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

Vulkan Vertex and Instance indices:
- gl_VertexIndex
- gl_InstanceIndex
  - 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 output 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;

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

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


Shader file extensions:
-vert Vertex
-tessControl Tessellation Control
-tese 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 shader file extension
-c Print out the maximum sizes of various properties

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
   (This is only available within 64-bit Windows 10.)

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

$ 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

You can also take a look at SPIR-V Assembly

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 . . .

You can also take a look at SPIR-V Assembly

This prints out the SPIR-V "assembly" to standard output.

Other than nerd interest, there is no graphics-programming reason to look at this.

OpenGL vs. Vulkan

Vulkan: Creating a Pipeline

Reading a SPIR-V File into a Vulkan Shader Module

Reading a SPIR-V File into a Shader Module

You can also run SPIR-V from a Linux Shell

You can also take a look at SPIR-V Assembly