Vulkanised 2026

The 8th Vulkan Developer Conference
San Diego, USA | February 9-11, 2026

GPU-Driven Rendering in the Quest 2 and 3

Lucas Miguel Antunes da Silva, Devsh Graphics Programming Sp. z O.O.

The Project

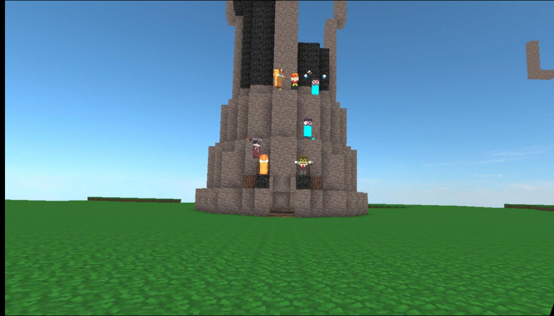
DEVSH GRAPHICS PROGRAMMING SP. z O. O.

- Factions: VR Community based voxel game
- User generated content
- Top-down landscape views are common
  - Worst-case scenario for occlusion culling

The Project

DEVSH GRAPHICS PROGRAMMING SP. Z O. O.

- Quest 2 and 3 game
  - 60% of the playerbase still uses Quest 2, so it needs to be supported
- Future plans for increasing complexity
  - Longer view distance, more players, more complex builds, etc.



Outline

DEVSH GRAPHICS PROGRAMMING SP. z O. O.

- Performance Targets for VR and our game
- Meta Quest 2 and 3 platform recap
- Recap GPU Driven Rendering and Occlusion Culling
- Our solution for low bandwidth tiler GPUs
- Our wishlist for Qualcomm

Time Constraints! [+1], [+2], … indicate topics we expand on in Bonus Slides

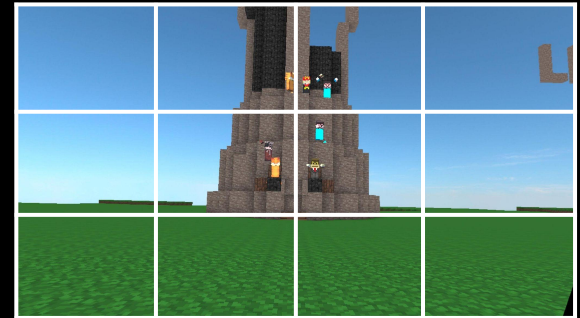
Glossary

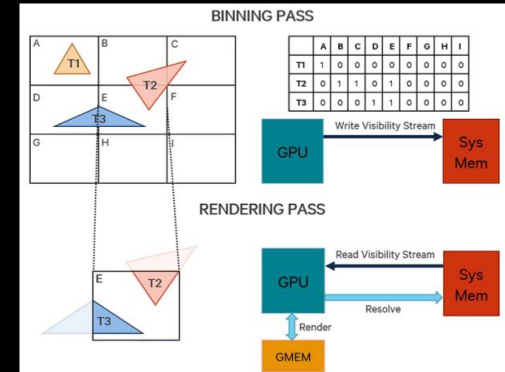
- HZB: Hierarchical Z Buffer, or Depth Pyramid
- LRZ: Low Resolution Z Buffer
- TBDR: Tile based deferred renderer (tiler GPU)
- Model: Collection of drawables
- Drawable: Renderable model with same pipeline and material
  - Equivalent to surface in the Doom Dark Ages FAST AS HELL talk [1]
- Instance: Copy of a drawable in the scene
- Meshlet: Subset of a drawable with fixed amount of geometry, used for culling and/or hierarchical LODs
- Visibility Buffer: Rendering triangles ID and then processing everything in deferred compute [8]

Recap: Mobile GPU

DEVSH GRAPHICS PROGRAMMING SP. z O. O.

- Tiled Based Rendering
- **Binning** : determine and mark all screen tiles each triangle **intersects or covers**
  - Invokes a simplified version of the vertex shader to get the transformed positions
- Rendering: for each screen tile, renders all triangles that are in that bin
  - Invokes the full vertex shader to get all other vertex attributes



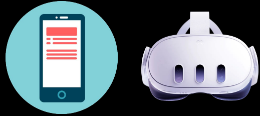


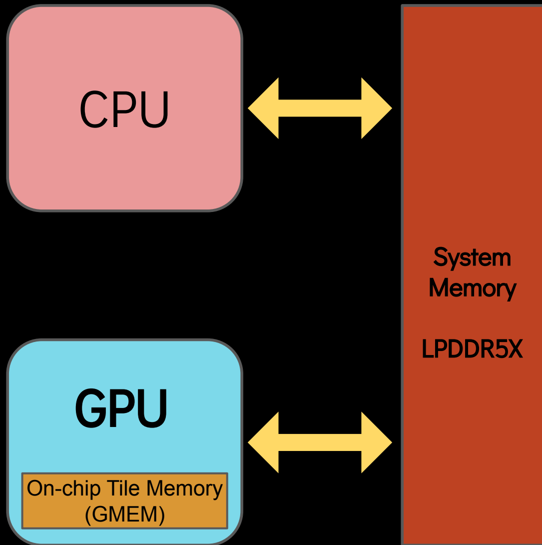
From Qualcomm documentation [14]

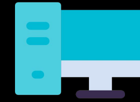
Surface ID 0x745ffb2080: 1680x1760x1, MSAA1x, 32bit color 32bit depth, tile size 864x224, 16 tiles

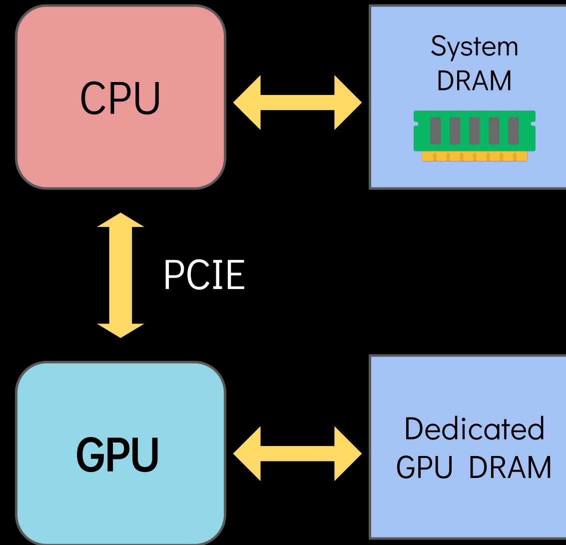
Binning | Bin 0 Render | Bin 1 Render | Bin 2 Render | Bin 3 Render | Bin 4 Render | Bin 5 Render | Bin 6 Render | Bin 7 Render | Bin 8 Render | Bin 9 Render | Bin 10 Ren... | Bin 11 Ren... | Bin 12 Ren... | Bin 13 Ren... | Bin 14 Re... | Bin 15 Ren...

