Vulkan

Oregon State University

Mike Bailey

mjb@cs.oregonstate.edu

Oregon State University
Computer Graphics

You can learn more at: **http://cs.oregonstate.edu/~mjb/vulkan**

Who is the Real Vulkan?

Can you notice the difference? It's subtle! ☺

Who is the Khronos Group?

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.

Playing "Where's Waldo" with Khronos Membership

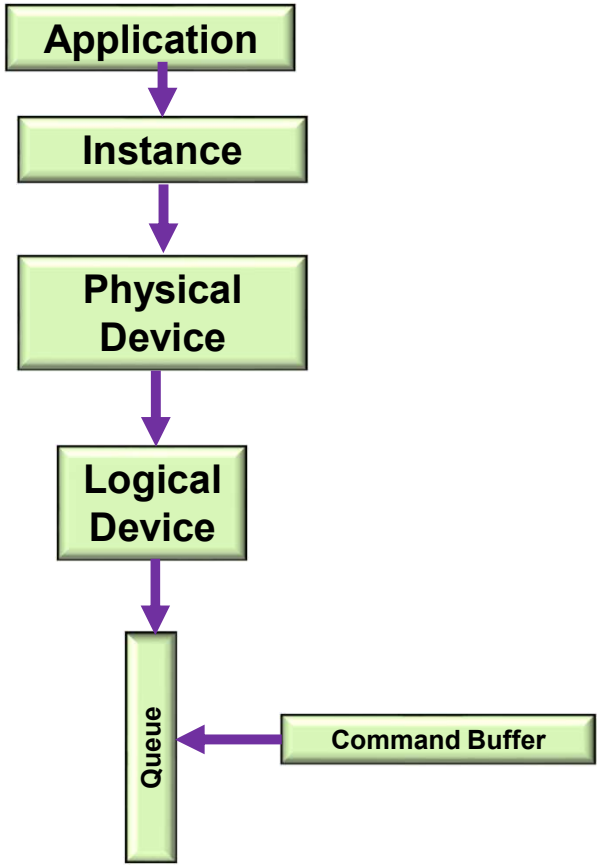
Oregon State University
Computer Graphics

Who's Been Specifically Working on Vulkan?

Vulkan

- Largely derived from AMD's *Mantle* API
- Also heavily influenced by Apple's *Metal* API and Microsoft's *DirectX 12*
- *There is no fixed-function – it is all shaders-based*
- Goal: much less driver complexity and overhead than OpenGL has
- Goal: much less user hand-holding
- Goal: higher single-threaded performance than OpenGL can deliver
- Goal: able to do multithreaded graphics
- Goal: able to run on desktops and mobile devices

