SIGGRAPH 2026
Los Angeles 19–23 JUL

The Premier Conference & Exhibition on
Computer Graphics & Interactive Techniques

State of GPU Driven Rendering on Mobile

About me

RODRIGO HOLZTRATTNER

Dev Tech Lead at Snapdragon Game Studios, Qualcomm, working directly with game studios on CPU/GPU architecture and optimization

https://www.linkedin.com/in/rodrigoholztrattner/

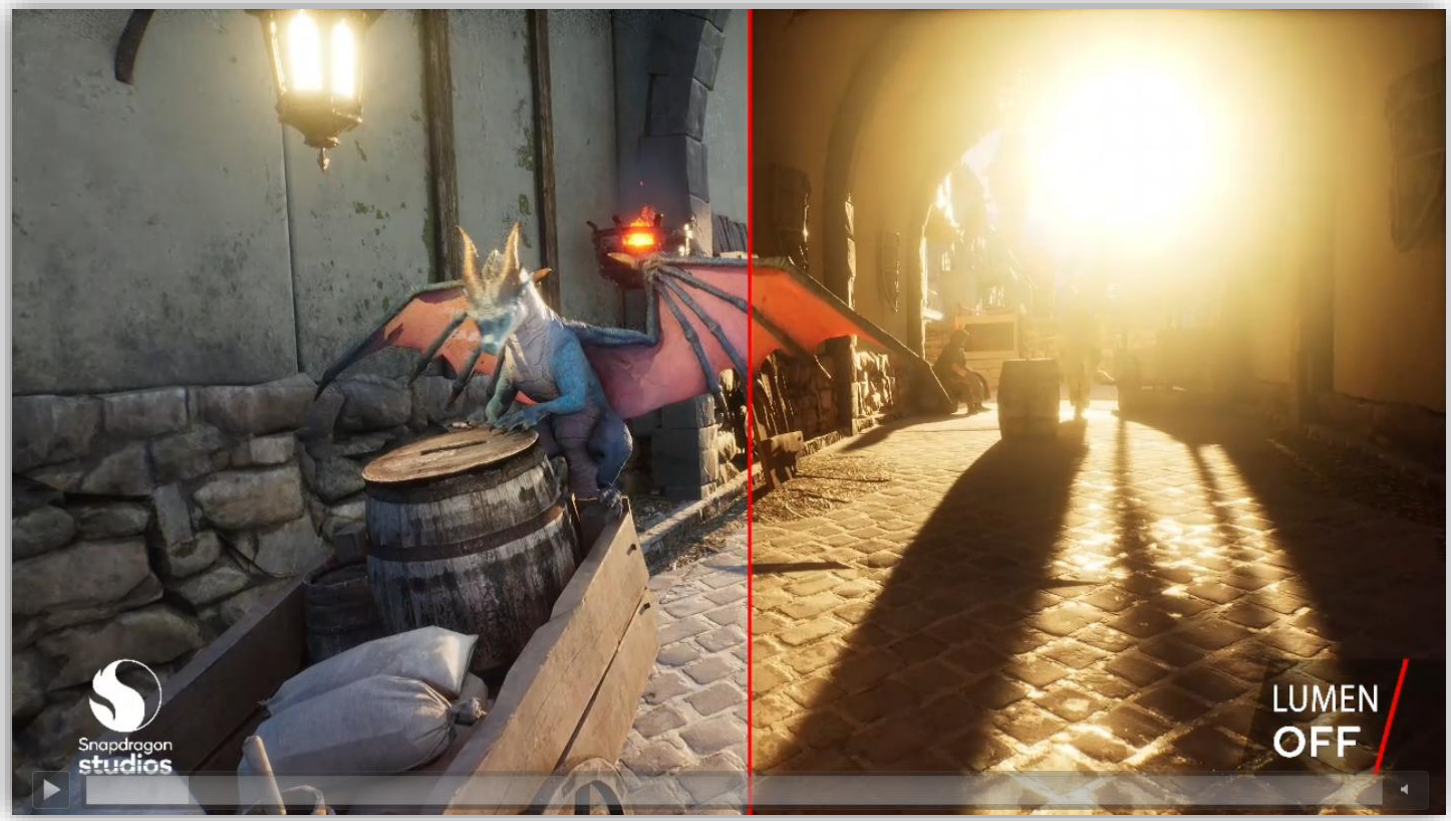
GPU-Driven Spectrum

- The industry often discusses GPU-driven rendering as a single destination, but in practice is an evolving feature.
- Mobile renderers gradually shift responsibilities from the CPU to the GPU.
- The question is not whether an engine is GPU-driven, but how GPU-driven it is.

Level	Label
Level 0	CPU driven/owned
Level 1	GPU assisted
Level 2	GPU visibility + indirect draw
Level 3	Mesh shaders
Level 4	Work graphs

GPU-Driven Spectrum

- Mobile GPU-driven must respect tilers, bandwidth, thermal limits, sync, and API portability. It is not the desktop problem adopted downward.
- The pressure is already here: Nanite, Lumen, clustered lighting, GPU particles, mesh shaders.
- Content scale is growing faster than CPU orchestration can keep up.



Running at realtime - Snapdragon 8 Gen 3

GPU-Driven Spectrum

- Mobile GPU-driven must respect tilers, bandwidth, thermal limits, sync, and API portability. It is not the desktop problem adopted downward.
- The pressure is already here: Nanite, Lumen, clustered lighting, GPU particles, mesh shaders.
- Content scale is growing faster than CPU orchestration can keep up.

