AMD EPYC

AMD RYZEN

AMD RADEON

RDNA™ 3: BEYOND THE CURRENT GEN

LOU KRAMER

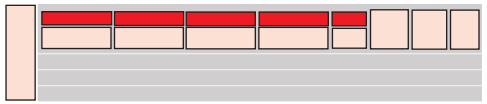
AMD
together we advance_

AMD GPUOpen

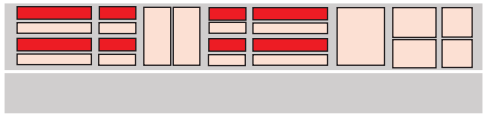
THE RDNA™ ARCHITECTURE

RDNA™ 1

New architecture
GCN → RDNA



4x Compute Unit



2x Dual Compute Unit

RDNA™ 2

New hardware features

- Raytracing
- Mesh Shader
- Variable Rate Shading
- Sampler Feedback

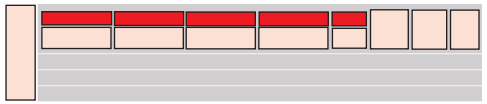
RDNA™ 3

- … ? ☺

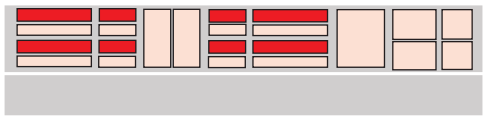
THE RDNA™ ARCHITECTURE

RDNA™ 1

New architecture
GCN → RDNA



4x Compute Unit



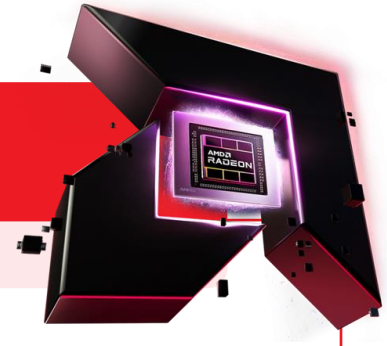
2x Dual Compute Unit

RDNA™ 2

New hardware features

- Raytracing
- Mesh Shader
- Variable Rate Shading
- Sampler Feedback

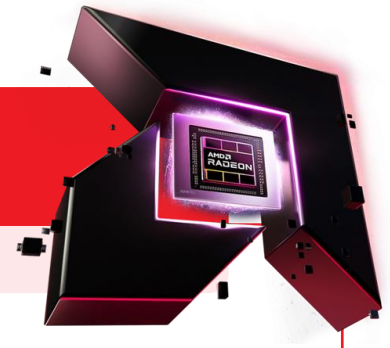
RDNA™ 3



- Raytracing 2nd Gen
- AMD Infinity Cache™ 2nd Gen
- AI Acceleration
- AMD Radiance Display™ Engine
- Chiplet Design

AGENDA

- RDNA™ 3 overview
- Draw/Dispatch Submission
  - Execute Indirect
- (Compute) Shaders
- Memory Bandwidth
  - Shader Write Compression
  - **AMD Infinity Cache™ 2nd Gen** → Work Tiling
- **Raytracing 2nd Gen**



- **Raytracing 2nd Gen**
- **AMD Infinity Cache™ 2nd Gen**
- AI Acceleration
- AMD Radiance Display™ Engine
- Chiplet Design

RDNA™ 3 - ARCHITECTURE OVERVIEW

- The flagship card AMD Radeon™ RX 7900 XTX has 96 compute units

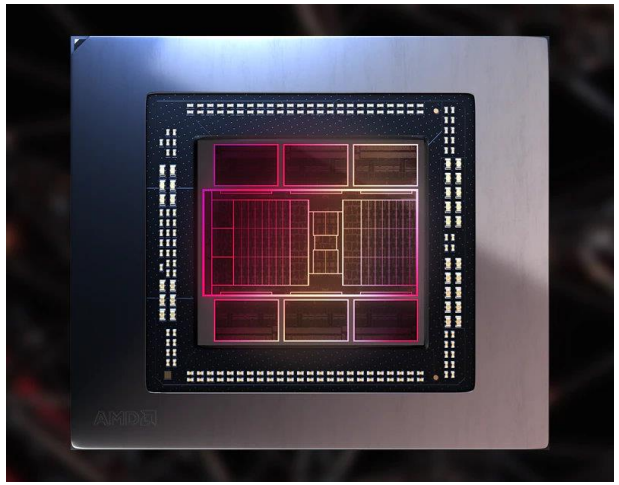
→ A lot of compute units to distribute work to

- To utilize the GPU efficiently, we want them to be busy

- AMD Infinity Cache™ helps to boost effective memory bandwidth

→ Good Infinity Cache™ utilization is important

	RX 6900 XT	RX 7900 XTX
Compute Units	80 CUs	96 CUs
Memory Bandwidth	512 GB/s	960 GB/s
Infinity Cache Size	128 MB	96 MB
Effective Memory Bandwidth	1024 GB/s	3500 GB/s



DRAW/DISPATCH SUBMISSIONS

- Do as much work as possible per draw/dispatch submission
- Aggregate work
- Reduce barriers that serialize small workloads
  - Async queue can help
- Sort by expensive state change that roll contexts
  - https://gpuopen.com/learn/understanding-gpu-context-rolls/
  - Still true today!
  - I highly recommend to read it ☺
  → Sort your draw calls by material!