Recap: Mobile GPU

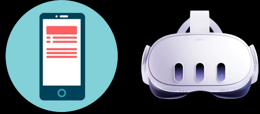


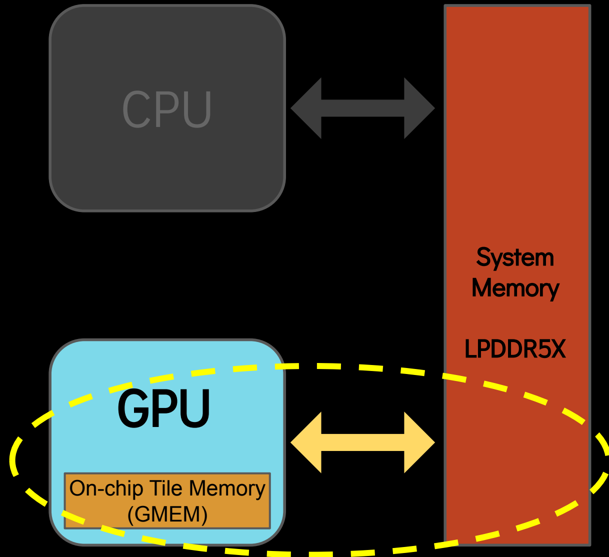






Recap: Mobile GPU





- Process **tiles** of the screen in fast, local **On-Chip Memory (GMEM)** .
- Best Practice: Use less Renderpasses
  - STORE_OP_STORE writes to System Memory
  - LOAD_OP_LOAD reads from System Memory
  - Multiple Renderpasses cause framebuffer **Roundtrip** to/from System Memory
- Subpasses keep data On-Chip.
  - reads from the **GMEM** instead of Sys Mem.
- If the bins are too complex, newer Adrenos will switch from tile to immediate rendering

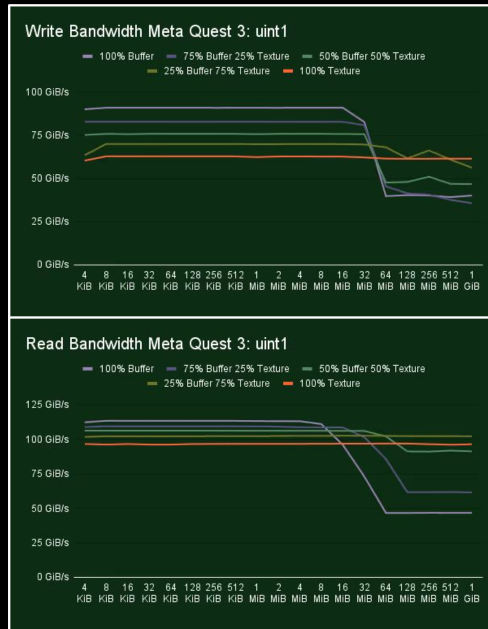
Meta Quest platform

- Quest 2 and 3 have Snapdragon Adreno XR chipsets (600 and 700 architecture)
  - Vulkan 1.3 if Board Support Package updated [2]
  - Non uniform descriptor indexing (*nonUniformEXT*) greatly impacts perf. [+2]
- VK_QCOM_tile_memory_heap allows you to allocate on **GMEM** and specify when they persist across renderpasses
  - Not available on Quest 😕
- Sloooow hardware with lots of expectations
  - 2 eyes, 2064x2208 resolution each on Quest 3
  - Target is around **72-120 FPS** to avoid motion sickness

Meta Quest platform

DEVSH GRAPHICS PROGRAMMING SP. z O. O.

- Quest 3: LPDDR5X memory with peak speeds up to 64GB/s for large resources (128GB/s for small) [7]
  - 72 FPS: Around ~910 MB/frame
  - 4 byte in, 4 byte out: Around ~113 Megapixels/frame
  - For both eyes, ~9 Megapixels spent on a swapchain image
  - Trivial Copy Pass will cost you ~7.9% of frame time (~1.1ms)
- Quest 2: LPDDR4X memory [7]
  - ~51GB/s and 7 Megapixel spent on swapchain image
  - Trivial Copy ~7.7% of frame time (~1.06ms)
- Usually around 1M tris/frame to keep 72 FPS
  - Imposed by Bandwidth spent on attributes and Binning's Visibility Streams
  - Multiview Feature usage is essential!
- Adreno 6xx storage buffer reads are uncached
  - Both Buffers and BDA reads

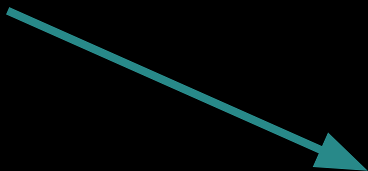


Sources: Evolve Benchmark [10], Native Mixed Reality Compositing on Meta Quest 3 [11], Meta Horizon Developer Page [12]

GPU Driven Rendering

```
vkCmdDrawIndexed(228, 1)
vkCmdDrawIndexed(408, 1)
vkCmdDrawIndexed(1506, 1)
vkCmdDrawIndexed(150, 1)
vkCmdDrawIndexed(1164, 1)
vkCmdDrawIndexed(6, 1)
vkCmdDrawIndexed(168, 1)
vkCmdDrawIndexed(264, 1)
vkCmdDrawIndexed(6, 1)
vkCmdDrawIndexed(270, 1)
vkCmdDrawIndexed(3882, 1)
vkCmdDrawIndexed(6, 1)
vkCmdDrawIndexed(84, 1)
vkCmdDrawIndexed(6, 1)
vkCmdDrawIndexed(192, 1)
vkCmdDrawIndexed(1410, 1)
vkCmdDrawIndexed(2784, 1)
vkCmdDrawIndexed(1290, 1)
vkCmdDrawIndexed(96, 1)
vkCmdDrawIndexed(6, 1)
vkCmdDrawIndexed(174, 1)
vkCmdDrawIndexed(54, 1)
vkCmdDrawIndexed(6, 1)
vkCmdDrawIndexed(102, 1)
vkCmdDrawIndexed(744, 1)
vkCmdDrawIndexed(3180, 1)
vkCmdDrawIndexed(54, 1)
vkCmdDrawIndexed(204, 1)
vkCmdDrawIndexed(1710, 1)
vkCmdDrawIndexed(420, 1)
vkCmdDrawIndexed(6, 1)
vkCmdDrawIndexed(6, 1)
vkCmdDrawIndexed(216, 1)
vkCmdDrawIndexed(846, 1)
vkCmdDrawIndexed(1716, 1)
vkCmdDrawIndexed(2676, 1)
```

- Reduce CPU work spent on rendering;
  - Quest's CPU already used up by other things
- Allow more drawables;
  - Especially cheaper low-poly draws
- Perform culling based on rendering without CPU readback.



