Vulkanised 2023

The 5th Vulkan Developer Conference
Munich, Germany / February 7–9

Using ANGLE as a System Graphics Driver

Gabe Dagani
Director GPU SW
Samsung Austin Research Center

Agenda

1. ANGLE
2. ANGLE instead of native GLES
3. The Productization Journey
4. VK Extensions and GLES 3.2
5. The Performance Story
6. Replacing GLSLANG
7. Future Work

Acknowledgements

- Google ANGLE and Android Teams
- Samsung ANGLE Team (US, UK, India, Korea)

SAMSUNG

What is ANGLE?

Almost Native Graphics Layer Engine

Mobile graphics engine which translates OpenGL ES to multiple output APIs

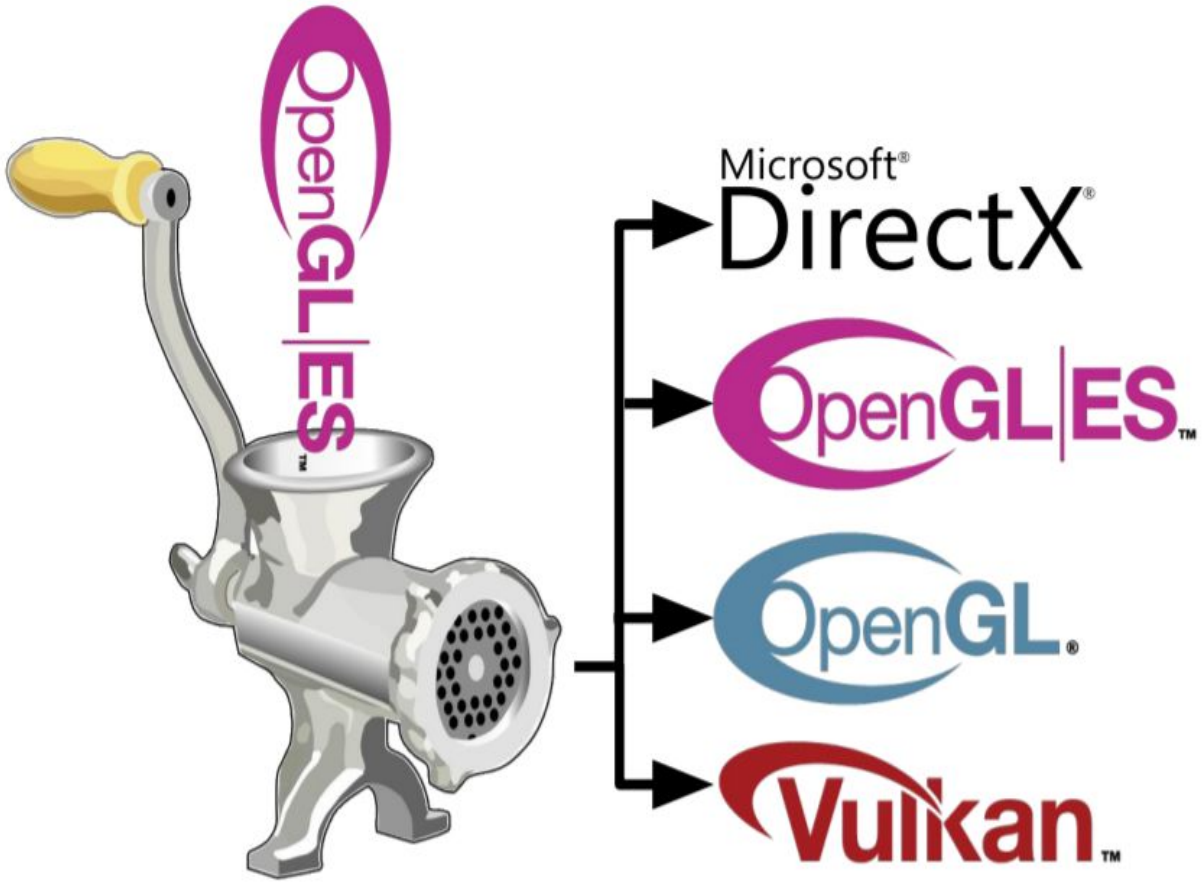


Image permission provided by Jamie Madill (Google)

SAMSUNG

What is ANGLE?

