

Vulkanised 2026

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

Making Friends and Influencing Bounding Volume Hierarchies.

John Talley, Samsung



The Problem

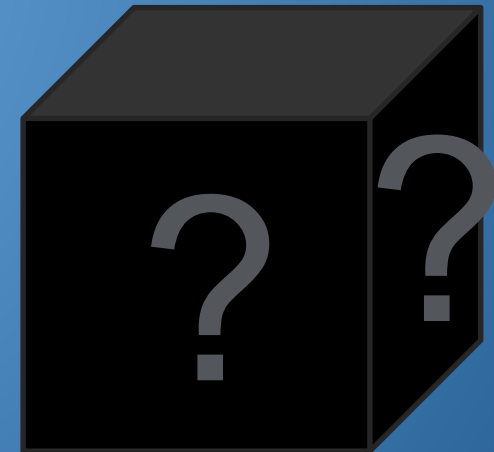
From a Developers' Perspective

The Vulkan API gave developers A LOT of control.

- We worked hard, but we were happy!

Then we finally got Ray Tracing capabilities, but...

- It feels like we lost control!
- `vkBuildAccelerationStructuresKHR` is a black box.
- We put geometry in, and a BVH comes out.



The Thesis

A New Hope

- The "friendship" between your geometry vertex/index buffers and the BVH builder isn't passive!
- How you organize data significantly impacts BVH footprint and traversal speed.

Session Goals

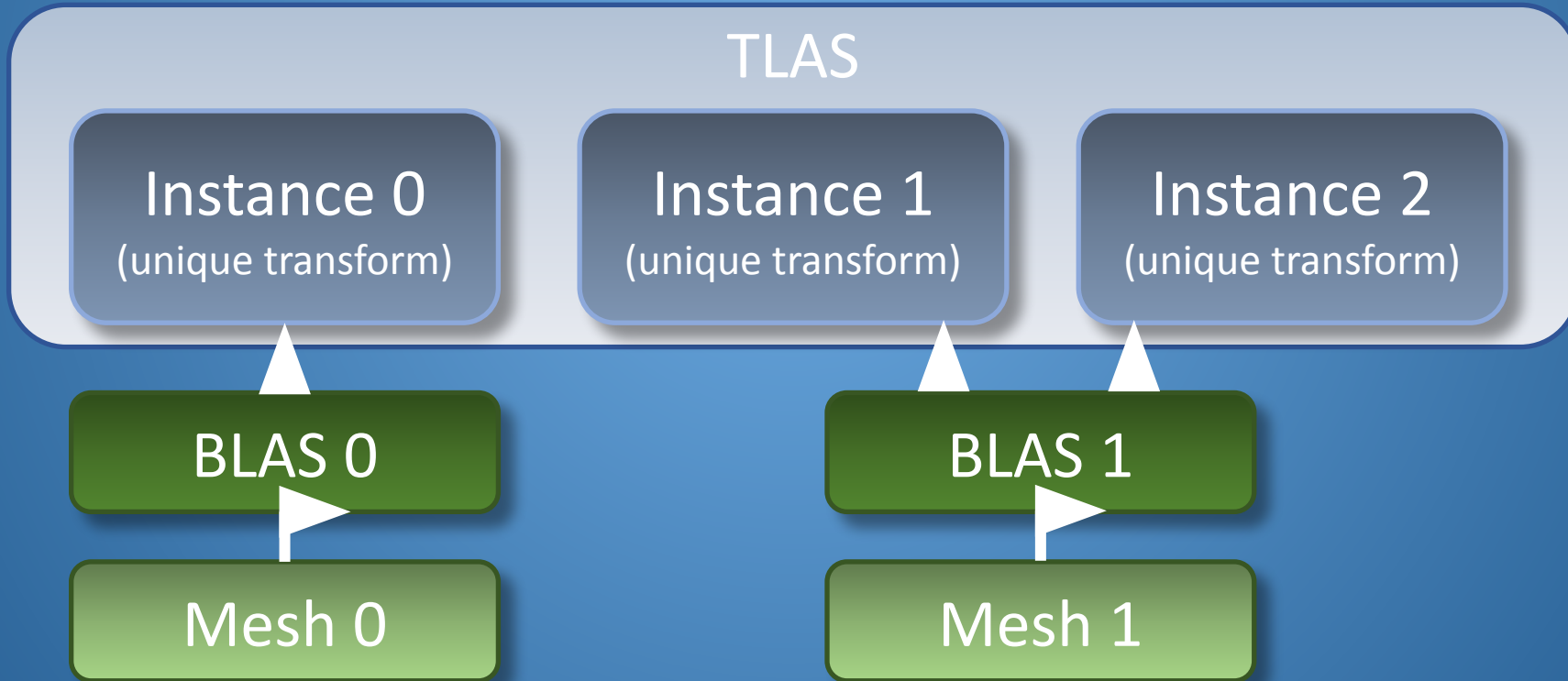
Return of the Geometry

- Quantify the impact of various options on BVH quality and build speed.
 - Vertex/Index Buffer Layouts
 - Acceleration Structure Geometry Merging & Subdivision
 - Build Flags
- Provide vendor-agnostic optimization strategies.
- Encourage developers to experiment

