Graphics virtualisation for automotive

Daniel Stone
Graphics Lead, Collabora @ London, UK
daniels@collabora.com

COLLABORA

Hi, I’m Daniel

Graphics lead at Collabora (2008)
Open-source mainline graphics
Mesa, Wayland, Linux kernel
Collabora are core VirGL engineers

Open First

Agenda points

- Automotive requirements for virtualised graphics
- Design and current status of mainline VirGL solution
- Summary of different virtualisation approaches
- Potential future developments
- Open discussion forum

Automotive requirements

Need for graphics virtualisation

- Instrument cluster and IVI displays both require advanced graphics functionality
- Functional integration and BoM requirements for **single-silicon approach**
- Graphics functionality must have **high assurance level** for safety certification requirements
- Graphics functionality must have **high performance level** for OEM and end-user requirements
- Architecture and platform must be **long-term sustainable**

Need for graphics virtualisation

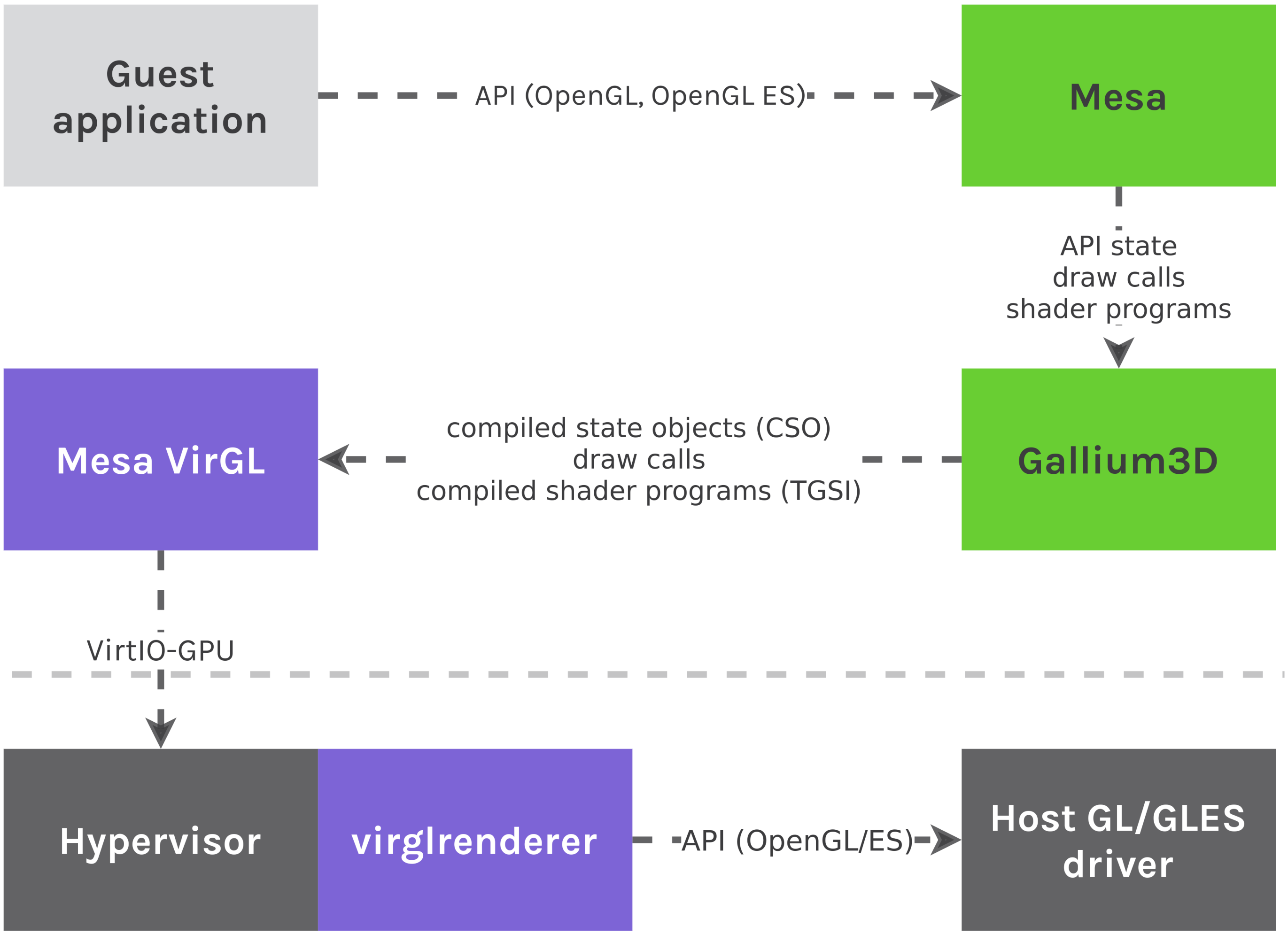
- And a lot more besides ... but the expert group are already the experts on this topic :)

