Vulkan

# Introduction

Oregon State University
Mike Bailey
mjb@cs.oregonstate.edu



Oregon State University Computer Graphics





## Acknowledgements

First of all, thanks to the inaugural class of 19 students who braved new, unrefined, and just-in-time course materials to take the first Vulkan class at Oregon State University – Winter Quarter, 2018. Thanks for your courage and patience!

Oregon State University

| | |
|---|---|
| Ali Alsalehy | Alan Neads |
| Natasha Anisimova | Raja Petroff |
| Jianchang Bi | Bei Rong |
| Christopher Cooper | Lawrence Roy |
| Richard Cunard | Lily Shellhammer |
| Braxton Cuneo | Hannah Solorzano |
| Benjamin Fields | Jian Tang |
| Trevor Hammock | Glenn Upthagrove |
| Zach Lerew | Logan Wingard |
| Victor Li | |

Second, thanks to NVIDIA! The GeForce 1080ti cards are what made this course possible.

NVIDIA

Third, thanks to Kathleen Mattson and the Khronos Group for the great laminated Vulkan Quick Reference Cards! (Look at those happy faces in the photo holding them.)

KHRONOS GROUP

Oregon State University Computer Graphics





## What Prompted the Move to Vulkan?

1. Performance
2. Performance
3. Performance

Vulkan is better at keeping the GPU busy than OpenGL is. OpenGL drivers need to do a lot of CPU work before handing work off to the GPU. Vulkan lets you get more power from the GPU card you already have.

This is especially important if you can hide the complexity of Vulkan from your customer base and just let them see the improved performance. Thus, Vulkan has had a lot of support and interest from game engine developers, 3rd party software vendors, etc.

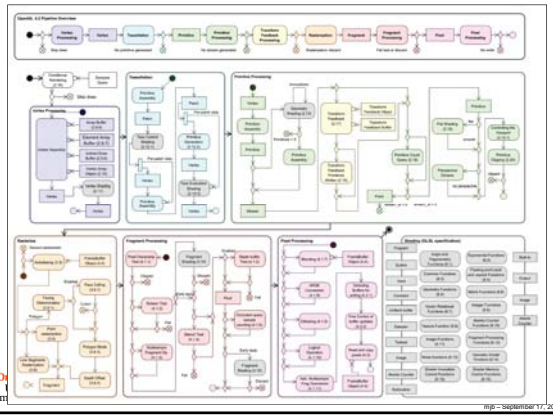
As an aside, the Vulkan development effort was originally called **glNext**, which created the false impression that this was a replacement for OpenGL. It's not.

Oregon State University Computer Graphics





## OpenGL 4.2 Pipeline Flowchart







## Why is it so important to keep the GPU Busy?

**NVidia Titan V Specs vs. Titan Xp, 1080 Ti**

| | Titan V | Tesla V100 | Tesla P100 | GTX 1080 Ti | GTX 1080 |
|---|---|---|---|---|---|
| GPU | GV100 | GV100 | GP100 Cut-Down Pascal | GP102 Pascal | GP104-400 Pascal |
| Transistor Count | 21.1B | 21.1B | 15.3B | 12B | 7.2B |
| Fab Process | 12nm FFN | 12nm FFN | 16nm FinFET | 16nm FinFET | 16nm FinFET |
| CUDA Cores / Tensor Cores | 5120 / 640 | 5120 / 640 | 3584 / 0 | 3584 / 0 | 2560 / 0 |
| TMUs | 320 | | 224 | 224 | 160 |
| ROPs | ? | | 96 (?) | 88 | 64 |
| Core Clock | 1200MHz | | 1328MHz | - | 1607MHz |
| Boost Clock | 1455MHz | 1370MHz | 1480MHz | 1600MHz | 1733MHz |
| FP32 TFLOPs | 15TFLOPs | 14TFLOPs | 10.6TFLOPs | ~11.4TFLOPs | 9TFLOPs |
| Memory Type | HBM2 | HBM2 | HBM2 | GDDR5X | GDDR5X |
| Memory Capacity | 12GB | 16GB | 16GB | 11GB | 8GB |
| Memory Clock | 1.7Gbps HBM2 | 1.75Gbps HBM2 | ? | 11Gbps | 10Gbps GDDR5X |
| Memory Interface | 3072-bit | 4096-bit | 4096-bit | 352-bit | 256-bit |
| Memory Bandwidth | 653GB/s | 900GB/s | ? | ~484GB/s | 320.32GB/s |
| Total Power Budget ("TDP") | 250W | 250W | 300W | 250W | 180W |
| Power Connectors | 1x 8-pin 1x 6-pin | | ? | 1x 8-pin 1x 6-pin | 1x 8-pin |
| Release Date | 12/07/2017 | | 4Q16-1Q17 | TBD | 5/27/2016 |
| Release Price | $3000 | $10000 | - | $700 | Reference: $700 MSRP: $600 Now: $500 |

