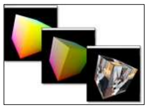
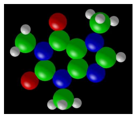
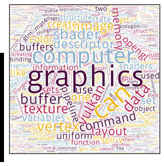
SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

The Vulkan Computer Graphics API

Mike Bailey
mjb@cs.oregonstate.edu







SIGGRAPH '23 Courses, August 06-10, 2023, Los Angeles, CA, USA
ACM 979-8-4007-0145-0/23/08.
10.1145/3587423.3595529

http://cs.oregonstate.edu/~mjb/vulkan

Mike Bailey

- Professor of Computer Science, Oregon State University
- Has been in computer graphics for over 30 years
- Has had over 11,000 students in his university classes
- Has taught over 100 conference and workshop short courses
- mjb@cs.oregonstate.edu

Oregon State University Computer Graphics

Welcome! I'm *happy* to be here. I hope you are too !

http://cs.oregonstate.edu/~mjb/vulkan

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

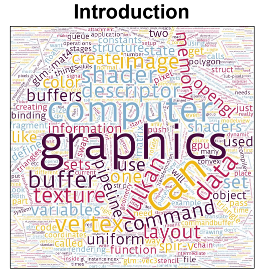
Course Goals

- **Better understand what Vulkan is and when it would be a good application solution (and when it wouldn't)**
- **Leave you with working, understandable Vulkan code**
- **Understand the inner workings of Vulkan, especially the parts that are different from OpenGL and Direct X 11**

http://cs.oregonstate.edu/~mjb/vulkan

Vulkan.

Introduction



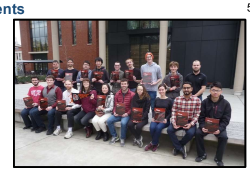
http://cs.oregonstate.edu/~mjb/vulkan

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

Acknowledgements

Oregon State University

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 always for your courage and patience!



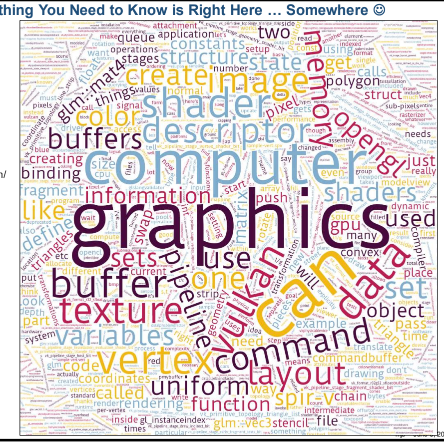
Second, thanks to NVIDIA for all of your support! These courses could not have ever happened without you!

Third, thanks to the Khronos Group for the great Vulkan teaching materials and other swag! (Look at those happy faces in the photo holding the reference cards.)

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

Everything You Need to Know is Right Here … Somewhere ☺

https://www.wordclouds.com/



SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

Top Three Reasons that Prompted the Development of Vulkan

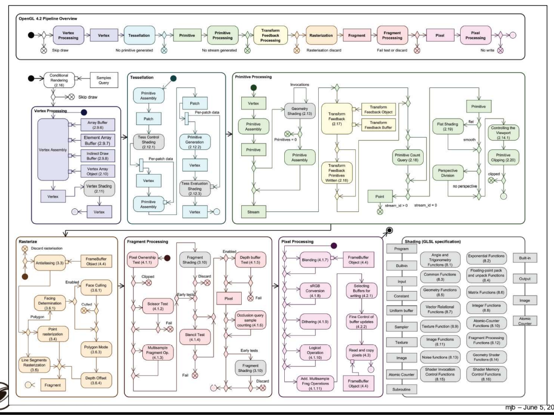
1. Performance
2. Performance
3. Performance

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.

OpenGL 4.2 Pipeline Flowchart



Why is it so important to keep the GPU Busy?

Graphics Processor	
GPU Name:	AD102
GPU Variant:	AD102-300-A1
Architecture:	Ada Lovelace
Foundry:	TSMC
Process Size:	5 nm
Transistors:	76,300 million
Density:	125.5M / mm²
Die Size:	608 mm²

Graphics Card	
Release Date:	Sep 20th, 2022
Availability:	Oct 12th, 2022
Generation:	GeForce 40
Predecessor:	GeForce 30
Production:	Active
Launch Price:	1,599 USD
Current Price:	Amazon / Newegg
Bus Interface:	PCIe 4.0 x16
Reviews:	66 in our database

Clock Speeds	
Base Clock:	2235 MHz
Boost Clock:	2520 MHz
Memory Clock:	1313 MHz 21 Gbps effective

Memory	
Memory Size:	24 GB
Memory Type:	GDDR6X
Memory Bus:	384 bit
Bandwidth:	1,008 GB/s

Render Config	
Shading Units:	16384
TMUs:	512
ROPs:	176
SM Count:	128
Tensor Cores:	512
RT Cores:	128
L1 Cache:	128 KB (per SM)
L2 Cache:	72 MB

Board Design	
Slot Width:	Triple-slot
Length:	304 mm 12 inches
Width:	137 mm 5.4 inches
Height:	61 mm 2.4 inches
TDP:	450 W
Suggested PSU:	850 W
Outputs:	1x HDMI 2.1 3x DisplayPort 1.4a
Power Connectors:	1x 16-pin
Board Number:	PG139 SKU 330

Graphics Features	
DirectX:	12 Ultimate (12_2)
OpenGL:	4.6
OpenCL:	3.0
Vulkan:	1.3
CUDA:	8.9
Shader Model:	6.7

Theoretical Performance	
Pixel Rate:	443.5 GPixel/s
Texture Rate:	1,290 GTexel/s
FP16 (half):	82.58 TFLOPS (1:1)
FP32 (float):	82.58 TFLOPS
FP64 (double):	1,290 GFLOPS (1:64)

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

PROMOTER MEMBERS

Over 100 members worldwide
Any company is welcome to join

Who's Been Specifically Working on Vulkan?

Vulkan

13

- Originally 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
- Goal: higher single-threaded performance than OpenGL can deliver
- Goal: able to do multithreaded graphics

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Vulkan Differences from OpenGL

14

- 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 and texture functionality must be done in shaders.
- Shaders are pre-"half-compiled" outside of your application. The compilation process is then finished during the runtime pipeline-building process.

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Moving part of the driver into the application

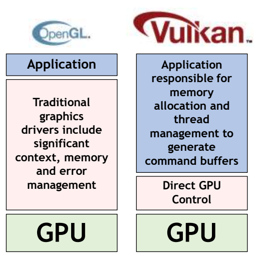
15

Complex drivers lead to driver overhead and cross vendor unpredictability

Error management is always active

Driver processes full shading language source

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

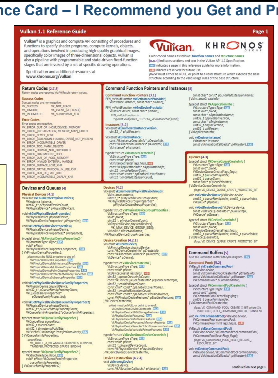
SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Vulkan Reference Card – I Recommend you Get and Print This!

16

Even though we are up to Vulkan 1.3, the Reference Card is 1.1



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

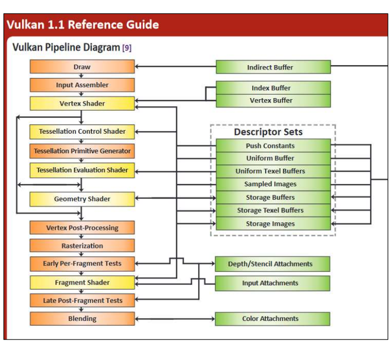
SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Vulkan Reference Card

17

Even though we are up to Vulkan 1.3, the Reference Card is 1.1



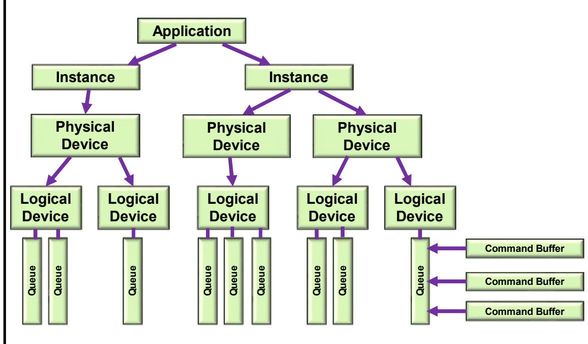
https://www.khronos.org/files/vulkan11-reference-guide.pdf

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Vulkan Highlights: Overall Block Diagram

18

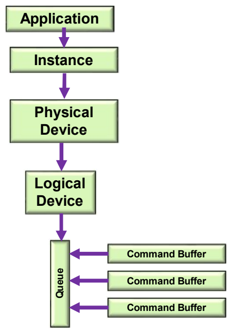


SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Vulkan Highlights: a More Typical Block Diagram

19



SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Steps in Creating Graphics using Vulkan

20

1. Create the Vulkan 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- …

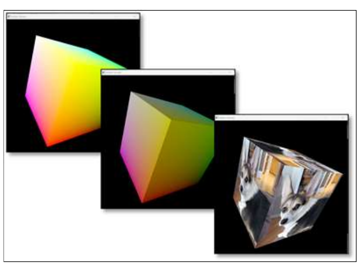
SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

21

Vulkan.

The Vulkan Sample Code Included with These Notes

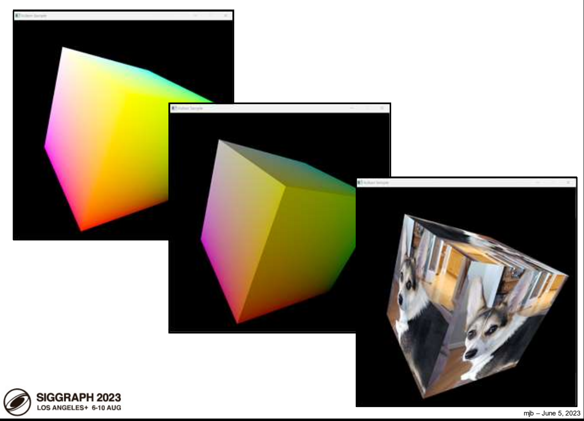


SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Sample Program Output

22

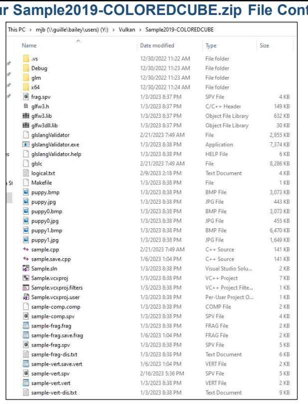


SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Your Sample2019-COLOREDCUBE.zip File Contains This

23



• • •

SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

The "19" refers to the version of Visual Studio, not the year of development.

mjb – June 5, 2023

Sample Program Keyboard Inputs

24

'l' (ell), 'L':	Toggle **l**ighting off and on
'm', 'M':	Toggle display **m**ode (textures vs. colors, for now)
'p', 'P':	**P**ause the animation
'q', 'Q':	**q**uit the program
Esc:	quit the program
'r', 'R':	Toggle **r**otation-animation and using the mouse
'i', 'I':	Toggle using a vertex buffer only vs. an **i**ndex buffer (in the index buffer version)
'1', …,'9','a',…'g'	Set the number of instances (in the instancing version)

SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Caveats on the Sample Code, I

1. I've written everything out in appalling longhand.
2. Everything is in one .cpp file (except the geometry data). It really should be broken up, but this way you can find everything easily.
3. At times, I could have hidden complexity, but I didn't. At all stages, I have tried to err on the side of showing you *everything*, so that nothing happens in a way that's kept a secret from you.
4. I've setup Vulkan structs every time they are used, even though, in many cases (most?), they could have been setup once and then re-used each time.
5. At times, I've setup things that didn't need to be setup just to show you what could go there.

Caveats on the Sample Code, II

6. There are great uses for C++ classes and methods here to hide some complexity, but I've not done that.
7. I've typedef'ed a couple things to make the Vulkan phraseology more consistent.
8. Even though it is not good software style, I have put persistent information in global variables, rather than a separate data structure
9. At times, I have copied lines from vulkan_core.h into the code as comments to show you what certain options could be.
10. I've divided functionality up into the pieces that make sense to me. Many other divisions are possible. Feel free to invent your own.

Main Program


```
int
main( int argc, char * argv[ ] )
{
	Width  = 1024;
	Height = 1024;

	errno_t err = fopen_s( &FpDebug, DEBUGFILE, "w" );
	if( err != 0 )
	{
		fprintf( stderr, "Cannot open debug print file '%s'\n", DEBUGFILE );
		FpDebug = stderr;
	}
	fprintf(FpDebug, "FpDebug: Width = %d ; Height = %d\n", Width, Height);

	Reset( );
	InitGraphics( );

	// loop until the user closes the window:

	while( glfwWindowShouldClose( MainWindow ) == 0 )
	{
		glfwPollEvents( );
		Time = glfwGetTime( );		// elapsed time, in double-precision seconds
		UpdateScene( );
		RenderScene( );
	}

	fprintf(FpDebug, "Closing the GLFW window\n");

	vkQueueWaitIdle( Queue );
	vkDeviceWaitIdle( LogicalDevice );
	DestroyAllVulkan( );
	glfwDestroyWindow( MainWindow );
	glfwTerminate( );
	return 0;
}
```


InitGraphics( ), I


```
void
InitGraphics( )
{
	HERE_I_AM( "InitGraphics" );

	VkResult result = VK_SUCCESS;

	Init01Instance( );

	InitGLFW( );

	Init02CreateDebugCallbacks( );

	Init03PhysicalDeviceAndGetQueueFamilyProperties( );

	Init04LogicalDeviceAndQueue( );

	Init05UniformBuffer( sizeof(Matrices),      &MyMatrixUniformBuffer );
	Fill05DataBuffer( MyMatrixUniformBuffer,    (void *) &Matrices );

	Init05UniformBuffer( sizeof(Light),     &MyLightUniformBuffer );
	Fill05DataBuffer( MyLightUniformBuffer, (void *) &Light );

	Init05MyVertexDataBuffer(  sizeof(VertexData), &MyVertexDataBuffer );
	Fill05DataBuffer( MyVertexDataBuffer,           (void *) VertexData );

	Init06CommandPool( );
	Init06CommandBuffers( );
```


InitGraphics( ), II


```
	Init07TextureSampler( &MyPuppyTexture.texSampler );
	Init07TextureBufferAndFillFromBmpFile("puppy.bmp", &MyPuppyTexture);

	Init08Swapchain( );

	Init09DepthStencilImage( );

	Init10RenderPasses( );

	Init11Framebuffers( );

	Init12SpirvShader( "sample-vert.spv", &ShaderModuleVertex );
	Init12SpirvShader( "sample-frag.spv", &ShaderModuleFragment );

	Init13DescriptorSetPool( );
	Init13DescriptorSetLayouts();
	Init13DescriptorSets( );

	Init14GraphicsVertexFragmentPipeline( ShaderModuleVertex, ShaderModuleFragment,
				VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST, &GraphicsPipeline  );
}
```


Vulkan Software Philosophy

Vulkan has lots of typedefs that define C/C++ structs and enums

Vulkan takes a non-C++ object-oriented approach in that those typedef'ed structs pass all the necessary information into a function. For example, where we might normally say, using C++ class methods:

result = **LogicalDevice->**vkGetDeviceQueue (queueFamilyIndex, queueIndex, OUT &Queue);

Vulkan has chosen to do it like this:

result = vkGetDeviceQueue (**LogicalDevice**, queueFamilyIndex, queueIndex, OUT &Queue);

Vulkan Conventions

31

VkXxx is a typedef, probably a struct

vkYyy( ) is a function call

VK_ZZZ is a constant

My Conventions

"Init" in a function call name means that something is being setup that only needs to be setup once

The number after "Init" gives you the ordering

In the source code, after main( ) comes InitGraphics( ), then all of the InitxxYYY( ) functions in numerical order. After that comes the helper functions

"Find" in a function call name means that something is being looked for

"Fill" in a function call name means that some data is being supplied to Vulkan

"IN" and "OUT" ahead of function call arguments are just there to let you know how an argument is going to be used by the function. Otherwise, IN and OUT have no significance. They are actually #define'd to nothing.

Querying the Number of Something and Allocating Enough Structures to Hold Them All

32

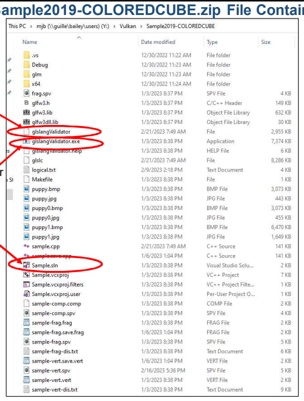
```
uint32_t  count;
result = vkEnumeratePhysicalDevices( Instance, OUT &count, OUT (VkPhysicalDevice *)nullptr );

VkPhysicalDevice * physicalDevices = new VkPhysicalDevice[ count ];
result = vkEnumeratePhysicalDevices( Instance, OUT &count, OUT &physicalDevices[0] );
```

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[0]);

Your Sample2019-COLOREDCUBE.zip File Contains This

33



Linux shader compiler

Windows shader compiler

Double-click here to launch Visual Studio 2019 with this solution

The "19" refers to the version of Visual Studio, not the year of development.

Vulkan Program Flow – the Setup

34

- Create a GLFW Vulkan Window
- Query the Physical Devices and Choose (1 in our case)
- Decide on the Extensions and Layers You Want
- Create the Logical Device
- Create the Queue(s) (1 in our case)
- Allocate and Fill memory for the Vertices and Indices
- Allocate and Fill memory for the Uniform Buffers
- Create the Command Buffers (3 in our case)
- If using Textures, create the Sampler, Read the Texture, and move it to Device Local Memory
- Create the Swap Chain (2 images in our case)
- Be sure you have Compiled the Shaders into .spv files
- Create the Descriptor Set Data Structures
- Create the Graphics Pipeline Data Structure Layout(s)
- Fill the Graphics Pipeline Data Structure(s)

Vulkan Program Flow – the Rendering Loop

35

```
while( the GLFW Window should not close )
{
    UpdateScene( )
    RenderScene( )
}
```

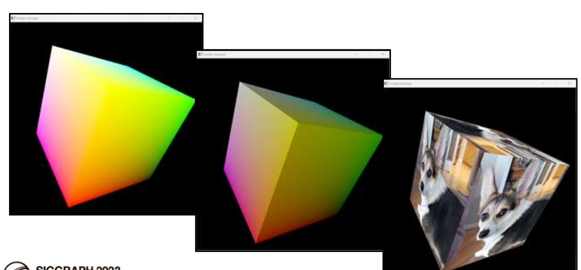
UpdateScene( ):
- Create the Transformations
- Fill the Uniform Buffers

RenderScene( ):
- Acquire the Next Swap Chain Image
- Begin its Command Buffer
  - Create the RenderPass with the Framebuffer information
    - for(all the different Graphics Pipeline Data Structures being used)
      {
      - Bind that Graphics Pipeline Data Structure
      - Set any Dynamic State Variables
      - Bind the Proper Descriptor Set Values
      - Do the Drawing
      }
  - End the RenderPass
- End the Command Buffer
- Submit the Command Buffer to a Queue
- Wait for the Queue to Finish Submitting
- Present the Image to the Viewer

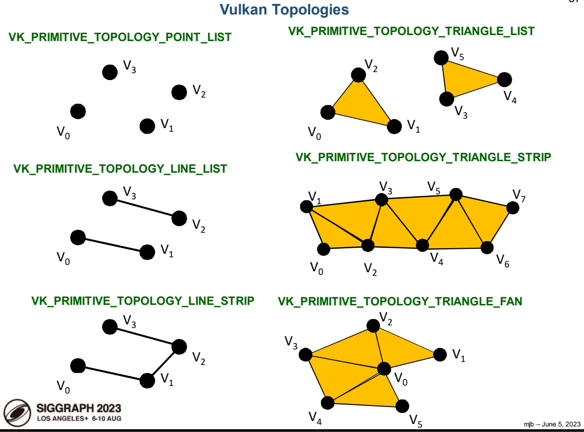
36

Vulkan.

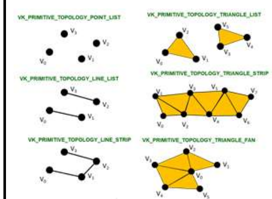
Drawing



Vulkan Topologies



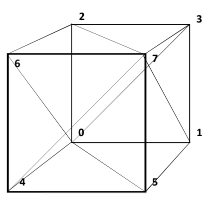
Vulkan Topologies

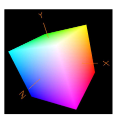


The same as OpenGL topologies, with a few left out.

```
typedef enum VkPrimitiveTopology
{
    VK_PRIMITIVE_TOPOLOGY_POINT_LIST
    VK_PRIMITIVE_TOPOLOGY_LINE_LIST
    VK_PRIMITIVE_TOPOLOGY_LINE_STRIP
    VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST
    VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP
    VK_PRIMITIVE_TOPOLOGY_TRIANGLE_FAN
    VK_PRIMITIVE_TOPOLOGY_LINE_LIST_WITH_ADJACENCY
    VK_PRIMITIVE_TOPOLOGY_LINE_STRIP_WITH_ADJACENCY
    VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST_WITH_ADJACENCY
    VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP_WITH_ADJACENCY
    VK_PRIMITIVE_TOPOLOGY_PATCH_LIST
} VkPrimitiveTopology;
```


A Colored Cube Example





```
static GLfloat CubeColors[ ][3] =
{
        { 0., 0., 0. },
        { 1., 0., 0. },
        { 0., 1., 0. },
        { 1., 1., 0. },
        { 0., 0., 1. },
        { 1., 0., 1. },
        { 0., 1., 1. },
        { 1., 1., 1. },
};
```

This data is contained in the file ***SampleVertexData.cpp***

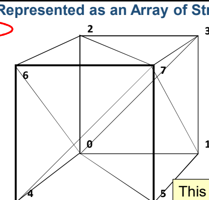
```
static GLfloat CubeVertices[ ][3] =
{
        { -1., -1., -1. },
        {  1., -1., -1. },
        { -1.,  1., -1. },
        {  1.,  1., -1. },
        { -1., -1.,  1. },
        {  1., -1.,  1. },
        { -1.,  1.,  1. },
        {  1.,  1.,  1. }
};
```

```
static GLuint CubeTriangleIndices[ ][3] =
{
        { 0, 2, 3 },
        { 0, 3, 1 },
        { 4, 5, 7 },
        { 4, 7, 6 },
        { 1, 3, 7 },
        { 1, 7, 5 },
        { 0, 4, 6 },
        { 0, 6, 2 },
        { 2, 6, 7 },
        { 2, 7, 3 },
        { 0, 1, 5 }
        { 0, 5, 4 }
};
```


Triangles Represented as an Array of Structures

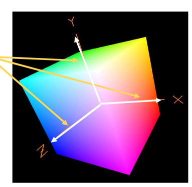
From the file **SampleVertexData.cpp**:

```
struct vertex
{
        glm::vec3    position;
        glm::vec3    normal;
        glm::vec3    color;
        glm::vec2    texCoord;
};

struct vertex VertexData[ ] =
{
        // triangle 0-2-3:
        // vertex #0:
        {
                { -1., -1., -1. },
                {  0.,  0., -1. },
                {  0.,  0.,  0. },
                {  1., 0. }
        },

        // vertex #2:
        {
                { -1.,  1., -1. },
                {  0.,  0., -1. },
                {  0.,  1.,  0. },
                {  1., 1. }
        },

        // vertex #3:
        {
                {  1.,  1., -1. },
                {  0.,  0., -1. },
                {  1.,  1.,  0. },
                {  0., 1. }
        },
```



This data is contained in the file ***SampleVertexData.cpp***

Modeled in right-handed coordinates

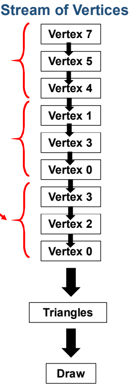


Non-indexed Buffer Drawing

From the file **SampleVertexData.cpp**:

```
struct vertex
{
        glm::vec3    position;
        glm::vec3    normal;
        glm::vec3    color;
        glm::vec2    texCoord;
};

struct vertex VertexData[ ] =
{
        // triangle 0-2-3:
        // vertex #0:
        {
                { -1., -1., -1. },
                {  0.,  0., -1. },
                {  0.,  0.,  0. },
                {  1., 0. }
        },

        // vertex #2:
        {
                { -1.,  1., -1. },
                {  0.,  0., -1. },
                {  0.,  1.,  0. },
                {  1., 1. }
        },

        // vertex #3:
        {
                {  1.,  1., -1. },
                {  0.,  0., -1. },
                {  1.,  1.,  0. },
                {  0., 1. }
        },
```

Stream of Vertices



Initializing and Filling the Vertex Buffer


```
struct vertex  VertexData[ ] =
{
        . . .
};

MyBuffer          MyVertexDataBuffer;

        . . .

Init05MyVertexDataBuffer(  sizeof(VertexData), OUT &MyVertexDataBuffer );    // create
Fill05DataBuffer( MyVertexDataBuffer,           (void *) VertexData );       // fill

        . . .

VkResult
Init05MyVertexDataBuffer( IN VkDeviceSize size, OUT MyBuffer * pMyBuffer )
{
        VkResult result;
        result = Init05DataBuffer( size, VK_BUFFER_USAGE_VERTEX_BUFFER_BIT, pMyBuffer );
        return result;
}

VkResult
Fill05DataBuffer( IN MyBuffer myBuffer, IN void * data )
{
        . . .
}
```

A Preview of What *Init05DataBuffer* Does

43

```
VkResult
Init05DataBuffer( VkDeviceSize size, VkBufferUsageFlags usage, OUT MyBuffer * pMyBuffer )
{
	VkResult result = VK_SUCCESS;
	VkBufferCreateInfo  vbci;
		vbci.sType = VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO;
		vbci.pNext = nullptr;
		vbci.flags = 0;
		vbci.size = pMyBuffer->size = size;
		vbci.usage = usage;
		vbci.sharingMode = VK_SHARING_MODE_EXCLUSIVE;
		vbci.queueFamilyIndexCount = 0;
		vbci.pQueueFamilyIndices = (const uint32_t *)nullptr;
	result = vkCreateBuffer ( LogicalDevice, IN &vbci, PALLOCATOR,  OUT &pMyBuffer->buffer );

	VkMemoryRequirements		vmr;
	vkGetBufferMemoryRequirements( LogicalDevice, IN pMyBuffer->buffer, OUT &vmr );		// fills vmr

	VkMemoryAllocateInfo		vmai;
		vmai.sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
		vmai.pNext = nullptr;
		vmai.allocationSize = vmr.size;
		vmai.memoryTypeIndex = FindMemoryThatIsHostVisible( );

	VkDeviceMemory		vdm;
	result = vkAllocateMemory( LogicalDevice, IN &vmai, PALLOCATOR, OUT &vdm );
	pMyBuffer->vdm = vdm;

	result = vkBindBufferMemory( LogicalDevice, pMyBuffer->buffer, IN vdm, 0 );		// 0 is the offset
	return result;
}
```


Telling the Pipeline about its Input

44

We will come to the Pipeline later, but for now, know that a Vulkan pipeline is essentially a very large data structure that holds (what OpenGL would call) the **state**, including how to parse its input.

C/C++:

```
struct vertex
{
	glm::vec3	position;
	glm::vec3	normal;
	glm::vec3	color;
	glm::vec2	texCoord;
};
```

GLSL Shader:

```
layout( location = 0 ) in vec3 aVertex;
layout( location = 1 ) in vec3 aNormal;
layout( location = 2 ) in vec3 aColor;
layout( location = 3 ) in vec2 aTexCoord;
```

```
VkVertexInputBindingDescription		vvibd[1];	// one of these per buffer data buffer
	vvibd[0].binding = 0;					// which binding # this is
	vvibd[0].stride = sizeof( struct vertex );		// bytes between successive structs
	vvibd[0].inputRate = VK_VERTEX_INPUT_RATE_VERTEX;	// read one value per vertex
```

Always use the C/C++ **sizeof( )** construct rather than hardcoding the byte count!

Telling the Pipeline about its Input

45

```
struct vertex
{
	glm::vec3	position;
	glm::vec3	normal;
	glm::vec3	color;
	glm::vec2	texCoord;
};
```

```
layout( location = 0 ) in vec3 aVertex;
layout( location = 1 ) in vec3 aNormal;
layout( location = 2 ) in vec3 aColor;
layout( location = 3 ) in vec2 aTexCoord;
```

```
VkVertexInputAttributeDescription	vviad[4];		// array per vertex input attribute
	// 4 = vertex, normal, color, texture coord
	vviad[0].location = 0;			// location in the layout decoration
	vviad[0].binding = 0;			// which binding description this is part of
	vviad[0].format = VK_FORMAT_VEC3;	// x, y, z
	vviad[0].offset = offsetof( struct vertex, position );		// 0

	vviad[1].location = 1;
	vviad[1].binding = 0;
	vviad[1].format = VK_FORMAT_VEC3;	// nx, ny, nz
	vviad[1].offset = offsetof( struct vertex, normal );		// 12

	vviad[2].location = 2;
	vviad[2].binding = 0;
	vviad[2].format = VK_FORMAT_VEC3;	// r, g, b
	vviad[2].offset = offsetof( struct vertex, color );		// 24

	vviad[3].location = 3;
	vviad[3].binding = 0;
	vviad[3].format = VK_FORMAT_VEC2;	// s, t
	vviad[3].offset = offsetof( struct vertex, texCoord );		// 36
```

Always use the C/C++ construct ***offsetof***, rather than hardcoding the byte offset!

Telling the Pipeline Data Structure about its Input

46

We will come to the Pipeline Data Structure later, but for now, know that a Vulkan Pipeline is essentially a very large data structure that holds (what OpenGL would call) the state, including how to parse its vertex input.

```
VkPipelineVertexInputStateCreateInfo		vpvisci;	// used to describe the input vertex attributes
	vpvisci.sType = VK_STRUCTURE_TYPE_PIPELINE_VERTEX_INPUT_STATE_CREATE_INFO;
	vpvisci.pNext = nullptr;
	vpvisci.flags = 0;
	vpvisci.vertexBindingDescriptionCount = 1;
	vpvisci.pVertexBindingDescriptions = vvibd;
	vpvisci.vertexAttributeDescriptionCount = 4;
	vpvisci.pVertexAttributeDescriptions = vviad;

VkPipelineInputAssemblyStateCreateInfo		vpiasci;
	vpiasci.sType = VK_STRUCTURE_TYPE_PIPELINE_INPUT_ASSEMBLY_STATE_CREATE_INFO;
	vpiasci.pNext = nullptr;
	vpiasci.flags = 0;
	vpiasci.topology = VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST;
```


Telling the Pipeline Data Structure about its Input

47

We will come to the Pipeline Data Structure later, but for now, know that a Vulkan Pipeline is essentially a very large data structure that holds (what OpenGL would call) the state, including how to parse its vertex input.

```
VkGraphicsPipelineCreateInfo			vgpci;
	vgpci.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
	vgpci.pNext = nullptr;
	vgpci.flags = 0;
	vgpci.stageCount = 2;			// number of shader stages in this pipeline
	vgpci.pStages = vpssci;
	vgpci.pVertexInputState = &vpvisci;
	vgpci.pInputAssemblyState = &vpiasci;
	vgpci.pTessellationState = (VkPipelineTessellationStateCreateInfo *)nullptr;		// &vptsci
	vgpci.pViewportState = &vpvsci;
	vgpci.pRasterizationState = &vprsci;
	vgpci.pMultisampleState = &vpmsci;
	vgpci.pDepthStencilState = &vpdssci;
	vgpci.pColorBlendState = &vpcbsci;
	vgpci.pDynamicState = &vpdsci;
	vgpci.layout = IN GraphicsPipelineLayout;
	vgpci.renderPass = IN RenderPass;
	vgpci.subpass = 0;			// subpass number
	vgpci.basePipelineHandle = (VkPipeline) VK_NULL_HANDLE;
	vgpci.basePipelineIndex = 0;

result = vkCreateGraphicsPipelines( LogicalDevice, VK_NULL_HANDLE, 1, IN &vgpci,
			PALLOCATOR,  OUT &GraphicsPipeline );
```


Telling the Command Buffer what Vertices to Draw

48

We will come to Command Buffers later, but for now, know that you will specify the vertex buffer that you want drawn.

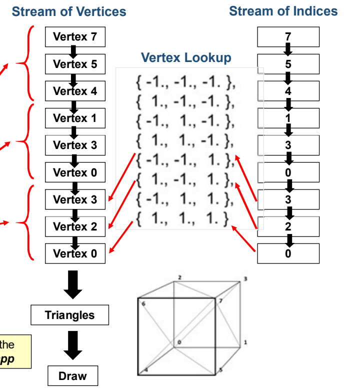
```
VkBuffer		buffers[1] = { MyVertexDataBuffer.buffer };
VkDeviceSize offsets[1]	= { 0 };

vkCmdBindVertexBuffers( CommandBuffers[nextImageIndex], 0, 1, buffers, offsets );

const uint32_t  firstInstance = 0;
const uint32_t  firstVertex = 0;
const uint32_t  instanceCount = 1;
const uint32_t  vertexCount = sizeof( VertexData ) / sizeof( VertexData[0] );

vkCmdDraw( CommandBuffers[nextImageIndex], vertexCount, instanceCount, firstVertex, firstInstance );
```

Always use the C/C++ construct ***sizeof***, rather than hardcoding a byte count!

Drawing with an Index Buffer


```
struct vertex JustVertexData[ ] =
{
	// vertex #0:
	{
		{ -1., -1., -1. },
		{  0.,  0., -1. },
		{  0.,  0.,  0. },
		{  1., 0. }
	},

	// vertex #1:
	{
		{  1., -1., -1. },
		{  0.,  0., -1. },
		{  1.,  0.,  0. },
		{  0., 0. }
	},
	. . .


int JustIndexData[ ] =
{
	0, 2, 3,
	0, 3, 1,
	4, 5, 7,
	4, 7, 6,
	1, 3, 7,
	1, 7, 5,
	0, 4, 6,
	0, 6, 2,
	2, 6, 7,
	2, 7, 3,
	0, 1, 5,
	0, 5, 4,
};
```

This data is contained in the file ***SampleVertexData.cpp***



Drawing with an Index Buffer

vkCmdBindVertexBuffers(commandBuffer, firstBinding, bindingCount, vertexDataBuffers, vertexOffsets);

vkCmdBindIndexBuffer(commandBuffer, indexDataBuffer, indexOffset, indexType);

```
typedef enum VkIndexType
{
	VK_INDEX_TYPE_UINT16 = 0,	// 0 –           65,535
	VK_INDEX_TYPE_UINT32 = 1,	// 0 – 4,294,967,295
} VkIndexType;
```

vkCmdDrawIndexed(commandBuffer, indexCount, instanceCount, firstIndex, vertexOffset, firstInstance);

Drawing with an Index Buffer

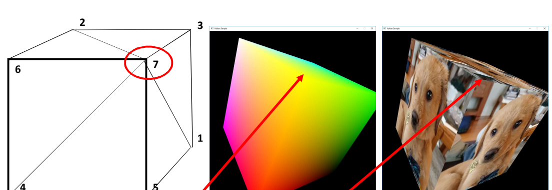

```
VkResult
Init05MyIndexDataBuffer(IN VkDeviceSize size, OUT MyBuffer * pMyBuffer)
{
	VkResult result = Init05DataBuffer(size, VK_BUFFER_USAGE_INDEX_BUFFER_BIT, pMyBuffer);
							// fills pMyBuffer
	return result;
}
```

```
Init05MyVertexDataBuffer(  sizeof(JustVertexData), IN &MyJustVertexDataBuffer );
Fill05DataBuffer( MyJustVertexDataBuffer,            (void *) JustVertexData  );

Init05MyIndexDataBuffer(  sizeof(JustIndexData),   IN &MyJustIndexDataBuffer );
Fill05DataBuffer( MyJustIndexDataBuffer,             (void *) JustIndexData );
```


Drawing with an Index Buffer


```
VkBuffer vBuffers[1] = { MyJustVertexDataBuffer.buffer };
VkBuffer iBuffer      = { MyJustIndexDataBuffer.buffer  };

vkCmdBindVertexBuffers( CommandBuffers[nextImageIndex], 0, 1, vBuffers, offsets );
				// 0, 1 = firstBinding, bindingCount
vkCmdBindIndexBuffer( CommandBuffers[nextImageIndex], iBuffer, 0, VK_INDEX_TYPE_UINT32 );

const uint32_t vertexCount = sizeof( JustVertexData ) / sizeof( JustVertexData[0] );
const uint32_t indexCount  = sizeof( JustIndexData  ) / sizeof( JustIndexData[0] );
const uint32_t instanceCount = 1;
const uint32_t firstVertex = 0;
const uint32_t firstIndex = 0;
const uint32_t firstInstance = 0;
const uint32_t vertexOffset  = 0;

vkCmdDrawIndexed( CommandBuffers[nextImageIndex], indexCount, instanceCount, firstIndex,
		vertexOffset, firstInstance );
```


Sometimes the Same Vertex Needs Multiple Values for the Attributes



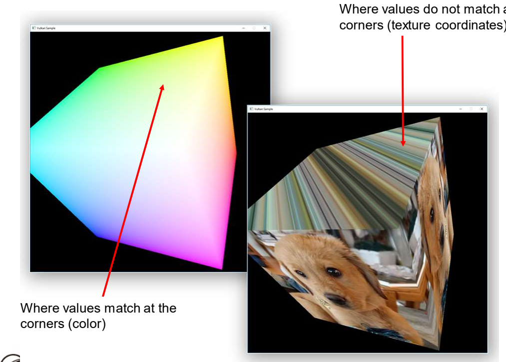
Sometimes a vertex that is common to multiple faces has the same attributes, no matter what face it is in. Sometimes it doesn't.