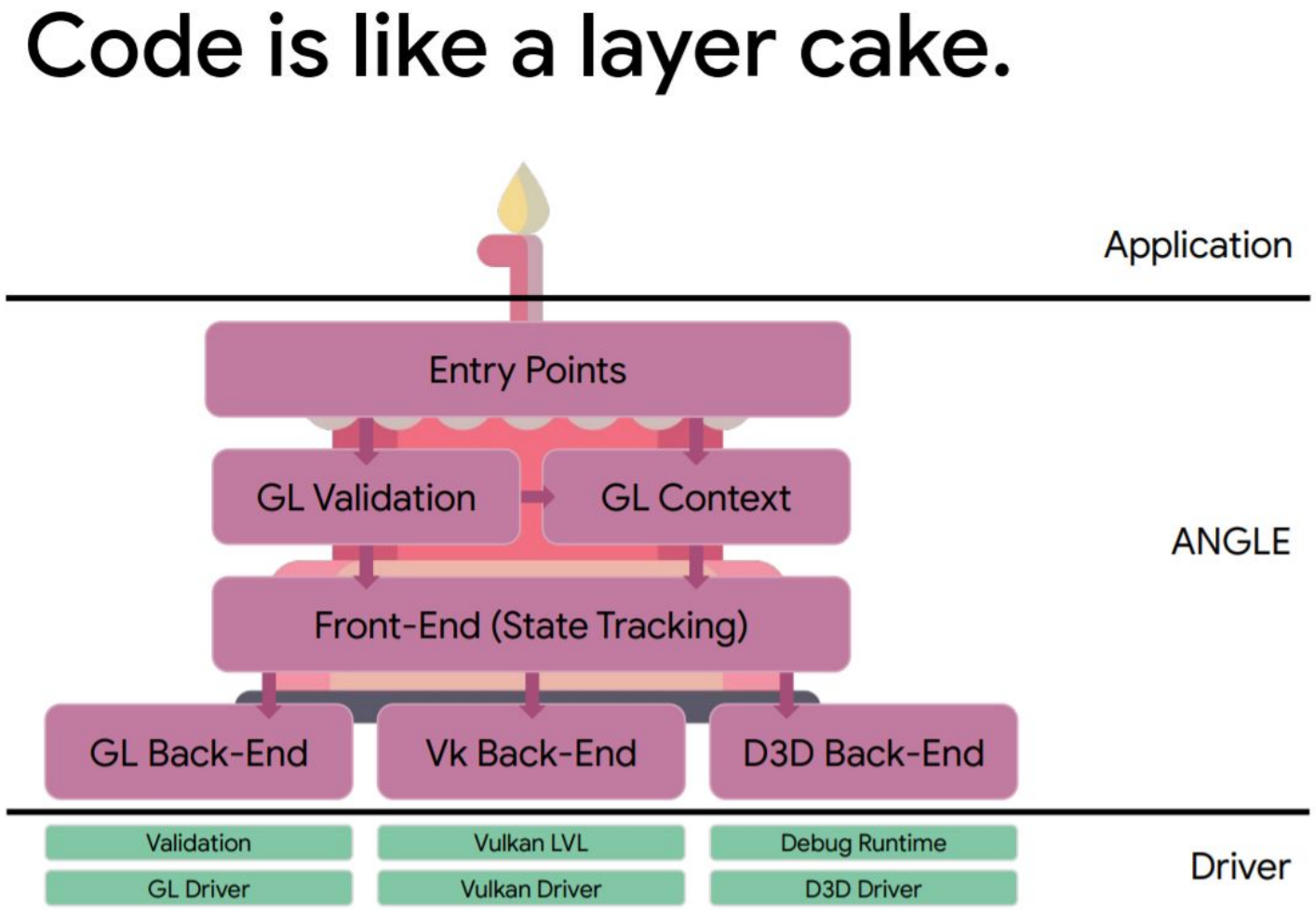


Image permission provided by Jamie Madill (Google)

ANGLE instead of a Native OpenGL ES Driver

Benefits of using ANGLE

Reduces GLES "Total Cost of Ownership"

Allows IHVs to focus all gfx efforts on Vulkan

- HW and FW updates serve GLES and VK API simultaneously
- Most new features are going to be available in VK only