BVH Fundamentals

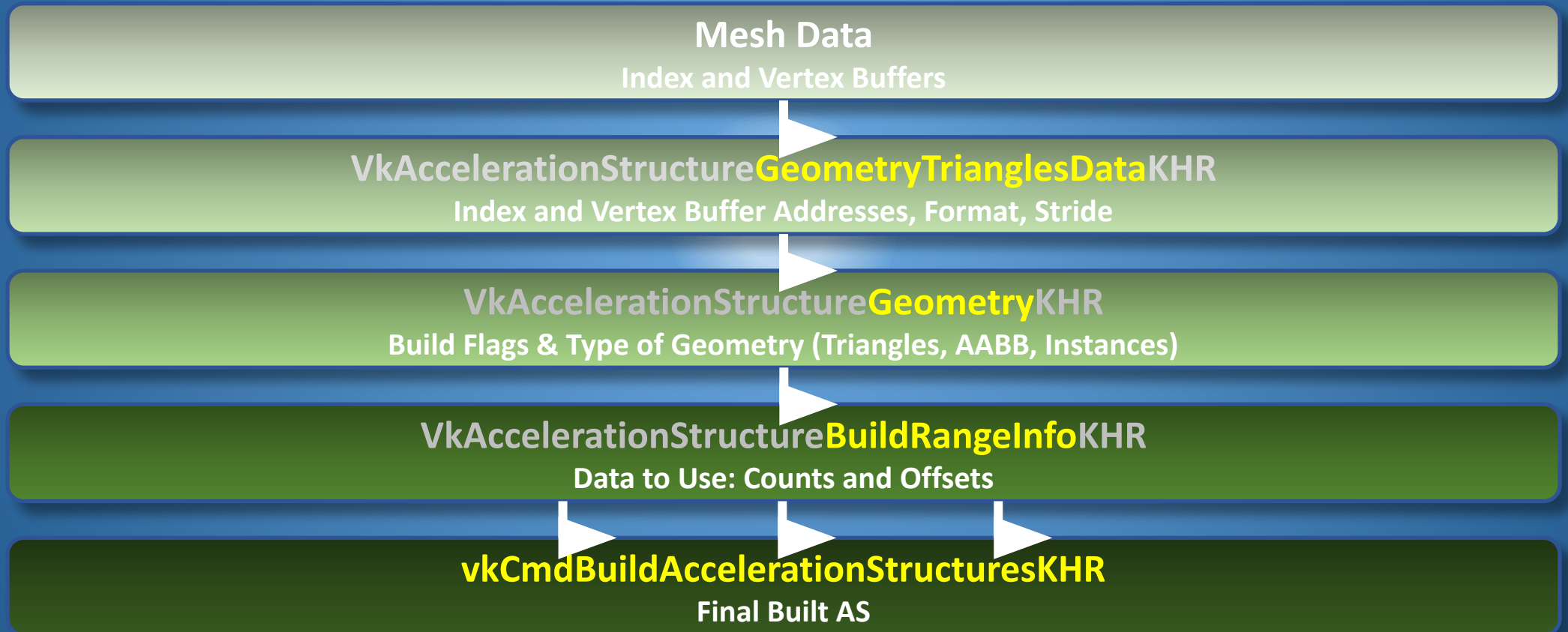
Vulkan Acceleration Structures

Top-Level (TLAS) & Bottom-Level (BLAS)



BVH Fundamentals

Building Acceleration Structures



BVH Fundamentals

Key Metrics

- **The "Black Box" Builder:** While the driver builds the BVH, it is constrained by the primitive data we provide.
- To provide good data, we need to understand what makes a "good" BVH.
 - **SAH (Surface Area Heuristic):** The gold standard for measuring BVH quality.
 - **Memory Footprint:** Size matters for cache hit rates during traversal.

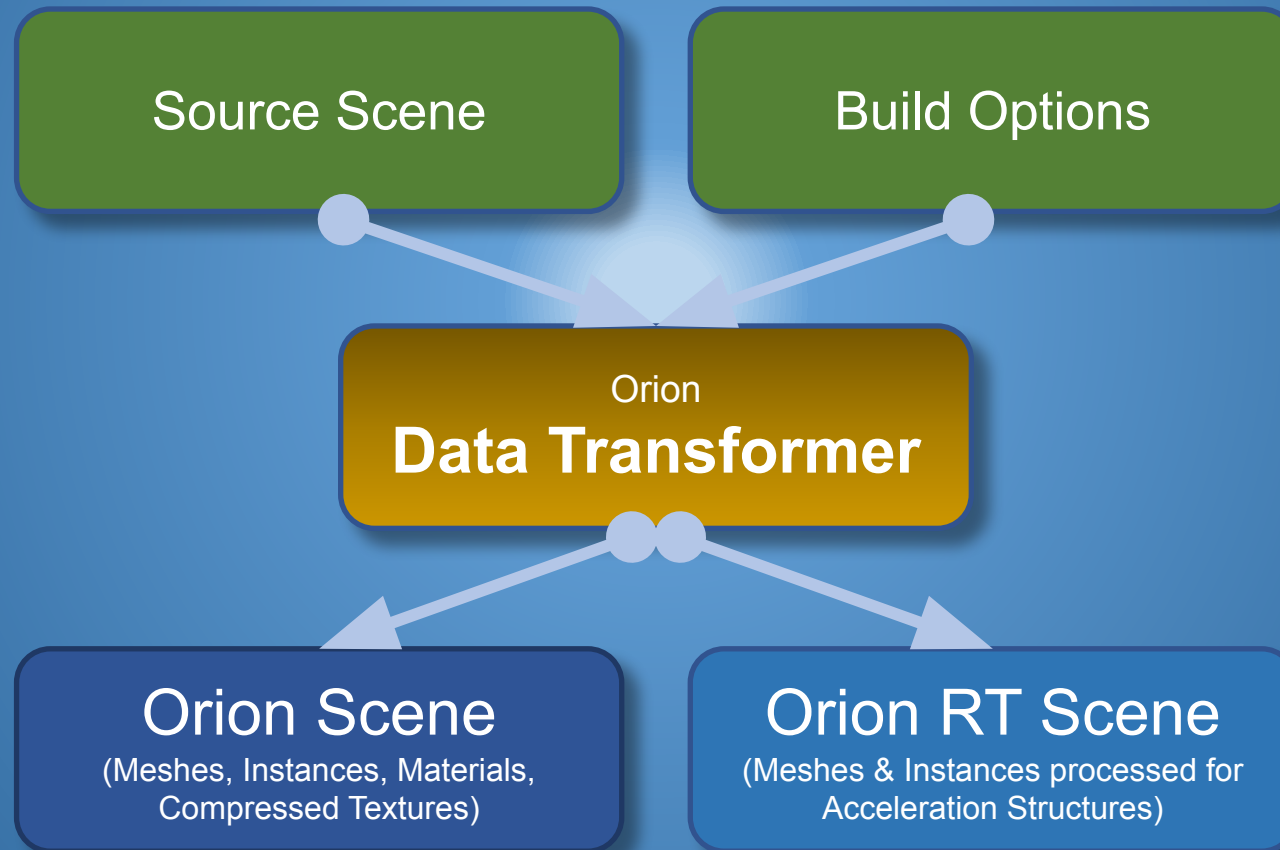
Methodology & Environment

Orion

- The Orion SDK
 - Rendering Framework and Toolset
 - Data Driven Render-Graph
 - Visual Graph Editor
 - Provides Rapid Experimentation
 - Vulkan 1.3 bindless design
 - Independent data transformation pipeline
 - Supports Android, Windows, Linux
 - Open source in Q2/Q3