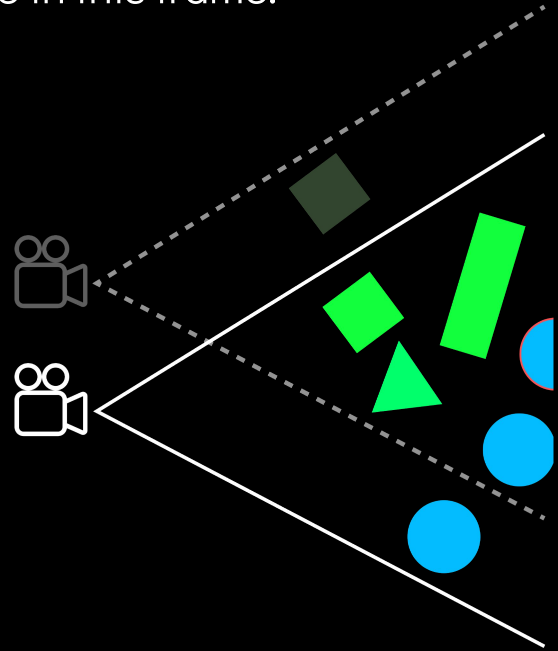
```
Indirect sub-command({ 330300, 1 })
vkCmdDrawIndexedIndirect(1) => <330300, 1>
```

GPU Driven Rendering Flavours

- Without `EXT_device_generated_commands`, fixed number of draw commands must still be recorded into Command Buffers ahead of time.
- Draw indirect (DI) - `vkCmdDrawIndexedIndirect`
  - Arguments (offsets, index buffers) to one draw command sourced from a Buffer
  - Indirect Index Buffer (like DOOM Eternal [6]).
- Multi-draw indirect (MDI) - `vkCmdDrawIndexedIndirectCount`
  - Slow on Adreno 600 [+2]
  - Same as DI but array of arguments is sourced, max draw count is recorded by CPU
  - Draw Count sourced from another buffer, must be less or equal to recorded Max
  - Ergo, GPU can compact and remove invalid draws from array
- Mesh shaders - `VK_EXT_mesh_shader` `vkCmdDrawMeshTasksIndirectEXT`
  - Not available in our target hardware.
  - Conceptually similar to MDI with tiny drawcalls, especially when task shader is used

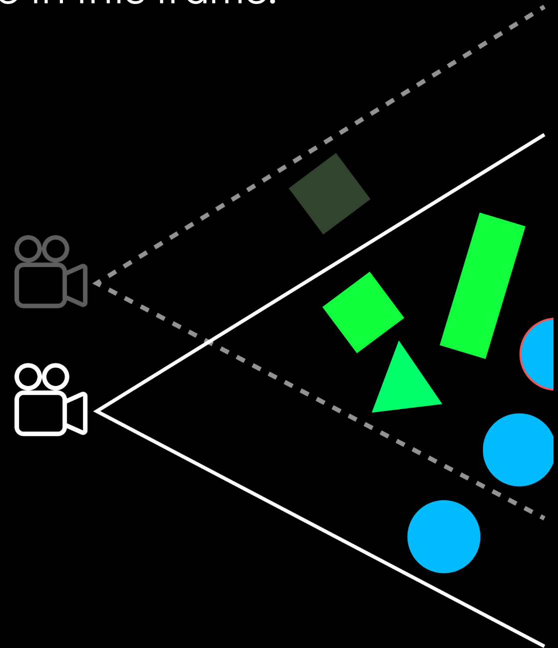
Two-pass Culling Scheme

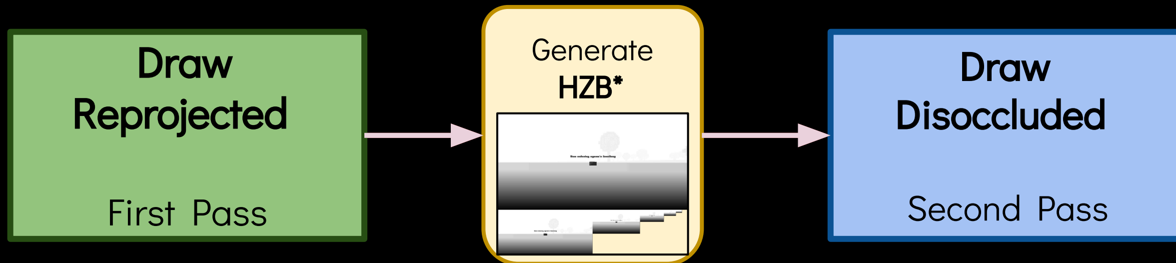
- Assumption: Most objects visible last frame remain visible in this frame.



Two-pass Culling Scheme

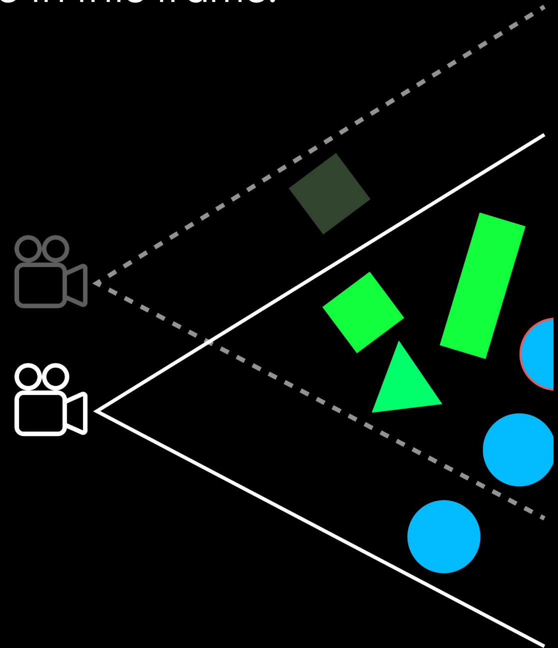
- Assumption: Most objects visible last frame remain visible in this frame.
- First Pass: Draw Reprojected
  - Visible in previous frame, currently in frustum
- Second Pass: Draw Disoccluded
  - NOT Visible in previous frame, currently in frustum
    - Tested for occlusion using HZB

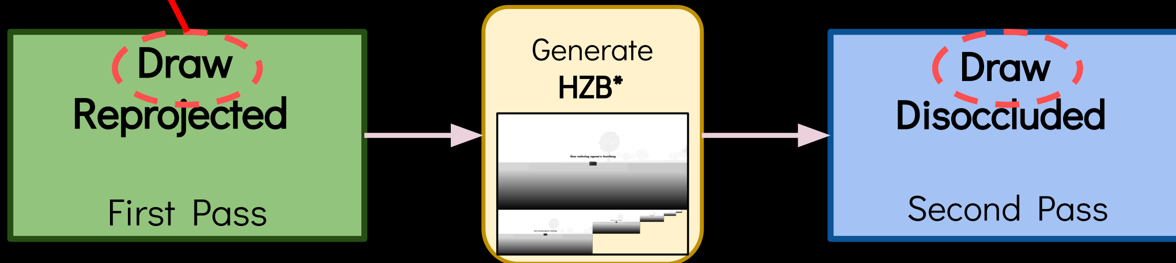




Two-pass Culling Scheme

- Assumption: Most objects visible last frame remain visible in this frame.
- First Pass: Draw **Reprojected**
  - Visible in previous frame, currently in frustum
- Second Pass: Draw **Disoccluded**
  - NOT Visible in previous frame, currently in frustum
    - Tested for occlusion using HZB
- **Draw** = write to Z-Buffer + Additional Data
  - For example, GBuffer or Visibility Buffer





Two-pass Culling Scheme - Limitations on Mobile

DEVSH GRAPHICS PROGRAMMING SP. z O. O.