Vulkan: a Simplified Block Diagram



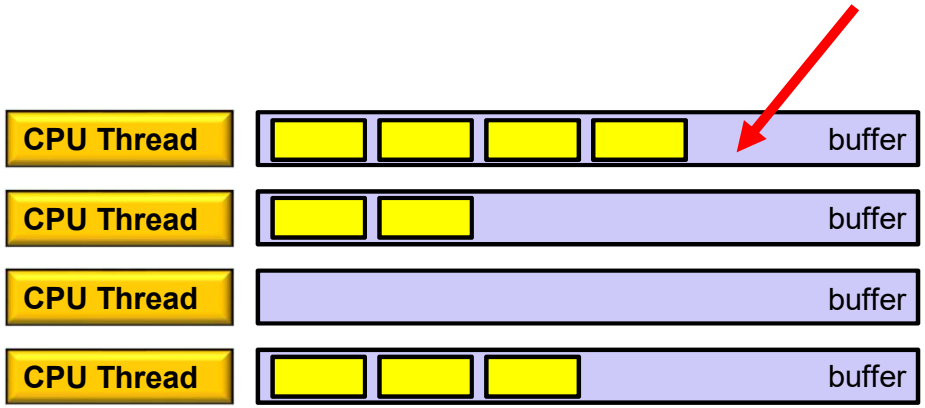
Vulkan Code has a Distinct "Style" of Setting Information in *structs* and then Passing that Information as a pointer-to-the-struct


```
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, OUT &MatrixBufferMemoryHandle
);

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

Oregon State University
Computer Graphics

Vulkan Command Buffers

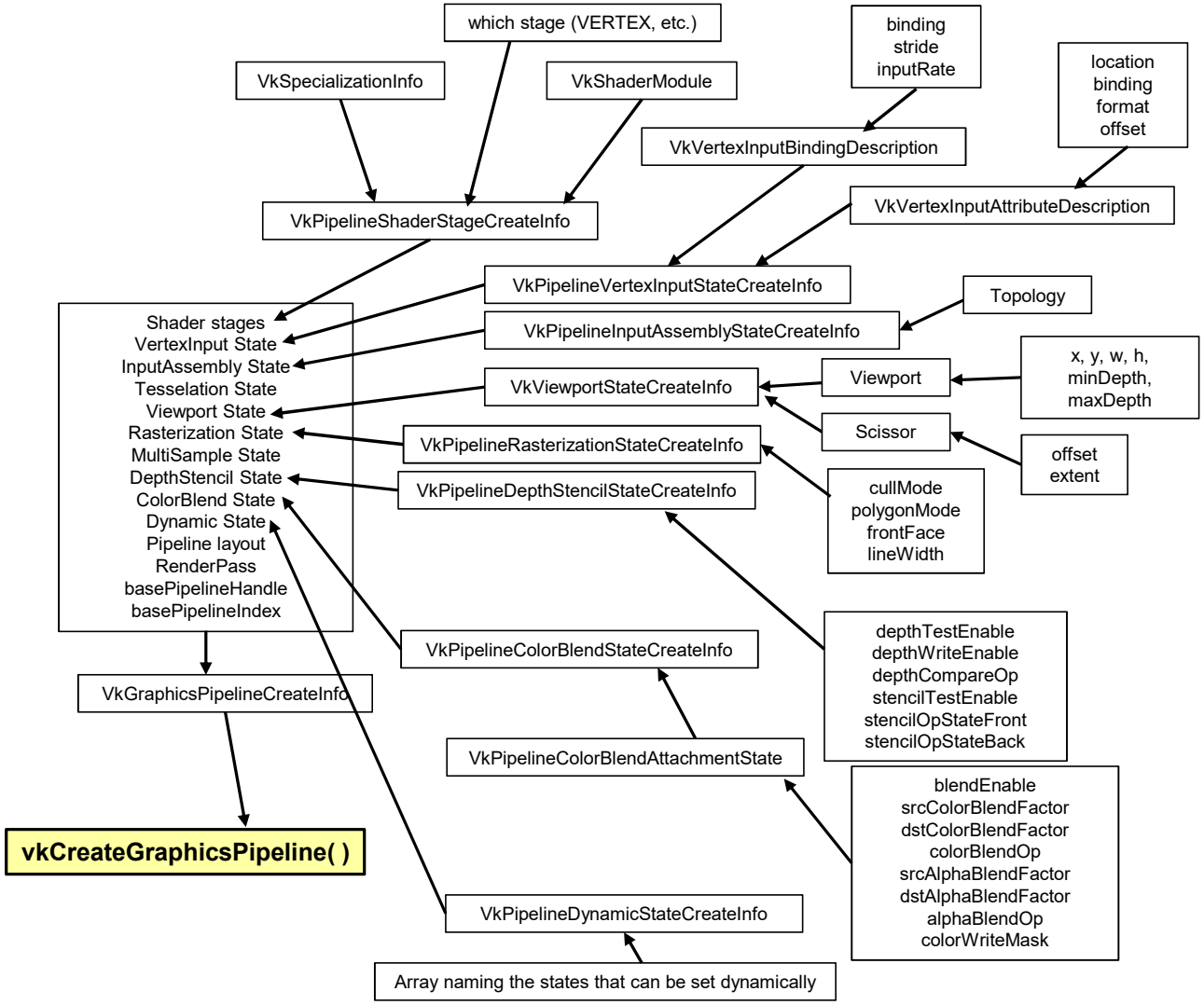
- Graphics commands are sent to command buffers
- Think OpenCL…
- E.g., *vkCmdDoSomething(cmdBuffer, …);*
- You can have as many simultaneous Command Buffers as you want
- Buffers are flushed when the application wants them flushed
- Each command buffer can be filled from a different thread (i.e., filling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



Vulkan Graphics Pipelines

- In OpenGL, your graphics “pipeline state” is whatever combination you most recently set: color, transformations, textures, shaders, etc.
- In OpenGL, changing the state 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 an entire PSO whenever you want to use that combination of state items