Understanding GPU context rolls

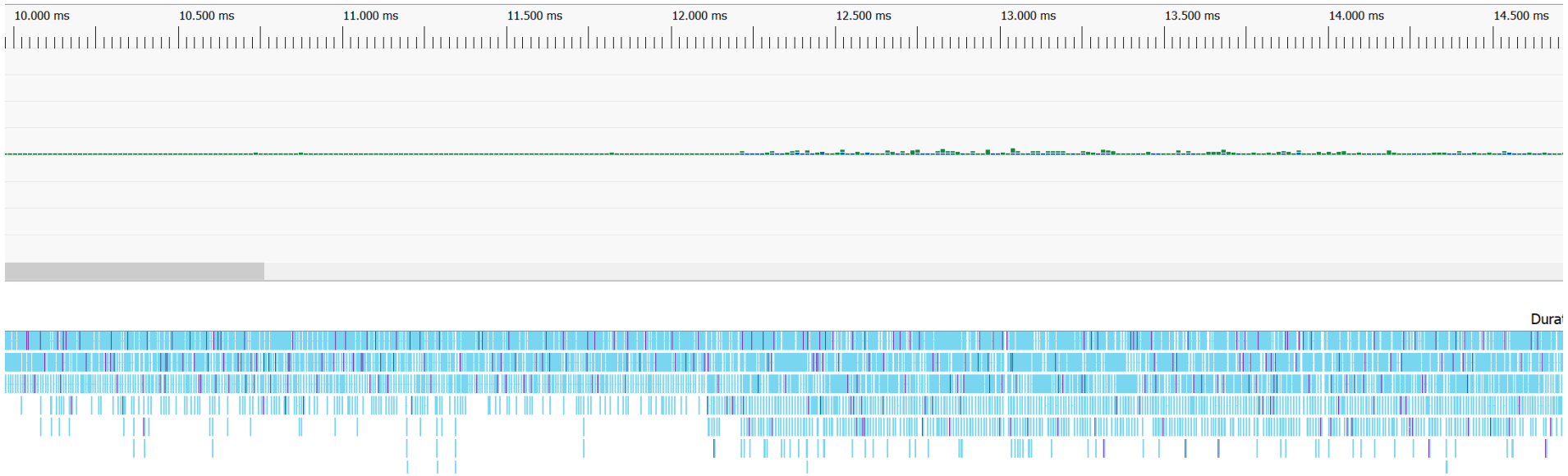
Rys Sommefeldt

Originally posted June 29, 2018

"So, if the timeline [in RGP] looks like a unicorn came to visit, … you [possibly] rolled too much ."

EXECUTE INDIRECT

- Allows generating work from the GPU without reading back to the CPU
- Offers opportunity to offload some work from the CPU to the GPU
- However …



- What happened?
- Depending on the command signature, the submission can be fast or **very** slow!

EXECUTE INDIRECT

- The fastest path in ExecuteIndirect is to avoid changing any root signature attributes

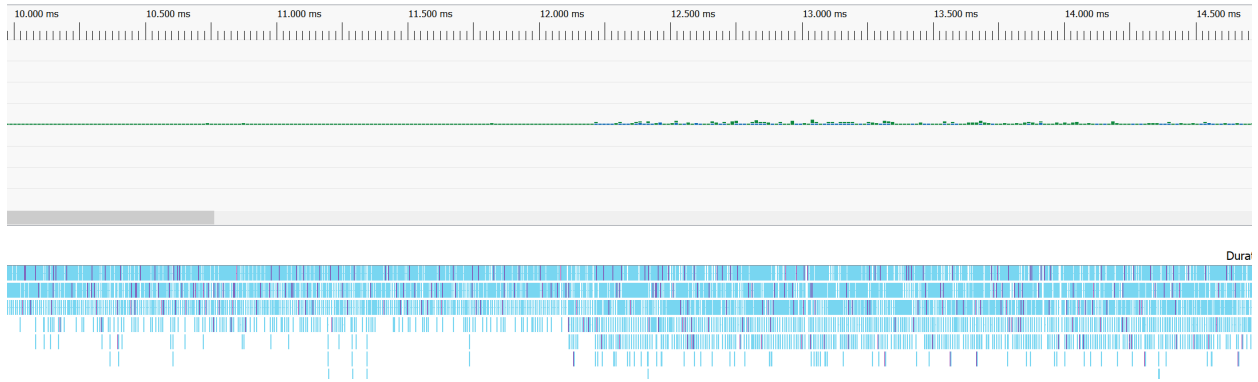
obj#73	D3D12 Command Signature
Desc	
ByteStride	12
NumArgumentDescs	1
ArgumentDesc[0]	
Type	DISPATCH
NodeMask	0
Root Signature	<not set>

- Changing root signature attributes can still be ok but
  - Avoid updating vertex buffer bindings through ExecuteIndirect
    - Use the vertex pull method instead by directly loading the vertices through Buffer/StructuredBuffer reads in the shader
  - Avoid updating any root signature values that may have spilled to memory
    - Keep values that change per draw to the lowest entries of the root signature
  - Keep root signature updates contiguous when updating through ExecuteIndirect
    - For example, updating entries [0, 2, 4] is slower than updating entries [0, 1, 2]
  - Always compact the argument buffer and provide a count buffer if any draws are culled out

EXECUTE INDIRECT

- What happened?
- Vertex buffer binding got updated!

obj#209	D3D12 Command Signature
Desc	
ByteStride	68
NumArgumentDescs	4
ArgumentDesc[0]	
Type	VERTEX_BUFFER_VIEW
Slot	0
ArgumentDesc[1]	
Type	INDEX_BUFFER_VIEW
ArgumentDesc[2]	
Type	CONSTANT
RootParameterIndex	13
DestOffsetIn32BitValues	0
Num32BitValuesToSet	4
ArgumentDesc[3]	
Type	DRAW_INDEXED
NodeMask	0
Root Signature	obj#207



→ Use the vertex pull method

- Directly load the vertices through Buffer/StructuredBuffer reads in the shader

(COMPUTE) SHADERS

- RDNA introduced native support for Wave32
- Wave64 also supported
- Which one is it better to use (very generically)
  - Wave32 for highly divergent code or long running threads
  - Wave64 for ALU heavy and better data access patterns
    - See later for shader write compression
- Can control this with WaveSize shader attribute in SM6.6
  - Only currently applies to compute shaders
- Vulkan® has an extension too: `VK_EXT_subgroup_size_control`

- Driver will try to pick the optimal wave size for the given GPU
  - The optimal wave size can differ between GPUs
- If you do set it manually

→ Always measure!