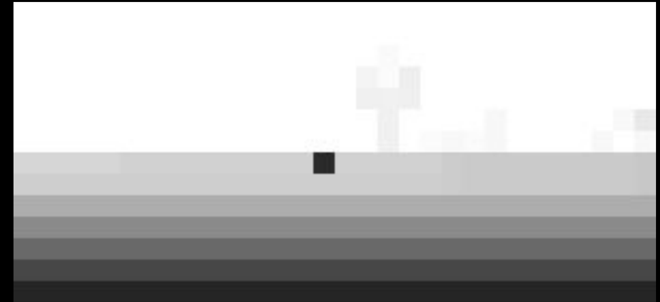
- Two renderpasses around a dispatch
  - The DEPTH32F attachment **round trip** to "resume a renderpass" costs > 0.5 ms!
- Tile Shading only introduced in Adreno 800
  - Wouldn't help here as we'd need to perform the instance culling per-tile
  - Different subsequent indirect draw per tile
- Full screen 8 byte in 4 byte out passes cost almost 1 ms due to BW constraints
  - Without HUAWEI_subpass_shading / QCOM_tile_shading, must shade in a subpass
  - Need groupshared memory to eliminate duplicate pixel-perfect visibility votes
  - Full Screen Triangle has no set mapping of gl_FragCoord to gl_SubgroupIndex [+1]
    - Can't downsample depth more than 2x2 in a subpass with a subgroup op

Our Predicament

- Per Triangle Culling counterproductive, duplicates HW Binning work, esp. BW
- Visibility Buffer impractical on Adreno, esp the 600 series
  - Barycentric interpolation in Fragment Shader
    - Buffer reads are uncached, every pixel pays for fetching 3 vertices' attributes
  - Deferred and Bindless Texturing
    - Slow descriptor indexing, guaranteed > 2-way divergence in subgroup [+2]
- Full resolution Bounding Volume (OBB) draw & ballot is more efficient than than HZB [3]
  - But... Fragment -> Indirect subpass dependencies disallowed, still requires two-pass

Low Resolution Z buffer

- Adreno uses a Low-Resolution Z-buffer (LRZ) during binning
  - Used to reject triangles against partially filled LRZ during binning
  - Rejects full-res pixels as well before doing a full-resolution raster with z-testing
  - Done before Early Z in the framebuffer stages
- We emulate it for our own culling
  - No API to access the existing Adreno LRZ pass
  - Adreno documentation is rife with all sorts of special cases which disable LRZ for a draw or remainder of draws [13]
    - Splitting a renderpass in two would discards the LRZ from the first one



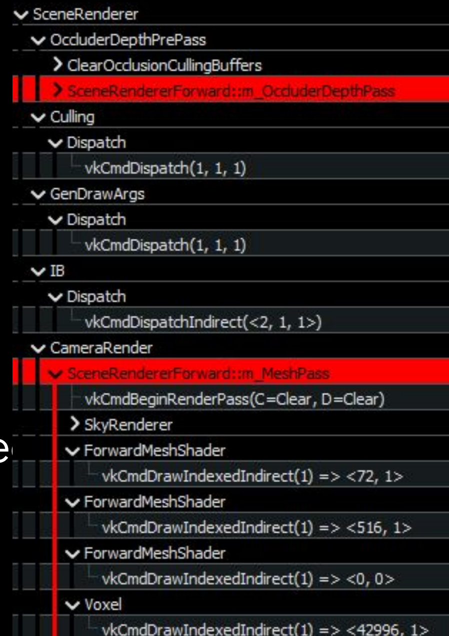
How we made it work on the Quest

- “First do no Harm” -> minimize fixed/overhead costs
  - Around 184 µs on an empty scene
- Draw “occluder” geometry into a low-resolution Z buffer
  - Why not downsample with subgroup instead? [+1]
- In the same render pass, draw bounding boxes for each drawable instance we query visibility of
  - Pixel shader with early depth testing writes ballot
- Discard the low-resolution Z buffer to save bandwidth [+]
  - Not if want to perform small OBB tests in compute

How we made it work on the Quest

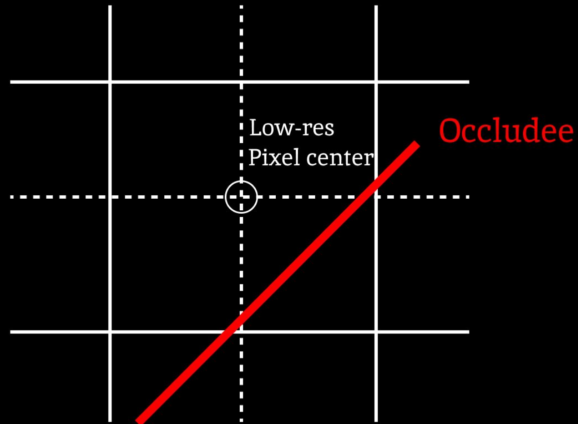
DEVSH GRAPHICS PROGRAMMING SP. Z O. O.

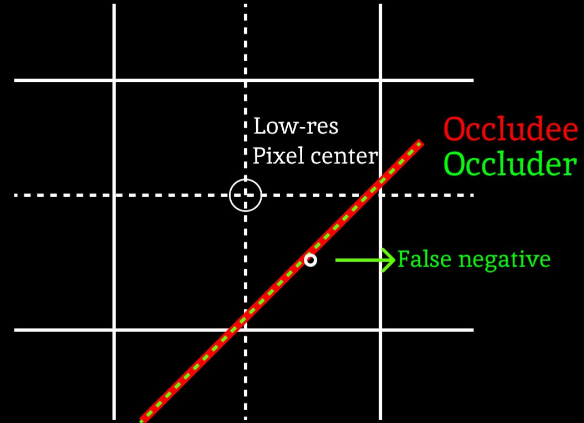
- Frustum Cull on CPU - not enough instances for compute
- Occluders render pass (¼ or ⅛ resolution *per axis*)
  - Draw occluders
  - Draw occludees
- Dispatch read occludee render results produce compact culled drawable list
- Dispatch write indirect draw index offsets and counts for e pipeline
- Indirect dispatch generate indirect index buffer
- Render pass
  - For each pipeline used: Single Indirect Draw with arguments generated above



LRZ Edge cases

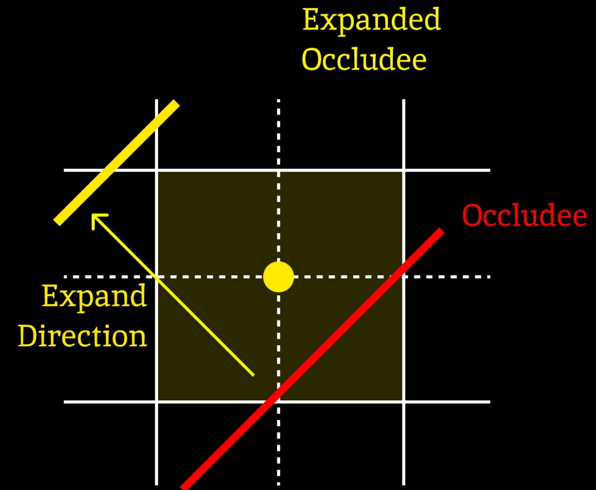
- Rasterization considers only the pixel center when determining visibility