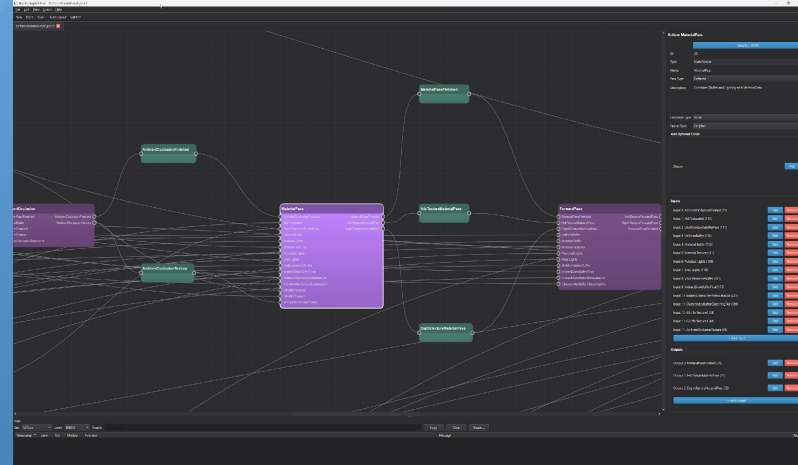
Methodology & Environment

Orion



Methodology & Environment

Orion



Methodology & Environment

Test Devices

- Tests were run on three separate vendor devices
 - Samsung Exynos 2500
 - Qualcomm Snapdragon Adreno 830
 - NVIDIA RTX 2000 Ada Generation (similar to RTX 4060)
- Premise:
 - Results are normalized to device baselines
 - Evaluate changes for each device independently
 - ****Not compare the devices to one another.**

Methodology & Environment

Test Scenes

Bistro

- Triangles: 4,209,006
- Instances: 2909
- Meshes: 551
- Lights: 116
- Credit: Amazon Lumberyard (2017)



San Miguel

- Triangles: 10,502,029
- Instances: 160,297
- Meshes: 809
- Lights: 81
- Credit: Guillermo M. Leal Llaguno from PBRT V3 archive



Parameter Sensitivity

Primitive Ordering

Does spatial sorting (e.g., Morton code sorting) of your index buffer prior to building lead to better node splitting?

- Morton Codes
- SAH optimization (mesh optimizer)

