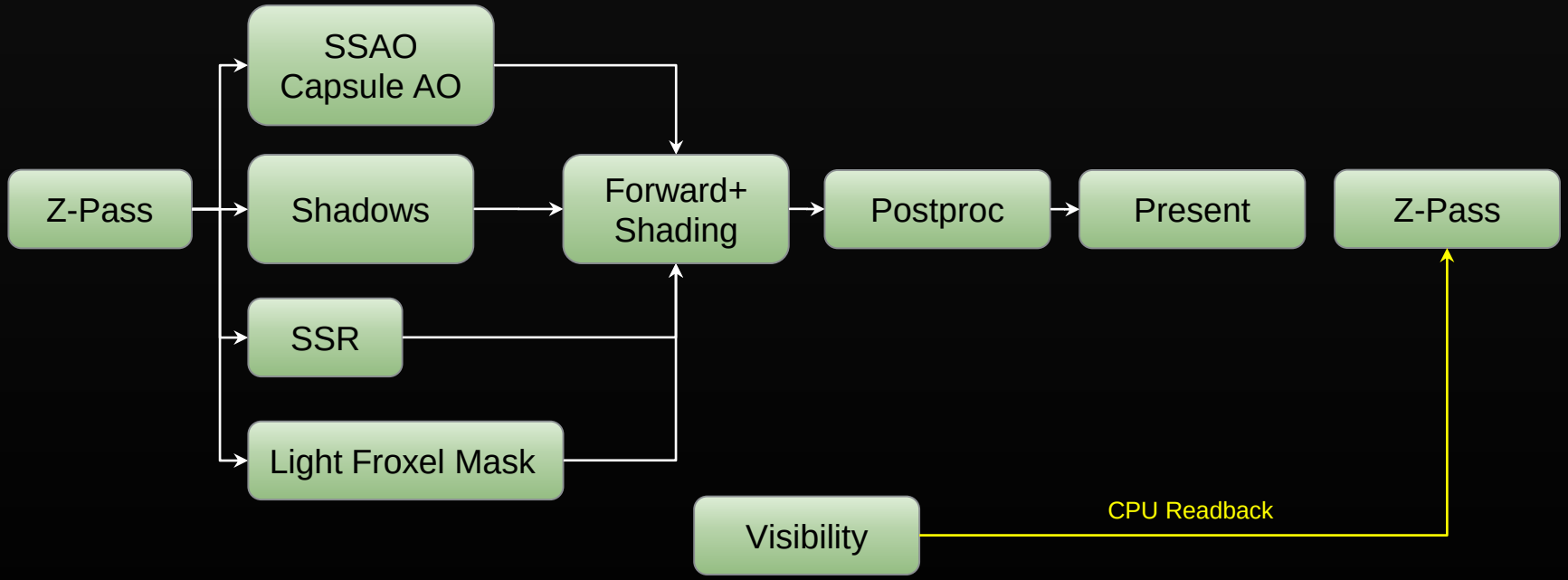
World War Z

- Cooperative 3rd person shooter, up to 4 players
- Large zombie crowds onto the screen
- PC rendering
  - Vulkan/DX11 backends
- Consoles support (Xbox One/PS4)
  - 30 fps
  - 4k rendering (dynamic resolution)

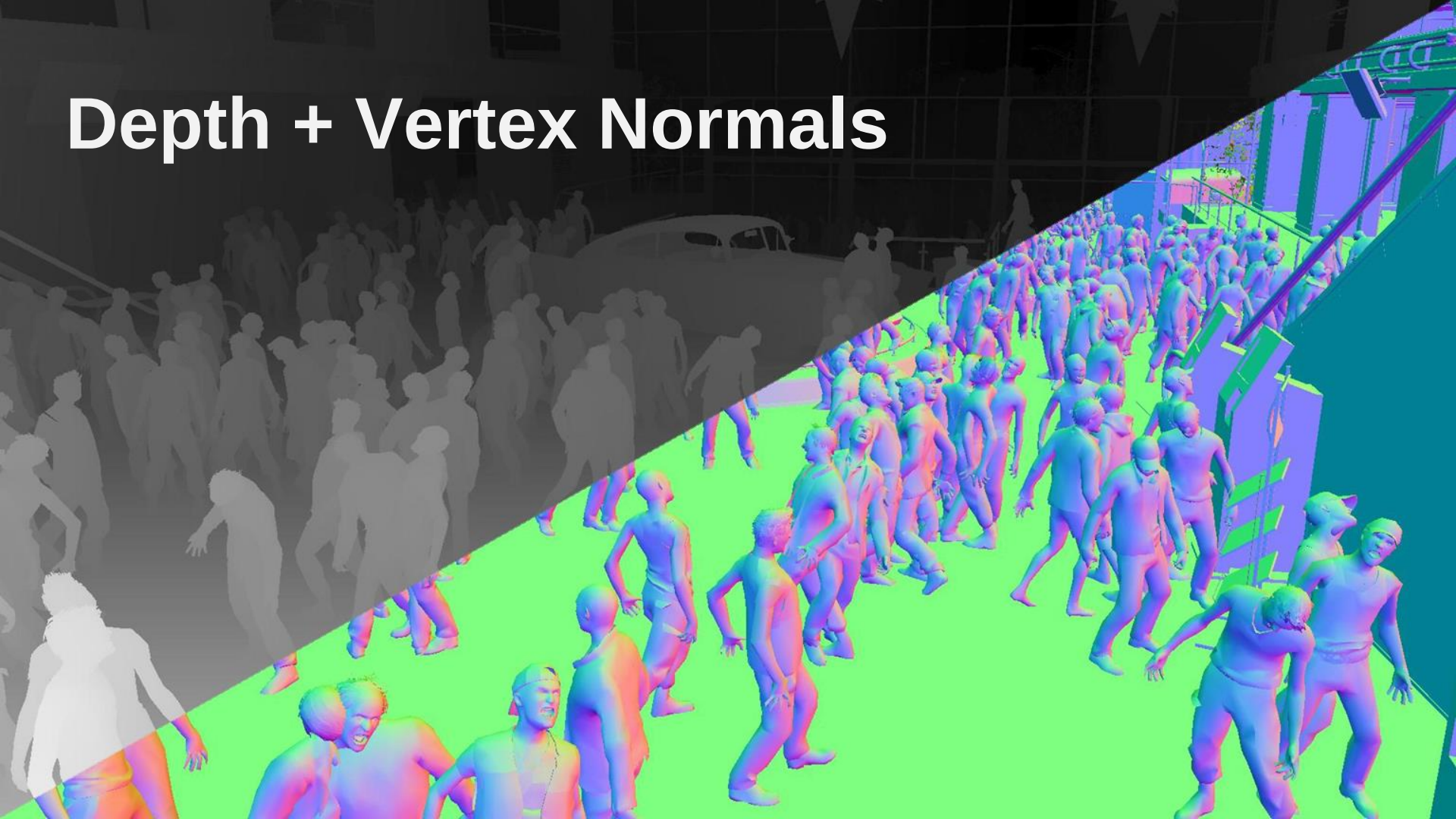
Pipeline overview

- Full depth prepass
- Deferred shadowmask (4 lights)
- Forward+ shading
- GPU-driven visibility system
  - 2 frames latency

GPU Workflow



Depth + Vertex Normals



Shadowmask: 4 lights, filtered

SSAO + Capsule shadows[Iwanicki13]

After shading & postproc

Short review

- Check presentations from the GDC 2019 for more details
  - Zombie rendering tech
    - “High zombie throughput in modern graphics”
  - Lightmap technology
    - “Enabling light baking workflows”

Why Vulkan?

