

# KosmicKrisp

Production Ready!

Get this Presentation



# Wait a minute, who are you?

- Aitor Camacho Larrondo
- Game and GPU driver development background
- With LunarG since 2022
  - Vulkan Validation Layers
  - Vulkan CTS
  - MoltenVK
  - KosmicKrisp
- Leading KosmicKrisp initiative since November 2024

# Agenda

- What is KosmicKrisp?
- “Production Ready!”?
- Conformance
- Performance
- CI
- Tips&Tricks
- What’s next?
- Closing comments

# What is KosmicKrisp?

In short:

- Vulkan conformant Vulkan-to-Metal driver
- macOS 26+
- Apple Silicon aka M-chip families and A18 Pro (macBook Neo)

For detailed information refer to previous year's presentation

<https://www.youtube.com/watch?v=bkKAQVSlGfY>

# So what's "Production Ready!"?

- An ask for adoption by developers
- Why?
  - KosmicKrisp has reached the point where it is ready for adoption
  - More applications to test -> easier to find gaps -> better KosmicKrisp
- What changed to be ready?
  - Conformance
  - Performance
  - CI

# Conformance

- Now Vulkan 1.4 conformant!
  - Community contributions helped greatly (@squidbus)!
- Support for considerable non-core extensions and features
  - Tessellation
  - Robustness2
  - Multi draw
- More workarounds for Metal bugs
  - MSL compiler being the biggest offender
- Support for all Apple Silicon hardware
  - M1, M2, M3, M4 and M5
  - A18 Pro (macBook Neo)

# KosmicKrisp since the Mesa merge: +79 extensions, +104 features



## Vulkan 1.0 core

alphaToOne, depthBiasClamp, dualSrcBlend, fragmentStoresAndAtomics, fullDrawIndexUint32, independentBlend, inheritedQueries, largePoints, logicOp, multiDrawIndirect, multiViewport, occlusionQueryPrecise, sampleRateShading, shaderClipDistance, shaderCullDistance, shaderImageGatherExtended, shaderStorageImageExtendedFormats, shaderStorageImageReadWithoutFormat, shaderStorageImageWriteWithoutFormat, shaderTessellationAndGeometryPointSize, tessellationShader, textureCompressionBC, vertexPipelineStoresAndAtomics

## AMD\_buffer\_marker

## AMD\_shader\_image\_load\_store\_lod

EXT\_4444\_formats: formatA4R4G4B4, formatA4B4G4R4

## EXT\_attachment\_feedback\_loop\_dynamic\_state

## EXT\_attachment\_feedback\_loop\_layout

## EXT\_blend\_operation\_advanced:

advancedBlendCoherentOperations

## EXT\_buffer\_device\_address

EXT\_conditional\_rendering: conditionalRendering, inheritedConditionalRendering

## EXT\_custom\_resolve

## EXT\_debug\_marker

## EXT\_depth\_clip\_control

## EXT\_depth\_clip\_enable

EXT\_extended\_dynamic\_state2: extendedDynamicState2, extendedDynamicState2PatchControlPoints

EXT\_extended\_dynamic\_state3: extendedDynamicState3: TessellationDomainOrigin, DepthClampEnable, DepthClipEnable, SampleLocationsEnable, ProvokingVertexMode, LineRasterizationMode, DepthClipNegativeOneToOne

## EXT\_external\_memory\_host

## EXT\_global\_priority\_query

## EXT\_hdr\_metadata

## EXT\_host\_image\_copy

EXT\_image\_2d\_view\_of\_3d: image2DViewOf3D, sampler2DViewOf3D

EXT\_map\_memory\_placed: memoryMapPlaced, memoryUnmapReserve

## EXT\_memory\_budget

## EXT\_multi\_draw

EXT\_nested\_command\_buffer: nestedCommandBuffer, nestedCommandBufferRendering, nestedCommandBufferSimultaneousUse

## EXT\_pipeline\_robustness

## EXT\_post\_depth\_coverage

## EXT\_primitive\_restart\_index

## EXT\_primitive\_topology\_list\_restart

EXT\_provoking\_vertex: provokingVertexLast

## EXT\_sample\_locations

EXT\_shader\_atomic\_float: shaderBufferFloat32Atomics, shaderBufferFloat32AtomicAdd, shaderSharedFloat32Atomics, shaderSharedFloat32AtomicAdd