MEMORY BANDWIDTH

- Memory Bandwidth is scarce

	RX 5700 XT	RX 6900 XT	RX 7900 XTX
Compute Units	40	80	96
Engine Clk (GHz)	1.9	2.2	2.4
Memory GB/s	448	512	960
Mem Bytes/CU/clk	5.9	2.9	4.2

- RDNA™ 2/3 have an Infinity Cache™

- Two main approaches to reduce bandwidth requirements
  - Compression
  - Caches → Infinity Cache™

DELTA COLOR COMPRESSION (DCC)

- Blogpost from 2016 on GPUOpen about DCC: https://gpuopen.com/learn/dcc-overview/
- It's still very relevant – if you have not yet, I recommend reading it ☺
- There have been gradual improvements
- Generally speaking: more resources benefit from compression compared to 2016!

RDNA™ 3:
Better handling of format differences in views as well as TYPELESS

How to prevent decompression? – RDNA™ 3

- Avoid `D3D12_RESOURCE_FLAG_ALLOW_SIMULTANEOUS_ACCESS`
- Avoid `VK_SHARING_MODE_CONCURRENT`
- Don't use the copy source or copy target flag if not needed
- Don't do partial writes

You can use RGP to check if your resource benefits from DCC!

LEVERAGING SHADER WRITE COMPRESSION

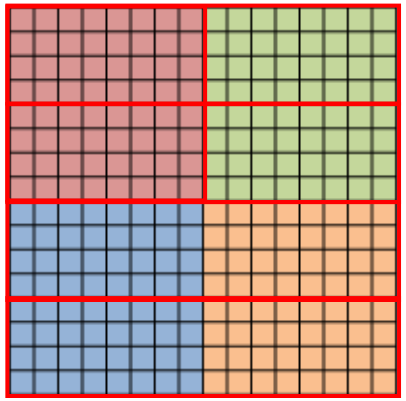
- Delta color compression (DCC) on UAV targets
- Shader write compression available since RDNA™ 1
  - Trade-off between added compression overhead and bandwidth reduction gains
- Making good use of DCC remains important for RDNA™ 2 and RDNA™ 3
- Reduces bandwidth requirements for the resource

	RX 5700 XT	RX 6900 XT	RX 7900 XTX
Compute Units	40	80	96
Engine Clk (GHz)	1.9	2.2	2.4
Memory GB/s	448	512	960
Mem Bytes/CU/clk	5.9	2.9	4.2

SHADER WRITE COMPRESSION

- Compressor is “finicky” for optimal performance
  - Surface are compressed as 2D 256Byte blocks
  - Single store operation from wave needs to fully update 256Byte block
  - → Sensitive to wave size, surface Byte per element (BPE), threadId mapping and thread-group dimensions
- Partial 256Byte block updates add significant compression overhead!

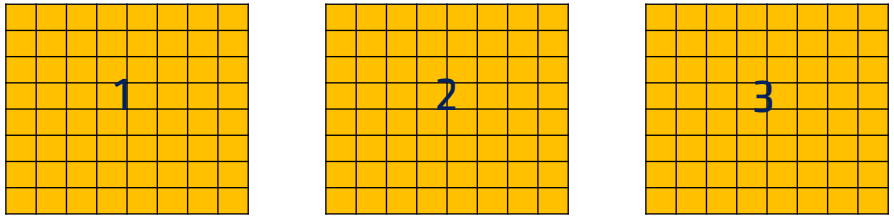
- For a 4BPE surface, assuming 1:1 threadId to UAV x,y
  - 2D 256Byte blocks: Arranged as 8x8 blocks (8*8*4 = 256)
  - Wave32 cannot do full overwrite (32*4 = 128)
  - 16x16 thread-group does not fully overwrite
    - Four 16x4 wave64
  - Best bet is wave64 with 8x8 thread-group
  - For large thread-groups remap threadIds to logically 8x8
- Write to all channels (don’t do partial writes)



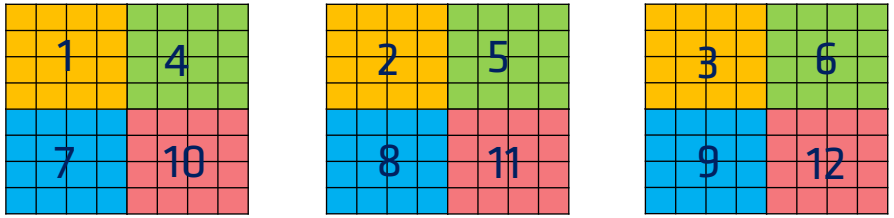
BANDWIDTH REDUCTION: STAYING ON INFINITY CACHE™

- Dispatch work tiling
- Dependent dispatches are tiled and re-issued in depth order
- Tight bounds on data re-use for multiple dispatches

Original 3 dispatches



Cached blocked dispatches (12). Order is illustrative, actual order will maximize overlap and minimize barriers time



- More involved and tricky with dilation of access between dispatches

WORK TILING EXAMPLE: FSR 2

On RDNA™ 3, Infinity Cache™ is smaller, but bandwidth gain is higher.
→ Work Tiling can be even more beneficial for RDNA™ 3