The nVidia Titan V graphics card is not targeted at gamers, but rather at scientific and machine/deep learning applications. That does not, however, mean that the card is incapable of gaming, nor does it mean that we can't extrapolate future key performance metrics for Volta. The Titan V is a derivative of the earlier released GV100 GPU, part of the Tesla accelerator card series. The key differentiator is that the Titan V ships at $3000, whereas the Tesla V100 was available as part of a $10,000 developer kit. The Tesla V100 still offers greater memory capacity by 4GB – 16GB HBM2 versus 12GB HBM2 – and has a wider memory interface, but other core features remain matched or nearly matched. Core count, for one, is 5120 CUDA cores on each GPU, with 640 Tensor cores (used for Tensorflow deep/machine learning workloads) on each GPU.

Oregon State University Computer Graphics





## Who was the original Vulcan?

**From WikiPedia:**

"Vulcan is the god of fire including the fire of volcanoes, metalworking, and the forge in ancient Roman religion and myth. Vulcan is often depicted with a blacksmith's hammer.The **Vulcanalia** was the annual festival held August 23 in his honor. His Greek counterpart is Hephaestus, the god of fire and smithery. In Etruscan religion, he is identified with Sethlans.
Vulcan belongs to the most ancient stage of Roman religion: Varro, the ancient Roman scholar and writer, citing the Annales Maximi, records that king Titus Tatius dedicated altars to a series of deities among which Vulcan is mentioned."

https://en.wikipedia.org/wiki/Vulcan_(mythology)

Oregon State University Computer Graphics

## Why Name it after the God of the Forge?

7


Oregon State University Computer Graphi



## Who is the Khronos Group?

8

**The Khronos Group, Inc.** is a non-profit member-funded industry consortium, focused on the creation of open standard, royalty-free application programming interfaces (APIs) for authoring and accelerated playback of dynamic media on a wide variety of platforms and devices. Khronos members may contribute to the development of Khronos API specifications, vote at various stages before public deployment, and accelerate delivery of their platforms and applications through early access to specification drafts and conformance tests.


Oregon State University Computer Graphics



## Playing "Where's Waldo" with Khronos Membership

9


Oregon State University Computer Graphics



## Who's Been Specifically Working on Vulkan?

10


Oregon State University Computer Graphics



## Vulkan

11

- Largely derived from AMD's *Mantle* API
- Also heavily influenced by Apple's *Metal* API and Microsoft's *DirectX 12*
- Goal: much less driver complexity and overhead than OpenGL has
- Goal: much less user hand-holding – Vulkan can crash
- Goal: higher single-threaded performance than OpenGL can deliver
- Goal: able to do multithreaded graphics
- Goal: able to handle tiled rendering

Oregon State University Computer Graphics



## Vulkan Differences from OpenGL

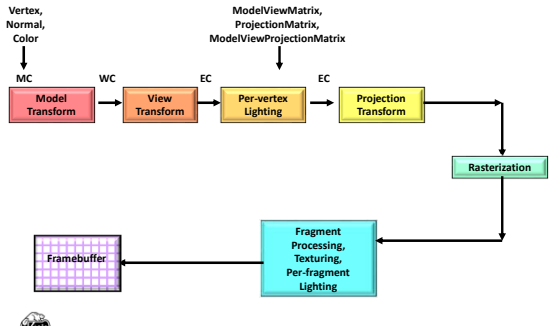
12

- More low-level information must be provided (by you!) in the application, rather than the driver
- Screen coordinate system is Y-down
- No "current state", at least not one maintained by the driver
- All of the things that we have talked about being ***deprecated*** in OpenGL are *really* **deprecated** in Vulkan: built-in pipeline transformations, begin-end, fixed-function, etc.
- You must manage your own transformations.
- All transformation, color, texture functionality must be done in shaders.
- Shaders are pre-"half-compiled" outside of your application. The compilation process is then finished during the pipeline-building process.