VirGL architecture and status

VirGL design and architecture

- Three core VirGL components
- **Guest**
  - Mesa-based, provides OpenGL / GLES / EGL
  - Compresses GL commands into efficient pipelines
- **Transport**
  - VirtIO-GPU protocol transports commands, shaders
- **Host**
  - Translates VirtIO-GPU stream into replayable commands
  - virglrenderer translates into GL/GLES

VirGL design



VirGL architecture: guest

- Using industry-standard Mesa/Gallium3D framework
- Mesa high level implements OpenGL / GLES / EGL APIs
- ‘state tracker’ translates Khronos API into Gallium3D ‘pipe’
- Gallium3D tracks verbose OpenGL state, compresses into persistent state objects (shaders, blend state, etc)
  - Similar to Vulkan (VkPipeline) approach: efficient for drivers and hardware
- Draw calls reference state objects
- Compiles shaders to TGSI intermediate representation
- Submits state objects and draw calls to VirGL

VirGL architecture: VirtIO-GPU

- OASIS standard transport layer using VirtIO
- Handles memory allocation, object tracking, command execution, synchronisation
- Guest allocates and transfers state objects, host tracks allocations
- Guest binds state objects to context
- Guest submits draw commands
- Host executes draw commands
- Guest can synchronise against command completion

VirGL architecture: host

- virglrender library used by hypervisor (QEMU, crosvm)
- Hypervisor implements resource and context allocation
- virglrenderer translates efficient VirtIO protocol into OpenGL / GLES commands
- Effectively mirrors guest commands by replaying them on top of existing driver
- Works with GL/GLES conformant host drivers (Arm Mali, Mesa, NVIDIA, etc)

VirGL status

- Implements OpenGL 4.3, OpenGL ES 3.2
  - Support for OpenGL 4.5 in development
- May provide layered support beyond what host provides, e.g. OpenGL guest on OpenGL ES host
- Focused performance work has provided massive improvement
  - AAA game & industry benchmark workloads at 70-80% of native performance
- Shipped as part of ChromeOS for guest environments, with Rust-based crosvm hypervisor

VirGL status

- Supports complicated mixed environments
- Wayland fully supported in both host and guest environments
- Possible to integrate guests seamlessly into host window system without explicit host knowledge
- Support for complicated guest window systems (e.g. full Android or native UI environment, browsers with WebGL)