A color-interpolated cube like this actually has both. Vertex #7 above has the same color, regardless of what face it is in. However, Vertex #7 has 3 different normal vectors, depending on which face you are defining. Same with its texture coordinates.

Thus, when using indexed buffer drawing, you need to create a new vertex struct if *any* of {position, normal, color, texCoords} changes from what was previously-stored at those coordinates.

Sometimes the Same Vertex Needs Multiple Values for the Attributes

Where values do not match at the corners (texture coordinates)



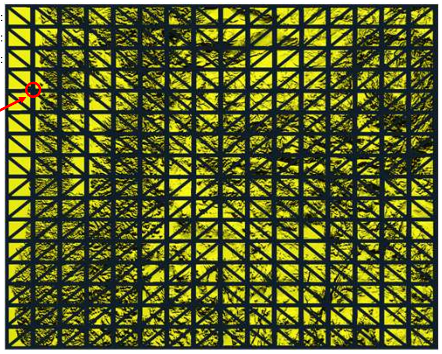
Where values match at the corners (color)

Terrain Surfaces are a Great Application of Indexed Drawing

55

Triangle Strip #0:
Triangle Strip #1:
Triangle Strip #2:
. . .

There is no question that it is OK for the (s,t) at these vertices to all be the same



SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

But, to Draw that Terrain Surface, You Need "Primitive Restart"

56

"Primitive Restart" is used with:
- Indexed drawing
- TRIANGLE_FAN and TRIANGLE_STRIP topologies

A special "index" is used to indicate that the triangle strip should start over. This is more efficient than explicitly ending the current triangle strip and explicitly starting a new one.

```
typedef enum VkIndexType
{
   VK_INDEX_TYPE_UINT16 = 0,        // 0 –           65,535
   VK_INDEX_TYPE_UINT32 = 1,        // 0 – 4,294,967,295
} VkIndexType;
```

If your VkIndexType is VK_INDEX_TYPE_UINT16, then the restart index is **0xffff**.
If your VkIndexType is VK_INDEX_TYPE_UINT32, then the restart index is **0xffffffff**.

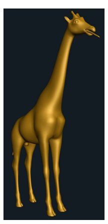
That is, a one in all available bits

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

The OBJ File Format – a triple-indexed way of Drawing

57



```
v 1.710541 1.283360 -0.040860
v 1.714593 1.273043 -0.041268
v 1.706114 1.279109 -0.040795
v 1.719083 1.277235 -0.041195
v 1.722786 1.267216 -0.041939
v 1.727196 1.271285 -0.041795
v 1.730680 1.261384 -0.042630
v 1.723121 1.280378 -0.037323
v 1.714513 1.286599 -0.037101

. . .

vn  0.1725  0.2557 -0.9512
vn -0.1979 -0.1899 -0.9616
vn -0.2050 -0.2127 -0.9554
vn  0.1664  0.3020 -0.9387
vn -0.2040 -0.1718 -0.9638
vn  0.1645  0.3203 -0.9329
vn -0.2055 -0.1698 -0.9638
vn  0.4419  0.6436 -0.6249
```

```
vt 0.816406 0.955536
vt 0.822754 0.959168
vt 0.815918 0.959442
vt 0.823242 0.955292
vt 0.829102 0.958862
vt 0.829590 0.955109
vt 0.835449 0.958618
vt 0.824219 0.951263

. . .

f 73/73/75 65/65/67 66/66/68
f 66/66/68 74/74/76 73/73/75
f 74/74/76 66/66/68 67/67/69
f 67/67/69 75/75/77 74/74/76
f 75/75/77 67/67/69 69/69/71
f 69/69/71 76/76/78 75/75/77
f 71/71/73 72/72/74 77/77/79
f 72/72/74 78/78/80 77/77/79
f 78/78/80 72/72/74 73/73/75
```

V / T / N

Note: The OBJ file format uses ***1-based*** indexing for faces!

We have a **vkuLoadObjFile( )** function to load an OBJ file into your Vulkan program!

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Drawing an OBJ Object

58

```
MyBuffer  MyObjBuffer;                 // global

      . . .

MyObjBuffer = VkOsuLoadObjFile( "filename.obj" );      // initializes and fills the buffer with
                               // triangles defined in GPU memory with an array of struct vertex
```

```
typedef struct MyBuffer
{
      VkDataBuffer        buffer;
      VkDeviceMemory      vdm;
      VkDeviceSize        size;      // in bytes
} MyBuffer;
```

```
VkPipelineInputAssemblyStateCreateInfo       vpiasci;
      vpiasci.sType = VK_STRUCTURE_TYPE_PIPELINE_INPUT_ASSEMBLY_STATE_CREATE_INFO;
      vpiasci.pNext = nullptr;
      vpiasci.flags = 0;
      vpiasci.topology = VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST;
```

```
struct vertex
{
      glm::vec3      position;
      glm::vec3      normal;
      glm::vec3      color;
      glm::vec2      texCoord;
};
```

```
VkBuffer        objBuffer[1] = { MyObjBuffer.buffer };
VkDeviceSize offsets[1]    = { 0 };
vkCmdBindVertexBuffers( CommandBuffers[nextImageIndex], 0, 1, objBuffer, offsets );

const uint32_t  firstInstance = 0;
const uint32_t  firstVertex = 0;
const uint32_t  instanceCount = 1;
const uint32_t  vertexCount = MyObjBuffer.size / sizeof( struct vertex );

vkCmdDraw( CommandBuffers[nextImageIndex], vertexCount, instanceCount, firstVertex, firstInstance );
```

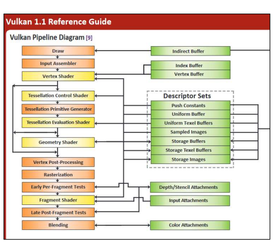
SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

59

Vulkan.

Data Buffers



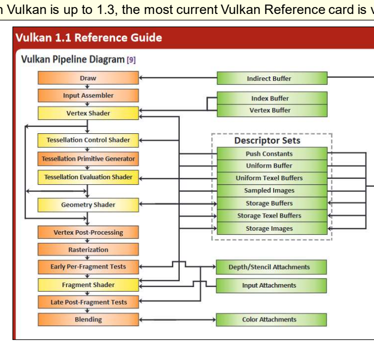
SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

From the Reference Card

60

Even though Vulkan is up to 1.3, the most current Vulkan Reference card is version 1.1



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

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Terminology Issues

A Vulkan **Data Buffer** is just a group of contiguous bytes in GPU memory. They have no inherent meaning. The data that is stored there is whatever you want it to be. (This is sometimes called a “Binary Large Object”, or “BLOB”.)

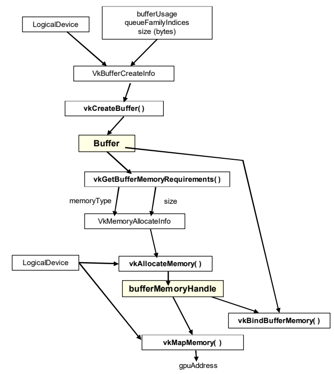
It is up to you to be sure that the writer and the reader of the Data Buffer are interpreting the bytes in the same way!

Vulkan calls these things “Buffers”. But, Vulkan calls other things “Buffers”, too, such as Texture Buffers and Command Buffers. So, I sometimes have taken to calling these things “Data Buffers” and have even gone so far as to extend some of Vulkan’s own terminology:

typedef VkBuffer VkDataBuffer;

This is probably a bad idea in the long run.

Creating and Filling Vulkan Data Buffers



Creating a Vulkan Data Buffer


```
VkBuffer  Buffer;              // or "VkDataBuffer   Buffer"

VkBufferCreateInfo   vbci;
      vbci.sType = VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO;
      vbci.pNext = nullptr;
      vbci.flags = 0;
      vbci.size = << buffer size in bytes >>
      vbci.usage = <<or'ed bits of: >>
            VK_USAGE_TRANSFER_SRC_BIT
            VK_USAGE_TRANSFER_DST_BIT
            VK_USAGE_UNIFORM_TEXEL_BUFFER_BIT
            VK_USAGE_STORAGE_TEXEL_BUFFER_BIT
            VK_USAGE_UNIFORM_BUFFER_BIT
            VK_USAGE_STORAGE_BUFFER_BIT
            VK_USAGE_INDEX_BUFFER_BIT
            VK_USAGE_VERTEX_BUFFER_BIT
            VK_USAGE_INDIRECT_BUFFER_BIT
      vbci.sharingMode = << one of: >>
            VK_SHARING_MODE_EXCLUSIVE
            VK_SHARING_MODE_CONCURRENT
      vbci.queueFamilyIndexCount = 0;
      vbci.pQueueFamilyIndices = (const iont32_t) nullptr;

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

“or” these bits together to specify how this buffer will be used

Allocating Memory for a Vulkan Data Buffer, Binding a Buffer to Memory, and Writing to the Buffer


```
VkMemoryRequirements          vmr;
result = vkGetBufferMemoryRequirements( LogicalDevice, Buffer, OUT &vmr );


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

. . .

VkDeviceMemory                vdm;
result = vkAllocateMemory( LogicalDevice, IN &vmai, PALLOCATOR,  OUT &vdm );

result = vkBindBufferMemory( LogicalDevice, Buffer, IN vdm, 0 );        // 0 is the offset

. . .

result = vkMapMemory( LogicalDevice, IN vdm, 0, VK_WHOLE_SIZE, 0, &ptr );

      << do the memory copy >>

result = vkUnmapMemory( LogicalDevice, IN vdm );
```


Finding the Right Type of Memory


```
int
FindMemoryThatIsHostVisible( )
{
      VkPhysicalDeviceMemoryProperties        vpdmp;
      vkGetPhysicalDeviceMemoryProperties( PhysicalDevice, OUT &vpdmp );
      for( unsigned int i = 0; i < vpdmp.memoryTypeCount; i++ )
      {
            VkMemoryType vmt = vpdmp.memoryTypes[ i ];
            if( ( vmt.propertyFlags & VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT ) != 0 )
            {
                  return i;
            }
      }
      return  -1;
}
```


Finding the Right Type of Memory


```
int
FindMemoryThatIsDeviceLocal( )
{
      VkPhysicalDeviceMemoryProperties        vpdmp;
      vkGetPhysicalDeviceMemoryProperties( PhysicalDevice, OUT &vpdmp );
      for( unsigned int i = 0; i < vpdmp.memoryTypeCount; i++ )
      {
            VkMemoryType vmt = vpdmp.memoryTypes[ i ];
            if( ( vmt.propertyFlags & VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT ) != 0 )
            {
                  return i;
            }
      }
      return  -1;
}
```

Finding the Right Type of Memory

67

```
VkPhysicalDeviceMemoryProperties                    vpdmp;
vkGetPhysicalDeviceMemoryProperties( PhysicalDevice, OUT &vpdmp );
```

```
6 Memory Types:
Memory  0:
Memory  1:  DeviceLocal
Memory  2:  HostVisible HostCoherent
Memory  3:  HostVisible HostCoherent HostCached
Memory  4:  DeviceLocal HostVisible HostCoherent
Memory  5:  DeviceLocal

4 Memory Heaps:
Heap 0:  size = 0xdbb00000 DeviceLocal
Heap 1:  size = 0xfd504000
Heap 2:  size = 0x0d600000 DeviceLocal
Heap 3:  size = 0x02000000 DeviceLocal
```

These are the numbers for the Nvidia A6000 cards

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Memory-Mapped Copying to GPU Memory, Example I

68

```
void *mappedDataAddr;

vkMapMemory( LogicalDevice, myBuffer.vdm, 0, VK_WHOLE_SIZE, 0, OUT (void *)&mappedDataAddr );

        memcpy( mappedDataAddr, &VertexData, sizeof(VertexData) );

vkUnmapMemory( LogicalDevice, myBuffer.vdm );
```

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Memory-Mapped Copying to GPU Memory, Example II

69

```
struct vertex *vp;

vkMapMemory( LogicalDevice, IN myBuffer.vdm, 0, VK_WHOLE_SIZE, 0, OUT (void *)&vp );

for( int i = 0; i < numTrianglesInObjFile; i++ )          // number of triangles
{
        for( int j = 0; j < 3; j++ )                      // 3 vertices per triangle
        {
                vp->position = glm::vec3( . . . );
                vp->normal = glm::vec3( . . . );
                vp->color = glm::vec3( . . . );
                vp->texCoord = glm::vec2( . . . );
                vp++;
        }
}

vkUnmapMemory( LogicalDevice, myBuffer.vdm );
```

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Sidebar: The Vulkan Memory Allocator (VMA)

70

The **Vulkan Memory Allocator** is a set of functions to simplify your view of allocating buffer memory. I am including its github link here and a little sample code in case you want to take a peek.

https://github.com/GPUOpen-LibrariesAndSDKs/VulkanMemoryAllocator

This repositoryalso includes a smattering of documentation.

See our class VMA noteset for more VMA details

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Sidebar: The Vulkan Memory Allocator (VMA)

71

```
#define VMA_IMPLEMENTATION
#include "vk_mem_alloc.h"
. . .
VkBufferCreateInfo                    vbci;
. . .
VmaAllocationCreateInfo               vaci;
        vaci.physicalDevice = PhysicalDevice;
        vaci.device = LogicalDevice;
        vaci.usage = VMA_MEMORY_USAGE_GPU_ONLY;

VmaAllocator                          var;
vmaCreateAllocator( IN &vaci, OUT &var );
. . .
. .
VkBuffer                              Buffer;
VmaAllocation                         van;
vmaCreateBuffer( IN var, IN &vbci, IN &vaci, OUT &Buffer. OUT &van, nullptr );
```

```
void *mappedDataAddr;
vmaMapMemory( var, van, OUT &mappedDataAddr );

        memcpy( mappedDataAddr, &VertexData, sizeof(VertexData) );

vmaUnmapMemory( var, van );
```

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Something I've Found Useful

72

I find it handy to encapsulate buffer information in a struct:

```
typedef struct MyBuffer
{
        VkDataBuffer         buffer;
        VkDeviceMemory       vdm;
        VkDeviceSize         size;      // in bytes
} MyBuffer;

. . .

// example:
MyBuffer                    MyObjectUniformBuffer;
```

It's the usual object-oriented benefit – you can pass around just one data-item and everyone can access whatever information they need.

It also makes it impossible to accidentally associate the wrong VkDeviceMemory and/or VkDeviceSize with the wrong data buffer.

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Initializing a Data Buffer

73

It's the usual object-oriented benefit – you can pass around just one data-item and everyone can access whatever information they need.

```
VkResult
Init05DataBuffer( VkDeviceSize size, VkBufferUsageFlags usage, OUT MyBuffer * pMyBuffer )
{
. . .
        vbci.size = pMyBuffer->size = size;
. . .
        result = vkCreateBuffer ( LogicalDevice, IN &vbci, PALLOCATOR,  OUT &pMyBuffer->buffer );
. . .
        pMyBuffer->vdm = vdm;
. . .
}
```


Here are C/C++ structs used by the Sample Code to hold some uniform variables

74

```
struct sceneBuf
{
        glm:: mat4      uProjection;
        glm:: mat4      uView;
        glm:: mat4      uSceneOrient;
        vec4            uLightPos;
        vec4            uLightColor;
        vec4            uLightKaKdKs;
        float           uTime;
} Scene;

struct objectBuf
{
        glm::mat4       uModel;
        glm:: mat4      uNormal;
        vec4            uColor;
        float           uShininess;
} Object;
```

The uNormal is set to:
glm::inverseTranspose(uView * uSceneOrient * uModel)

Here's the associated GLSL shader code to access those uniform variables:

```
layout( std140, set = 1, binding = 0 ) uniform sceneBuf
{
        mat4    uProjection;
        mat4    uView;
        mat4    uSceneOrient;
        vec4    uLightPos;
        vec4    uLightColor;
        vec4    uLightKaKdKs;
        float   uTime;
} Scene;

layout( std140, set = 2, binding = 0 ) uniform objectBuf
{
        mat4    uModel;
        mat4    uNormal;
        vec4    uColor;
        float   uShininess;
} Object;
```

In the vertex shader, each object vertex gets transformed by:
uProjection* uView * uSceneOrient * uModel

In the vertex shader, each surface normal vector gets transformed by the **uNormal**

Filling those Uniform Variables

75

```
const float EYEDIST =   3.0f;
const double FOV    =   glm::radians(60.);      // field-of-view angle in radians

glm::vec3  eye(0.,0.,EYEDIST);
glm::vec3  look(0.,0.,0.);
glm::vec3  up(0.,1.,0.);

Scene.uProjection          = glm::perspective( FOV, (double)Width/(double)Height, 0.1, 1000. );
Scene.uProjection[1][1]  *= -1.;               // account for Vulkan's LH screen coordinate system
Scene.uView                = glm::lookAt( eye, look, up );
Scene.uSceneOrient         = glm::mat4( 1. );

Object.uModelOrient = glm::mat4( 1. );           // identity
Object.uNormal      =  glm::inverseTranspose( Scene.uView * Scene.uSceneOrient * Object.uModel )
```

This code assumes that this line:

#define   GLM_FORCE_RADIANS

is listed before GLM is #included!

The Parade of Buffer Data

76

MyBuffer   MyObjectUniformBuffer;

The MyBuffer does not hold any actual data itself. It just information about what is in the data buffer

```
VkResult
Init05DataBuffer( VkDeviceSize size, VkBufferUsageFlags usage, OUT MyBuffer * pMyBuffer )
{
. . .
        vbci.size = pMyBuffer->size = size;
. . .
        result = vkCreateBuffer ( LogicalDevice, IN &vbci, PALLOCATOR,  OUT &pMyBuffer->buffer );
. . .
        pMyBuffer->vdm = vdm;
. . .
}
```

This C struct is holding the original data, written by the application.

Memory-mapped copy operation

The Data Buffer in GPU memory is holding the copied data. It is readable by the shaders

struct objectBuf   Object;

```
Object.uModelOrient = glm::mat4( 1. );           // identity
Object.uNormal      =  glm::inverseTranspose( Scene.uView * Scene.uSceneOrient * Object.uModel )
```

uniform objectBuf   Object;

```
layout( std140, set = 2, binding = 0 ) uniform objectBuf
{
        mat4    uModel;
        mat4    uNormal;
        vec4    uColor;
        float   uShininess;
} Object;
```


Filling the Data Buffer

77

```
typedef struct MyBuffer
{
        VkDataBuffer        buffer;
        VkDeviceMemory      vdm;
        VkDeviceSize        size;       // in bytes
} MyBuffer;

. . .

// example:
MyBuffer                MyObjectUniformBuffer;
```

```
Init05UniformBuffer( sizeof(Object),          OUT &MyObjectUniformBuffer );

Fill05DataBuffer( MyObjectUniformBuffer,      IN (void *) &Object );
```

```
struct objectBuf
{
        glm::mat4       uModel;
        glm:: mat4      uNormal;
        vec4            uColor;
        float           uShininess;
} Object;
```


Creating and Filling the Data Buffer – the Details

78

```
VkResult
Init05DataBuffer( VkDeviceSize size, VkBufferUsageFlags usage, OUT MyBuffer * pMyBuffer )
{
        VkResult result = VK_SUCCESS;
        VkBufferCreateInfo  vbci;
                vbci.sType = VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO;
                vbci.pNext = nullptr;
                vbci.flags = 0;
                vbci.size = pMyBuffer->size = size;
                vbci.usage = usage;
                vbci.sharingMode = VK_SHARING_MODE_EXCLUSIVE;
                vbci.queueFamilyIndexCount = 0;
                vbci.pQueueFamilyIndices = (const uint32_t *)nullptr;
        result = vkCreateBuffer ( LogicalDevice, IN &vbci, PALLOCATOR,  OUT &pMyBuffer->buffer );

        VkMemoryRequirements                vmr;
        vkGetBufferMemoryRequirements( LogicalDevice, IN pMyBuffer->buffer, OUT &vmr );        // fills vmr

        VkMemoryAllocateInfo                vmai;
                vmai.sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
                vmai.pNext = nullptr;
                vmai.allocationSize = vmr.size;
                vmai.memoryTypeIndex = FindMemoryThatIsHostVisible( );

        VkDeviceMemory                      vdm;
        result = vkAllocateMemory( LogicalDevice, IN &vmai, PALLOCATOR, OUT &vdm );
        pMyBuffer->vdm = vdm;

        result = vkBindBufferMemory( LogicalDevice, pMyBuffer->buffer, IN vdm, OFFSET_ZERO );
        return result;
}
```

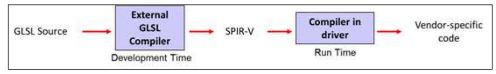
Creating and Filling the Data Buffer – the Details


```
VkResult
Fill05DataBuffer( IN MyBuffer myBuffer, IN void * data )
{
      // the size of the data had better match the size that was used to Init the buffer!

      void * pGpuMemory;
      vkMapMemory( LogicalDevice, IN myBuffer.vdm, 0, VK_WHOLE_SIZE, 0, OUT &pGpuMemory );
                                                        // 0 and 0 are offset and flags
      memcpy( pGpuMemory, data, (size_t)myBuffer.size );

     vkUnmapMemory( LogicalDevice, IN myBuffer.vdm );
      return VK_SUCCESS;
}
```

Remember – to Vulkan and GPU memory, these are just *bits*. It is up to *you* to handle their meaning correctly.

Vulkan.

Shaders and SPIR-V

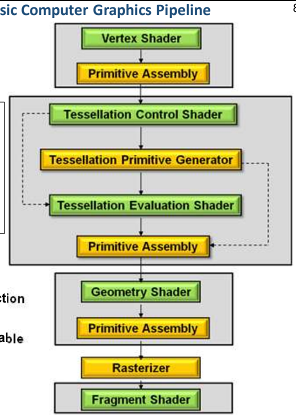


The Shaders' View of the Basic Computer Graphics Pipeline

- You need to have a vertex and fragment shader as a minimum.
- A missing stage is OK. The output from one stage becomes the input of the next stage that is there.
- The last stage before the fragment shader feeds its output variables into the **rasterizer**. The interpolated values then go to the fragment shaders.

= Fixed Function

= Programmable



Vulkan Shader Stages

Shader stages

```
typedef enum VkPipelineStageFlagBits {
   VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT = 0x00000001,
   VK_PIPELINE_STAGE_DRAW_INDIRECT_BIT = 0x00000002,
   VK_PIPELINE_STAGE_VERTEX_INPUT_BIT = 0x00000004,
   VK_PIPELINE_STAGE_VERTEX_SHADER_BIT = 0x00000008,
   VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT = 0x00000010,
   VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT = 0x00000020,
   VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT = 0x00000040,
   VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT = 0x00000080,
   VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT = 0x00000100,
   VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT = 0x00000200,
   VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT = 0x00000400,
   VK_PIPELINE_STAGE_COMPUTE_SHADER_BIT = 0x00000800,
   VK_PIPELINE_STAGE_TRANSFER_BIT = 0x00001000,
   VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT = 0x00002000,
   VK_PIPELINE_STAGE_HOST_BIT = 0x00004000,
   VK_PIPELINE_STAGE_ALL_GRAPHICS_BIT = 0x00008000,
   VK_PIPELINE_STAGE_ALL_COMMANDS_BIT = 0x00010000,
} VkPipelineStageFlagBits;
```


How Vulkan GLSL Differs from OpenGL GLSL

Detecting that a GLSL Shader is being used with Vulkan/SPIR-V:

- In the compiler, there is an automatic
  #define VULKAN 130
  or whatever the current version number is.
  Typically you use this like:
  #ifdef VULKAN
  . . .
  #endif

Vulkan Vertex and Instance indices:

gl_VertexIndex
gl_InstanceIndex

OpenGL uses:

gl_VertexID
gl_InstanceID

- Both are 0-based

gl_FragColor:

- In OpenGL, gl_FragColor broadcasts to all color attachments
- In Vulkan, it just broadcasts to color attachment location #0
- Best idea: don't use it at all – explicitly declare out variables to have specific location numbers:
  layout (location = 0) out vec4 fFragColor;

How Vulkan GLSL Differs from OpenGL GLSL

Shader combinations of separate texture data and samplers as an option:

uniform sampler s;
uniform texture2D t;
vec4 rgba = texture(sampler2D(t, s), vST);

Note: our sample code doesn't use this.

Descriptor Sets:

layout(set=0, binding=0) . . . ;

Push Constants:

layout(push_constant) . . . ;

Specialization Constants:

layout(constant_id = 3) const int N = 5;

- Only for scalars, but a vector's components can be constructed from specialization constants

For example, Specialization Constants can be used with Compute Shaders:

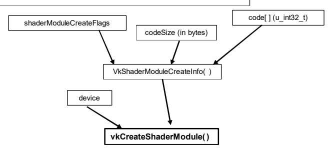
layout(local_size_x_id = 8, local_size_y_id = 16);

- This sets gl_WorkGroupSize.x and gl_WorkGroupSize.y
- gl_WorkGroupSize.z is set as a constant

Vulkan: Shaders' use of Layouts for Uniform Variables

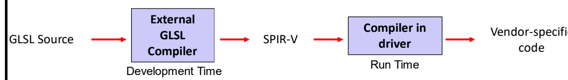

```
layout( std140, set = 0, binding = 0 ) uniform sceneMatBuf
{
        mat4 uProjectionMatrix;
        mat4 uViewMatrix;
        mat4 uSceneMatrix;
} SceneMatrices;

layout( std140, set = 1, binding = 0 ) uniform objectMatBuf
{
        mat4 uModelMatrix;
        mat4 uNormalMatrix;
} ObjectMatrices;

layout( set = 2, binding = 0 ) uniform sampler2D uTexUnit;
```

All non-sampler uniform variables *must* be in block buffers



Vulkan Shader Compiling

- You half-precompile your shaders with an external compiler
- Your shaders get turned into an intermediate form known as SPIR-V, which stands for **Standard Portable Intermediate Representation.**
- SPIR-V gets turned into fully-compiled code at runtime, when the pipeline structure is finally created
- The SPIR-V spec has been public for a few years –new shader languages are surely being developed
- OpenGL and OpenCL have now adopted SPIR-V as well



Advantages:

1. Software vendors don't need to ship their shader source
2. Syntax errors appear during the SPIR-V step, not during runtime
3. Software can launch faster because half of the compilation has already taken place
4. This guarantees a common front-end syntax
5. This allows for other language front-ends

SPIR-V:
Standard Portable Intermediate Representation for Vulkan

glslangValidator shaderFile -V [-H] [-I<dir>] [-S <stage>] -o shaderBinaryFile.spv

Shaderfile extensions:

.vert	Vertex
.tesc	Tessellation Control
.tese	Tessellation Evaluation
.geom	Geometry
.frag	Fragment
.comp	Compute

(Can be overridden by the –S option)

-V	**Compile for Vulkan**
-G	Compile for OpenGL
-I	Directory(ies) to look in for #includes
-S	Specify stage rather than get it from shaderfile extension
-c	Print out the maximum sizes of various properties

Windows: glslangValidator.exe

Linux: glslangValidator

You Can Run the SPIR-V Compiler on Windows from a Bash Shell

You can run the glslangValidator program from the Windows Command Prompt, but I have found it easier to run the SPIR-V compiler from Windows-Bash.

To install the bash shell on your own Windows machine, go to this URL:

https://www.msn.com/en-us/news/technology/how-to-install-and-run-bash-on-windows-11/ar-AA10EoPk

Or, follow these instructions:

1. Head to the **Start menu** search bar, type in 'terminal,' and launch the Windows Terminal as administrator. (On some systems, this is called the **Command Prompt**.)
2. Type in the following command in the administrator: **wsl --install**
3. Restart your PC once the installation is complete.

As soon as your PC boots up, the installation will begin again. Your PC will start downloading and installing the Ubuntu software. You'll soon get asked to set up a username and password. This can be the same as your system's username and password, but doesn't have to be. The installation will automatically start off from where you left it.

Reading a SPIR-V File into a Vulkan Shader Module

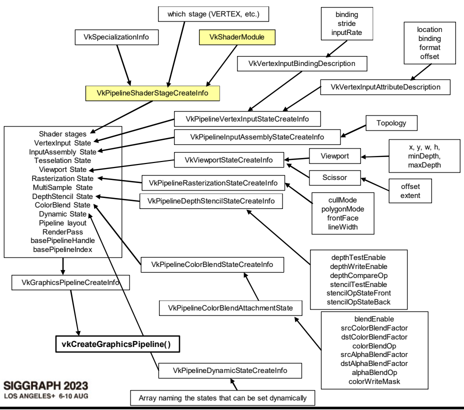

```
#ifndef _WIN32
        typedef int errno_t;
        int     fopen_s( FILE**, const char *, const char * );
#endif

#define SPIRV_MAGIC         0x07230203
. . .

VkResult
Init12SpirvShader( std::string filename, VkShaderModule * pShaderModule )
{
      FILE *fp;
#ifdef WIN32
         errno_t err = fopen_s( &fp, filename.c_str( ), "rb" );
         if( err != 0 )
#else
         fp = fopen( filename.c_str( ), "rb" );
         if( fp == NULL )
#endif
      {
            fprintf( FpDebug, "Cannot open shader file '%s'\n", filename.c_str( ) );
            return VK_SHOULD_EXIT;
      }
      uint32_t magic;
      fread( &magic, 4, 1, fp );
      if( magic != SPIRV_MAGIC )
      {
            fprintf( FpDebug, "Magic number for spir-v file '%s is 0x%08x -- should be 0x%08x\n", filename.c_str( ), magic, SPIRV_MAGIC );
            return VK_SHOULD_EXIT;
      }

      fseek( fp, 0L, SEEK_END );
      int size = ftell( fp );
      rewind( fp );
      unsigned char *code = new unsigned char [size];
      fread( code, size, 1, fp );
      fclose( fp );

      . . .
```


Reading a SPIR-V File into a Shader Module


```
. . .
VkShaderModule            ShaderModuleVertex;
. . .

