SAMSUNG

Vulkan on Android: Benefits for mobile & getting stuff on screen

Alon Or-bach, Samsung Electronics

GDC 2016

Overview

- **Why Vulkan is great for Android**
  - Key features that will help games get more out of mobile GPUs
  - Important-to-understand differences for getting best performance on mobile
- **Vulkan hooking up with Android**
  - Vulkan loader and layers on Android
  - Vulkan presentation on Android
- **What next? + Q&A**

DISCLAIMER

Details of how Android support for Vulkan is structured are subject to change

SAMSUNG

Vulkan on Android: Key benefits for mobile developers

Alon Or-bach, Samsung Electronics

GDC 2016

We changed – now YOU need to change

- **Vulkan gives you control over when and how things happen**
- **Compared with GLES, you get**
  - better multithreading
  - better control over command buffers
  - better control over memory and the state it's in
  - better bandwidth efficiency
  - more predictability
- **But!**
- **Your application needs to be written to use all this control**
  - More work for you if you want performance (not all apps need this)
- **Old scalar code/poor resource management = GLES performance**
  - Actually, worse, because the driver won't optimise it for you
- **Or use a game engine...**

SAMSUNG

Multithreading

- **Modern mobiles have a lot of CPU power for generating commands**
  - It's common for mobile CPUs to have many cores
    - 8 on recent Samsung Exynos chips
  - You want to be able to have all those threads running on something useful asynchronously

- **Multithreading an application with OpenGL ES not always straightforward**
  - Extensions have improved this significantly
  - But not easy to cleanly farm out form across threads

SAMSUNG

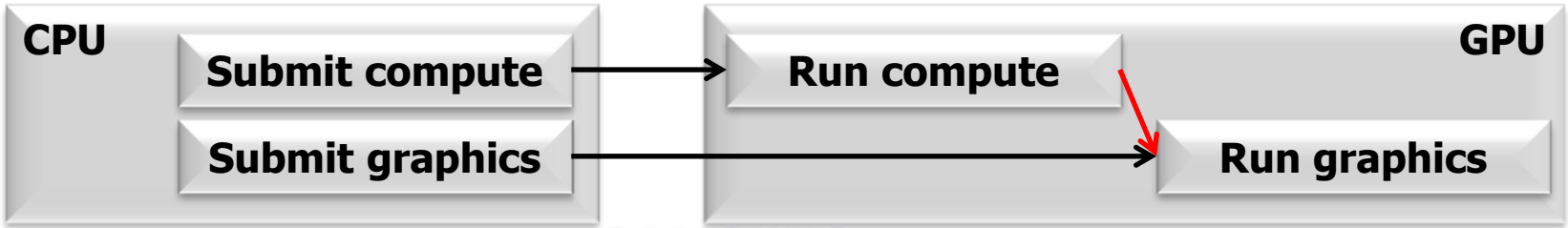
Multithreading

- **Vulkan is much better suited at asynchronous work generation**
  - Command buffer generation can be done in parallel
    - This is the expensive bit (compilation, checking, patching)
- **Different threads can handle different scene components**
- **Synchronisation only required when command buffers are submitted**
  - Buffer submission should be light-weight

Command buffer use control

- **Tiled renderers need to process the command buffer repeatedly**
  - Typically once for binning, once per tile (though it's more efficient than this)
  - Many tiled renderers have binning in one frame and rasterising in another

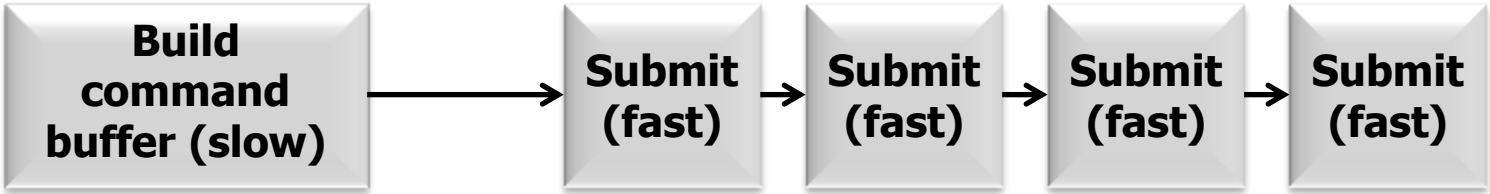
- **Vulkan supports explicit lifetime support for command buffers**
  - The driver should not need to stall just in case you do something weird
- **There is explicit support for synchronisation of queued work**
  - e.g. queue up a rendering job to run as soon as a compute job is complete