- Designed to make CPU/GPU frame time lower
  - Multithreading
  - Async Compute
- Explicitly manage memory
  - Implement dynamic resolution
  - Alias memory resources
- Runs on different operating systems

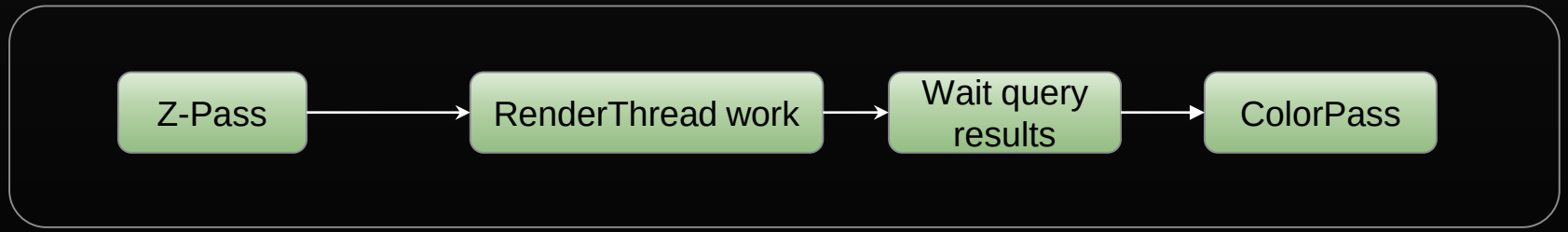
CPU performance

- Zombie crowds
- Large number of drawcalls (up to 3k)
  - Chance to be CPU-bound

CPU performance

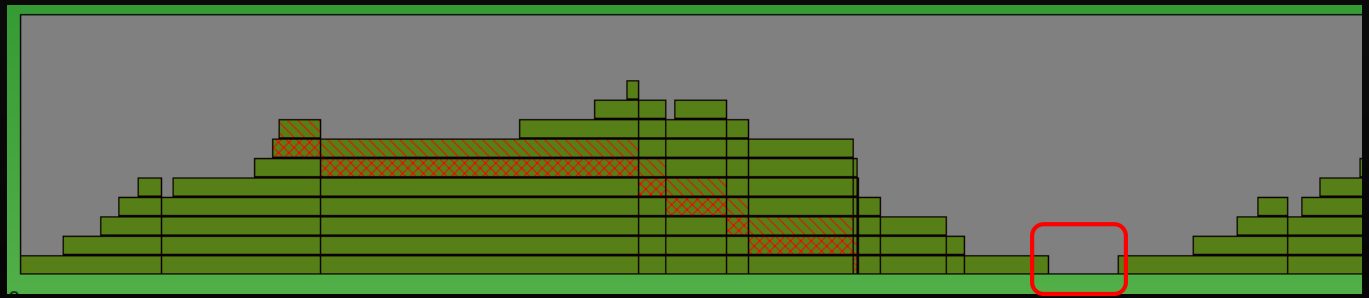
- Wait GPU occlusion query (OQ) results
  - Reduce colorpass DIPs number by 30%

CPU render submission



CPU performance

- Possible GPU idle
  - GPU has done its job, but CPU is not ready to submit new commands

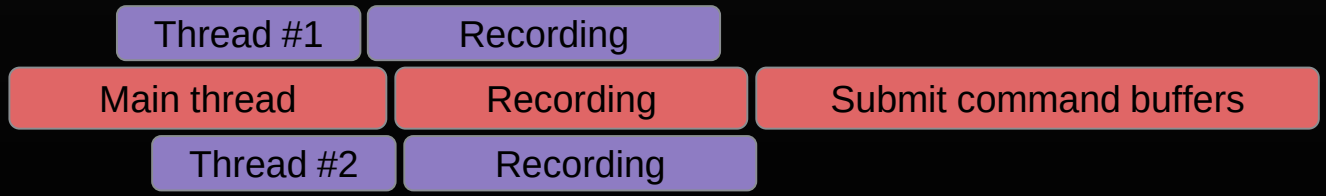


CPU performance

- Direct3D 11
  - Dedicated driver thread
  - Explicitly flush command buffer queue
    - Extra CPU time cost
- Vulkan
  - Manually control submission
  - Split work across several threads

CPU performance

- Record command buffers in parallel
  - Check # of physical cores
    - If >= 4, use 1 main + 2 extra threads scheme
- Split the most largest passes
  - Z Prepass
  - Shadowmap
  - Colorpass



CPU performance

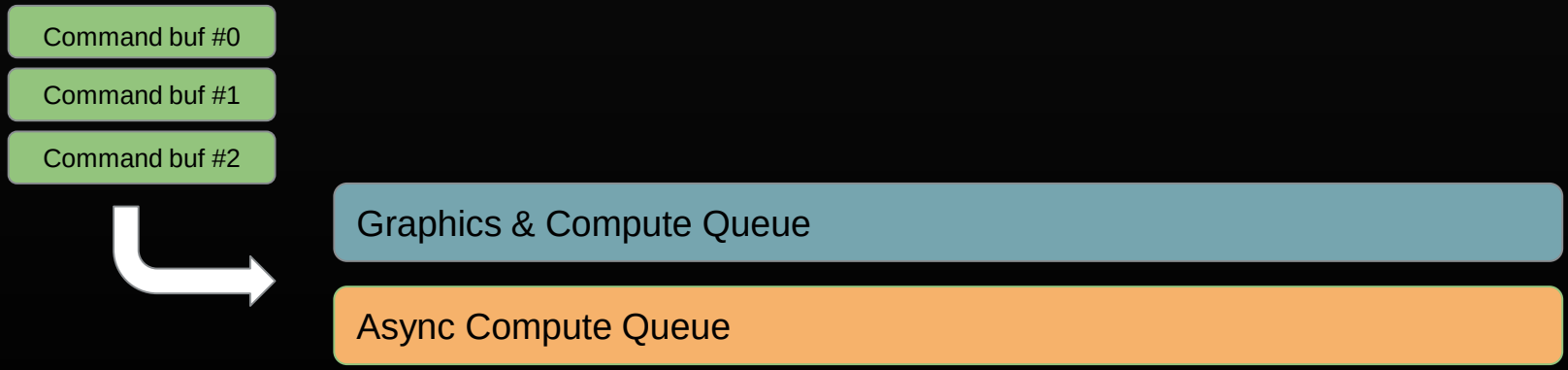
- World War Z
  - 24 command buffers per frame
    - Double buffering to avoid synchronization
    - Wait for GPU before the main shading pass
  - Only 5 queue submissions
    - Each vkQueueSubmit has limited CPU overhead