- Potentially, you could have thousands of these pre-prepared states – if there are N things to set, there would be N! possible combinations.
- Think of pipeline state as being immutable (although it isn’t in some very specific circumstances).
- This is a good time to talk about how game companies view Vulkan state … it’s not what you would think

Vulkan: Creating a Pipeline State Object from Individual State Items



Vulkan GPU Memory

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

Vulkan Render Passes

- 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

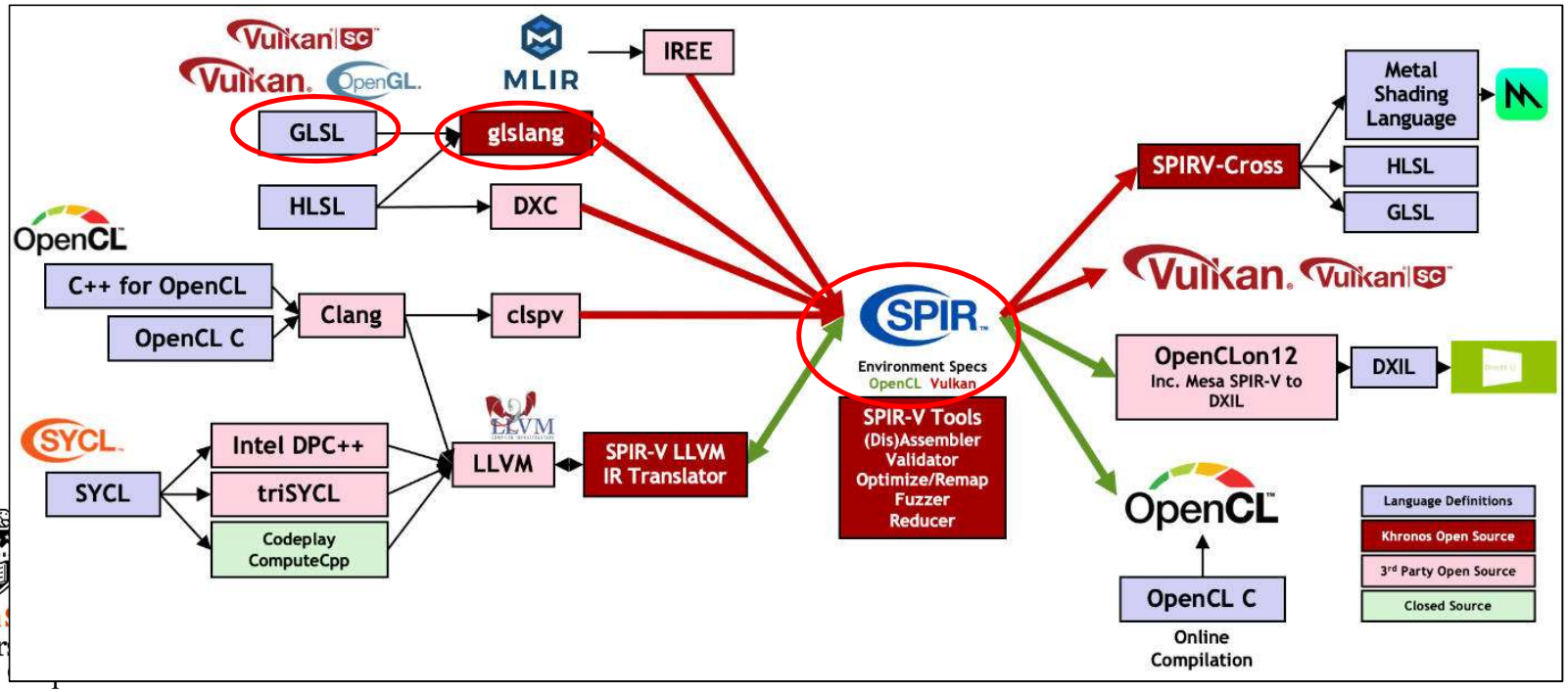
Vulkan Synchronization

- Synchronization is the responsibility of the *application*
- Events can be set, polled, and waited for (much like OpenCL)
- Vulkan does not ever synchronize – that's the application's (i.e., your) job
- Threads can concurrently read from the same object
- Threads can concurrently write to different objects

SPIR-V