SAMSUNG

Command buffer use control

- **You can reuse command buffers**
  - Prepare a command buffer once, then use it repeatedly with little CPU work
  - Use the same rendering commands for multiple objects in a frame
  - Render the same thing (with different parameters) in different frames

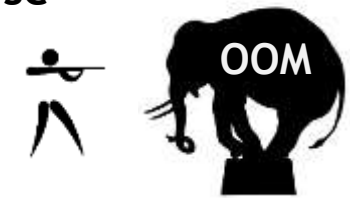


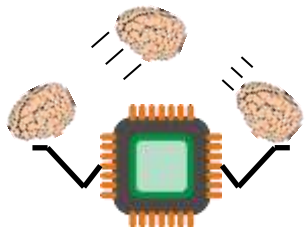
- **Command buffer generation can be optimised off the critical path**
  - May require recompilation for state changes, format changes, etc.
  - Point of overhead is now predictable and can be moved away from rendering

SAMSUNG

Direct control over memory

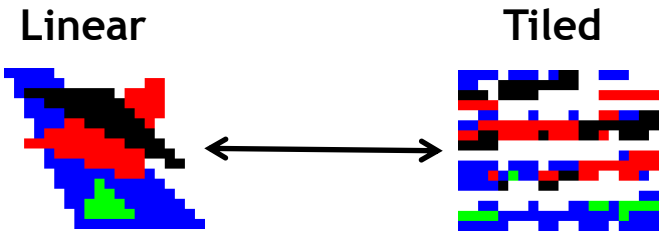
- **Despite phones with lots of RAM, Android limits what apps can use**
  - You can't just rely on swap like desktop systems
- **OpenGL ES drivers do their own graphics memory allocation**
  - You rely on drivers to work out when to reuse and free memory
  - Easier – but may not be as optimal
- **Vulkan gives memory allocation control to the application**
  - Total memory usage is more visible, simplifying streaming
  - On most mobiles, graphics memory *is* app memory
    - No 12GB GDDR5 graphics cards in phones
- **Vulkan allows smaller memory usage by explicit re-use of the same memory area for different objects**
  - Lifetime management is more explicit, in both directions
  - Application is in charge (i.e. doing it right is your problem!)





State transitioning

- **GPUs are most efficient with proprietary internal memory representations**
  - This is why you still “upload” in shared-memory GLES systems
  - But most GPUs can use simple layouts less efficiently

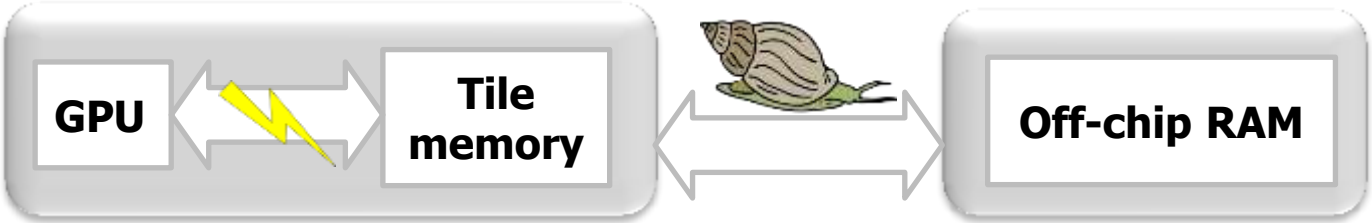


- **Vulkan has explicit control over the type of layout in use**
  - If you want to read it from the CPU, request that in the layout
  - If you want maximum GPU performance, don’t ask for CPU access
  - In desktop systems, this also applies to local vs system memory
- **State transitions in Vulkan are explicit**
  - Layout conversion happens when you ask for it (no unexpected overheads)
  - Multisample resolve is in your control
  - The overhead of “uploading” textures should be much reduced