Multithreading benefits

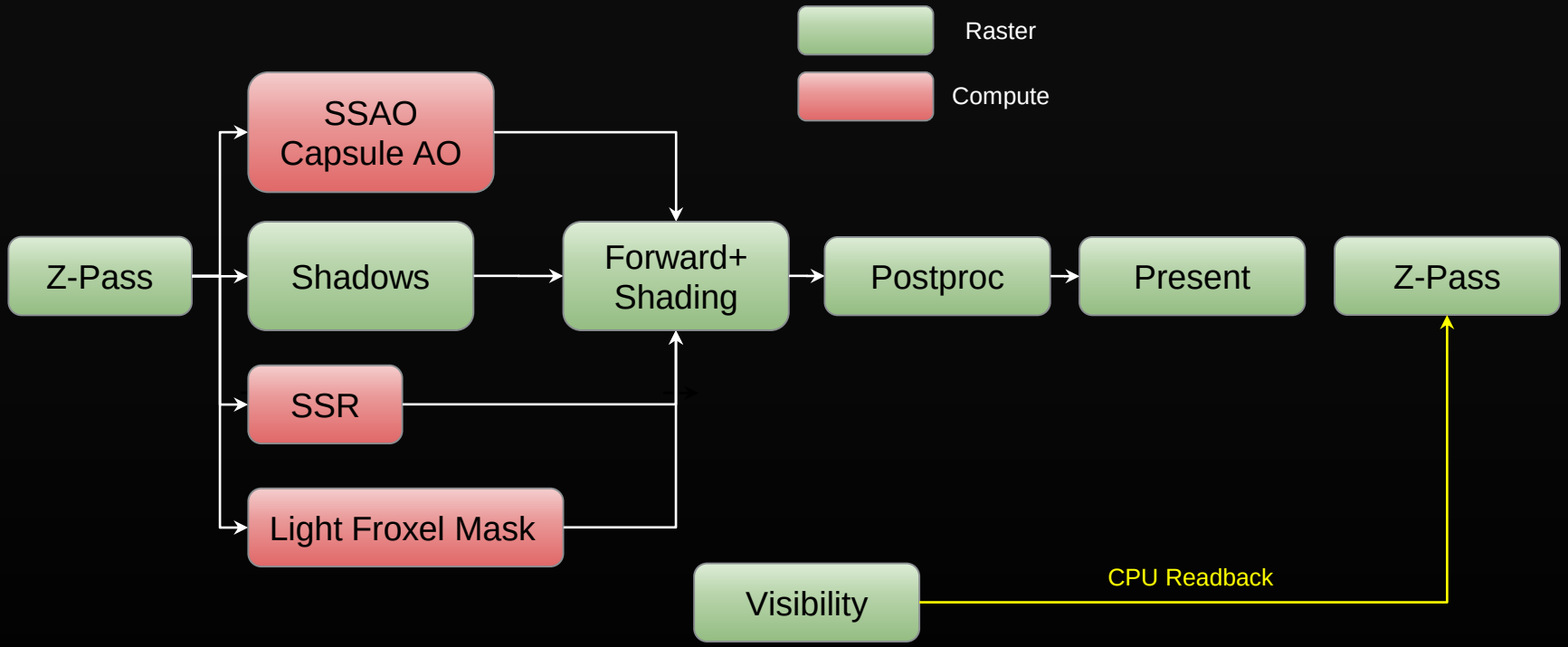
- Separate recording from submission
  - Allows for much higher throughput
- In critical cases may save more than 40% of CPU time
  - From 18-20 ms down to 12-16 ms (AMD Ryzen 7 2700X)

GPU performance

- Improve GPU utilization
- Share resources with ROP-bound passes
  - Shadowmaps, occlusion testing...
- Use another hw queue
  - Run compute shaders simultaneously



GPU async workflow



GPU occupancy

- VGPR pressure
  - Material fetching
  - Reflection & lighting calculation loops
- Try to keep your registers amount as low as possible
  - Helps to hide memory latency

VGPR	<=24	28	32	36	40	48	64	84	<=128	> 128
Waves	10	9	8	7	6	5	4	3	2	1

GPU occupancy

- Pack divergent data tightly
  - 2xfp32 to 1xfp16
    - packHalf2x16 / unpackHalf2x16
  - GL_ARB_gpu_shader_int64
    - 64-bit bitwise operations
- Apply cross-lane wave intrinsics
  -

GPU occupancy

```
vec4 albedoCol = GetAlbedoColor
(texUV.xy);
uint2 packedAlbedo =
uint2(packHalf2x16(albedoCol.xy),
packHalf2x16(albedoCol.zw));
....
vec4 unpackedAlbedo =
float4(unpackHalf2x16(packedAlbedo.x),
unpackHalf2x16(packedAlbedo.y));
```

```
image_sample  v[0:3], v[0:2], s[8:15],
s[16:19] dmask:0xf
v_cvt_pkrtz_f16_f32  v0, v0, v1
v_cvt_pkrtz_f16_f32  v1, v2, v3
....
v_cvt_f32_f16  v2, v0
v_cvt_f32_f16  v0, v0 src0_sel: WORD_1
v_cvt_f32_f16  v4, v1
v_cvt_f32_f16  v1, v1 src0_sel: WORD_1
```

GPU performance

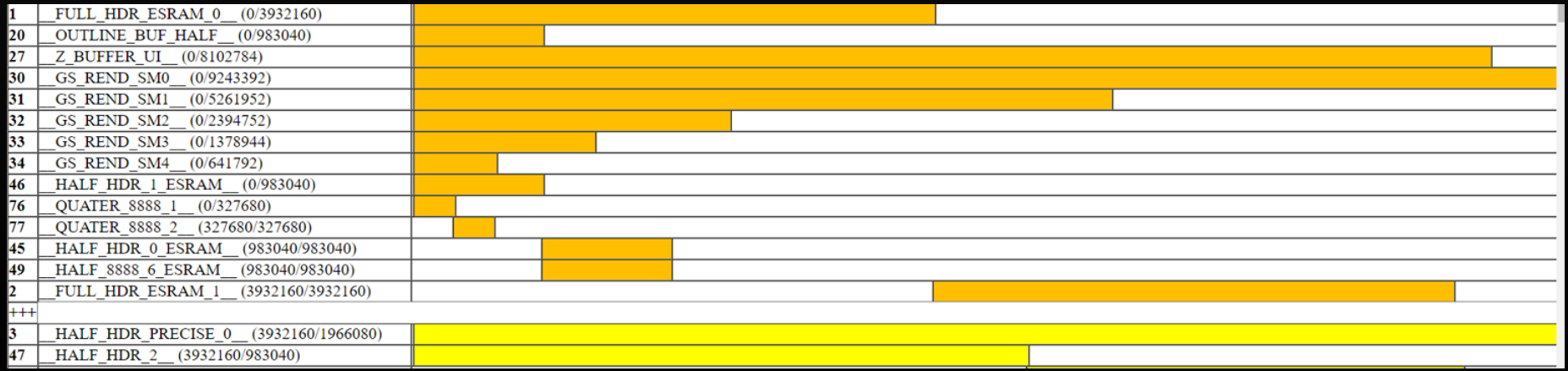
- Run compute shaders in parallel
  - Can save up to 1.5 ms (10 %) in some cases (AMD Radeon RX480)
- Can greatly reduce VGPRs num from intrinsics & packing
  - Best case: from 113 up to 64
  - Decrease GPU frame time by 33%