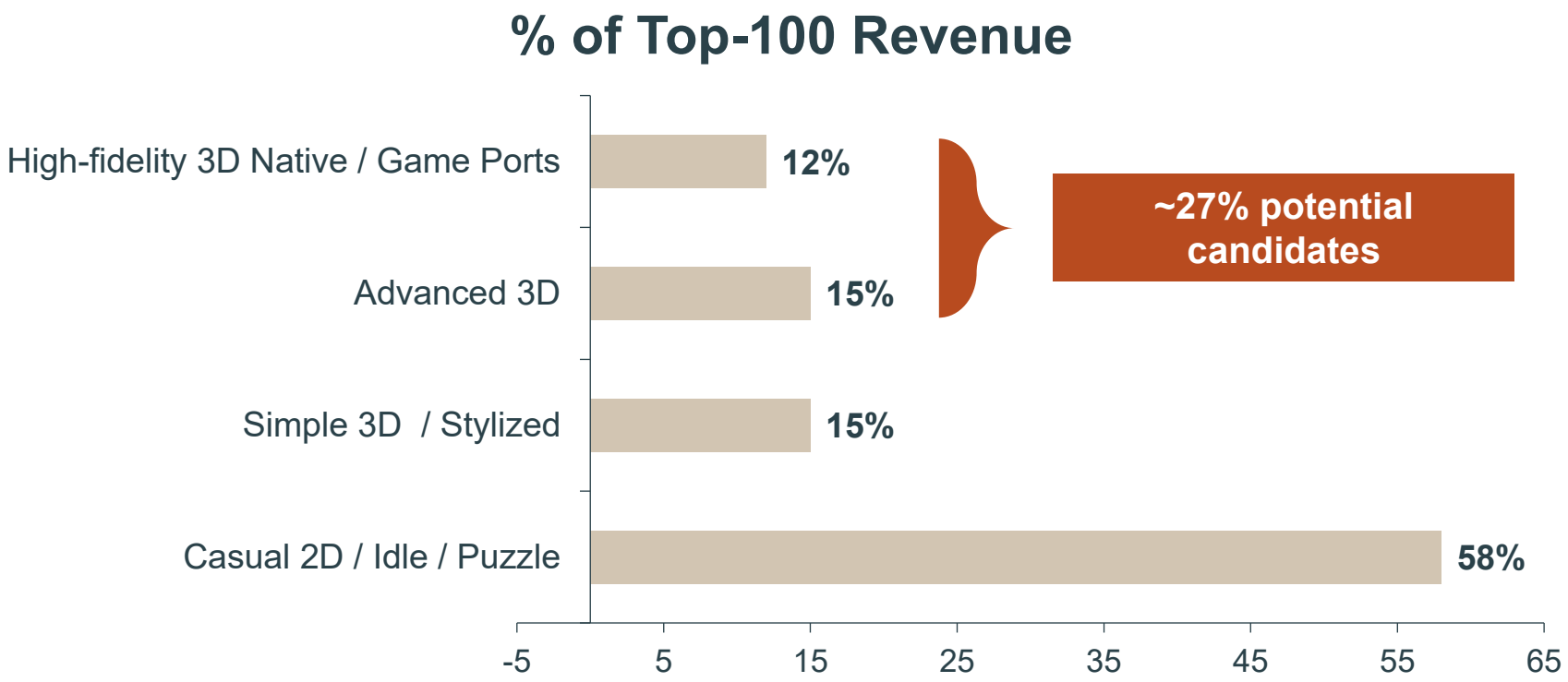
Running at realtime - Snapdragon 8 Gen 5

GPU-Driven Spectrum

Technique	Mobile status	Notes
Nanite (or equivalent)*	**Shipped**	Available on high-end devices
Lumen (or equivalent)*	**Shipped**	UE5.4+ partial support on high-end Android deferred path
MegaLights (or equivalent)*	**Prototype / Shipping**	First few mobile games coming soon
Mesh / Task / Object Shaders	**Shipped**	Power cost is still very high even on uniform scenes.
Compute culling / HZB	**Widely available**	Workhorse of Level 2; supported across all high-end mobile GPUs
Indirect draw / MDI	**Widely available**	*vkCmdDrawIndexedIndirectCount* since Vulkan 1.2 Native on Metal

* many of these techniques also exist on a smaller or partial implementation, not all games need the full end to end solution, or they potentially implemented their own

Mobile Landscape



Based on published renderer analyses, developer conference disclosures, and store category research.

Mobile Landscape

Adoption of demanding rendering techniques takes time

- Deferred rendering was rare on mobile a decade ago. Today it is the default for high-fidelity 3D.

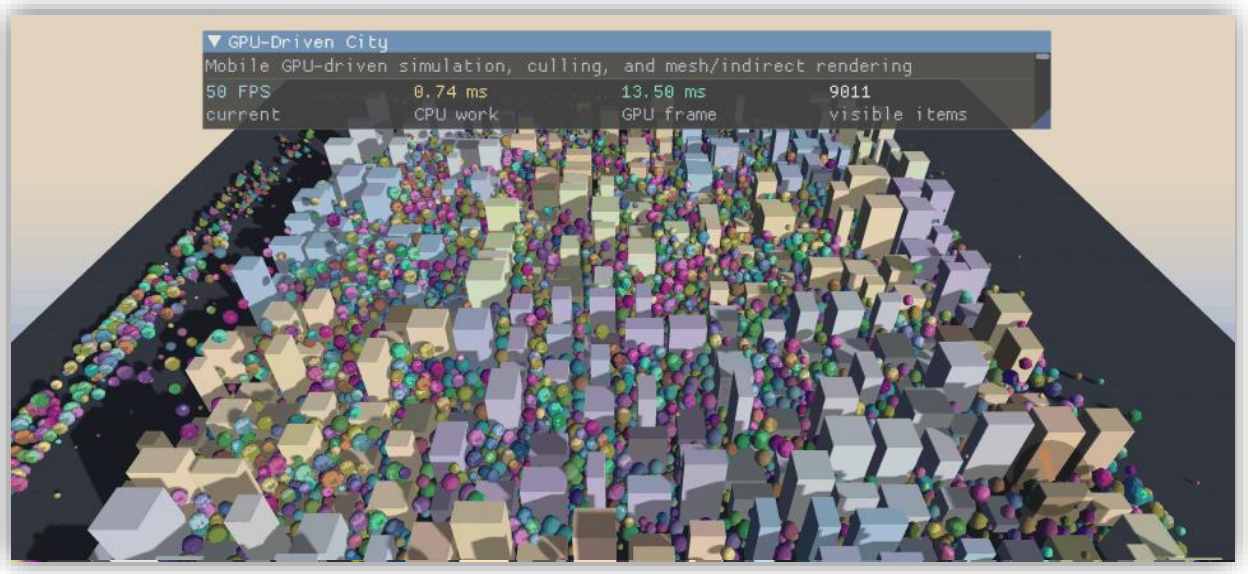
Indirect draw and mesh shaders are following the same curve. The 27% share today is the starting point, not the ceiling.