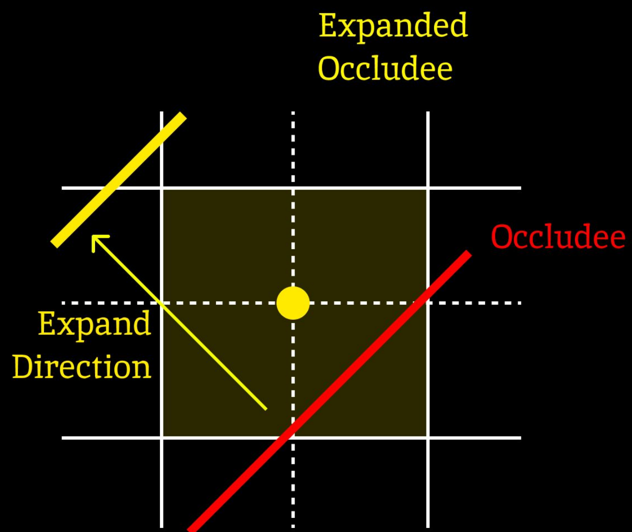


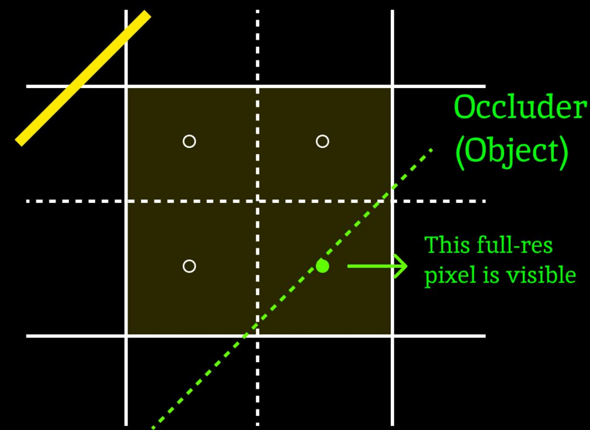
LRZ Edge cases

- Conservative rasterization extensions not supported on Quest 2
  - Even with them, we can get gaps between pixels
    - One pixel gap can turn your instance visible!
- Expand the geometry's vertices
  - Occludee gets expanded one extra low-resolution pixel, to check against occluders;
  - We can also erode occluders by one pixel.
- Expand direction
  - Voxel geometry normals;
  - Regular meshes converted to convex hulls with Jolt
    - Treat all verts as shared, smooth normal is the expansion direction



LRZ Edge cases





Inspired by “Inner Conservative Rasterization” by Marcus Svensson [5]

DEVSH GRAPHICS PROGRAMMING SP. z O. O.

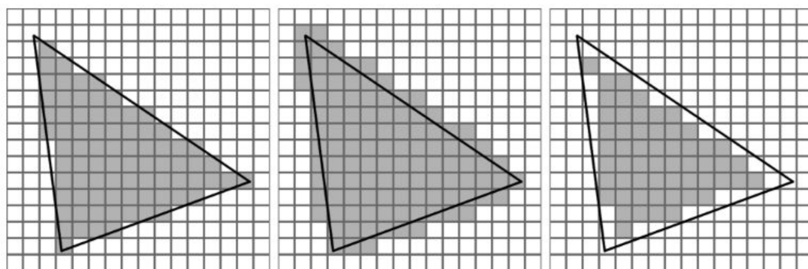


Figure 2.2: *To the left, a triangle rendered using standard rasterization. In the middle, a triangle rendered using outer conservative rasterization. To the right, a triangle rendered using inner conservative rasterization.*

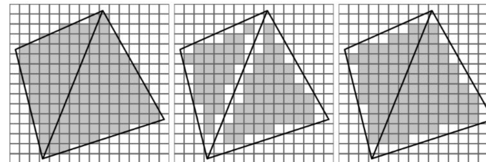


Figure 3.2: *To the left, two triangles are rendered using standard rasterization. Notice how some pixels are partially outside the outline of the triangles. In the middle, the same triangles are rendered using inner conservative rasterization on a primitive basis. Now all pixels are within the outline of the triangles, but the crack running between them is making them lose connectivity. To the right, the triangles are rendered by applying inner conservative rasterization on the silhouette edges and standard rasterization on the remaining edges. All pixels are within the outline of the triangles without losing connectivity. This is the result the algorithm attempts to achieve.*

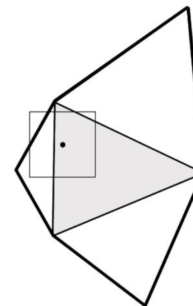
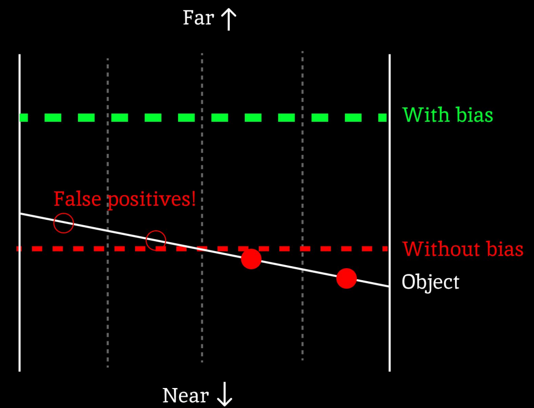


Figure 3.3: *An illustration of a problem with applying inner conservative rasterization on a primitive basis. Assume that inner conservative rasterization is only applied to the silhouette edges and that standard rasterization is used for the rest. The pixel is partially outside the mesh and should not be rasterized. However, as the pixel is covered by a primitive that is not part of the silhouette, it is still included.*

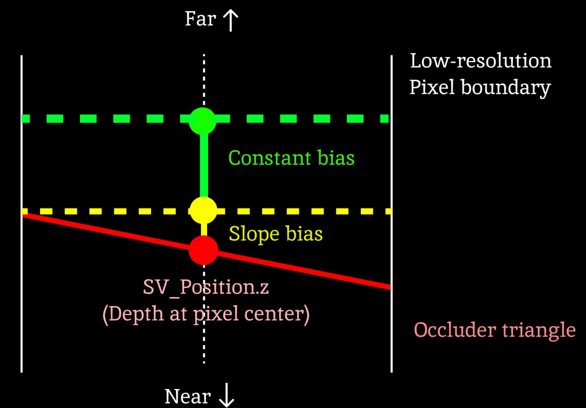
LRZ Edge cases - Depth Edition

- Depth value for the LRZ pixel is at the center
- Biases prevent high resolution rasterized object to go behind pixel center of low resolution occluder



LRZ Edge cases - Depth Edition

- If you modify `gl_FragDepth`, you lose Early-Z!
- Use slope bias in the graphics pipeline
  `VkPipelineRasterizationStateCreateInfo.depthBiasSlopeFactor`
  - This returns the maximum depth buffer value for the triangle in the pixel
  - Multiplier of `fwidth(gl_FragDepth)`
- Use constant bias as well
  `VkPipelineRasterizationStateCreateInfo.depthBiasConstantFactor`
  - Spec has it as a constant for a whole triangle, nasty implications for DEPTH_32F formats - 1 ULP of depth of farthest vertex from camera
    - Need to use DEPTH24 or DEPTH16 unorm