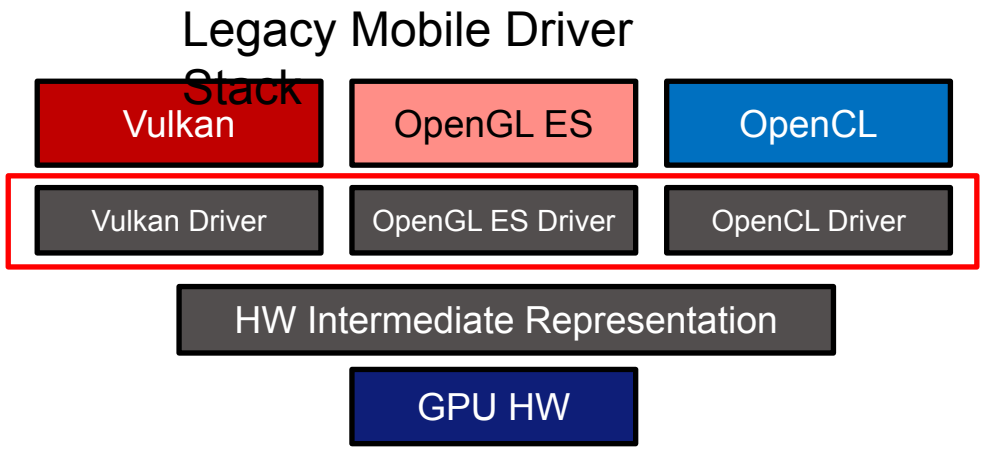
Vulkan can improve some GLES intrinsically.

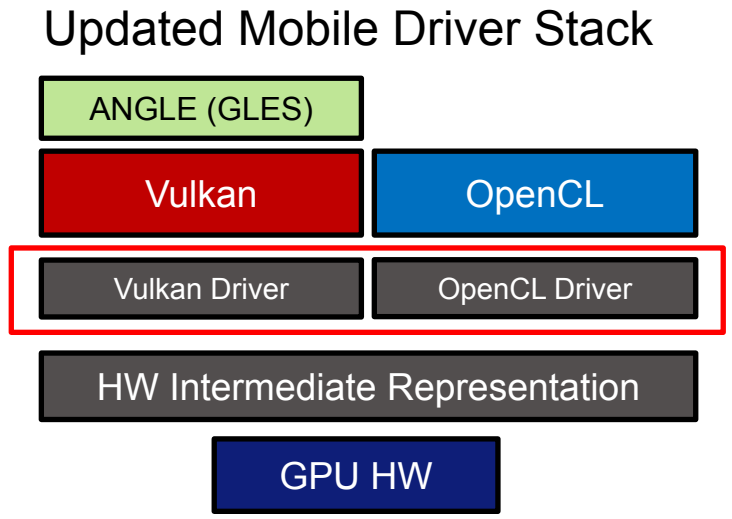
- Mipmap generation improvements
- Finer grained barriers
- Pipeline caching

Conformance debugging of GLES is simplified

- Run ANGLE on multiple drivers (Vendor Driver / Swiftshader) for process of elimination

Better Security: not having an GLES native driver reduces the attack surface for the platform.





ANGLE instead of a Native OpenGL ES Driver

Costs of Using ANGLE

Not all GLES features can be directly translated to Vulkan and require emulation.

- Private extensions are required until Vulkan APIs can be developed and approved.
- Example: YUV extensions for render target are not available yet in Vulkan API

Performance overhead can reduce application framerate.

- This overhead can be absorbed from hardware year-over-year improvement (YOY)
- See the following slide.

Pipeline creation introduces additional delays and startup costs

- Vulkan APIs restricts pipeline creation until drawcall.
- Therefore pipeline caches need to be warmed up in both ANGLE and VK layers.