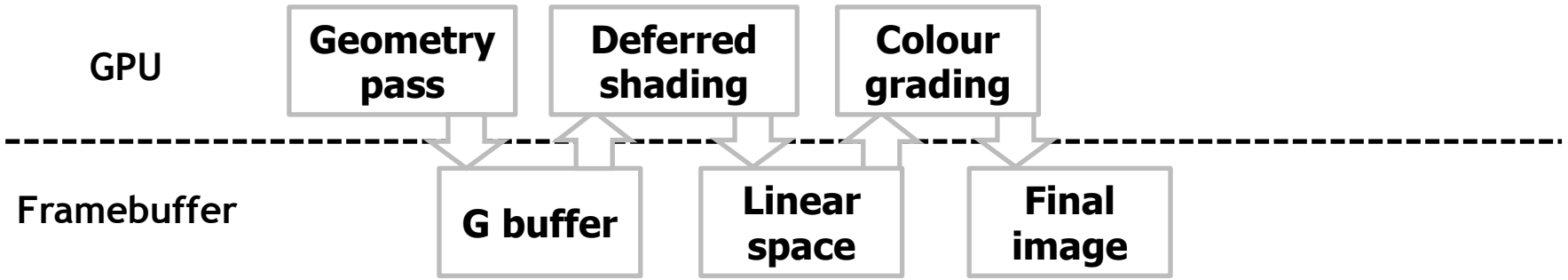
SAMSUNG

Subpasses

- Tiled renderers try to minimise access to external framebuffer memory



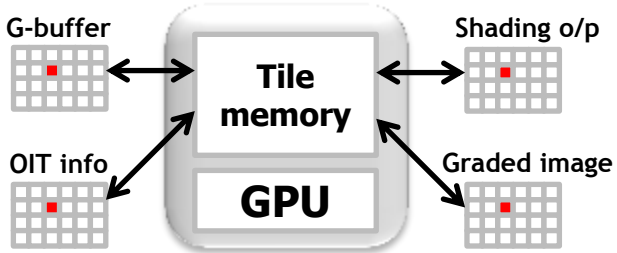
- Some rendering operations require multiple iterations over the framebuffer
  - Deferred shading, tone mapping, order-independent transparency, HDR
  - Writing the whole framebuffer to external memory and reading it back is slow

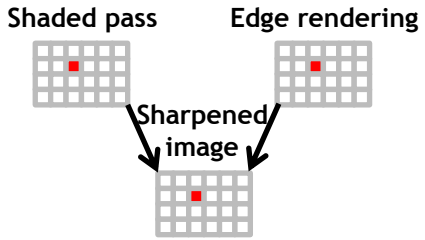


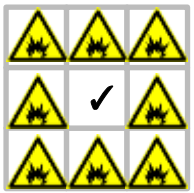
SAMSUNG

Subpasses

- **Vulkan supports a dependency chain of different rendering attachments**
  - A tiler will try to do all operations within a tile before moving to the next
  - The driver will schedule subpasses optimally for memory access
  - Immediate-mode renderers may efficiently use off-chip RAM (unlike PLS)
- **Multiple inputs and outputs can coexist**
  - Spilling to off-chip memory happens only if necessary
- **Currently (because of how tilers work) you can only see your own coordinate across inputs and outputs**
  - You still have to do area operations like depth of field and bloom the hard way!







SAMSUNG

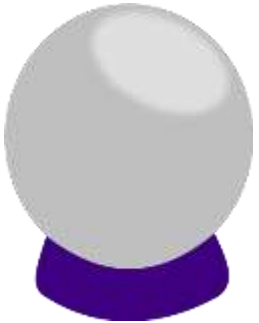
Predictability

- **Vulkan drivers are simpler than OpenGL ES drivers**
  - Game developers waste time trying to work out when a driver would hitch
  - Drivers which try to work around slow applications can be rather complicated!
  - Different platforms had very different behaviour
- **Vulkan makes overheads explicit**
  - Not everything is slow on every platform, but you know when you're doing something that might take time
  - You can usually move the slow things off the critical path

- **Validation allows checks on portability**
  - The validation layer allows you to check whether your application is portable
- **Error checking when you want it**
  - Vulkan's error layer lets you debug efficiently, but doesn't introduce overhead on shipping applications

Predictability

- **In conventional APIs, the driver does a lot of work**
  - Performance benefits come from driver optimisation as much as hardware
- **Drivers are clever, but not always at the same things**
  - Mobile architectures are very different, not all speed-ups are easy to achieve
- **Mobile driver upgrades are less frequent than desktop**
  - Drivers are part of OS updates
  - Operators test and certify platform changes
- **Moving work to the application or game engine means improvements could get rolled out faster**
  - If drivers are kept simpler → more consistent → more predictable
- **This comes from the driver not doing the app's optimisation work**
  - Performance tuning is easier, but it's the app's responsibility
  - Layers and game engines can help



SAMSUNG

SPIR-V

- **Consistency**
  - No reliance on different shading language compilers
  - Much more tightly defined
  - Lots of mobile GPU vendors
  - Still discovering shading language ambiguities that affect ES 2.0

- **Portability**
  - Don’t have to start with GLSL – easier to generate from other sources
  - Simplifies sharing a tool chain across platforms
  - Dynamic shader generation is more predictable
- **Potential compilation cost reduction**
  - Heavy lifting of optimisation can be done when generating SPIR-V
  - Reduces work driver compilation must do at runtime