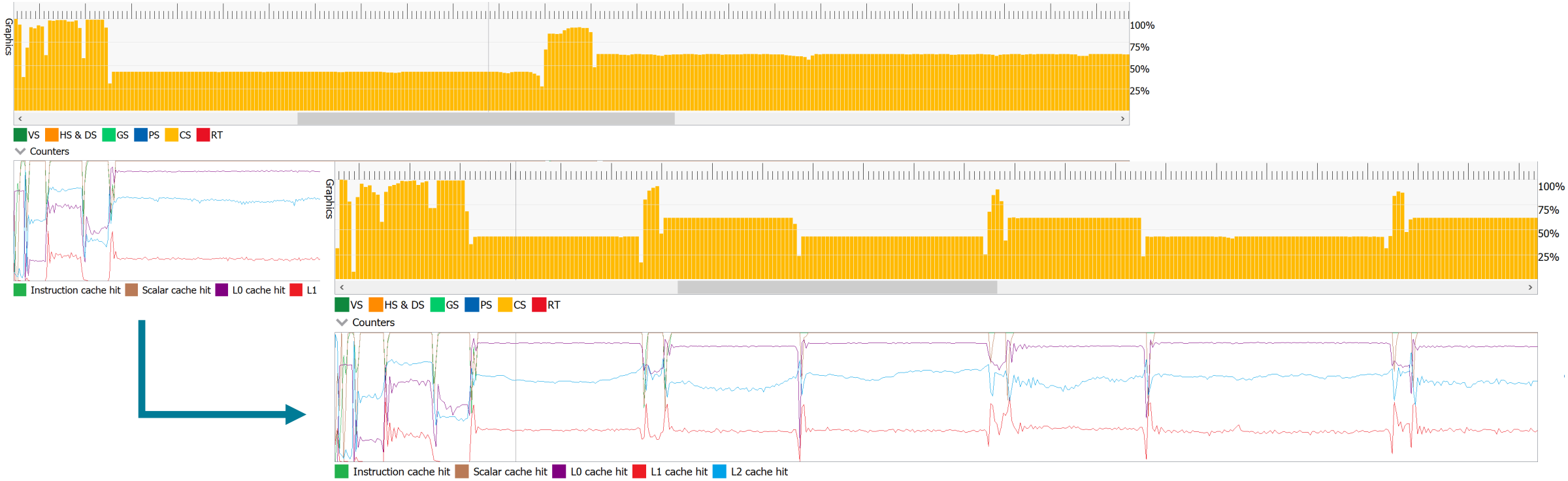
- The FSR 2 algorithm can be very dependent on memory bandwidth
  - Keeping as much data in caches is essential
- For a 4K scene, the amount of data read in a pass outstrips the caches
  - In RDNA™ 2 with AMD Infinity Cache™, there can be some spilling

- To help alleviate this, compute dispatches can be split into multiple workloads, ensuring sampling operations remain within a generally known window
  - This can increase cache hit rates, and therefore allow for faster execution of the workload
  - Similar to how tiled GPU architectures operate, but manually implemented in software

Single whole-screen dispatch

Three dispatches of 1/3 screen y area each

WORK TILING EXAMPLE: FSR 2

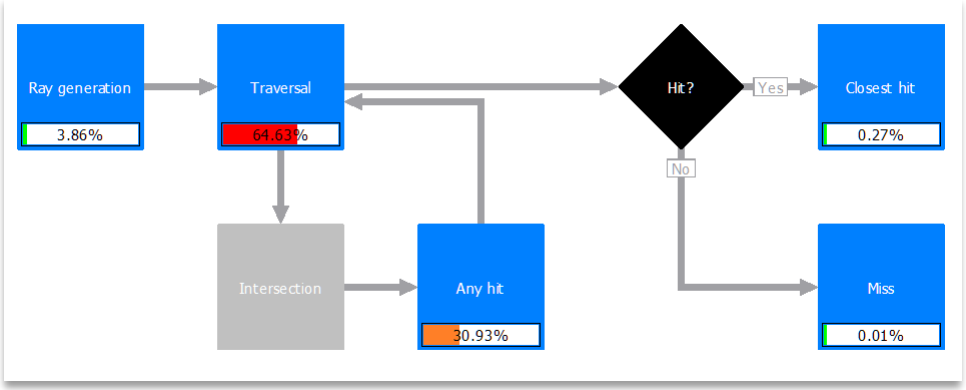


- L0 cache hit rates increased 37% on AMD Radeon™ RX 6800XT when splitting workloads into 3 blocks

RDNA™ 3 RAY TRACING

- RDNA™ 2/3 ray tracing implemented with shader-based traversal and intersection accelerator
- RDNA™ 3 RT architectural changes are evolutionary
- Some improvements specifically done for traversal

→ Proved to be a common bottleneck in RT workloads



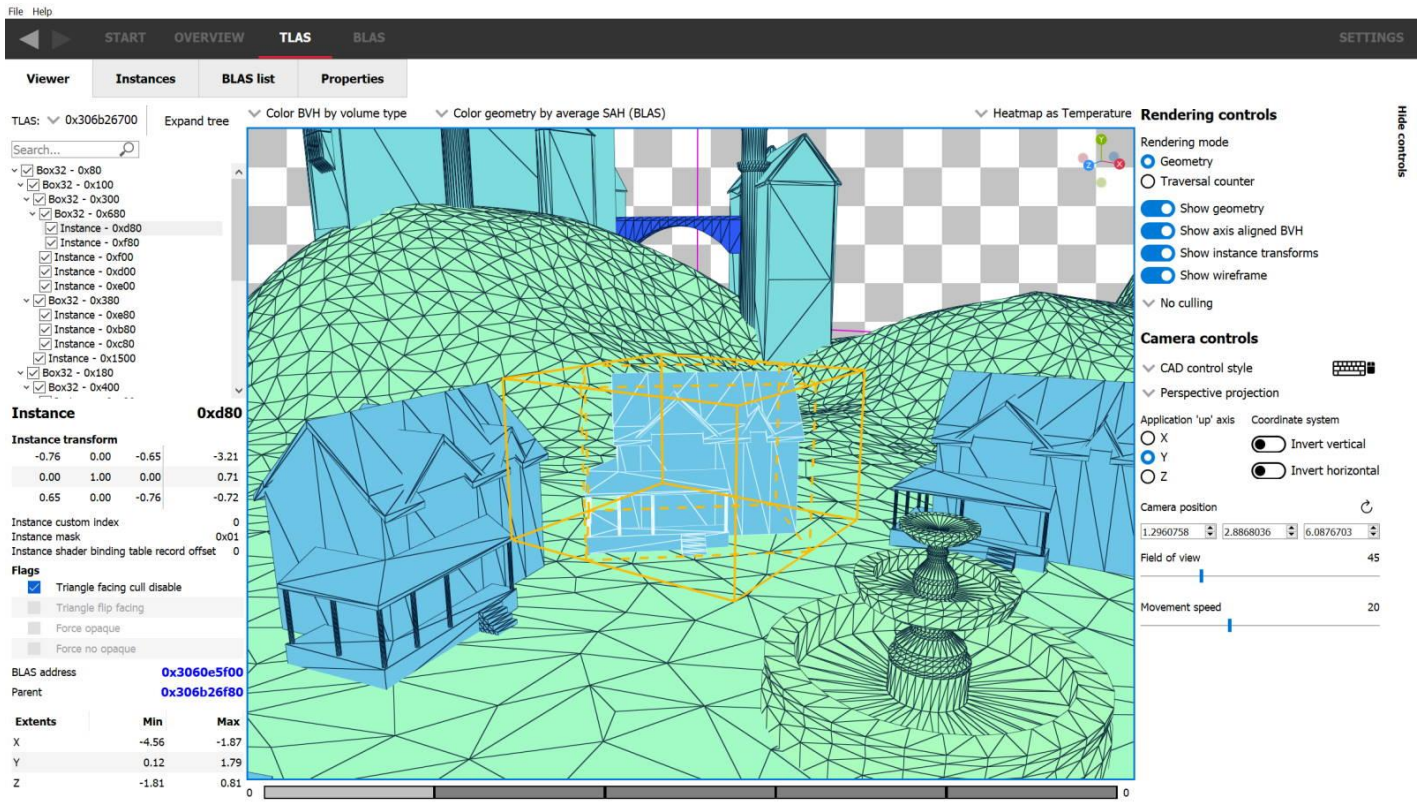
RGP screenshot: workload distribution of a RT dispatch