The Productization Journey

DEQP Pass Rate

Where we started from Circa 2019

- deqp 2.0 pass rate ~91%
- deqp 3.0 pass rate ~16%
- deqp 3.1 pass rate ~5%
- EGL pass rate ~16%

Initial Perf Median at 50% native

Android N (2018 Device)

- Trex onscreen
  - Native GLES – ~60 FPS
  - ANGLE – ~25 FPS
- Manhattan 3.0 crashed
  - No PBO support

The Productization Journey

Where we ended up 2022

- Khronos dEQP 3.2 conformant (100%)

Samsung Electronics 2022-02-12 OpenGL_ES_3_2	975
Xclipse 920	CPU: ARMv8 OS: Android 12.0 API Pipeline: OpenGL ES 3.2 gl_renderer: ANGLE (Samsung Xclipse 920) on Vulkan 1.1.179 gl_shading_language_version: OpenGL ES GLSL ES 3.20 gl_version: OpenGL ES 3.2

SAMSUNG

Vulkan Extensions to Support GLES 3.2

Public Extensions

- VK_EXT_line_rasterization
- VK_EXT_provoking_vertex
- VK_KHR_create_renderpass2
- VK_KHR_fragment_shading_rate
- VK_KHR_incremental_present
- VK_ANDROID_external_memory_android_hardware_buffer
- VK_KHR_sampler_ycbcr_conversion
- VK_EXT_transform_feedback
- VK_EXT_custom_border_color
- VK_EXT_swapchain_colorspace
- VK_KHR_image_format_list
- VK_KHR_surface_protected_capabilities
- VK_KHR_shared_presentable_images
- VK_EXT_depth_clip_control
- VK_EXT_device_memory_report
- …

Vulkan Work-Arounds

- Support for 2D image slices of 3D images to enable EGL_KHR_gl_texture_3D_image
- Support for YUV images as render targets
- Avoided swap-chain acquisition when rasterizer discard is enabled.
- Vulkan pipeline cache statistics feedback to better support ANGLE cache management

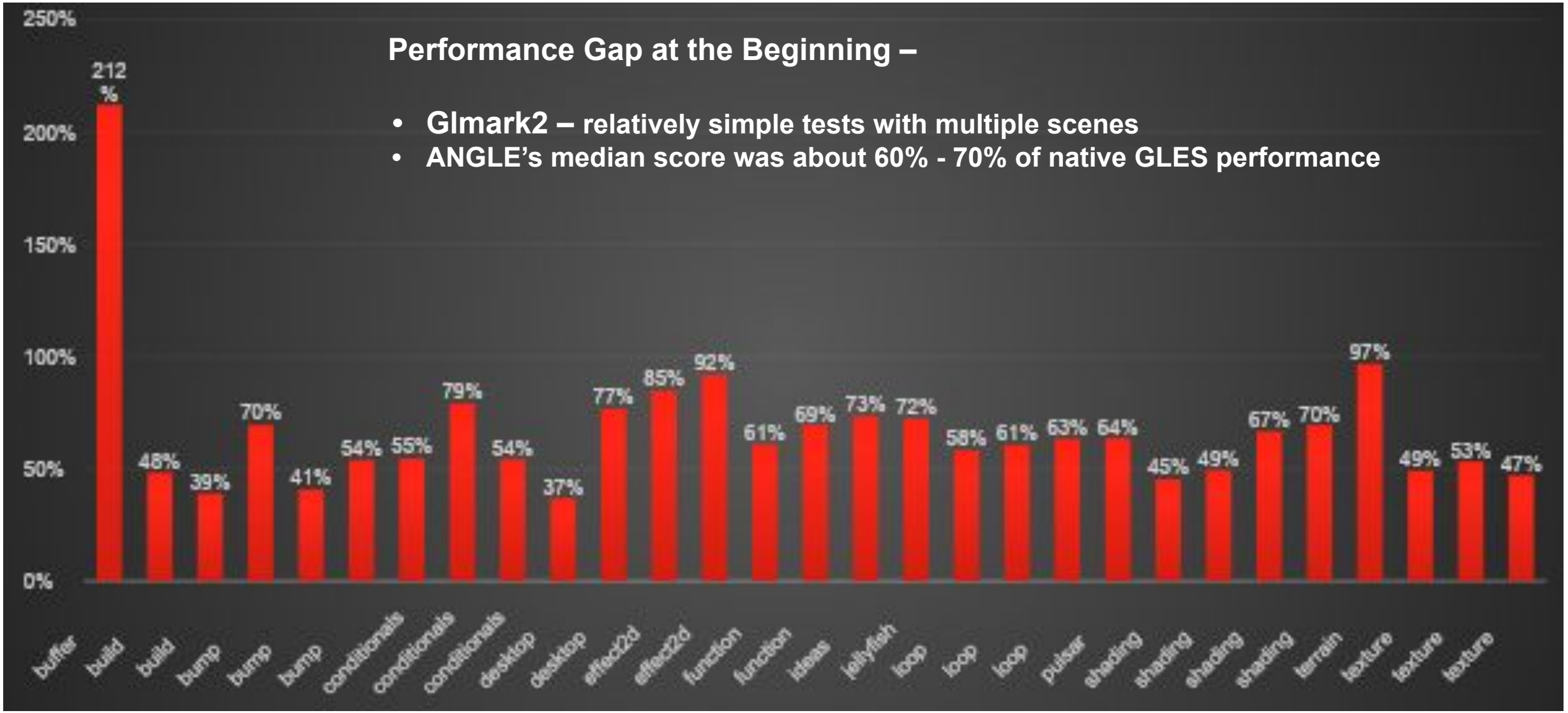
Real World App testing with ANGLE Traces