      VkShaderModuleCreateInfo          vsmci;
            vsmci.sType = VK_STRUCTURE_TYPE_SHADER_MODULE_CREATE_INFO;
            vsmci.pNext = nullptr;
            vsmci.flags = 0;
            vsmci.codeSize = size;
            vsmci.pCode = (uint32_t *)code;

      VkResult result = vkCreateShaderModule( LogicalDevice, &vsmci, PALLOCATOR, OUT & ShaderModuleVertex );
      fprintf( FpDebug, "Shader Module '%s' successfully loaded\n", filename.c_str() );
      delete [ ] code;
      return result;
}
```

Vulkan: Creating a Graphics Pipeline Data Structure

91



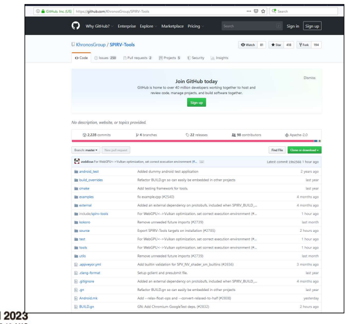
SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

SPIR-V: More Information

92

SPIR-V Tools:
http://github.com/KhronosGroup/SPIRV-Tools



SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

A Google-Wrapped Version of glslangValidator

93

The shaderc project from Google (https://github.com/google/shaderc) provides a glslangValidator wrapper program called ***glslc*** that has a much improved command-line interface. You use, basically, the same way:

glslc.exe –target-env=vulkan sample-vert.vert -o sample-vert.spv

There are several really nice features. The two I really like are:

1. You can #include files into your shader source
2. You can "#define" definitions on the command line like this:
   glslc.exe --target-env=vulkan -DNUMPONTS=4 sample-vert.vert -o sample-vert.spv

glslc is included in your Sample .zip file

This causes a:
#define NUMPOINTS 4
to magically be inserted into the top of your source code.

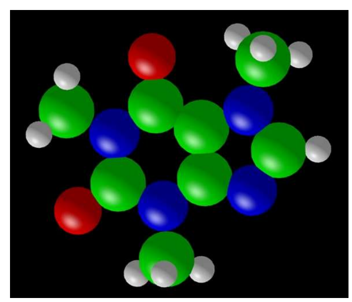
SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

94

Vulkan.

Instancing



SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Instancing – What and why?

95

- Instancing is the ability to draw the same object multiple times
- It uses all the same vertices and the same graphics pipeline data structure each time
- It avoids the overhead of the program asking to have the object drawn again, letting the GPU/driver handle all of that

vkCmdDraw(CommandBuffers[nextImageIndex], vertexCount, **instanceCount**, firstVertex, **firstInstance**);

BTW, when not using instancing, be sure the **instanceCount** is **1**, not **0** !

But, this will only get us multiple instances of identical objects drawn on top of each other. How can we make each instance look differently?

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Making each Instance look differently -- Approach #1

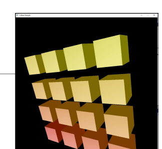
96

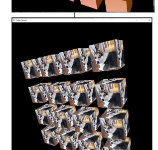
Use the built-in vertex shader variable **gl_InstanceIndex** to define a unique display property, such as position or color.

gl_InstanceIndex starts at 0

In the vertex shader:

```
layout( std140, set = 0, binding = 0 ) uniform sporadicBuf
{
        int        uMode;
        int        uUseLighting;
        int        uNumInstances;
} Sporadic;
. . .
void
main( )
{
    . . .

        float DELTA       = 3.0;
        float s = sqrt( float( Sporadic.uNumInstances ) );
        float c = ceil( float(s) );
        int cols = int( c );
        int fullRows     = gl_InstanceIndex / cols;
        int remainder = gl_InstanceIndex % cols;

        float xdelta = DELTA * float( remainder );
        float ydelta = DELTA * float( fullRows );
        vColor = vec3( 1., float( (1.+ gl_InstanceIndex) ) / float( Sporadic.uNumInstances ), 0. );

        vec4 vertex = vec4( aVertex.xyz + vec3( xdelta, ydelta, 0. ), 1. );
        gl_Position = PVM * vertex;
}
```





SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Making each Instance look differently -- Approach #2

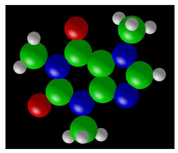
97

Put the unique characteristics in a uniform buffer array and reference them

Still uses **gl_InstanceIndex**

In the vertex shader:

```
layout( std140, set = 4, binding = 0 ) uniform colorBuf
{
        vec3  uColors[1024];
} Colors;

out vec3 vColor;

        . . .

int index = gl_InstanceIndex % 1024;      // gives 0 - 1023
vColor = Colors.uColors[ index ];

. . .
 vec4 vertex = vec4( aVertex.xyz  + vec3( xdelta, ydelta, 0. ), 1. );
gl_Position = PVM * vertex;
```



An Example of Using Approach #2

98

The 24 Atoms in a Caffeine Molecule

Symbol	Atomic Number	X	Y	Z	Radius	Color
C	6	-1.799	0.022	0.602	0.77	Green
N	7	-1.586	-0.945	-0.363	0.70	Blue
C	6	-2.731	-1.654	-0.954	0.77	Green
C	6	-0.301	-1.248	-0.776	0.77	Green
C	6	0.789	-0.574	-0.214	0.77	Green
C	6	0.563	0.404	0.763	0.77	Green
N	7	-0.728	0.694	1.163	0.70	Blue
C	6	-0.964	1.720	2.189	0.77	Green
N	7	1.752	0.896	1.139	0.70	Blue
C	6	2.729	0.287	0.454	0.77	Green
N	7	2.158	-0.638	-0.400	0.70	Blue
C	6	2.871	-1.524	-1.331	0.77	Green
O	8	-0.101	-2.186	-1.712	0.66	Red
O	8	-3.048	0.309	0.996	0.66	Red
H	1	3.786	0.485	0.553	0.37	White
H	1	-2.947	-2.544	-0.368	0.37	White
H	1	-2.491	-1.941	-1.976	0.37	White
H	1	-3.601	-1.000	-0.955	0.37	White
H	1	-1.088	2.689	1.711	0.37	White
H	1	-0.114	1.756	2.868	0.37	White
H	1	-1.864	1.473	2.748	0.37	White
H	1	2.981	-1.026	-2.293	0.37	White
H	1	2.305	-2.444	-1.462	0.37	White
H	1	3.855	-1.757	-0.929	0.37	White

Referencing that Table as a Uniform Buffer

99

The 24 Atoms in a Caffeine Molecule

Symbol	Atomic Number	X	Y	Z	Radius	Color
C	6	-1.799	0.022	0.602	0.77	Green
N	7	-1.586	-0.945	-0.363	0.70	Blue
C	6	-2.731	-1.654	-0.954	0.77	Green
C	6	-0.301	-1.248	-0.776	0.77	Green
C	6	0.789	-0.574	-0.214	0.77	Green
C	6	0.563	0.404	0.763	0.77	Green
N	7	-0.728	0.694	1.163	0.70	Blue
C	6	-0.964	1.720	2.189	0.77	Green
N	7	1.752	0.896	1.139	0.70	Blue
C	6	2.729	0.287	0.454	0.77	Green
N	7	2.158	-0.638	-0.400	0.70	Blue
C	6	2.871	-1.524	-1.331	0.77	Green
O	8	-0.101	-2.186	-1.712	0.66	Red
O	8	-3.048	0.309	0.996	0.66	Red
H	1	3.786	0.485	0.553	0.37	White
H	1	-2.947	-2.544	-0.368	0.37	White
H	1	-2.491	-1.941	-1.976	0.37	White
H	1	-3.601	-1.000	-0.955	0.37	White
H	1	-1.088	2.689	1.711	0.37	White
H	1	-0.114	1.756	2.868	0.37	White
H	1	-1.864	1.473	2.748	0.37	White
H	1	2.981	-1.026	-2.293	0.37	White
H	1	2.305	-2.444	-1.462	0.37	White
H	1	3.855	-1.757	-0.929	0.37	White

```
struct atom
{
        vec3 position;
        int atomicNumber;
};

layout( std140, set = 4, binding = 0 ) uniform moleculeBuf
{
        atom            atoms[24];
};
```


The Transformation Setup

100

```
void
main( )
{
        mat4  P = Scene.uProjection;
        mat4  V = Scene.uView;
        mat4  SO = Scene.uSceneOrient;
        mat4  M  = Object.uModel;
        mat4 VM = V * SO * M;
        mat4 PVM = P * VM;

        vColor      = aColor;
        vTexCoord  = aTexCoord;

        vN = normalize( mat3( Object.uNormal ) * aNormal );     // surface normal vector

        vec4 ECposition = VM * vec4( aVertex,  1. );
        vec4 lightPos = vec4( Scene.uLightPos.xyz, 1. );       // light source in fixed location because not transformed
        vL = normalize( lightPos.xyz  -  ECposition.xyz );      // vector from the point to the light

        vec4 eyePos = vec4( 0., 0., 0., 1. );                  // eye position after applying the viewing matrix
        vE = normalize( eyePos.xyz  -  ECposition.xyz );       // vector from the point to the eye
```


Using the gl_InstanceIndex Variable

101

The 24 Atoms in a Caffeine Molecule

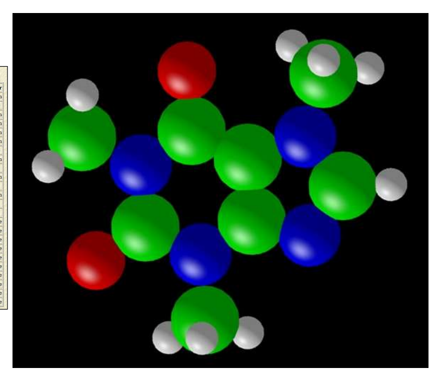
Symbol	Atomic Number	X	Y	Z	Radius	Color
C	6	-1.799	0.022	0.602	0.77	Green
N	7	-1.586	-0.945	-0.363	0.70	Blue
C	6	-2.731	-1.654	-0.954	0.77	Green
C	6	-0.301	-1.248	-0.776	0.77	Green
C	6	0.789	-0.574	-0.214	0.77	Green
C	6	0.563	0.404	0.763	0.77	Green
N	7	-0.728	0.694	1.163	0.70	Blue
C	6	-0.964	1.720	2.189	0.77	Green
N	7	1.752	0.896	1.139	0.70	Blue
C	6	2.729	0.287	0.454	0.77	Green
N	7	2.158	-0.638	-0.400	0.70	Blue
C	6	2.871	-1.524	-1.331	0.77	Green
O	8	-0.101	-2.186	-1.712	0.66	Red
O	8	-3.048	0.309	0.996	0.66	Red
H	1	3.786	0.485	0.553	0.37	White
H	1	-2.947	-2.544	-0.368	0.37	White
H	1	-2.491	-1.941	-1.976	0.37	White
H	1	-3.601	-1.000	-0.955	0.37	White
H	1	-1.088	2.689	1.711	0.37	White
H	1	-0.114	1.756	2.868	0.37	White
H	1	-1.864	1.473	2.748	0.37	White
H	1	2.981	-1.026	-2.293	0.37	White
H	1	2.305	-2.444	-1.462	0.37	White
H	1	3.855	-1.757	-0.929	0.37	White

```
int atomicNumber = atoms[gl_InstanceIndex].atomicNumber;
vec3 position       = atoms[gl_InstanceIndex].position;
float radius;

if( atomicNumber == 1 )
{
        radius = 0.37;
        vColor = vec3( 1., 1., 1. );
} else if( atomicNumber == 6 )
{
        radius = 0.77;
        vColor = vec3( 0., 1., 0. );
}
else if( atomicNumber == 7 )
{
        radius = 0.70;
        vColor = vec3( 0., 0., 1. );
}
else if( atomicNumber == 8 )
{
        radius = 0.66;
        vColor = vec3( 1., 0., 0. );
}
else
{
        radius = 0.75;
        vColor = vec3( 1., 0., 1. );    // big magenta ball to tell us something is wrong
}

vec3 bVertex = aVertex;
bVertex.xyz *= radius;
bVertex.xyz += position;
gl_Position  = PVM * vec4( bVertex, 1. );
}
```


102

The 24 Atoms in a Caffeine Molecule

Symbol	Atomic Number	X	Y	Z	Radius	Color
C	6	-1.799	0.022	0.602	0.77	Green
N	7	-1.586	-0.945	-0.363	0.70	Blue
C	6	-2.731	-1.654	-0.954	0.77	Green
C	6	-0.301	-1.248	-0.776	0.77	Green
C	6	0.789	-0.574	-0.214	0.77	Green
C	6	0.563	0.404	0.763	0.77	Green
N	7	-0.728	0.694	1.163	0.70	Blue
C	6	-0.964	1.720	2.189	0.77	Green
N	7	1.752	0.896	1.139	0.70	Blue
C	6	2.729	0.287	0.454	0.77	Green
N	7	2.158	-0.638	-0.400	0.70	Blue
C	6	2.871	-1.524	-1.331	0.77	Green
O	8	-0.101	-2.186	-1.712	0.66	Red
O	8	-3.048	0.309	0.996	0.66	Red
H	1	3.786	0.485	0.553	0.37	White
H	1	-2.947	-2.544	-0.368	0.37	White
H	1	-2.491	-1.941	-1.976	0.37	White
H	1	-3.601	-1.000	-0.955	0.37	White
H	1	-1.088	2.689	1.711	0.37	White
H	1	-0.114	1.756	2.868	0.37	White
H	1	-1.864	1.473	2.748	0.37	White
H	1	2.981	-1.026	-2.293	0.37	White
H	1	2.305	-2.444	-1.462	0.37	White
H	1	3.855	-1.757	-0.929	0.37	White



Vulkan

GLFW

```
while( glfwWindowShouldClose( MainWindow ) == 0 )
{
        glfwPollEvents( );
        Time = glfwGetTime( );          // elapsed time, in double-precision seconds
        UpdateScene( );
        RenderScene( );
}

vkQueueWaitIdle( Queue );
vkDeviceWaitIdle( LogicalDevice );
DestroyAllVulkan( );
glfwDestroyWindow( MainWindow );
glfwTerminate( );
```


Setting Up GLFW

```
#define GLFW_INCLUDE_VULKAN
#include "glfw3.h"
        . . .

uint32_t                    Width, Height;
VkSurfaceKHR            Surface;
        . . .

void
InitGLFW( )
{
        glfwInit( );
        if( ! glfwVulkanSupported( ) )
        {
                fprintf( stderr, "Vulkan is not supported on this system!\n" );
                exit( 1 );
        }
        glfwWindowHint( GLFW_CLIENT_API, GLFW_NO_API );
        glfwWindowHint( GLFW_RESIZABLE, GLFW_FALSE );
        MainWindow    = glfwCreateWindow( Width, Height, "Vulkan Sample", NULL, NULL );
        VkResult result = glfwCreateWindowSurface( Instance, MainWindow, NULL, OUT &Surface );

        glfwSetErrorCallback( GLFWErrorCallback );
        glfwSetKeyCallback( MainWindow,                GLFWKeyboard );
        glfwSetCursorPosCallback( MainWindow,       GLFWMouseMotion );
        glfwSetMouseButtonCallback( MainWindow,  GLFWMouseButton );
}
```


You Can Also Query What Vulkan Extensions GLFW Requires

```
uint32_t count;
const char ** extensions = glfwGetRequiredInstanceExtensions (&count);

fprintf( FpDebug, "\nFound %d GLFW Required Instance Extensions:\n", count );

for( uint32_t  i = 0; i < count; i++ )
{
        fprintf( FpDebug, "\t%s\n", extensions[ i ] );
}
```

```
Found 2 GLFW Required Instance Extensions:
        VK_KHR_surface
        VK_KHR_win32_surface
```


GLFW Keyboard Callback

```
void
GLFWKeyboard( GLFWwindow * window, int key, int scancode, int action, int mods )
{
        if( action == GLFW_PRESS )
        {
                switch( key )
                {
                        //case GLFW_KEY_M:
                        case 'm':
                        case 'M':
                                Mode++;
                                if( Mode >= 2 )
                                        Mode = 0;
                                break;

                        default:
                                fprintf( FpDebug, "Unknow key hit: 0x%04x = '%c'\n", key, key );
                                fflush(FpDebug);
                }
        }
}
```


GLFW Mouse Button Callback

```
void
GLFWMouseButton( GLFWwindow *window, int button, int action, int mods )
{
        int b = 0;              // LEFT, MIDDLE, or RIGHT

        // get the proper button bit mask:
        switch( button )
        {
                case GLFW_MOUSE_BUTTON_LEFT:
                        b = LEFT;               break;

                case GLFW_MOUSE_BUTTON_MIDDLE:
                        b = MIDDLE;             break;

                case GLFW_MOUSE_BUTTON_RIGHT:
                        b = RIGHT;              break;

                default:
                        b = 0;
                        fprintf( FpDebug, "Unknown mouse button: %d\n", button );
        }

        // button down sets the bit, up clears the bit:
        if( action == GLFW_PRESS )
        {
                double xpos, ypos;
                glfwGetCursorPos( window, &xpos, &ypos);
                Xmouse = (int)xpos;
                Ymouse = (int)ypos;
                ActiveButton |= b;              // set the proper bit
        }
        else
        {
                ActiveButton &= ~b;             // clear the proper bit
        }
}
```


GLFW Mouse Motion Callback

```
void
GLFWMouseMotion( GLFWwindow *window, double xpos, double ypos )
{
        int dx = (int)xpos - Xmouse;            // change in mouse coords
        int dy = (int)ypos - Ymouse;

        if( ( ActiveButton & LEFT ) != 0 )
        {
                Xrot += ( ANGFACT*dy );
                Yrot += ( ANGFACT*dx );
        }

        if( ( ActiveButton & MIDDLE ) != 0 )
        {
                Scale += SCLFACT * (float) ( dx - dy );

                // keep object from turning inside-out or disappearing:

                if( Scale < MINSCALE )
                        Scale = MINSCALE;
        }

        Xmouse = (int)xpos;                     // new current position
        Ymouse = (int)ypos;
}
```

Looping and Closing GLFW


```
while( glfwWindowShouldClose( MainWindow ) == 0 )
{
        glfwPollEvents( );
        Time = glfwGetTime( );          // elapsed time, in double-precision seconds
        UpdateScene( );
        RenderScene( );
}

vkQueueWaitIdle( Queue );
vkDeviceWaitIdle( LogicalDevice );
DestroyAllVulkan( );
glfwDestroyWindow( MainWindow );
glfwTerminate( );
```

Does not block – processes any waiting events, then returns

Looping and Closing GLFW

If you would like to *block* waiting for events, use:

glfwWaitEvents( );

You can have the blocking wake up after a timeout period with:

glfwWaitEventsTimeout(double secs);

You can wake up one of these blocks from another thread with:

glfwPostEmptyEvent( );

Vulkan.

GLM

```
if( UseMouse )
{
        if( Scale < MINSCALE )
           Scale = MINSCALE;
        Matrices.uModelMatrix = glm::mat4( 1. );          // identity
        Matrices.uModelMatrix = glm::rotate( Matrices.uModelMatrix, Yrot, glm::vec3( 0.,1.,0.) );
        Matrices.uModelMatrix = glm::rotate( Matrices.uModelMatrix, Xrot, glm::vec3( 1.,0.,0.) );
        Matrices.uModelMatrix = glm::scale(  Matrices.uModelMatrix, glm::vec3(Scale,Scale,Scale) );
                              // done this way, the Scale is applied first, then the Xrot, then the Yrot
}
```


What is GLM?

GLM is a set of C++ classes and functions to fill in the programming gaps in writing the basic vector and matrix mathematics for OpenGL applications. However, even though it was written for OpenGL, it works fine with Vulkan.

Even though GLM looks like a library, it actually isn't – it is all specified in ***.hpp** header files so that it gets compiled in with your source code.

You can find it at:
http://glm.g-truc.net/0.9.8.5/

You invoke GLM like this:

```
#define   GLM_FORCE_RADIANS

#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/matrix_inverse.hpp>
```

OpenGL treats all angles as given in *degrees*. This line forces GLM to treat all angles as given in *radians*. I recommend this so that *all* angles you create in *all* programming will be in radians.

If GLM is not installed in a system place, put it somewhere you can get access to. Later on, these notes will show you how to use it from there.

Why are we even talking about this?

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. So, where you might have said in OpenGL:

```
glMatrixMode( GL_MODELVIEW );
glLoadIdentity( );
gluLookAt( 0., 0., 3.,     0., 0., 0.,     0., 1., 0. );
glRotatef( (GLfloat)Yrot, 0., 1., 0. );
glRotatef( (GLfloat)Xrot, 1., 0., 0. );
glScalef( (GLfloat)Scale, (GLfloat)Scale, (GLfloat)Scale );
```

you would now say:

```
glm::mat4 modelview = glm::mat4( 1. );      // identity
glm::vec3 eye(0.,0.,3.);
glm::vec3 look(0.,0.,0.);
glm::vec3   up(0.,1.,0.);
modelview = glm::lookAt( eye, look, up );                                  // {x',y',z'} = [v]*{x,y,z}
modelview = glm::rotate( modelview, D2R*Yrot, glm::vec3(0.,1.,0.) );   // {x',y',z'} = [v]*[yr]*{x,y,z}
modelview = glm::rotate( modelview, D2R*Xrot, glm::vec3(1.,0.,0.) );   // {x',y',z'} = [v]*[yr]*[xr]*{x,y,z}
modelview = glm::scale( modelview, glm::vec3(Scale,Scale,Scale) );   // {x',y',z'} = [v]*[yr]*[xr]*[s]*{x,y,z}
```

This is exactly the same concept as OpenGL, but a different expression of it. Read on for details …

The Most Useful GLM Variables, Operations, and Functions

// constructor:

```
glm::mat4( 1. );               // identity matrix
glm::vec4( );
glm::vec3( );
```

GLM recommends that you use the **"glm::"** syntax and avoid **"using namespace"** syntax because they have not made any effort to create unique function names

// multiplications:

```
glm::mat4  *  glm::mat4
glm::mat4  *  glm::vec4
glm::mat4  *  glm::vec4(  glm::vec3,  1.  )        // promote a vec3 to a vec4 via a constructor
```

// emulating OpenGL transformations *with concatenation*:

glm::mat4 **glm::rotate**(glm::mat4 const & m, float angle, glm::vec3 const & axis);

glm::mat4 **glm::scale**(glm::mat4 const & m, glm::vec3 const & factors);

glm::mat4 **glm::translate**(glm::mat4 const & m, glm::vec3 const & translation);

The Most Useful GLM Variables, Operations, and Functions


```
// viewing volume  (assign, not concatenate):

glm::mat4  glm::ortho(  float left,  float right,  float bottom,  float top,  float near,  float far );
glm::mat4  glm::ortho(  float left,  float right,  float bottom,  float top );

glm::mat4  glm::frustum(  float left,  float right,  float bottom,  float top,  float near,  float far );
glm::mat4  glm::perspective(  float fovy,  float aspect,  float near,  float far);


// viewing  (assign, not concatenate):

glm::mat4  glm::lookAt(  glm::vec3 const & eye,  glm::vec3 const & look,  glm::vec3 const & up );
```


GLM in the Vulkan sample.cpp Program


```
 if( UseMouse )
{
        if( Scale < MINSCALE )
           Scale = MINSCALE;
        Matrices.uModelMatrix = glm::mat4( 1. );        // identity
        Matrices.uModelMatrix = glm::rotate( Matrices.uModelMatrix,  Yrot, glm::vec3( 0.,1.,0.) );
        Matrices.uModelMatrix = glm::rotate( Matrices.uModelMatrix,  Xrot, glm::vec3( 1.,0.,0.) );
        Matrices.uModelMatrix = glm::scale(  Matrices.uModelMatrix,  glm::vec3(Scale,Scale,Scale) );
                              // done this way, the Scale is applied first, then the Xrot, then the Yrot
}
else
{
      if( ! Paused )
      {
           const glm::vec3 axis = glm::vec3( 0., 1., 0. );
           Matrices.uModelMatrix     = glm::rotate( glm::mat4( 1. ),  (float)glm::radians(  360.f*Time/SECONDS_PER_CYCLE ),   axis );
       }
}

glm::vec3 eye(0.,0.,EYEDIST );
glm::vec3 look(0.,0.,0.);
glm::vec3   up(0.,1.,0.);
Matrices.uVewMatrix  = glm::lookAt( eye,  look,  up );

Matrices.uProjectionMatrix  = glm::perspective( FOV, (double)Width/(double)Height, 0.1f, 1000.f );
Matrices.uProjectionMatrix[1][1] *= -1.;              // Vulkan's projected Y is inverted from OpenGL

Matrices.uNormalMatrix  = glm::inverseTranspose(   glm::mat3( Matrices.uModelMatrix  );   // note: inverseTransform !

Fill05DataBuffer(  MyMatrixUniformBuffer,  (void *) &Matrices  );

Misc.uTime = (float)Time;
Misc.uMode = Mode;
Fill05DataBuffer(  MyMiscUniformBuffer,  (void *) &Misc  );
```


From the Data Buffer Noteset

Here's the vertex shader shader code to use the matrices:

```
layout( std140, set = 0, binding = 0 ) uniform sceneMatBuf
{
        mat4 uProjectionMatrix;
        mat4 uViewMatrix;
        mat4 uSceneMatrix;
} SceneMatrices;

layout( std140, set = 1, binding = 0 ) uniform objectMatBuf
{
        mat4 uModelMatrix;
        mat4 uNormalMatrix;
} ObjectMatrices;
```

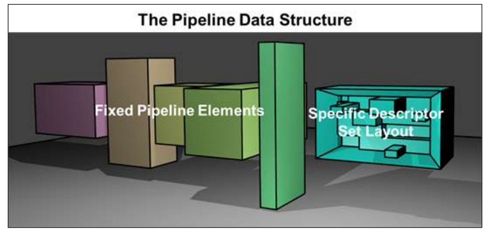
```
vNormal = uNormalMatrix * aNormal;

gl_Position = uProjectMatrix * uViewMatrix * uSceneMatrix * uModelMatrix * aVertex;
```

"CTM"

Vulkan.

Descriptor Sets

The Pipeline Data Structure



In OpenGL

OpenGL puts all uniform data in the same "set", but with different binding numbers, so you can get at each one.

Each uniform variable gets updated one-at-a-time.

Wouldn't it be nice if we could update a collection of related uniform variables all at once, without having to update the uniform variables that are not related to this collection?

```
layout( std140, binding = 0 )  uniform mat4       uModelMatrix;
layout( std140, binding = 1 ) uniform mat4        uViewMatrix;
layout( std140, binding = 2 ) uniform mat4        uProjectionMatrix;
layout( std140, binding = 3 ) uniform mat3        uNormalMatrix;
layout( std140, binding = 4 ) uniform vec4         uLightPos;
layout( std140, binding = 5 ) uniform float         uTime;
layout( std140, binding = 6 ) uniform  int           uMode;
layout(             binding = 7 ) uniform sampler2D uSampler;
```

std140 has to do with the alignment of the different data types. It is the simplest, and so we use it in class to give everyone the highest probability that their system will be compatible with the alignment.

What are Descriptor Sets?

Descriptor Sets are an intermediate data structure that tells shaders how to connect information held in GPU memory to groups of related uniform variables and texture sampler declarations in shaders. There are three advantages in doing things this way:

- Related uniform variables can be updated as a group, gaining efficiency.
- Descriptor Sets are activated when the Command Buffer is filled. Different values for the uniform buffer variables can be toggled by just swapping out the Descriptor Set that points to GPU memory, rather than re-writing the GPU memory.
- Values for the shaders' uniform buffer variables can be compartmentalized into what quantities change often and what change seldom (scene-level, model-level, draw-level), so that uniform variables need to be re-written no more often than is necessary.

```
for( sporadically )
{
        Bind Descriptor Set #0
        for( the entire scene )
        {
                Bind Descriptor Set #1
                for( each object in the scene )
                {
                        Bind Descriptor Set #2
                        Do the drawing
                }
        }
}
```

Descriptor Sets

Our example will assume the following shader uniform variables:

```
// non-opaque must be in a uniform block:
layout( std140, set = 0, binding = 0 ) uniform sporadicBuf
{
        int             uMode;
        int             uUseLighting;
        int             uNumInstances;
} Sporadic;

layout( std140, set = 1, binding = 0 ) uniform sceneBuf
{
        mat4            uProjection;
        mat4            uView;
        mat4            uSceneOrient;
        vec4            uLightPos;
        vec4            uLightColor;
        vec4            uLightKaKdKs;
        float           uTime;
} Scene;

layout( std140, set = 2, binding = 0 ) uniform objectBuf
{
        mat4            uModel;
        mat4            uNormal;
        vec4            uColor;
        float           uShininess;
} Object;

layout( set = 3, binding = 0 ) uniform sampler2D uSampler;
```


Descriptor Sets

CPU:	GPU:	GPU:
Uniform data created in a C++ data structure	Uniform data in a "blob"*	Uniform data used in the shader

```
struct sporadicBuf
{
        int             uMode;
        int             uUseLighting;
        int             uNumInstances;
} Sporadic;

struct sceneBuf
{
        glm::mat4       uProjection;
        glm::mat4       uView;
        glm::mat4       uSceneOrient;
        glm::vec4       uLightPos;
        glm::vec4       uLightColor;
        glm::vec4       uLightKaKdKs;
        float           uTime;
} Scene;

struct objectBuf
{
        glm::mat4       uModel;
        glm::mat4       uNormal;
        glm::vec4       uColor;
        float           uShininess;
} Object;
```

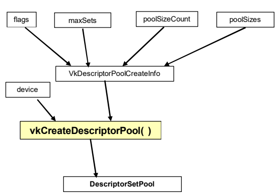
10111001010101111110100010
00010111011010111101001101
10011000000111010111110011
10110100110010111111010111
00110110101000000100100011
11010001001010101010011111
11001000011100101010100011
00011011010110111111011111
11111001101010100001011110
11001100100011010001010001
[illegible]
[illegible]
01101001001000110101011000
00001111100011000010000001
10110011101010001110100001
10011001100100001100001110
01111001001111010010001000
10110011100010111000000010
10000101111110101111100011
10001011100001100111010001
11100111111011101111111101
11111010011110111101011111
00101010000001111001001110
01110011111010010110011100
11001110001101100011110111
00001111100011100101100100
01110011010111011100101000

```
// non-opaque must be in a uniform block:
layout( std140, set = 0, binding = 0 ) uniform sporadicBuf
{
        int             uMode;
        int             uUseLighting;
        int             uNumInstances;
} Sporadic;

layout( std140, set = 1, binding = 0 ) uniform sceneBuf
{
        mat4            uProjection;
        mat4            uView;
        mat4            uSceneOrient;
        vec4            uLightPos;
        vec4            uLightColor;
        vec4            uLightKaKdKs;
        float           uTime;
} Scene;

layout( std140, set = 2, binding = 0 ) uniform objectBuf
{
        mat4            uModel;
        mat4            uNormal;
        vec4            uColor;
        float           uShininess;
} Object;

layout( set = 3, binding = 0 ) uniform sampler2D uSampler;
```

* "binary large object"

Step 1: Descriptor Set Pools

You don't allocate Descriptor Sets on the fly – that is too slow.
Instead, you allocate a "pool" of Descriptor Sets during initialization and then pull from that pool later.




```
VkResult
Init13DescriptorSetPool( )
{
        VkResult result;

        VkDescriptorPoolSize                    vdps[4];
                vdps[0].type = VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER;
                vdps[0].descriptorCount = 1;
                vdps[1].type = VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER;
                vdps[1].descriptorCount = 1;
                vdps[2].type = VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER;
                vdps[2].descriptorCount = 1;
                vdps[3].type = VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER;
                vdps[3].descriptorCount = 1;

#ifdef CHOICES
VK_DESCRIPTOR_TYPE_SAMPLER
VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE
VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER
VK_DESCRIPTOR_TYPE_STORAGE_IMAGE
VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER
VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER
VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER
VK_DESCRIPTOR_TYPE_STORAGE_BUFFER
VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC
VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC
VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT
#endif

        VkDescriptorPoolCreateInfo              vdpci;
                vdpci.sType = VK_STRUCTURE_TYPE_DESCRIPTOR_POOL_CREATE_INFO;
                vdpci.pNext = nullptr;
                vdpci.flags = 0;
                vdpci.maxSets = 4;
                vdpci.poolSizeCount = 4;
                vdpci.pPoolSizes = &vdps[0];

        result = vkCreateDescriptorPool( LogicalDevice, IN &vdpci, PALLOCATOR, OUT &DescriptorPool );
        return result;
}
```


Step 2: Define the Descriptor Set Layouts

I think of Descriptor Set Layouts as a kind of "Rosetta Stone" that allows the Graphics Pipeline data structure to allocate room for the uniform variables and to access them.

https://www.britishmuseum.org

```
// non-opaque must be in a uniform block:
layout( std140, set = 0, binding = 0 ) uniform sporadicBuf
{
        int             uMode;
        int             uUseLighting;
        int             uNumInstances;
} Sporadic;

layout( std140, set = 1, binding = 0 ) uniform sceneBuf
{
        mat4            uProjection;
        mat4            uView;
        mat4            uSceneOrient;
        vec4            uLightPos;
        vec4            uLightColor;
        vec4            uLightKaKdKs;
        float           uTime;
} Scene;

layout( std140, set = 2, binding = 0 ) uniform objectBuf
{
        mat4            uModel;
        mat4            uNormal;
        vec4            uColor;
        float           uShininess;
} Object;

layout( set = 3, binding = 0 ) uniform sampler2D uSampler;
```

SporadicSet DS Layout Binding:	SceneSet DS Layout Binding:	ObjectSet DS Layout Binding:	TexSamplerSet DS Layout Binding:
binding descriptorType descriptorCount pipeline stage(s)	binding descriptorType descriptorCount pipeline stage(s)	binding descriptorType descriptorCount pipeline stage(s)	binding descriptorType descriptorCount pipeline stage(s)
set = 0	set = 1	set = 2	set = 3

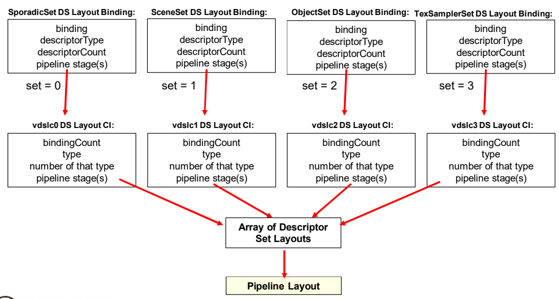

```
VkResult
Init13DescriptorSetLayouts( )
{
        VkResult result;

        //DS #0:
        VkDescriptorSetLayoutBinding            SporadicSet[1];
                SporadicSet[0].binding                  = 0;
                SporadicSet[0].descriptorType           = VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER;
                SporadicSet[0].descriptorCount          = 1;
                SporadicSet[0].stageFlags               = VK_SHADER_STAGE_VERTEX_BIT | VK_SHADER_STAGE_FRAGMENT_BIT;
                SporadicSet[0].pImmutableSamplers = (VkSampler *)nullptr;

        // DS #1:
        VkDescriptorSetLayoutBinding            SceneSet[1];
                SceneSet[0].binding                     = 0;
                SceneSet[0].descriptorType              = VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER;
                SceneSet[0].descriptorCount             = 1;
                SceneSet[0].stageFlags                  = VK_SHADER_STAGE_VERTEX_BIT | VK_SHADER_STAGE_FRAGMENT_BIT;
                SceneSet[0].pImmutableSamplers = (VkSampler *)nullptr;

        //DS #2:
        VkDescriptorSetLayoutBinding            ObjectSet[1];
                ObjectSet[0].binding                    = 0;
                ObjectSet[0].descriptorType             = VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER;
                ObjectSet[0].descriptorCount            = 1;
                ObjectSet[0].stageFlags                 = VK_SHADER_STAGE_VERTEX_BIT | VK_SHADER_STAGE_FRAGMENT_BIT;
                ObjectSet[0].pImmutableSamplers = (VkSampler *)nullptr;

        // DS #3:
        VkDescriptorSetLayoutBinding            TexSamplerSet[1];
                TexSamplerSet[0].binding                = 0;
                TexSamplerSet[0].descriptorType         = VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER;
                TexSamplerSet[0].descriptorCount        = 1;
                TexSamplerSet[0].stageFlags             = VK_SHADER_STAGE_FRAGMENT_BIT;
                TexSamplerSet[0].pImmutableSamplers = (VkSampler *)nullptr;
```

```
uniform sampler2D   uSampler;
vec4 rgba = texture( uSampler, vST );
```

Step 2: Define the Descriptor Set Layouts

127

```
// globals:
VkDescriptorPool          DescriptorPool;
VkDescriptorSetLayout     DescriptorSetLayouts[4];
VkDescriptorSet           DescriptorSets[4];
```



SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

128

```
VkDescriptorSetLayoutCreateInfo          vdslc0;
      vdslc0.sType = VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_CREATE_INFO;
      vdslc0.pNext = nullptr;
      vdslc0.flags = 0;
      vdslc0.bindingCount = 1;
      vdslc0.pBindings = &SporadicSet[0];

VkDescriptorSetLayoutCreateInfo          vdslc1;
      vdslc1.sType = VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_CREATE_INFO;
      vdslc1.pNext = nullptr;
      vdslc1.flags = 0;
      vdslc1.bindingCount = 1;
      vdslc1.pBindings = &SceneSet[0];

VkDescriptorSetLayoutCreateInfo          vdslc2;
      vdslc2.sType = VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_CREATE_INFO;
      vdslc2.pNext = nullptr;
      vdslc2.flags = 0;
      vdslc2.bindingCount = 1;
      vdslc2.pBindings = &ObjectSet[0];

VkDescriptorSetLayoutCreateInfo          vdslc3;
      vdslc3.sType = VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_CREATE_INFO;
      vdslc3.pNext = nullptr;
      vdslc3.flags = 0;
      vdslc3.bindingCount = 1;
      vdslc3.pBindings = &TexSamplerSet[0];

result = vkCreateDescriptorSetLayout( LogicalDevice,  IN &vdslc0, PALLOCATOR, OUT &DescriptorSetLayouts[0] );
result = vkCreateDescriptorSetLayout( LogicalDevice,  IN &vdslc1, PALLOCATOR, OUT &DescriptorSetLayouts[1] );
result = vkCreateDescriptorSetLayout( LogicalDevice,  IN &vdslc2, PALLOCATOR, OUT &DescriptorSetLayouts[2] );
result = vkCreateDescriptorSetLayout( LogicalDevice,  IN &vdslc3, PALLOCATOR, OUT &DescriptorSetLayouts[3] );

return result;
}
```

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Step 3: Include the Descriptor Set Layouts in a Graphics Pipeline Layout

129

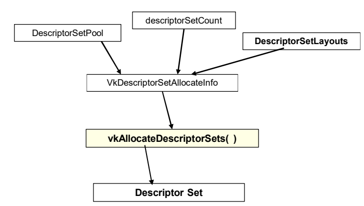
```
VkResult
Init14GraphicsPipelineLayout( )
{
      VkResult result;

      VkPipelineLayoutCreateInfo          vplci;
            vplci.sType = VK_STRUCTURE_TYPE_PIPELINE_LAYOUT_CREATE_INFO;
            vplci.pNext = nullptr;
            vplci.flags = 0;
            vplci.setLayoutCount = 4;
            vplci.pSetLayouts = &DescriptorSetLayouts[0];
            vplci.pushConstantRangeCount = 0;
            vplci.pPushConstantRanges = (VkPushConstantRange *)nullptr;

      result = vkCreatePipelineLayout( LogicalDevice, IN &vplci, PALLOCATOR, OUT &GraphicsPipelineLayout );

      return result;
}
```

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Step 4: Allocating the Memory for Descriptor Sets

130



SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Step 4: Allocating the Memory for Descriptor Sets

131

```
VkResult
Init13DescriptorSets( )
{
      VkResult result;