Memory management

- Vulkan® Memory Allocator from AMD
  - https://gpuopen.com/gaming-product/vulkan-memory-allocator/
- Designed to:
  - Better manage memory
  - Optimize for specific platforms
  - Alias transient resources

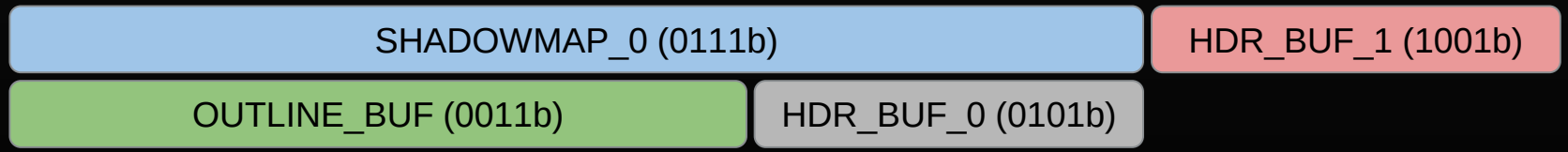
Alias transient resources

- Fixate render target amount beforehand
- Analyze lifetime dependencies
  - Store sharemasks for each RT
- Want to achieve lower upper memory bound



Alias transient resources

- For each target get aliased resources
- Calculate memory pool block layout
  - Share space with most similar placed RT
    - respect mask bits
- Allocate device memory block to cover all laid out targets

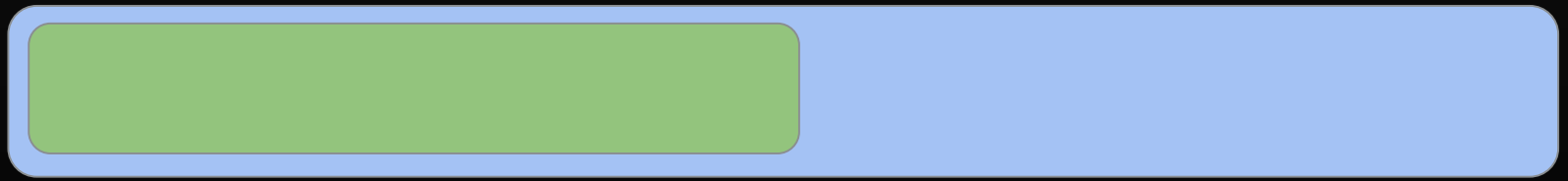


Alias transient resources

- Allocate first target #0 (sharemask: 0111b)

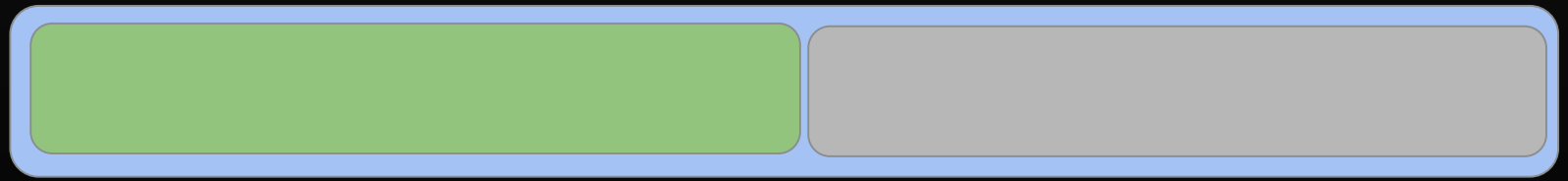
Alias transient resources

- Process target #1 (0011b)
  - It’s size is lower and (mask0 & mask1) != 0
  - Use same address as previous rt



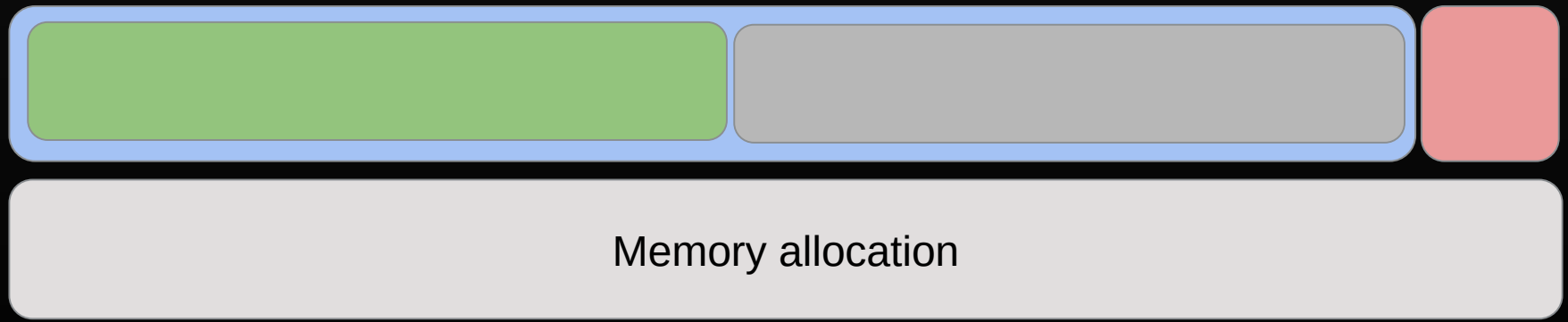
Alias transient resources

- Target #2 (sharemask 0101b)
  - Skip resource #1 block (no common bits)
  - Use remaining space within resource #0 space



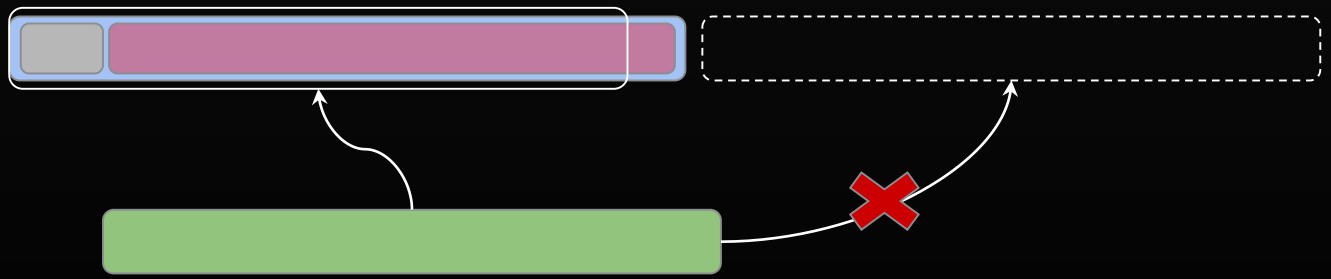
Alias transient resources

- Target #3 (1000b)
  - No target to share with, place to the end of the pool



Alias transient resources

- Take into account alignment requirements
  - Calculate appropriate offsets
- Utilize produced alignment holes
  - Try to overlap them with next blocks

