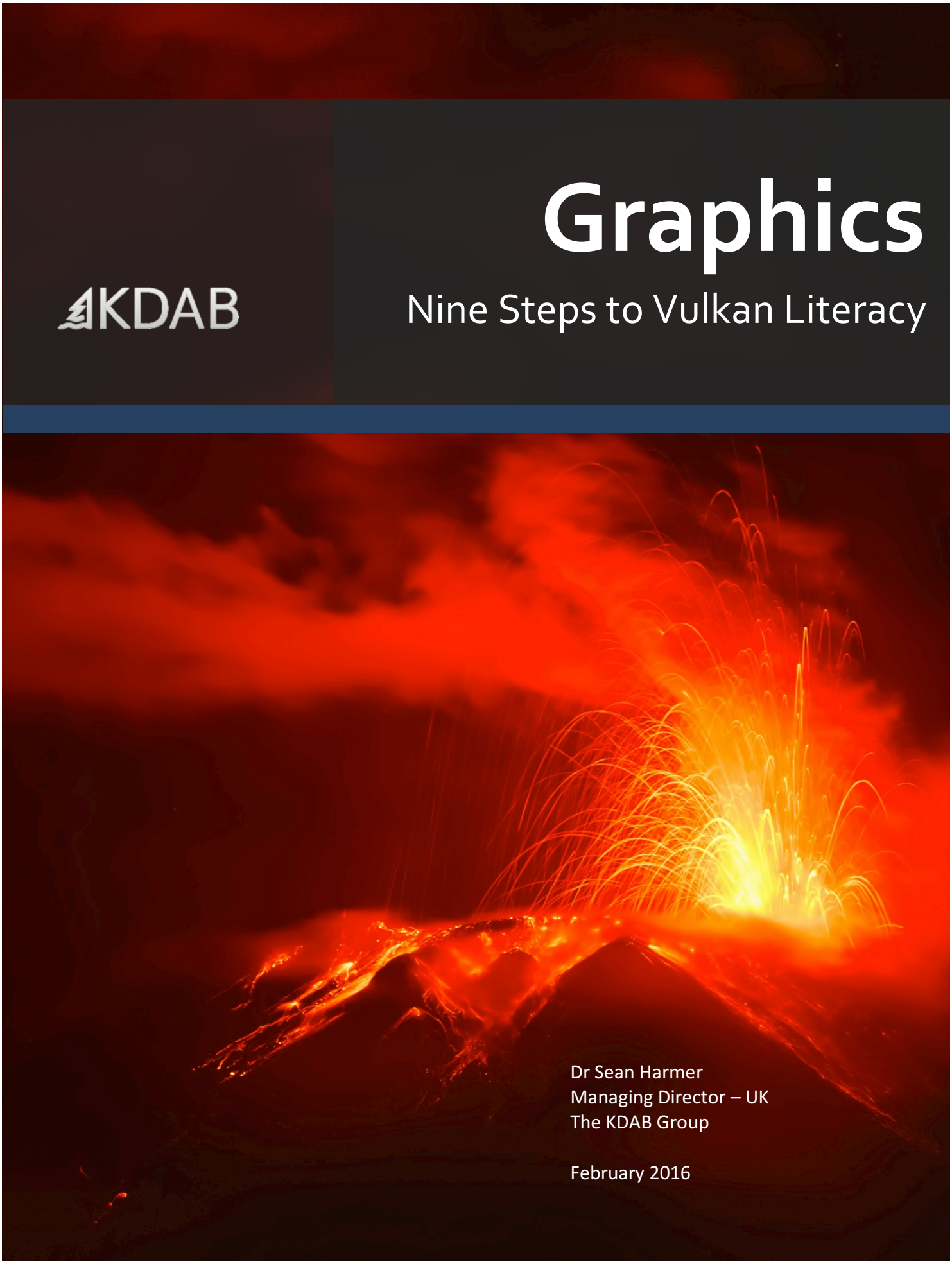


Graphics

Nine Steps to Vulkan Literacy

Dr Sean Harmer
Managing Director – UK
The KDAB Group

February 2016

Vulkan 1.0 was released on February 16th and provides a variety of advantages over other APIs. Here's a quick-start primer on what you need to know.

Nine Steps to Vulkan Literacy

"The coexistence of Vulkan and OpenGL+ES is one of our design goals"

The Khronos Group

Vulkan (spelled with a "k", not a "c") is a powerful new 3D graphics API from the Khronos Group, the same consortium that developed its spiritual predecessor, OpenGL and other related standards.