Occlusion test Oriented Bounding Boxes

- Simple draw command
  `gl_VertexID`: Cube vertex; `gl_InstanceID`: Entity ID
- Balloting the visibility in the fragment shader
  - Bitfield atomic OR

```
uint valueIndex = entity / 32;
uint valueOffset = entity % 32;
InterlockedOr(EntityVisibility[valueIndex], 1u << valueOffset);
```

  - Entire value offsets in the Buffer

```
EntityVisibility[entity] = 1;
```

  - We tried this with uint32, 16, 8 and atomic OR, found no performance difference
- For OBBs smaller than 2x2 LRZ pixels, can skip rasterizing and instead of reading the ballot in compute, perform a HZB-like test with 2x2 from mip 0
- NVIDIA: `VK_NV_representative_fragment_test` reduces fragment shader invocations

Nabla Chad Append-Scan (not TM)

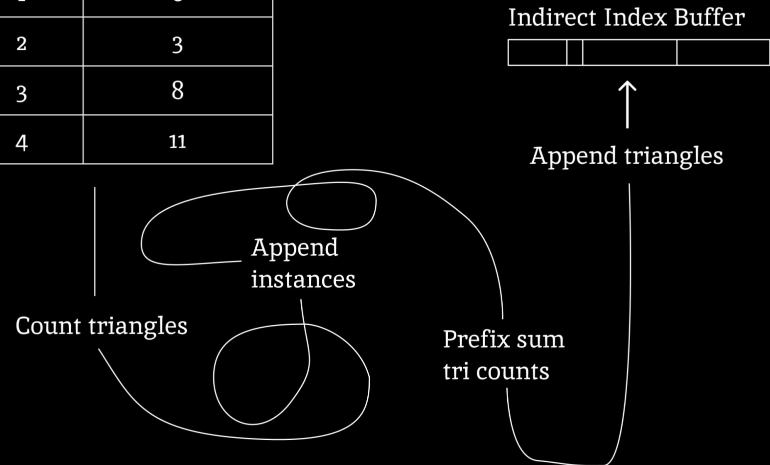
- Varying number of entities spawning varying number of triangles

Instance	Triangle Count
0	6
1	6
2	3
3	8
4	11

Nabla Chad Append-Scan (not TM)

DEVSH GRAPHICS PROGRAMMING SP. Z O. O.

- How to compact it
  - Onesweep? Prefix sum step?



Instance	Triangle Count
0	6
1	6
2	3
3	8
4	11

Nabla Chad Append-Scan (not TM)

DEVSH GRAPHICS PROGRAMMING SP. Z O. O.

- How to compact it
  - Nabla Chad Append-Scan to the rescue!

Instance	Triangle Count
0	6
1	6
2	3
3	8
4	11

→ Nabla Chad Append Scan →

Indirect Index Buffer

Nabla Chad Append-Scan (not TM)

- Builds upon how you normally append to a list on the GPU
  - Normally: Atomic add value count to counter value, place item on a buffer at previous value
  - We instead reserve some bits for the instance count (1) and some for triangle count
  - Previous value will have instance ID and running sum of triangle counts
  - Write the running sum of triangle count to instance ID to get a histogram buffer
- IB generation shader performs binary search on the histogram of triangle counts
  - Not the vertex shader, it runs twice on tiling GPUs

Counter

Instance Count	4
Triangle Count	34
Value	262178

Histogram

0	1	3	25

Atomic Add

Instance Count	1
Triangle Count	19
Value	65555

Using previous value:
Place value 34
at index 4

Counter

Instance Count	5
Triangle Count	53
Value	327733

(262178 + 65555 = 327733)

Histogram

0	1	3	25	34

Multiple Pipelines

- Each pipeline gets its own indirect draw call argument
    - Contiguous subset of indices in the Global Unified Index Buffer for that call
- Buffer with count of triangles per pipeline
    - Append triangle count for the pipeline ID when doing cull
    - Per instance buffer with the running count of triangles for that instance's pipeline

Pipeline Triangle Counts

0	1	2	3
12	3	0	19

Indirect Index Buffer

Instance Tri Offset In Pipeline

0	6	0	0	8

Pipeline	1
Triangle Count	19

Previous value at pipeline 1

Pipeline Triangle Counts

0	1	2	3
12	22	0	19

3

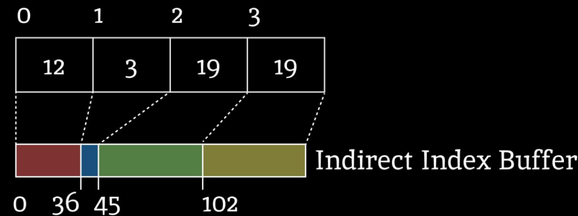
Instance Tri Offset In Pipeline

0	6	0	0	8	3

Multiple Pipelines

DEVSH GRAPHICS PROGRAMMING SP. Z O. O.

- Draw calls are generated
  - Index count: Triangle count * 3
  - Index offset: Prefix sum of previous values in the triangle count per pipeline
- It's a counting sort in disguise!
  - The pipeline ID is the key



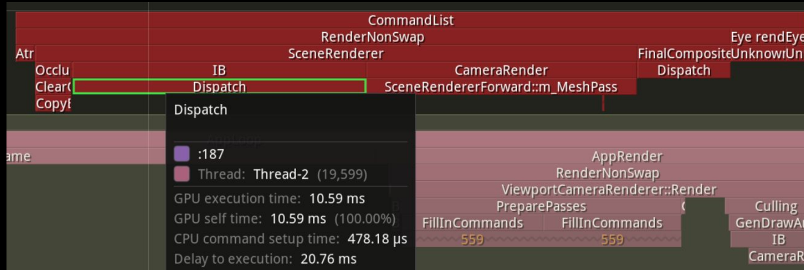
indexCount	instanceCount	firstIndex	vertexOffset	firstInstance
36	1	0	0	0
9	1	36	0	0
57	1	45	0	0
57	1	102	0	0

Practical Considerations

- We join together N triangles in each entity
  - N = 32 for us
  - Reduces number of reads for binary search during IB generation
- Use 32 bit integer to avoid 64 bit atomics
  - Atomic add instance count in 16 MSB and triangle count / N in 16 LSB
  - This only limits VISIBLE occlusion culled instance count to 2^{16}
  - Resulting 16 MSB have the allocated instance ID, and the 16 LSB have the running sum of triangle counts
- Up to $2^{16} * 32 = 2\ 097\ 152$ triangles per entity
  - Quest 3 can only theoretically do 500 000 at target FPS
- No need to pad the index buffer with invalid tris
  - Bundle triangles only in the global scan! Pipelines count in individual triangles
  - Compute invocations go unused to prevent unused vertex invocations

Workgroup Cooperative Binary Search

- Naive version of the instance search takes up to 10ms with ~12k triangles



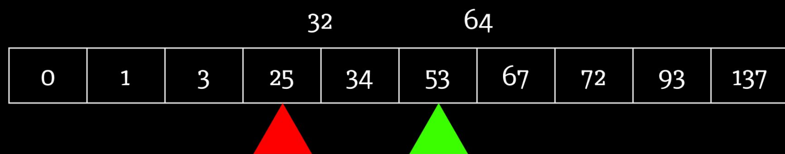
- Around 115 µs after optimizations
- Get our shirt during the break ->

- Things we know about our binary search of the histogram
  - Each drawable instance has one triangle or more;
  - Each invocation of our IB search will evaluate 1 bundle of tris;
  - Because of this, the workgroup size is the upper bound of values our binary search will read.
- We can leverage subgroups and workgroups to perform collaborative work

Workgroup Cooperative Binary Search

- First subgroup in the workgroup searches for the minimum value in the WG
- Second subgroup searches the maximum value
- Load every value between minimum and maximum into groupshared memory
- Each invocation binary searches on the groupshared memory to do a final resolve

Example: WG with invocations 32-64
Search value minimum is 32, maximum is 64
groupSearchBufferMinIndex = 3
groupSearchBufferMaxIndex = 5



shared groupSearchValues

25	34	53

Threads 0-1: (32-33): 0
Threads 2-20: (34-52): 1
Threads 21-31: (53-64): 2

+ groupSearchBufferMinIndex → 3, 4, 5

Workgroup Cooperative Binary Search

DEVSH GRAPHICS PROGRAMMING SP. Z O. O.

