Vulkan

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

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

Who is the Real Vulkan?

Do 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.

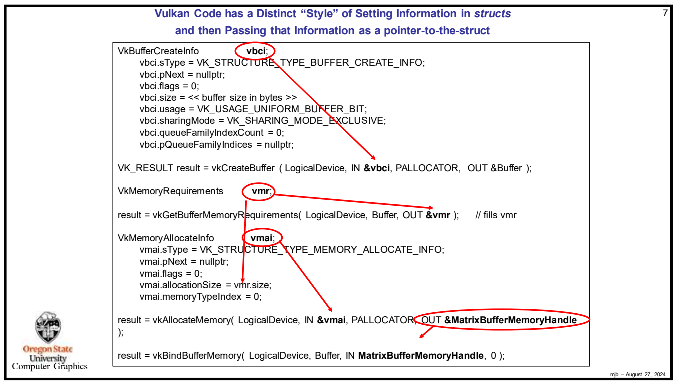
COLLADA DataFormat EGL glTF NNEF
OpenCL OpenGL ES OpenGL OpenGL SC OpenVG
OpenXR OpenVX SPIR SYCL Vulkan
WebGL

Playing "Where's Waldo" with Khronos Membership

Who's Been Specifically Working on Vulkan?

Vulkan

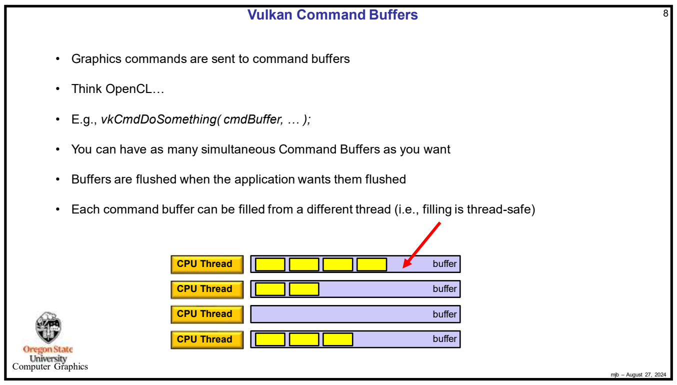
- Largely derived from AMD's *Mantle* API
- Also heavily influenced by Apple's *Metal* API and Microsoft's *DirectX 12*
- *There is no fixed-function ever – it is all shaders-based*
- Fortunately, the shader language Vulkan uses is GLSL with a few modifications
- Goal: much less driver complexity and overhead than OpenGL has
- Goal: much less user hand-holding
- Goal: able to do multithreaded graphics
- Goal: able to run on desktops and mobile devices



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

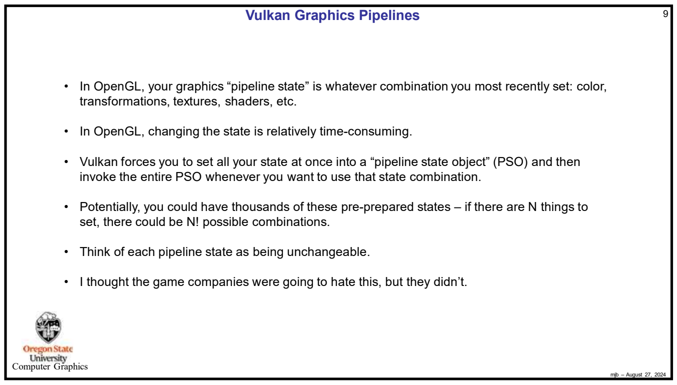


Vulkan Command Buffers

- 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)

CPU Thread
buffer
CPU Thread
buffer
CPU Thread
buffer
CPU Thread
buffer

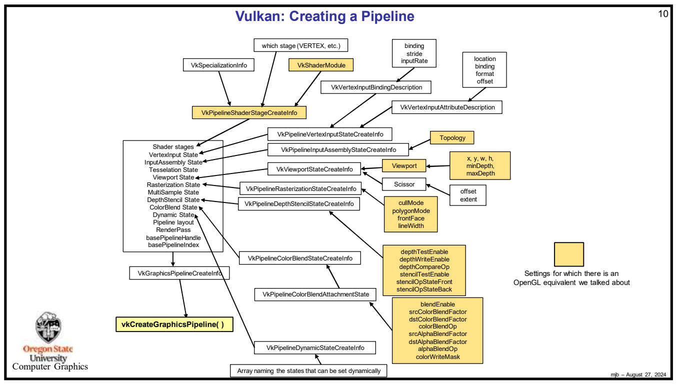
Oregon State University Computer Graphics



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 is relatively time-consuming.
- 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.
- Potentially, you could have thousands of these pre-prepared states – if there are N things to set, there could be N! possible combinations.
- Think of each pipeline state as being unchangeable.
- I thought the game companies were going to hate this, but they didn't.