      VkDescriptorSetAllocateInfo          vdsai;
            vdsai.sType = VK_STRUCTURE_TYPE_DESCRIPTOR_SET_ALLOCATE_INFO;
            vdsai.pNext = nullptr;
            vdsai.descriptorPool = DescriptorPool;
            vdsai.descriptorSetCount = 4;
            vdsai.pSetLayouts = DescriptorSetLayouts;

      result = vkAllocateDescriptorSets( LogicalDevice, IN &vdsai, OUT &DescriptorSets[0] );
```

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Step 5: Tell the Descriptor Sets where their CPU Data is

132

```
VkDescriptorBufferInfo                 vdbi0;
      vdbi0.buffer = MySporadicUniformBuffer.buffer;
      vdbi0.offset = 0;
      vdbi0.range = sizeof(Sporadic);

VkDescriptorBufferInfo                 vdbi1;
      vdbi1.buffer = MySceneUniformBuffer.buffer;
      vdbi1.offset = 0;
      vdbi1.range = sizeof(Scene);

VkDescriptorBufferInfo                 vdbi2;
      vdbi2.buffer = MyObjectUniformBuffer.buffer;
      vdbi2.offset = 0;
      vdbi2.range = sizeof(Object);

VkDescriptorImageInfo                  vdii0;
      vdii.sampler     = MyPuppyTexture.texSampler;
      vdii.imageView = MyPuppyTexture.texImageView;
      vdii.imageLayout = VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL;
```

This struct identifies what buffer it owns and how big it is

This struct identifies what buffer it owns and how big it is

This struct identifies what buffer it owns and how big it is

This struct identifies what texture sampler and image view it owns

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Step 5: Tell the Descriptor Sets where their CPU Data is

133

```
VkWriteDescriptorSet                    vwds0;
        // ds 0:
        vwds0.sType = VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET;
        vwds0.pNext = nullptr;
        vwds0.dstSet = DescriptorSets[0];
        vwds0.dstBinding = 0;
        vwds0.dstArrayElement = 0;
        vwds0.descriptorCount = 1;
        vwds0.descriptorType = VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER;
        vwds0.pBufferInfo = IN &vdbi0;
        vwds0.pImageInfo = (VkDescriptorImageInfo *)nullptr;
        vwds0.pTexelBufferView = (VkBufferView *)nullptr;

        // ds 1:
    VkWriteDescriptorSet                    vwds1;
        vwds1.sType = VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET;
        vwds1.pNext = nullptr;
        vwds1.dstSet = DescriptorSets[1];
        vwds1.dstBinding = 0;
        vwds1.dstArrayElement = 0;
        vwds1.descriptorCount = 1;
        vwds1.descriptorType = VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER;
        vwds1.pBufferInfo = IN &vdbi1;
        vwds1.pImageInfo = (VkDescriptorImageInfo *)nullptr;
        vwds1.pTexelBufferView = (VkBufferView *)nullptr;
```

This struct links a Descriptor Set to the buffer it is pointing to

This struct links a Descriptor Set to the buffer it is pointing to

SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Step 5: Tell the Descriptor Sets where their data is

134

```
VkWriteDescriptorSet                    vwds2;
    // ds 2:
    vwds2.sType = VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET;
    vwds2.pNext = nullptr;
    vwds2.dstSet = DescriptorSets[2];
    vwds2.dstBinding = 0;
    vwds2.dstArrayElement = 0;
    vwds2.descriptorCount = 1;
    vwds2.descriptorType = VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER;
    vwds2.pBufferInfo = IN &vdbi2;
    vwds2.pImageInfo = (VkDescriptorImageInfo *)nullptr;
    vwds2.pTexelBufferView = (VkBufferView *)nullptr;

    // ds 3:
VkWriteDescriptorSet                    vwds3;
    vwds3.sType = VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET;
    vwds3.pNext = nullptr;
    vwds3.dstSet = DescriptorSets[3];
    vwds3.dstBinding = 0;
    vwds3.dstArrayElement = 0;
    vwds3.descriptorCount = 1;
    vwds3.descriptorType = VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER;
    vwds3.pBufferInfo = (VkDescriptorBufferInfo *)nullptr;
    vwds3.pImageInfo = IN &vdii0;
    vwds3.pTexelBufferView = (VkBufferView *)nullptr;

uint32_t copyCount = 0;

// this could have been done with one call and an array of VkWriteDescriptorSets:

vkUpdateDescriptorSets( LogicalDevice, 1, IN &vwds0, IN copyCount, (VkCopyDescriptorSet *)nullptr );
vkUpdateDescriptorSets( LogicalDevice, 1, IN &vwds1, IN copyCount, (VkCopyDescriptorSet *)nullptr );
vkUpdateDescriptorSets( LogicalDevice, 1, IN &vwds2, IN copyCount, (VkCopyDescriptorSet *)nullptr );
vkUpdateDescriptorSets( LogicalDevice, 1, IN &vwds3, IN copyCount, (VkCopyDescriptorSet *)nullptr );
```

This struct links a Descriptor Set to the buffer it is pointing to

This struct links a Descriptor Set to the image it is pointing to

SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Step 6: Include the Descriptor Set Layout when Creating a Graphics Pipeline

135

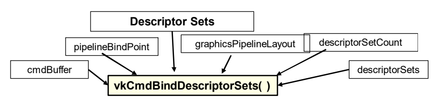
```
VkGraphicsPipelineCreateInfo                    vgpci;
        vgpci.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
        vgpci.pNext = nullptr;
        vgpci.flags = 0;
#ifdef CHOICES
VK_PIPELINE_CREATE_DISABLE_OPTIMIZATION_BIT
VK_PIPELINE_CREATE_ALLOW_DERIVATIVES_BIT
VK_PIPELINE_CREATE_DERIVATIVE_BIT
#endif
        vgpci.stageCount = 2;                   // number of stages in this pipeline
        vgpci.pStages = vpssci;
        vgpci.pVertexInputState = &vpvisci;
        vgpci.pInputAssemblyState = &vpiasci;
        vgpci.pTessellationState = (VkPipelineTessellationStateCreateInfo *)nullptr;
        vgpci.pViewportState = &vpvsci;
        vgpci.pRasterizationState = &vprsci;
        vgpci.pMultisampleState = &vpmsci;
        vgpci.pDepthStencilState = &vpdssci;
        vgpci.pColorBlendState = &vpcbsci;
        vgpci.pDynamicState = &vpdsci;
        vgpci.layout = IN GraphicsPipelineLayout;
        vgpci.renderPass = IN RenderPass;
        vgpci.subpass = 0;                      // subpass number
        vgpci.basePipelineHandle = (VkPipeline) VK_NULL_HANDLE;
        vgpci.basePipelineIndex = 0;

    result = vkCreateGraphicsPipelines( LogicalDevice, VK_NULL_HANDLE, 1, IN &vgpci,
                                        PALLOCATOR, OUT &GraphicsPipeline );
```

SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Step 7: Bind Descriptor Sets into the Command Buffer when Drawing

136




```
vkCmdBindDescriptorSets( CommandBuffers[nextImageIndex],
        VK_PIPELINE_BIND_POINT_GRAPHICS, GraphicsPipelineLayout,
        0, 4, DescriptorSets, 0, (uint32_t *)nullptr );
```

So, the Pipeline Layout contains the ***structure*** of the Descriptor Sets.
Any collection of Descriptor Sets that match that structure can be bound into that pipeline.

SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Sidebar: The Entire Descriptor Set Journey

137

VkDescriptorPoolCreateInfo
vkCreateDescriptorPool( )
} Create the pool of Descriptor Sets for future use

VkDescriptorSetLayoutBinding
VkDescriptorSetLayoutCreateInfo
vkCreateDescriptorSetLayout( )
vkCreatePipelineLayout( )
} Describe a particular Descriptor Set layout and use it in a specific Pipeline layout

VkDescriptorSetAllocateInfo
vkAllocateDescriptorSets( )
} Allocate memory for particular Descriptor Sets

VkDescriptorBufferInfo
VkDescriptorImageInfo
VkWriteDescriptorSet
} Tell a particular Descriptor Set where its CPU data is
vkUpdateDescriptorSets( )
} Re-write CPU data into a particular Descriptor Set

vkCmdBindDescriptorSets( )
} Make a particular Descriptor Set "current" for rendering

SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

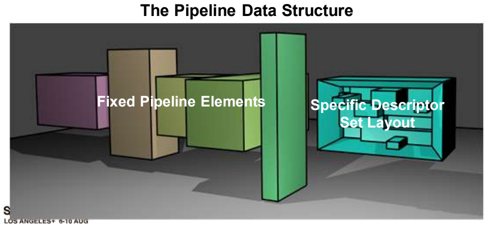
Sidebar: Why Do Descriptor Sets Need to Provide Layout Information to the Pipeline Data Structure?

138

The pieces of the Pipeline Data Structure are fixed in size – with the exception of the Descriptor Sets and the Push Constants. Each of these two can be any size, depending on what you allocate for them. So, the Pipeline Data Structure needs to know how these two are configured before it can set its own total layout.

Think of the DS layout as being a particular-sized hole in the Pipeline Data Structure. Any data you have that matches this hole's shape and size can be plugged in there.

The Pipeline Data Structure



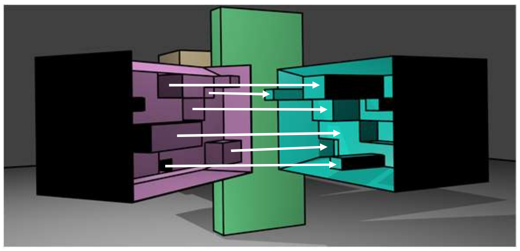
SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Sidebar: Why Do Descriptor Sets Need to Provide Layout Information to the Pipeline Data Structure?

139

Any set of data that matches the Descriptor Set Layout can be plugged in there.



140

Vulkan.

Textures

The Basic Idea

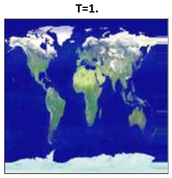
141

Texture mapping is a computer graphics operation in which a separate image, referred to as the **texture**, is stretched onto a piece of 3D geometry and follows it however it is transformed. This image is also known as a ***texture map***.

Also, to prevent confusion, the texture pixels are not called ***pixels***. A pixel is a dot in the final screen image. A dot in the texture image is called a ***texture element***, or ***texel***.

Similarly, to avoid terminology confusion, a texture's width and height dimensions are not called *X* and *Y*. They are called **S** and **T**. A texture map is not generally indexed by its actual resolution coordinates. Instead, it is indexed by a coordinate system that is resolution-independent. The left side is always **S=0.**, the right side is **S=1.**, the bottom is **T=0.**, and the top is **T=1.** Thus, you do not need to be aware of the texture's resolution when you are specifying coordinates that point into it. Think of S and T as a measure of what fraction of the way you are into the texture.

T=1.

S=0.  S=1.

T=0.

In OpenGL terms: assigning an (s,t) to each vertex

142

Enable texture mapping:

glEnable(GL_TEXTURE_2D);

Draw your polygons, specifying **s** and **t** at each vertex:

```
glBegin( GL_TRIANGLES );
    glTexCoord2f( s0, t0 );
    glNormal3f( nx0, ny0, nz0 );
    glVertex3f( x0, y0, z0 );

    glTexCoord2f( s1, t1 );
    glNormal3f( nx1, ny1, nz1 );
    glVertex3f( x1, y1, z1 );

    . . .
glEnd( );
```

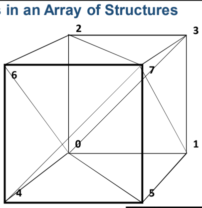
Disable texture mapping:

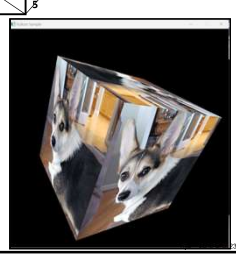
glDisable(GL_TEXTURE_2D);

Triangles in an Array of Structures

143

```
struct vertex
{
    glm::vec3    position;
    glm::vec3    normal;
    glm::vec3    color;
    glm::vec2    texCoord;
};

struct vertex VertexData[ ] =
{
    // triangle 0-2-3:
    // vertex #0:
    {
        { -1., -1., -1. },
        {  0.,  0., -1. },
        {  0.,  0.,  0. },
        {  1., 0. }
    },

    // vertex #2:
    {
        { -1.,  1., -1. },
        {  0.,  0., -1. },
        {  0.,  1.,  0. },
        {  1., 1. }
    },

    // vertex #3:
    {
        {  1.,  1., -1. },
        {  0.,  0., -1. },
        {  1.,  1.,  0. },
        {  0., 1. }
    },
```

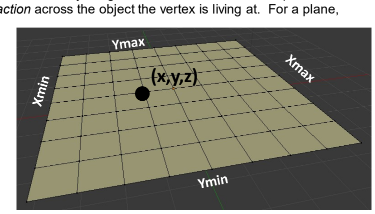




Using a Texture: How do you know what (s,t) to assign to each vertex?

144

The easiest way to figure out what s and t are at a particular vertex is to figure out what *fraction* across the object the vertex is living at. For a plane,

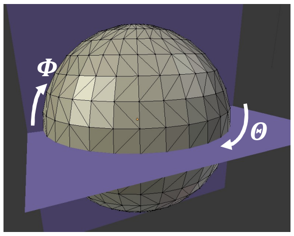


$$s = \frac{x - Xmin}{Xmax - Xmin} \qquad t = \frac{y - Ymin}{Ymax - Ymin}$$

Using a Texture: How do you know what (s,t) to assign to each vertex?

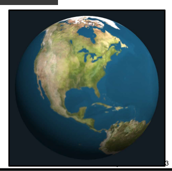
145

Or, for a sphere,



$$s = \frac{\Theta - (-\pi)}{2\pi} \qquad t = \frac{\Phi - (-\frac{\pi}{2})}{\pi}$$

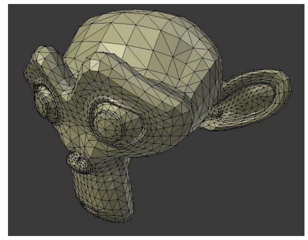
s = (lng + M_PI) / (2.*M_PI);
t = (lat + M_PI/2.) / M_PI;



Using a Texture: How do you know what (s,t) to assign to each vertex?

146

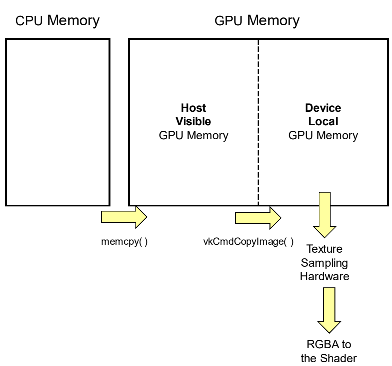
Uh-oh. Now what? Here's where it gets tougher…,



$s = ?$ $\qquad$ $t = ?$

Memory Types

147



Memory Types

148

NVIDIA A6000 Graphics:

6 Memory Types:
Memory 0:
Memory 1: DeviceLocal
Memory 2: HostVisible HostCoherent
Memory 3: HostVisible HostCoherent HostCached
Memory 4: DeviceLocal HostVisible HostCoherent
Memory 5: DeviceLocal

Intel Integrated Graphics:

3 Memory Types:
Memory 0: DeviceLocal
Memory 1: DeviceLocal HostVisible HostCoherent
Memory 2: DeviceLocal HostVisible HostCoherent HostCached

Something I've Found Useful

149

I find it handy to encapsulate texture information in a struct, just like I do with buffer information:

```
// holds all the information about a data buffer so it can be encapsulated in one variable:

typedef struct MyBuffer
{
	VkDataBuffer		buffer;
	VkDeviceMemory		vdm;
	VkDeviceSize		size;
} MyBuffer;

// holds all the information about a texture so it can be encapsulated in one variable:

typedef struct MyTexture
{
	uint32_t		width;
	uint32_t		height;
	unsigned char *		pixels;
	VkImage			texImage;
	VkImageView		texImageView;
	VkSampler		texSampler;
	VkDeviceMemory		vdm;
} MyTexture;
```


Texture Sampling Parameters

150

OpenGL

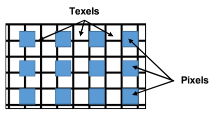
```
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR );
```

Vulkan

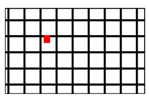
```
MyTexture  MyPuppyTexture;
. . .
VkSamplerCreateInfo			vsci;
	vsci.magFilter = VK_FILTER_LINEAR;
	vsci.minFilter = VK_FILTER_LINEAR;
	vsci.mipmapMode    = VK_SAMPLER_MIPMAP_MODE_LINEAR;
	vsci.addressModeU  = VK_SAMPLER_ADDRESS_MODE_REPEAT;
	vsci.addressModeV  = VK_SAMPLER_ADDRESS_MODE_REPEAT;
	vsci.addressModeW  = VK_SAMPLER_ADDRESS_MODE_REPEAT;

	. . .
result = vkCreateSampler( LogicalDevice, IN &vsci, PALLOCATOR, OUT &MyPuppyTexture->texSampler);
```

Textures' Undersampling Artifacts

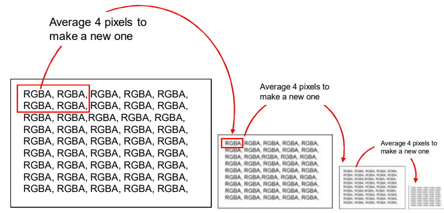
As an object gets farther away and covers a smaller and smaller part of the screen, the **texels : pixels ratio** used in the coverage becomes larger and larger. This means that there are pieces of the texture leftover in between the pixels that are being drawn into, so that some of the texture image is not being taken into account in the final image. This means that the texture is being undersampled and could end up producing artifacts in the rendered image.



Consider a texture that consists of one red texel and all the rest white. It is easy to imagine an object rendered with that texture as ending up all *white*, with the red texel having never been included in the final image. The solution is to create lower-resolutions of the same texture so that the red texel gets included somehow in all resolution-level textures.



Texture Mip*-mapping



- Total texture storage is ~ 2x what it was without mip-mapping
- Graphics hardware determines which level to use based on the texels : pixels ratio.
- In addition to just picking one mip-map level, the rendering system can sample from two of them, one less that the Texture:Pixel ratio and one more, and then blend the two RGBAs returned. This is known as **VK_SAMPLER_MIPMAP_MODE_LINEAR**.

* Latin: *multim in parvo*, "many things in a small place"


```
VkResult
Init07TextureSampler( MyTexture * pMyTexture )
{
	VkResult result;

	VkSamplerCreateInfo			vsci;
		vsci.sType = VK_STRUCTURE_TYPE_SAMPLER_CREATE_INFO;
		vsci.pNext = nullptr;
		vsci.flags = 0;
		vsci.magFilter = VK_FILTER_LINEAR;
		vsci.minFilter = VK_FILTER_LINEAR;
		vsci.mipmapMode = VK_SAMPLER_MIPMAP_MODE_LINEAR;
		vsci.addressModeU = VK_SAMPLER_ADDRESS_MODE_REPEAT;
		vsci.addressModeV = VK_SAMPLER_ADDRESS_MODE_REPEAT;
		vsci.addressModeW = VK_SAMPLER_ADDRESS_MODE_REPEAT;
#ifdef CHOICES
VK_SAMPLER_ADDRESS_MODE_REPEAT
VK_SAMPLER_ADDRESS_MODE_MIRRORED_REPEAT
VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_EDGE
VK_SAMPLER_ADDRESS_MODE_CLAMP_TO_BORDER
VK_SAMPLER_ADDRESS_MODE_MIRROR_CLAMP_TO_EDGE
#endif
		vsci.mipLodBias = 0.;
		vsci.anisotropyEnable = VK_FALSE;
		vsci.maxAnisotropy = 1.;
		vsci.compareEnable = VK_FALSE;
		vsci.compareOp = VK_COMPARE_OP_NEVER;
#ifdef CHOICES
VK_COMPARE_OP_NEVER
VK_COMPARE_OP_LESS
VK_COMPARE_OP_EQUAL
VK_COMPARE_OP_LESS_OR_EQUAL
VK_COMPARE_OP_GREATER
VK_COMPARE_OP_NOT_EQUAL
VK_COMPARE_OP_GREATER_OR_EQUAL
VK_COMPARE_OP_ALWAYS
#endif
		vsci.minLod = 0.;
		vsci.maxLod = 0.;
		vsci.borderColor = VK_BORDER_COLOR_FLOAT_OPAQUE_BLACK;
#ifdef CHOICES
VK_BORDER_COLOR_FLOAT_TRANSPARENT_BLACK
VK_BORDER_COLOR_INT_TRANSPARENT_BLACK
VK_BORDER_COLOR_FLOAT_OPAQUE_BLACK
VK_BORDER_COLOR_INT_OPAQUE_BLACK
VK_BORDER_COLOR_FLOAT_OPAQUE_WHITE
VK_BORDER_COLOR_INT_OPAQUE_WHITE
#endif
		vsci.unnormalizedCoordinates = VK_FALSE;	// VK_TRUE means we are use raw texels as the index
							// VK_FALSE means we are using the usual 0. - 1.

	result = vkCreateSampler( LogicalDevice, IN &vsci, PALLOCATOR, OUT &MyPuppyTexture->texSampler );
```



```
VkResult
Init07TextureBuffer( INOUT MyTexture * pMyTexture)
{
	VkResult result;

	uint32_t texWidth = pMyTexture->width;;
	uint32_t texHeight = pMyTexture->height;
	unsigned char *texture = pMyTexture->pixels;
	VkDeviceSize textureSize = texWidth * texHeight * 4;		// rgba, 1 byte each

	VkImage  stagingImage;
	VkImage  textureImage;

	// ******************************************************************************
	// this first {...} is to create the staging image:
	// ******************************************************************************
	{
		VkImageCreateInfo			vici;
			vici.sType = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;
			vici.pNext = nullptr;
			vici.flags = 0;
			vici.imageType = VK_IMAGE_TYPE_2D;
			vici.format = VK_FORMAT_R8G8B8A8_UNORM;
			vici.extent.width = texWidth;
			vici.extent.height = texHeight;
			vici.extent.depth = 1;
			vici.mipLevels = 1;
			vici.arrayLayers = 1;
			vici.samples = VK_SAMPLE_COUNT_1_BIT;
			vici.tiling = VK_IMAGE_TILING_LINEAR;
#ifdef CHOICES
VK_IMAGE_TILING_OPTIMAL
VK_IMAGE_TILING_LINEAR
#endif
			vici.usage = VK_IMAGE_USAGE_TRANSFER_SRC_BIT;
#ifdef CHOICES
VK_IMAGE_USAGE_TRANSFER_SRC_BIT
VK_IMAGE_USAGE_TRANSFER_DST_BIT
VK_IMAGE_USAGE_SAMPLED_BIT
VK_IMAGE_USAGE_STORAGE_BIT
VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT
VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT
VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT
VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT
#endif
			vici.sharingMode = VK_SHARING_MODE_EXCLUSIVE;
```



```
#ifdef CHOICES
VK_IMAGE_LAYOUT_UNDEFINED
VK_IMAGE_LAYOUT_PREINITIALIZED
#endif
			vici.queueFamilyIndexCount = 0;
			vici.pQueueFamilyIndices = (const uint32_t *)nullptr;

		result = vkCreateImage(LogicalDevice, IN &vici, PALLOCATOR, OUT &stagingImage); // allocated, but not filled

		VkMemoryRequirements			vmr;
		vkGetImageMemoryRequirements( LogicalDevice, IN stagingImage, OUT &vmr);

		if (Verbose)
		{
			fprintf(FpDebug, "Image vmr.size = %lld\n", vmr.size);
			fprintf(FpDebug, "Image vmr.alignment = %lld\n", vmr.alignment);
			fprintf(FpDebug, "Image vmr.memoryTypeBits = 0x%08x\n", vmr.memoryTypeBits);
			fflush(FpDebug);
		}

		VkMemoryAllocateInfo			vmai;
			vmai.sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
			vmai.pNext = nullptr;
			vmai.allocationSize = vmr.size;
			vmai.memoryTypeIndex = FindMemoryThatIsHostVisible();	// because we want to mmap it

		VkDeviceMemory				vdm;
		result = vkAllocateMemory( LogicalDevice, IN &vmai, PALLOCATOR, OUT &vdm);
		pMyTexture->vdm = vdm;

		result = vkBindImageMemory( LogicalDevice, IN stagingImage, IN vdm, 0);	// 0 = offset

		// we have now created the staging image -- fill it with the pixel data:

		VkImageSubresource			vis;
			vis.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
			vis.mipLevel = 0;
			vis.arrayLayer = 0;

		VkSubresourceLayout			vsl;
		vkGetImageSubresourceLayout( LogicalDevice, stagingImage, IN &vis, OUT &vsl);

		if (Verbose)
		{
			fprintf(FpDebug, "Subresource Layout:\n");
			fprintf(FpDebug, "\toffset = %lld\n", vsl.offset);
			fprintf(FpDebug, "\tsize = %lld\n", vsl.size);
			fprintf(FpDebug, "\trowPitch = %lld\n", vsl.rowPitch);
			fprintf(FpDebug, "\tarrayPitch = %lld\n", vsl.arrayPitch);
			fprintf(FpDebug, "\tdepthPitch = %lld\n", vsl.depthPitch);
			fflush(FpDebug);
		}
```



```
		void * gpuMemory;
		vkMapMemory( LogicalDevice, vdm, 0, VK_WHOLE_SIZE, 0, OUT &gpuMemory);
						// 0 and 0 = offset and memory map flags

		if (vsl.rowPitch == 4 * texWidth)
		{
			memcpy(gpuMemory, (void *)texture, (size_t)textureSize);
		}
		else
		{
			unsigned char *gpuBytes = (unsigned char *)gpuMemory;
			for (unsigned int y = 0; y < texHeight; y++)
			{
				memcpy(&gpuBytes[y * vsl.rowPitch], &texture[4 * y * texWidth], (size_t)(4*texWidth) );
			}
		}

		vkUnmapMemory( LogicalDevice, vdm);

	}
	// ******************************************************************************
```

157

```
// ****************************************************************************
// this second {...} is to create the actual texture image:
// ****************************************************************************
{
	VkImageCreateInfo			vici;
		vici.sType = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;
		vici.pNext = nullptr;
		vici.flags = 0;
		vici.imageType = VK_IMAGE_TYPE_2D;
		vici.format = VK_FORMAT_R8G8B8A8_UNORM;
		vici.extent.width  = texWidth;
		vici.extent.height = texHeight;
		vici.extent.depth  = 1;
		vici.mipLevels = 1;
		vici.arrayLayers = 1;
		vici.samples = VK_SAMPLE_COUNT_1_BIT;
		vici.tiling = VK_IMAGE_TILING_OPTIMAL;
		vici.usage = VK_IMAGE_USAGE_TRANSFER_DST_BIT | VK_IMAGE_USAGE_SAMPLED_BIT;
				// because we are transferring into it and will eventual sample from it
		vici.sharingMode = VK_SHARING_MODE_EXCLUSIVE;
		vici.initialLayout = VK_IMAGE_LAYOUT_PREINITIALIZED;
		vici.queueFamilyIndexCount = 0;
		vici.pQueueFamilyIndices = (const uint32_t *)nullptr;

	result = vkCreateImage(LogicalDevice, IN &vici, PALLOCATOR, OUT &textureImage); // allocated, but not filled

	VkMemoryRequirements			vmr;
	vkGetImageMemoryRequirements( LogicalDevice, IN textureImage, OUT &vmr);

	if( Verbose )
	{
		fprintf( FpDebug, "Texture vmr.size = %lld\n", vmr.size );
		fprintf( FpDebug, "Texture vmr.alignment = %lld\n", vmr.alignment );
		fprintf( FpDebug, "Texture vmr.memoryTypeBits = 0x%08x\n", vmr.memoryTypeBits );
		fflush( FpDebug );
	}
	VkMemoryAllocateInfo			vmai;
		vmai.sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
		vmai.pNext = nullptr;
		vmai.allocationSize = vmr.size;
		vmai.memoryTypeIndex = FindMemoryThatIsDeviceLocal( ); // because we want to sample from it

	VkDeviceMemory				vdm;
	result = vkAllocateMemory( LogicalDevice, IN &vmai, PALLOCATOR, OUT &vdm);

	result = vkBindImageMemory( LogicalDevice, IN textureImage, IN vdm, 0 );	// 0 = offset
}
// ****************************************************************************
```

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

158

```
// copy pixels from the staging image to the texture:

VkCommandBufferBeginInfo		vcbbi;
	vcbbi.sType = VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO;
	vcbbi.pNext = nullptr;
	vcbbi.flags = VK_COMMAND_BUFFER_USAGE_ONE_TIME_SUBMIT_BIT;
	vcbbi.pInheritanceInfo = (VkCommandBufferInheritanceInfo *)nullptr;

result = vkBeginCommandBuffer( TextureCommandBuffer, IN &vcbbi);

// ****************************************************************************
// transition the staging buffer layout:
// ****************************************************************************
{
	VkImageSubresourceRange		visr;
		visr.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
		visr.baseMipLevel = 0;
		visr.levelCount = 1;
		visr.baseArrayLayer = 0;
		visr.layerCount = 1;

	VkImageMemoryBarrier			vimb;
		vimb.sType = VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER;
		vimb.pNext = nullptr;
		vimb.oldLayout = VK_IMAGE_LAYOUT_PREINITIALIZED;
		vimb.newLayout = VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL;
		vimb.srcQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
		vimb.dstQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
		vimb.image = stagingImage;
		vimb.srcAccessMask = VK_ACCESS_HOST_WRITE_BIT;
		vimb.dstAccessMask = 0;
		vimb.subresourceRange = visr;

	vkCmdPipelineBarrier( TextureCommandBuffer,
			VK_PIPELINE_STAGE_HOST_BIT, VK_PIPELINE_STAGE_HOST_BIT, 0,
			0, (VkMemoryBarrier *)nullptr,
			0, (VkBufferMemoryBarrier *)nullptr,
			1, IN &vimb );
}
// ****************************************************************************
```

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

159

```
// ****************************************************************************
// transition the texture buffer layout:
// ****************************************************************************
{
	VkImageSubresourceRange		visr;
		visr.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
		visr.baseMipLevel = 0;
		visr.levelCount = 1;
		visr.baseArrayLayer = 0;
		visr.layerCount = 1;

	VkImageMemoryBarrier			vimb;
		vimb.sType = VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER;
		vimb.pNext = nullptr;
		vimb.oldLayout = VK_IMAGE_LAYOUT_PREINITIALIZED;
		vimb.newLayout = VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL;
		vimb.srcQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
		vimb.dstQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
		vimb.image = textureImage;
		vimb.srcAccessMask = 0;
		vimb.dstAccessMask = VK_ACCESS_TRANSFER_WRITE_BIT;
		vimb.subresourceRange = visr;

	vkCmdPipelineBarrier( TextureCommandBuffer,
		VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT, VK_PIPELINE_STAGE_TRANSFER_BIT, 0,
		0, (VkMemoryBarrier *)nullptr,
		0, (VkBufferMemoryBarrier *)nullptr,
		1, IN &vimb);

	// now do the final image transfer:

	VkImageSubresourceLayers		visl;
		visl.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
		visl.baseArrayLayer = 0;
		visl.mipLevel = 0;
		visl.layerCount = 1;

	VkOffset3D				vo3;
		vo3.x = 0;
		vo3.y = 0;
		vo3.z = 0;

	VkExtent3D				ve3;
		ve3.width = texWidth;
		ve3.height = texHeight;
		ve3.depth = 1;
```

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

160

```
	VkImageCopy				vic;
		vic.srcSubresource = visl;
		vic.srcOffset = vo3;
		vic.dstSubresource = visl;
		vic.dstOffset = vo3;
		vic.extent = ve3;

	vkCmdCopyImage(TextureCommandBuffer,
		stagingImage, VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL,
		textureImage, VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL, 1, IN &vic);
}
// ****************************************************************************
```

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

161

```
// ****************************************************************************
// transition the texture buffer layout a second time:
// ****************************************************************************
	{
		VkImageSubresourceRange		visr;
			visr.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
			visr.baseMipLevel = 0;
			visr.levelCount = 1;
			visr.baseArrayLayer = 0;
			visr.layerCount = 1;

		VkImageMemoryBarrier			vimb;
			vimb.sType = VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER;
			vimb.pNext = nullptr;
			vimb.oldLayout = VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL;
			vimb.newLayout = VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL;
			vimb.srcQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
			vimb.dstQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
			vimb.image = textureImage;
			vimb.srcAccessMask = 0;
			vimb.dstAccessMask = VK_ACCESS_SHADER_READ_BIT;
			vimb.subresourceRange = visr;

		vkCmdPipelineBarrier(TextureCommandBuffer,
			VK_PIPELINE_STAGE_TRANSFER_BIT, VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT, 0,
			0, (VkMemoryBarrier *)nullptr,
			0, (VkBufferMemoryBarrier *)nullptr,
			1, IN &vimb);
	}
	// ****************************************************************************

	result = vkEndCommandBuffer( TextureCommandBuffer );

	VkSubmitInfo			vsi;
		vsi.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
		vsi.pNext = nullptr;
		vsi.commandBufferCount = 1;
		vsi.pCommandBuffers = &TextureCommandBuffer;
		vsi.waitSemaphoreCount = 0;
		vsi.pWaitSemaphores = (VkSemaphore *)nullptr;
		vsi.signalSemaphoreCount = 0;
		vsi.pSignalSemaphores = (VkSemaphore *)nullptr;
		vsi.pWaitDstStageMask = (VkPipelineStageFlags *)nullptr;

	result = vkQueueSubmit( Queue, 1, IN &vsi, VK_NULL_HANDLE );
	result = vkQueueWaitIdle( Queue );
```

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

162

```
	// create an image view for the texture image:
	// (an "image view" is used to indirectly access an image)

	VkImageSubresourceRange		visr;
		visr.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
		visr.baseMipLevel = 0;
		visr.levelCount = 1;
		visr.baseArrayLayer = 0;
		visr.layerCount = 1;

	VkImageViewCreateInfo		vivci;
		vivci.sType = VK_STRUCTURE_TYPE_IMAGE_VIEW_CREATE_INFO;
		vivci.pNext = nullptr;
		vivci.flags = 0;
		vivci.image = textureImage;
		vivci.viewType = VK_IMAGE_VIEW_TYPE_2D;
		vivci.format = VK_FORMAT_R8G8B8A8_UNORM;
		vivci.components.r = VK_COMPONENT_SWIZZLE_R;
		vivci.components.g = VK_COMPONENT_SWIZZLE_G;
		vivci.components.b = VK_COMPONENT_SWIZZLE_B;
		vivci.components.a = VK_COMPONENT_SWIZZLE_A;
		vivci.subresourceRange = visr;

	result = vkCreateImageView( LogicalDevice, IN &vivci, PALLOCATOR, OUT &pMyTexture->texImageView);

	return result;
}
```

Note that, at this point, the Staging Buffer is no longer needed, and can be destroyed.

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Reading in a Texture from a BMP File

163

```
typedef struct MyTexture
{
        uint32_t                width;
        uint32_t                height;
        VkImage                 texImage;
        VkImageView             texImageView;
        VkSampler               texSampler;
        VkDeviceMemory          vdm;
} MyTexture;

...