Mobile Landscape

Adoption of demanding rendering techniques takes time

- Deferred rendering was rare on mobile a decade ago. Today it is the default for high-fidelity 3D.

Indirect draw and mesh shaders are following the same curve. The 27% share today is the starting point, not the ceiling.



Experiment

Experiment

Simple Ball Collision/Occlusion Simulation

- Stress benchmark for specific simulation scenarios
  - Tuned for GPU driven sim
- Multi-LOD
- Multi-material (32 different ball material groups)
- Fullscreen lighting/shadow pass
- Tilt sensor because it feels right

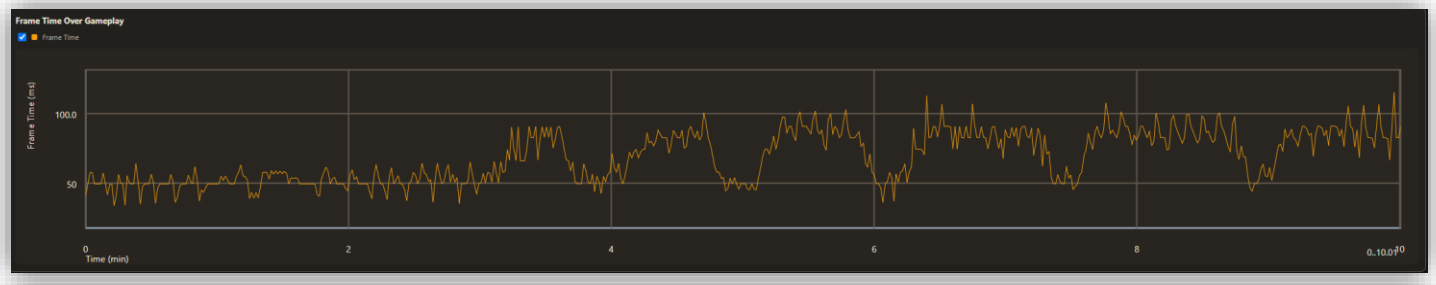
Tested on Snapdragon 8 Elite Gen 5

Experiment

Full CPU (don’t do this)

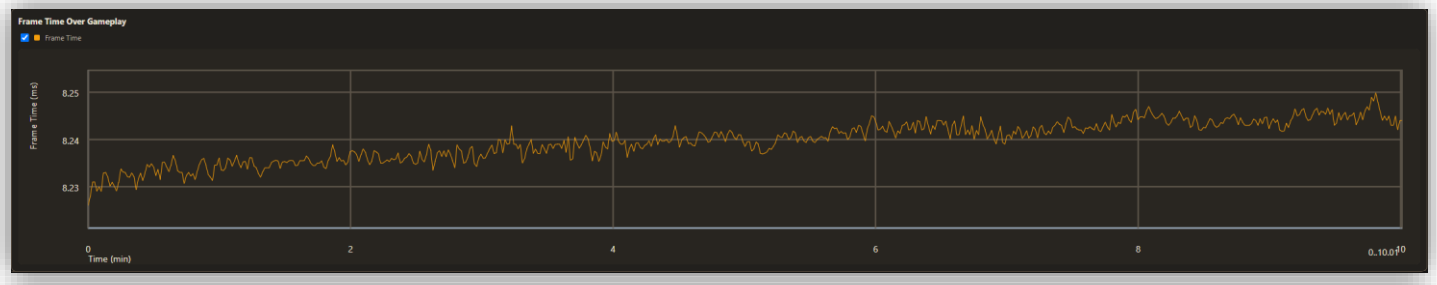
- Simulation + Collision: **CPU**
- Culling: **CPU Frustum**
- Batched draw
- Obviously inefficient

FPS	Power	GPU Usage	GPU Freq
14.6 FPS	5.59 W	82.3%	815 MHz



I wasn’t aware our lab devices could still be operable at such high temperatures

Experiment

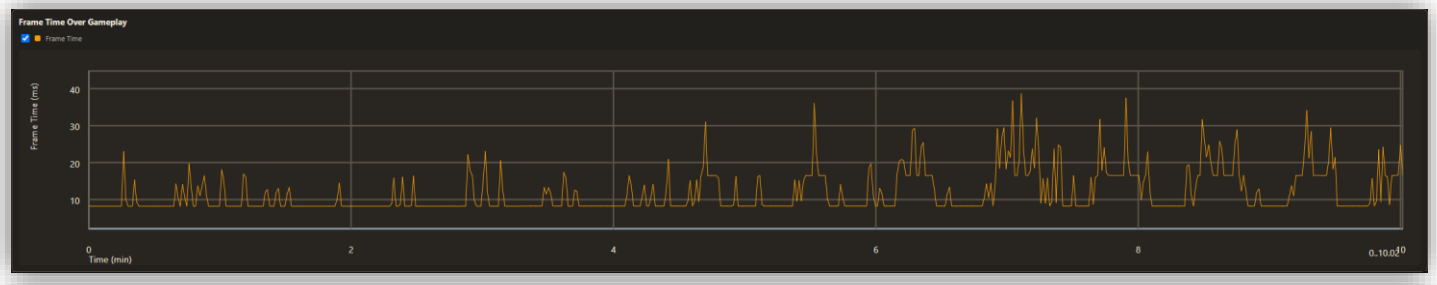


GPU Indexed (best in slot)