SAMSUNG

The API got out of your way

- **Vulkan gives you an abstraction which is much closer to the behaviour of actual hardware than traditional APIs**
  - Even more true on mobile (mobile GPUs are *weird*)

- **Your app is driving the hardware directly**
  - Just enough abstraction to make things portable
  - You're not being second-guessed by the driver...
  - ...but you're not being *first*-guessed either
  - Your chauffeur went away; it's up to you not to drive off the road

- **You now have all the control you need to get the best out of the hardware**
  - If it doesn't go fast in Vulkan, it's your fault!
  - With great power comes great responsibility

SAMSUNG

SAMSUNG

Vulkan on Android: Hooking up

Alon Or-bach, Samsung Electronics

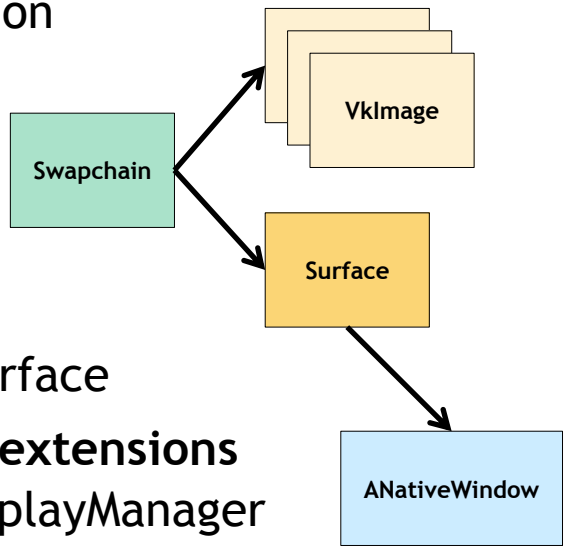
GDC 2016

Vulkan Loader and Layers on Android

- **Vulkan is not yet included in an Android API level in Marshmallow**
  - Applications should load libvulkan.so dynamically
- **Android loader imposes a few more rules than vanilla Khronos / LunarG loader**
  - Same, standardised interface
  - Uses set file locations rather than JSON and env vars
- **Restrictions on how layers are installed to address security concerns**
  - Developers free to include any layer in their own APK
  - Layers can be enabled on any **debuggable application** with ADB

Presenting to Vulkan on Android

- **Majority of Window System Integration implemented in platform code**
  - Avoids inconsistent behaviour of things like orientation switching
  - Implemented in Android's libvulkan.so, alongside the ICD loader
- **Android support the following Vulkan Window System Integration extensions**
  - **VK_KHR_surface** – cross-platform instance extension that introduces the VkSurface object to abstracts from a native platform surface or window
  - **VK_KHR_android_surface** – allows creation of a VkSurface object from an Android Native Window
  - **VK_KHR_swapchain** – introduces the VkSwapchain object that enables applications to present to a surface



- **Android does not use the WSI display management extensions**
  - External displays managed via SurfaceFlinger / DisplayManager
  - No exclusive control of display

SAMSUNG

Presenting to Vulkan on Android cont.

- **Android does not support the IMMEDIATE or FIFO_RELAXED presentation modes**
  - Only FIFO (guaranteed on all platforms) and MAILBOX currently supported
- **If `swapchain.imageExtent!=surface.currentExtent` => scaling will occur**
  - Use the surface's current extent to match the window size
  - Scaling will not respect aspect ratio - so use right values, or be stretched
- **Stick to triple-buffering, unless you know what you're doing!**
  - Possible to ask for `swapchain.minImageCount=2`, but may result in a bubbly pipeline, so use with caution
  - Application can request >3 as well, but should consider memory usage

SAMSUNG

Vulkan on Android: What next?

Alon Or-bach, Samsung Electronics

GDC 2016

So what's next for Vulkan on Android?

- **We want your feedback on Vulkan 1.0!**
  - Please engage with us on http://github.com/KhronosGroup
  - Take the Samsung developer survey http://goo.gl/QGoZ5Q
  - Provide Google your feedback on the N Developer Preview
- **Direction of travel will depend on your input**
  - Yes, really!
- **Features we're keen to see (note: this is just our opinion)**
  - Front-buffer rendering
  - Other features helpful to VR
  - Consuming external data
  - Feature sets
- **But most keen on your requirements**
  - What would make your life better developing for Vulkan on Android?

SAMSUNG

Vulkan on Android: Any questions?

Contact me: alon.orbach@samsung.com / @alonorbach
Take the Samsung developer survey http://goo.gl/QGoZ5Q

Alon Or-bach, Samsung Electronics

GDC 2016