Shaders and SPIR-V

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

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_TRANSITION_BIT = 0x00001000,
    VK_PIPELINE_STAGE_INTERFACE_BIT = 0x00002000,
    VK_PIPELINE_STAGE_HOST_BIT = 0x00004000,
    VK_PIPELINE_STAGE_ALL_GRAPHICS_BIT = 0x00008000,
    VK_PIPELINE_STAGE_ALL_COMMANDS_BIT = 0x00010000,
    VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT = 0x00000080,
}; VkPipelineStageFlagBits;

Vulkan: GLSL Differences from OpenGL

- 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 ) . . .  ;

Specialization Constants:
layout( constant_id = 3 )  const int N = 5;
- Can only use basic operators, declarations, and constructions
- 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.x is still as it was

Vulkan: Shaders’ use of Layouts for Uniform Variables

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

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 a 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

SPIR-V:

Standard Portable Intermediate Representation for Vulkan

```
```

- -V: Compile for Vulkan
- -G: Compile for OpenGL
- -d: 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
3. Can get to your personal folders
4. Does not have make
5. Cannot get to your personal folders
6. Does have make

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

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

Running glslangValidator.exe
You can also run SPIR-V from a Linux Shell

```
$ glslangValidator.exe -V sample-vert.vert -o sample-vert.spv
$ glslangValidator.exe -V sample-frag.frag -o sample-frag.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
- .tcs Tessellation Control Shader
- .tcse Tessellation Evaluation Shader
- .geom Geometry shader
- .frag Fragment shader
- .comp Compute shader

Specify the output file

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 ...

How do you know if SPIR-V compiled successfully?

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 );
//vkCreateShaderModule( LogicalDevice, &vsmci, PALLOCATOR, pShaderModule );
 VkPipelineShaderStageCreateInfo
 VkPipelineVertexInputStateCreateInfo
 VkVertexInputBindingDescription
 VkViewportStateCreateInfo
 VkPipelineRasterizationStateCreateInfo
 VkPipelineColorBlendAttachmentState
 VkPipelineDynamicStateCreateInfo

VkGraphicsPipelineCreateInfo

Shader stages
VertexInput State
InputAssembly State
Tesselation State
Viewport State
Rasterization State
MultiSample State
DepthStencil State
ColorBlend State
Dynamic State
Pipeline layout

RenderPass

basePipelineHandle
basePipelineIndex
You can also take a look at SPIR-V Assembly

```
glslangValidator.exe -V -H sample.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.
SPIR-V: More Information

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