- Simulation + Collision: **GPU**
- Culling: **CPU Frustum**
- Batched draw
- Standard/traditional rendering
- Stable

FPS	Power	GPU Usage	GPU Freq
121.4 FPS	2.02 W	82.8%	656 MHz

Experiment



GPU Indirect

- Simulation + Collision: **GPU**
- Culling: **GPU Frustum + GPU HZB**
- Batched draw
- Thermal hit at ~4.3 mins mark

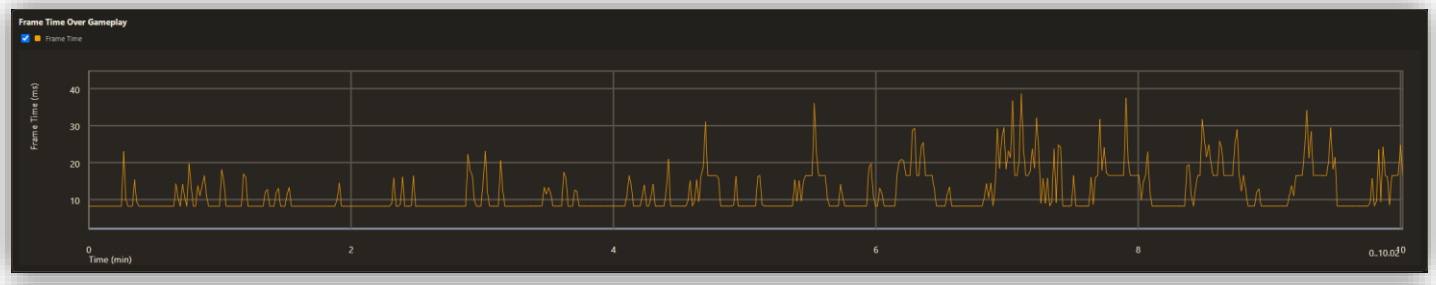
FPS	Power	GPU Usage	GPU Freq
84.4 FPS	3.84 W	98.7%	1044 MHz

Experiment

GPU Indirect

- Simulation + Collision: **GPU**
- Culling: **GPU Frustum + GPU HZB**
- Batched draw
- Thermal hit at ~4.3 mins mark

FPS	Power	GPU Usage	GPU Freq
84.4 FPS	3.84 W	98.7%	1044 MHz



	Total Instructions	Mem Write	Text Read	Cycles Text
Indexed	1215	16	2	12.1
Indirect	**1479**	**42**	**10**	**40.5**