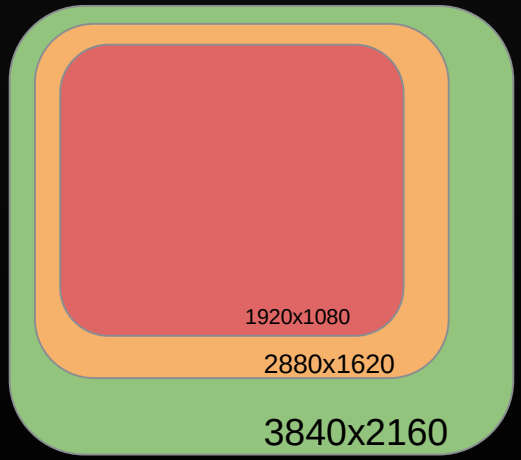


Alias transient resources

- Can be used to save video memory
  - More than 50% (351 vs 198 Mb)
- Carefully share compressed RT with UAVs
  - Use explicit barrier to switch between 2 images
    - Old layout = UNDEFINED

Dynamic resolution

- For each render target
  - Create alternate size versions
  - Map them to the mem address of original target



Dynamic resolution

- Set the FPS goal target
- Measure frame statistics:
  - CPU/GPU timings
- Use exponential smooth average
  - 2 frames history
  - Faster response to frameload changes

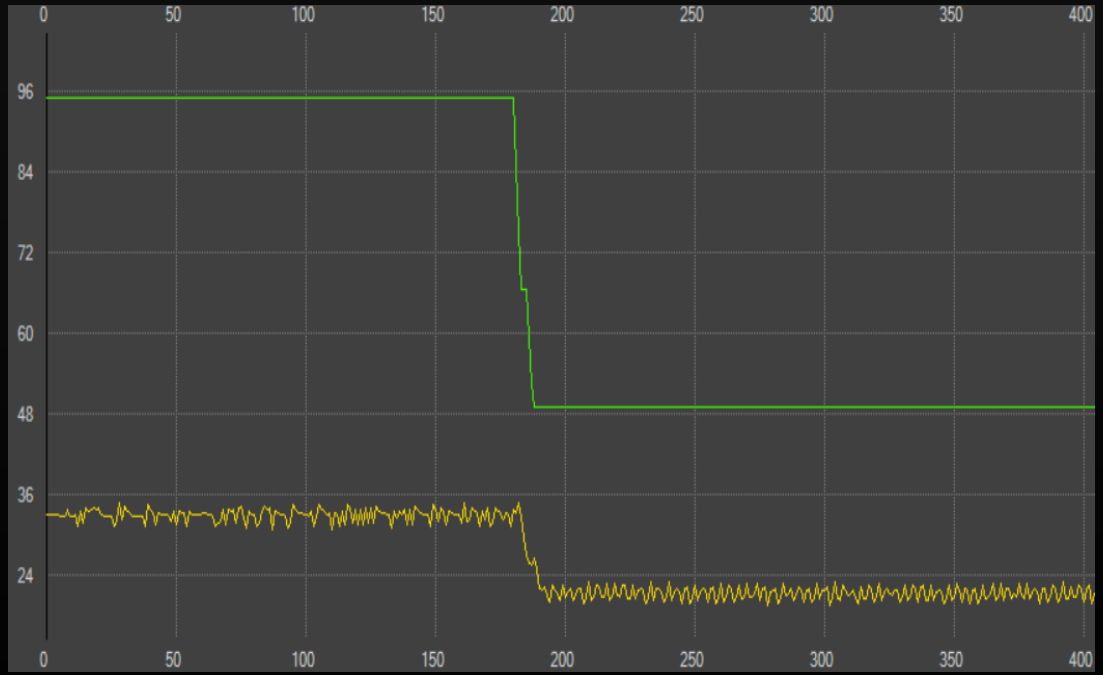
Dynamic resolution

- GPU bound
  - Average fps < target fps
    - Drop resolution by 1 step (5 %)
  - Average GPU time higher than desired
    - Use more aggressive scheme (2 steps) (10 %)
- Otherwise (GPU usage < 90%)
  - Increase res by 1 step (5%)

Dynamic resolution

- Apply downscale immediately
  - Near constant framerate
- Upscale resolution after specified delay (20 frames)
  - Don’t want to switch resolution too often
  - Can makes the final image sharper

Dynamic resolution

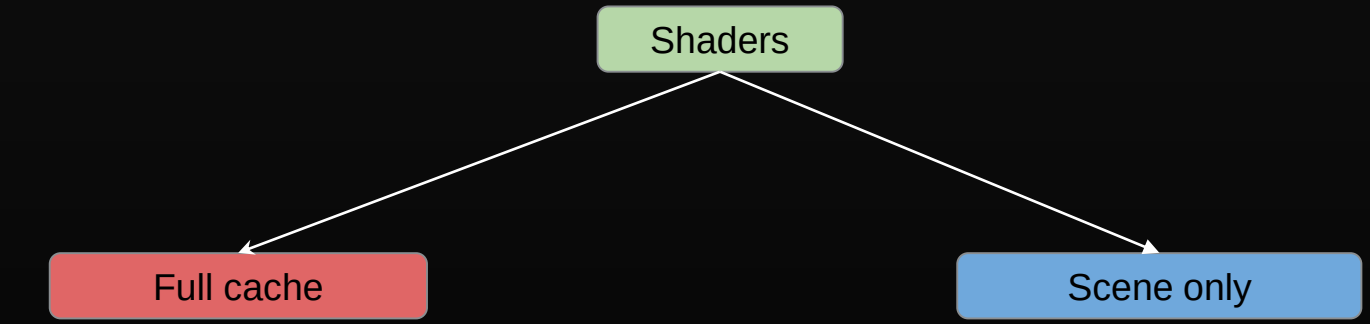


- 3840x2160 resolution
- Horizontal
  - Frame number
- Vertical:
  - Render target percentage
  - Average GPU time
- Low GPU time is better

PSO management

- vkCreateGraphicsPipelines works rather slow
  - Especially for the first time calls
- Want to decrease level loading time
- Want to eliminate potential spikes during gameplay sessions

PSO management



Particles
Dynamic decals
Post-process
(all combination accessible)

Objects materials
Static SFXes
(combinations used in scene)

PSO management

- Serialize scene PSO creation data during export
  - Shader defines, renderstates, rendertarget formats,...
- Create shaders during the level start
  - But what we should do with the full cache ones?

PSO management

- Simple solution :)
- Ask QA to play a couple of sessions for each level
  - Record data about used full cache PSO
- Use this information on export stage
- Just works in our case

PSO management

- Can reduce shader creation time significantly
  - Level loading: from 10 min up to 1.5
- Delete unused PSOs
  - Sometimes migrate to system ram
- Always enable pipeline objects cache
  - Could help when run game not for the first time

Jordan Logan

AMD DevTech

- We provide direct support with developers.
- Help with optimizations and profiling.
- Work with the driver teams to make sure that consumers have better experience.
- Deal with GPU specific issues.