- Fastest effect is shadow rays with `ACCEPT_FIRST_HIT_AND_END_SEARCH` flag

- It remains important that the application optimizes for fast traversal as well
- RDNA™ 2 optimizations are applicable to RDNA™ 3 and vice versa

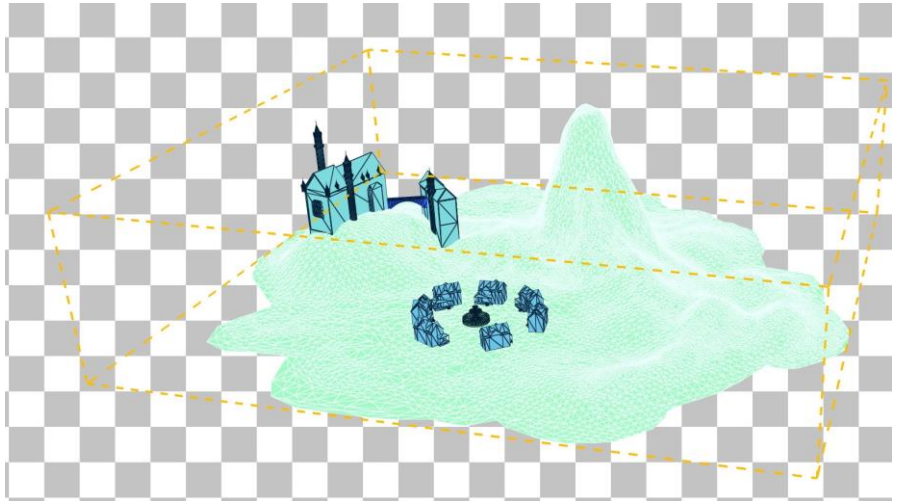
RAY TRACING: REDUCING TRAVERSAL COSTS

- Traversal cost is highly influenced by BVH quality
- To inspect the BVH, you can use Radeon™ Raytracing Analyzer (RRA)

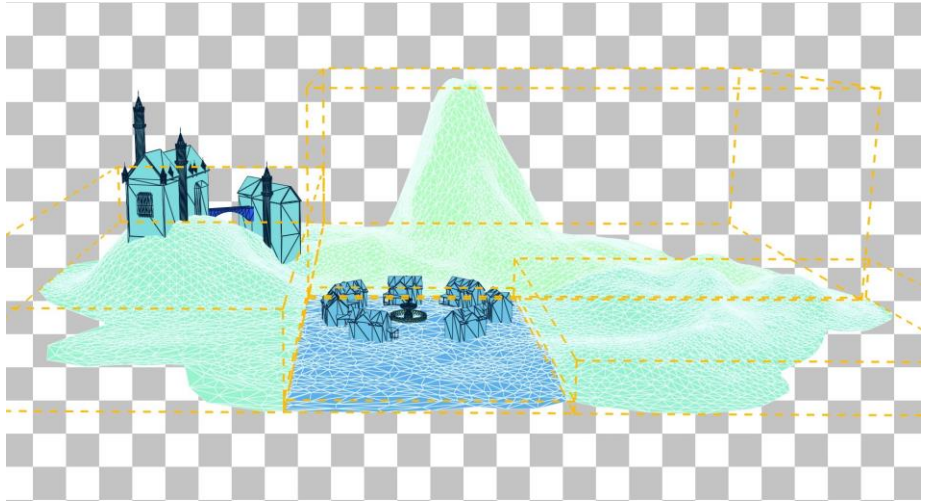


RAY TRACING: IMPROVE BVH QUALITY

- Minimize instance overlap and Minimize empty space
  - Reducing instance overlap gives the driver opportunity to make more optimal acceleration structures
  - A simple trick to reduce instance overlap is to split the terrain into chunks:



One big BLAS for the whole terrain



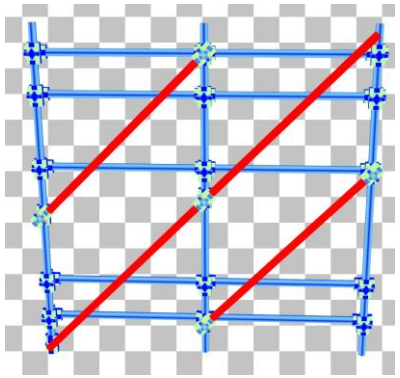
Terrain split into chunks

  - There is a trade-off between tighter fit BLASes and longer TLAS build time due to more instances