Oregon State University Computer Graphics

## The Basic OpenGL Computer Graphics Pipeline, OpenGL-style

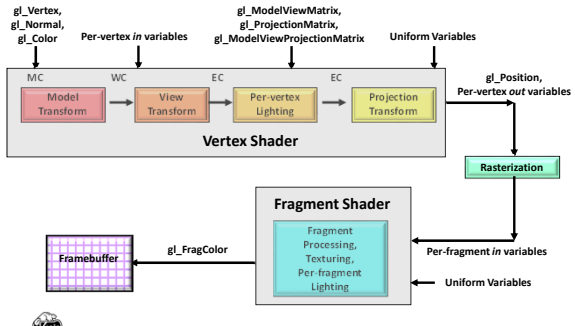
13




Oregon State University Computer Graphics

## The Basic Computer Graphics Pipeline, Shader-style

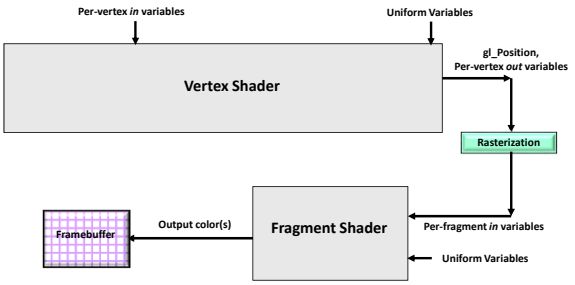
14




Oregon State University Computer Graphics

## The Basic Computer Graphics Pipeline, Vulkan-style

15




Oregon State University Computer Graphics

mjb – September 17, 2018

## A Complete API Redesign

16

| OpenGL | Vulkan |
|---|---|
| Originally architected for graphics workstations with direct renderers and split memory | Matches architecture of modern platforms including mobile platforms with unified memory, tiled rendering |
| Driver does lots of work: state validation, dependency tracking, error checking. Limits and randomizes performance | Explicit API – the application has direct, predictable control over the operation of the GPU |
| Threading model doesn't enable generation of graphics commands in parallel to command execution | Multi-core friendly with multiple command buffers that can be created in parallel |
| Syntax evolved over twenty years – complex API choices can obscure optimal performance path | Removing legacy requirements simplifies API design, reduces specification size and enables clear usage guidance |
| Shader language compiler built into driver. Only GLSL supported. Have to ship shader source | SPIR-V as compiler target simplifies driver and enables front-end language flexibility and reliability |
| Despite conformance testing, developers must often handle implementation variability between vendors | Simpler API, common language front-ends, more rigorous testing increase cross vendor functional/performance portability |

Khronos Group

Oregon State University Computer Graphics

mjb – September 17, 2018

## Moving part of the driver into the application

17

| | OpenGL | Vulkan |
|---|---|---|
| Complex drivers lead to driver overhead and cross vendor unpredictability | Application | Application responsible for memory allocation and thread management to generate command buffers |
| Error management is always active | Traditional graphics drivers include significant context, memory and error management | Direct GPU Control |
| Driver processes full shading language source | GPU | GPU |
| Separate APIs for desktop and mobile markets | | |

Simpler drivers for low-overhead efficiency and cross vendor portability

Layered architecture so validation and debug layers can be unloaded when not needed

Run-time only has to ingest SPIR-V intermediate language

Unified API for mobile, desktop, console and embedded platforms

Khronos Group

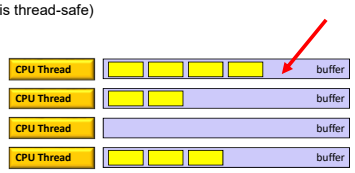
Oregon State University Computer Graphics

mjb – September 17, 2018

## Vulkan Highlights: Command Buffers

18

- Graphics commands are sent to command buffers
- Think OpenCL…
- E.g., *vkCmdDoSomething( cmdBuffer, … );*
- You can have as many simultaneous Command Buffers as you want
- Buffers are flushed when the application wants them flushed
- Each command buffer can be filled from a different thread (i.e., filling is thread-safe)




Oregon State University Computer Graphics

mjb – September 17, 2018

## Vulkan Highlights: Pipelines

19

- In OpenGL, your "pipeline state" is whatever your current graphics attributes are: color, transformations, textures, shaders, etc.
- Changing the state on-the-fly one item at-a-time is very expensive
- Vulkan forces you to set all your state at once into a "pipeline state object" (PSO) and then invoke the entire PSO whenever you want to use that state combination
- Think of the pipeline state as being immutable.
- Potentially, you could have thousands of these pre-prepared states
- This is a good time to talk about how game companies view Vulkan…

