The Shaders’ View of the Basic Computer Graphics Pipeline

- In general, you want 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.
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;
Vulkan: GLSL Differences from OpenGL

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

• In the compiler, there is an automatic
  #define VULKAN 100

Vertex and Instance indices:

  gl_VertexIndex
  gl_InstanceIndex

• Both are 0-based

gl_FragColor:

• In OpenGL, it broadcasts to all color attachments
• In Vulkan, it just broadcasts to color attachment location #0
• Best idea: don’t use it – explicitly declare out variables to have specific location numbers
Shader combinations of separate texture data and samplers:
    uniform sampler s;
    uniform texture2D t;
    vec4 rgba = texture( sampler2D( t, s ), vST );

Descriptor Sets:
    layout( set=0, binding=0 ) . . . ;

Push Constants:
    layout( push_constant) . . . ;

Specialization Constants:
    layout( constant_id = 3 ) const int N = 5;

• Can only use basic operators, declarations, and constructors
• Only for scalars, but a vector can be constructed from specialization constants

Specialization Constants for Compute Shaders:
    layout( local_size_x_id = 8, local_size_y_id = 16 );

• gl_WorkGroupSize.z is still as it was
Vulkan: Shaders’ use of Layouts for Uniform Variables

layout( std140, set = 0, binding = 0 ) uniform matBuf
{
    mat4 uModelMatrix;
    mat4 uViewMatrix;
    mat4 uProjectionMatrix;
    mat3 uNormalMatrix;
} Matrices;

// non-opaque must be in a uniform block:
layout( std140, set = 1, binding = 0 ) uniform lightBuf
{
    vec4 uLightPos;
} Light;

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

All opaque (non-sampler) uniform variables must be in block buffers
Vulkan Shader Compiling

- You pre-compile your shaders with an external compiler
- Your shaders get turned into an intermediate form known as SPIR-V
- SPIR-V gets turned into fully-compiled code at runtime
- SPIR-V spec has been public for a couple of years – new shader languages are surely being developed
- OpenGL and OpenCL will be moving to 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
The first open standard intermediate language for parallel compute and graphics:

- SPIR (Standard Portable Intermediate Representation) was initially developed for use by OpenCL and SPIR versions 1.2 and 2.0 were based on LLVM. SPIR has now evolved into a true cross-API standard that is fully defined by Khronos with native support for shader and kernel features – called SPIR-V.

- SPIR-V is the first open standard, cross-API intermediate language for natively representing parallel compute and graphics and is incorporated as part of the core specification of both OpenCL 2.1 and OpenCL 2.2 and the new Vulkan graphics and compute API.

- SPIR-V exposes the machine model for OpenCL 1.2, 2.0, 2.1, 2.2 and Vulkan - including full flow control, and graphics and parallel constructs not supported in LLVM. SPIR-V also supports OpenCL C and OpenCL C++ kernel languages as well as the GLSL shader language for Vulkan.

- SPIR-V 1.1, launched in parallel with OpenCL 2.2, now supports all the kernel language features of OpenCL C++ in OpenCL 2.2, including initializer and finalizer function execution modes to support constructors and destructors. SPIR-V 1.1 also enhances the expressiveness of kernel programs by supporting named barriers, subgroup execution, and program scope pipes.

- SPIR-V is catalyzing a revolution in the language compiler ecosystem - it can split the compiler chain across multiple vendors’ products, enabling high-level language front-ends to emit programs in a standardized intermediate form to be ingested by Vulkan or OpenCL drivers. For hardware vendors, ingesting SPIR-V eliminate the need to build a high-level language source compiler into device drivers, significantly reducing driver complexity, and will enable a broad range of language and framework front-ends to run on diverse hardware architectures.