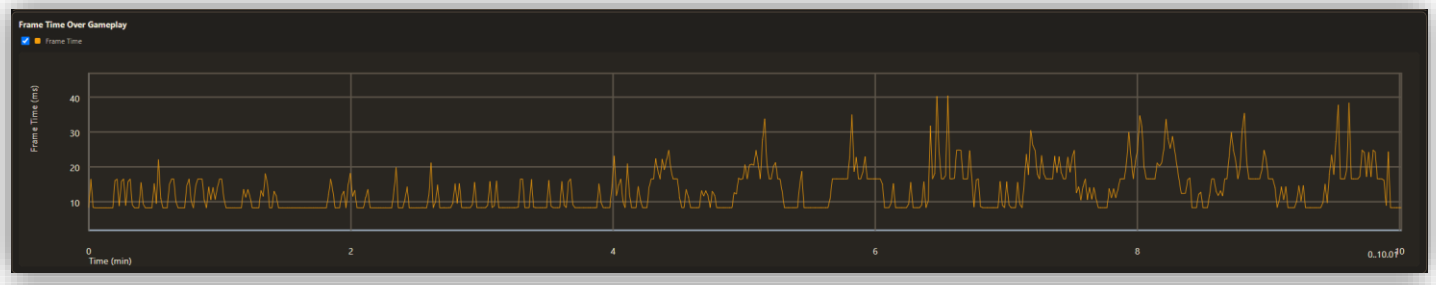
Adreno Offline Compiler data

Experiment

GPU Mesh Shaders

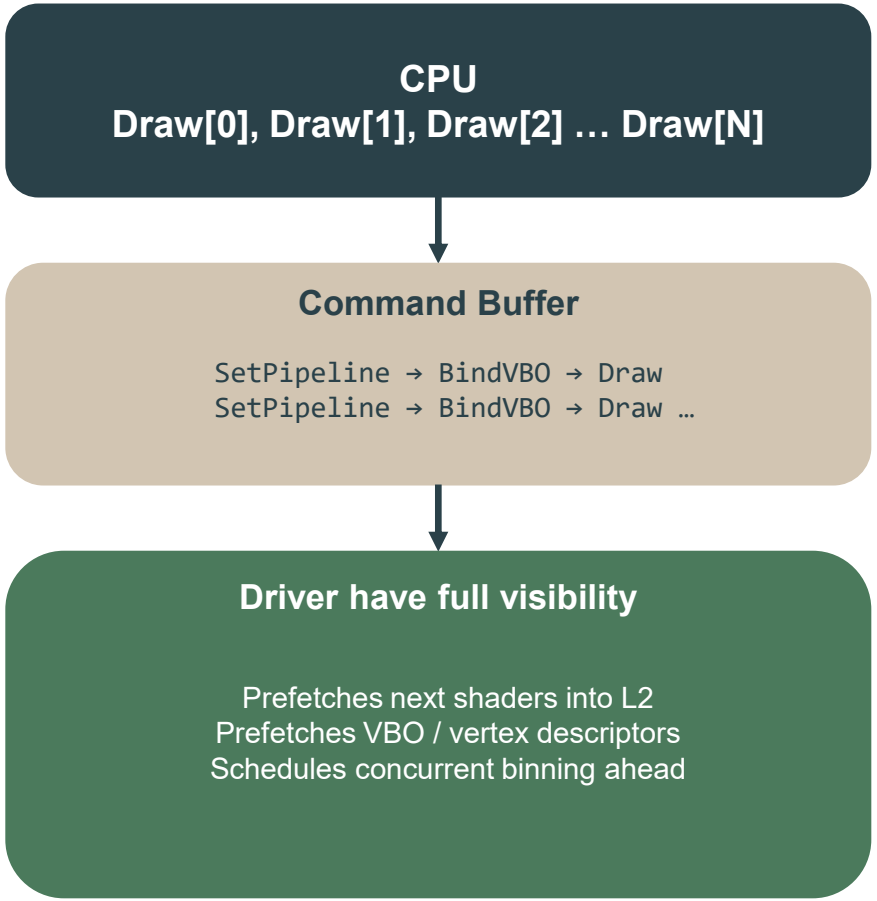
- Simulation + Collision: **GPU**
- Culling: **GPU Frustum + GPU HZB**
- Batched draw
- Thermal hit at ~4.1 mins mark

FPS	Power	GPU Usage	GPU Freq
75.1 FPS	3.77 W	98.7%	1012 MHz



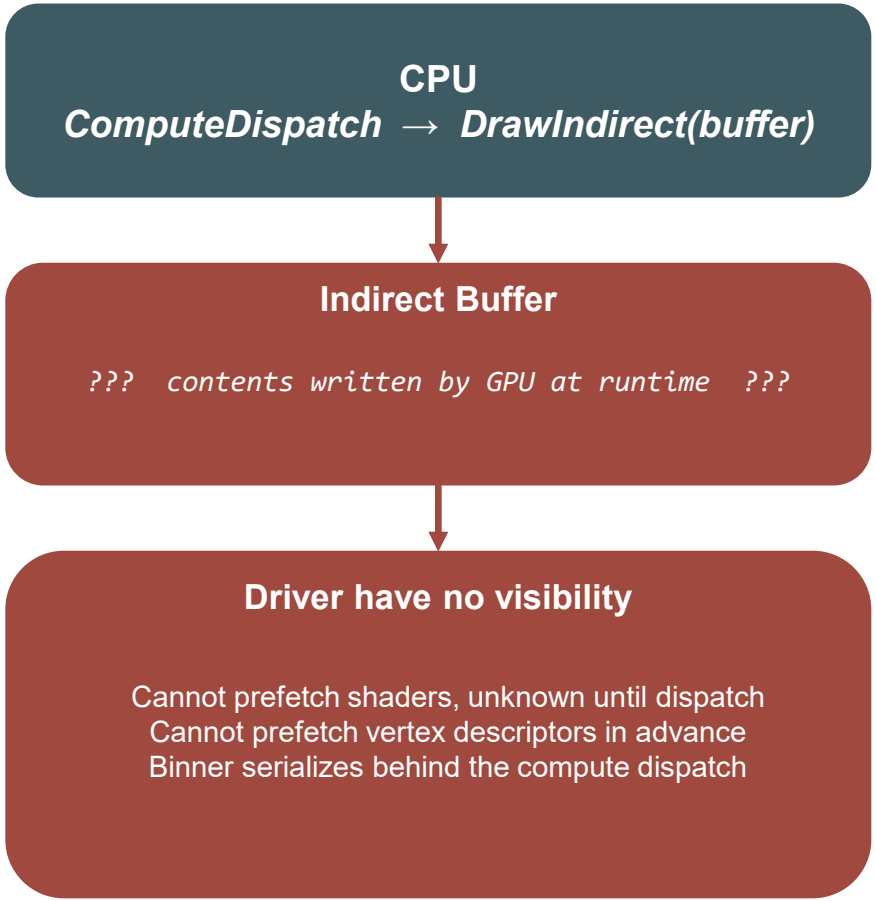
Why is traditional rendering this competitive?

- GPUs (mobile or not) are known to predict which resources will be needed according to what is being submitted
- Commands without GPU indirection clearly defines what and when things will run



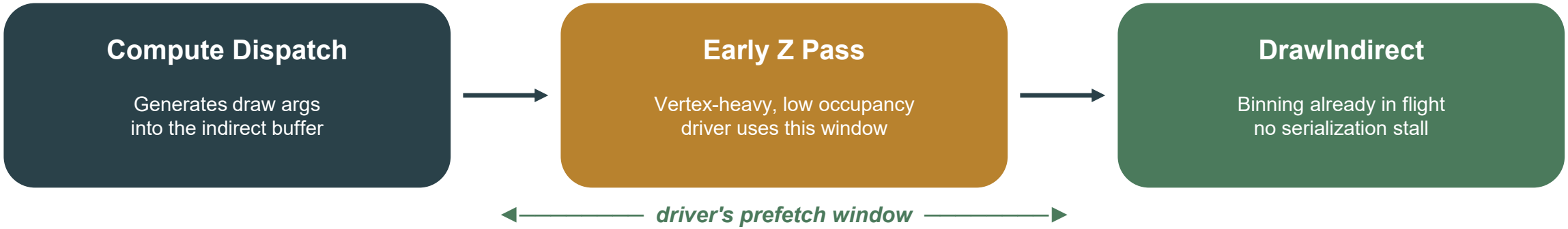
Why is traditional rendering this competitive?