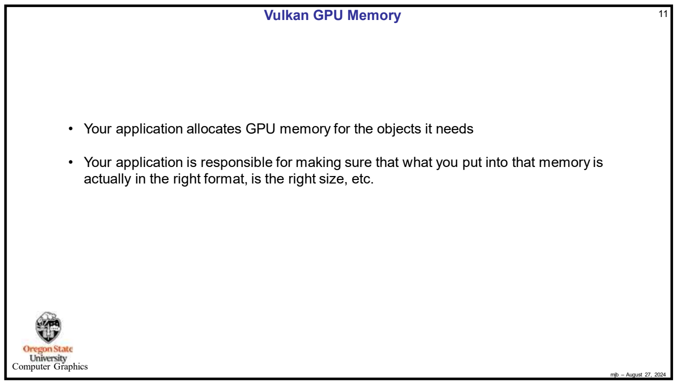
Oregon State University Computer Graphics



Vulkan: Creating a Pipeline

which stage (VERTEX, etc.)
VkSpecializationInfo
VkShaderModule
binding stride inputRate
location binding format offset
VkVertexInputBindingDescription
VkVertexInputAttributeDescription
VkPipelineShaderStageCreateInfo
VkPipelineVertexInputStateCreateInfo
VkPipelineInputAssemblyStateCreateInfo
Topology
Shader stages
VertexInput State
InputAssembly State
Tesselation State
Viewport State
Rasterization State
MultiSample State
DepthStencil State
ColorBlend State
Dynamic State
Pipeline layout
RenderPass
basePipelineHandle
basePipelineIndex
VkViewportStateCreateInfo
Viewport
x, y, w, h, minDepth, maxDepth
Scissor
offset extent
VkPipelineRasterizationStateCreateInfo
VkPipelineDepthStencilStateCreateInfo
cullMode polygonMode frontFace lineWidth
depthTestEnable depthWriteEnable depthCompareOp stencilTestEnable stencilOpStateFront stencilOpStateBack
VkPipelineColorBlendStateCreateInfo
VkGraphicsPipelineCreateInfo
VkPipelineColorBlendAttachmentState
blendEnable srcColorBlendFactor dstColorBlendFactor colorBlendOp srcAlphaBlendFactor dstAlphaBlendFactor alphaBlendOp colorWriteMask
vkCreateGraphicsPipeline( )
VkPipelineDynamicStateCreateInfo
Array naming the states that can be set dynamically
Settings for which there is an OpenGL equivalent we talked about

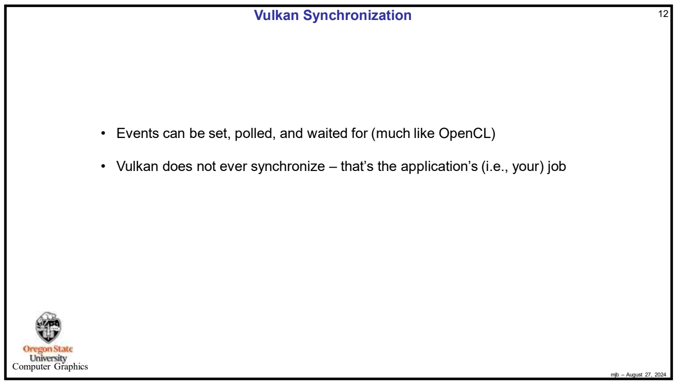
Oregon State University Computer Graphics



Vulkan GPU Memory

- Your application allocates GPU memory for the objects it needs
- 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.