- For developers, using SPIR-V means that kernel source code no longer has to be directly exposed, kernel load times can be accelerated and developers can choose the use of a common language front-end, improving kernel reliability and portability across multiple hardware implementations.

https://www.khronos.org/spir
SPIR-V:
Standard Portable Intermediate Representation for Vulkan


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:   setenv LD_LIBRARY_PATH /usr/local/common/gcc-6.3.0/lib64/
You Can Run the SPIR-V Compiler on Windows from a Bash Shell

1. Click on the Microsoft Start icon
2. Type word `bash`
You Can Run the SPIR-V Compiler on Windows from a Bash Shell

Pick one:

- Can get to your personal folders
- Does not have make

- Cannot get to your personal folders
- Does have make
Running glslangValidator.exe

```
MINGW64:/y/Vulkan/Sample2017

ONID+mb@pooh MINGW64 /y/Vulkan/Sample2017
$ !85

glslangValidator.exe -V sample-vert.vert -o sample-vert.spv
sample-vert.vert

ONID+mb@pooh MINGW64 /y/Vulkan/Sample2017
$ !86

glslangValidator.exe -V sample-frag.frag -o sample-frag.spv
sample-frag.frag

ONID+mb@pooh MINGW64 /y/Vulkan/Sample2017
$
```
You can also run SPIR-V from a Linux Shell

```bash
$ glslangValidator.exe -V sample-vert.vert -o sample-vert.spv
$ glslangValidator.exe -V sample-frag.frag -o sample-frag.spv
```
You can also run SPIR-V from a Linux Shell

```
glslangValidator.exe -V sample-vert.vert -o sample-vert.spv
```

Compile for Vulkan ("-G" is compile for OpenGL)

The input file. The compiler determines the shader type by the file extension:

- `.vert` Vertex shader
- `.tccs` Tessellation Control Shader
- `.tecs` Tessellation Evaluation Shader
- `.geom` Geometry shader
- `.frag` Fragment shader
- `.comp` Compute shader

Specify the output file
How do you know if SPIR-V compiled successfully?

Same as C/C++ -- the compiler gives you no nasty messages.

Also, if you care, legal .spv files have a magic number of 0x07230203

So, if you do an `od -x` on the .spv file, the magic number looks like this:

0203 0723 . . .
VkResult
Init12SpirvShader( std::string filename, VkShaderModule * pShaderModule )
{
    FILE *fp;
    (void) fopen_s( &fp, filename.c_str(), "rb" );
    if( fp == NULL )
    {
        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

```c
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, pShaderModule );
fprintf( FpDebug, "Shader Module '%s' successfully loaded\n", filename.c_str() );
delete [] code;
return result;
```
Vulkan: Creating a Pipeline

vkCreateGraphicsPipeline

VkPipelineShaderStageCreateInfo

 VkSpecializationInfo

 which stage (VERTEX, etc.)

 VkShaderModule

 binding stride inputRate

 VkVertexInputBindingDescription

 location binding format offset

 VkVertexInputAttributeDescription

 Shader stages
 VertexInput State
 InputAssembly State
 Tessellation State
 Viewport State
 Rasterization State
 MultiSample State
 DepthStencil State
 ColorBlend State
 Dynamic State
 Pipeline layout
 RenderPass
 basePipelineHandle
 basePipelineIndex

 VkPipelineVertexInputStateCreateInfo

 VkPipelineVertexInputStateCreateInfo

 VkPipelineInputAssemblyStateCreateInfo

 VkViewportStateCreateInfo

 Viewport

 x, y, w, h, minDepth, maxDepth

 VkPipelineRasterizationStateCreateInfo

 Scissor

 cullMode polygonMode frontFace lineWidth

 VkPipelineDepthStencilStateCreateInfo

 VkPipelineColorBlendStateCreateInfo

 blendEnable srcColorBlendFactor dstColorBlendFactor colorBlendOp srcAlphaBlendFactor dstAlphaBlendFactor alphaBlendOp colorWriteMask

 depthTestEnable depthWriteEnable depthCompareOp stencilTestEnable stencilOpStateFront stencilOpStateBack