- Subgroups can cooperatively search with subgroup intrinsics
  - N-pivot search where N is subgroup size
  - Each invocation in the subgroup searches a value
  - Partition size is reduced by the subgroup size in each iteration
- Subgroup ballot the `searchValue < searchBuffer[pivotIndex]` conditional
- Count ballot bits to find where the partition begins next

`OpGroupNonUniformBallotBitCount`

Subgroup size of 4 (for example)

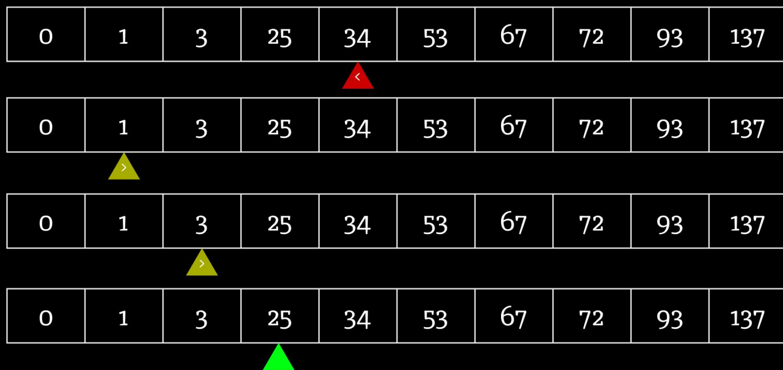
0	1	3	25	34	53	67	72	93	137

0	1	3	25	34	53	67	72	93	137

Workgroup Cooperative Binary Search

Example: Searching value 25

- Regular binary search: $\mathcal{O}(\log n)$ complexity



- Subgroup binary search: $\mathcal{O}(\log_s n)$, where s = subgroup size (32 on NVIDIA and 64 on Quest)

Wave size of 32: We can do this one in a single iteration

- With s = 64, we can do up to 262 144 values in 3 iterations!

Waterfall Looping Optimization

- Only one invocation in a subgroup needs to atomic append in most cases
  - Chad Append-Scan is a prime example
- Prefix sum the subgroup sum of values - `subgroupExclusiveAdd()`
  - On the last invocation in the subgroup - `subgroupBallotFindMSB()`
  - Prefix sum at the last invocation + current value = total sum in the subgroup
  - Last invocation writes total sum and other invocations shuffle that value in
  - Previous value + prefix sum of values at current invocation = current invocation value

```
    uint lastActiveInvocation = spv::subgroupBallotFindMSB(spv::subgroupBallot(active));
    uint valuePrefixSum = spv::subgroupExclusiveAdd(value);
    uint subgroupAppendResult;
    if (spv::gl_SubgroupInvocationID == lastActiveInvocation) {
        subgroupAppendResult = spv::atomicAdd(Counters[0], valuePrefixSum + value);
    }
    uint appendResult = valuePrefixSum + spv::subgroupShuffle(subgroupAppendResult,
lastActiveInvocation);
```

Waterfall Looping Optimization

DEVSH GRAPHICS PROGRAMMING SP. Z O. O.

- But my invocations could all be writing to different places!
  - E.g.: Entity visibility in BBox fragment shader, or `TriangleCountsInPipeline`
- `subgroupBroadcastFirst` returns the value on the first active invocation in the subgroup
  - With this, we can loop through every distinct value within the subgroup;
  - Each operation is executed ONCE, per distinct value in the subgroup.
- Use with `VK_KHR_shader_maximal_reconvergence` [9]
- Sample code in bonus slide! [+2]

Performance numbers (Quest 3)

DEVSH GRAPHICS PROGRAMMING SP. z O. O.

Performance takeaways + future work

DEVSH GRAPHICS PROGRAMMING SP. Z O. O.

- GPU Driven without culling has 18% perf penalty on Quest 2
  - Only 3% on Quest 3
  - We mainly attribute this to our entity ID becoming non-uniform
- Then, adding in culling got us +5% net gain on Quest 3 on a hard scene
  - Outdoors and low render distance, suboptimal for culling
  - This win is despite the 3% penalty from GPU driven
- We should be able to mitigate the Quest 2 regression with
  - Scalarization loop for reading vertex data
  - Or scalarizing the entire vertex shader
  - Scalarizing texture sampling
- Priming the main Renderpass Z Buffer with big Quads at LRZ texel depths

Our wishlist for Qualcomm

- Programmable blending
  - GLES has it for 10 years on the same architecture
- Subpass shading link/reference, like in HUAWEI extension or in Metal
  - Downsampling on-chip could be possible
  - Would really like coherent access for Multiple Layer Alpha Blending OIT
- Access to LRZ memory
  - LRZ access for readback
  - LRZ priming - we have a 100% conservative approximation

Special Thanks to Matt Pettineo a.k.a. MJP

Matt has kindly shared some conclusions from his study of the Quest 2

- Descriptor Reads get proportionally slower to the degree of divergence
- Adreno 6x0 hates Multi Draw Indirect
- SSBO reads are uncached

DEVSH GRAPHICS PROGRAMMING SP. Z O. O.

Want to work side-by side with Experts?

- Had your team member poached by a Stealth Mode AI Startup?
- Your organisation hasn't allocated the resources to replace them?
- Talk about us to your Project Manager
  - Or mention us in your trip report!

Visit our website ->

DEVSH GRAPHICS PROGRAMMING SP. z O. O.

Questions?

Bibliography

DEVSH GRAPHICS PROGRAMMING SP. z O. O.

