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.
Shader stages

```c
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`

**Vulkan Vertex and Instance indices:**

<table>
<thead>
<tr>
<th>Vulkan</th>
<th>OpenGL</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>gl_VertexIndex</code></td>
<td><code>gl_VertexID</code></td>
</tr>
<tr>
<td><code>gl_InstanceIndex</code></td>
<td><code>gl_InstanceID</code></td>
</tr>
</tbody>
</table>

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

Specialization Constants for 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: GLSL Differences from OpenGL

Vulkan: Shaders’ use of Layouts for Uniform Variables

All 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, which stands for **Standard Portable Intermediate Representation**.
- 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 have 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

```bash
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: `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 the word `bash`

Pick one:

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

- Can get to your personal folders
- Does have make

This is only available within 64-bit Windows 10.
Running glslangValidator.exe

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 . . .
```
#define SPIRV_MAGIC             0x07230203

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'
", 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
" , 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 );

    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
", filename.c_str() );
    delete [] code;
    return result;
}
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. 😊
For example, if this is your Shader Source

```glsl
#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

```
0:   OpCapability Shader
1:   OpRequirement GLSL.std.450
2:   OpMemoryModel Logical GLSL450
3:   OpEntryPoint Vertex 4 "main" 34 37 48 53 56 57 61 63
4:   OpSource GLSL 400
5:   OpSourceExtension "GL_ARB_separate_shader_objects"
6:   OpSourceExtension "GL_ARB_shading_language_420pack"
7:   OpName 4 "main"
8:   OpName 10 "PVM"
9:   OpName 13 "matBuf"
10:  OpMemberName 13(matBuf) 0 "uModelMatrix"
11:  OpMemberName 13(matBuf) 1 "uViewMatrix"
12:  OpMemberName 13(matBuf) 2 "uProjectionMatrix"
13:  OpMemberName 13(matBuf) 3 "uNormalMatrix"
14:  OpName 15 "Matrices"
15:  OpName 32 "gl_PerVertex"
16:  OpMemberName 32(gl_PerVertex) 0 "gl_Position"
17:  OpMemberName 32(gl_PerVertex) 1 "gl_PointSize"
18:  OpMemberName 32(gl_PerVertex) 2 "gl_ClipDistance"
19:  OpName 34 ...
20:  OpName 37 "aVertex"
21:  OpName 48 "vNormal"
22:  OpName 53 "aNormal"
23:  OpName 56 "vColor"
24:  OpName 57 "aColor"
25:  OpName 61 "vTexCoord"
26:  OpName 63 "aTexCoord"
27:  OpName 65 "lightBuf"
28:  OpMemberName 65(lightBuf) 0 "uLightPos"
29:  OpMemberDecorate 13(matBuf) Block
30:  OpDecorate 10(matBuf) 0 CoMajor
31:  OpDecorate 13(matBuf) 0 Offset 0
32:  OpDecorate 13(matBuf) 1 CoMajor
33:  OpDecorate 13(matBuf) 1 Offset 64
34:  OpDecorate 13(matBuf) 2 CoMajor
35:  OpDecorate 13(matBuf) 2 Offset 128
36:  OpDecorate 13(matBuf) 3 CoMajor
37:  OpDecorate 13(matBuf) 3 Offset 192
38:  OpDecorate 13(matBuf) 3 MatStride 16
39:  OpDecorate 13(matBuf) Block
40:  OpDecorate 15(Matrices) DescriptorSet 0
```
This is the SPIR-V Assembly, Part II

```

This is the SPIR-V Assembly, Part III

```

```
### SPIR-V: Printing the Configuration

```bash
$ glslangValidator --c
```

### SPIR-V: More Information

**SPIR-V Tools:**

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