## EXT\_shader\_stencil\_export

## EXT\_shader\_subgroup\_ballot

## EXT\_shader\_subgroup\_vote

## EXT\_shader\_viewport\_index\_layer

## EXT\_texel\_buffer\_alignment

## EXT\_texture\_compression\_astc\_hdr

## EXT\_vertex\_attribute\_robustness

## EXT\_ycbcr\_2plane\_444\_formats

KHR\_calibrated\_timestamps +EXT

## KHR\_deferred\_host\_operations

## KHR\_draw\_indirect\_count

## KHR\_dynamic\_rendering\_local\_read

## KHR\_external\_fence\_fd

## KHR\_external\_semaphore\_fd

KHR\_global\_priority +EXT: globalPriorityQuery

KHR\_index\_type\_uint8 +EXT

KHR\_line\_rasterization +EXT: bresenhamLines

KHR\_load\_store\_op\_none +EXT

## KHR\_maintenance5

## KHR\_maintenance6

## KHR\_maintenance7

## KHR\_maintenance8

## KHR\_maintenance9

## KHR\_maintenance10

## KHR\_map\_memory2

## KHR\_present\_id

## KHR\_present\_id2

## KHR\_present\_wait

## KHR\_present\_wait2

KHR\_robustness2 +EXT: robustBufferAccess2, robustImageAccess2, nullDescriptor

## KHR\_sampler\_mirror\_clamp\_to\_edge

## KHR\_shader\_draw\_parameters

KHR\_shader\_float16\_int8: shaderFloat16, shaderInt8

## KHR\_shader\_float\_controls2

KHR\_shader\_fma: shaderFmaFloat16, shaderFmaFloat32

KHR\_shader\_subgroup\_rotate: shaderSubgroupRotate, shaderSubgroupRotateClustered

## KHR\_shader\_untyped\_pointers

KHR\_swapchain\_maintenance1 +EXT

## KHR\_unified\_image\_layouts

KHR\_vertex\_attribute\_divisor +EXT:

vertexAttributeInstanceRateDivisor, vertexAttributeInstanceRateZeroDivisor

## KHR\_workgroup\_memory\_explicit\_layout:

workgroupMemoryExplicitLayout, workgroupMemoryExplicitLayoutScalarBlockLayout, workgroupMemoryExplicitLayout8BitAccess, workgroupMemoryExplicitLayout16BitAccess

## Existing exts, new features

## EXT\_descriptor\_indexing:

shaderUniformBufferArrayNonUniformIndexing, shaderStorageImageArrayNonUniformIndexing, shaderStorageTexelBufferArrayNonUniformIndexing

EXT\_extended\_dynamic\_state: extendedDynamicState

KHR\_push\_descriptor: pushDescriptor

➤ Required for Vulkan 1.4   ■ Notable   name: its new features   +EXT: same extension also exposed under its EXT name