Like OpenGL, Vulkan targets high-performance real-time 3D graphics applications such as games and interactive media, but offers higher performance and lower CPU usage, much like Direct3D 12 and Metal. Released on February 16th 2016, it provides a variety of advantages over these other APIs.

If you're not yet fully up to speed on Vulkan, read on for a quick-start primer on the nine things you need to know about this shiny new API.

1. Vulkan won't replace OpenGL or OpenGL ES overnight

True, Vulkan is the heir apparent to OpenGL. And it will most likely replace OpenGL and OpenGL ES at some point in the future. But if you're a programmer relying on OpenGL+ES (OpenGL and OpenGL ES), don't panic!

There are several reasons to stay calm and code on. First of all, Vulkan requires hardware that can support OpenGL 4.5 and up, or OpenGL ES 3.1 and up; the existing OpenGL+ES APIs are still required for older hardware. Secondly, Vulkan is a more explicit API while OpenGL+ES is a bit more hands-on (with the driver taking care of much of the memory management) and is an easier place for programmers to start. Thirdly, OpenGL+ES is currently enabling billions of devices so there are business reasons that the Khronos Group will not just maintain these APIs but evolve them as well. Finally, and most importantly, the Khronos Group clearly states that the coexistence of Vulkan and OpenGL+ES is one of its design goals. Vulkan is modularly designed such that an OpenGL+ES API can potentially be layered on top of a bare Vulkan API, a win/win strategy for compatibility.

Vulkan removes the distinction between mobile and desktop. It is a single API that works on all platforms—desktop, mobile, console and embedded.

Vulkan enables milliwatt rendering, opening up rich graphics for IoT and wearables

2. Vulkan closes the gap between mobile and desktop

If you've done any OpenGL ES programming for mobiles, you know it is mostly a subset of OpenGL (even though there are some oddities like texture-format support that break the strict subset nature). In some cases, you probably had to really thrift down on the APIs you used, and if you ported over existing OpenGL code, you may have had a rude awakening. Indeed, removal of some of the OpenGL niceties means that heavy lifting is necessary to convert structures or program flows over to being ES compliant.

Thankfully, Vulkan removes the distinction between mobile and desktop. It is a single API that works on all platforms—desktop, mobile, console and embedded—yet still allows hardware vendors to expose additional functionality where available. This extensibility has been, and continues to be, a crucial aspect of OpenGL and Vulkan continues this ethos.

3. Vulkan is a complete API rewrite with no OpenGL legacy

Older APIs tend to gather a lot of cruft when new methodologies come into vogue or new hardware capabilities become available. To maintain backward compatibility and to coexist with legacy APIs, they can be subjected to lots of awkward, kludgy changes. That makes long-standing APIs that much more difficult to deal with and reduces the performance of your underlying product.

For Vulkan, the Khronos Group decided to completely scrap the old OpenGL+ES APIs in favor of a clean-slate approach. Vulkan started life from AMD's generous contribution of the Mantle API, which was modified to be fully open standard and multi-party approved. While this approach does mean breaking backward compatibility, it also ensures this new API will be clean and optimal.

4. Vulkan gives "bare to the metal" performance

OpenGL is great for cross-platform work because you can run OpenGL programs on Windows, Linux/Unix and OSX and across mobile platforms if you also consider OpenGL ES. However, OpenGL, like Direct3D 11 and

Vulkan stays out of the way whenever possible, allowing you to talk to the GPU on its own terms. This has several distinct benefits.

earlier, are limited by how much work the CPU has to do in order to feed data and commands to the GPU. What's more, this work could only be issued from a single thread in the user application.