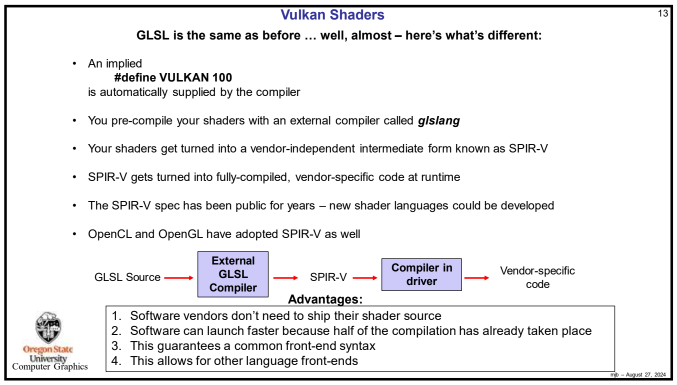
Oregon State University Computer Graphics



Vulkan Synchronization

- 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

Oregon State University Computer Graphics



Vulkan Shaders

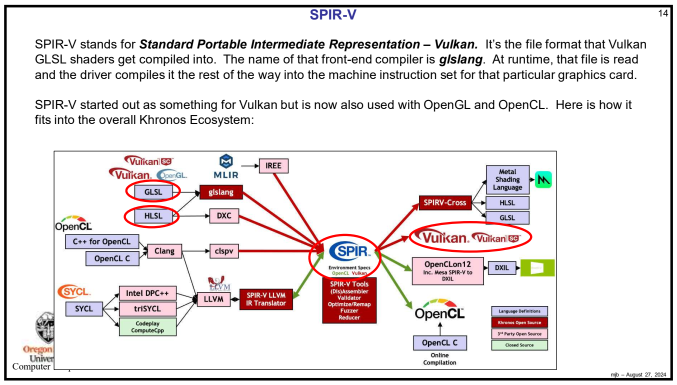
GLSL is the same as before … well, almost – here's what's different:

- An implied
  #define VULKAN 100
  is automatically supplied by the compiler
- You pre-compile your shaders with an external compiler called ***glslang***
- Your shaders get turned into a vendor-independent intermediate form known as SPIR-V
- SPIR-V gets turned into fully-compiled, vendor-specific code at runtime
- The SPIR-V spec has been public for years – new shader languages could be developed
- OpenCL and OpenGL have adopted SPIR-V as well

$$\begin{tikzcd}\text{GLSL Source} \arrow[r] & \boxed{\textbf{External GLSL Compiler}} \arrow[r] & \text{SPIR-V} \arrow[r] & \boxed{\textbf{Compiler in driver}} \arrow[r] & \text{Vendor-specific code}\end{tikzcd}$$

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

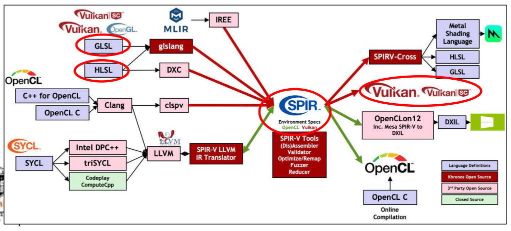
Oregon State University Computer Graphics



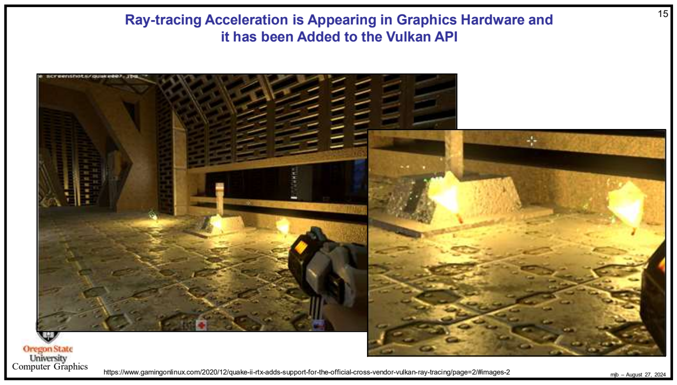
SPIR-V

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 graphics card.

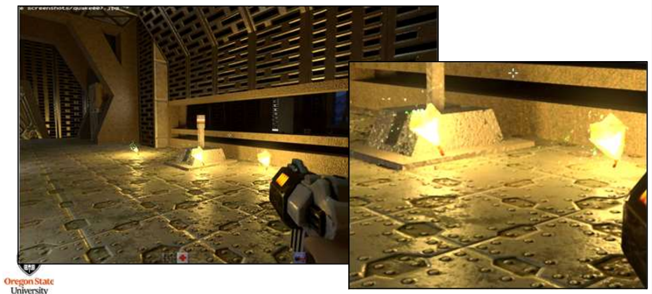
SPIR-V started out as something for Vulkan but is now also used with OpenGL and OpenCL. Here is how it fits into the overall Khronos Ecosystem:



Oregon State University Computer Graphics

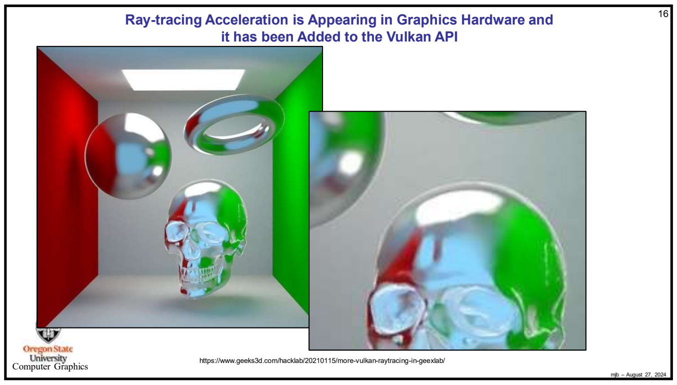


Ray-tracing Acceleration is Appearing in Graphics Hardware and it has been Added to the Vulkan API

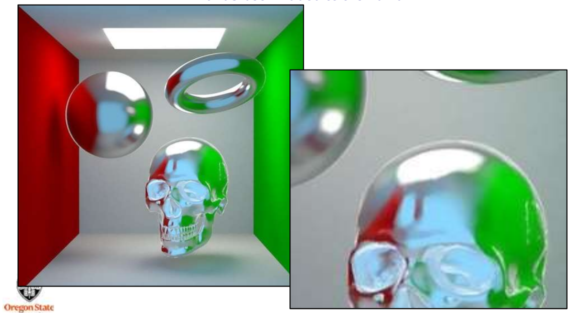


Oregon State University Computer Graphics

https://www.gamingonlinux.com/2020/12/quake-ii-rtx-adds-support-for-the-official-cross-vendor-vulkan-ray-tracing/page=2/#images-2

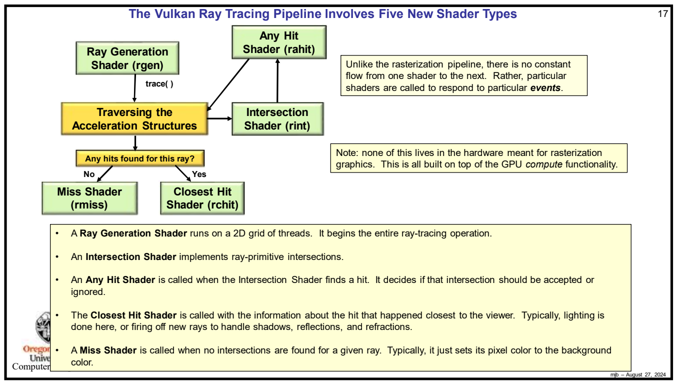


Ray-tracing Acceleration is Appearing in Graphics Hardware and it has been Added to the Vulkan API

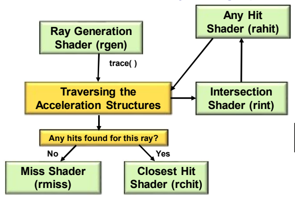


Oregon State University Computer Graphics

https://www.geeks3d.com/hacklab/20210115/more-vulkan-raytracing-in-geexlab/



The Vulkan Ray Tracing Pipeline Involves Five New Shader Types

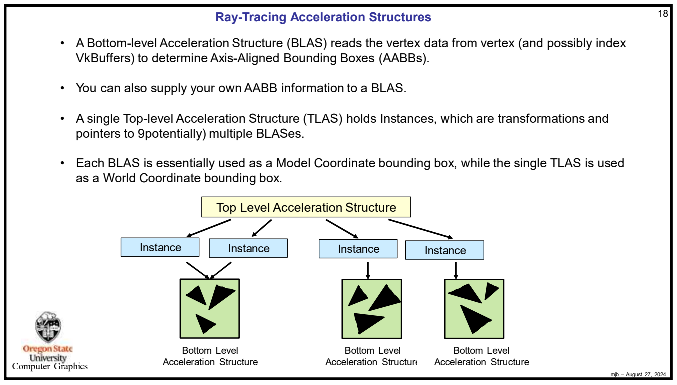

Ray Generation Shader (rgen)
trace( )
Any Hit Shader (rahit)
Traversing the Acceleration Structures
Intersection Shader (rint)
Any hits found for this ray?
No
Yes
Miss Shader (rmiss)
Closest Hit Shader (rchit)

Unlike the rasterization pipeline, there is no constant flow from one shader to the next. Rather, particular shaders are called to respond to particular ***events***.