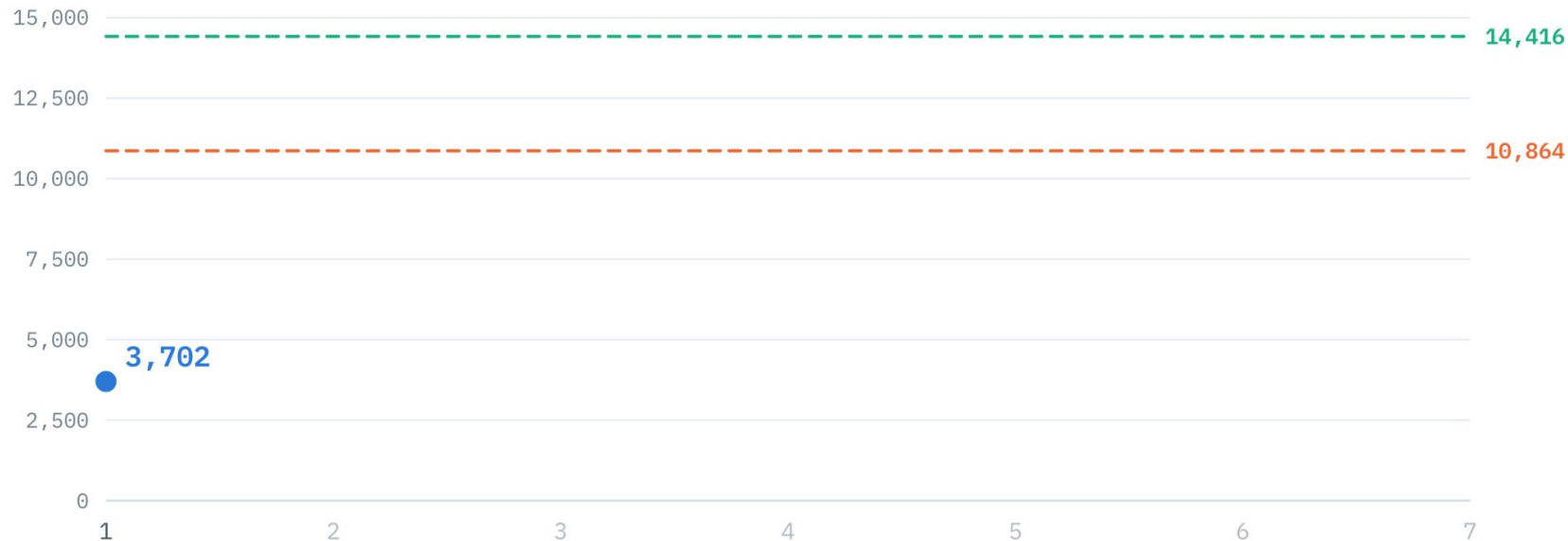
## KosmicKrisp on Aztec Ruins

High · 2560×1440 · GFXBench 5 offscreen · Apple M3 Pro · median of 10 runs

### 1. Baseline

19 Jun · 56588ef0665

— KosmicKrisp — MoltenVK — Native Metal



## KosmicKrisp on Aztec Ruins

High · 2560×1440 · GFXBench 5 offscreen · Apple M3 Pro · median of 10 runs

### 2. Fast math

2.33× baseline · 8,622

this step: +132.9%

— KosmicKrisp — MoltenVK — Native Metal



# Community contribution by @k0bin



KosmicKrisp

## KosmicKrisp on Aztec Ruins

High · 2560×1440 · GFXBench 5 offscreen · Apple M3 Pro · median of 10 runs

### 3. Dynamic store op

2.46× baseline · 9,101

this step: +5.6%

— KosmicKrisp — MoltenVK — Native Metal



## KosmicKrisp on Aztec Ruins

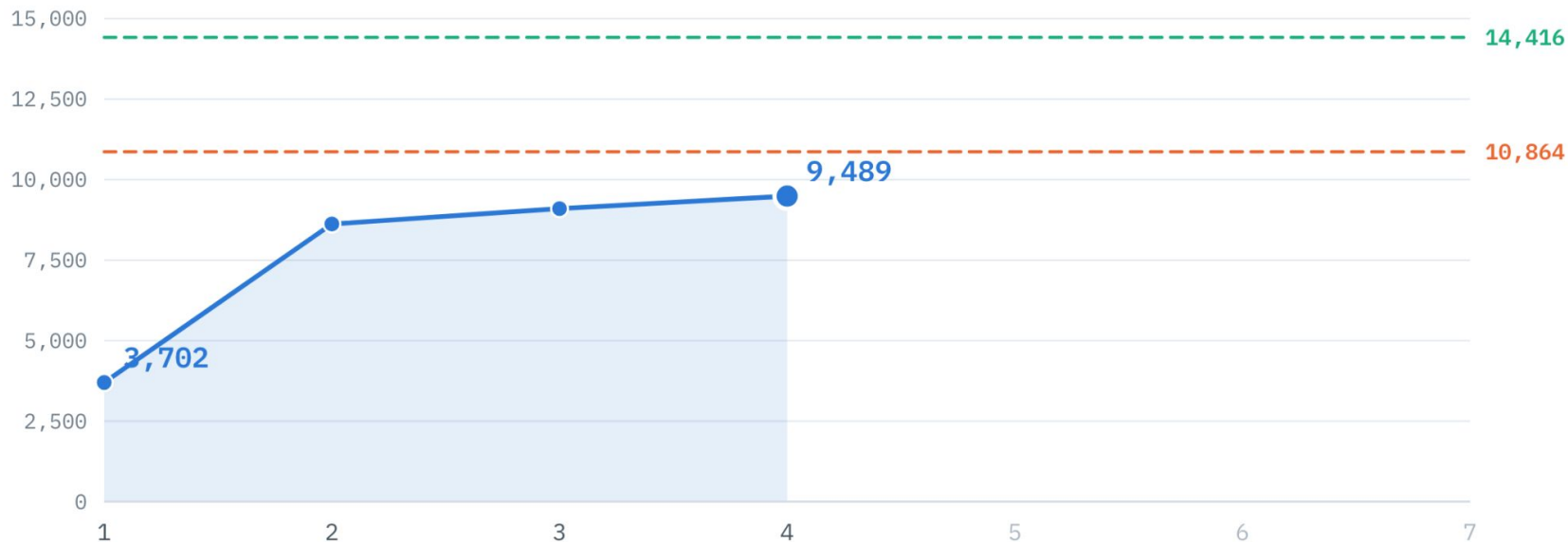
High · 2560×1440 · GFXBench 5 offscreen · Apple M3 Pro · median of 10 runs