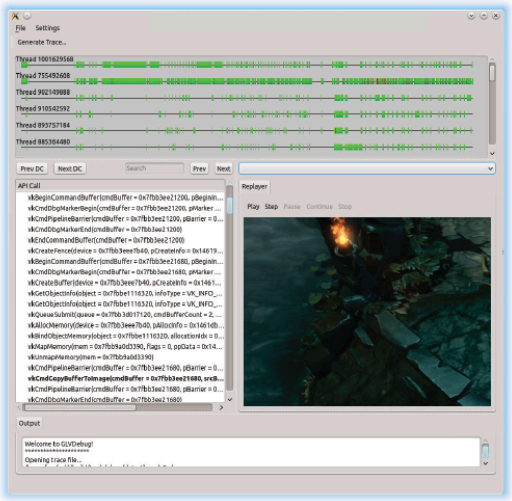
Vulkan fixes all of that. It stays out of the way whenever possible, allowing you to talk to the GPU on its own terms. That frees up the CPU from doing work based upon complicated heuristics in the driver and puts that responsibility in the hands of the application developer. This makes much more sense as only the application developer has the high-level knowledge of what the application needs. This makes the Vulkan drivers much simpler than their OpenGL counterparts. The need for the application developer to handle more of the work is what makes people think of Vulkan (and Direct3D 12/Metal) as lower level. These APIs are not really lower level, just more explicit. The upshot of this is that with full control the application developer can get much more predictable performance. A set of middleware utilities will likely emerge to make Vulkan adoption easier.

In preliminary tests, ARM's Vulkan driver overhead was only 20% of their OpenGL ES driver

Jesse Barker of ARM showed very preliminary results that their Vulkan driver overhead was already only 20% that of their OpenGL ES driver. This is partly because of the above-mentioned reduction in CPU load and partly because Vulkan allows prepared command buffers to be reused multiple times, either within a frame or across frames. With OpenGL, a draw call such as glDrawElements(), prepares the command in the driver and queues it for execution. As such there is no scope for reuse of the internal GPU command. Preparing the command is the expensive part of the operation and so repeated OpenGL draw calls soon add up. Vulkan makes the preparation of commands and their submission two distinct steps. Vulkan allows the user to store prepared commands in a command buffer that can be submitted multiple times. Furthermore, Vulkan allows for two levels of command buffers where secondary buffers can be referenced by primary ones. This comes in useful when you need to draw the same geometry multiple times—for example in a shadow map pass followed by a main lighting pass.

OpenGL programs that are now CPU bound will run significantly faster even if still using just a single CPU core. More advanced threading models (see below) will further improve this. Reducing the CPU load directly translates into bigger thermal headroom and longer battery life on mobile devices.

Vulkan puts the power of multi threading back in your hands, especially important if you're trying to squeeze performance out of multiple cores.



Prototype Vulkan debugger from Valve and LunarG

5. Vulkan is very multi-threading friendly

Another major change for Vulkan is its threading model. OpenGL+ES only allowed API calls from one thread—the thread on which the OpenGL context was current. That limitation ensured the driver wouldn't mangle structures accessed from multiple threads but it was a rather harsh fix, putting a big constraint on how you could feed commands and data to the GPU.

Vulkan puts the power of multi threading back in the programmer's hands, especially important if you're trying to squeeze performance out of multiple cores. Vulkan allows you to create render commands on any thread and provides some support for making memory allocations efficient across multiple threads (pooled allocators). Submission of prepared command buffers can also be done from any thread but you are responsible for ensuring correct ordering. However, with submissions being so inexpensive, it may well be beneficial to build command buffers across many threads and then use a single thread for submission.

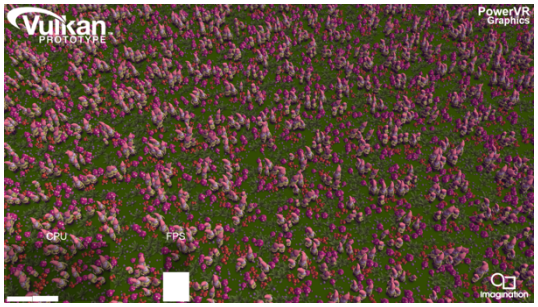
It goes without saying that with power comes responsibility—now you're the one who has to ensure you don't step on yourself across threads. Once again, the anticipated middleware layers are expected to help here.

6. Vulkan makes error checking optional

APIs that check parameters for errors are both a blessing and a curse. While they're invaluable in getting your program off the ground and signaling any problems in program flow, that error checking imposes a significant penalty for every call. As you're probably aware, OpenGL+ES follows the error-checking model: safer to use when getting your program working but it comes at a cost.

Vulkan by default has no error checking. You want direct-access performance? You got it! No error checking will get in between you and the GPU. Of course, that's a bit scary and a recipe for shooting off your foot. Thankfully, Vulkan is a modular, layered API stack. You can add an error-checking module to your initial development phase for safety and strip it out during your release build for performance. Other layers such as reference counting of objects, profiling and validation should soon become available.

