The News from Planet Vulkan

Tom Olson
Distinguished Engineer, Arm Chair, Vulkan Working Group
Everything Else

VULKAN EVERYWHERE kinda
Availability

This work is licensed under a Creative Commons Attribution 4.0 International License
Vulkan support on Android

Support is mandatory on recent 64-bit devices

Vulkan is on 66% of Android devices - and growing fast!

<table>
<thead>
<tr>
<th>Vulkan version</th>
<th>January 2021</th>
<th>July 2021</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>42%</td>
<td>34%</td>
</tr>
<tr>
<td>Vulkan 1.0</td>
<td>22%</td>
<td>19%</td>
</tr>
<tr>
<td>Vulkan 1.1</td>
<td>36%</td>
<td>47%</td>
</tr>
</tbody>
</table>

https://developer.android.com/about/dashboards
Vulkan Games Shipping on Desktop

Over 160 Vulkan Titles shipping across PC, Linux, Stadia, and MacOS with Molten VK
Vulkan Games Shipping on Mobile

Vulkan’s lower CPU overhead enables better performance and/or lower power

*Vulkan is the default renderer for mobile projects on Unity*
It’s not just about games...

Adobe Premiere Rush
- A pro-quality, cross-platform video capture and editing system

Vulkan powered on Android
- On select phones
- Several hundred thousand lines of OpenCL C compute shaders
- Translated to SPIR-V by clspv
- Run in Vulkan command buffers

Eric Berdahl at SIGGRAPH 2018: https://youtu.be/5rxQ77nZits?t=21980
More Not Games!

Basemark’s Rocksolid Engine uses Vulkan for high performance instrument cluster rendering on automotive GPUs
More Not Games!

Instrument Cluster Rendering – Vulkan is 138% Faster than OpenGL ES

- Various instrument cluster reference designs by Basemark, powered by Basemark's Rocksolid Engine
- Rocksolid Engine has optimized OpenGL ES and Vulkan rendering pipelines
- Running on the same hardware, Vulkan brings substantial performance benefits compared to OpenGL ES
New Functionality
Variable Rate Shading enables focusing of rendering power for more perf/less power. Shading Rate selects how many pixels’ color values are affected by each fragment. ‘Spreads’ a fragment between 1 to 4 times independently on both X and Y axis.

Rate Per Draw Call (optional)
Lower rate for background or low-detail objects

Rate Per Triangle (optional)
Lower rate for low-detail primitives

Rate Per Region (optional)
Lower rate for periphery

Combining Rates
- Keep or Replace or Min or Max (per axis) or Mul (per axis)

Final Rate

Color overlay indicates region shading rates (image courtesy of NVIDIA). Lower-resolution image (render pass attachment) defines a shading rate for each rectangular region.
Vulkan Ray Tracing *(final specifications)*

- Core extension factored into three sub-extensions
- Changes to acceleration structure creation and layout
- Explicit stack size management for ray pipelines
- Consistency, ease of use, and efficiency

[Without Ray Tracing](#)  [With Ray Tracing](#)

Provisional Vulkan Video Extensions

- Seamless integration of video encode / decode into Vulkan
- Expose resource sharing on discrete graphics cards
- Leverage existing Vulkan resources and synchronization mechanisms

- VK_KHR_video_queue
- VK_KHR_video_decode_queue
- VK_KHR_video_encode_queue
- VK_EXT_video_decode_h265
- VK_EXT_video_decode_h264
- VK_EXT_video_decode_AV1
- VK_EXT_video_decode_VP9
- VK_EXT_video_encode_h264
- VK_KHR_video_decode_queue
- VK_KHR_video_encode_queue
- VK_KHR_video_queue
Video Extensions - Status

- Provisional version released in April 2021

- Beta implementation at

- Detailed description in this blog post:
  - [https://www.khronos.org/blog/an-introduction-to-vulkan-video](https://www.khronos.org/blog/an-introduction-to-vulkan-video)

- Please give us your feedback!
  - [https://github.com/KhronosGroup/Vulkan-Docs/issues/1497](https://github.com/KhronosGroup/Vulkan-Docs/issues/1497)
VK_KHR_synchronization2

- Replaces pipeline stage flags and access flags with 64-bit types
  - Needed for the ray tracing and video extensions

- Restructures queue submission, event, and pipeline barrier APIs / types
  - Makes the API easier to understand, less error-prone

- Key changes
  - Use an array of structures rather than separate arrays
  - Store barrier pipeline stage masks in the barrier itself
  - Specify pipeline barriers at vkCmdSetEvent time rather than at Wait time
  - Image layout types are contextual, reducing the chance for error

- We strongly recommend using this synchronization API for your future projects
How to start using VK_KHR_synchronization2

• The Vulkan SDK provides an emulation layer
  - Advertises support even if the underlying ICD does not
  - Implements synchronization2 using old-style synchronization APIs
  - Very light-weight and efficient