### 4. Triangle-merge rework

2.56× baseline · 9,489

this step: +4.3%

— KosmicKrisp — MoltenVK — Native Metal



## KosmicKrisp on Aztec Ruins

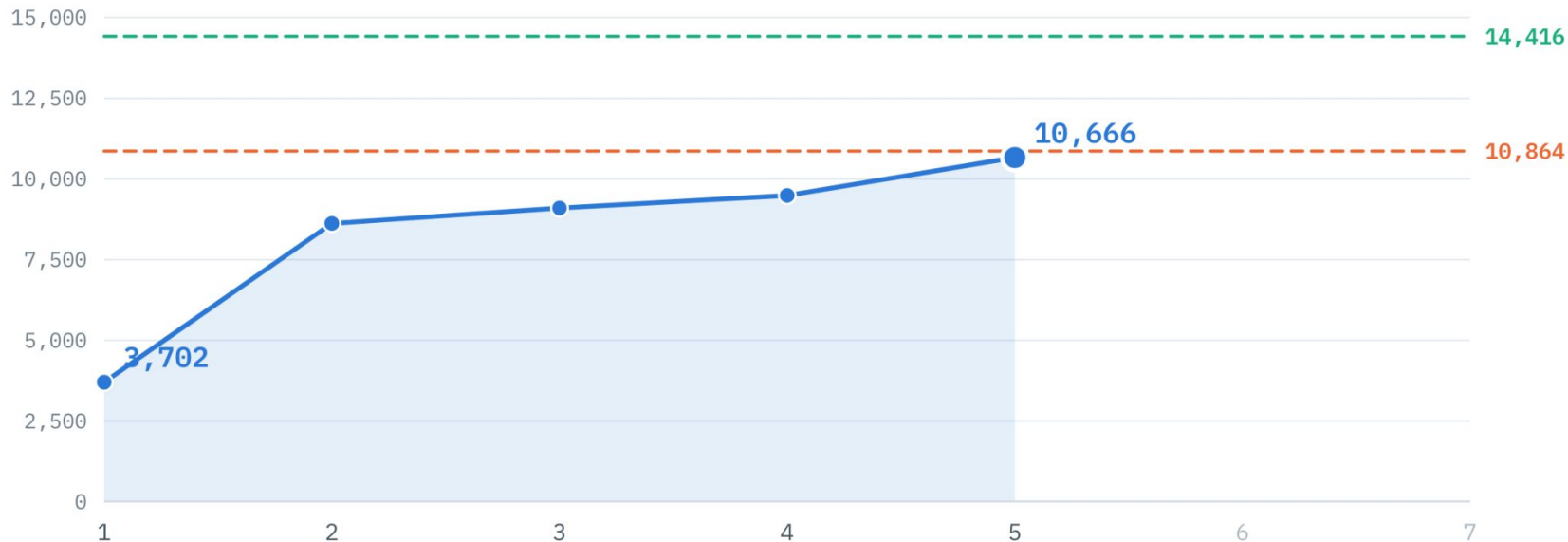
High · 2560×1440 · GFXBench 5 offscreen · Apple M3 Pro · median of 10 runs