Vulkan changes the game by using an intermediate representation for shader execution, SPIR-V— probably its most understated feature.



Imagination PowerVR Vulkan prototype achieves 11.5 times faster framerate than OpenGL ES on Gnome Horde demo

7. Vulkan will have wide (but not universal) platform support

As far as choice of API goes nearly everybody will be supporting Vulkan, including Windows 7/8/10, Linux, Android and SteamOS. Silicon support is practically universal; AMD, ARM, Broadcom, Imagination, Intel, NVIDIA, Qualcomm, Samsung and Vivante are all building Vulkan support into future products. And gaming companies like Unity, Oxide, Blizzard, Epic, EA and Valve are all porting their engines.

Right now Apple is still a holdout, supporting their own proprietary Metal for OSX/iOS. It's unlikely that OSX or iOS drivers can be written for Vulkan without Apple's support, although Vulkan+Metal coexistence should technically be possible. Let's hope the folks in Cupertino see the value of opening up the walled garden with official Vulkan support in the future, which would make all of our lives easier.

8. Vulkan redesigns shaders

In the OpenGL+ES world, GLSL is the only official shader language. That means three things. One, your drivers will need a relatively sophisticated GLSL compiler. Two, you will have to ship unvarnished shader source code with your app. And three, you have a single option for shader control. None of these are really deal-breakers but they aren't optimal either. Vulkan changes the game by using an intermediate representation for shader execution, SPIR-V. Similar to byte code, SPIR-V is neither a human-readable language nor a GPU directly executable one, but somewhere in-between.

What does SPIR-V get us for Vulkan? Without needing a full-featured compiler, your drivers become a lot less complicated and more consistent, and you won't need to expose your shader source. Most importantly, you won't need to constrain the shader language to GLSL. While GLSL to SPIR-V compilers will be made available, SPIR-V opens up the field, allowing you to use other languages to write shaders.

Similar to OpenCL, Vulkan also supports parallel processing applications or other GPU-utilization tools—useful in diverse fields like image processing, machine learning and physical simulations. It supports parallel processing by providing both graphics *and* compute queues to which command buffers

Vulkan specifically supports modern GPU architectures, making it a significantly easier task for developers to create Vulkan device drivers.

can be submitted. This opens up the possibilities for other queue types in the future. Who knows, perhaps one day we will have a ray-tracing queue, allowing Vulkan to mix compute, rasterized content and ray-traced graphics.

The change from GLSL to SPIR-V is probably the most understated component that Vulkan brings and one where I predict we'll see some of the most exciting developments.

"It took two developers just two months to create a preliminary Vulkan driver"

Imagination

9. Vulkan makes writing a driver fundamentally easier

Vulkan is created to specifically support modern GPU architectures and doesn't need significant translation to use a GPU's resources. That means it's a significantly easier task to create Vulkan device drivers. By way of example, Rys Sommefeldt from Imagination said that it took two developers just two months to create their preliminary Vulkan driver.

Why should the rest of us care? Because there are very few graphical device driver authors out there. The easier it is to build new drivers on new hardware, the sooner the rest of us can get our hands on those devices, and the less complicated and buggy those drivers will be. And with an API more in-line with GPU capabilities, we're going to get even better performance.

That's Vulkan in a nutshell. Maybe the best part of all this newfound Vulkan goodness is that work is underway to ensure that Qt will transparently support Vulkan. If you're using Qt, you'll automatically get the benefit of Vulkan when a Vulkan-ready Qt arrives. When will that be? Stay tuned…

About the KDAB Group

The KDAB Group is the world's leading software consultancy for architecture, development and design of Qt, C++ and OpenGL applications across desktop, embedded and mobile platforms. KDAB is the biggest independent contributor to Qt. Our experts build run-times, mix native and web technologies, solve hardware stack performance issues and porting problems for hundreds of customers, many among the Fortune 500. KDAB's tools and extensive experience in creating, debugging, profiling and porting complex applications help developers worldwide to deliver successful projects. KDAB's trainers, all full-time developers, provide market leading, hands-on, training for Qt, OpenGL and modern C++ in multiple languages. Founded in 1999, KDAB has offices throughout North America and Europe.

www.kdab.com

© 2016 the KDAB Group. KDAB is a registered trademark of the KDAB Group. All other trademarks belong to their respective owners.