MyTexture    MyPuppyTexture;
```

```
result = Init06TextureBufferAndFillFromBmpFile ( "puppy1.bmp", &MyPuppyTexture);
Init06TextureSampler( &MyPuppyTexture.texSampler );
```

This function can be found in the **sample.cpp** file. The BMP file needs to be created by something that writes uncompressed 24-bit color BMP files, or was converted to the uncompressed BMP format by a tool such as ImageMagick's *convert*, Adobe *Photoshop*, or GNU's *GIMP*.

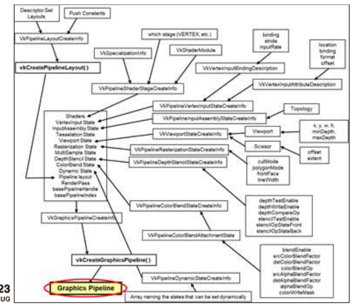
SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

164

Vulkan.

The Graphics Pipeline Data Structure (GPDS)



SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

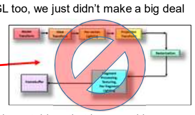
mjb – June 5, 2023

What is the Vulkan Graphics Pipeline Data Structure (GPDS)?

165

Here's what you need to know:

1. The Vulkan Graphics Pipeline is like what OpenGL would call "The State", or "The Context". It is a ***data structure***.
2. Since you know the OpenGL state, a lot of the Vulkan GPDS will seem familiar to you.
3. The current shader program is part of the state. (It was in OpenGL too, we just didn't make a big deal of it.)
4. The Vulkan Graphics Pipeline is *not* the processes that OpenGL would call "the graphics pipeline".



5. For the most part, the Vulkan Graphics Pipeline Data Structure is immutable – that is, once this combination of state variables is combined into a Pipeline, that Pipeline never gets changed. To make new combinations of state variables, create a new GPDS.
6. The shaders get compiled the rest of the way when their Graphics Pipeline Data Structure gets created.

There are also a Vulkan **Compute Pipeline Data Structure** and a **Raytrace Pipeline Data Structure** – we will get to those later.

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Vulkan Graphics Pipeline Stages and what goes into Them

166

The GPU and Driver specify the Pipeline Stages – the Vulkan Graphics Pipeline declares what goes in them

What goes in	Pipeline Stage
Vertex Shader module Specialization info Vertex Input binding Vertex Input attributes	**Vertex Input Stage**
Topology	**Input Assembly**
Tessellation Shaders, Geometry Shader	**Tesselation, Geometry Shaders**
Viewport Scissoring	**Viewport**
Depth Clamping DiscardEnable PolygonMode CullMode FrontFace LineWidth	**Rasterization**
Which states are dynamic	**Dynamic State**
DepthTestEnable DepthWriteEnable DepthCompareOp StencilTestEnable	**Depth/Stencil**
Fragment Shader module Specialization info	**Fragment Shader Stage**
Color Blending parameters	**Color Blending Stage**

SIGGRAPH 2023

mjb – June 5, 2023

The First Step: Create the Graphics Pipeline Layout

167

The Graphics Pipeline Layout is fairly static. Only the layout of the Descriptor Sets and information on the Push Constants need to be supplied.

```
VkPipelineLayout        GraphicsPipelineLayout;        // global
. . .
VkResult
Init14GraphicsPipelineLayout( )
{
        VkResult result;

        VkPipelineLayoutCreateInfo              vplci;
                vplci.sType = VK_STRUCTURE_TYPE_PIPELINE_LAYOUT_CREATE_INFO;
                vplci.pNext = nullptr;
                vplci.flags = 0;
                vplci.setLayoutCount = 4;
                vplci.pSetLayouts = &DescriptorSetLayouts[0];
                vplci.pushConstantRangeCount = 0;
                vplci.pPushConstantRanges = (VkPushConstantRange *)nullptr;

        result = vkCreatePipelineLayout( LogicalDevice, IN &vplci, PALLOCATOR, OUT &GraphicsPipelineLayout );

        return result;
}
```

Let the Pipeline Layout know about the Descriptor Set and Push Constant layouts.

Why is this necessary? It is because the Descriptor Sets and Push Constants data structures have different sizes depending on how many of each you have. So, the exact structure of the Pipeline Layout depends on you telling Vulkan about the Descriptor Sets and Push Constants that you will be using.

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

A Graphics Pipeline Data Structure Contains the Following State Items:

168

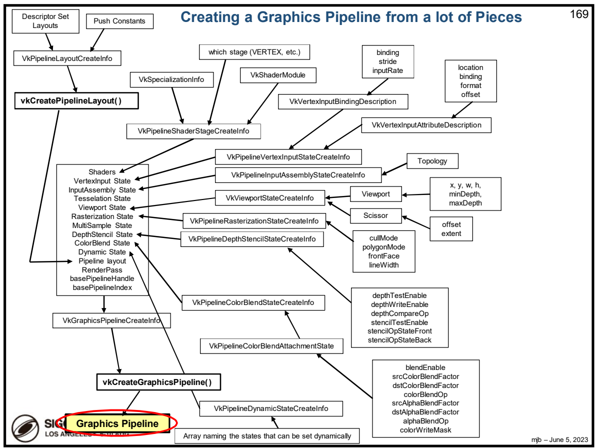
- Pipeline Layout: Descriptor Sets, Push Constants
- Which Shaders to use (half-compiled SPIR-V modules)
- Per-vertex input attributes: location, binding, format, offset
- Per-vertex input bindings: binding, stride, inputRate
- Assembly: topology (e.g., **VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST)**
- ***Viewport***: x, y, w, h, minDepth, maxDepth
- ***Scissoring***: x, y, w, h
- Rasterization: cullMode, polygonMode, frontFace, ***lineWidth***
- Depth: depthTestEnable, depthWriteEnable, depthCompareOp
- Stencil: stencilTestEnable, stencilOpStateFront, stencilOpStateBack
- Blending: blendEnable, ***srcColorBlendFactor, dstColorBlendFactor,*** colorBlendOp, ***srcAlphaBlendFactor, dstAlphaBlendFactor,*** alphaBlendOp, colorWriteMask
- DynamicState: which states can be set dynamically (bound to the command buffer, outside the Pipeline)

Bold/Italics indicates that this state item can be changed with Dynamic State Variables

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Creating a Graphics Pipeline from a lot of Pieces



Creating a Typical Graphics Pipeline


```
VkResult
Init14GraphicsVertexFragmentPipeline( VkShaderModule vertexShader, VkShaderModule fragmentShader,
                                      VkPrimitiveTopology topology, OUT VkPipeline *pGraphicsPipeline )
{
#ifdef ASSUMPTIONS
        vvibd[0].inputRate = VK_VERTEX_INPUT_RATE_VERTEX;
        vprsci.depthClampEnable = VK_FALSE;
        vprsci.rasterizerDiscardEnable = VK_FALSE;
        vprsci.polygonMode = VK_POLYGON_MODE_FILL;
        vprsci.cullMode = VK_CULL_MODE_NONE;    // best to do this because of the projectionMatrix[1][1] *= -1.;
        vprsci.frontFace = VK_FRONT_FACE_COUNTER_CLOCKWISE;
        vpmsci.rasterizationSamples = VK_SAMPLE_COUNT_ONE_BIT;
        vpcbas.blendEnable = VK_FALSE;
        vpcbsci.logicOpEnable = VK_FALSE;
       vpdssci.depthTestEnable = VK_TRUE;
        vpdssci.depthWriteEnable = VK_TRUE;
        vpdssci.depthCompareOp = VK_COMPARE_OP_LESS;
#endif

    . . .
```

These settings seem pretty typical to me. Let's write a simplified Pipeline-creator that accepts Vertex and Fragment shader modules and the topology, and always uses the settings in red above.

The Shaders to Use


```
 VkPipelineShaderStageCreateInfo              vpssci[2];
        vpssci[0].sType = VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_CREATE_INFO;
        vpssci[0].pNext = nullptr;
        vpssci[0].flags = 0;
        vpssci[0].stage = VK_SHADER_STAGE_VERTEX_BIT;
        vpssci[0].module = vertexShader;
        vpssci[0].pName = "main";
        vpssci[0].pSpecializationInfo = (VkSpecializationInfo *)nullptr;
#ifdef BITS
VK_SHADER_STAGE_VERTEX_BIT
VK_SHADER_STAGE_TESSELLATION_CONTROL_BIT
VK_SHADER_STAGE_TESSELLATION_EVALUATION_BIT
VK_SHADER_STAGE_GEOMETRY_BIT
VK_SHADER_STAGE_FRAGMENT_BIT
VK_SHADER_STAGE_COMPUTE_BIT
VK_SHADER_STAGE_ALL_GRAPHICS
VK_SHADER_STAGE_ALL
#endif
        vpssci[1].sType = VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_CREATE_INFO;
        vpssci[1].pNext = nullptr;
        vpssci[1].flags = 0;
        vpssci[1].stage = VK_SHADER_STAGE_FRAGMENT_BIT;
        vpssci[1].module = fragmentShader;
        vpssci[1].pName = "main";
        vpssci[1].pSpecializationInfo = (VkSpecializationInfo *)nullptr;

    VkVertexInputBindingDescription          vvibd[1];   // an array containing one of these per buffer being used
        vvibd[0].binding = 0;        // which binding # this is
        vvibd[0].stride = sizeof( struct vertex );        // bytes between successive
        vvibd[0].inputRate = VK_VERTEX_INPUT_RATE_VERTEX;
#ifdef CHOICES
VK_VERTEX_INPUT_RATE_VERTEX
VK_VERTEX_INPUT_RATE_INSTANCE
#endif
```

Use one **vpssci** array member per shader module you are using

Use one **vvibd** array member per vertex input array-of-structures you are using

Link in the Per-Vertex Attributes


```
 VkVertexInputAttributeDescription        vviad[4];      // an array containing one of these per vertex attribute in all bindings
        // 4 = vertex, normal, color, texture coord
        vviad[0].location = 0;          // location in the layout
        vviad[0].binding = 0;           // which binding description this is part of
        vviad[0].format = VK_FORMAT_VEC3;     // x, y, z
        vviad[0].offset = offsetof( struct vertex, position );          // 0
#ifdef EXTRAS_DEFINED_AT_THE_TOP
// these are here for convenience and readability:
#define VK_FORMAT_VEC4        VK_FORMAT_R32G32B32A32_SFLOAT
#define VK_FORMAT_XYZW        VK_FORMAT_R32G32B32A32_SFLOAT
#define VK_FORMAT_VEC3        VK_FORMAT_R32G32B32_SFLOAT
#define VK_FORMAT_STP         VK_FORMAT_R32G32B32_SFLOAT
#define VK_FORMAT_XYZ         VK_FORMAT_R32G32B32_SFLOAT
#define VK_FORMAT_VEC2        VK_FORMAT_R32G32_SFLOAT
#define VK_FORMAT_ST          VK_FORMAT_R32G32_SFLOAT
#define VK_FORMAT_XY          VK_FORMAT_R32G32_SFLOAT
#define VK_FORMAT_FLOAT       VK_FORMAT_R32_SFLOAT
#define VK_FORMAT_S           VK_FORMAT_R32_SFLOAT
#define VK_FORMAT_X           VK_FORMAT_R32_SFLOAT
#endif
        vviad[1].location = 1;
        vviad[1].binding = 0;
        vviad[1].format = VK_FORMAT_VEC3;     // nx, ny, nz
        vviad[1].offset = offsetof( struct vertex, normal );          // 12

        vviad[2].location = 2;
        vviad[2].binding = 0;
        vviad[2].format = VK_FORMAT_VEC3;     // r, g, b
        vviad[2].offset = offsetof( struct vertex, color );          // 24

        vviad[3].location = 3;
        vviad[3].binding = 0;
        vviad[3].format = VK_FORMAT_VEC2;     // s, t
        vviad[3].offset = offsetof( struct vertex, texCoord );          // 36
```

Use one **vviad** array member per element in the struct for the array-of-structures element you are using as vertex input

I #defined these at the top of the sample code so that you don't need to use confusing image-looking formats for positions, normals, and tex coords


```
 VkPipelineVertexInputStateCreateInfo          vpvisci;          // used to describe the input vertex attributes
        vpvisci.sType = VK_STRUCTURE_TYPE_PIPELINE_VERTEX_INPUT_STATE_CREATE_INFO;
        vpvisci.pNext = nullptr;
        vpvisci.flags = 0;
        vpvisci.vertexBindingDescriptionCount = 1;
        vpvisci.pVertexBindingDescriptions = vvibd;
        vpvisci.vertexAttributeDescriptionCount = 4;
        vpvisci.pVertexAttributeDescriptions = vviad;

    VkPipelineInputAssemblyStateCreateInfo     vpiasci;
        vpiasci.sType = VK_STRUCTURE_TYPE_PIPELINE_INPUT_ASSEMBLY_STATE_CREATE_INFO;
        vpiasci.pNext = nullptr;
        vpiasci.flags = 0;
        vpiasci.topology = VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST;
#ifdef CHOICES
VK_PRIMITIVE_TOPOLOGY_POINT_LIST
VK_PRIMITIVE_TOPOLOGY_LINE_LIST
VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST
VK_PRIMITIVE_TOPOLOGY_LINE_STRIP
VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP
VK_PRIMITIVE_TOPOLOGY_TRIANGLE_FAN
VK_PRIMITIVE_TOPOLOGY_LINE_LIST_WITH_ADJACENCY
VK_PRIMITIVE_TOPOLOGY_LINE_STRIP_WITH_ADJACENCY
VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST_WITH_ADJACENCY
VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP_WITH_ADJACENCY
#endif
        vpiasci.primitiveRestartEnable = VK_FALSE;

    VkPipelineTessellationStateCreateInfo       vptsci;
        vptsci.sType = VK_STRUCTURE_TYPE_PIPELINE_TESSELLATION_STATE_CREATE_INFO;
        vptsci.pNext = nullptr;
        vptsci.flags = 0;
        vptsci.patchControlPoints = 0;      // number of patch control points

    VkPipelineGeometryStateCreateInfo          vpgsci;
        vptsci.sType = VK_STRUCTURE_TYPE_PIPELINE_TESSELLATION_STATE_CREATE_INFO;
        vptsci.pNext = nullptr;
        vptsci.flags = 0;
```

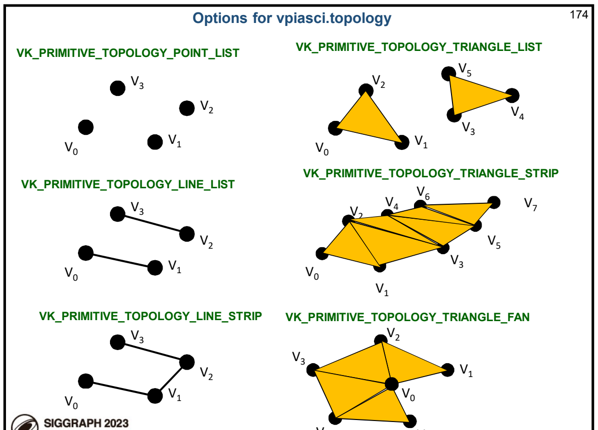
Declare the binding descriptions and attribute descriptions

Declare the vertex topology

Tessellation Shader info

Geometry Shader info

Options for vpiasci.topology



What is "Primitive Restart Enable"?

175

```
vpiasci.primitiveRestartEnable = VK_FALSE;
```

"Restart Enable" is used with:
- Indexed drawing.
- TRIANGLE_FAN and TRIANGLE_STRIP topologies

If vpiasci.primitiveRestartEnable is VK_TRUE, then a special "index" can be used to indicate that the primitive should start over. This is more efficient than explicitly ending the current triangle strip and explicitly starting a new one.

```
typedef enum VkIndexType
{
    VK_INDEX_TYPE_UINT16 = 0,      // 0 –           65,535
    VK_INDEX_TYPE_UINT32 = 1,      // 0 – 4,294,967,295
} VkIndexType;
```

If your VkIndexType is VK_INDEX_TYPE_UINT16, then the special index is **0xffff**.
If your VkIndexType is VK_INDEX_TYPE_UINT32, then the special index is **0xffffffff**.

That is, a one in all available bits

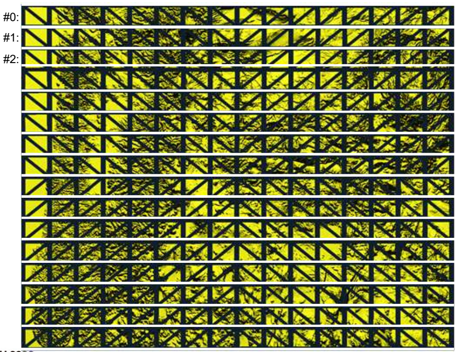
SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

One Really Good use of Indexed Drawing and Restart Enable is in Drawing Terrain Surfaces with Triangle Strips

176

Triangle Strip #0:
Triangle Strip #1:
Triangle Strip #2:
. . .



SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

177

```
VkViewport                                  vv;
        vv.x = 0;
        vv.y = 0;
        vv.width = (float)Width;
        vv.height = (float)Height;
        vv.minDepth = 0.0f;
        vv.maxDepth = 1.0f;

VkRect2D                                    vr;
        vr.offset.x = 0;
        vr.offset.y = 0;
        vr.extent.width = Width;
        vr.extent.height = Height;

VkPipelineViewportStateCreateInfo           vpvsci;
        vpvsci.sType = VK_STRUCTURE_TYPE_PIPELINE_VIEWPORT_STATE_CREATE_INFO;
        vpvsci.pNext = nullptr;
        vpvsci.flags = 0;
        vpvsci.viewportCount = 1;
        vpvsci.pViewports = &vv;
        vpvsci.scissorCount = 1;
        vpvsci.pScissors = &vr;
```

Declare the viewport information

Declare the scissoring information

Group the viewport and scissor information together

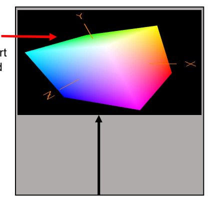
SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

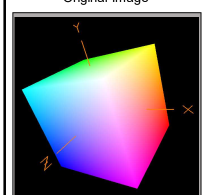
What is the Difference Between Changing the Viewport and Changing the Scissoring?

178

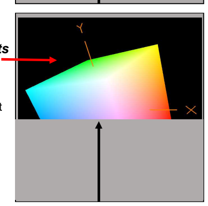
Viewport:
Viewporting operates on ***vertices*** and takes place right ***before*** the rasterizer. Changing the vertical part of the **viewport** causes the entire scene to get scaled (scrunched) into the viewport area.



Original Image



Scissoring:
Scissoring operates on ***fragments*** and takes place right ***after*** the rasterizer. Changing the vertical part of the **scissor** causes the entire scene to get clipped where it falls outside the scissor area.

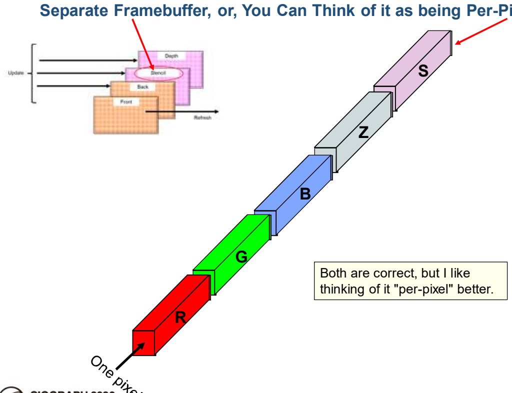


SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

You Can Think of the Stencil Buffer as a Separate Framebuffer, or, You Can Think of it as being Per-Pixel

179



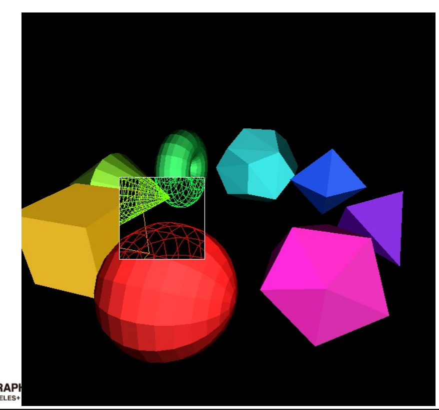
Both are correct, but I like thinking of it "per-pixel" better.

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Using the Stencil Buffer to Create a *Magic Lens*

180

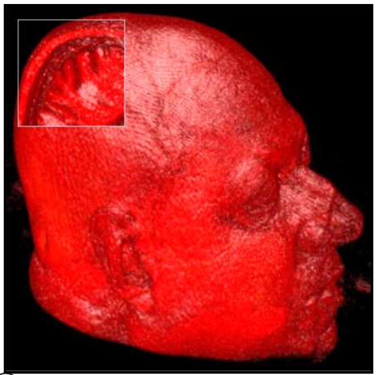


SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

I Once Used the Stencil Buffer to Create a *Magic Lens* for Volume Data

181

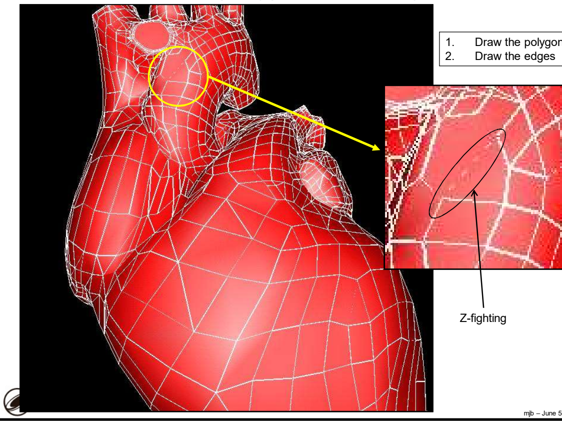


In this case, the scene inside the lens was created by drawing the same object, but drawing it with its near clipping plane being farther away from the eye position

Outlining Polygons the Naïve Way

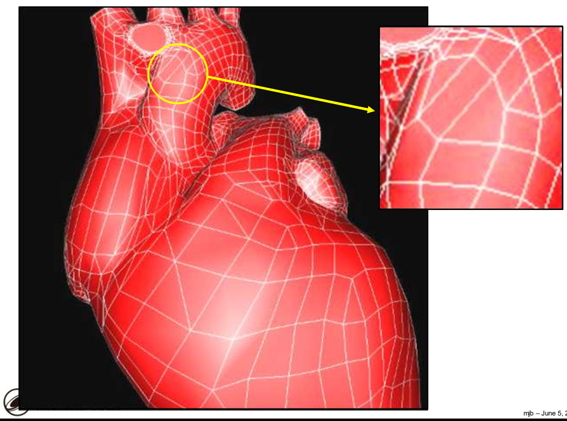
182

1. Draw the polygons
2. Draw the edges



Using the Stencil Buffer to Better Outline Polygons

183



Stencil Operations for Front and Back Faces

184

```
     VkStencilOpState                              vsosf;     // front
          vsosf.depthFailOp = VK_STENCIL_OP_KEEP;  // what to do if depth operation fails
          vsosf.failOp      = VK_STENCIL_OP_KEEP;  // what to do if stencil operation fails
          vsosf.passOp      = VK_STENCIL_OP_KEEP;  // what to do if stencil operation succeeds
#ifdef CHOICES
VK_STENCIL_OP_KEEP                        -- keep the stencil value as it is
VK_STENCIL_OP_ZERO                        -- set stencil value to 0
VK_STENCIL_OP_REPLACE                     -- replace stencil value with the reference value
VK_STENCIL_OP_INCREMENT_AND_CLAMP         -- increment stencil value
VK_STENCIL_OP_DECREMENT_AND_CLAMP         -- decrement stencil value
VK_STENCIL_OP_INVERT                      -- bit-invert stencil value
VK_STENCIL_OP_INCREMENT_AND_WRAP          -- increment stencil value
VK_STENCIL_OP_DECREMENT_AND_WRAP          -- decrement stencil value
#endif
          vsosf.compareOp = VK_COMPARE_OP_NEVER;
#ifdef CHOICES
VK_COMPARE_OP_NEVER                       -- never succeeds
VK_COMPARE_OP_LESS                        -- succeeds if stencil value is <  the reference value
VK_COMPARE_OP_EQUAL                       -- succeeds if stencil value is == the reference value
VK_COMPARE_OP_LESS_OR_EQUAL               -- succeeds if stencil value is <= the reference value
VK_COMPARE_OP_GREATER                     -- succeeds if stencil value is >  the reference value
VK_COMPARE_OP_NOT_EQUAL                   -- succeeds if stencil value is != the reference value
VK_COMPARE_OP_GREATER_OR_EQUAL            -- succeeds if stencil value is >= the reference value
VK_COMPARE_OP_ALWAYS                      -- always succeeds
#endif
          vsosf.compareMask = ~0;
          vsosf.writeMask = ~0;
          vsosf.reference = 0;

     VkStencilOpState                              vsosb;     // back
          vsosb.depthFailOp = VK_STENCIL_OP_KEEP;
          vsosb.failOp      = VK_STENCIL_OP_KEEP;
          vsosb.passOp      = VK_STENCIL_OP_KEEP;
          vsosb.compareOp = VK_COMPARE_OP_NEVER;
          vsosb.compareMask = ~0;
          vsosb.writeMask = ~0;
          vsosb.reference = 0;
```


Operations for Depth Values

185

```
     VkPipelineDepthStencilStateCreateInfo          vpdssci;
          vpdssci.sType = VK_STRUCTURE_TYPE_PIPELINE_DEPTH_STENCIL_STATE_CREATE_INFO;
          vpdssci.pNext = nullptr;
          vpdssci.flags = 0;
          vpdssci.depthTestEnable  = VK_TRUE;
          vpdssci.depthWriteEnable = VK_TRUE;
          vpdssci.depthCompareOp = VK_COMPARE_OP_LESS;
VK_COMPARE_OP_NEVER                  -- never succeeds
VK_COMPARE_OP_LESS                   -- succeeds if new depth value is <  the existing value
VK_COMPARE_OP_EQUAL                  -- succeeds if new depth value is == the existing value
VK_COMPARE_OP_LESS_OR_EQUAL          -- succeeds if new depth value is <= the existing value
VK_COMPARE_OP_GREATER                -- succeeds if new depth value is >  the existing value
VK_COMPARE_OP_NOT_EQUAL              -- succeeds if new depth value is != the existing value
VK_COMPARE_OP_GREATER_OR_EQUAL       -- succeeds if new depth value is >= the existing value
VK_COMPARE_OP_ALWAYS                 -- always succeeds
#endif
          vpdssci.depthBoundsTestEnable = VK_FALSE;
          vpdssci.front  = vsosf;
          vpdssci.back  = vsosb;
          vpdssci.minDepthBounds  = 0.;
          vpdssci.maxDepthBounds = 1.;
          vpdssci.stencilTestEnable = VK_FALSE;
```


Putting it all Together! (finally…)

186

```
VkPipeline   GraphicsPipeline;          // global
. . .

     VkGraphicsPipelineCreateInfo                    vgpci;
          vgpci.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
          vgpci.pNext = nullptr;
          vgpci.flags = 0;
#ifdef CHOICES
VK_PIPELINE_CREATE_DISABLE_OPTIMIZATION_BIT
VK_PIPELINE_CREATE_ALLOW_DERIVATIVES_BIT
VK_PIPELINE_CREATE_DERIVATIVE_BIT
#endif
          vgpci.stageCount = 2;                 // number of stages in this pipeline
          vgpci.pStages = vpssci;
          vgpci.pVertexInputState = &vpvisci;
          vgpci.pInputAssemblyState = &vpiasci;
          vgpci.pTessellationState = (VkPipelineTessellationStateCreateInfo *)nullptr;
          vgpci.pViewportState = &vpvsci;
          vgpci.pRasterizationState = &vprsci;
          vgpci.pMultisampleState = &vpmsci;
          vgpci.pDepthStencilState = &vpdssci;
          vgpci.pColorBlendState = &vpcbsci;
          vgpci.pDynamicState = &vpdsci;
          vgpci.layout = IN GraphicsPipelineLayout;
          vgpci.renderPass = IN RenderPass;
          vgpci.subpass = 0;                    // subpass number
          vgpci.basePipelineHandle = (VkPipeline) VK_NULL_HANDLE;
          vgpci.basePipelineIndex = 0;

     result = vkCreateGraphicsPipelines( LogicalDevice, VK_NULL_HANDLE, 1, IN &vgpci,
                                         PALLOCATOR, OUT &GraphicsPipeline );

     return result;
}
```

Group all of the individual state information and create the pipeline

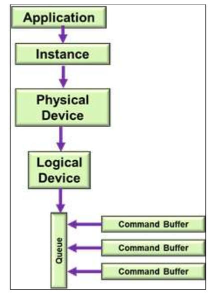
When Drawing, We will Bind a Specific Graphics Pipeline Data Structure to the Command Buffer

187

```
VkPipeline        GraphicsPipeline;              // global
. . .
vkCmdBindPipeline( CommandBuffers[nextImageIndex],
                       VK_PIPELINE_BIND_POINT_GRAPHICS, GraphicsPipeline );
```

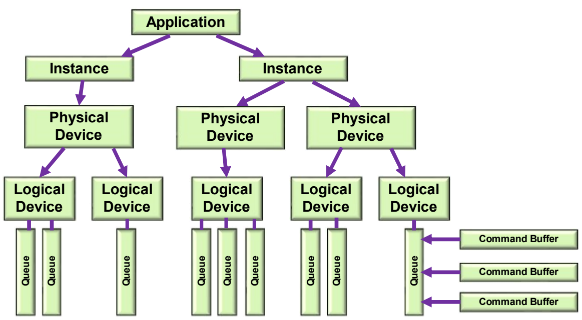

188

Queues and Command Buffers



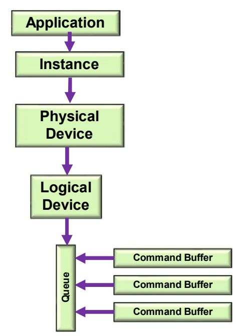
Vulkan: Overall Block Diagram

189



Simplified Block Diagram

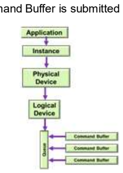
190

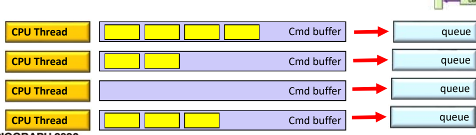


Vulkan Queues and Command Buffers

191

- Graphics commands are recorded in command buffers, e.g., *vkCmdDoSomething(cmdBuffer, …);*
- You can have as many simultaneous Command Buffers as you want
- Each command buffer can be filled from a different thread, but doesn't have to be
- Command Buffers record commands, but no work takes place until a Command Buffer is submitted to a Queue
- We don't create Queues – the Logical Device already has them
- Each Queue belongs to a Queue Family
- We don't create Queue Families – the Physical Device already has them





Querying what Queue Families are Available

192

```
uint32_t  count;
vkGetPhysicalDeviceQueueFamilyProperties( IN PhysicalDevice, &count, OUT (VkQueueFamilyProperties *) nullptr );

VkQueueFamilyProperties *vqfp = new VkQueueFamilyProperties[ count ];
vkGetPhysicalDeviceFamilyProperties(  PhysicalDevice, &count, OUT &vqfp, );

for( unsigned int i = 0; i < count; i++ )
{
        fprintf( FpDebug, "\t%d: Queue Family Count = %2d  ;   ", i, vqfp[i].queueCount );
        if( ( vqfp[i].queueFlags & VK_QUEUE_GRAPHICS_BIT ) != 0 )      fprintf( FpDebug, " Graphics" );
        if( ( vqfp[i].queueFlags & VK_QUEUE_COMPUTE_BIT  ) != 0 )      fprintf( FpDebug, " Compute " );
        if( ( vqfp[i].queueFlags & VK_QUEUE_TRANSFER_BIT ) != 0 )      fprintf( FpDebug, " Transfer" );
        fprintf(FpDebug, "\n");
}
```

For the Nvidia A6000 cards:

```
Found 3 Queue Families:
      0: Queue Family Count = 16  ;    Graphics Compute Transfer
      1: Queue Family Count =   2  ;    Transfer
      2: Queue Family Count =   8  ;    Compute Transfer
```

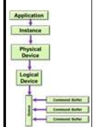
Similarly, we Can Write a Function that Finds the Proper Queue Family

193

```
int
FindQueueFamilyThatDoesGraphics( )
{
        uint32_t count = -1;
        vkGetPhysicalDeviceQueueFamilyProperties( IN PhysicalDevice, OUT &count, OUT (VkQueueFamilyProperties *)nullptr );

        VkQueueFamilyProperties *vqfp = new VkQueueFamilyProperties[ count ];
        vkGetPhysicalDeviceQueueFamilyProperties( IN PhysicalDevice, IN &count, OUT vqfp );

        for( unsigned int i = 0; i < count; i++ )
        {
                if( ( vqfp[ i ].queueFlags & VK_QUEUE_GRAPHICS_BIT ) != 0 )
                        return i;
        }
        return -1;
}
```


Creating a Logical Device Needs to Know Queue Family Information

194




```
float   queuePriorities[ ] =
{
                1.              // one entry per queueCount
};

VkDeviceQueueCreateInfo   vdqci[1];
        vdqci[0].sType = VK_STRUCTURE_TYPE_QUEUE_CREATE_INFO;
        vdqci[0].pNext = nullptr;
        vdqci[0].flags = 0;
        vdqci[0].queueFamilyIndex = FindQueueFamilyThatDoesGraphics( );
        vdqci[0].queueCount = 1;
        vdqci[0].queuePriorities = (float *) queuePriorities;

VkDeviceCreateInfo   vdci;
                vdci.sType = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
                vdci.pNext = nullptr;
                vdci.flags = 0;
                vdci.queueCreateInfoCount = 1;                  // # of device queues wanted
                vdci.pQueueCreateInfos = IN &vdqci[0];          // array of VkDeviceQueueCreateInfo's
                vdci.enabledLayerCount = sizeof(myDeviceLayers) / sizeof(char *);
                vdci.ppEnabledLayerNames = myDeviceLayers;
                vdci.enabledExtensionCount = sizeof(myDeviceExtensions) / sizeof(char *);
                vdci.ppEnabledExtensionNames = myDeviceExtensions;
                vdci.pEnabledFeatures = IN &PhysicalDeviceFeatures;     // already created

result = vkCreateLogicalDevice( PhysicalDevice, IN &vdci, PALLOCATOR, OUT &LogicalDevice );

VkQueue Queue;
uint32_t  queueFamilyIndex = FindQueueFamilyThatDoesGraphics( );
uint32_t  queueIndex = 0;

result = vkGetDeviceQueue ( LogicalDevice, queueFamilyIndex, queueIndex,   OUT &Queue );
```

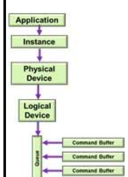

Creating the Command Pool as part of the Logical Device

195

```
VkResult
Init06CommandPool( )
{
        VkResult result;

        VkCommandPoolCreateInfo                 vcpci;
                vcpci.sType = VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO;
                vcpci.pNext = nullptr;
                vcpci.flags =        VK_COMMAND_POOL_CREATE_RESET_COMMAND_BUFFER_BIT
                                   | VK_COMMAND_POOL_CREATE_TRANSIENT_BIT;
#ifdef CHOICES
VK_COMMAND_POOL_CREATE_TRANSIENT_BIT
VK_COMMAND_POOL_CREATE_RESET_COMMAND_BUFFER_BIT
#endif
                vcpci.queueFamilyIndex = FindQueueFamilyThatDoesGraphics( );

        result = vkCreateCommandPool( LogicalDevice, IN &vcpci, PALLOCATOR, OUT &CommandPool );

        return result;
}
```


Creating the Command Buffers

196




```
VkResult
Init06CommandBuffers( )
{
        VkResult result;

        // allocate 2 command buffers for the double-buffered rendering:

        {
                VkCommandBufferAllocateInfo             vcbai;
                        vcbai.sType = VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO;
                        vcbai.pNext = nullptr;
                        vcbai.commandPool = CommandPool;
                        vcbai.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
                        vcbai.commandBufferCount = 2;           // 2, because of double-buffering

                result = vkAllocateCommandBuffers( LogicalDevice, IN &vcbai, OUT &CommandBuffers[0] );
        }


        // allocate 1 command buffer for the transferring pixels from a staging buffer to a texture buffer:

        {
                VkCommandBufferAllocateInfo             vcbai;
                        vcbai.sType = VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO;
                        vcbai.pNext = nullptr;
                        vcbai.commandPool = CommandPool;
                        vcbai.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
                        vcbai.commandBufferCount = 1;

                result = vkAllocateCommandBuffers( LogicalDevice, IN &vcbai, OUT &TextureCommandBuffer );
        }

        return result;
}
```

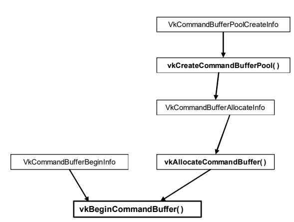

Beginning a Command Buffer – One per Image

197