Oregon State University
Computer Graphics



## Querying the Number of Things

20

```
uint32_t  count;
result = vkEnumeratePhysicalDevices( Instance, OUT &count, OUT (VkPhysicalDevice *)nullptr );

VkPhysicalDevice * physicalDevices = new VkPhysicalDevice[ count ];
result = vkEnumeratePhysicalDevices( Instance, OUT &count, OUT physicalDevices  );
```

**This way of querying information is a recurring OpenCL and Vulkan pattern (get used to it):**

How many total there are — Where to put them

```
result = vkEnumeratePhysicalDevices( Instance,   &count,   nullptr   );

result = vkEnumeratePhysicalDevices( Instance,   &count,   physicalDevices );
```

Oregon State University
Computer Graphics



## Vulkan Code has a Distinct "Style"

21

```
VkBufferCreateInfo              vbci;
        vbci.sType = VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO;
        vbci.pNext = nullptr;
        vbci.flags = 0;
        vbci.size = << buffer size in bytes >>
        vbci.usage = VK_USAGE_UNIFORM_BUFFER_BIT;
        vbci.sharingMode = VK_SHARING_MODE_EXCLUSIVE;
        vbci.queueFamilyIndexCount = 0;
        vbci.pQueueFamilyIndices = nullptr;

VK_RESULT result = vkCreateBuffer ( LogicalDevice, IN &vbci, PALLOCATOR,  OUT &Buffer );

VkMemoryRequirements          vmr;

result = vkGetBufferMemoryRequirements( LogicalDevice, Buffer, OUT &vmr );      // fills vmr

VkMemoryAllocateInfo          vmai;
        vmai.sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
        vmai.pNext = nullptr;
        vmai.flags = 0;
        vmai.allocationSize = vmr.size;
        vmai.memoryTypeIndex = 0;

result = vkAllocateMemory( LogicalDevice, IN &vmai, PALLOCATOR, &MatrixBufferMemoryHandle );

result = vkBindBufferMemory( LogicalDevice, Buffer, MatrixBufferMemoryHandle, 0 );
```