 VkPipelineColorBlendAttachmentState

 VkPipelineDynamicStateCreateInfo

 Array naming the states that can be set dynamically

 VkGraphicsPipelineCreateInfo

 Shader stages
 VertexInput State
 InputAssembly State
 Tessellation State
 Viewport State
 Rasterization State
 MultiSample State
 DepthStencil State
 ColorBlend State
 Dynamic State
 Pipeline layout
 RenderPass
 basePipelineHandle
 basePipelineIndex

 Vulkan: Creating a Pipeline

 vkCreateGraphicsPipeline
You can also take a look at SPIR-V Assembly

```
glslangValidator.exe  -V  -H sample-vert.vert  -o sample-vert.spv
```

This prints out the SPIR-V “assembly” to standard output. Other than nerd interest, there is no graphics-programming reason to look at this. 😊
#version 400
#extension GL_ARB_separate_shader_objects : enable
#extension GL_ARB_shading_language_420pack : enable
layout( std140, set = 0, binding = 0 ) uniform matBuf
{
    mat4 uModelMatrix;
    mat4 uViewMatrix;
    mat4 uProjectionMatrix;
    mat3 uNormalMatrix;
} Matrices;

// non-opaque must be in a uniform block:
layout( std140, set = 1, binding = 0 ) uniform lightBuf
{
    vec4 uLightPos;
} Light;

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

layout( location = 0 ) out vec3 vNormal;
layout( location = 1 ) out vec3 vColor;
layout( location = 2 ) out vec2 vTexCoord;

void
main( )
{
    mat4 PVM = Matrices.uProjectionMatrix * Matrices.uViewMatrix * Matrices.uModelMatrix;
    gl_Position = PVM * vec4( aVertex, 1. );

    vNormal = Matrices.uNormalMatrix * aNormal;
    vColor = aColor;
    vTexCoord = aTexCoord;
}
This is the SPIR-V Assembly, Part I

```spir-v
#version 400
#extension GL_ARB_separate_shader_objects : enable
#extension GL_ARB_shading_language_420pack : enable
layout( std140, set = 0, binding = 0 ) uniform matBuf
{
  mat4 uModelMatrix;
  mat4 uViewMatrix;
  mat4 uProjectionMatrix;
  mat3 uNormalMatrix;
} Matrices;

// non-opaque must be in a uniform block:
layout( std140, set = 1, binding = 0 ) uniform lightBuf
{
  vec4 uLightPos;
} Light;

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

void main( )
{
  mat4 PVM = Matrices.uProjectionMatrix * Matrices.uViewMatrix * Matrices.uModelMatrix;
  gl_Position = PVM * vec4( aVertex, 1. );
  vNormal = Matrices.uNormalMatrix * aNormal;
  vColor = aColor;
  vTexCoord = aTexCoord;
}
```
#version 400
#extension GL_ARB_separate_shader_objects : enable
#extension GL_ARB_shading_language_420pack : enable
layout( std140, set = 0, binding = 0 ) uniform matBuf
{
    mat4 uModelMatrix;
    mat4 uViewMatrix;
    mat4 uProjectionMatrix;
    mat3 uNormalMatrix;
} Matrices;

// non-opaque must be in a uniform block:
layout( std140, set = 1, binding = 0 ) uniform lightBuf
{
    vec4 uLightPos;
} Light;

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

void main( )
{
    mat4 PVM = Matrices.uProjectionMatrix * Matrices.uViewMatrix * Matrices.uModelMatrix;
    gl_Position = PVM * vec4( aVertex, 1. );
    vNormal = Matrices.uNormalMatrix * aNormal;
    vColor = aColor;
    vTexCoord = aTexCoord;
}

32(gl_PerVertex):   TypeStruct 7(fvec4) 6(float) 31
33: TypePointer Output 32(gl_PerVertex)
34: (ptr) Variable Output
36: TypePointer Input 11(fvec3)
37(aVertex):   36(ptr) Variable Input
39: 6(float) Constant 1065353216
45: TypePointer Output 7(fvec4)
47: TypePointer Output 11(fvec3)
48(vNormal): 47(ptr) Variable Output
49: 16(int) Constant 3