```
VkSemaphoreCreateInfo                   vsci;
        vsci.sType = VK_STRUCTURE_TYPE_SEMAPHORE_CREATE_INFO;
        vsci.pNext = nullptr;
        vsci.flags = 0;

VkSemaphore  imageReadySemaphore;
result = vkCreateSemaphore( LogicalDevice, IN &vsci, PALLOCATOR, OUT &imageReadySemaphore );

uint32_t  nextImageIndex;
vkAcquireNextImageKHR( LogicalDevice, IN SwapChain, IN UINT64_MAX,
                                IN imageReadySemaphore, IN VK_NULL_HANDLE, OUT &nextImageIndex );


VkCommandBufferBeginInfo            vcbbi;
            vcbbi.sType = VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO;
            vcbbi.pNext = nullptr;
            vcbbi.flags = VK_COMMAND_BUFFER_USAGE_ONE_TIME_SUBMIT_BIT;
            vcbbi.pInheritanceInfo = (VkCommandBufferInheritanceInfo *)nullptr;

result = vkBeginCommandBuffer( CommandBuffers[nextImageIndex], IN &vcbbi );

        . . .

vkEndCommandBuffer( CommandBuffers[nextImageIndex] );
```


Beginning a Command Buffer

198



These are the Commands that could be entered into a Command Buffer, I

vkCmdBeginConditionalRendering
vkCmdBeginDebugUtilsLabel
vkCmdBeginQuery
vkCmdBeginQueryIndexed
vkCmdBeginRendering
vkCmdBeginRenderPass
vkCmdBeginRenderPass2
vkCmdBeginTransformFeedback
vkCmdBindDescriptorSets
vkCmdBindIndexBuffer
vkCmdBindInvocationMask
vkCmdBindPipeline
vkCmdBindPipelineShaderGroup
vkCmdBindShadingRateImage
vkCmdBindTransformFeedbackBuffers
vkCmdBindVertexBuffers
vkCmdBindVertexBuffers2
vkCmdBlitImage

vkCmdBlitImage2
vkCmdBuildAccelerationStructure
vkCmdBuildAccelerationStructuresIndirect
vkCmdBuildAccelerationStructures
vkCmdClearAttachments
vkCmdClearColorImage
vkCmdClearDepthStencilImage
vkCmdCopyAccelerationStructure
vkCmdCopyAccelerationStructureToMemory
vkCmdCopyBuffer
vkCmdCopyBuffer2
vkCmdCopyBufferToImage
vkCmdCopyBufferToImage2
vkCmdCopyImage
vkCmdCopyImage2
vkCmdCopyImageToBuffer
vkCmdCopyImageToBuffer2
vkCmdCopyMemoryToAccelerationStructure

These are the Commands that could be entered into a Command Buffer, II

vkCmdCopyQueryPoolResults
vkCmdCuLaunchKernelX
vkCmdDebugMarkerBegin
vkCmdDebugMarkerEnd
vkCmdDebugMarkerInsert
vkCmdDispatch
vkCmdDispatchBase
vkCmdDispatchIndirect
vkCmdDraw
vkCmdDrawIndexed
vkCmdDrawIndexedIndirect
vkCmdDrawIndexedIndirectCount
vkCmdDrawIndirect
vkCmdDrawIndirectByteCount
vkCmdDrawIndirectCount
vkCmdDrawMeshTasksIndirectCount
vkCmdDrawMeshTasksIndirect
vkCmdDrawMeshTasks

vkCmdDrawMulti
vkCmdDrawMultiIndexed
vkCmdEndConditionalRendering
vkCmdEndDebugUtilsLabel
vkCmdEndQuery
vkCmdEndQueryIndexed
vkCmdEndRendering
vkCmdEndRenderPass
vkCmdEndRenderPass2
vkCmdEndTransformFeedback
vkCmdExecuteCommands
vkCmdExecuteGeneratedCommands
vkCmdFillBuffer
vkCmdInsertDebugUtilsLabel
vkCmdNextSubpass
vkCmdNextSubpass2
vkCmdPipelineBarrier
vkCmdPipelineBarrier2

These are the Commands that could be entered into a Command Buffer, III

vkCmdPreprocessGeneratedCommands
vkCmdPushConstants
vkCmdPushDescriptorSet
vkCmdPushDescriptorSetWithTemplate
vkCmdResetEvent
vkCmdResetEvent2
vkCmdResetQueryPool
vkCmdResolveImage
vkCmdResolveImage2
vkCmdSetBlendConstants
vkCmdSetCheckpoint
vkCmdSetCoarseSampleOrder
vkCmdSetCullMode
vkCmdSetDepthBias
vkCmdSetDepthBiasEnable
vkCmdSetDepthBounds
vkCmdSetDepthBoundsTestEnable
vkCmdSetDepthCompareOp

vkCmdSetDepthTestEnable
vkCmdSetDepthWriteEnable
vkCmdSetDeviceMask
vkCmdSetDiscardRectangle
vkCmdSetEvent
vkCmdSetEvent2
vkCmdSetExclusiveScissor
vkCmdSetFragmentShadingRateEnum
vkCmdSetFragmentShadingRate
vkCmdSetFrontFace
vkCmdSetLineStipple
vkCmdSetLineWidth
vkCmdSetLogicOp
vkCmdSetPatchControlPoints
vkCmdSetPrimitiveRestartEnable
vkCmdSetPrimitiveTopology
vkCmdSetRasterizerDiscardEnable
vkCmdSetRayTracingPipelineStackSize

These are the Commands that could be entered into a Command Buffer, IV

vkCmdSetSampleLocations
vkCmdSetScissor
vkCmdSetScissorWithCount
vkCmdSetStencilCompareMask
vkCmdSetStencilOp
vkCmdSetStencilReference
vkCmdSetStencilTestEnable
vkCmdSetStencilWriteMask
vkCmdSetVertexInput
vkCmdSetViewport
vkCmdSetViewportShadingRatePalette
vkCmdSetViewportWithCount
vkCmdSetViewportWScaling

vkCmdSubpassShading
vkCmdTraceRaysIndirect2
vkCmdTraceRaysIndirect
vkCmdTraceRays
vkCmdUpdateBuffer
vkCmdWaitEvents
vkCmdWaitEvents2
vkCmdWriteAccelerationStructuresProperties
vkCmdWriteBufferMarker2
vkCmdWriteBufferMarker
vkCmdWriteTimestamp
vkCmdWriteTimestamp2

How the *RenderScene( )* Function Works


```
VkResult
RenderScene( )
{
    VkResult result;
    VkSemaphoreCreateInfo           vsci;
      vsci.sType = VK_STRUCTURE_TYPE_SEMAPHORE_CREATE_INFO;
      vsci.pNext = nullptr;
      vsci.flags = 0;

    VkSemaphore  imageReadySemaphore;
    result = vkCreateSemaphore( LogicalDevice, IN &vsci, PALLOCATOR, OUT &imageReadySemaphore );

    uint32_t nextImageIndex;
    vkAcquireNextImageKHR( LogicalDevice,   IN SwapChain, IN UINT64_MAX, IN VK_NULL_HANDLE,
                                   IN VK_NULL_HANDLE, OUT &nextImageIndex );

    VkCommandBufferBeginInfo        vcbbi;
        vcbbi.sType = VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO;
        vcbbi.pNext = nullptr;
        vcbbi.flags = VK_COMMAND_BUFFER_USAGE_ONE_TIME_SUBMIT_BIT;
        vcbbi.pInheritanceInfo = (VkCommandBufferInheritanceInfo *)nullptr;

    result = vkBeginCommandBuffer( CommandBuffers[nextImageIndex], IN &vcbbi );
```



```
    VkClearColorValue              vccv;
        vccv.float32[0] = 0.0;
        vccv.float32[1] = 0.0;
        vccv.float32[2] = 0.0;
        vccv.float32[3] = 1.0;

    VkClearDepthStencilValue       vcdsv;
        vcdsv.depth = 1.f;
        vcdsv.stencil = 0;

    VkClearValue               vcv[2];
        vcv[0].color = vccv;
        vcv[1].depthStencil = vcdsv;

    VkOffset2D o2d = { 0, 0 };
    VkExtent2D e2d = { Width, Height };
    VkRect2D r2d = { o2d, e2d };

    VkRenderPassBeginInfo              vrpbi;
        vrpbi.sType = VK_STRUCTURE_TYPE_RENDER_PASS_BEGIN_INFO;
        vrpbi.pNext = nullptr;
        vrpbi.renderPass = RenderPass;
        vrpbi.framebuffer = Framebuffers[ nextImageIndex ];
        vrpbi.renderArea = r2d;
        vrpbi.clearValueCount = 2;
        vrpbi.pClearValues = vcv;          // used for VK_ATTACHMENT_LOAD_OP_CLEAR

    vkCmdBeginRenderPass( CommandBuffers[nextImageIndex], IN &vrpbi, IN VK_SUBPASS_CONTENTS_INLINE );
```

205

```
VkViewport viewport =
{
      0.,                  // x
      0.,                  // y
      (float)Width,
      (float)Height,
      0.,                  // minDepth
      1.                   // maxDepth
};

vkCmdSetViewport( CommandBuffers[nextImageIndex], 0, 1, IN &viewport );      // 0=firstViewport, 1=viewportCount

VkRect2D scissor =
{
      0,
      0,
      Width,
      Height
};

vkCmdSetScissor( CommandBuffers[nextImageIndex], 0, 1, IN &scissor );

vkCmdBindDescriptorSets( CommandBuffers[nextImageIndex], VK_PIPELINE_BIND_POINT_GRAPHICS,
                         GraphicsPipelineLayout, 0, 4, DescriptorSets, 0, (uint32_t *)nullptr );
                                                  // dynamic offset count, dynamic offsets
vkCmdBindPushConstants( CommandBuffers[nextImageIndex], PipelineLayout, VK_SHADER_STAGE_ALL, offset, size, void *values );

VkBuffer buffers[1] = { MyVertexDataBuffer.buffer };

VkDeviceSize offsets[1] = { 0 };

vkCmdBindVertexBuffers( CommandBuffers[nextImageIndex], 0, 1, buffers, offsets );      // 0, 1 = firstBinding, bindingCount

const uint32_t vertexCount = sizeof(VertexData) / sizeof(VertexData[0]);
const uint32_t instanceCount = 1;
const uint32_t firstVertex = 0;
const uint32_t firstInstance = 0;
vkCmdDraw( CommandBuffers[nextImageIndex], vertexCount, instanceCount, firstVertex, firstInstance );

vkCmdEndRenderPass( CommandBuffers[nextImageIndex] );

vkEndCommandBuffer( CommandBuffers[nextImageIndex] );
```

5, 2023

206

Submitting a Command Buffer to a Queue for Execution

```
VkSubmitInfo                    vsi;
      vsi.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
      vsi.pNext = nullptr;
      vsi.commandBufferCount = 1;
      vsi.pCommandBuffers = &CommandBuffer;
      vsi.waitSemaphoreCount = 1;
      vsi.pWaitSemaphores = imageReadySemaphore;
      vsi.signalSemaphoreCount = 0;
      vsi.pSignalSemaphores = (VkSemaphore *)nullptr;
      vsi.pWaitDstStageMask = (VkPipelineStageFlags *)nullptr;
```

SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

207

The Entire Submission / Wait / Display Process

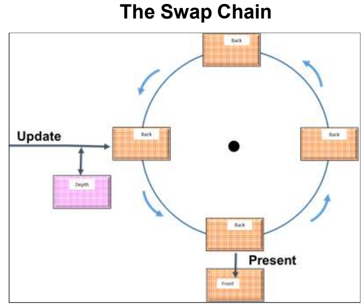
```
VkFenceCreateInfo               vfci;
      vfci.sType = VK_STRUCTURE_TYPE_FENCE_CREATE_INFO;
      vfci.pNext = nullptr;
      vfci.flags = 0;

VkFence  renderFence;
vkCreateFence( LogicalDevice, IN &vfci, PALLOCATOR, OUT &renderFence );
result = VK_SUCCESS;

VkPipelineStageFlags waitAtBottom = VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT;
VkQueue presentQueue;
vkGetDeviceQueue( LogicalDevice, FindQueueFamilyThatDoesGraphics( ), 0, OUT &presentQueue );\
                                     // 0 =, queueIndex

VkSubmitInfo                    vsi;
      vsi.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
      vsi.pNext = nullptr;
      vsi.waitSemaphoreCount = 1;
      vsi.pWaitSemaphores = &imageReadySemaphore;
      vsi.pWaitDstStageMask = &waitAtBottom;
      vsi.commandBufferCount = 1;
      vsi.pCommandBuffers = &CommandBuffers[nextImageIndex];
      vsi.signalSemaphoreCount = 0;
      vsi.pSignalSemaphores = &SemaphoreRenderFinished;

result = vkQueueSubmit( presentQueue,   1, IN &vsi, IN renderFence );     // 1 = submitCount
result = vkWaitForFences( LogicalDevice, 1, IN &renderFence, VK_TRUE, UINT64_MAX );     // waitAll, timeout

vkDestroyFence( LogicalDevice, renderFence, PALLOCATOR );

VkPresentInfoKHR                      vpi;
      vpi.sType = VK_STRUCTURE_TYPE_PRESENT_INFO_KHR;
      vpi.pNext = nullptr;
      vpi.waitSemaphoreCount = 0;
      vpi.pWaitSemaphores = (VkSemaphore *)nullptr;
      vpi.swapchainCount = 1;
      vpi.pSwapchains = &SwapChain;
      vpi.pImageIndices = &nextImageIndex;
      vpi.pResults = (VkResult *)nullptr;

result = vkQueuePresentKHR( presentQueue, IN &vpi );
```

June 5, 2023

208

Vulkan.

The Swap Chain

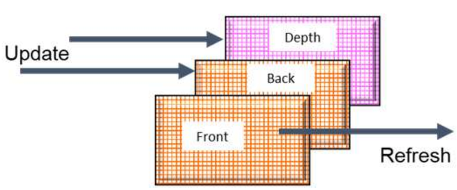


SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

209

How OpenGL Thinks of Framebuffers

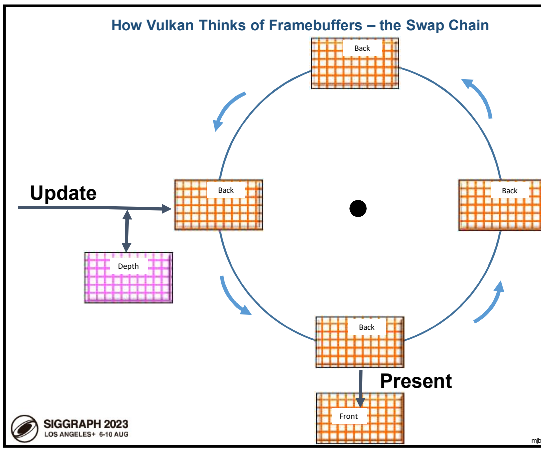


SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

210

How Vulkan Thinks of Framebuffers – the Swap Chain



SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

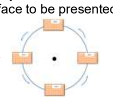
mjb – June 5, 2023

What is a Swap Chain?

Vulkan does not use the idea of a “back buffer”. So, we need a place to render into before moving an image into place for viewing. The is called the **Swap Chain**.

In essence, the Swap Chain manages one or more image objects that form a sequence of images that can be drawn into and then given to the Surface to be presented to the user for viewing.

Swap Chains are arranged as a ring buffer



Swap Chains are tightly coupled to the window system.

After creating the Swap Chain in the first place, the process for using the Swap Chain is:

1. Ask the Swap Chain for an image
2. Render into it via the Command Buffer and a Queue
3. Return the image to the Swap Chain for presentation
4. Present the image to the viewer (copy to “front buffer”)

We Need to Find Out What our Display Capabilities Are


```
VkSurfaceCapabilitiesKHR          vsc;
vkGetPhysicalDeviceSurfaceCapabilitiesKHR( PhysicalDevice, Surface, OUT &vsc );
VkExtent2D surfaceRes = vsc.currentExtent;
fprintf( FpDebug, "\nvkGetPhysicalDeviceSurfaceCapabilitiesKHR:\n" );

        . . .

VkBool32  supported;
result = vkGetPhysicalDeviceSurfaceSupportKHR( PhysicalDevice, FindQueueFamilyThatDoesGraphics( ), Surface, &supported );
if( supported == VK_TRUE )
        fprintf( FpDebug, "** This Surface is supported by the Graphics Queue **\n" );


uint32_t formatCount;
vkGetPhysicalDeviceSurfaceFormatsKHR( PhysicalDevice, Surface, &formatCount, (VkSurfaceFormatKHR *) nullptr );
VkSurfaceFormatKHR * surfaceFormats = new VkSurfaceFormatKHR[ formatCount ];
vkGetPhysicalDeviceSurfaceFormatsKHR( PhysicalDevice, Surface, &formatCount, surfaceFormats );
fprintf( FpDebug, "\nFound %d Surface Formats:\n", formatCount )

        . . .

uint32_t presentModeCount;
vkGetPhysicalDeviceSurfacePresentModesKHR( PhysicalDevice, Surface, &presentModeCount, (VkPresentModeKHR *) nullptr );
VkPresentModeKHR * presentModes = new VkPresentModeKHR[ presentModeCount ];
vkGetPhysicalDeviceSurfacePresentModesKHR( PhysicalDevice, Surface, &presentModeCount, presentModes );
fprintf( FpDebug, "\nFound %d Present Modes:\n", presentModeCount );

        . . .
```


We Need to Find Out What our Display Capabilities Are

VulkanDebug.txt output for an Nvidia A6000:

```
***** Init08Swapchain *****

vkGetPhysicalDeviceSurfaceCapabilitiesKHR:

        minImageCount = 2 ; maxImageCount = 8
        currentExtent = 1024 x 1024
        minImageExtent = 1024 x 1024
        maxImageExtent = 1024 x 1024
        maxImageArrayLayers = 1
        supportedTransforms = 0x0001
        currentTransform = 0x0001
        supportedCompositeAlpha = 0x0001
        supportedUsageFlags = 0x009f

vkGetPhysicalDeviceSurfaceSupportKHR:

** This Surface is supported by the Graphics Queue **

Found 3 Surface Formats:
 0:    44        0  VK_COLOR_SPACE_SRGB_NONLINEAR_KHR
 1:    50        0  VK_COLOR_SPACE_SRGB_NONLINEAR_KHR
 2:    64        0  VK_COLOR_SPACE_SRGB_NONLINEAR_KHR

Found 4 Present Modes:
 0:     2  VK_PRESENT_MODE_FIFO_KHR
 1:     3  VK_PRESENT_MODE_FIFO_RELAXED_KHR
 2:     1  VK_PRESENT_MODE_MAILBOX_KHR
 3:     0  VK_PRESENT_MODE_IMMEDIATE_KHR
```


Here's What the Vulkan Spec Has to Say About Present Modes, I

`VK_PRESENT_MODE_IMMEDIATE_KHR` specifies that the presentation engine does not wait for a vertical blanking period to update the current image, meaning this mode **may** result in visible tearing. No internal queuing of presentation requests is needed, as the requests are applied immediately.

`VK_PRESENT_MODE_MAILBOX_KHR` specifies that the presentation engine waits for the next vertical blanking period to update the current image. Tearing **cannot** be observed. An internal single-entry queue is used to hold pending presentation requests. If the queue is full when a new presentation request is received, the new request replaces the existing entry, and any images associated with the prior entry become available for re-use by the application. One request is removed from the queue and processed during each vertical blanking period in which the queue is non-empty.

`VK_PRESENT_MODE_FIFO_KHR` specifies that the presentation engine waits for the next vertical blanking period to update the current image. Tearing **cannot** be observed. An internal queue is used to hold pending presentation requests. New requests are appended to the end of the queue, and one request is removed from the beginning of the queue and processed during each vertical blanking period in which the queue is non-empty. This is the only value of `presentMode` that is **required** to be supported.

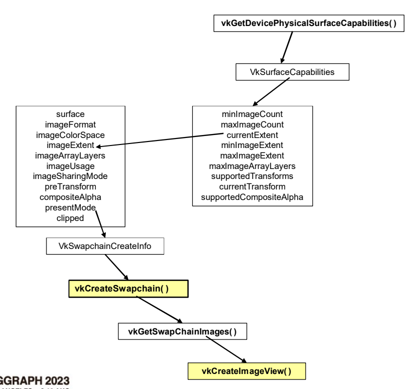
`VK_PRESENT_MODE_FIFO_RELAXED_KHR` specifies that the presentation engine generally waits for the next vertical blanking period to update the current image. If a vertical blanking period has already passed since the last update of the current image then the presentation engine does not wait for another vertical blanking period for the update, meaning this mode **may** result in visible tearing in this case. This mode is useful for reducing visual stutter with an application that will mostly present a new image before the next vertical blanking period, but may occasionally be late, and present a new image just after the next vertical blanking period. An internal queue is used to hold pending presentation requests. New requests are appended to the end of the queue, and one request is removed from the beginning of the queue and processed during or after each vertical blanking period in which the queue is non-empty.

Here's What the Vulkan Spec Has to Say About Present Modes, II

`VK_PRESENT_MODE_SHARED_DEMAND_REFRESH_KHR` specifies that the presentation engine and application have concurrent access to a single image, which is referred to as a *shared presentable image*. The presentation engine is only required to update the current image after a new presentation request is received. Therefore the application **must** make a presentation request whenever an update is required. However, the presentation engine **may** update the current image at any point, meaning this mode **may** result in visible tearing.

`VK_PRESENT_MODE_SHARED_CONTINUOUS_REFRESH_KHR` specifies that the presentation engine and application have concurrent access to a single image, which is referred to as a *shared presentable image*. The presentation engine periodically updates the current image on its regular refresh cycle. The application is only required to make one initial presentation request, after which the presentation engine **must** update the current image without any need for further presentation requests. The application **can** indicate the image contents have been updated by making a presentation request, but this does not guarantee the timing of when it will be updated. This mode **may** result in visible tearing if rendering to the image is not timed correctly.

Creating a Swap Chain



Creating a Swap Chain

217

```
VkSurfaceCapabilitiesKHR          vsc;
vkGetPhysicalDeviceSurfaceCapabilitiesKHR( PhysicalDevice, Surface, OUT &vsc );
VkExtent2D surfaceRes = vsc.currentExtent;

VkSwapchainCreateInfoKHR          vscci;
        vscci.sType = VK_STRUCTURE_TYPE_SWAPCHAIN_CREATE_INFO_KHR;
        vscci.pNext = nullptr;
        vscci.flags = 0;
        vscci.surface = Surface;
        vscci.minImageCount = 2;                // double buffering
        vscci.imageFormat       = VK_FORMAT_B8G8R8A8_UNORM;
        vscci.imageColorSpace = VK_COLORSPACE_SRGB_NONLINEAR_KHR;
        vscci.imageExtent.width = surfaceRes.width;
        vscci.imageExtent.height = surfaceRes.height;
        vscci.imageUsage     = VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT;
        vscci.preTransform    = VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR;
        vscci.compositeAlpha = VK_COMPOSITE_ALPHA_OPAQUE_BIT_KHR;
        vscci.imageArrayLayers = 1;
        vscci.imageSharingMode = VK_SHARING_MODE_EXCLUSIVE;
        vscci.queueFamilyIndexCount = 0;
        vscci.pQueueFamilyIndices = (const uint32_t *)nullptr;
        vscci.presentMode = VK_PRESENT_MODE_MAILBOX_KHR;
        vscci.oldSwapchain = VK_NULL_HANDLE;
        vscci.clipped = VK_TRUE;

result = vkCreateSwapchainKHR( LogicalDevice, IN &vscci, PALLOCATOR, OUT &SwapChain );
```

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Creating the Swap Chain Images and Image Views

218

```
uint32_t  imageCount;          // # of display buffers – 2? 3?
result = vkGetSwapchainImagesKHR( LogicalDevice, IN SwapChain, OUT &imageCount, (VkImage *)nullptr );

PresentImages = new VkImage[ imageCount ];
result = vkGetSwapchainImagesKHR( LogicalDevice, SwapChain, OUT &imageCount, PresentImages );

// present views for the double-buffering:

PresentImageViews = new VkImageView[ imageCount ];

for( unsigned int i = 0; i < imageCount; i++ )
{
    VkImageViewCreateInfo       vivci;
        vivci.sType = VK_STRUCTURE_TYPE_IMAGE_VIEW_CREATE_INFO;
        vivci.pNext = nullptr;
        vivci.flags = 0;
        vivci.viewType = VK_IMAGE_VIEW_TYPE_2D;
        vivci.format = VK_FORMAT_B8G8R8A8_UNORM;
        vivci.components.r = VK_COMPONENT_SWIZZLE_R;
        vivci.components.g = VK_COMPONENT_SWIZZLE_G;
        vivci.components.b = VK_COMPONENT_SWIZZLE_B;
        vivci.components.a = VK_COMPONENT_SWIZZLE_A;
        vivci.subresourceRange.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
        vivci.subresourceRange.baseMipLevel = 0;
        vivci.subresourceRange.levelCount = 1;
        vivci.subresourceRange.baseArrayLayer = 0;
        vivci.subresourceRange.layerCount = 1;
        vivci.image = PresentImages[ i ];

    result = vkCreateImageView( LogicalDevice, IN &vivci, PALLOCATOR, OUT &PresentImageViews[ i ] );
}
```

Rendering into the Swap Chain, I

219

```
VkSemaphoreCreateInfo          vsci;
    vsci.sType = VK_STRUCTURE_TYPE_SEMAPHORE_CREATE_INFO;
    vsci.pNext = nullptr;
    vsci.flags = 0;

VkSemaphore  imageReadySemaphore;
result = vkCreateSemaphore( LogicalDevice, IN &vsci, PALLOCATOR, OUT &imageReadySemaphore );

uint32_t  nextImageIndex;
uint64_t  tmeout = UINT64_MAX;
vkAcquireNextImageKHR( LogicalDevice,   IN SwapChain, IN timeout, IN imageReadySemaphore,
                       IN VK_NULL_HANDLE,  OUT &nextImageIndex );
    . . .

result = vkBeginCommandBuffer( CommandBuffers[ nextImageIndex ], IN &vcbbi );

    . . .

vkCmdBeginRenderPass( CommandBuffers[nextImageIndex], IN &vrpbi,
                      IN  VK_SUBPASS_CONTENTS_INLINE );

vkCmdBindPipeline( CommandBuffers[nextImageIndex], VK_PIPELINE_BIND_POINT_GRAPHICS, GraphicsPipeline );

    . . .

vkCmdEndRenderPass( CommandBuffers[ nextImageIndex ] );
vkEndCommandBuffer(  CommandBuffers[ nextImageIndex ] );
```

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Rendering into the Swap Chain, II

220

```
VkFenceCreateInfo              vfci;
        vfci.sType = VK_STRUCTURE_TYPE_FENCE_CREATE_INFO;
        vfci.pNext = nullptr;
        vfci.flags = 0;

VkFence  renderFence;
vkCreateFence( LogicalDevice, &vfci, PALLOCATOR, OUT &renderFence );

VkQueue presentQueue;
vkGetDeviceQueue( LogicalDevice, FindQueueFamilyThatDoesGraphics( ), 0,
                  OUT &presentQueue );

    . . .

VkSubmitInfo                   vsi;
        vsi.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
        vsi.pNext = nullptr;
        vsi.waitSemaphoreCount = 1;
        vsi.pWaitSemaphores = &imageReadySemaphore;
        vsi.pWaitDstStageMask = &waitAtBottom;
        vsi.commandBufferCount = 1;
        vsi.pCommandBuffers = &CommandBuffers[ nextImageIndex ];
        vsi.signalSemaphoreCount = 0;
        vsi.pSignalSemaphores = &SemaphoreRenderFinished;

result = vkQueueSubmit( presentQueue, 1, IN &vsi, IN renderFence );    // 1 = submitCount
```

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Rendering into the Swap Chain, III

221

```
result = vkWaitForFences( LogicalDevice, 1, IN &renderFence, VK_TRUE, UINT64_MAX );

VkPresentInfoKHR               vpi;
        vpi.sType = VK_STRUCTURE_TYPE_PRESENT_INFO_KHR;
        vpi.pNext = nullptr;
        vpi.waitSemaphoreCount = 0;
        vpi.pWaitSemaphores = (VkSemaphore *)nullptr;
        vpi.swapchainCount = 1;
        vpi.pSwapchains = &SwapChain;
        vpi.pImageIndices = &nextImageIndex;
        vpi.pResults = (VkResult *) nullptr;

result = vkQueuePresentKHR( presentQueue, IN &vpi );
```

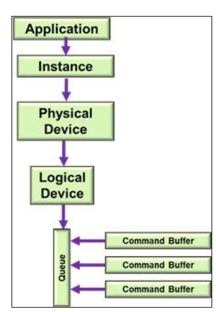
SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

222

Vulkan.

Physical Devices

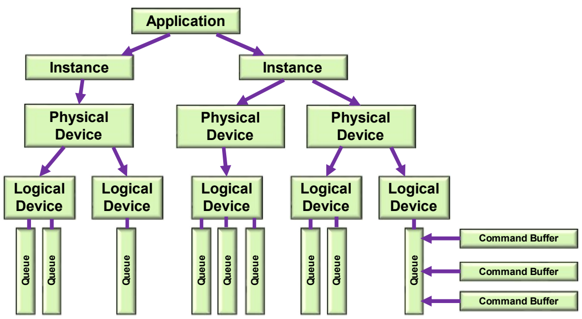


SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

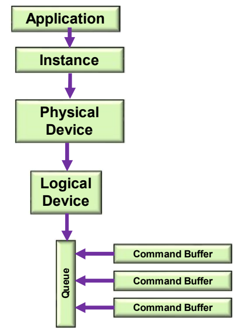
Vulkan: Overall Block Diagram

223



Vulkan: a More Typical (and Simplified) Block Diagram

224



Querying the Number of Physical Devices

225

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

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

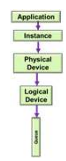
This way of querying information is a recurring OpenCL and Vulkan pattern (get used to it):

How many total there are | Where to put them

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

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


Vulkan: Identifying the Physical Devices

226




```
VkResult result = VK_SUCCESS;

result = vkEnumeratePhysicalDevices( Instance, OUT &PhysicalDeviceCount, (VkPhysicalDevice *)nullptr );
if( result != VK_SUCCESS || PhysicalDeviceCount <= 0 )
{
	fprintf( FpDebug, "Could not count the physical devices\n" );
	return VK_SHOULD_EXIT;
}

fprintf(FpDebug, "\n%d physical devices found.\n", PhysicalDeviceCount);

VkPhysicalDevice * physicalDevices = new VkPhysicalDevice[ PhysicalDeviceCount ];
result = vkEnumeratePhysicalDevices( Instance, OUT &PhysicalDeviceCount, OUT physicalDevices  );
if( result != VK_SUCCESS )
{
	fprintf( FpDebug, "Could not enumerate the %d physical devices\n", PhysicalDeviceCount );
	return VK_SHOULD_EXIT;
}
```


Which Physical Device to Use, I

227

```
int discreteSelect   = -1;
int integratedSelect = -1;
for( unsigned int i = 0; i < PhysicalDeviceCount; i++ )
{
	VkPhysicalDeviceProperties vpdp;
	vkGetPhysicalDeviceProperties( IN physicalDevices[i], OUT &vpdp );
	if( result != VK_SUCCESS )
	{
		fprintf( FpDebug, "Could not get the physical device properties of device %d\n", i );
		return VK_SHOULD_EXIT;
	}

	fprintf( FpDebug, " \n\nDevice %2d:\n", i );
	fprintf( FpDebug, "\tAPI version: %d\n", vpdp.apiVersion );
	fprintf( FpDebug, "\tDriver version: %d\n", vpdp.apiVersion );
	fprintf( FpDebug, "\tVendor ID: 0x%04x\n", vpdp.vendorID );
	fprintf( FpDebug, "\tDevice ID: 0x%04x\n", vpdp.deviceID );
	fprintf( FpDebug, "\tPhysical Device Type: %d =", vpdp.deviceType );
	if( vpdp.deviceType == VK_PHYSICAL_DEVICE_TYPE_DISCRETE_GPU )     fprintf( FpDebug, " (Discrete GPU)\n" );
	if( vpdp.deviceType == VK_PHYSICAL_DEVICE_TYPE_INTEGRATED_GPU )   fprintf( FpDebug, " (Integrated GPU)\n" );
	if( vpdp.deviceType == VK_PHYSICAL_DEVICE_TYPE_VIRTUAL_GPU )      fprintf( FpDebug, " (Virtual GPU)\n" );
	if( vpdp.deviceType == VK_PHYSICAL_DEVICE_TYPE_CPU )              fprintf( FpDebug, " (CPU)\n" );
	fprintf( FpDebug, "\tDevice Name: %s\n", vpdp.deviceName );
	fprintf( FpDebug, "\tPipeline Cache Size: %d\n", vpdp.pipelineCacheUUID[0] );
```


Which Physical Device to Use, II

228

```
		// need some logical here to decide which physical device to select:

		if( vpdp.deviceType == VK_PHYSICAL_DEVICE_TYPE_DISCRETE_GPU )
			discreteSelect = i;

		if( vpdp.deviceType == VK_PHYSICAL_DEVICE_TYPE_INTEGRATED_GPU )
			integratedSelect = i;
	}

	int which = -1;
	if( discreteSelect >= 0 )
	{
		which = discreteSelect;
		PhysicalDevice = physicalDevices[which];
	}
	else if( integratedSelect >= 0 )
	{
		which = integratedSelect;
		PhysicalDevice = physicalDevices[which];
	}
	else
	{
		fprintf( FpDebug, "Could not select a Physical Device\n" );
		return VK_SHOULD_EXIT;
	}
```

Asking About the Physical Device's Features

229

```
VkPhysicalDeviceProperties  PhysicalDeviceFeatures;
vkGetPhysicalDeviceFeatures( IN PhysicalDevice, OUT &PhysicalDeviceFeatures );

fprintf( FpDebug, "\nPhysical Device Features:\n");
fprintf( FpDebug, "geometryShader = %2d\n", PhysicalDeviceFeatures.geometryShader);
fprintf( FpDebug, "tessellationShader = %2d\n", PhysicalDeviceFeatures.tessellationShader );
fprintf( FpDebug, "multiDrawIndirect = %2d\n", PhysicalDeviceFeatures.multiDrawIndirect );
fprintf( FpDebug, "wideLines = %2d\n", PhysicalDeviceFeatures.wideLines );
fprintf( FpDebug, "largePoints = %2d\n", PhysicalDeviceFeatures.largePoints );
fprintf( FpDebug, "multiViewport = %2d\n", PhysicalDeviceFeatures.multiViewport );
fprintf( FpDebug, "occlusionQueryPrecise = %2d\n", PhysicalDeviceFeatures.occlusionQueryPrecise );
fprintf( FpDebug, "pipelineStatisticsQuery = %2d\n", PhysicalDeviceFeatures.pipelineStatisticsQuery );
fprintf( FpDebug, "shaderFloat64 = %2d\n", PhysicalDeviceFeatures.shaderFloat64 );
fprintf( FpDebug, "shaderInt64 = %2d\n", PhysicalDeviceFeatures.shaderInt64 );
fprintf( FpDebug, "shaderInt16 = %2d\n", PhysicalDeviceFeatures.shaderInt16 );
```

SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Here's What the Nvidia A6000 Produced

230

```
Init03PhysicalDeviceAndGetQueueFamilyProperties

Device  0:
      API version: 4206797
      Driver version: 4206797
      Vendor ID: 0x10de
      Device ID: 0x2230
      Physical Device Type: 2 = (Discrete GPU)
      Device Name: NVIDIA RTX A6000
      Pipeline Cache Size: 72
Device #0 selected ('NVIDIA RTX A6000')

