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.

**Fixed Function**

**Programmable**

Oregon State University
Computer Graphics

ShadersAndSpirv.pptx
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_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;

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:

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

Descriptor Sets:

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

Push Constants:

```glsl
layout( push_constant ) . . . ;
```

Specialization Constants:

```glsl
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:

```glsl
layout( local_size_x_id = 8, local_size_y_id = 16 );
```

- `gl_WorkGroupSize.z` is still as it was

Vulkan: GLSL Differences from OpenGL

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

**External GLSL Compiler** ➔ **SPIR-V** ➔ **Compiler in driver** ➔ **Vendor-specific code**

**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, from the Khronos Group

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 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

```
ONID=mjbpooh MINGW64 /y/Vulkan/Sample2017
$ 185
gslangValidator.exe -V sample-vert.vert -o sample-vert.spv
Sample.vert

ONID=mjbpooh MINGW64 /y/Vulkan/Sample2017
$ 186
gslangValidator.exe -V sample-frag.frag -o sample-frag.spv
Sample.frag

ONID=mjbpooh MINGW64 /y/Vulkan/Sample2017
$ 
```
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
- .tcs   Tessellation Evaluation Shader
- .geom  Geometry shader
- .frag  Fragment shader
- .comp  Compute shader
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 . . .
```

---

Reading a SPIR-V File into a Vulkan Shader Module

```cpp
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'
    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 );
}```
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;
```
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;
}
```
SPIR-V: More Information

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