### 5. Depth-write narrowing

2.88× baseline · 10,666

this step: +12.4%

— KosmicKrisp — MoltenVK — Native Metal



## KosmicKrisp on Aztec Ruins

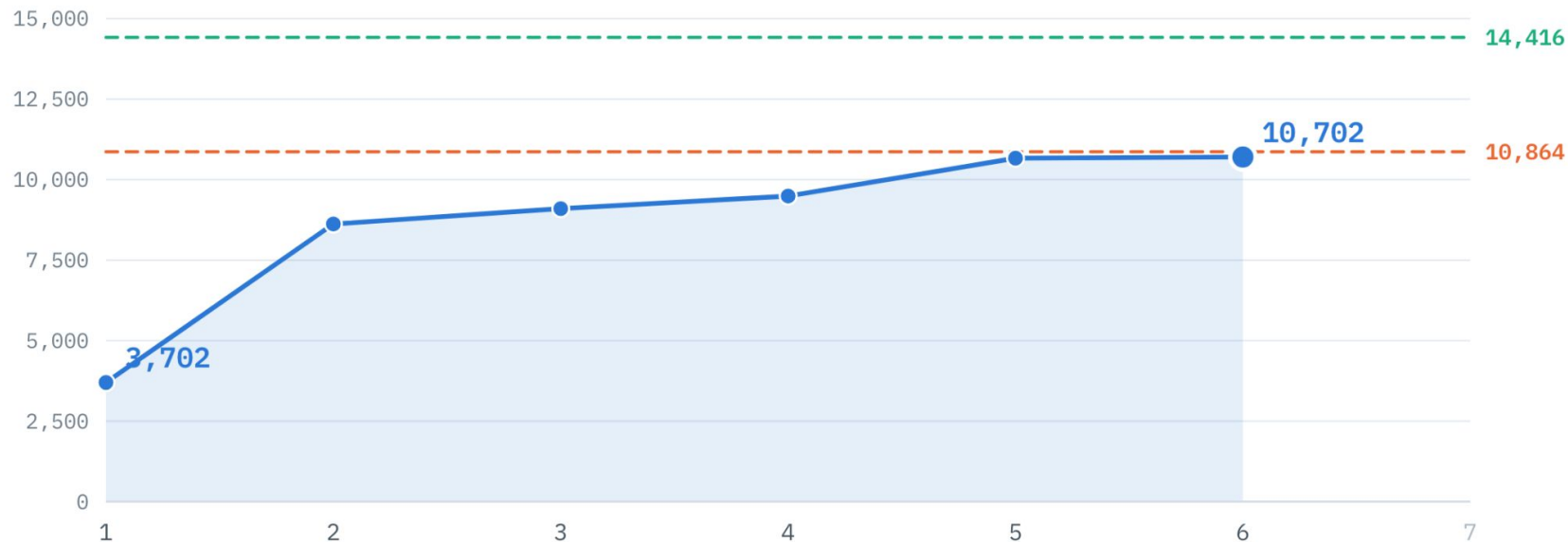
High · 2560×1440 · GFXBench 5 offscreen · Apple M3 Pro · median of 10 runs