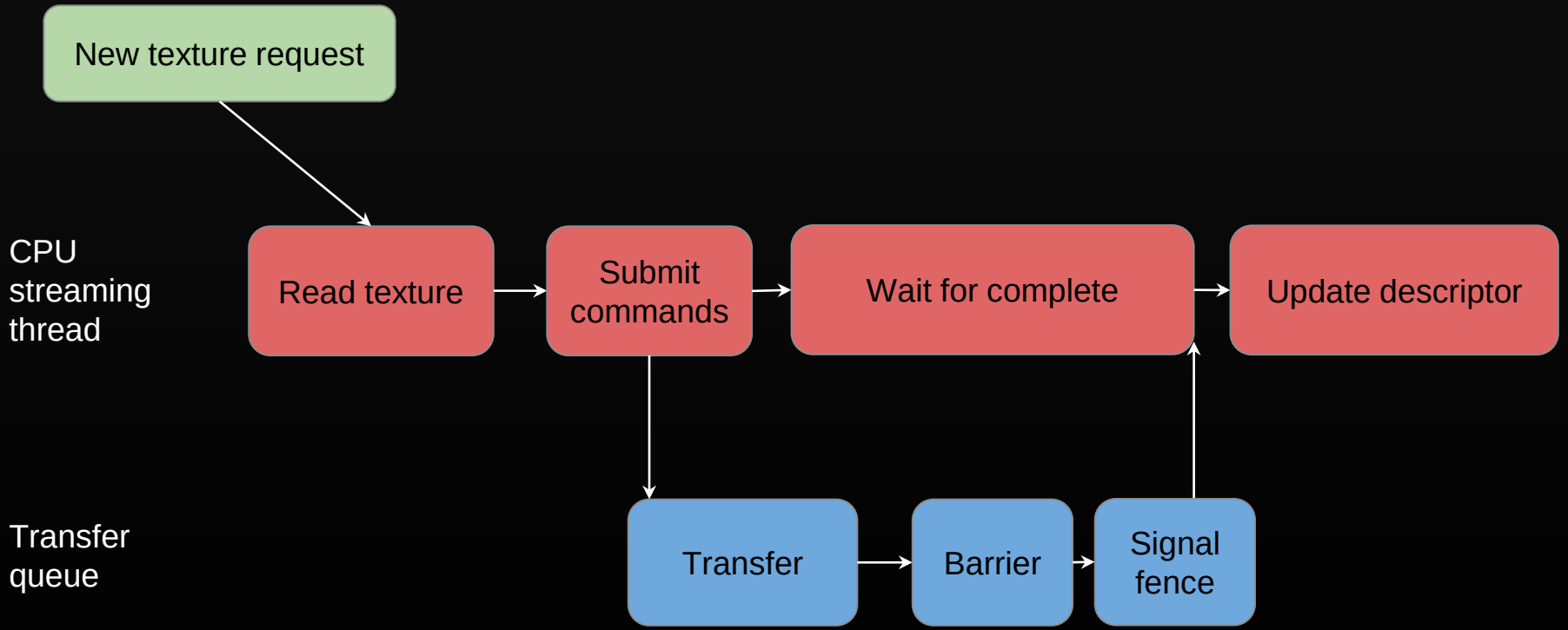
Transfer queue

- Vulkan exposes using the hardware DMA engine though the use of transfer queues.
  - The transfer queue is helpful on all platforms except APUs.
  - This piece of hardware can run completely async to the graphics and compute queues.
  - It is a faster way to transfer data across the PCI-e® bus.
- Must be explicitly used.
  - If you don’t use the transfer queue, then uploads will be going down the slow path.
  - Best used asynchronously. Uploads and downloads should not block the rendering loop.
  - The graphics queue should not wait for the transfer queue.

Texture Streaming

- Transfer queue is designed for texture uploads and streaming.
- In the streaming case the old texture can be used while the texture is streaming.
- Once the texture is uploaded all that has to be done is to update the next frames descriptors.
  - With persistent descriptors you will want 2 copies of every descriptor
  - This can allow updating without doing a full GPU/CPU sync.

Texture Streaming

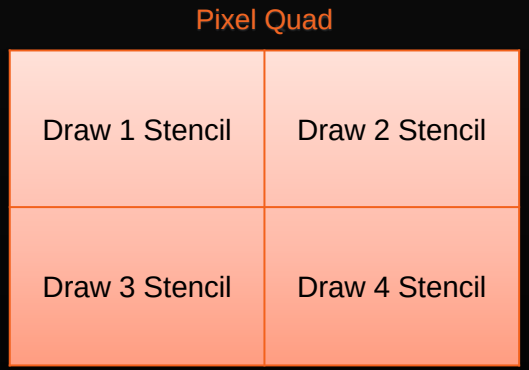


Transfer queue gotchas

- Transfer queue can have a different granularity then other queues.
  - The copy must be either a full sub-resource copy or be divisible by the queue granularity.
  - Undefined behavior can happen if you don’t follow the rules.
    - Common seen behavior is that the transfer queue will hang.
- Missing barrier on queue could cause corruption.
  - Stale data in cache, etc.

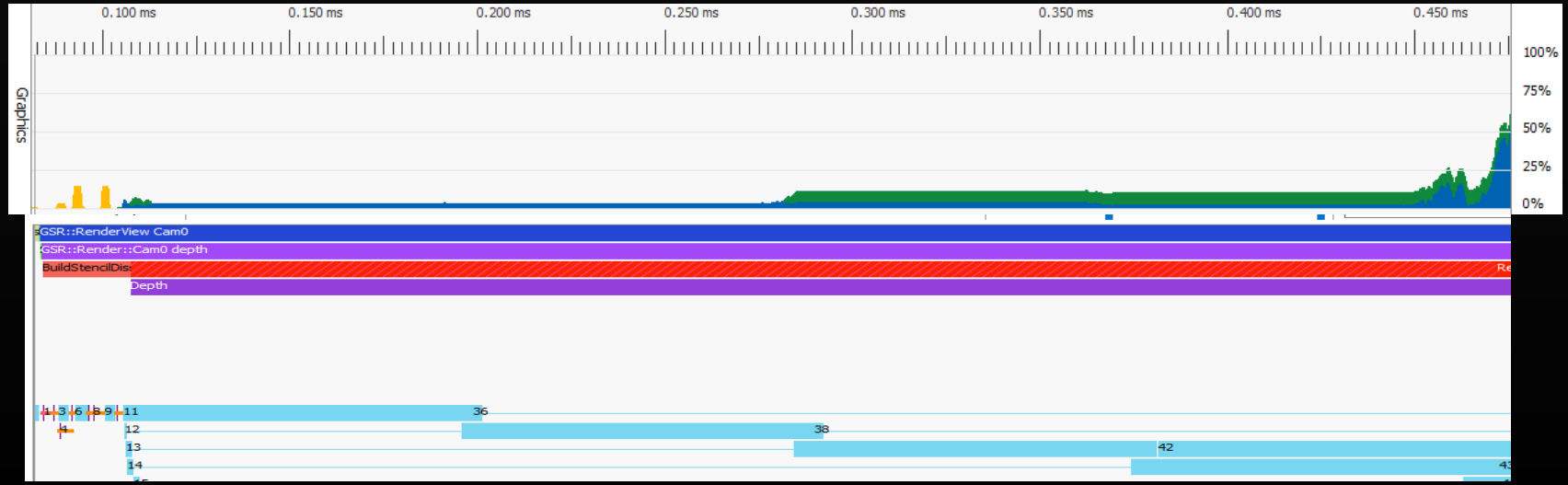
Stencil optimizations

- A stencil mask is created with a checkerboard like pattern.
  - This is done with 4 draws.
  - Each draw has a different stencil ref and rejects pixels based on the position.