175+ Traces

Image Courtesy of Geoff Lang (Google)

SAMSUNG

The Performance Story



SAMSUNG

The Performance Story

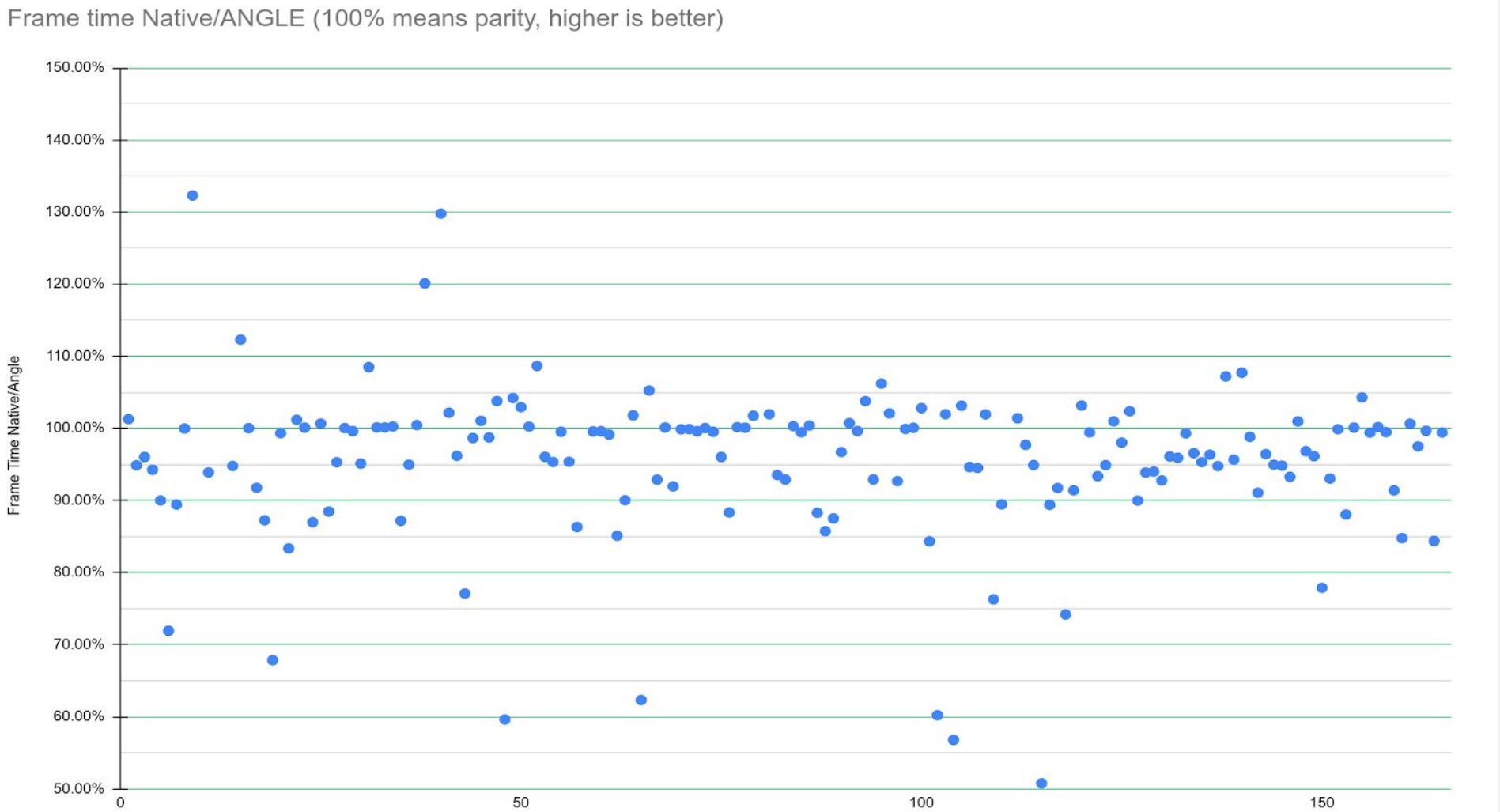
Addressing the performance gaps

- Finer-grained locks (not global) for API entry points
- Mem-pooling and sub-allocations for buffer storage objects
- Early pipeline compilation with “default” state, based on heuristics
- Fast mipmap generation using AMD’s SPD library
- Avoid “vkCmdClearDepthStencilImage” when possible
- Larger pipeline caches supporting LRU-based eviction
- Propagate improved precision of shader-variables
- Remove duplicate texture updates.
- Deferred acquisition of swap-chain images

The Performance Story

Where did we end up?

- ANGLE has about 150 app traces
- Each blue dot is an app trace
- ANGLE perf_test suite provides data about time taken per frame
- 100% means parity with native GLES
  - higher is better



SAMSUNG

The Performance Story

Where did we end up?

- ANGLE has about 175 app traces
- Each dot is an app trace
- ANGLE perf_test suite provides data about time taken per frame
- 100% means parity with native GLES (higher is better)