Note: none of this lives in the hardware meant for rasterization graphics. This is all built on top of the GPU *compute* functionality.

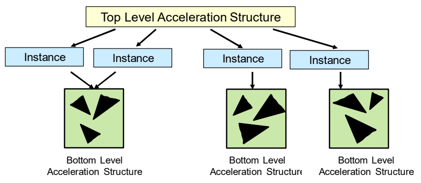
- A **Ray Generation Shader** runs on a 2D grid of threads. It begins the entire ray-tracing operation.
- An **Intersection Shader** implements ray-primitive intersections.
- An **Any Hit Shader** is called when the Intersection Shader finds a hit. It decides if that intersection should be accepted or ignored.
- The **Closest Hit Shader** is called with the information about the hit that happened closest to the viewer. Typically, lighting is done here, or firing off new rays to handle shadows, reflections, and refractions.
- A **Miss Shader** is called when no intersections are found for a given ray. Typically, it just sets its pixel color to the background color.

Oregon State University Computer Graphics

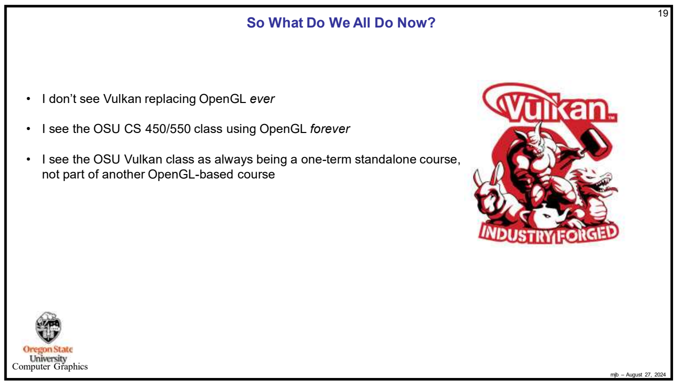


Ray-Tracing Acceleration Structures

- A Bottom-level Acceleration Structure (BLAS) reads the vertex data from vertex (and possibly index VkBuffers) to determine Axis-Aligned Bounding Boxes (AABBs).
- You can also supply your own AABB information to a BLAS.
- A single Top-level Acceleration Structure (TLAS) holds Instances, which are transformations and pointers to 9potentially) multiple BLASes.
- Each BLAS is essentially used as a Model Coordinate bounding box, while the single TLAS is used as a World Coordinate bounding box.


Top Level Acceleration Structure
Instance
Instance
Instance
Instance
Bottom Level Acceleration Structure
Bottom Level Acceleration Structure
Bottom Level Acceleration Structure

Oregon State University Computer Graphics



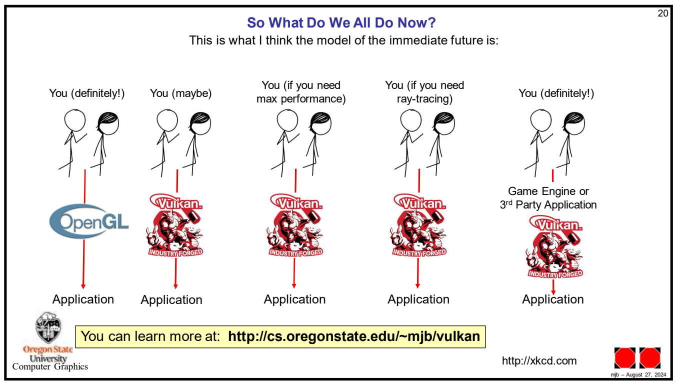
So What Do We All Do Now?

19

- I don't see Vulkan replacing OpenGL *ever*
- I see the OSU CS 450/550 class using OpenGL *forever*
- I see the OSU Vulkan class as always being a one-term standalone course, not part of another OpenGL-based course

Oregon State University Computer Graphics

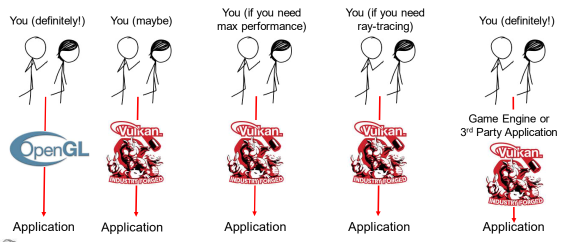
mjb – August 27, 2024



So What Do We All Do Now?

20

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



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

http://xkcd.com

Oregon State University Computer Graphics

mjb – August 27, 2024