• The new API is mostly backward compatible
  - Can mix and match new and old types freely
    - EXCEPT: must use SetEvent with WaitEvents, and SetEvent2 with WaitEvents2
  - You can convert your code a bit at a time

• Lots of information to help you
The Vulkan Ecosystem
The Vulkan Ecosystem

- Support
- Documentation
- Sample code
- Tutorials
- Best practices
- Vulkan SDK
- Vulkan CTS
- Implementations
- Validation layers
- Language toolchain
-spirv-opt
- spirv-cross
- DXC
- glslang
- Shaderc
- IHV tools
- Debug/Perf tools
- Debug extensions
- Vulkan
- SPIR
- Specs
- SPIR-VM
- Validation layers
- Spirv-val
- Spirv-dis
- Spirv-opt
- Spirv-cross
- RenderDoc
SDK Improvements

• Synchronization validation
  - Phase 1: single command buffer (shipping)
  - Phase 2: multiple command buffers (in development)
  - Synchronization2 support (shipping)

• See Ensure Correct Vulkan Synchronization by using Synchronization Validation

• Vulkan Configurator tool (vkconfig) now works with 3rd party layers

• Validation performance improvements
  - 20%-100% speedup

• Debug printf from a shader

• See Simplify Vulkan Development with new Ecosystem Enhancements
New website: https://www.vulkan.org

- Home page for Vulkan on the web
- Make tools and resources easier to find
- Highlight new Vulkan content
- Updated regularly
New Youtube Channel

- www.youtube.com/vulkan

- Videos from Vulkan events
  - GDC
  - CEDEC
  - SIGGRAPH
  - Vulkanised!
  - Reboot Develop
  - ...

© The Khronos Group Inc. 2021 - Page 21

This work is licensed under a Creative Commons Attribution 4.0 International License
Vulkan Samples Repository

- A home for Vulkan sample code
  - Intended to help you learn to use Vulkan effectively
  - GPU, OS, and platform neutral, tested on a wide variety of implementations
  - Open Source under the Apache 2.0 license

- A community effort with participation from
  - Sample code developers, including Sascha Willems and others
  - Khronos member ISVs and IHVs

- Developed on GitHub - all are welcome to contribute and participate
Vulkan Samples available today

https://github.com/KhronosGroup/Vulkan-Samples

- Performance samples
  - 16-bit storage Input/Output
  - 16-bit arithmetic
  - Async compute
  - Basis universal supercompressed GPU textures
  - AFBC
  - Command buffer management
  - Constant data
  - Descriptor and buffer management
  - Impact of vkDeviceWaitIdle()
- Layout transitions
- Load/store operations
- MSAA
- Multi-threading
- N-buffering and presentation modes
- Pipeline barriers
- Pipeline cache
- Pre-rotation
- Specialization constants
- Subpass merging and G-buffer size
Vulkan Samples available today

https://github.com/KhronosGroup/Vulkan-Samples

- **API examples**
  - Compute shader N-body simulation
  - Dynamic uniform buffers
  - High Dynamic Range rendering
  - Instanced mesh rendering
  - Dynamic terrain tessellation
  - Texture loading and display
  - Runtime mipmap generation

- **Extension examples**
  - VK_KHR_buffer_device_address
  - VK_EXT_descriptor_indexing
  - VK_KHR_timeline_semaphore
  - VK_KHR_fragment_shading_rate
  - VK_KHR_ray_tracing_pipeline
  - VK_KHR_synchronization2
  - VK_EXT_debug_utils
  - VK_EXT_conservative_rasterization
  - VK_KHR_push_descriptor
  - VK_KHR_external_memory
ASTC Encoder (ASTCenc)

- ASTC is a multi-format, multi-bit-rate, high quality texture compression format
  - Exposed as an extension in OpenGL ES 3.1
  - LDR subset is required in OpenGL ES 3.2
  - Supported on essentially all Vulkan-enabled mobile devices

- ASTCenc is Arm’s reference codec
  - Developed by Arm GPU R&D as part of format development
  - Optimized for exploring the design space, not for performance
  - Never intended to be a product

- Available on GitHub since 2012
  - Under a forbidding EULA

- Reboot in 2019 led by Pete Harris
ASTC Encoder Improvements since 2019

- **Licensing**
  - Now available under Apache 2.0

- **Productization**
  - Refactored into an embeddable library and a front end
  - Builds robustly across many compilers and Oses
  - Output is now invariant across platforms
  - Proper CI prevents regressions

- **Performance**
  - Improved code quality
  - Improved search heuristics
  - Extensive vectorization using SSE2, SSE4.1, AVX2, Neon
Results

- 12x speedup (typical) at a cost of 0.1 dB PSNR
- ASTCenc is shipping in the Unity toolchain
- ASTC is now the default texture compression format for Unity on mobile
- Source code: https://github.com/ARM-software/astc-encoder
- Details: https://community.arm.com/developer/tools-software/graphics/b/blog/posts/astcenc-31-high-performance-astc-texture-compressor

Thanks Pete!
Thank You!