Decorate 15(Matrices) Binding 0
MemberDecorate 32(gl_PerVertex) 0 Builtln Position
MemberDecorate 32(gl_PerVertex) 1 Builtln PointSize
MemberDecorate 32(gl_PerVertex) 2 Builtln ClipDistance
Decorate 32(gl_PerVertex) Block
Decorate 37(aVertex) Location 0
Decorate 48(vNormal) Location 0
Decorate 53(aNormal) Location 1
Decorate 56(vColor) Location 1
Decorate 57(aColor) Location 2
Decorate 61(vTexCoord) Location 2
Decorate 63(aTexCoord) Location 3
MemberDecorate 65(lightBuf) 0 Offset 0
Decorate 65(lightBuf) Block
Decorate 67(Light) DescriptorSet 1
Decorate 67(Light) Binding 0

2: TypeVoid
3: TypeFunction 2
6: TypeFloat 32
7: TypeVector 6(float) 4
8: TypeMatrix 7(fvec4) 4
9: TypePointer Function 8
11: TypeVector 6(float) 3
12: TypeMatrix 11(fvec3) 3
13(matBuf): TypeStruct 8 8 8 12
14: TypePointer Uniform 13(matBuf)
15(Matrices): 14(ptr) Variable Uniform
16: TypeInt 32 1
17: 16(int) Constant 2
18: TypePointer Uniform 8
21: 16(int) Constant 1
25: 16(int) Constant 0
29: TypeInt 32 0
30: 29(int) Constant 1
31: TypeArray 6(float) 30

This is the SPIR-V Assembly, Part II
This is the SPIR-V Assembly, Part III

```assembly
#version 400
#extension GL_ARB_separate_shader_objects : enable
#extension GL_ARB_shading_language_420pack : enable
layout( std140, set = 0, binding = 0 ) uniform mat4Buf  {
  mat4 uModelMatrix;
  mat4 uViewMatrix;
  mat4 uProjectionMatrix;
  mat3 uNormalMatrix;
} Matrices;

// non-opaque must be in a uniform block
layout( std140, set = 1, binding = 0 ) uniform lightBuf  {
  vec4 uLightPos;
} Light;

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

void main()
{
  mat4 PVM = Matrices.uProjectionMatrix * Matrices.uViewMatrix * Matrices.uModelMatrix;
  gl_Position = PVM * vec4( aVertex, 1. );
  vNormal = Matrices.uNormalMatrix * aNormal;
  vColor = aColor;
  vTexCoord = aTexCoord;
}
```

