The Shaders’ View of the Basic Computer Graphics Pipeline

- You need 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
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;

How Vulkan GLSL Differs from OpenGL GLSL

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

- In the compiler, there is an automatic
  #define VULKAN 130
or whatever the current version number is.
Typically you use this like:
  #ifdef VULKAN
  . . .
  #endif

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 out variables to have specific location numbers:

```
layout ( location = 0 ) out vec4 fFragColor;
```
Shader combinations of separate texture data and samplers as an option:

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

Note: our sample code doesn’t use this.

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

For example, Specialization Constants can be used with 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: Shaders’ use of Layouts for Uniform Variables

```glsl
layout( std140, set = 0, binding = 0 ) uniform sceneMatBuf 
{
    mat4 uProjectionMatrix;
    mat4 uViewMatrix;
    mat4 uSceneMatrix;
} SceneMatrices;
layout( std140, set = 1, binding = 0 ) uniform objectMatBuf 
{
    mat4 uModelMatrix;
    mat4 uNormalMatrix;
} ObjectMatrices;
layout( set = 2, binding = 0 ) uniform sampler2D uTexUnit;
```

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

- You half-precompile 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, when the pipeline structure is finally created
- The SPIR-V spec has been public for a few years—new shader languages are surely being developed
- OpenGL and OpenCL have now 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
```

**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: `glslangValidator`
You Can Run the SPIR-V Compiler on Windows from a Bash Shell

You can run the glslangValidator program from the Windows Command Prompt, but I have found it easier to run the SPIR-V compiler from Windows-Bash.

To install the bash shell on your own Windows machine, go to this URL:

Or, follow these instructions:

1. Head to the **Start menu** search bar, type in ‘terminal,’ and launch the Windows Terminal as administrator. (On some systems, this is called the **Command Prompt**.)
2. Type in the following command in the administrator: `wsl --install`
3. Restart your PC once the installation is complete.

As soon as your PC boots up, the installation will begin again. Your PC will start downloading and installing the Ubuntu software. You’ll soon get asked to set up a username and password. This can be the same as your system’s username and password, but doesn’t have to be. The installation will automatically start off from where you left it.

To install the bash shell on your own Windows machine, go to this URL:

Or, follow these instructions:

1. Click on the Microsoft Start icon
2. Type the word **bash**
3. Click on the Microsoft Start icon

BTW, within bash, if you want to list your files without that sometimes-hard-to-read filename coloring, do this:
`ls -l --color=none`

(ell-ess minus-ell minus-minus-color=none)
Running glslangValidator.exe in bash

As long as I am on bash, I like using the `make` utility. To do that, put these shader compile lines in a file called `Makefile`:

```
ALLSHADERS: sample-vert.vert sample-frag.frag
glslangValidator.exe -V sample-vert.vert -o sample-vert.spv
```

Then type `make ALLSHADERS`:

```
@PC:/mnt/c/MJB/Vulkan/Sample2019-COLOREDCUBE$ make ALLSHADERS
```

Running glslangValidator.exe

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

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

Specify the SPIR-V output file

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

```
```

mjb – December 30, 2022
How do you know if SPIR-V compiled successfully?

Same as C/C++ -- the compiler gives you no nasty messages, it just prints the name of the source file you just compiled.

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

So, if you use the Linux command \texttt{od -x} on the .spv file, like this:

\begin{verbatim}
    od -x sample-vert.spv
\end{verbatim}

the magic number shows up like this:

\begin{verbatim}
    00000000 0203 0723 0000 0001 0008 007e 0000
    00000200 0000 0000 0011 0002 0001 0000 000b 0006

\ldots
\end{verbatim}

"od" stands for "octal dump", even though it can format the raw bits as most anything: octal, hexadecimal, bytes, characters, etc. "-x" means to format in hexadecimal.

Reading a SPIR-V File into a Vulkan Shader Module

\begin{verbatim}
#ifndef \_WIN32
    typedef int errno_t;
    int     fopen_s( FILE**, const char *, const char * );
#endif
#define SPIRV\_MAGIC             0x07230203

VkResult
Init12SpirvShader( std::string filename, VkShaderModule * pShaderModule )
{
    FILE *fp;
#ifdef WIN32
    errno_t err = fopen_s( &fp, filename.c_str( ), "rb" );
    if( err != 0 )
#else
    fp = fopen( filename.c_str( ), "rb" );
    if( fp == NULL )
#endif
    {
        fprintf( FpDebug, "Cannot open shader file '%s'
", filename.c_str( ) );
        return VK\_SHOULD\_EXIT;
    }
    uint32_t magic;
    fread( &magic, sizeof( magic ), 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, OL\_SEEK\_END );
    int size = ftell( fp );
    rewind( fp );
    unsigned char *code = new unsigned char [size];
    fread( code, size, 1, fp );
    fclose( fp );

    
\end{verbatim}

\textcopyright{} Oregon State University

Computer Graphics

\textcopyright{} December 30, 2022

VkShaderModule ShaderModuleVertex;

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\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;
}
```
This is the SPIR-V Assembly, Part I

1. Extensiom "GLSL.std.450"
2. MemoryModel Logical GLSL450
3. EntryPoint Vertex 4 "main" 34 37 35 36 37 38 39 32
4. Source GLSL 400
5. SourceExtension "GL_ARB_separate_shader_objects"
6. SourceExtension "GL_ARB_shading_language_420pack"
7. Name 4 "main"
8. Name 10 "PVM"
9. Name 13 "matBuf"
10. MemberName 13(matBuf) 0 "uModelMatrix"
11. MemberName 13(matBuf) 1 "uViewMatrix"
12. MemberName 13(matBuf) 2 "uProjectionMatrix"
13. Name 15 "Matrices"
14. Name 32 (gl_PerVertex) 0 "gl_Position"
15. Name 32 (gl_PerVertex) 1 "gl_PointSize"
16. Name 32 (gl_PerVertex) 2 "gl_ClipDistance"
17. Name 34 ""
18. Name 37 "aVertex"
19. Name 48 "vNormal"
20. Name 53 "vColor"
21. Name 61 "vTexCoord"
22. Name 62 "sTexCoord"
23. Name 65 "lightBuf"
24. MemberName 65(lightBuf) 0 "uLightPos"
25. Name 67 "Light"
This is the SPIR-V Assembly, Part III

SPIR-V: Printing the Configuration

glsLangValidator --c
SPIR-V: More Information

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

A Google-Wrapped Version of glslangValidator

The shaderc project from Google (https://github.com/google/shaderc) provides a glslangValidator wrapper program called glslc that has a much improved command-line interface. You use, basically, the same way:

```
$ glslc.exe --target-env=vulkan sample-vert.vert -o sample-vert.spv
```

There are several really nice features. The two I really like are:

1. You can #include files into your shader source
2. You can “#define” definitions on the command line like this:

```
$ glslc.exe --target-env=vulkan -DNUMPOINTS=4 sample-vert.vert -o sample-vert.spv
```

glslc is included in your Sample .zip file

This causes a:

```
#define NUMPOINTS 4
```

to magically be inserted into the top of your source code.