Stencil optimizations

- Radeon GPU Profiler shows very low occupancy.
- Why is the occupancy low?
  - Shader is very small and does not do much.
  - Shader waves are finishing faster than they can be launched.



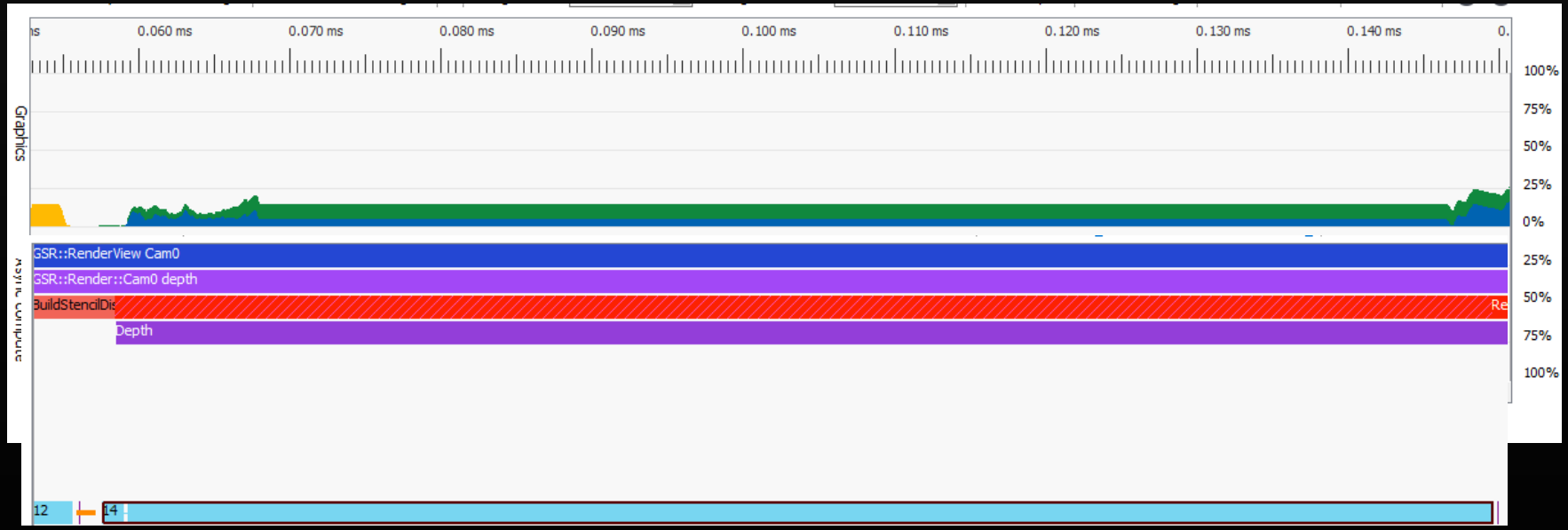
Stencil optimizations

- Enter VK_EXT_shader_stencil_export
  - VK_EXT_shader_stencil_export is an extension supported by multiple vendors that allows the pixel shader to set the stencil ref value per pixel.
  - With this we can combine the 4 draws into 1.

- GLSL
  - gl_FragStencilRefARB = int(lut[y * DITHER_PATTERN_SIZE + x]);
- HLSL
  - int main() : SV_StencilRef

Stencil optimizations

- Saw ~75% savings for the pass.

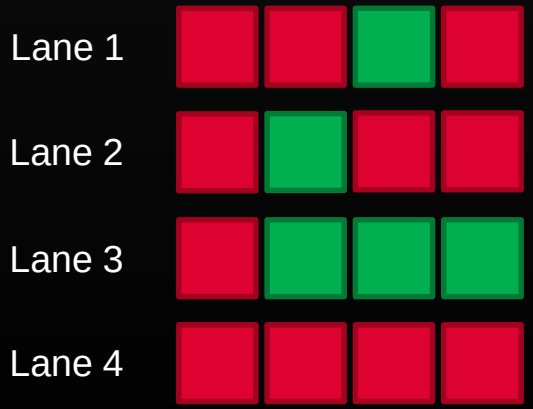


Subgroup ops

- Subgroup ops introduced in Vulkan 1.1, supported by most desktop hardware, including AMD
  - Enable bringing over optimizations from other gaming platforms.
  - Allow lots of new potential optimizations.
  - Query the driver to see what ops are supported.

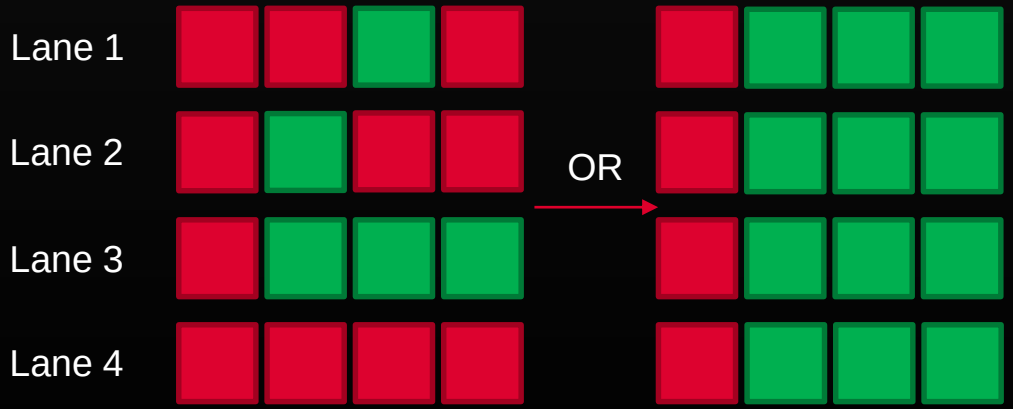
Subgroup ops

- Reduced divergence in the wave and scalarized some resources by using subgroupOr to unify the lighting bitmask.
  - Before the shader would loop though every light in the bitmask.
  - Changed it so every lane goes though the same lights. This allows some resources to be converted to scalars.



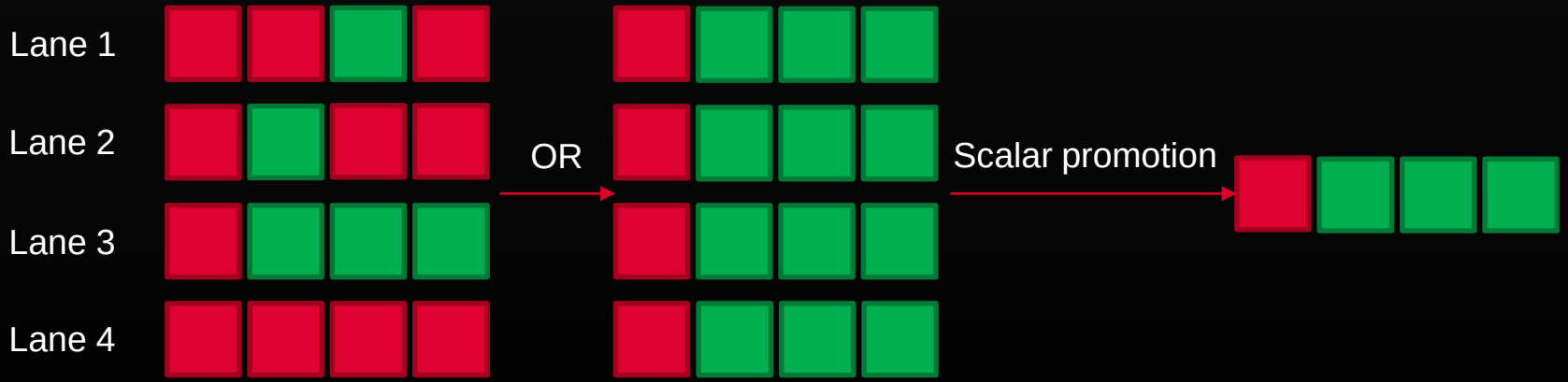
Subgroup ops

- Reduced divergence in the wave and scalarized some resources by using subgroupOr to unify the lighting bitmask.
  - Before the shader would loop though every light in the bitmask.
  - Changed it so every lane goes though the same lights. This allows some resources to be converted to scalars.