— KosmicKrisp — MoltenVK — Native Metal

### 6. Constant data

2.89× baseline · 10,702

this step: +0.3%



## KosmicKrisp on Aztec Ruins

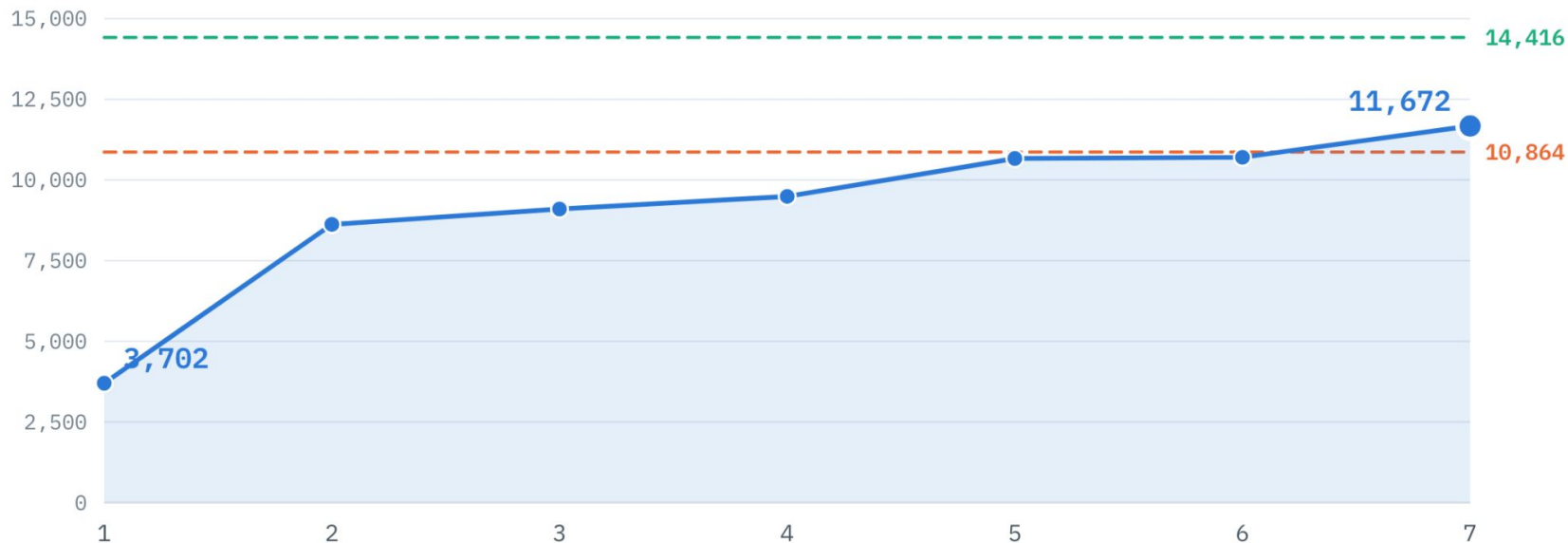
High · 2560×1440 · GFXBench 5 offscreen · Apple M3 Pro · median of 10 runs

### 7. Sync rework

3.15× baseline · 11,672

this step: +9.1%

— KosmicKrisp — MoltenVK — Native Metal



# Performance, where do we stand?

| Tested on an M3 Pro<br>(macOS 27)             | Metal | MoltenVK | KosmicKrisp<br>(% of MoltenVK) |
|-----------------------------------------------|-------|----------|--------------------------------|
| GFXBench<br>aztec_high                        | 14416 | 10864    | 11672 (107.4%)                 |
| GFXBench<br>aztec_normal                      | 38274 | 26991    | 27747 (102.8%)                 |
| 3DMark WildLife<br>(Android emulator)         | NONE  | 29893    | 29595 (99%)                    |
| 3DMark WildLife Extreme<br>(Android emulator) | NONE  | 11090    | 10595 (95.54%)                 |

# Performance, are we done?

- No, we are not
- Mainly tested with GFXBench (and 3DMark)
  - Similar rendering patterns
  - Vulkan 1.1 usage ("old")
- Started testing with other applications
  - Godot Third-Person Shooter (found shader constant data issue!)
  - Blender
  - Apps we can hook KosmicKrisp where MoltenVK is already in use
  - Developer provided builds of their applications