Oregon State University
Computer Graphics



## Vulkan Quick Reference Card

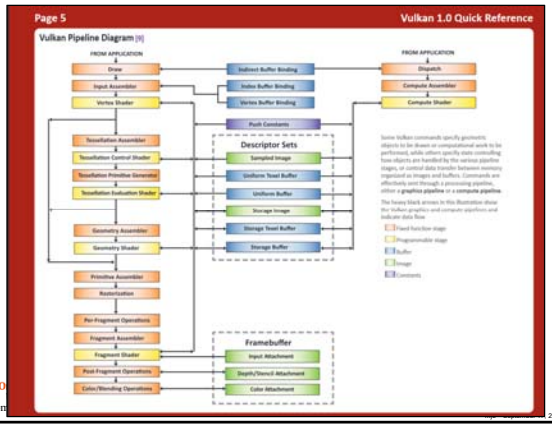
22

https://www.khronos.org/files/vulkan11-reference-guide.pdf



## Vulkan Quick Reference Card

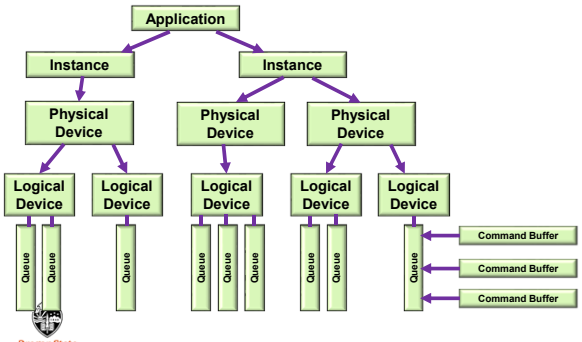
23






## Vulkan Highlights: Overall Block Diagram

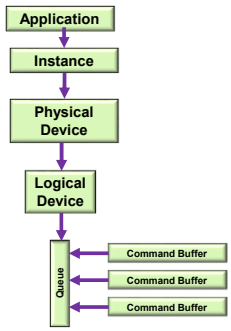
24




Oregon State University
Computer Graphics

## Vulkan Highlights: a More Typical Block Diagram

25




Oregon State University Computer Graphics



## Steps in Creating Graphics using Vulkan

26

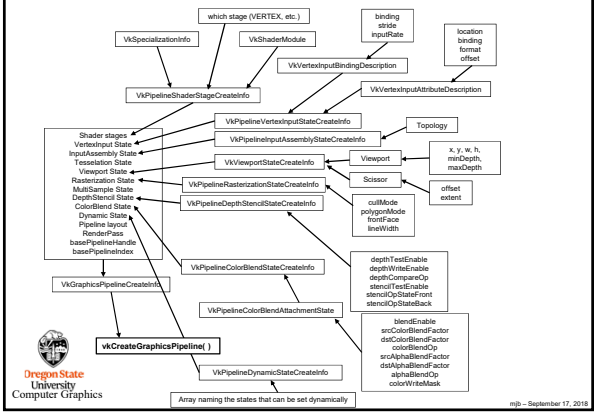
1. Create the Instance
2. Setup the Debug Callbacks
3. Create the Surface
4. List the Physical Devices
5. Pick the right Physical Device
6. Create the Logical Device
7. Create the Uniform Variable Buffers
8. Create the Vertex Data Buffers
9. Create the texture sampler
10. Create the texture images
11. Create the Swap Chain
12. Create the Depth and Stencil Images
13. Create the RenderPass
14. Create the Framebuffer(s)
15. Create the Descriptor Set Pool
16. Create the Command Buffer Pool
17. Create the Command Buffer(s)
18. Read the shaders
19. Create the Descriptor Set Layouts
20. Create and populate the Descriptor Sets
21. Create the Graphics Pipeline(s)
22. Update-Render-Update-Render- …

Oregon State University Computer Graphics



## Vulkan: Creating a Pipeline

27




Oregon State University Computer Graphics



## Vulkan GPU Memory

28

- Your application allocates GPU memory for the objects it needs
- You map GPU memory to the CPU address space for access
- Your application is responsible for making sure what you put into that memory is actually in the right format, is the right size, has the right alignment, etc.

**From the OpenGL Shader Storage Buffer notes:**

```
glGenBuffers( 1, &posSSbo);
glBindBuffer( GL_SHADER_STORAGE_BUFFER, posSSbo );
glBufferData( GL_SHADER_STORAGE_BUFFER, NUM_PARTICLES * sizeof(struct pos), NULL, GL_STATIC_DRAW );

GLint bufMask = GL_MAP_WRITE_BIT | GL_MAP_INVALIDATE_BUFFER_BIT ;     // the invalidate makes a big difference when re-writing

struct pos *points = (struct pos *) glMapBufferRange( GL_SHADER_STORAGE_BUFFER, 0, NUM_PARTICLES * sizeof(struct pos), bufMask );
```

Oregon State University Computer Graphics



## Vulkan Render Passes

29

- Drawing is done inside a render pass
- Each render pass contains what framebuffer attachments to use
- Each render pass is told what to do when it begins and ends
- Multiple render passes can be merged

Oregon State University Computer Graphics



## Vulkan Compute Shaders

30

- Compute pipelines are allowed, but they are treated as something special (just like OpenGL does)
- Compute passes are launched through dispatches
- Compute command buffers can be run asynchronously

Oregon State University Computer Graphics

## Vulkan Synchronization

31

- Vulkan tries to run "flat out"
- Therefore, synchronization is the responsibility of the application
- Events can be set, polled, and waited for (much like OpenCL)
- Vulkan does not ever lock – that's the application's job
- Threads can concurrently read from the same object
- Threads can concurrently write to different objects

Oregon State University Computer Graphics

mjb – September 17, 2018

## Vulkan Shaders

32

- GLSL is the same as before … almost
- For places it's not, an implied
  **#define VULKAN 100**
  is automatically supplied by the compiler
- You pre-compile your shaders with an external compiler
- Your shaders get turned into an intermediate form known as SPIR-V (Standard Portable Intermediate Representation for Vulkan)
- SPIR-V gets turned into fully-compiled code at runtime
- The SPIR-V spec has been public for months –new shader languages are surely being developed
- OpenCL and OpenGL will be moving to SPIR-V as well

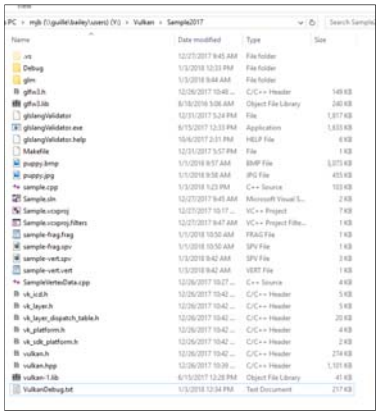
GLSL Source → **External GLSL Compiler** → SPIR-V → **Compiler in driver** → Vendor-specific code

**Advantages:**

1. Software vendors don't need to ship their shader source
2. Software can launch faster because half of the compilation has already taken place
3. This guarantees a common front-end syntax
4. This allows for other language front-ends

## Your Sample2017.zip File Contains This

33



Oregon State University Computer Graphics

mjb – September 17, 2018