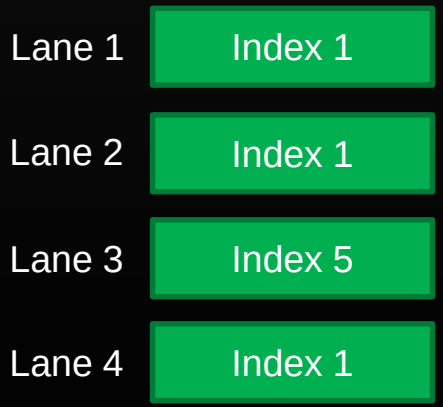
Subgroup ops

- Reduced divergence in the wave and scalarized some resources by using subgroupOr to unify the lighting bitmask.
  - Before the shader would loop though every light in the bitmask.
  - Changed it so every lane goes though the same lights. This allows some resources to be converted to scalars.



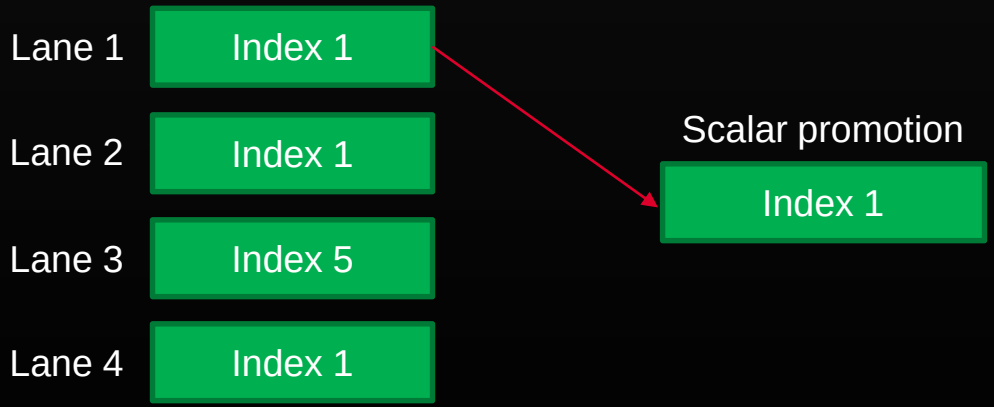
Subgroup ops

- Scalarized cubemask look ups by using subgroupBroadcastFirst to convert cubemask index to a scaler.
- subgroupBroadcastFirst used a lot for scalarization of shader code.



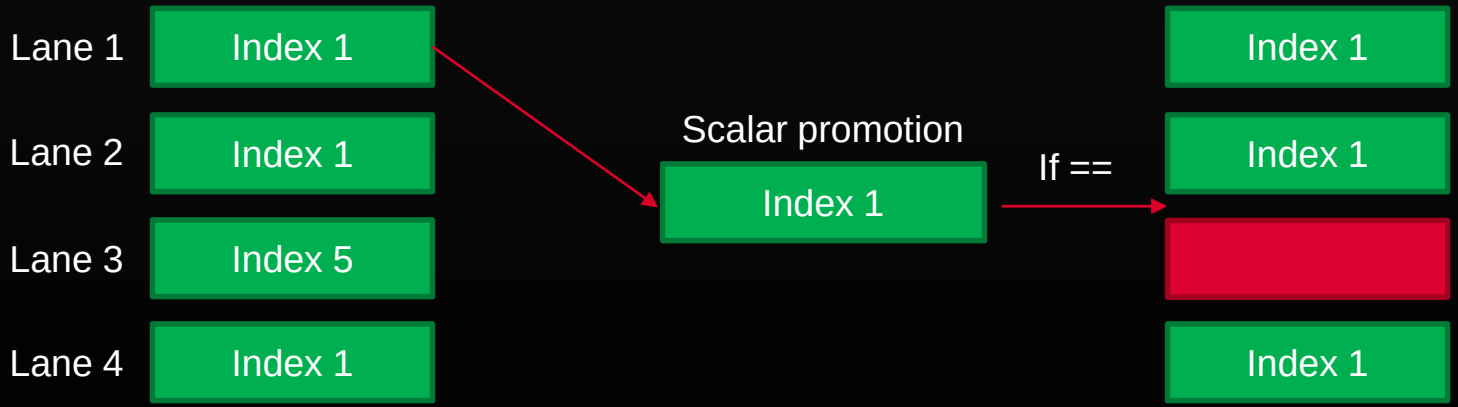
Subgroup ops

- Scalarized cubemask look ups by using subgroupBroadcastFirst to convert cubemask index to a scaler.
- subgroupBroadcastFirst used a lot for scalarization of shader code.



Subgroup ops

- Scalarized cubemask look ups by using subgroupBroadcastFirst to convert cubemask index to a scaler.
- subgroupBroadcastFirst used a lot for scalarization of shader code.



Subgroup ops

- Scalarized cubemask look ups by using subgroupBroadcastFirst to convert cubemask index to a scaler.
- subgroupBroadcastFirst used a lot for scalarization of shader code.

Subgroup ops

- Scalarized cubemask look ups by using subgroupBroadcastFirst to convert cubemask index to a scaler.
- subgroupBroadcastFirst used a lot for scalarization of shader code.

Subgroup ops

- Scalarized cubemask look ups by using subgroupBroadcastFirst to convert cubemask index to a scaler.
- subgroupBroadcastFirst used a lot for scalarization of shader code.

Thanks

- AMD
  - Mike Smith
  - Jordan Logan
  - Adam Sawicki
- Saber Interactive
  - Denis Sladkov
  - Max Gridnev
  - Ivan Popov
  - Ivan Shostak
  - Aleksander Skolunov
  - Timur Gagiev
  - Timur Popov

Q & A

Addendum

Testing done by Jordan Logan and Nikolai Petrov.

Testing by Jordan done on AMD Ryzen™ 7 1800x Processor, 2x16GB DDR4-2666, Vega64 (driver 19.10.2), ASUS Prime X370-PRO Socket AM4 motherboard, WD Blue 250GB M.2 SSD, Windows 10 x64 Pro (RS4).

Testing by Nikolai done on AMD Ryzen™ 7 2700x , 32GB, RX 580 (driver 19.10.2), Windows 10 x64.

Disclaimers and Attribution

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

THIS INFORMATION IS PROVIDED ‘AS IS.” AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

ATTRIBUTION