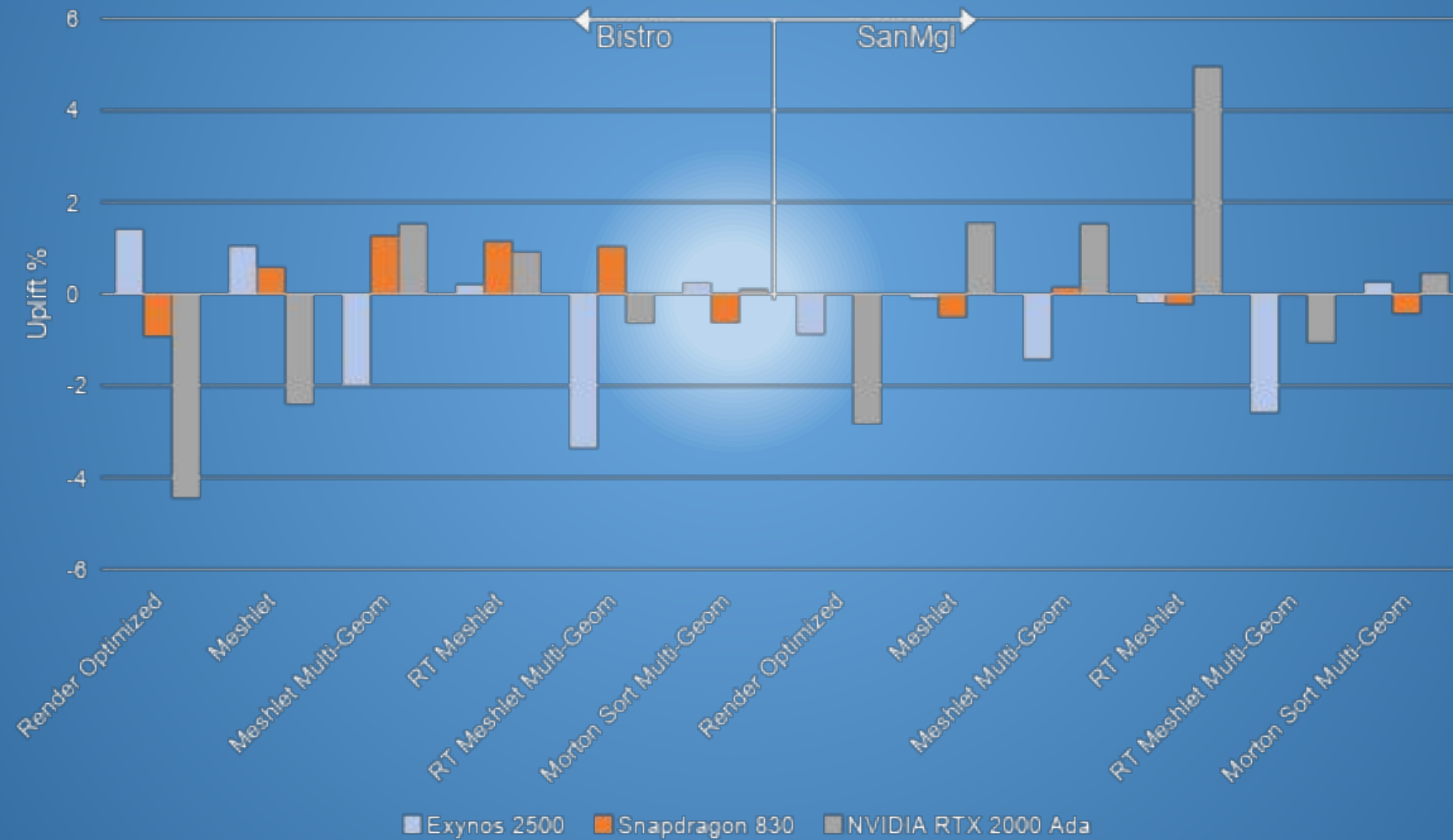
Parameter Sensitivity

Geometry Ordering

- **Geometry Layouts:** Subdividing or Merging Meshes into pre-grouped Geometries
- **Geometry Size:** Impact of individual geometry size on the build and trace performance.