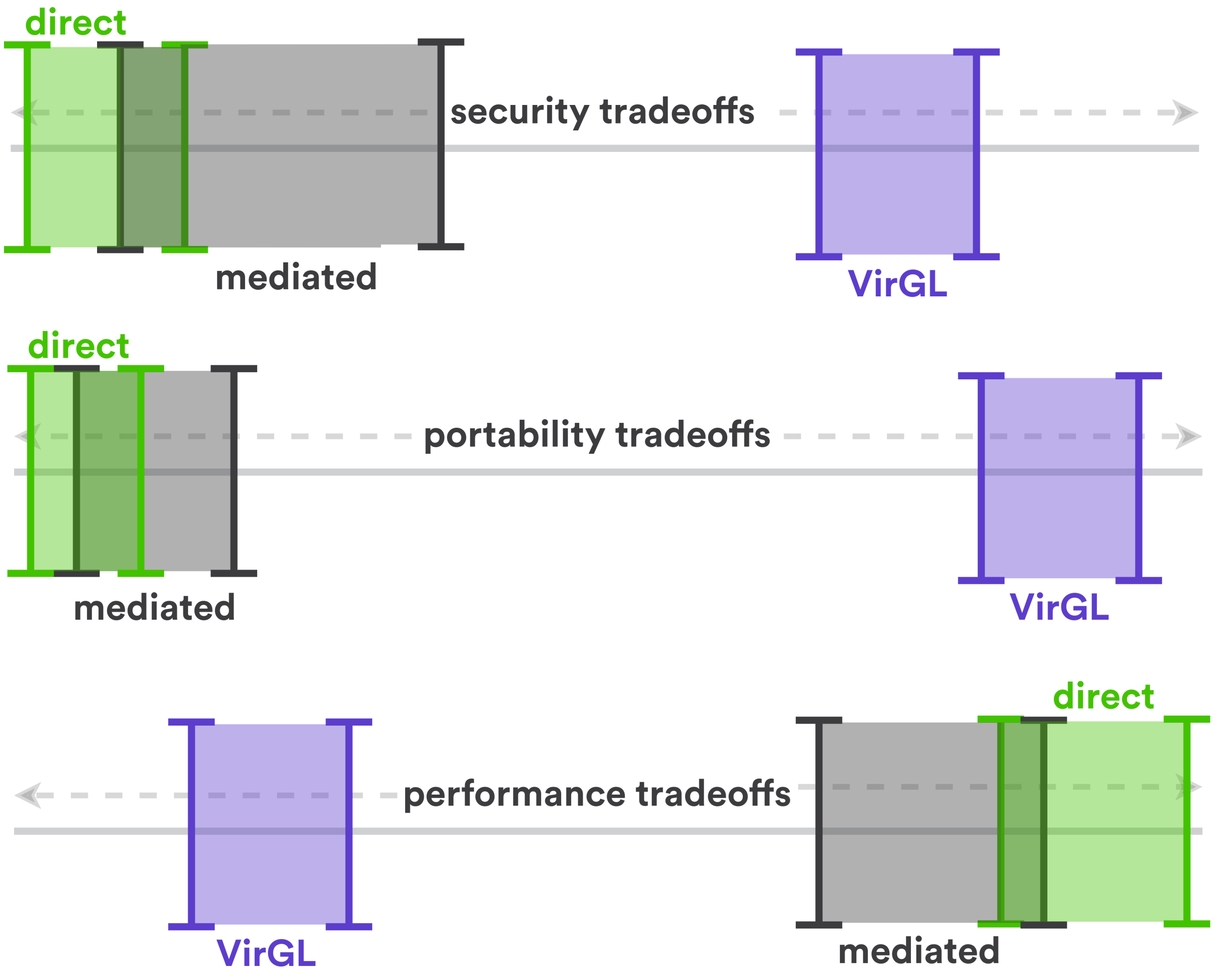
COLLABORA

Alternative approaches

Overview of alternative approaches

- Three primary approaches to virtualised graphics
- **Direct hardware access**: guest has full access to hardware
  - Most performant solution; little or no performance penalty to native access
  - Relies on hardware vendor security (e.g. hard partitioning between worlds)
- **Mediated hardware access**: host provides limited access
  - Intermediate performance: commands, events must pass through translation layer
  - Reliant on vendor security of both raw hardware and hypervisor/kernel translation
- **API layering**: host provides no hardware access
  - Lower performance compared to hardware options
  - Not reliant on hardware security: security is enforced via Khronos APIs
  - Maximum compatibility

Tradeoffs of graphics virtualisation designs



Alternate approaches in the market

- Direct hardware access
  - Offered by Imagination PowerVR RGX; hardware partitioning enforced via closed-source vendor microcode
  - Arm Mali suggesting this approach is now possible with Valhall
- Mediated hardware access
  - Offered by Intel GVT-g; hardware context separation used with modifications to host & guest kernel drivers to forward commands
  - Similar design offered by NVIDIA/Xen
- API streaming
  - VirGL/VirtIO-GPU leading solution for GL/GLES on GL/GLES; Vulkan support in development

Potential future directions

Potential future directions

- Open solutions are important to Collabora and the industry
- OEMs and tier-1s must reconcile the challenge of diverse SKU portfolio vs. integrated platform architecture

- Our challenge – can we:
  - help vendors unlock product value with **high performance and functionality**
  - help OEMs and tier-1s reduce maintenance cost with **high portability**
  - help the open-source ecosystem with an **open community effort**

- So far these these tradeoffs have been **mutually exclusive**!

Future basis: graphics standards

- Vulkan and SPIR-V provide a strong baseline
  - Vulkan designed for **high performance**, unlocking hardware potential with **low overhead API**
  - Tight specification, strict conformance testing, and validation layers enable **high assurance** execution
  - SPIR-V provides tightly-specified **efficient intermediate transport**
  - Mature tooling provides **powerful development support**
  - Khronos standards governance ensures **open community foundation**
- These attributes improve on existing solutions in more than one dimension

Future basis: layering on Vulkan

- Using Vulkan as the baseline does not mean only exposing Vulkan
- Vulkan can serve as the basis for:
  - OpenGL / OpenGL ES: **Zink** (Collabora)
  - OpenCL: **clspv/clvk** (Google)
  - DirectX: **dxvk** (Valve)
- Use Vulkan as host baseline, other API support within guest

Vulkan: designed for portability and layering

Layers Over	Vulkan	OpenGL	OpenCL	OpenGL ES	DX12	DX9-11
Vulkan		Zink	clspv clvk	GLOVE Angle	vkd3d	DXVK WineD3D
OpenGL	gfx-rs Ashes			Angle		WineD3D
DX12	gfx-rs	Microsoft 'GLOn12'	Microsoft 'CLOn12'			Microsoft D3D11On12
DX9-11	gfx-rs Ashes			Angle		
Metal	MoltenVK gfx-rs		clspv over MoltenVK?	MoltenGL Angle		

Vulkan is effective porting layer for API portability and stack simplification

'Vulkan everywhere'! Even if no native drivers on platform

Working towards 'OpenCL Everywhere'!

Future basis: efficient transport

- io_uring kernel API added to allow hyper-efficient command transport between kernel and userspace
- Initially used for disk I/O operations but now seeing wider use
- Red Hat experimenting with bridging io_uring in guest kernel through VirtIO
- Graphics commands naturally expressed as command queue
- Can we build the most efficient transport?

Future basis: a new start?

- build on Vulkan and SPIR-V in the host for **low overhead**, **high assurance** execution
- virtualise io_uring as a **hyper efficient transport**
- offer Vulkan, OpenGL / GLES and other APIs in the guest for **maximum compatibility**
- build this as a **genuine community effort** involving all stakeholders
- achieve the best possible tradeoff between end user, product vendor, platform vendor, open-source community

Open discussion

Thank you!

daniels@collabora.com