Physical Device Features:
geometryShader =  1
tessellationShader =  1
multiDrawIndirect =  1
wideLines =  1
largePoints =  1
multiViewport =  1
occlusionQueryPrecise =  1
pipelineStatisticsQuery =  1
shaderFloat64 =  1
shaderInt64 =  1
shaderInt16 =  1
```

SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Here's What the Intel HD Graphics 520 Produced

231

```
Init03PhysicalDeviceAndGetQueueFamilyProperties

Device  0:
      API version: 4194360
      Driver version: 4194360
      Vendor ID: 0x8086
      Device ID: 0x1916
      Physical Device Type: 1 = (Integrated GPU)
      Device Name: Intel(R) HD Graphics 520
      Pipeline Cache Size: 213
Device #0 selected ('Intel(R) HD Graphics 520')

Physical Device Features:
geometryShader =  1
tessellationShader =  1
multiDrawIndirect =  1
wideLines =  1
largePoints =  1
multiViewport =  1
occlusionQueryPrecise =  1
pipelineStatisticsQuery =  1
shaderFloat64 =  1
shaderInt64 =  1
shaderInt16 =  1
```

SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Asking About the Physical Device's Different Memories

232

```
VkPhysicalDeviceMemoryProperties          vpdmp;
vkGetPhysicalDeviceMemoryProperties( PhysicalDevice, OUT &vpdmp );

fprintf( FpDebug, "\n%d Memory Types:\n", vpdmp.memoryTypeCount );
for( unsigned int i = 0; i < vpdmp.memoryTypeCount; i++ )
 {
      VkMemoryType vmt = vpdmp.memoryTypes[i];
      fprintf( FpDebug, "Memory %2d: ", i );
      if( ( vmt.propertyFlags & VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT      )  != 0 )   fprintf( FpDebug, " DeviceLocal" );
      if( ( vmt.propertyFlags & VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT      )  != 0 )   fprintf( FpDebug, " HostVisible" );
      if( ( vmt.propertyFlags & VK_MEMORY_PROPERTY_HOST_COHERENT_BIT )  != 0 )   fprintf( FpDebug, " HostCoherent" );
      if( ( vmt.propertyFlags & VK_MEMORY_PROPERTY_HOST_CACHED_BIT       )  != 0 )   fprintf( FpDebug, " HostCached" );
      if( ( vmt.propertyFlags & VK_MEMORY_PROPERTY_LAZILY_ALLOCATED_BIT ) != 0 )   fprintf( FpDebug, " LazilyAllocated" );
      fprintf(FpDebug, "\n");
}

fprintf( FpDebug, "\n%d Memory Heaps:\n", vpdmp.memoryHeapCount );
for( unsigned int  i = 0; i < vpdmp.memoryHeapCount; i++ )
{
      fprintf(FpDebug, "Heap %d: ", i);
      VkMemoryHeap vmh = vpdmp.memoryHeaps[i];
      fprintf( FpDebug, " size = 0x%08lx", (unsigned long int)vmh.size );
      if( ( vmh.flags & VK_MEMORY_HEAP_DEVICE_LOCAL_BIT  ) != 0 )    fprintf( FpDebug, " DeviceLocal" );    // only one in use
      fprintf(FpDebug, "\n");
}
```

SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Here's What I Got on the Nvidia A6000

233

```
6 Memory Types:
Memory  0:
Memory  1:  DeviceLocal
Memory  2:  HostVisible  HostCoherent
Memory  3:  HostVisible  HostCoherent HostCached
Memory  4:  DeviceLocal HostVisible HostCoherent
Memory  5:  DeviceLocal

4 Memory Heaps:
Heap 0:  size = 0xdbb00000 DeviceLocal
Heap 1:  size = 0xfd504000
Heap 2:  size = 0x0d600000 DeviceLocal
Heap 3:  size = 0x02000000 DeviceLocal
```

SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Asking About the Physical Device's Queue Families

234

```
uint32_t count = -1;
vkGetPhysicalDeviceQueueFamilyProperties( IN PhysicalDevice, &count, OUT (VkQueueFamilyProperties *)nullptr );
fprintf( FpDebug, "\nFound %d Queue Families:\n", count );

VkQueueFamilyProperties *vqfp = new VkQueueFamilyProperties[ count ];
vkGetPhysicalDeviceQueueFamilyProperties( IN PhysicalDevice, &count, OUT vqfp );
for( unsigned int i = 0; i < count; i++ )
{
      fprintf( FpDebug, "\t%d: queueCount = %2d  ;   ", i, vqfp[i].queueCount );
      if( ( vqfp[i].queueFlags & VK_QUEUE_GRAPHICS_BIT ) != 0 )      fprintf( FpDebug, " Graphics" );
      if( ( vqfp[i].queueFlags & VK_QUEUE_COMPUTE_BIT  ) != 0 )      fprintf( FpDebug, " Compute " );
      if( ( vqfp[i].queueFlags & VK_QUEUE_TRANSFER_BIT ) != 0 )      fprintf( FpDebug, " Transfer" );
      fprintf(FpDebug, "\n");
}
```

SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

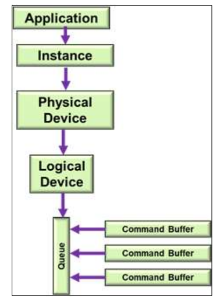
mjb – June 5, 2023

Here's What I Got on the Nvidia A6000

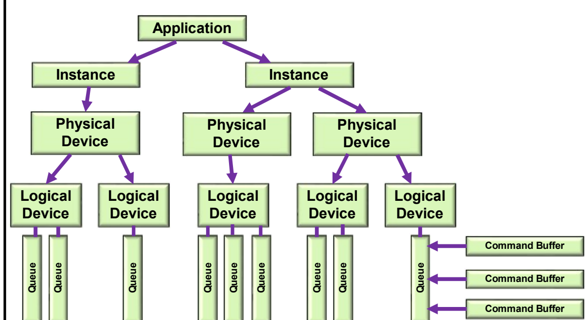

```
Found 3 Queue Families:
     0: Queue Family Count = 16  ;    Graphics Compute Transfer
     1: Queue Family Count =  2  ;    Transfer
     2: Queue Family Count =  8  ;    Compute Transfer
```


Vulkan.

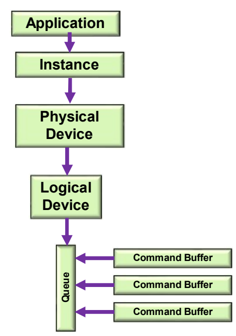
Logical Devices



Vulkan: Overall Block Diagram



Vulkan: a More Typical (and Simplified) Block Diagram



Looking to See What Device Layers are Available


```
const char * myDeviceLayers[ ] =
{
        // "VK_LAYER_LUNARG_api_dump",
        // "VK_LAYER_LUNARG_core_validation",
        // "VK_LAYER_LUNARG_image",
        "VK_LAYER_LUNARG_object_tracker",
        "VK_LAYER_LUNARG_parameter_validation",
        // "VK_LAYER_NV_optimus"
};

const char * myDeviceExtensions[ ] =
{
        "VK_KHR_surface",
        "VK_KHR_win32_surface",
        "VK_EXT_debug_report"
        // "VK_KHR_swapchains"
};

// see what device layers are available:

uint32_t layerCount;
vkEnumerateDeviceLayerProperties(PhysicalDevice, &layerCount, (VkLayerProperties *)nullptr);

VkLayerProperties * deviceLayers = new VkLayerProperties[layerCount];

result = vkEnumerateDeviceLayerProperties( PhysicalDevice, &layerCount, deviceLayers);
```


Looking to See What Device Extensions are Available


```
// see what device extensions are available:

uint32_t extensionCount;
vkEnumerateDeviceExtensionProperties(PhysicalDevice, deviceLayers[i].layerName,
                                     &extensionCount, (VkExtensionProperties *)nullptr);

VkExtensionProperties * deviceExtensions = new VkExtensionProperties[extensionCount];

result = vkEnumerateDeviceExtensionProperties(PhysicalDevice, deviceLayers[i].layerName,
                                     &extensionCount, deviceExtensions);
```

What Device Layers and Extensions are Available

241

```
4 physical device layers enumerated:

0x004030cd   1  'VK_LAYER_NV_optimus'  'NVIDIA Optimus layer'
        160 device extensions enumerated for 'VK_LAYER_NV_optimus':

0x00400033   1  'VK_LAYER_LUNARG_core_validation'  'LunarG Validation Layer'
        0 device extensions enumerated for 'VK_LAYER_LUNARG_core_validation':

0x00400033   1  'VK_LAYER_LUNARG_object_tracker'  'LunarG Validation Layer'
        160 device extensions enumerated for 'VK_LAYER_LUNARG_object_tracker':

0x00400033   1  'VK_LAYER_LUNARG_parameter_validation'  'LunarG Validation Layer'
        160 device extensions enumerated for 'VK_LAYER_LUNARG_parameter_validation':
```

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

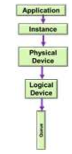
mjb – June 5, 2023

Vulkan: Creating a Logical Device

242

```
float   queuePriorities[1] =
{
        1.
};
VkDeviceQueueCreateInfo   vdqci;
        vdqci.sType = VK_STRUCTURE_TYPE_DEVICE_QUEUE_CREATE_INFO;
        vdqci.pNext = nullptr;
        vdqci.flags = 0;
        vdqci.queueFamiltyIndex = 0;
        vdqci.queueCount = 1;
        vdqci.pQueueProperties = queuePriorities;
```

```
VkDeviceCreateInfo   vdci;
        vdci.sType = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
        vdci.pNext = nullptr;
        vdci.flags = 0;
        vdci.queueCreateInfoCount = 1;                 // # of device queues
        vdci.pQueueCreateInfos = IN vdqci;             // array of VkDeviceQueueCreateInfo's
        vdci.enabledLayerCount = sizeof(myDeviceLayers) / sizeof(char *);
        vdci.enabledLayerCount = 0;
        vdci.ppEnabledLayerNames = myDeviceLayers;
        vdci.enabledExtensionCount = sizeof(myDeviceExtensions) / sizeof(char *);
        vdci.ppEnabledExtensionNames = myDeviceExtensions;
        vdci.pEnabledFeatures = IN &PhysicalDeviceFeatures;

    result = vkCreateLogicalDevice( PhysicalDevice, IN &vdci, PALLOCATOR, OUT &LogicalDevice );
```



SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Vulkan: Creating the Logical Device's Queue

243

```
// get the queue for this logical device:

vkGetDeviceQueue( LogicalDevice, 0, 0,  OUT &Queue );            // 0, 0 = queueFamilyIndex, queueIndex
```

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

244

Layers and Extensions


```
vkEnumerateInstanceLayerProperties:

13 instance layers enumerated:
0x00400033  2  'VK_LAYER_LUNARG_api_dump'               'LunarG debug layer'
0x00400033  1  'VK_LAYER_LUNARG_core_validation'        'LunarG Validation Layer'
0x00400033  1  'VK_LAYER_LUNARG_monitor'                'Execution Monitoring Layer'
0x00400033  1  'VK_LAYER_LUNARG_object_tracker'         'LunarG Validation Layer'
0x00400033  1  'VK_LAYER_LUNARG_parameter_validation'   'LunarG Validation Layer'
0x00400033  1  'VK_LAYER_LUNARG_screenshot'             'LunarG image capture layer'
0x00400033  1  'VK_LAYER_LUNARG_standard_validation'    'LunarG Standard Validation'
0x00400033  1  'VK_LAYER_GOOGLE_threading'              'Google Validation Layer'
0x00400033  1  'VK_LAYER_GOOGLE_unique_objects'         'Google Validation Layer'
0x00400033  1  'VK_LAYER_LUNARG_vktrace'                'Vktrace tracing library'
0x00400038  1  'VK_LAYER_NV_optimus'                    'NVIDIA Optimus layer'
0x0040000d  1  'VK_LAYER_NV_nsight'                     'NVIDIA Nsight interception layer'
0x00400000  34  'VK_LAYER_RENDERDOC_Capture'            'Debugging capture layer for RenderDoc'
```


SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

245

```
vkEnumerateInstanceLayerProperties:

13 instance layers enumerated:
0x00400033  2  'VK_LAYER_LUNARG_api_dump'               'LunarG debug layer'
0x00400033  1  'VK_LAYER_LUNARG_core_validation'        'LunarG Validation Layer'
0x00400033  1  'VK_LAYER_LUNARG_monitor'                'Execution Monitoring Layer'
0x00400033  1  'VK_LAYER_LUNARG_object_tracker'         'LunarG Validation Layer'
0x00400033  1  'VK_LAYER_LUNARG_parameter_validation'   'LunarG Validation Layer'
0x00400033  1  'VK_LAYER_LUNARG_screenshot'             'LunarG image capture layer'
0x00400033  1  'VK_LAYER_LUNARG_standard_validation'    'LunarG Standard Validation'
0x00400033  1  'VK_LAYER_GOOGLE_threading'              'Google Validation Layer'
0x00400033  1  'VK_LAYER_GOOGLE_unique_objects'         'Google Validation Layer'
0x00400033  1  'VK_LAYER_LUNARG_vktrace'                'Vktrace tracing library'
0x00400038  1  'VK_LAYER_NV_optimus'                    'NVIDIA Optimus layer'
0x0040000d  1  'VK_LAYER_NV_nsight'                     'NVIDIA Nsight interception layer'
0x00400000  34  'VK_LAYER_RENDERDOC_Capture'            'Debugging capture layer for RenderDoc'
```

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

246

```
vkEnumerateInstanceExtensionProperties:

11 extensions enumerated:
0x00000008  'VK_EXT_debug_report'
0x00000001  'VK_EXT_display_surface_counter'
0x00000001  'VK_KHR_get_physical_device_properties2'
0x00000001  'VK_KHR_get_surface_capabilities2'
0x00000019  'VK_KHR_surface'
0x00000006  'VK_KHR_win32_surface'
0x00000001  'VK_KHX_device_group_creation'
0x00000001  'VK_KHR_external_fence_capabilities'
0x00000001  'VK_KHR_external_memory_capabilities'
0x00000001  'VK_KHR_external_semaphore_capabilities'
0x00000001  'VK_NV_external_memory_capabilities'
```

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

247

```
vkEnumerateDeviceLayerProperties:

3 physical device layers enumerated:
0x00400038   1  'VK_LAYER_NV_optimus'  'NVIDIA Optimus layer'
        0 device extensions enumerated for 'VK_LAYER_NV_optimus':

0x00400033   1  'VK_LAYER_LUNARG_object_tracker'  'LunarG Validation Layer'
        0 device extensions enumerated for 'VK_LAYER_LUNARG_object_tracker':

0x00400033   1  'VK_LAYER_LUNARG_parameter_validation'  'LunarG Validation Layer'
        0 device extensions enumerated for 'VK_LAYER_LUNARG_parameter_validation':
```

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

248

```
const char * instanceLayers[ ] =
{
        //"VK_LAYER_LUNARG_api_dump",   // turn this on if want to see each function call and its arguments (very slow!)
        "VK_LAYER_LUNARG_core_validation",
        "VK_LAYER_LUNARG_object_tracker",
        "VK_LAYER_LUNARG_parameter_validation",
        "VK_LAYER_NV_optimus"
;

const char * instanceExtensions[ ] =
{
        "VK_KHR_surface",
#ifdef _WIN32
        "VK_KHR_win32_surface",
#endif
        "VK_EXT_debug_report",
};
uint32_t numExtensionsWanted = sizeof(instanceExtensions) / sizeof(char *);

// see what layers are available:

vkEnumerateInstanceLayerProperties( &numLayersAvailable, (VkLayerProperties *)nullptr );
InstanceLayers = new VkLayerProperties[ numLayersAvailable ];
result = vkEnumerateInstanceLayerProperties( &numLayersAvailable, InstanceLayers );

// see what extensions are available:

uint32_t numExtensionsAvailable;
vkEnumerateInstanceExtensionProperties( (char *)nullptr, &numExtensionsAvailable, (VkExtensionProperties *)nullptr );
InstanceExtensions = new VkExtensionProperties[ numExtensionsAvailable ];
result = vkEnumerateInstanceExtensionProperties( (char *)nullptr, &numExtensionsAvailable, InstanceExtensions );
```

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

249

```
13 instance layers available:
0x00400033  2  'VK_LAYER_LUNARG_api_dump'  'LunarG debug layer'
0x00400033  1  'VK_LAYER_LUNARG_core_validation'  'LunarG Validation Layer'
0x00400033  1  'VK_LAYER_LUNARG_monitor'  'Execution Monitoring Layer'
0x00400033  1  'VK_LAYER_LUNARG_object_tracker'  'LunarG Validation Layer'
0x00400033  1  'VK_LAYER_LUNARG_parameter_validation'  'LunarG Validation Layer'
0x00400033  1  'VK_LAYER_LUNARG_screenshot'  'LunarG image capture layer'
0x00400033  1  'VK_LAYER_LUNARG_standard_validation'  'LunarG Standard Validation'
0x00400033  1  'VK_LAYER_GOOGLE_threading'  'Google Validation Layer'
0x00400033  1  'VK_LAYER_GOOGLE_unique_objects'  'Google Validation Layer'
0x00400033  1  'VK_LAYER_LUNARG_vktrace'  'Vktrace tracing library'
0x00400038  1  'VK_LAYER_NV_optimus'  'NVIDIA Optimus layer'
0x0040000d  1  'VK_LAYER_NV_nsight'  'NVIDIA Nsight interception layer'
0x00400000  34  'VK_LAYER_RENDERDOC_Capture'  'Debugging capture layer for RenderDoc'
```

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

250

```
11 instance extensions available:
0x00000008  'VK_EXT_debug_report'
0x00000001  'VK_EXT_display_surface_counter'
0x00000001  'VK_KHR_get_physical_device_properties2'
0x00000001  'VK_KHR_get_surface_capabilities2'
0x00000019  'VK_KHR_surface'
0x00000006  'VK_KHR_win32_surface'
0x00000001  'VK_KHX_device_group_creation'
0x00000001  'VK_KHR_external_fence_capabilities'
0x00000001  'VK_KHR_external_memory_capabilities'
0x00000001  'VK_KHR_external_semaphore_capabilities'
0x00000001  'VK_NV_external_memory_capabilities'
```

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

251

```
// look for extensions both on the wanted list and the available list:

std::vector<char *> extensionsWantedAndAvailable;
extensionsWantedAndAvailable.clear( );
for( uint32_t wanted = 0; wanted < numExtensionsWanted; wanted++ )
{
        for( uint32_t available = 0; available < numExtensionsAvailable; available++ )
        {
                if( strcmp( instanceExtensions[wanted], InstanceExtensions[available].extensionName ) == 0 )
                {
                        extensionsWantedAndAvailable.push_back( InstanceExtensions[available].extensionName );
                        break;
                }
        }
}

// create the instance, asking for the layers and extensions:

VkInstanceCreateInfo vici;
        vici.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
        vici.pNext = nullptr;
        vici.flags = 0;
        vici.pApplicationInfo = &vai;
        vici.enabledLayerCount = sizeof(instanceLayers) / sizeof(char *);
        vici.ppEnabledLayerNames = instanceLayers;
        vici.enabledExtensionCount = extensionsWantedAndAvailable.size( );
        vici.ppEnabledExtensionNames = extensionsWantedAndAvailable.data( );;


result = vkCreateInstance( IN &vici, PALLOCATOR, OUT &Instance );
```

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

252

```
Will now ask for 3 instance extensions
        VK_KHR_surface
        VK_KHR_win32_surface
        VK_EXT_debug_report
```

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

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

int discreteSelect   = -1;
int integratedSelect = -1;
for( unsigned int i = 0; i < PhysicalDeviceCount; i++ )
{
        VkPhysicalDeviceProperties vpdp;
        vkGetPhysicalDeviceProperties( IN physicalDevices[i], OUT &vpdp );

        // need some logical here to decide which physical device to select:
        if( vpdp.deviceType == VK_PHYSICAL_DEVICE_TYPE_DISCRETE_GPU )
            discreteSelect = i;

        if( vpdp.deviceType == VK_PHYSICAL_DEVICE_TYPE_INTEGRATED_GPU )
            integratedSelect = i;
}

int which = -1;
if( discreteSelect >= 0 )
{
        which = discreteSelect;
        PhysicalDevice = physicalDevices[which];
}
else if( integratedSelect >= 0 )
{
        which = integratedSelect;
        PhysicalDevice = physicalDevices[which];
}
else
{
        fprintf( FpDebug, "Could not select a Physical Device\n" );
        return VK_SHOULD_EXIT;
}
delete[ ] physicalDevices;
```



```
vkGetPhysicalDeviceProperties( PhysicalDevice, OUT &PhysicalDeviceProperties );

vkGetPhysicalDeviceFeatures( IN PhysicalDevice, OUT &PhysicalDeviceFeatures );

vkGetPhysicalDeviceFormatProperties( PhysicalDevice, IN VK_FORMAT_R32G32B32A32_SFLOAT, &vfp );
vkGetPhysicalDeviceFormatProperties( PhysicalDevice, IN VK_FORMAT_R8G8B8A8_UNORM, &vfp );
vkGetPhysicalDeviceFormatProperties( PhysicalDevice, IN VK_FORMAT_B8G8R8A8_UNORM, &vfp );

VkPhysicalDeviceMemoryProperties            vpdmp;
vkGetPhysicalDeviceMemoryProperties( PhysicalDevice, OUT &vpdmp );

uint32_t count = -1;
vkGetPhysicalDeviceQueueFamilyProperties( IN PhysicalDevice, &count, OUT (VkQueueFamilyProperties *)nullptr );
VkQueueFamilyProperties *vqfp = new VkQueueFamilyProperties[ count ];
vkGetPhysicalDeviceQueueFamilyProperties( IN PhysicalDevice, &count, OUT vqfp );

delete[ ] vqfp;
```



```
VkResult result;
float   queuePriorities[NUM_QUEUES_WANTED] =
{
        1.
};

VkDeviceQueueCreateInfo            vdqci[NUM_QUEUES_WANTED];
        vdqci[0].sType = VK_STRUCTURE_TYPE_DEVICE_QUEUE_CREATE_INFO;
        vdqci[0].pNext = nullptr;
        vdqci[0].flags = 0;
        vdqci[0].queueFamilyIndex = FindQueueFamilyThatDoesGraphics( );
        vdqci[0].queueCount = 1;          // how many queues to create
        vdqci[0].pQueuePriorities = queuePriorities;   // array of queue priorities [0.,1.]

const char * myDeviceLayers[ ] =
{
        ////"VK_LAYER_LUNARG_api_dump",
        ////"VK_LAYER_LUNARG_core_validation",
        //"VK_LAYER_LUNARG_image",
        "VK_LAYER_LUNARG_object_tracker",
        "VK_LAYER_LUNARG_parameter_validation",
        //"VK_LAYER_NV_optimus"
};

const char * myDeviceExtensions[ ] =
{
        "VK_KHR_swapchain",
};
```



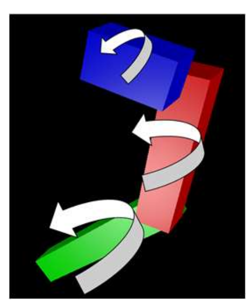
```
uint32_t layerCount;
vkEnumerateDeviceLayerProperties(PhysicalDevice, &layerCount, (VkLayerProperties *)nullptr);
VkLayerProperties * deviceLayers = new VkLayerProperties[layerCount];
result = vkEnumerateDeviceLayerProperties( PhysicalDevice, &layerCount, deviceLayers);
for (unsigned int i = 0; i < layerCount; i++)
{
        // see what device extensions are available:

        uint32_t extensionCount;
        vkEnumerateDeviceExtensionProperties(PhysicalDevice, deviceLayers[i].layerName, &extensionCount,
(VkExtensionProperties *)nullptr);
        VkExtensionProperties * deviceExtensions = new VkExtensionProperties[extensionCount];
        result = vkEnumerateDeviceExtensionProperties(PhysicalDevice, deviceLayers[i].layerName, &extensionCount,
deviceExtensions);

}

delete[ ] deviceLayers;
```



```
4 physical device layers enumerated:
0x00400038  1  'VK_LAYER_NV_optimus'  'NVIDIA Optimus layer'
vkEnumerateDeviceExtensionProperties: Successful
      0 device extensions enumerated for 'VK_LAYER_NV_optimus':

0x00400033  1  'VK_LAYER_LUNARG_core_validation'  'LunarG Validation Layer'
vkEnumerateDeviceExtensionProperties: Successful
      0 device extensions enumerated for 'VK_LAYER_LUNARG_core_validation':

0x00400033  1  'VK_LAYER_LUNARG_object_tracker'  'LunarG Validation Layer'
vkEnumerateDeviceExtensionProperties: Successful
      0 device extensions enumerated for 'VK_LAYER_LUNARG_object_tracker':

0x00400033  1  'VK_LAYER_LUNARG_parameter_validation'  'LunarG Validation Layer'
vkEnumerateDeviceExtensionProperties: Successful
      0 device extensions enumerated for 'VK_LAYER_LUNARG_parameter_validation':
```


Vulkan.

Push Constants



Push Constants

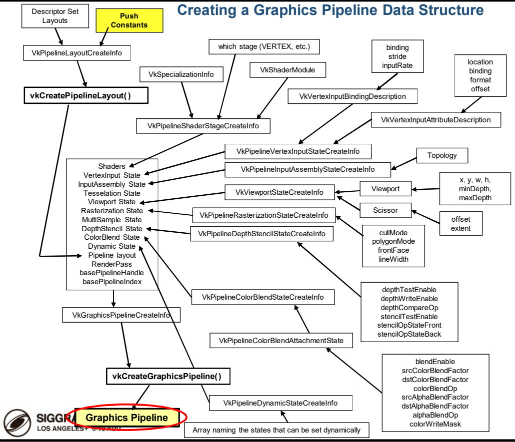
259

In an effort to expand flexibility and retain efficiency, Vulkan provides something called **Push Constants**. Like the name implies, these let you “push” constant values out to the shaders. These are typically used for small, frequently-updated data values, such as mat4 transformation matrices. This is a good feature, since Vulkan, at times, makes it cumbersome to send changes to the graphics.

By “small”, Vulkan specifies that there will be at least 128 bytes that can be used, although they can be larger. For example, the maximum size is 256 bytes on the NVIDIA 1080ti. (You can query this limit by looking at the **maxPushConstantSize** parameter in the **VkPhysicalDeviceLimits** structure.) Unlike uniform buffers and vertex buffers, these do not live in their own GPU memory. They are actually included inside the Vulkan graphics pipeline data structure.

Creating a Graphics Pipeline Data Structure

260



Push Constants

261

On the shader side, if, for example, you are sending a 4x4 matrix, the use of push constants in the shader looks like this:

```
layout( push_constant ) uniform matrix
{
        mat4 modelMatrix;
} Matrix;
```

On the application side, push constants are pushed at the shaders by giving them to the Vulkan Command Buffer:

vkCmdPushConstants(CommandBuffer, PipelineLayout, stageFlags, offset, size, pValues);

where:
stageFlags are or’ed bits of:
VK_PIPELINE_STAGE_VERTEX_SHADER_BIT
VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT
VK_PIPELINE_STAGE_TESSELATION_EVALUATION_SHADER_BIT
VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT
VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT

size is in bytes

pValues is a void * pointer to the data, which, in this 4x4 matrix example, would be of type **glm::mat4**.

Setting up the Push Constants for the Graphics Pipeline Data Structure

262

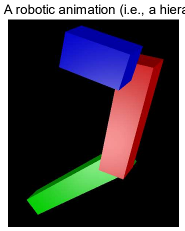
Prior to that, however, the pipeline layout needs to be told about the Push Constants:

```
VkPushConstantRange                    vpcr[1];
        vpcr[0].stageFlags =
                        VK_PIPELINE_STAGE_VERTEX_SHADER_BIT
                      | VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT;
        vpcr[0].offset = 0;
        vpcr[0].size = sizeof( glm::mat4 );

VkPipelineLayoutCreateInfo             vplci;
        vplci.sType = VK_STRUCTURE_TYPE_PIPELINE_LAYOUT_CREATE_INFO;
        vplci.pNext = nullptr;
        vplci.flags = 0;
        vplci.setLayoutCount = 4;
        vplci.pSetLayouts = DescriptorSetLayouts;
        vplci.pushConstantRangeCount = 1;
        vplci.pPushConstantRanges = vpcr;

result = vkCreatePipelineLayout( LogicalDevice, IN &vplci, PALLOCATOR,
                OUT &GraphicsPipelineLayout );
```


A Robotic Example using Push Constants

263

A robotic animation (i.e., a hierarchical transformation system)

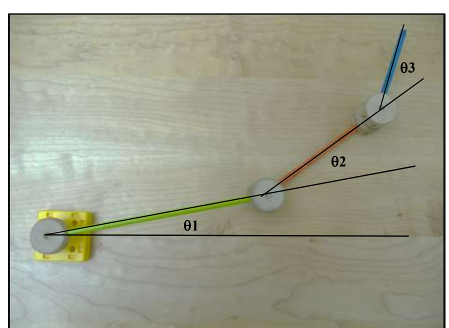


Where each arm is represented by:

```
struct arm
{
        glm::mat4   armMatrix;
        glm::vec3   armColor;
        float           armScale;     // scale factor in x
};

struct  arm       Arm1;
struct  arm       Arm2;
struct  arm       Arm3;
```


Forward Kinematics: Hook the Pieces Together, Change Parameters, and Things Move (All Young Children Understand This)

264



In the *Reset( )* Function

265

```
struct arm     Arm1;
struct arm     Arm2;
struct arm     Arm3;

. . .

        Arm1.armMatrix = glm::mat4( 1. );
        Arm1.armColor  = glm::vec3( 0.f, 1.f, 0.f );     // green
        Arm1.armScale  = 6.f;

        Arm2.armMatrix = glm::mat4( 1. );
        Arm2.armColor  = glm::vec3( 1.f, 0.f, 0.f );     // red
        Arm2.armScale  = 4.f;

        Arm3.armMatrix = glm::mat4( 1. );
        Arm3.armColor  = glm::vec3( 0.f, 0.f, 1.f );     // blue
        Arm3.armScale  = 2.f;
```

The constructor **glm::mat4(1.)** produces an identity matrix. The actual transformation matrices will be set in *UpdateScene( )*.

Set the Push Constant for the Graphics Pipeline Data Structure

266

```
VkPushConstantRange                    vpcr[1];
        vpcr[0].stageFlags =
                VK_PIPELINE_STAGE_VERTEX_SHADER_BIT
              | VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT;

        vpcr[0].offset = 0;
        vpcr[0].size = sizeof( struct arm );

VkPipelineLayoutCreateInfo                  vplci;
        vplci.sType = VK_STRUCTURE_TYPE_PIPELINE_LAYOUT_CREATE_INFO;
        vplci.pNext = nullptr;
        vplci.flags = 0;
        vplci.setLayoutCount = 5;
        vplci.pSetLayouts = DescriptorSetLayouts;
        vplci.pushConstantRangeCount = 1;
        vplci.pPushConstantRanges = vpcr;

result = vkCreatePipelineLayout( LogicalDevice, IN &vplci, PALLOCATOR,
                OUT &GraphicsPipelineLayout );
```


In the *UpdateScene( )* Function

267

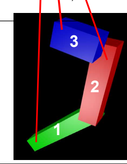
```
float rot1 = (float)(2.*M_PI*Time);        // rotation for arm1, in radians
float rot2 = 2.f * rot1;                   // rotation for arm2, in radians
float rot3 = 2.f * rot2;                   // rotation for arm3, in radians

glm::vec3 zaxis  =  glm::vec3(0., 0., 1.);

glm::mat4 m1g = glm::mat4( 1. );    // identity
m1g = glm::translate(m1g, glm::vec3(0., 0., 0.));
m1g = glm::rotate(m1g, rot1, zaxis);                 // [T]*[R]

glm::mat4 m21 = glm::mat4( 1. );    // identity
m21 = glm::translate(m21, glm::vec3(2.*Arm1.armScale, 0., 0.));
m21 = glm::rotate(m21, rot2, zaxis);                 // [T]*[R]
m21 = glm::translate(m21, glm::vec3(0., 0., 2.));    // z-offset from previous arm

glm::mat4 m32 = glm::mat4( 1. );    // identity
m32 = glm::translate(m32, glm::vec3(2.*Arm2.armScale, 0., 0.));
m32 = glm::rotate(m32, rot3, zaxis);                 // [T]*[R]
m32 = glm::translate(m32, glm::vec3(0., 0., 2.));    // z-offset from previous arm

Arm1.armMatrix = m1g;                    // m1g
Arm2.armMatrix = m1g * m21;              // m2g
Arm3.armMatrix = m1g * m21 * m32;        // m3g
```


In the *RenderScene( )* Function

268

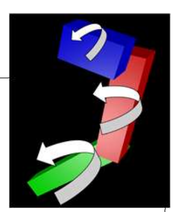
```
VkBuffer buffers[1]   = { MyVertexDataBuffer.buffer };

vkCmdBindVertexBuffers( CommandBuffers[nextImageIndex], 0, 1, buffers, offsets );

vkCmdPushConstants( CommandBuffers[nextImageIndex], GraphicsPipelineLayout,
            VK_SHADER_STAGE_ALL, 0, sizeof(struct arm), (void *)&Arm1 );
vkCmdDraw( CommandBuffers[nextImageIndex], vertexCount, instanceCount, firstVertex, firstInstance );

vkCmdPushConstants( CommandBuffers[nextImageIndex], GraphicsPipelineLayout,
            VK_SHADER_STAGE_ALL, 0, sizeof(struct arm), (void *)&Arm2 );
vkCmdDraw( CommandBuffers[nextImageIndex], vertexCount, instanceCount, firstVertex, firstInstance );

vkCmdPushConstants( CommandBuffers[nextImageIndex], GraphicsPipelineLayout,
            VK_SHADER_STAGE_ALL, 0, sizeof(struct arm), (void *)&Arm3 );
vkCmdDraw( CommandBuffers[nextImageIndex], vertexCount, instanceCount, firstVertex, firstInstance );
```

The strategy is to draw each link using the same vertex buffer, but modified with a unique color, length, and matrix transformation



In the Vertex Shader

269

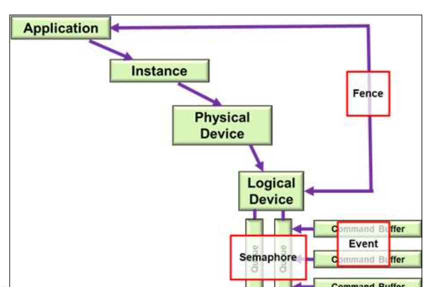


```
layout( push_constant ) uniform arm
{
        mat4  armMatrix;
        vec3  armColor;
        float   armScale;       // scale factor in x
} RobotArm;

layout( location = 0 ) in vec3 aVertex;

        . . .

vec3 bVertex = aVertex;                      // arm coordinate system is [-1., 1.] in X
bVertex.x += 1.;                             // now is [0., 2.]
bVertex.x  /=  2.;                           // now is [0., 1.]
bVertex.x *= (RobotArm.armScale );           // now is [0., RobotArm.armScale]
bVertex = vec3(  RobotArm.armMatrix * vec4( bVertex, 1. ) );

        . . .

gl_Position = PVMM * vec4( bVertex, 1. );    // Projection * Viewing * Modeling matrices
```


270

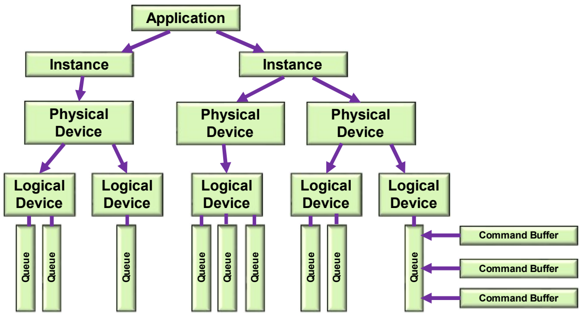
Vulkan.