- GPUs (mobile or not) are known to predict which resources will be needed according to what is being submitted
- Commands without GPU indirection clearly defines what and when things will run
- **GPU driven takes away years of driver improvements/hints by removing submission these prediction capabilities**
- **GPUs also need a stronger execution view model since multiple parallel queues could be executing at once**



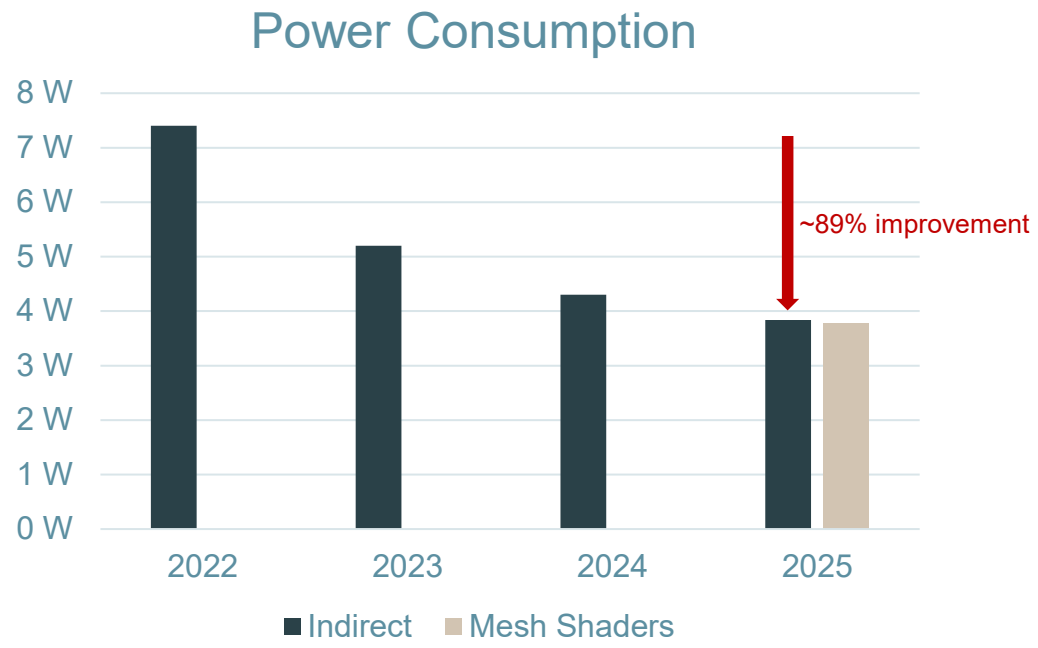
Why is traditional rendering this competitive?

- We are slowly moving optimizations from submission (CPU) level into execution level (GPU)
- GPU can perform certain optimizations dynamically according to the current or previous frame info



New GPU features follow a maturation pattern

- Hardware ray tracing launched on mobile (2022–2023) **with significant power cost**
- By 2024: second-generation RT hardware on high-end mobile delivers roughly 2× the RT throughput.
- The same pattern applies to **indirect draws**, **mesh shaders** and soon **work graphs**:
  - early hardware
  - driver optimization + next-gen hardware
  - increased adoption



GPU-driven today

What mobile GPU-driven games normally share

- Base algorithm:
  - *PreZ*
  - *HZB*
  - *HZB test*
  - *Visibility*
  - *ICB*
  - *IndirectDraw*
- Deferred renderer
- ~3 MB mandatory buffers (low overhead)
- Vertex streaming dominates (400 MB+)
  - Tile memory heap helps quite a bit here

	Game A	Game B	Game C	Game D	Game E
HZB	current	last	last (2x)	history + rast	last
Cull	instance	instance	instance	cluster+inst	inst+sphere
MVP	2.7 MB	30 MB	60 MB	7 MB (FP16)	1 MB

Metric	Indirect ON	Indirect OFF	Delta
Frame GPU time	18.28 ms	16.69 ms	+9.5 %
Main surface	11.50 ms	9.93 ms	+1.58 ms
Draw calls	1024	1027	-3
Read bandwidth	99.8 MB	104.4 MB	-4.6 MB
Main scene read	51.3 MB	56.2 MB	-4.9 MB

5 AAA mobile titles profiled (Snapdragon 8 Gen 3 / 8 Elite).

GPU-driven today

What mobile GPU-driven games normally share

- Base algorithm:
  - *PreZ*
  - *HZB*
  - *HZB test*
  - *Visibility*
  - *ICB*
  - *IndirectDraw*
- Deferred renderer
- ~3 MB mandatory buffers (low overhead)
- Vertex streaming dominates (400 MB+)
  - Tile memory heap helps quite a bit here

	Game A	Game B	Game C	Game D	Game E
HZB	current	last	last (2x)	history + rast	last
Cull	instance	instance	instance	cluster+inst	inst+sphere
MVP	2.7 MB	30 MB	60 MB	7 MB (FP16)	1 MB

Metric	Indirect ON	Indirect OFF	Delta
Frame GPU time	18.28 ms	16.69 ms	**+9.5 %**
Main surface	11.50 ms	9.93 ms	+1.58 ms
Draw calls	1024	1027	-3
Read bandwidth	99.8 MB	104.4 MB	-4.6 MB
Main scene read	51.3 MB	56.2 MB	-4.9 MB