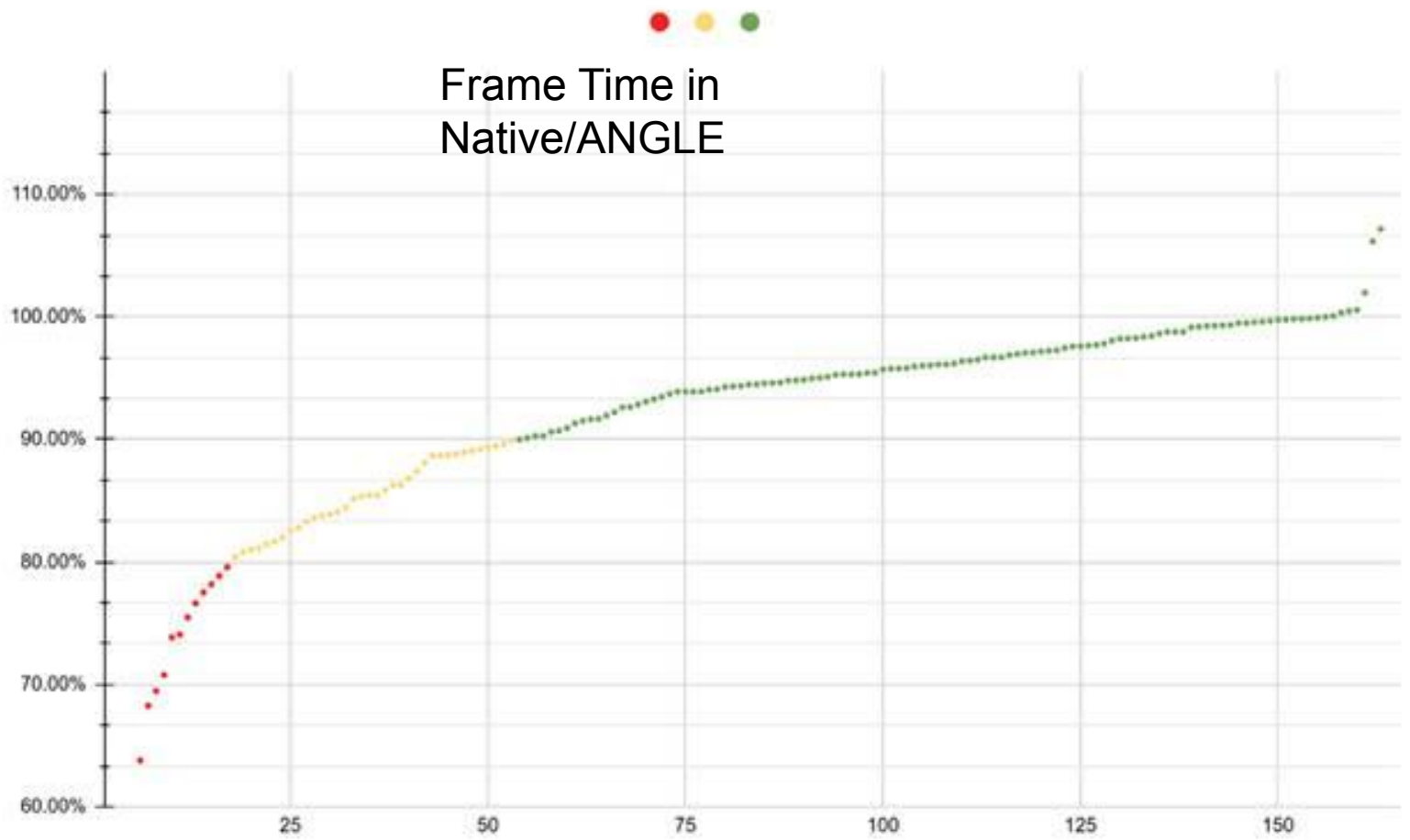


Chart provided by Charlie Lao (Google)

Data is about 3-4 months old

Replacing GLSLANG

Why was this done?

- Old flow with **glslang**:
  - ESSL □ Vulkan compatible GLSL AST □ GLSL String □ glslang □ AST □ SPIR-V
- Flow with **directSpirvGen**:
  - ESSL □ Vulkan compatible GLSL AST □ SPIR-V

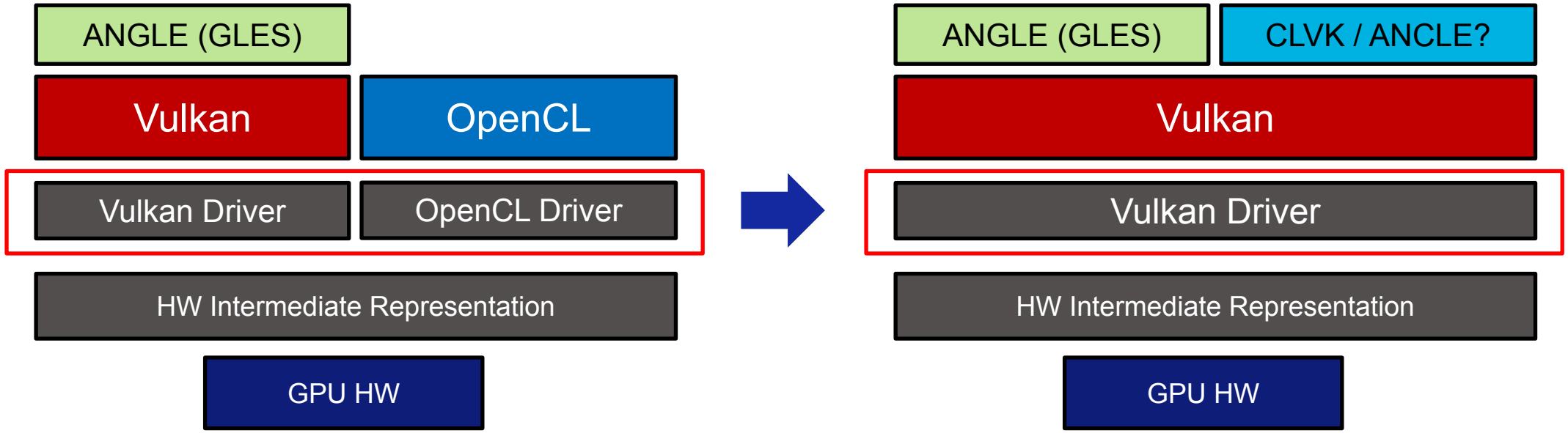
What was the benefit?

- De-duplication of AST transformations
- Removing **glslang** also reduced app startup time because glslang had a large warm-up time.
  - Removed ~20ms of warm-up time for glslang on app start.
- Reduced code size of ANGLE library and decoupled code-churn of the external module
  - Up to 3X reduction in compile time for complex shaders

Future Work

ANGLE and ANDROID

- Progressing towards **Single API** graphics ecosystem
- Removing additional parts for driver/OS upgrade
- Decouple GLES enhancements and bug fixes from OTA graphics driver updates



Q&A