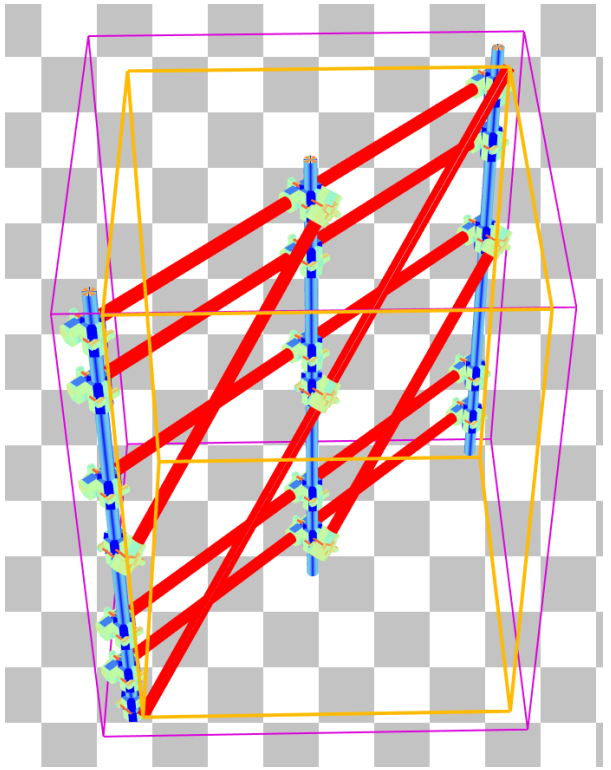
→ Always measure ☺

RAY TRACING: IMPROVE BVH QUALITY

- Axis align triangles in local (BLAS) space
  - Irrelevant for rasterization pipeline
  - Relevant in ray tracing pipeline!
  - → BVH building methods use axis aligned bounding boxes (AABBs)



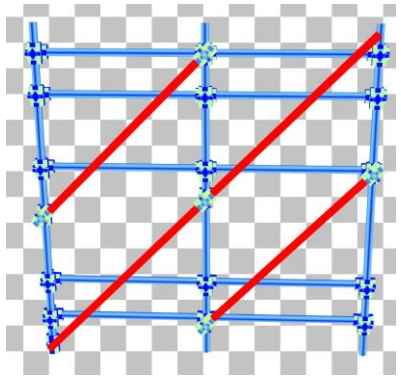
Axis-aligned. Much tighter BLAS ☺



Not axis-aligned. Rotated by 45°!

- You only win ☺ no tradeoff!

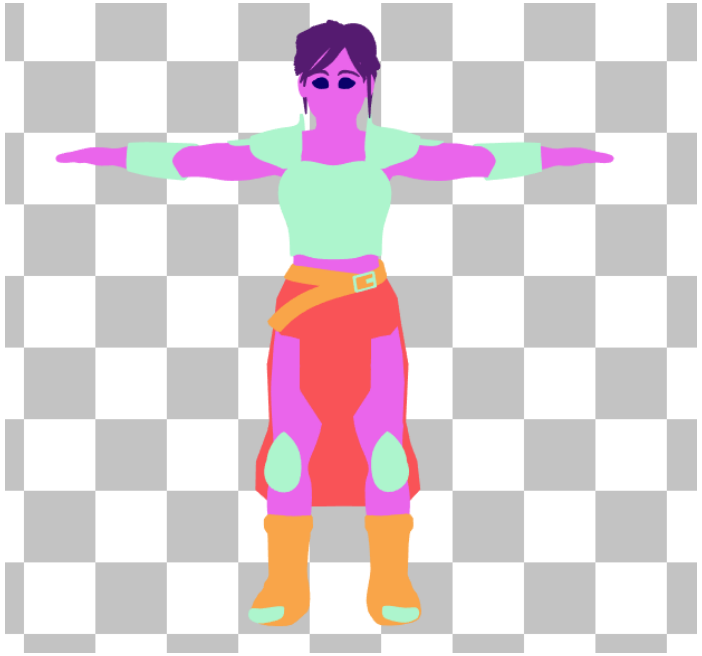
RAY TRACING: IMPROVE BVH QUALITY



- The red diagonal bars are still problematic
- You can split the geometry and use separate BLASes (like the terrain)
- But then again: trade-off with TLAS build time → **Measure!**

- Don't split your geometry by material!
- Each color indicates a unique instance:
  - It shows how the mesh is split up into one BLAS per material

  → leads to significant instance overlap ☹



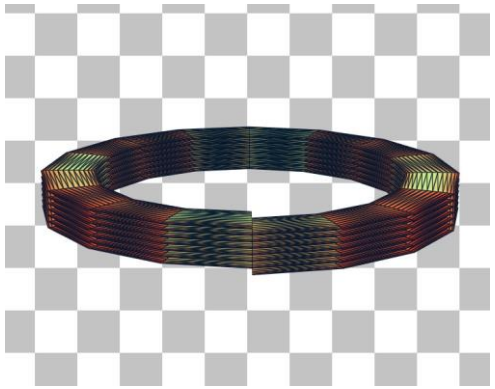
- Use one BLAS instead, but one geometry index per material

RAY TRACING: IMPROVE BVH QUALITY

- We talked about rotation of geometry in local space (axis-aligned)
- What about scaling?
  - BLASes are built relative to the non-deformed mesh
  - Instance transforms can significantly stretch or skew the underlying BLAS

- Example: Same mesh, but different scaling on the y-axis
- Question: Which one is better to use for your BLAS?
- Answer: The one with the smaller instance deformation