5 AAA mobile titles profiled (Snapdragon 8 Gen 3 / 8 Elite).

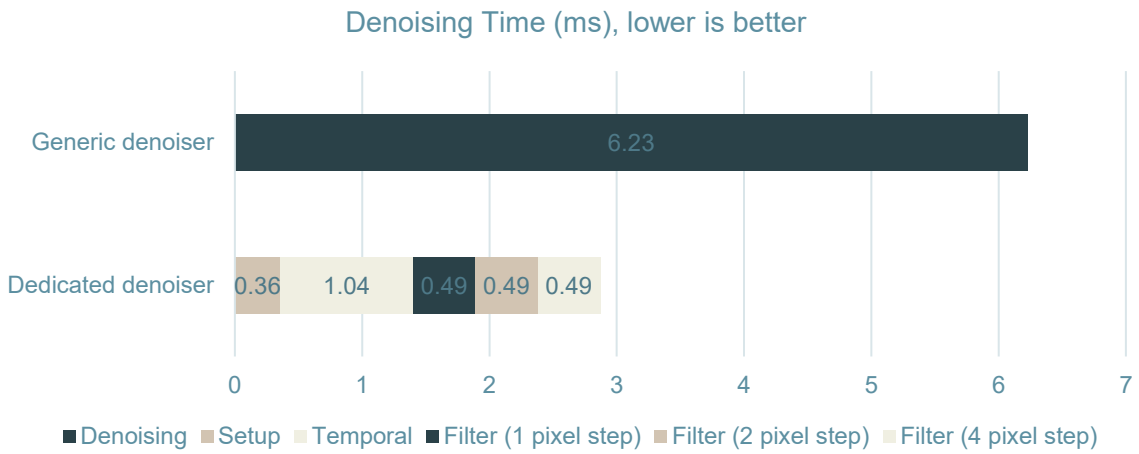
GPU-driven today

How can YOU do it today

- The biggest challenges are:
  - Memory management/bandwidth
  - Occupancy
  - Work scheduling
- The good news: The same principles that make Lumen, Nanite, and modern GPU-driven renderers efficient can be applied to any engine today.

GPU-driven today: Experiment often

- Tile shading / tile shaders
- Device-specific optimizations (texture x buffer balance, 16-bit vs 32-bit data and arithmetic)
- Persistent tile memory: bind buffers to the tile heap so they survive pass boundaries
- Break large compute passes into smaller ones
  - e.g. RT denoising improvements, one traditional big compute broken into 5 smaller ones



GPU-driven today: Shader optimizations

What to do

- For 16-bit image loads, use relaxed precision (*RelaxedPrecision* decoration via inline SPIR-V in HLSL).
- Use FP16 with workgroup size that is a multiple of 128. Halves register pressure; doubles theoretical ALU throughput.
- Maximum thread group dispatch count can be a limitation
  - Work-around it by adapting algorithms to your target device

Watch out for

- Scalarization is not always a win. Many mobile GPUs have scalar registers (*uGPRs*) but the uniform register file is limited.
  Overly complex scalarized paths can hit *i-cache* misses.
- FP16 isn't universally safe. Audit per-shader, some values cannot be made 16-bit without precision loss.
- Large register footprints kill occupancy. Split passes before the compiler starts spilling to LDS.

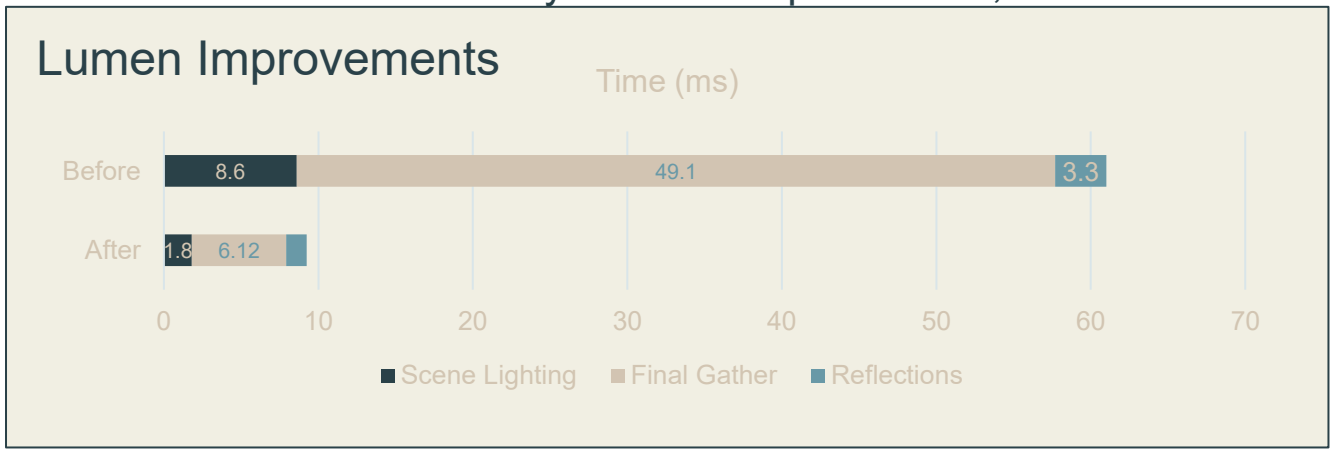
GPU-driven today: Shader optimizations

What to do

- For 16-bit image loads, use relaxed precision (*RelaxedPrecision* decoration via inline SPIR-V in HLSL).
- **Use FP16 with workgroup size that is a multiple of 128. Halves register pressure; doubles theoretical ALU throughput.**
- Maximum thread group dispatch count can be a limitation
  - Work-around it by adapting algorithms to your target device

Watch out for

- Scalarization is not always a win. Many mobile GPUs have scalar registers (*uGPRs*) but the uniform register file is limited.
  Overly complex scalarized paths can hit *i-cache* misses.
- FP16 isn't universally safe. Audit per-shader, some



GPU-driven today: Indirect draw generation

Generate and consume pitfall

- Compute shader generates draw commands, then *DrawIndirect* is issued immediately after.
- Driver does not have vertex data early enough to schedule binning in advance.
- Binning serializes behind the compute dispatch. GPU idles on the transition.
- This pattern is common in large engines for GI card capture and shadow map passes.

What to do

- Move the draw-generating compute shader earlier in the frame.
- Insert other GPU work between the compute dispatch and the *DrawIndirect*.
- Good fit: early Z pass. Vertex-heavy, low occupancy, does not conflict with the cull compute.