SPIR-V stands for ***Standard Portable Intermediate Representation – Vulkan.*** It's the file format that Vulkan GLSL shaders get compiled into. The name of that front-end compiler is ***glslang***. At runtime, that file is read and the driver compiles it the rest of the way into the machine instruction set for that particular vendor.

SPIR-V started out as something for Vulkan but is now also used with OpenGL and OpenCL. Here is how it fits into the overall Khronos Ecosystem:

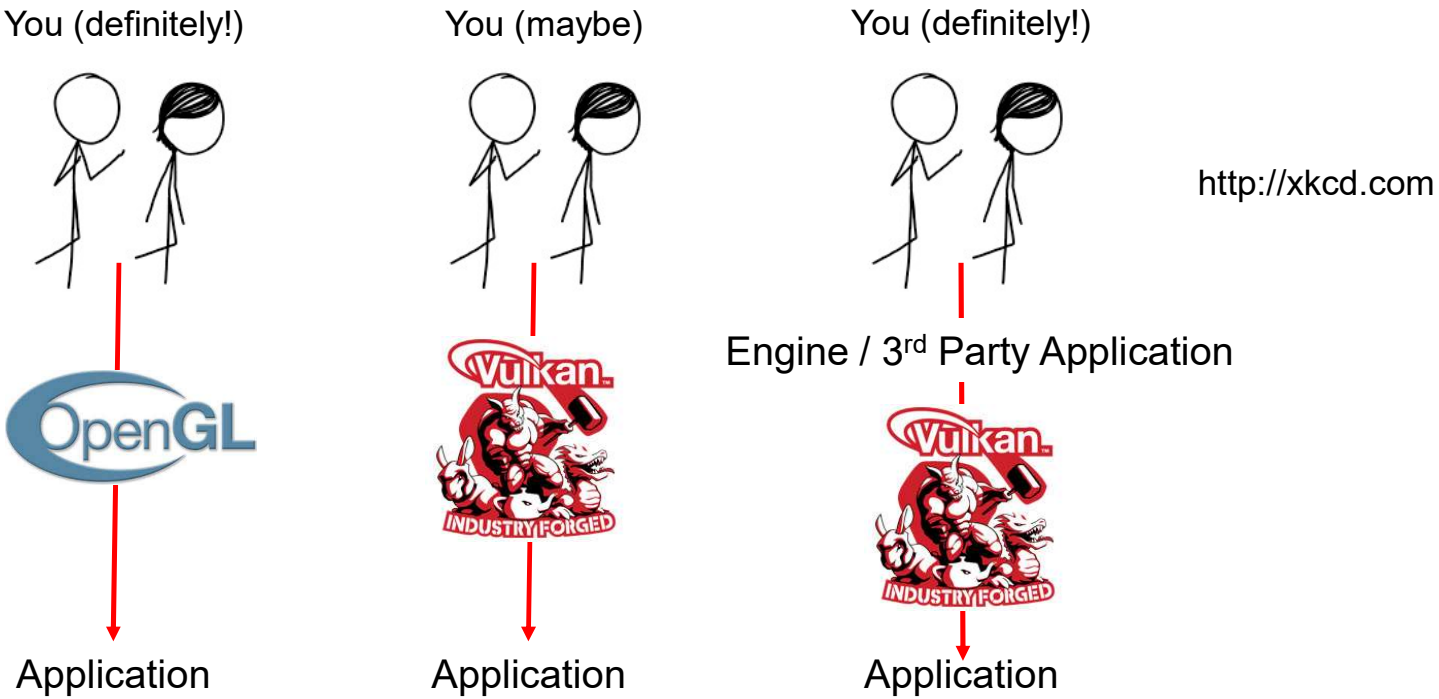


So What Do We All Do Now?

- I don't see Vulkan replacing OpenGL *ever*
- However, I wonder if Khronos will become less and less excited about adding new extensions to OpenGL. I see no evidence of this neglect right now.
- And, I also wonder if vendors will become less and less excited about improving OpenGL drivers. I see no evidence of this neglect right now.
- I see the OSU Vulkan class as always being a one-term standalone course, not part of another OpenGL-based course.

So What Do We All Do Now?

This is what I think the model of the immediate future is:



http://xkcd.com

You can learn more at: **http://cs.oregonstate.edu/~mjb/vulkan**