→ Avoid large deformations of instances in global space



Mesh vA

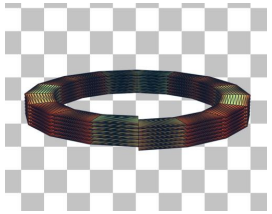
Mesh vB

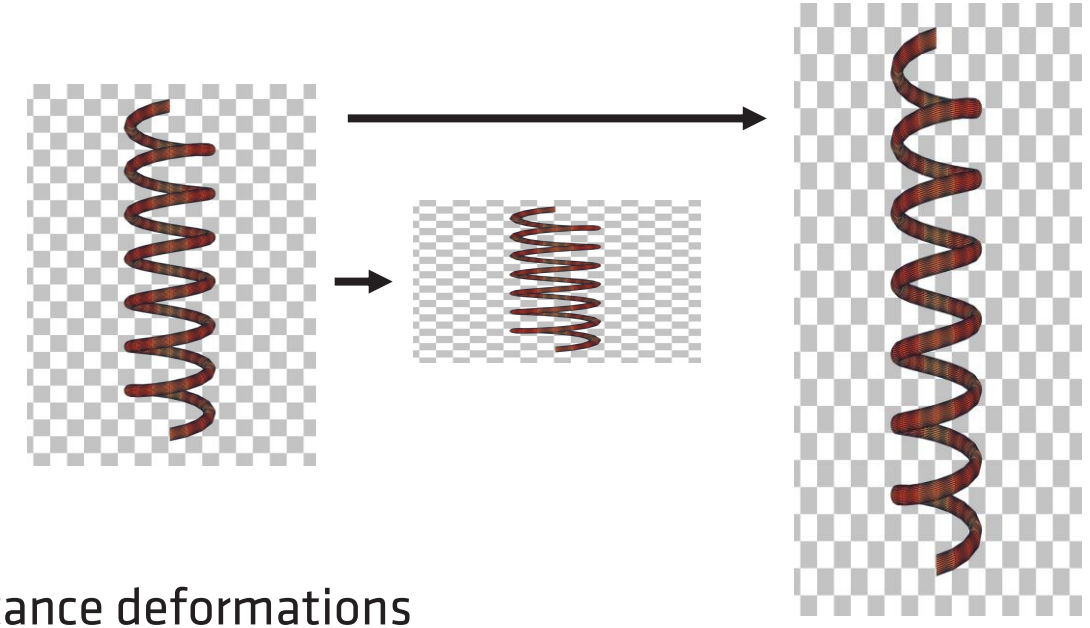
- If your mesh represents a spring, build the BLAS also on the spring and not on a ring 😉

RAY TRACING: IMPROVE BVH QUALITY

- Build the BLAS from scratch or use an existing source BLAS to construct the new BLAS (update)
- Using update can result in faster BLAS build times

- E.g., the spring could be pushed or pulled
- Or even become the ring ☺



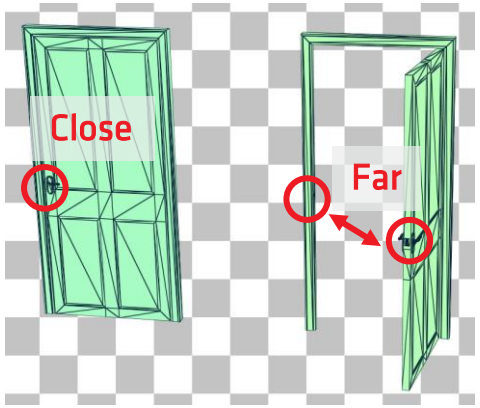


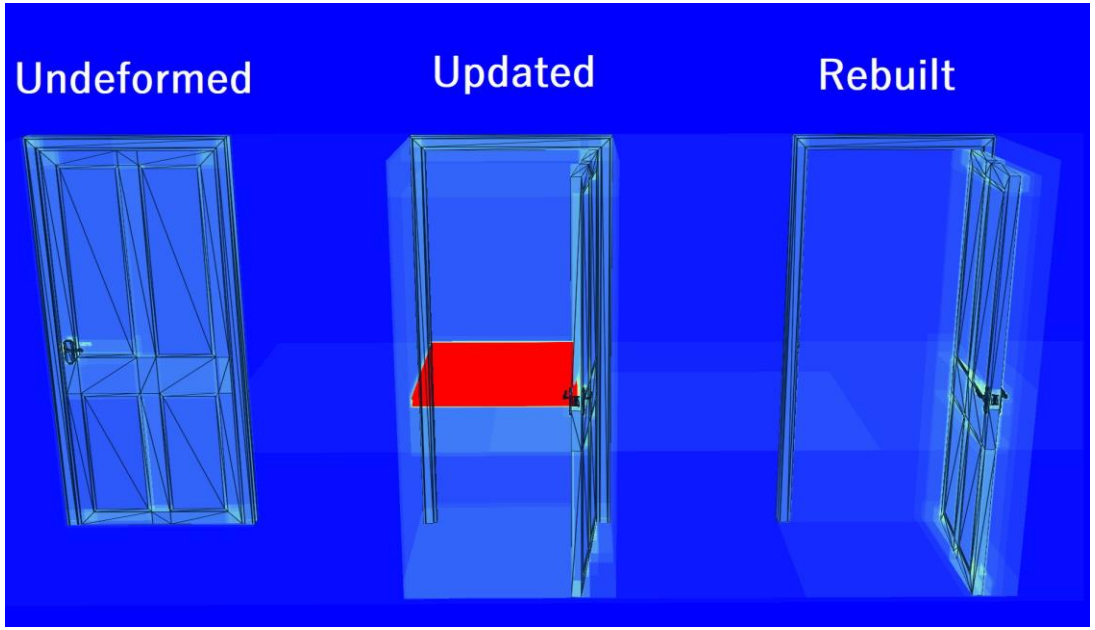
- You can update the BLAS instead of using large instance deformations