Synchronization



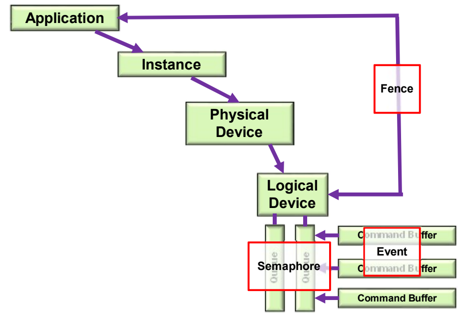
Remember the Overall Block Diagram?

271



Where Synchronization Fits in the Overall Block Diagram

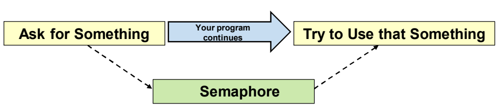
272



Semaphores

273

- Indicates that a batch of commends has been processed from a queue. Basically announces "I am finished!".
- You create one and give it to a Vulkan function which sets it. Later on, you tell another Vulkan function to wait for this semaphore to be signaled.
- You don't end up setting, resetting, or checking the semaphore yourself.
- Semaphores must be initialized ("created") before they can be used.



Creating a Semaphore

274

```
VkSemaphoreCreateInfo            vsci;
        vsci.sType = VK_STRUCTURE_TYPE_SEMAPHORE_CREATE_INFO;
        vsci.pNext = nullptr;
        vsci.flags = 0;;

VkSemaphore          semaphore;
result = vkCreateSemaphore( LogicalDevice, IN &vsci, PALLOCATOR, OUT &semaphore );
```

This doesn't actually do anything with the semaphore – it just sets it up

Semaphores Example during the Render Loop

275

```
VkSemaphore imageReadySemaphore;

VkSemaphoreCreateInfo            vsci;
            vsci.sType = VK_STRUCTURE_TYPE_SEMAPHORE_CREATE_INFO;
            vsci.pNext = nullptr;
            vsci.flags = 0;

result = vkCreateSemaphore( LogicalDevice, IN &vsci, PALLOCATOR, OUT &imageReadySemaphore );

uint32_t nextImageIndex;
vkAcquireNextImageKHR( LogicalDevice, IN SwapChain, IN UINT64_MAX,
                              IN imageReadySemaphore, IN VK_NULL_HANDLE, OUT &nextImageIndex );
      . . .

VkPipelineStageFlags waitAtBottomOfPipe = VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT;
VkSubmitInfo                 vsi;
            vsi.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
            vsi.pNext = nullptr;
            vsi.waitSemaphoreCount = 1;
            vsi.pWaitSemaphores = &imageReadySemaphore;
            vsi.pWaitDstStageMask = &waitAtBottomOfPipe;
            vsi.commandBufferCount = 1;
            vsi.pCommandBuffers = &CommandBuffers[nextImageIndex];
            vsi.signalSemaphoreCount = 0;
            vsi.pSignalSemaphores = (VkSemaphore) nullptr;

result = vkQueueSubmit( presentQueue, 1, IN &vsi, IN renderFence );
```

Set the semaphore

Wait on the semaphore

You do this to wait for an image to be ready to be rendered into

Fences

276

- Used to synchronize CPU-GPU tasks.
- Used when the host needs to wait for the device to complete something big.
- Announces that queue-submitted work is finished.
- You can un-signal, signal, test or block-while-waiting.

Fences

277

```
#define VK_FENCE_CREATE_UNSIGNALED_BIT          0

VkFenceCreateInfo                vfci;
        vfci.sType = VK_STRUCTURE_TYPE_FENCE_CREATE_INFO;
        vfci.pNext = nullptr;
        vfci.flags = VK_FENCE_CREATE_UNSIGNALED_BIT;           // = 0
                // VK_FENCE_CREATE_SIGNALED_BIT is only other option

VkFence            fence;
result = vkCreateFence( LogicalDevice, IN &vfci, PALLOCATOR, OUT &fence );

        , , ,

// returns to the host right away:
result = vkGetFenceStatus( LogicalDevice, IN fence );
                // result = VK_SUCCESS means it has signaled
                // result = VK_NOT_READY means it has not signaled


// blocks the host from executing:
result = vkWaitForFences( LogicalDevice, 1, IN &fence, waitForAll, timeout );
               // waitForAll = VK_TRUE:   wait for all fences in the list
               // waitForAll = VK_FALSE: wait for any one fence in the list
               // timeout is a uint64_t timeout in nanoseconds  (could be 0, which means to return immediately)
               // timeout can be up to UINT64_MAX  = 0xffffffffffffffff  (= 580+ years)
               // result = VK_SUCCESS means it returned because a fence  (or all fences) signaled
               // result = VK_TIMEOUT means it returned because the timeout was exceeded
```

Set the fence

Wait on the fence(s)

Fence Example

278

```
VkFence                          renderFence;
vkCreateFence( LogicalDevice, &vfci, PALLOCATOR, OUT &renderFence );

VkPipelineStageFlags waitAtBottom = VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT;

VkQueue presentQueue;
vkGetDeviceQueue( LogicalDevice, FindQueueFamilyThatDoesGraphics( ), 0, OUT &presentQueue );

VkSubmitInfo                   vsi;
            vsi.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
            vsi.pNext = nullptr;
            vsi.waitSemaphoreCount = 1;
            vsi.pWaitSemaphores = &imageReadySemaphore;
            vsi.pWaitDstStageMask = &waitAtBottom;
            vsi.commandBufferCount = 1;
            vsi.pCommandBuffers = &CommandBuffers[nextImageIndex];
            vsi.signalSemaphoreCount = 0;
            vsi.pSignalSemaphores = (VkSemaphore) nullptr;

result = vkQueueSubmit( presentQueue, 1, IN &vsi, IN renderFence );

        . . .

result = vkWaitForFences( LogicalDevice, 1, IN &renderFence, VK_TRUE, UINT64_MAX );

        . . .

result = vkQueuePresentKHR( presentQueue, IN &vpi );            // don't present the image until done rendering
```


Events

279

- Events provide even finer-grained synchronization.
- Events are a primitive that can be signaled by the host or the device.
- Can even signal at one place in the pipeline and wait for it at another place in the pipeline.
- Signaling in the pipeline means "signal me as the last piece of this draw command passes that point in the pipeline".
- You can signal, un-signal, or test from a vk function or from a vkCmd function.
- Can wait from a vkCmd function.

Controlling Events from the Host

280

```
VkEventCreateInfo                 veci;
        veci.sType = VK_STRUCTURE_TYPE_EVENT_CREATE_INFO;
        veci.pNext = nullptr;
        veci.flags = 0;

VkEvent            event;
result = vkCreateEvent( LogicalDevice, IN &veci, PALLOCATOR, OUT &event );

result = vkSetEvent( LogicalDevice, IN event );

result = vkResetEvent( LogicalDevice, IN event );

result = vkGetEventStatus( LogicalDevice, IN event );
            // result = VK_EVENT_SET: signaled
            // result = VK_EVENT_RESET: not signaled
```

Note: the host cannot *block* waiting for an event, but it can test for it

Controlling Events from the Device

281

```
result = vkCmdSetEvent(     CommandBuffer, IN event, pipelineStageBits  );

result = vkCmdResetEvent( CommandBuffer, IN event, pipelineStageBits  );

result = vkCmdWaitEvents( CommandBuffer, 1, &event,
            srcPipelineStageBits, dstPipelineStageBits,
              memoryBarrierCount, pMemoryBarriers,
              bufferMemoryBarrierCount, pBufferMemoryBarriers,
              imageMemoryBarrierCount, pImageMemoryBarriers
        );
```

Could be an array of events

Where signaled, where wait for the signal

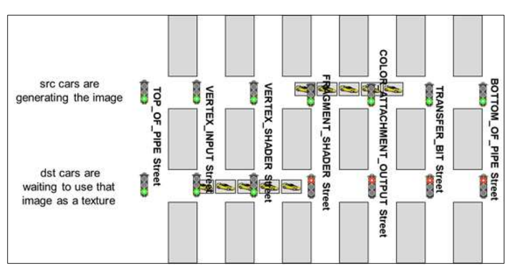
Memory barriers get executed after events have been signaled

Note: the device cannot *test* for an event, but it can block

282

Vulkan.

Pipeline Barriers



Why Do We Need Pipeline Barriers?

283

A series of vkCmdxxx( ) calls are meant to run "flat-out", that is, as fast as the Vulkan runtime can get them executing. But, many times, that is not desirable because the output of one command might be needed as the input to a subsequent command.

Pipeline Barriers solve this problem by declaring which stages of the hardware pipeline in subsequent vkCmdyyy( ) calls need to wait until which stages in previous vkCmdxxx( ) calls are completed.

SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Potential Memory Race Conditions that Pipeline Barriers can Prevent

284

1. Read-after-Write (R-a-W) – the memory write in one operation starts overwriting the memory that another operation's read needs to use.
2. Write-after-Read (W-a-R) – the memory read in one operation hasn't yet finished before another operation starts overwriting that memory.
3. Write-after-Write (W-a-W) – two operations start overwriting the same memory and the end result is non-deterministic.

Note: there is no problem with Read-after-Read (R-a-R) as no data gets changed.

SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

vkCmdPipelineBarrier( ) Function Call

285

A **Pipeline Barrier** is a way to establish a dependency between commands that were submitted before the barrier and commands that are submitted after the barrier

```
vkCmdPipelineBarrier( commandBuffer,
        srcStageMask,
        dstStageMask,
        VK_DEPENDENCY_BY_REGION_BIT,
        memoryBarrierCount,          pMemoryBarriers,
        bufferMemoryBarrierCount,    pBufferMemoryBarriers,
        imageMemoryBarrierCount,     pImageMemoryBarriers
);
```

srcStageMask: Guarantee that *this* pipeline stage is completely done being used by the previous vkCmdxxx before …

dstStageMask: … allowing *this* pipeline stage to be used by the next vkCmdyyy

Defines what data we will be blocking on or un-blocking on

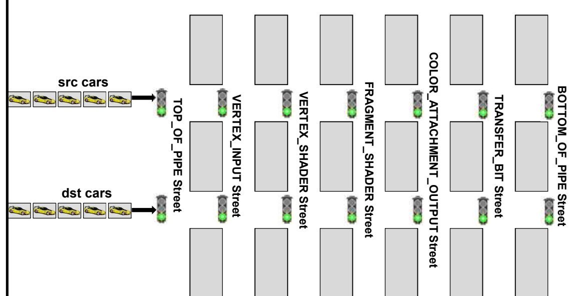
The hope is maximize the number of unblocked stages: produce data *early* and consume date *late*

SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

The Scenario

286



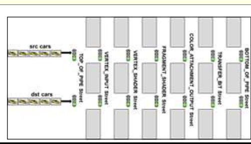
SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

The Scenario

287

1. The cross-streets are named after pipeline stages
2. All traffic lights start out green
3. There are special sensors at all intersections that will know when ***any car in the src group*** is in that intersection
4. There are connections from those sensors to the traffic lights so that when ***any car in the src group*** is in the intersection, the proper ***dst*** traffic lights will be turned red
5. When the ***last car in the src group*** completely makes it through its intersection, the proper ***dst*** traffic lights are turned back to green
6. The Vulkan command pipeline ordering is this: (1) the **src** cars get released by the previous vkCmdxxx, (2) the pipeline barrier is invoked (which turns some lights red), (3) the dst cars get released by the next vkCmdyyy, (4) the **dst** cars stop at the red light, (5) the **src** cars clear the intersection, (6) the dst lights turn green, (6) the **dst** cars continue.



SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Pipeline Stage Masks – Where in the Pipeline is this Memory Data being Generated or Consumed?

288

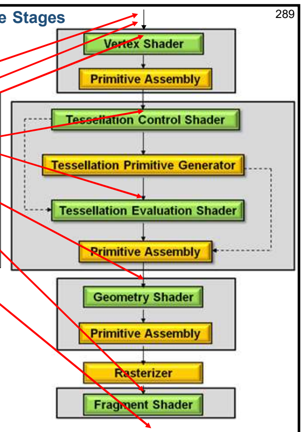
VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT
VK_PIPELINE_STAGE_DRAW_INDIRECT_BIT
VK_PIPELINE_STAGE_VERTEX_INPUT_BIT
VK_PIPELINE_STAGE_VERTEX_SHADER_BIT
VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT
VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT
VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT
VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT
VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT
VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT
VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT
VK_PIPELINE_STAGE_COMPUTE_SHADER_BIT
VK_PIPELINE_STAGE_TRANSFER_BIT
VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT
VK_PIPELINE_STAGE_HOST_BIT
VK_PIPELINE_STAGE_ALL_GRAPHICS_BIT
VK_PIPELINE_STAGE_ALL_COMMANDS_BIT

SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Pipeline Stages

- VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT
- VK_PIPELINE_STAGE_DRAW_INDIRECT_BIT
- VK_PIPELINE_STAGE_VERTEX_INPUT_BIT
- VK_PIPELINE_STAGE_VERTEX_SHADER_BIT
- VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT
- VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT
- VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT
- VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT
- VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT
- VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT
- VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT
- VK_PIPELINE_STAGE_TRANSFER_BIT
- VK_PIPELINE_STAGE_COMPUTE_SHADER_BIT
- VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT
- VK_PIPELINE_STAGE_HOST_BIT
- VK_PIPELINE_STAGE_ALL_GRAPHICS_BIT
- VK_PIPELINE_STAGE_ALL_COMMANDS_BIT



Access Masks – What are you Interested in Generating or Consuming this Memory for?

- VK_ACCESS_INDIRECT_COMMAND_READ_BIT
- VK_ACCESS_INDEX_READ_BIT
- VK_ACCESS_VERTEX_ATTRIBUTE_READ_BIT
- VK_ACCESS_UNIFORM_READ_BIT
- VK_ACCESS_INPUT_ATTACHMENT_READ_BIT
- VK_ACCESS_SHADER_READ_BIT
- VK_ACCESS_SHADER_WRITE_BIT
- VK_ACCESS_COLOR_ATTACHMENT_READ_BIT
- VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT
- VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_READ_BIT
- VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT
- VK_ACCESS_TRANSFER_READ_BIT
- VK_ACCESS_TRANSFER_WRITE_BIT
- VK_ACCESS_HOST_READ_BIT
- VK_ACCESS_HOST_WRITE_BIT
- VK_ACCESS_MEMORY_READ_BIT
- VK_ACCESS_MEMORY_WRITE_BIT

Pipeline Stages and what Access Operations are Allowed

		VK_ACCESS_INDIRECT_COMMAND_READ_BIT	VK_ACCESS_INDEX_READ_BIT	VK_ACCESS_VERTEX_ATTRIBUTE_READ_BIT	VK_ACCESS_UNIFORM_READ_BIT	VK_ACCESS_INPUT_ATTACHMENT_READ_BIT	VK_ACCESS_SHADER_READ_BIT	VK_ACCESS_SHADER_WRITE_BIT	VK_ACCESS_COLOR_ATTACHMENT_READ_BIT	VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT	VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_READ_BIT	VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT	VK_ACCESS_TRANSFER_READ_BIT	VK_ACCESS_TRANSFER_WRITE_BIT	VK_ACCESS_HOST_READ_BIT	VK_ACCESS_HOST_WRITE_BIT	VK_ACCESS_MEMORY_READ_BIT	VK_ACCESS_MEMORY_WRITE_BIT
1	VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT																	
2	VK_PIPELINE_STAGE_DRAW_INDIRECT_BIT	•																
3	VK_PIPELINE_STAGE_VERTEX_INPUT_BIT		•	•														
4	VK_PIPELINE_STAGE_VERTEX_SHADER_BIT				•		•	•										
5	VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT				•		•	•										
6	VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT				•		•	•										
7	VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT				•		•	•										
8	VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT										•	•						
9	VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT				•	•	•	•										
10	VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT										•	•						
11	VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT								•	•								
12	VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT																	
	VK_PIPELINE_STAGE_COMPUTE_SHADER				•		•	•										
	VK_PIPELINE_STAGE_TRANSFER_BIT												•	•				
	VK_PIPELINE_STAGE_HOST_BIT														•	•		

Access Operations and what Pipeline Stages they can be used In

	1	2	3	4	5	6	7	8	9	10	11	12			
	VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT	VK_PIPELINE_STAGE_DRAW_INDIRECT_BIT	VK_PIPELINE_STAGE_VERTEX_INPUT_BIT	VK_PIPELINE_STAGE_VERTEX_SHADER_BIT	VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT	VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT	VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT	VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT	VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT	VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT	VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT	VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT	VK_PIPELINE_STAGE_COMPUTE_SHADER	VK_PIPELINE_STAGE_TRANSFER_BIT	VK_PIPELINE_STAGE_HOST_BIT
VK_ACCESS_INDIRECT_COMMAND_READ_BIT		•													
VK_ACCESS_INDEX_READ_BIT			•												
VK_ACCESS_VERTEX_ATTRIBUTE_READ_BIT			•												
VK_ACCESS_UNIFORM_READ_BIT				•	•	•	•		•				•		
VK_ACCESS_INPUT_ATTACHMENT_READ_BIT									•						
VK_ACCESS_SHADER_READ_BIT				•	•	•	•		•				•		
VK_ACCESS_SHADER_WRITE_BIT				•	•	•	•		•				•		
VK_ACCESS_COLOR_ATTACHMENT_READ_BIT											•				
VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT											•				
VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_READ_BIT								•		•					
VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT								•		•					
VK_ACCESS_TRANSFER_READ_BIT														•	
VK_ACCESS_TRANSFER_WRITE_BIT														•	
VK_ACCESS_HOST_READ_BIT															•
VK_ACCESS_HOST_WRITE_BIT															•
VK_ACCESS_MEMORY_READ_BIT															
VK_ACCESS_MEMORY_WRITE_BIT															

Example: Be sure we are done writing an Output image before using it as a Fragment Shader Texture

Stages

- VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT
- VK_PIPELINE_STAGE_DRAW_INDIRECT_BIT
- VK_PIPELINE_STAGE_VERTEX_INPUT_BIT
- VK_PIPELINE_STAGE_VERTEX_SHADER_BIT
- VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT
- VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT
- VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT
- **VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT** ← src, dst
- VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT
- VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT
- VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT
- VK_PIPELINE_STAGE_COMPUTE_SHADER_BIT
- VK_PIPELINE_STAGE_TRANSFER_BIT
- VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT
- VK_PIPELINE_STAGE_HOST_BIT
- VK_PIPELINE_STAGE_ALL_GRAPHICS_BIT
- VK_PIPELINE_STAGE_ALL_COMMANDS_BIT

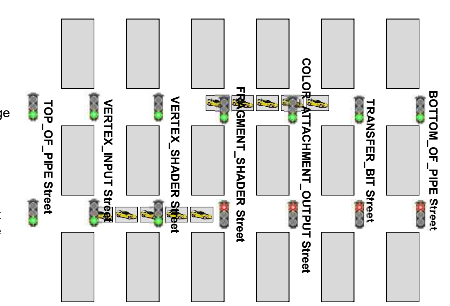
Access types

- VK_ACCESS_INDIRECT_COMMAND_READ_BIT
- VK_ACCESS_INDEX_READ_BIT
- VK_ACCESS_VERTEX_ATTRIBUTE_READ_BIT
- VK_ACCESS_UNIFORM_READ_BIT
- VK_ACCESS_INPUT_ATTACHMENT_READ_BIT
- **VK_ACCESS_SHADER_READ_BIT** ← dst
- **VK_ACCESS_SHADER_WRITE_BIT** ← src
- VK_ACCESS_COLOR_ATTACHMENT_READ_BIT
- VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT
- VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_READ_BIT
- VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT
- VK_ACCESS_TRANSFER_READ_BIT
- VK_ACCESS_TRANSFER_WRITE_BIT
- VK_ACCESS_HOST_READ_BIT
- VK_ACCESS_HOST_WRITE_BIT
- VK_ACCESS_MEMORY_READ_BIT
- VK_ACCESS_MEMORY_WRITE_BIT

Example: The Scenario

src cars are generating the image

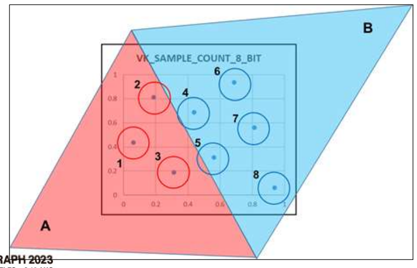
dst cars are waiting to use that image as a texture



295

Vulkan

Antialiasing and Multisampling

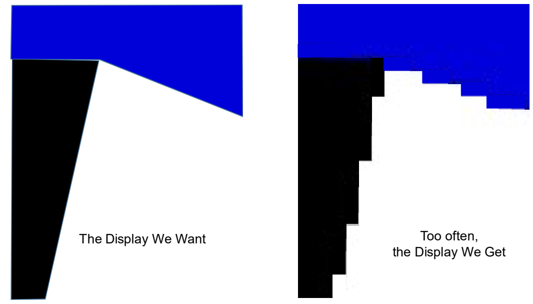


SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

296

Aliasing



SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

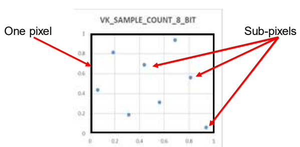
297

MultiSampling

Oversampling is a computer graphics technique to improve the quality of your output image by looking inside every pixel to see what the rendering is doing there.

There are two approaches to this:

1. **Supersampling**: Pick some number of sub-pixels within that pixel that pass the depth and stencil tests. Render the image at each of these sub-pixels. **Results in the best image,but the most rendering time.**



2. **Multisampling**: Pick some number of sub-pixels within that pixel that pass the depth and stencil tests. If any of them pass, then perform a single color render for the one pixel and assign that single color to all the sub-pixels that passed the depth and stencil tests. **Results in a good image, with less rendering time.**

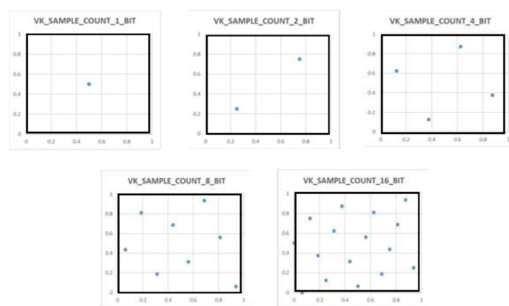
The final step is to average those sub-pixels' colors to produce one final color for this whole pixel. This is called ***resolving*** the pixel.

SIGGRAPH 20
LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

298

Vulkan Specification Distribution of Sampling Points within a Pixel



SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

299

Vulkan Specification Distribution of Sampling Points within a Pixel

VK_SAMPLE_COUNT_2_BIT	VK_SAMPLE_COUNT_4_BIT	VK_SAMPLE_COUNT_8_BIT	VK_SAMPLE_COUNT_16_BIT
		(0.5625, 0.3125)	(0.5625, 0.5625)
	(0.375, 0.125)		(0.4375, 0.3125)
		(0.4375, 0.6875)	(0.3125, 0.625)
			(0.75, 0.4375)
(0.25,0.25)		(0.8125, 0.5625)	(0.1875, 0.375)
	(0.875, 0.375)		(0.625, 0.8125)
		(0.3125, 0.1875)	(0.8125, 0.6875)
			(0.6875, 0.1875)
		(0.1875, 0.8125)	(0.375, 0.875)
	(0.125, 0.625)		(0.5, 0.0625)
		(0.0625, 0.4375)	(0.25, 0.125)
(0.75,0.75)			(0.125, 0.75)
		(0.6875, 0.9375)	(0.0, 0.5)
	(0.625, 0.875)		(0.9375, 0.25)
		(0.9375, 0.0625)	(0.875, 0.9375)
			(0.0625, 0.0)

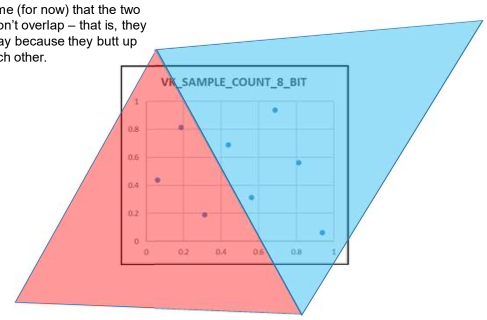
SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

300

Consider Two Triangles That Pass Through the Same Pixel

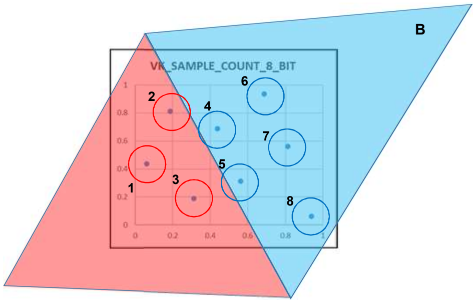
Let's assume (for now) that the two triangles don't overlap – that is, they look this way because they butt up against each other.



SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

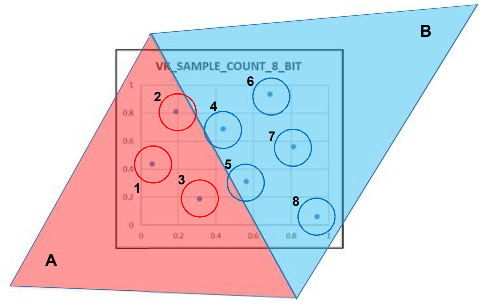
Supersampling



$$Final\ Pixel\ Color = \frac{\sum_{i=1}^{8} Color\ sample\ from\ subpixel_i}{8}$$

 # Fragment Shader calls = 8

Multisampling

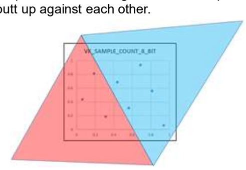


$$Final\ Pixel\ Color = \frac{3 * One\ color\ sample\ from\ A\ +\ 5 * One\ color\ sample\ from\ B}{8}$$

 # Fragment Shader calls = 2

Consider Two Triangles Who Pass Through the Same Pixel

Let's assume (for now) that the two triangles don't overlap – that is, they look this way because they butt up against each other.

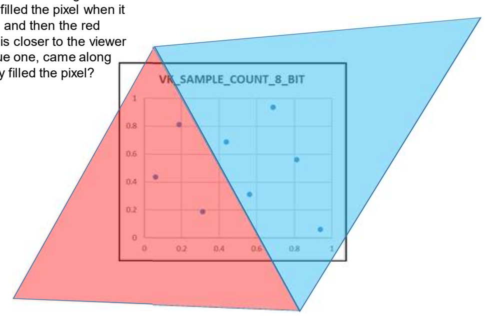


Number of Fragment Shader Calls

	Multisampling	Supersampling
Blue fragment shader calls	1	5
Red fragment shader calls	1	3

Consider Two Triangles Who Pass Through the Same Pixel

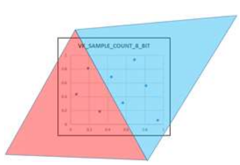
Q: What if the blue triangle completely filled the pixel when it was drawn, and then the red one, which is closer to the viewer than the blue one, came along and partially filled the pixel?



A: The ideas are all still the same, but the blue one had to deal with 8 sub-pixels (instead of 5 like before). But, the red triangle came along and obsoleted 3 of those blue sub-pixels. Note that the "resolved" image will still turn out the same as before.

Consider Two Triangles Who Pass Through the Same Pixel

What if the blue triangle completely filled the pixel when it was drawn, and then the red one, which is closer to the viewer than the blue one, came along and partially filled the pixel?



Number of Fragment Shader Calls

	Multisampling	Supersampling
Blue fragment shader calls	1	8
Red fragment shader calls	1	3

Setting up the Image

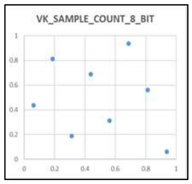

```
VkPipelineMultisampleStateCreateInfo            vpmsci;
        vpmsci.sType = VK_STRUCTURE_TYPE_PIPELINE_MULTISAMPLE_STATE_CREATE_INFO;
        vpmsci.pNext = nullptr;
        vpmsci.flags = 0;
        vpmsci.rasterizationSamples = VK_SAMPLE_COUNT_8_BIT;
        vpmsci.sampleShadingEnable = VK_TRUE;
        vpmsci.minSampleShading = 0.5f;
        vpmsci.pSampleMask = (VkSampleMask *)nullptr;
        vpmsci.alphaToCoverageEnable = VK_FALSE;
        vpmsci.alphaToOneEnable = VK_FALSE;

 VkGraphicsPipelineCreateInfo           vgpci;
        vgpci.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
        vgpci.pNext = nullptr;
    . . .
        vgpci.pMultisampleState = &vpmsci;

result = vkCreateGraphicsPipelines( LogicalDevice, VK_NULL_HANDLE, 1, IN &vgpci,\
                        PALLOCATOR, OUT pGraphicsPipeline );
```

How dense is the sampling

VK_TRUE means to allow some sort of multisampling to take place

Setting up the Image

307




```
VkPipelineMultisampleStateCreateInfo          vpmsci;
    . . .

    vpmsci.minSampleShading = 0.5;

    . . .
```

At least this fraction of samples will get their own fragment shader calls (as long as they pass the depth and stencil tests).

0. produces simple multisampling

(0. - 1.) produces partial supersampling

1. Produces complete supersampling

SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Setting up the Image

308

```
VkAttachmentDescription                 vad[2];
        vad[0].format = VK_FORMAT_B8G8R8A8_SRGB;                  // 24-bit color
        vad[0].samples = VK_SAMPLE_COUNT_8_BIT;
        vad[0].loadOp = VK_ATTACHMENT_LOAD_OP_CLEAR;
        vad[0].storeOp = VK_ATTACHMENT_STORE_OP_STORE;
        vad[0].stencilLoadOp = VK_ATTACHMENT_LOAD_OP_DONT_CARE;
        vad[0].stencilStoreOp = VK_ATTACHMENT_STORE_OP_DONT_CARE;
        vad[0].initialLayout = VK_IMAGE_LAYOUT_UNDEFINED;
        vad[0].finalLayout = VK_IMAGE_LAYOUT_PRESENT_SRC_KHR;
        vad[0].flags = 0;

        vad[1].format = VK_FORMAT_D32_SFLOAT_S8_UINT;             // 32-bit floating-point depth
        vad[1].samples = VK_SAMPLE_COUNT_8_BIT;
        vad[1].loadOp = VK_ATTACHMENT_LOAD_OP_CLEAR;
        vad[1].storeOp = VK_ATTACHMENT_STORE_OP_DONT_CARE;
        vad[1].stencilLoadOp = VK_ATTACHMENT_LOAD_OP_DONT_CARE;
        vad[1].stencilStoreOp = VK_ATTACHMENT_STORE_OP_DONT_CARE;
        vad[1].initialLayout = VK_IMAGE_LAYOUT_UNDEFINED;
        vad[1].finalLayout = VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL;
        vad[1].flags = 0;

VkAttachmentReference                   colorReference;
        colorReference.attachment = 0;
        colorReference.layout = VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL;

VkAttachmentReference                   depthReference;
        depthReference.attachment = 1;
        depthReference.layout = VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL;
```

to next slide

Setting up the Image

309

from previous slide

```
VkSubpassDescription                    vsd;
        vsd.flags = 0;
        vsd.pipelineBindPoint = VK_PIPELINE_BIND_POINT_GRAPHICS;
        vsd.inputAttachmentCount = 0;
        vsd.pInputAttachments = (VkAttachmentReference *)nullptr;
        vsd.colorAttachmentCount = 1;
        vsd.pColorAttachments = &colorReference;
        vsd.pResolveAttachments = (VkAttachmentReference *)nullptr;
        vsd.pDepthStencilAttachment = &depthReference;
        vsd.preserveAttachmentCount = 0;
        vsd.pPreserveAttachments = (uint32_t *)nullptr;

VkRenderPassCreateInfo                  vrpci;
        vrpci.sType = VK_STRUCTURE_TYPE_RENDER_PASS_CREATE_INFO;
        vrpci.pNext = nullptr;
        vrpci.flags = 0;
        vrpci.attachmentCount = 2;          // color and depth/stencil
        vrpci.pAttachments = vad;
        vrpci.subpassCount = 1;
        vrpci.pSubpasses = IN &vsd;
        vrpci.dependencyCount = 0;
        vrpci.pDependencies = (VkSubpassDependency *)nullptr;

result = vkCreateRenderPass( LogicalDevice, IN &vrpci, PALLOCATOR, OUT &RenderPass );
```

SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Resolving the Image:
Converting the Multisampled Image to a VK_SAMPLE_COUNT_1_BIT image

310

```
VlOffset3D                              vo3;
        vo3.x = 0;
        vo3.y = 0;
        vo3.z = 0;

VkExtent3D                              ve3;
        ve3.width = Width;
        ve3.height = Height;
        ve3.depth = 1;

VkImageSubresourceLayers                visl;
        visl.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
        visl.mipLevel = 0;
        visl.baseArrayLayer = 0;
        visl.layerCount = 1;

VkImageResolve                          vir;
        vir.srcSubresource = visl;
        vir.srcOffset = vo3;
        vir.dstSubresource = visl;
        vir.dstOffset = vo3;
        vir.extent = ve3;

vkCmdResolveImage( cmdBuffer, srcImage, srcImageLayout,  dstImage, dstImageLayout, 1, IN &vir );
```

For the *ImageLayout, use VK_IMAGE_LAYOUT_GENERAL

SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

311

Vulkan.

Summary

SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Vulkan Program Flow – the Setup

312

- Create a GLFW Vulkan Window
- Query the Physical Devices and Choose (1 in our case)
- Decide on the Extensions and Layers You Want
- Create the Logical Device
- Create the Queue(s) (1 in our case)
- Allocate and Fill memory for the Vertices and Indices
- Allocate and Fill memory for the Uniform Buffers
- Create the Command Buffers (3 in our case)
- If using Textures, create the Sampler, Read the Texture, and move it to Device Local Memory
- Create the Swap Chain (2 images in our case)
- Be sure you have Compiled the Shaders into .spv files
- Create the Descriptor Set Data Structures
- Create the Graphics Pipeline Data Structure Layout(s)
- Fill the Graphics Pipeline Data Structure(s)

SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

Vulkan Program Flow – the Rendering Loop

313

while(the GLFW Window should not close)
{
UpdateScene( )
RenderScene( )
}

Create the Transformations
Fill the Uniform Buffers

Acquire the Next Swap Chain Image
Begin its Command Buffer
Create the RenderPass with the Framebuffer information
for(all the different Graphics Pipeline Data Structures being used)
{
Bind that Graphics Pipeline Data Structure
Set any Dynamic State Variables
Bind the Proper Descriptor Set Values
Do the Drawing
}
End the RenderPass
End the Command Buffer
Submit the Command Buffer to a Queue
Wait for the Queue to Finish Submitting
Present the Image to the Viewer

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

mjb – June 5, 2023

So What Do We All Do Now?

314

- 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
- And, I also wonder if vendors will become less and less excited about improving OpenGL drivers

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

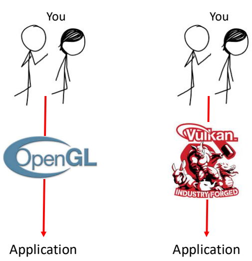
mjb – June 5, 2023

So What Do We All Do Now?

315

- Performance-uncritical

- Performance-critical
- Need ray-tracing



xkcd.com

SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

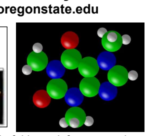
mjb – June 5, 2023

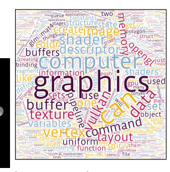
SIGGRAPH 2023 LOS ANGELES+ 6-10 AUG

316

The Vulkan Computer Graphics API

Mike Bailey
mjb@cs.oregonstate.edu





SIGGRAPH '23 Courses, August 06-10, 2023, Los Angeles, CA, USA
ACM 979-8-4007-0145-0/23/08.
10.1145/3587423.3595529

http://cs.oregonstate.edu/~mjb/vulkan

mjb – June 5, 2023