50: TypePointer Uniform 12
53(aNormal): 36(ptr) Variable Input
56(vColor): 47(ptr) Variable Output
57(aColor): 36(ptr) Variable Input
59: TypeVector 6(float) 2
60: TypePointer Output 59(fvec2)
61(vTexCoord): 60(ptr) Variable Input
62: TypePointer Input 59(fvec2)
63(aTexCoord): 62(ptr) Variable Input
65(lightBuf): TypeStruct 7(fvec4)
66: TypePointer Uniform 65(lightBuf)
67(Light): 66(ptr) Variable Uniform
4(main): Function None 3
5: Label
10(PVM): 9(ptr) Variable Function
19: 18(ptr) AccessChain 15(Matrices) 17
20: 8 Load 19
22: 18(ptr) AccessChain 15(Matrices) 21
23: 8 Load 22
24: 8 MatrixTimesMatrix 20 23
26: 18(ptr) AccessChain 15(Matrices) 25
27: 8 Load 26
28: 8 MatrixTimesMatrix 24 27
35: 8 Load 10(PVM)
38: 11(fvec3) Load 37(aVertex)
40: 6(float) CompositeExtract 38 0
41: 6(float) CompositeExtract 38 1
42: 6(float) CompositeExtract 38 2
43: 7(fvec4) CompositeConstruct 40 41 42 39
44: 7(fvec4) MatrixTimesVector 35 43
46: 45(ptr) AccessChain 34 25
50(ptr) AccessChain 15(Matrices) 49
52: 12 Load 51
54: 11(fvec3) Load 53(aNormal)
55: 11(fvec3) MatrixTimesVector 52 54
58: 11(fvec3) Load 57(aColor)
59(fvec2) Load 63(aTexCoord)
64: 59(fvec2) Load 63(aTexCoord)
```
SPIR-V: Printing the Configuration

glslangValidator --c

MaxLights 32
MaxClipPlanes 6
MaxTextureUnits 32
MaxTextureCoords 32
MaxVertexAttribs 64
MaxVertexUniformComponents 4096
MaxVaryingFloats 64
MaxVertexTextureImageUnits 32
MaxCombinedTextureImageUnits 80
MaxTextureImageUnits 32
MaxFragmentUniformComponents 4096
MaxDrawBuffers 32
MaxVertexUniformVectors 128
MaxVaryingVectors 8
MaxFragmentUniformVectors 16
MaxVertexOutputVectors 16
MaxFragmentInputVectors 15
MinProgramTexelOffset -8
MaxProgramTexelOffset 7
MaxClipDistances 8
MaxComputation 65535
MaxComputeWorkGroupCountX 65535
MaxComputeWorkGroupCountY 65535
MaxComputeWorkGroupCountZ 65535
MaxComputeWorkGroupSizeX 1024
MaxComputeWorkGroupSizeY 1024
MaxComputeWorkGroupSizeZ 64
MaxComputeUniformComponents 1024
MaxComputeTextureImageUnits 16
MaxComputeAtomicCounters 128
MaxComputeAtomicCounterBuffers 1
MaxVaryingComponents 60
MaxVertexTextureImageUnits 64
MaxGeometryInputComponents 64
MaxGeometryOutputComponents 128
MaxFragmentInputComponents 128
MaxImageUnits 8
MaxCombinedImageUnitsAndFragmentOutputs 8
MaxCombinedShaderOutputResources 8
MaxImageSamples 0
MaxVertexImageUniforms 0
MaxTessControlImageUniforms 0
MaxTessEvaluationImageUniforms 0
MaxGeometryImageUniforms 0
MaxFragmentImageUniforms 81
MaxGeometryTextureImageUnits 16
MaxGeometryOutputVertices 256
MaxGeometryTotalOutputComponents 1024
MaxGeometryUniformComponents 1024
MaxGeometryVaryingComponents 64
MaxTessControlInputComponents 128
MaxTessControlOutputComponents 128
MaxTessControlTextureImageUnits 16
MaxTessControlUniformComponents 1024
MaxTessControlTotalOutputComponents 4096
MaxTessEvaluationInputComponents 128
MaxTessEvaluationOutputComponents 128
MaxTessEvaluationTextureImageUnits 16
MaxTessEvaluationUniformComponents 1024
MaxTessPatchComponents 120
MaxTessPatchVertices 32
MaxTessGenLevel 64
MaxViewports 16
MaxVertexAtomicCounters 0
MaxTessControlAtomicCounters 0
MaxTessEvaluationAtomicCounters 0
MaxGeometryAtomicCounters 0
MaxFragmentAtomicCounters 8
MaxCombinedAtomicCounters 8
MaxAtomicCounterBindings 1
MaxVertexAtomicCounterBuffers 0
MaxTessControlAtomicCounterBuffers 0
MaxTessEvaluationAtomicCounterBuffers 0
MaxGeometryAtomicCounterBuffers 0
MaxFragmentAtomicCounterBuffers 1
MaxCombinedAtomicCounterBuffers 1
MaxAtomicCounterBufferSize 16384
MaxTransformFeedbackBuffers 4
MaxTransformFeedbackInterleavedComponents 64
MaxCullDistances 8
MaxCombinedClipAndCullDistances 8
MaxSamples 4
nonInductiveForLoops 1
whileLoops 1
doWhileLoops 1
generalUniformIndexting 1
generalAttributeMatrixVectorIndexting 1
generalVaryingIndexting 1
generalSamplerIndexting 1
generalVariableIndexting 1
generalConstantMatrixVectorIndexting 1
SPIR-V: More Information

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