RAY TRACING: IMPROVE BVH QUALITY

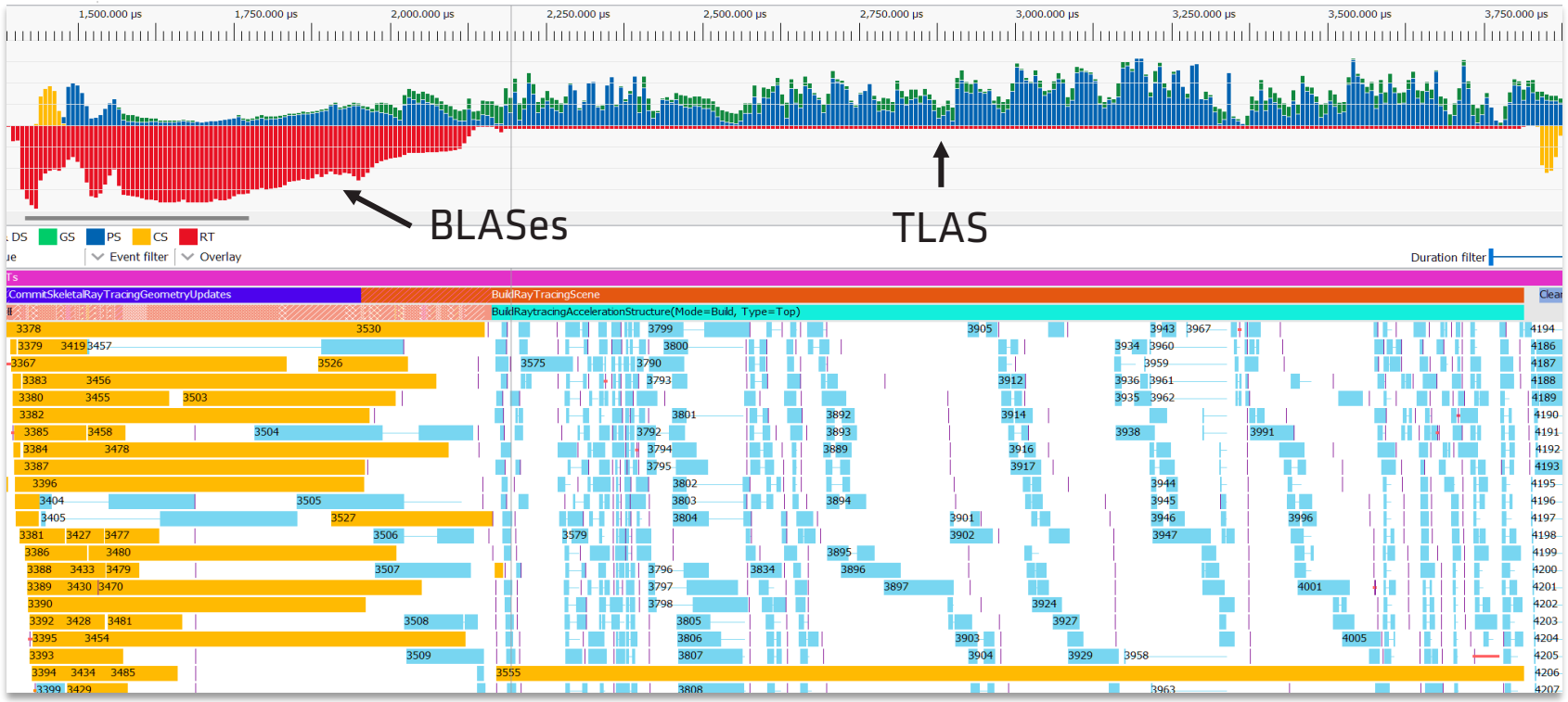
- Be careful to use update with “tearing” deformations
- Door latch and strike (metal piece screwed to the frame) are very close to one another
- Deformation tears them far apart ☹

→ Suboptimal BLAS





RAY TRACING: BVH BUILD



- A last note on BVH builds because they are expensive ☹
  - Minimize rebuilds from scratch
  - Only rebuild or update the LODs as needed
- Rebuild TLAS every frame and run them as async workloads if possible

RAY TRACING: REDUCING TRAVERSAL COSTS

- Not only does BVH quality affect traversal costs, but also which traversal flags you use
  - `ACCEPT_FIRST_HIT_AND_END_SEARCH`
  - `RAY_FLAG_FORCE_OPAQUE` eliminates calls to AnyHit shaders which are expensive
  - Avoid face culling: they **hurt** ray traversal performance 😱
    → Culled triangles are still tested against, but not considered a candidate for closest hit

- More information can be found here: https://gpuopen.com/learn/improving-rt-perf-with-rra/
- In fact, most of the content from this and the previous slides was stolen from this blogpost ☺

Improving raytracing performance with the Radeon™ Raytracing Analyzer (RRA)

David DiGioia

Originally posted September 15, 2022

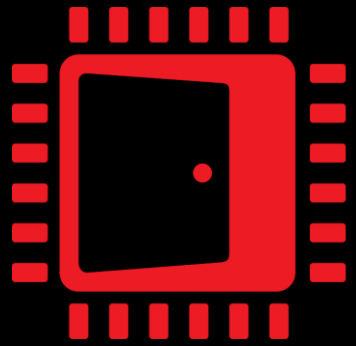
IN CLOSING

- As GPUs become bigger, they become more easily starved for work
- Do more work per draw/dispatch submission
  - Continue to sort by expensive state change that roll contexts
  - Aggregate work and reduce barriers that serialize small workloads
- ExecuteIndirect (with caveats) and submit as many draws/dispatches as possible
- Be aware of caches
  - Shape and arrange workloads to maximize effectiveness
  - Leverage compression as much as possible
- Performance tips and best practices https://gpuopen.com/performance/

DISCLAIMER

The information contained herein is for informational purposes only and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale. GD-18

Use of third-party marks / products is for informational purposes only and no endorsement of or by AMD is intended or implied. GD-83



AMD
GPUOpen

AMD
together we advance_

AMD
EPYC

AMD
RYZEN

AMD
RADEON