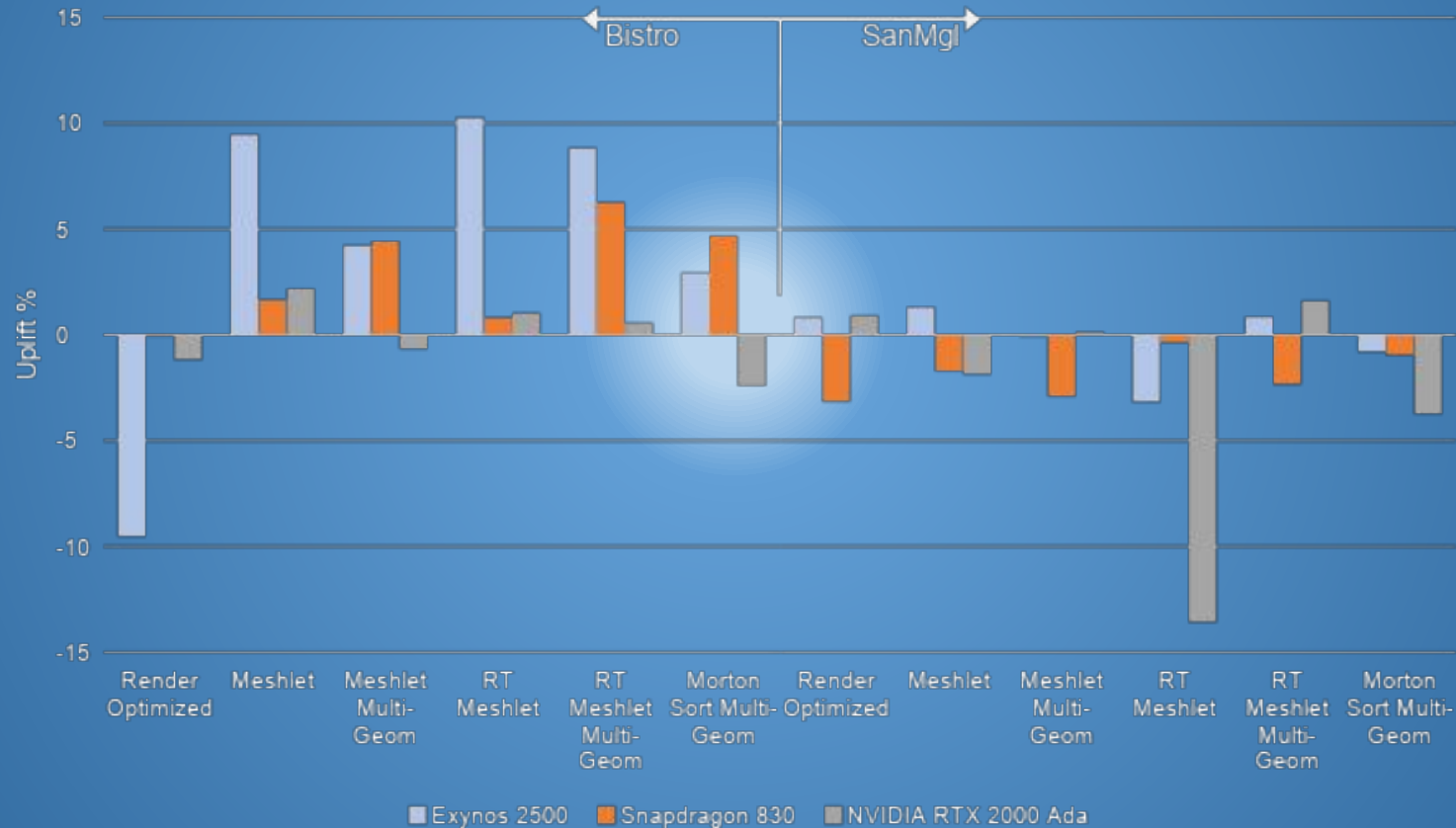
Results

Primitive & Geometry effects on Build Time



Results

Primitive & Geometry effects on Trace Time



Parameter Sensitivity

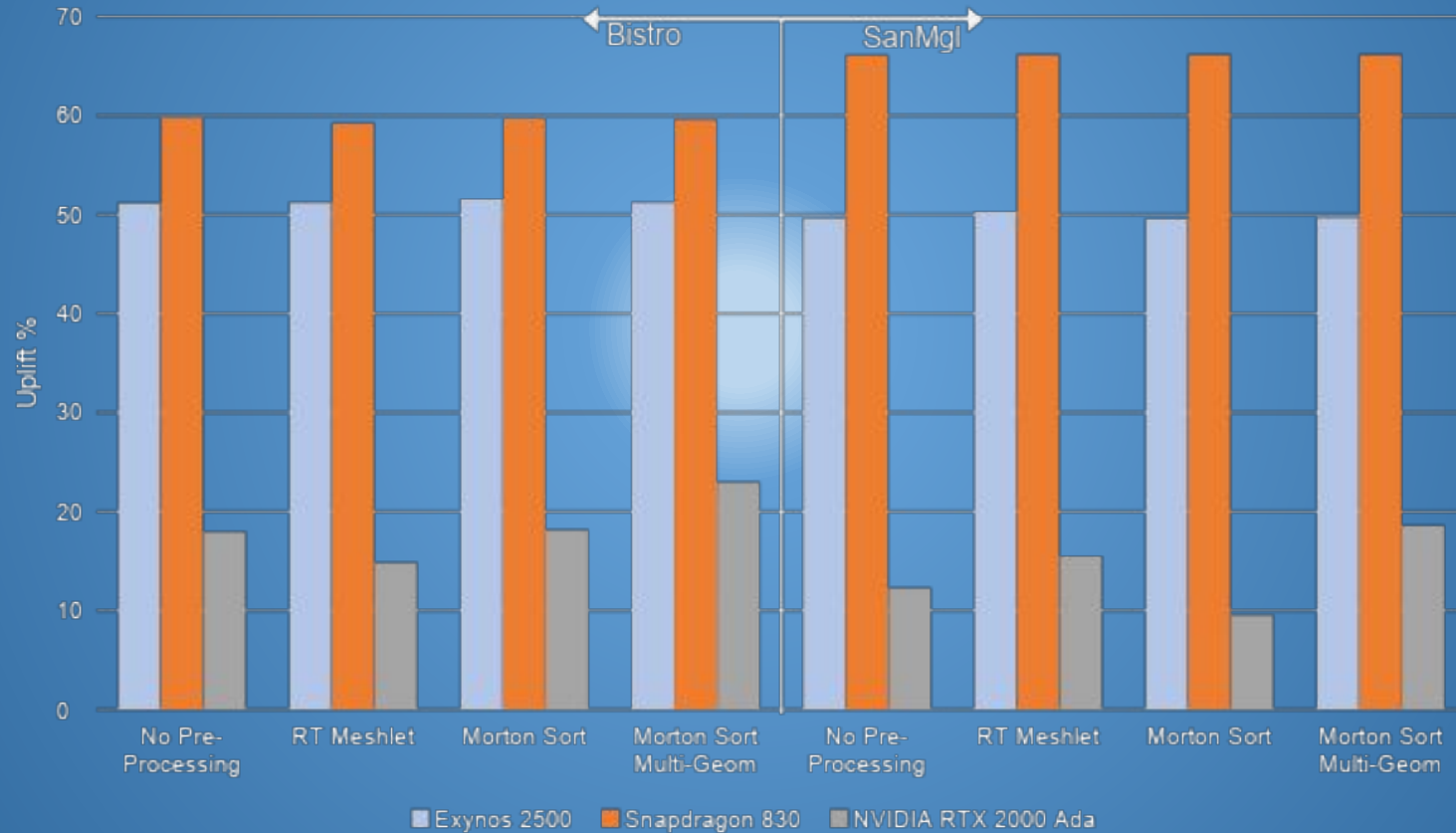
Fast Build

Fast Build vs Fast Trace

- What are the trade offs of building with the fast build option vs the fast trace option?
- How does Fast Build interact with primitive and geometry ordering strategies?