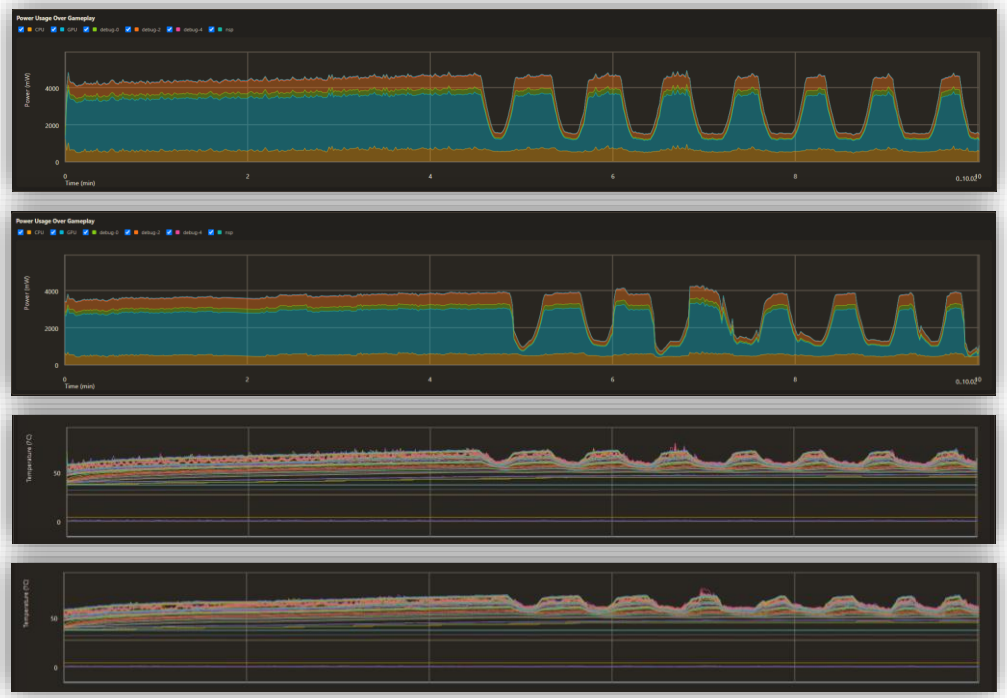
Result: driver have time to property prepare for the indirect call (e.g. enabling concurrent binning). No shader changes. Just reorder your frame graph.

GPU-driven today: Other

- Respect storage buffer size limitations
  - Be mindful of the meshlet sizes and their impact
- High geometry scenes benefit hugely from GPU driven rendering
- Meshlet culling is fast and effective
- Depth buffer culling of meshlets is great, if you have depth
- **Bindless resources aren’t too expensive anymore**

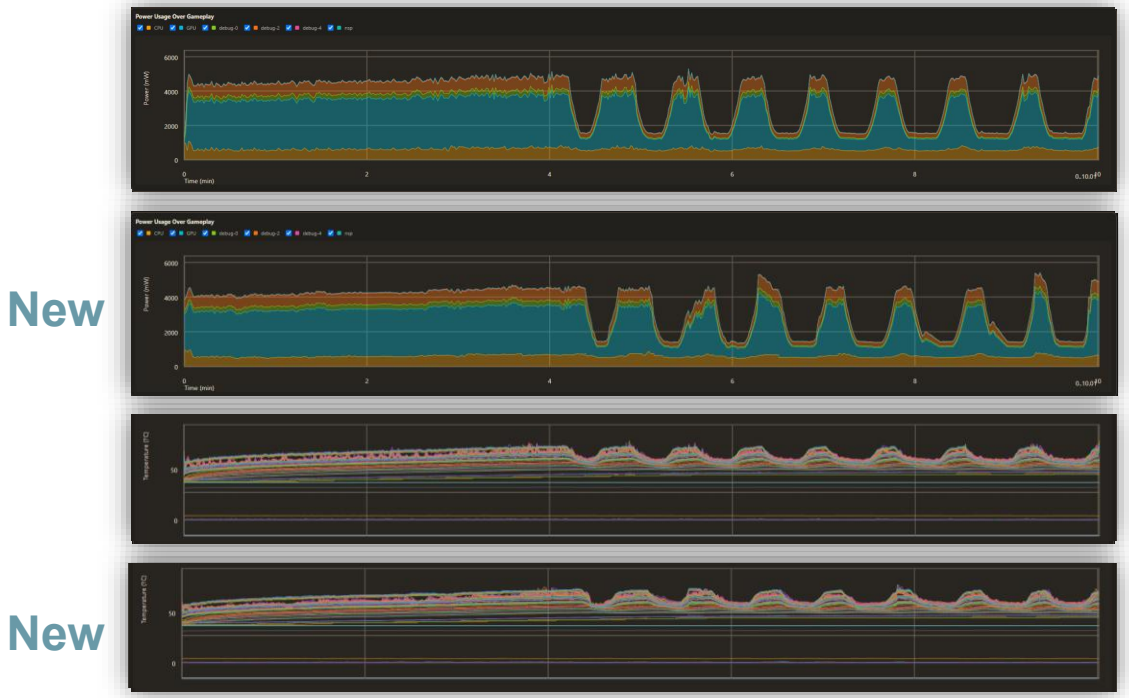
Experiment: After Improvements

GPU Indirect



	FPS	Power	GPU Usage	GPU Freq
Old	84.4 FPS	3.84 W	98.7%	1044 MHz
New	**107.2 FPS**	**3.43 W**	**98.4%**	**1026 MHz**

GPU Mesh Shaders



	FPS	Power	GPU Usage	GPU Freq
Old	75.1 FPS	3.77 W	98.7%	1012 MHz
New	**88.6 FPS**	**3.58 W**	**98.0%**	**1008 MHz**

Thank you