[1]: FAST AS HELL idTech 8 Global Illumination, Tiago Sousa at SIGGRAPH 2025
[2]: GPU Info report
[3]: OpenGL Scene Rendering Techniques, Christoph Kubisch at GTC and SIGGRAPH 2014
[4]: "Nvidia SPIR-V Compiler Bug or Do Subgroup Shuffle Operations Not Imply Execution Dependency?", Sorakrit Chonwattanagul at our blog
[5]: Occlusion Culling on the GPU: Inner Conservative Occluder Rasterization by Marcus Svensson
[6]: Rendering the Hellscape of DOOM Eternal at SIGGRAPH 2020
[7]: LPDDR Wikipedia Page
[8]: Visibility Buffer Rendering with Material Graphs by Filmic Worlds
[9]: What is Maximal Reconvergence And Why It Matters, Hugo Devillers at Vulkanized 2025
[10]: Learning About GPUs Through Measuring Memory Bandwidth, by Manon Oomen at Evolve Benchmark
[11]: Native Mixed Reality Compositing on Meta Quest 3: A Quantitative Feasibility Study of ARM-Based SoCs and Thermal Headroom, by Muhammad Kaif, Laghari Areeb Ahmed Shaikh, Faiz Khan, and Aafia Gul Siddiqui
[12]: Meta Quest Documentation on Testing and Performance
[13]: https://docs.qualcomm.com/doc/80-78185-2/topic/mobile_best_practices.html#low-resolution-z-pass
[14]: "Moving Mobile Graphics" by Qualcomm

DEVSH GRAPHICS PROGRAMMING SP. z O. O.

Bonus Slides!

[+1] Subgroup Input Attachment Downscale

- Subpass input attachment for downscaling
  - Have a subpass that draws a fullscreen triangle and perform a subgroupMax
    - Both Quests have 64 sized subgroups
  - Write the reduction of 8x8 to a low-res buffer
  - We can then do two-pass culling with the full depth buffer
- Sounds great, why didn't you do this?
  - No guarantee that the subgroup dispatch on an FS tri will be done in 8x8 or aligned
  - Still can't rasterize bounding boxes in the same render pass
  - Or do Compute Shader dispatches with full framebuffer dependencies
  - This means we'd still need to split the main renderpass in two

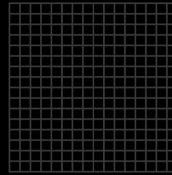
[+2] Scalarize Sample Code

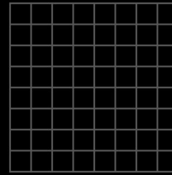
```
template<typename I, typename F>
void scalarize(const I ix, inout F f, bool outstanding)
{
    bool currentInvocationIsActive = any(glsl::gl_SubgroupEqMask()&glsl::subgroupBallot(true));
    scalarize<I,F>(ix,f,currentInvocationIsActive);
    [loop] while (outstanding)
    {
#if !defined(MAXIMAL_RECONVERGENCE)
        glsl::subgroupBarrier();
#endif
        const I scalar = glsl::subgroupBroadcastFirst(ix);
        [branch] if (all(scalar==ix))
        {
            f(scalar);
#if !defined(MAXIMAL_RECONVERGENCE)
            glsl::subgroupBarrier();
#endif
            outstanding = false;
        }
    }
}
```

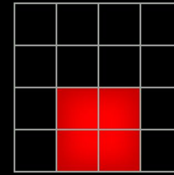
`subgroupBarrier` is a `OpOpControlBarrier`, and we ran into GPU hang on NVIDIA without it [4].

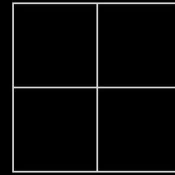
[+3] Hierarchical Z-Buffer

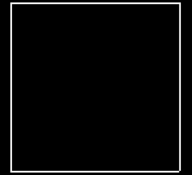
- For each drawable:
  - Sample a 2x2 footprint from the same HZB level;
  - Conservatively covers the object in screenspace.

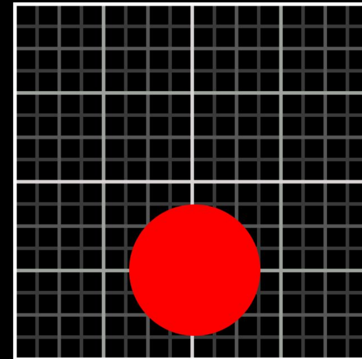






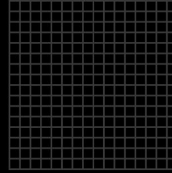


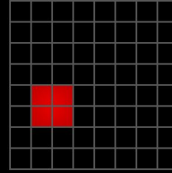


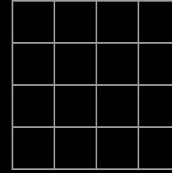


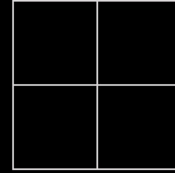
[+3] Hierarchical Z-Buffer

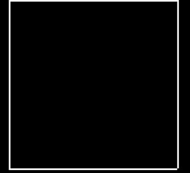
- For each drawable:
  - Sample a 2x2 footprint from the same HZB level;
  - Conservatively covers the object in screenspace.

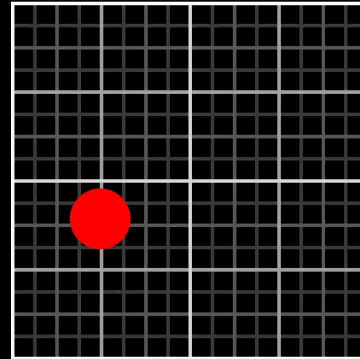






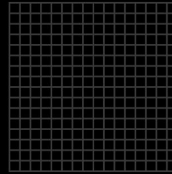


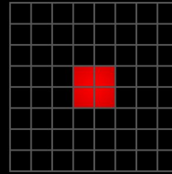


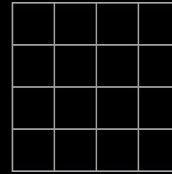


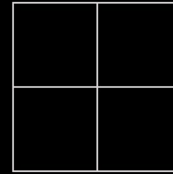
[+3] Hierarchical Z-Buffer

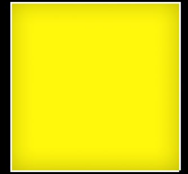
- Why not 1x1 ? Quadtree-like alignment
    - Screen Space AABB, no matter how small, straddling the middle screen lines is always only covered by last pyramid level texel
    - Similar problem for any 2^{-N} aligned coordinate
    - Conservative, false negative galore!


Mip 0

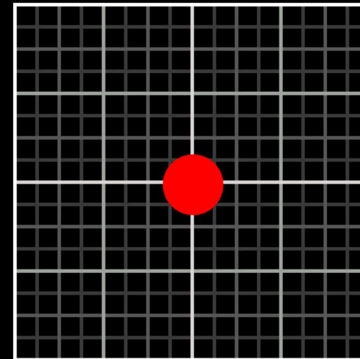

Mip 1


Mip 2


Mip 3

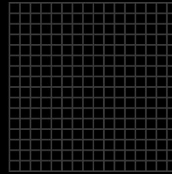

Mip 4

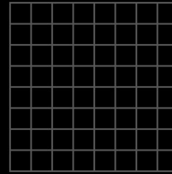
What you would need to sample with a single pixel

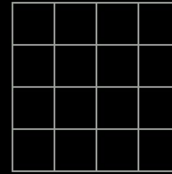


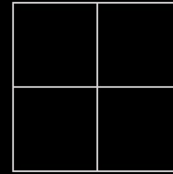
[+3] Hierarchical Z-Buffer

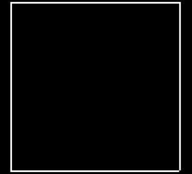
- For each drawable:
  - Sample a 2x2 footprint from the same HZB level;
  - Conservatively covers the object in screenspace.


Mip 0


Mip 1


Mip 2


Mip 3


Mip 4