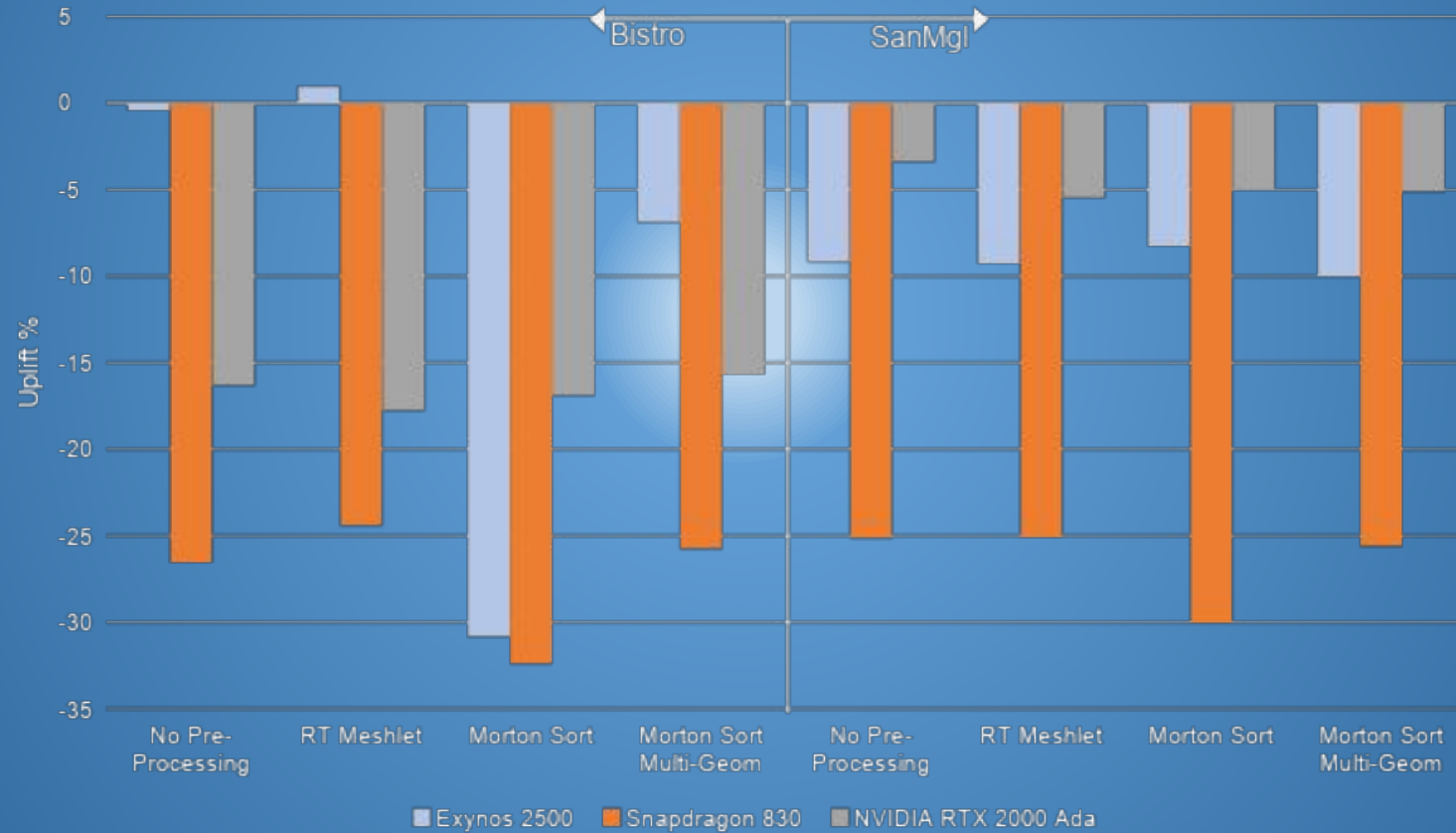
Results

Fast Build effects on Build Time



Results

Fast Build effects on Trace Time



Parameter Sensitivity

Data Size Optimization

- **Compaction:** Impact of BVH Compaction via `vkCmdCopyAccelerationStructureKHR` with `VK_COPY_ACCELERATION_STRUCTURE_MODE_COMPACT_KHR`
- **Alpha Test:** Does `VK_GEOMETRY_OPAQUE_BIT_KHR` influence the builder's ability to optimize or trace times?
 - Leave non-opaques out of BVH when possible?

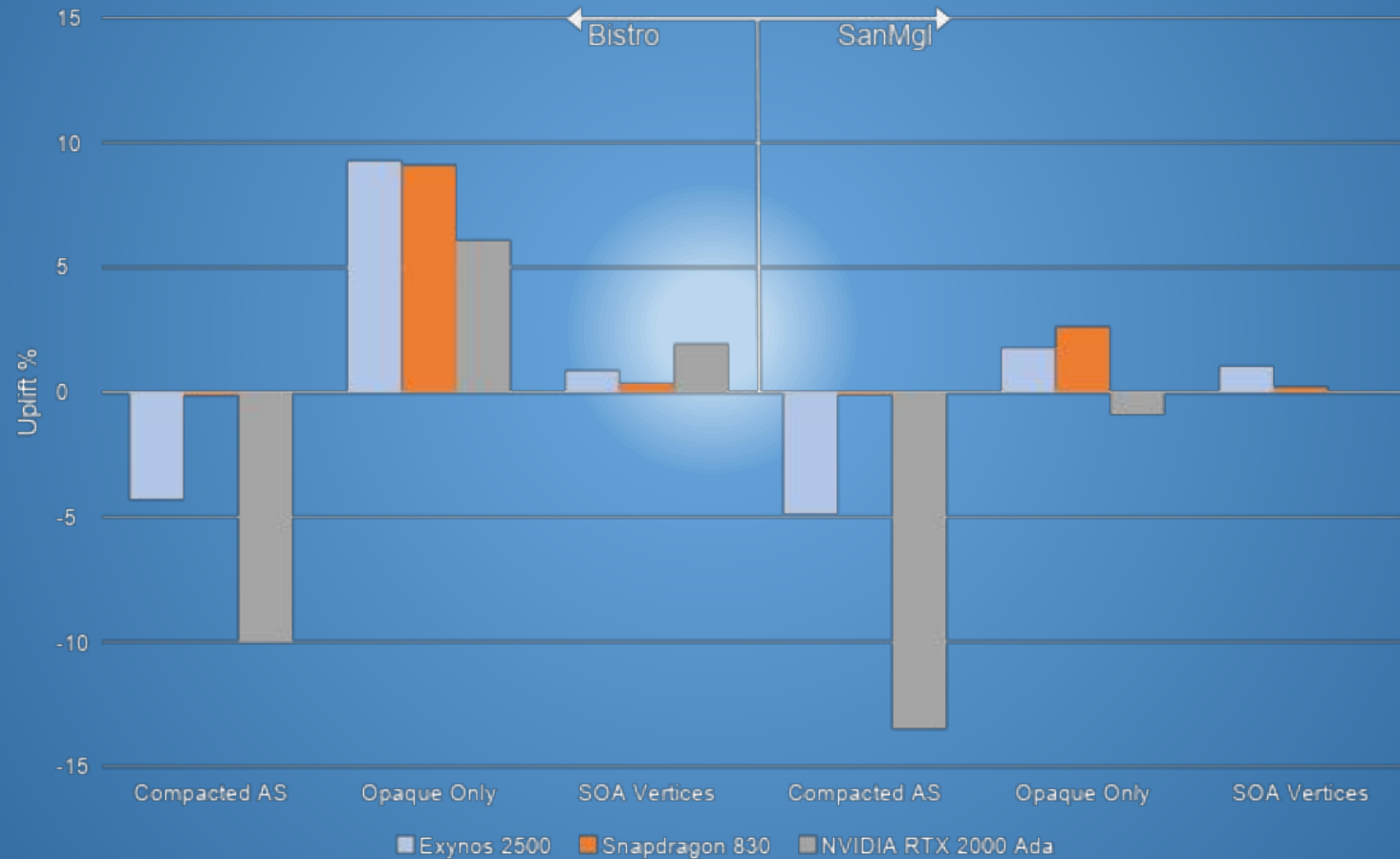
Parameter Sensitivity

Data Size Optimization

- **Vertex Layouts:** AOS (Array of Structures) vs. SOA (Structure of Arrays).
- **Vertex Size:** Impact of data size on the builder's memory access patterns.