# KosmicKrisp's CI

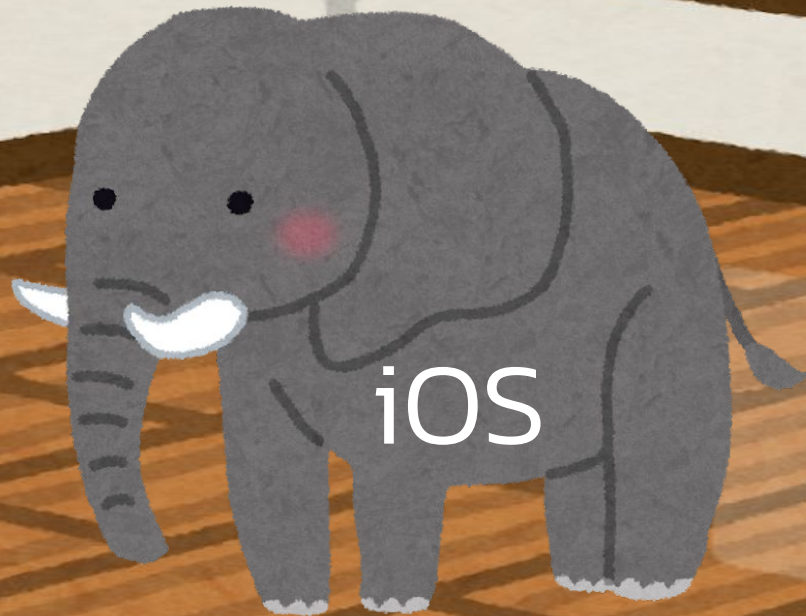
- LunarG principle: good software requires good CI
  - Ensure regression testing and driver quality as codebase evolves
- Run within LunarG, not part of Mesa's CI
  - M1, M2 and M4 machines with macOS 26
- Nightly Vulkan CTS runs
  - vulkan-cts-1.5.2
- Nightly application (real workload) testing
  - Capture replays through GFX Reconstruct
  - Extensive collection of applications through the Android Emulator
- Nightly Fossilize runs
  - Collection of pipelines that caught driver compiler issues in the past

# Tips & tricks

- Avoid breaking render passes by not doing the following
  - Local read with multisample and depth attachments
    - No way to specify sample ID for framebuffer fetch
    - Framebuffer fetch for depth attachment is not possible, use it as color!
  - Unrolling, tessellation and indirect draw count (for now)
    - Render pass broken to spawn a compute dispatch
    - Plan to move them to mesh shaders and vertex dispatch to avoid break
  - Switching enablement of Bresenham lines
    - Requires changing sample data (can only be accomplished by restarting pass)
- Use `ONE_TIME_SUBMIT` when recording command buffers
  - Directly records Metal commands instead of at queue submission

# What's next?

- More performance
  - Increase application coverage and investigate gaps
  - Move certain features to mesh shaders
- More extensions/features
  - Geometry shaders
  - Descriptor heaps
  - Shader objects
- Finish MoltenVK macOS parity (just a few low impact items)
- Improved tooling support (RenderDoc, GFXReconstruct, VVL best practices...)



# The elephant in the room

- As of now, no iOS support
- We could enable builds and hope for the best, but
  - LunarG principle, good software requires good CI
  - Enabling builds means maintaining iOS: Resource constrained
- What is needed?
  - Upstream changes to enable Vulkan CTS iOS builds
  - Devices on CI
  - Content for regression testing
  - Find disparities between M-chips and A-chips
  - **Additional resources (iOS ownership/maintainer needed)**

# We were curious about KK on iOS

| Tested Vulkan CTS on an M5 iPad | MoltenVK | KosmicKrisp     |
|---------------------------------|----------|-----------------|
| Passed                          | 544,622  | 560,355         |
| Failed                          | 14,230   | 80              |
| Crashes                         | 28,646   | 1 (GPU timeout) |

- Current KK on iOS has better quality than MoltenVK
- Applications were also tested in addition to CTS

# Closing comments

- Start using KosmicKrisp! You may get hooked to supporting a single API across all platforms (ignore the elephant)
- KosmicKrisp works around Metal bugs for you
- You break it, we fix it
- Your application runs slower with KosmicKrisp than MoltenVK?
  - Happy to take a look to identify bottlenecks in KosmicKrisp
- Provided as part of Mesa releases and Vulkan macOS SDK
  - New Vulkan macOS SDK just released with brand new shiny KosmicKrisp
- If you don't know how to use KosmicKrisp, refer to:
  - Paper: <https://www.lunarg.com/the-state-of-vulkan-on-apple-jan-2026/>
  - Video: [https://www.youtube.com/watch?v=IOSu\\_JcbHkA](https://www.youtube.com/watch?v=IOSu_JcbHkA)

# Questions?

---

Get this Presentation



KosmicKrisp Information