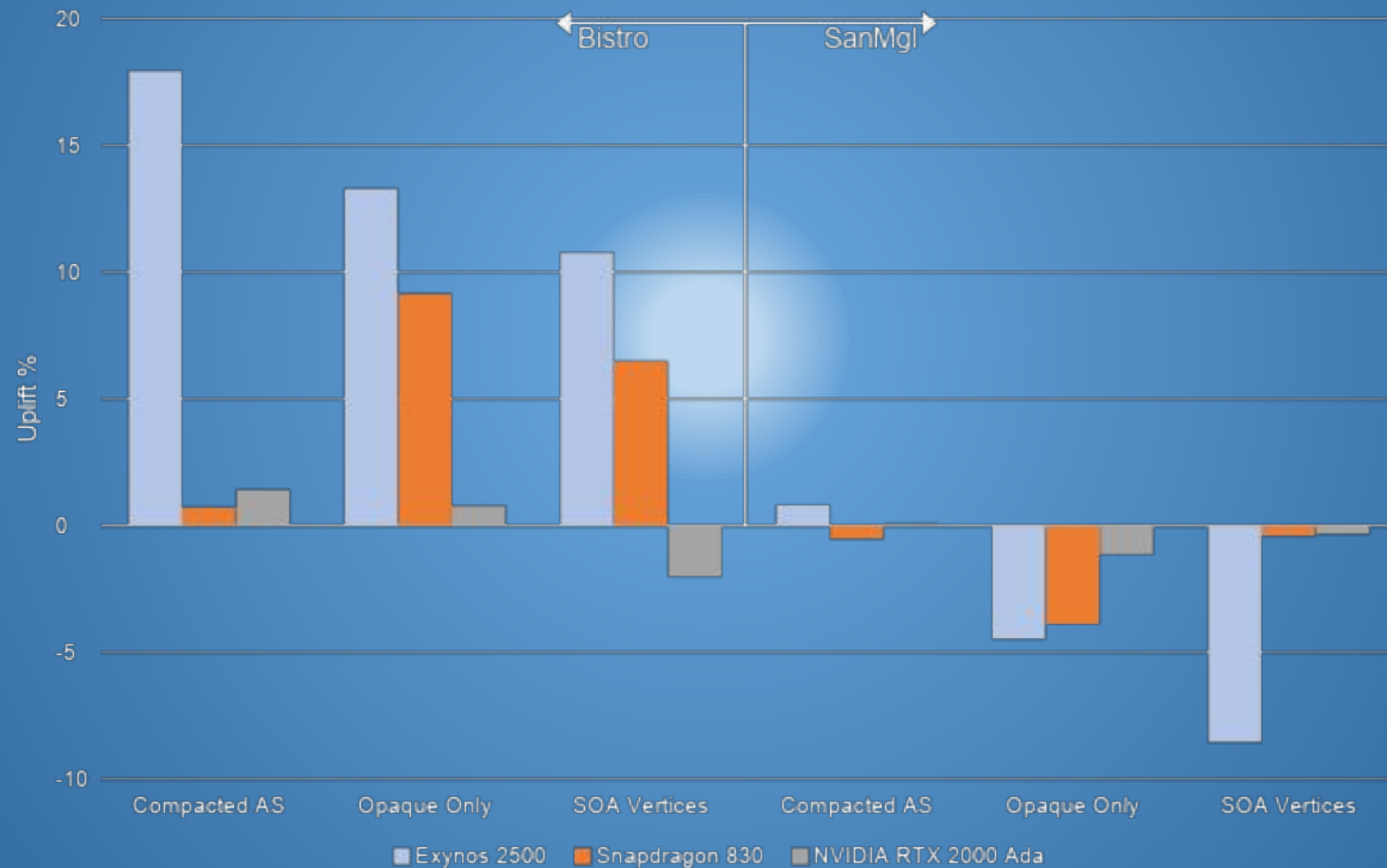
Results

Data Size effects on Build Time



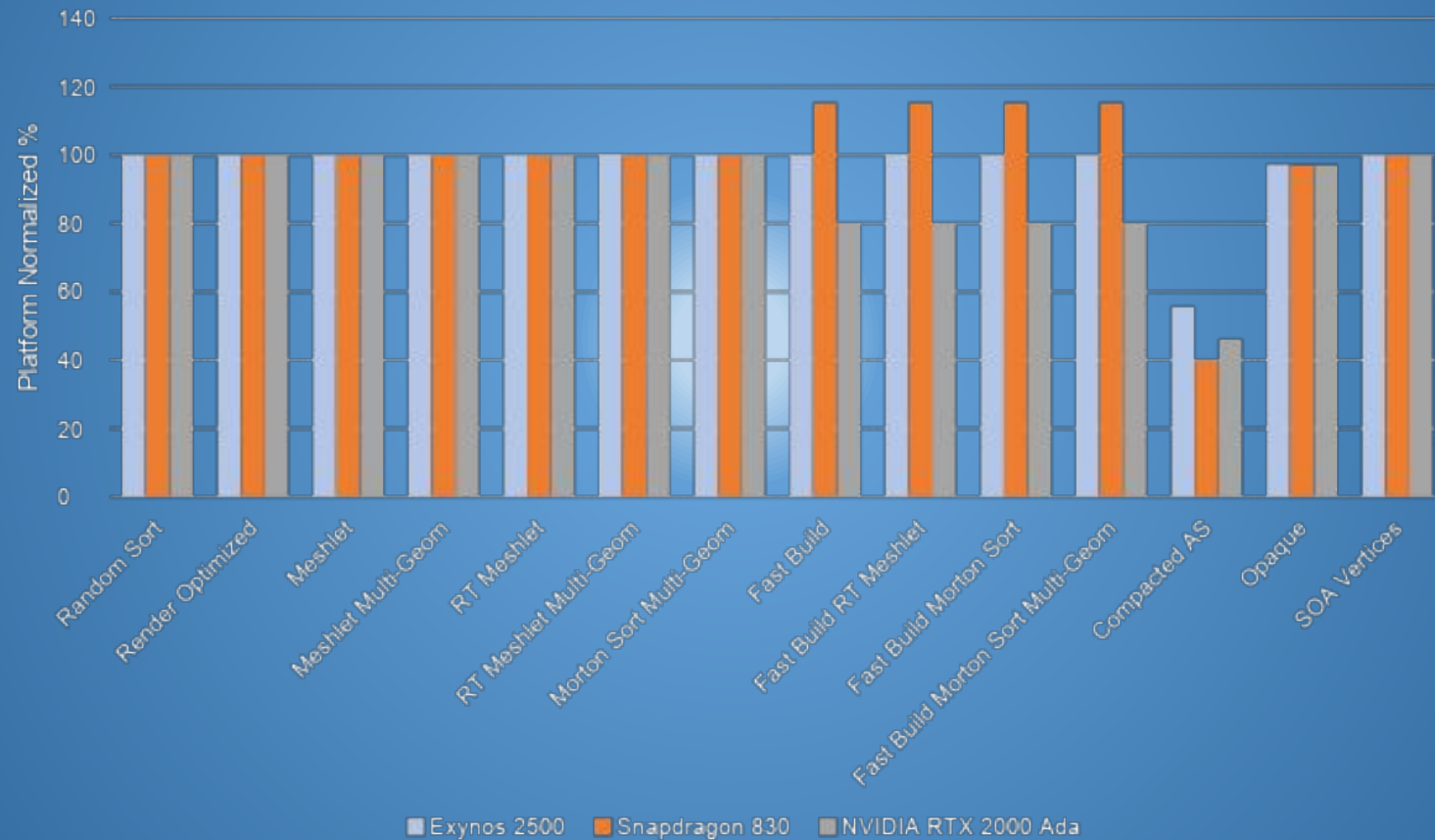
Results

Data Size effects on Trace Time



Results

Influences on Build Size



Recommendations

No Universal “Golden Rules”

- **Find Your Rules:** To truly master ray tracing in Vulkan, we must stop treating the Acceleration Structure as an “uncontrollable” object.
 - **Test Your Data:** Don't assume all data is the same. Implement various compression, ordering, and update options.
 - **Measure the Build vs. Trace:** Don't just optimize for the fastest build time. If a 3ms longer build saves 2ms on every subsequent frame's trace, that is a massive win for your frame budget.
 - **Cross-Vendor Validation:** Test on at least two different Devices. What is "free" on one architecture may be a bottleneck on another.

Conclusion

Happy Buffer Happy BVH

Summary: Good BVHs start at the buffer level.

- A BVH is only as good as the indices that define it. By treating our geometry buffers as spatial databases rather than simple lists, we can 'free' up some of our frame budget without changing a single line of ray-tracing shader code.

Final Thought: Influence your BVH early so it treats your rays friendly later.

Coming Soon

Exynos Tools & Technology

Sokatoa, our new graphics profiling solution for Android, will be available to download for free.

<https://github.com/sarc-acl/sokatoa>



We Are Hiring

Come join our team



<https://semiconductor.samsung.com/about-us/careers/us/sarc-acl/>

Thank You!

Samsung SARC Advanced Rendering Team

Tim Little, John Talley, Mike Kelley

BACKUP

RenderGraph Editor - ForwardRenderGraphDebug.json

File Edit View Graph Help

New Open Save Auto layout Validate

ForwardRenderGraphDebug.json

```

    graph LR
      subgraph Prepare_Base_GPU [Prepare Base GPU]
        direction TB
        P1[Prepare Base GPU]
        P2[Prepare Base GPU]
        P3[Prepare Base GPU]
        P4[Prepare Base GPU]
        P5[Prepare Base GPU]
        P6[Prepare Base GPU]
        P7[Prepare Base GPU]
        P8[Prepare Base GPU]
        P9[Prepare Base GPU]
        P10[Prepare Base GPU]
      end
      subgraph Call_Base_GPU [Call Base GPU]
        direction TB
        C1[Call Base GPU]
        C2[Call Base GPU]
        C3[Call Base GPU]
        C4[Call Base GPU]
        C5[Call Base GPU]
      end
      subgraph Main_Render_Loop_GPU [Main Render Loop GPU]
        direction TB
        M1[Main Render Loop GPU]
        M2[Main Render Loop GPU]
        M3[Main Render Loop GPU]
        M4[Main Render Loop GPU]
        M5[Main Render Loop GPU]
        M6[Main Render Loop GPU]
        M7[Main Render Loop GPU]
        M8[Main Render Loop GPU]
        M9[Main Render Loop GPU]
        M10[Main Render Loop GPU]
      end
      subgraph Forward_Render [Forward Render]
        direction TB
        F1[Forward Render]
        F2[Forward Render]
        F3[Forward Render]
        F4[Forward Render]
        F5[Forward Render]
        F6[Forward Render]
        F7[Forward Render]
        F8[Forward Render]
        F9[Forward Render]
        F10[Forward Render]
      end
      subgraph Post_Process [Post-Process]
        direction TB
        P1[Post-Process]
        P2[Post-Process]
        P3[Post-Process]
        P4[Post-Process]
        P5[Post-Process]
        P6[Post-Process]
        P7[Post-Process]
        P8[Post-Process]
        P9[Post-Process]
        P10[Post-Process]
      end
      subgraph Render_Queue [Render Queue]
        direction TB
        R1[Render Queue]
        R2[Render Queue]
        R3[Render Queue]
        R4[Render Queue]
        R5[Render Queue]
        R6[Render Queue]
        R7[Render Queue]
        R8[Render Queue]
        R9[Render Queue]
        R10[Render Queue]
      end
      subgraph Clear_GPU [Clear GPU]
        direction TB
        C1[Clear GPU]
        C2[Clear GPU]
        C3[Clear GPU]
        C4[Clear GPU]
      end
      subgraph Main_Render_Loop_CPU [Main Render Loop CPU]
        direction TB
        M1[Main Render Loop CPU]
        M2[Main Render Loop CPU]
        M3[Main Render Loop CPU]
        M4[Main Render Loop CPU]
      end
      Prepare_Base_GPU --> Call_Base_GPU
      Call_Base_GPU --> Main_Render_Loop_GPU
      Main_Render_Loop_GPU --> Forward_Render
      Forward_Render --> Post_Process
      Post_Process --> Render_Queue
      Render_Queue --> Clear_GPU
      Clear_GPU --> Main_Render_Loop_CPU
      Main_Render_Loop_CPU --> Main_Render_Loop_GPU
  
```

Graph Metadata

Version

Major: 0

Minor: 4

Rendergraph type: Base

Name: Forward Render - GPU Cull And Cluster Lights Debug

Comment: Debug renders in LDR buffer

Logs

Tab: All Tabs Level: ERROR Search: [Empty]

Timestamp	Level	Tab	Module	Function	Message	Details
0 of 41 log entries						

Node: 0 Validation: Not checked Tab: 1/1



Exported from Unreal store to glTF

BACKUP