GLSL for Vulkan

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

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;

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 100

Vulkan Vertex and Instance indices:

<table>
<thead>
<tr>
<th>Vulkan Vertex and Instance indices</th>
<th>OpenGL uses</th>
</tr>
</thead>
<tbody>
<tr>
<td>gl_VertexIndex</td>
<td>gl_VertexID</td>
</tr>
<tr>
<td>gl_InstanceIndex</td>
<td>gl_InstanceID</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: Shaders’ use of Layouts for Uniform Variables

// non-sampler variables must be in a uniform block:
layout( std140, set = 0, binding = 0 ) uniform matBuf
{
    mat4 uModelMatrix;
    mat4 uViewMatrix;
    mat4 uProjectionMatrix;
    mat3 uNormalMatrix;
} Matrices;

// non-sampler variables 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 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**

```sh
```

- **Shaderfile extensions:**
  - .vert: Vertex
  - .tesc: Tessellation Control
  - .tese: Tessellation Evaluation
  - .geom: Geometry
  - .frag: Fragment
  - .comp: Compute
  (Can be overridden by the -S option)

- **Options:**
  - -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

1. Click on the Microsoft Start icon

2. Type the word `bash`

This is only available within 64-bit Windows 10.

Pick one:

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

- Can get to your personal folders
- Does have make
Running glslangValidator.exe

```
ONID+mjb&pooh MINGW64 /y/Vulkan/Sample2017
$ 185
glslangValidator.exe -V sample-vert.vert -o sample-vert.spv
$ 186
ONID+mjb&pooh MINGW64 /y/Vulkan/Sample2017
$ glslangValidator.exe -V sample-frag.frag -o sample-frag.spv
```

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

Specify the 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
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

```
#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 );
    return VK_SUCCESS;
}
```
Reading a SPIR-V File into a Shader Module

VkShaderModule ShaderModuleVertex;

VkShaderModuleCreateInfo vsmci;
vsmci.sType = VK_STRUCTURE_TYPE_SHADER_MODULE_CREATE_INFO;
// more...
VkResult result = vkCreateShaderModule( LogicalDevice, &vsmci, PALLOCATOR, OUT & ShaderModuleVertex );
// more...
You can also take a look at SPIR-V Assembly

```bash
$ 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

Capability Shader

1:    ExtInstImport "GLSL.std.450"

MemoryModel Logical GLSL450

EntryPoint Vertex 4  "main"  34  37  48  53  56  57  61  63

Source GLSL 400

SourceExtension "GL_ARB_separate_shader_objects"

SourceExtension "GL_ARB_shading_language_420pack"

Name 4  "main"

Name 10  "PVM"

Name 13  "matBuf"

MemberName 13(matBuf) 0  "uModelMatrix"

MemberName 13(matBuf) 1  "uViewMatrix"

MemberName 13(matBuf) 2  "uProjectionMatrix"

MemberName 13(matBuf) 3  "uNormalMatrix"

Name 15  "Matrices"

Name 32  "gl_PerVertex"

MemberName 32(gl_PerVertex) 0  "gl_Position"

MemberName 32(gl_PerVertex) 1  "gl_PointSize"

MemberName 32(gl_PerVertex) 2  "gl_ClipDistance"

Name 34  """"

Name 37  """"

Name 48  """"

Name 53  """"

Name 56  """"

Name 57  """"

Name 61  """"

Name 62  """"

Name 65  """"

Name 69  """"

Name 67  """"

Name 66  """"

Name 72  """"

Name 73  """"

Name 74  """"

Name 76  """"

Name 77  """"

Name 78  """"

Name 79  """"

MemberDecorate 13(matBuf) 0 ColMajor

MemberDecorate 13(matBuf) 0 Offset 0

MemberDecorate 13(matBuf) 0 MatrixStride 16

MemberDecorate 13(matBuf) 1 ColMajor

MemberDecorate 13(matBuf) 1 Offset 64

MemberDecorate 13(matBuf) 1 MatrixStride 16

MemberDecorate 13(matBuf) 2 ColMajor

MemberDecorate 13(matBuf) 2 Offset 128

MemberDecorate 13(matBuf) 2 MatrixStride 16

MemberDecorate 13(matBuf) 3 ColMajor

MemberDecorate 13(matBuf) 3 Offset 192

MemberDecorate 13(matBuf) 3 MatrixStride 16

Decorate 13(matBuf) Block

Decorate 15(Matrices) DescriptorSet 0

This is the SPIR-V Assembly, Part II

Decorate 15(Matrices) Binding 0

MemberDecorate 32(gl_PerVertex) 0 BuiltIn Position

MemberDecorate 32(gl_PerVertex) 1 BuiltIn PointSize

MemberDecorate 32(gl_PerVertex) 2 BuiltIn ClipDistance

Decorate 32(gl_PerVertex) Block

Decorate 37(Vectors) Location 0

Decorate 48(Normal) Location 0

Decorate 53(Normal) Location 1

Decorate 56(Color) Location 1

Decorate 59(Color) Location 2

Decorate 61(TextCoord) Location 2

Decorate 63(TextCoord) Location 3

MemberDecorate 65(lightBuf) 0 Offset 0

MemberDecorate 65(lightBuf) Block

Decorate 67(Light) DescriptorSet 1

Decorate 67(Light) Binding 0

2:     TypeVoid

3:     TypeFunction 2

6:     TypeFloat 32

7:     TypeVector 6(float) 4

8:     TypeMatrix 7(fvec4) 4

9:     TypePointer Function 8

11:    TypeVector 6(float) 3

12:    TypeMatrix 11(fvec3) 3

13(matBuf) TypeStruct 8 8 12

14:    TypePointer Uniform 13(matBuf)

15(Matrices) 14(ptr) Variable Uniform

16:    TypeInt 32 1

17:    16(int) Constant 2

18:    16(int) Constant 1

19:    16(int) Constant 0

20:    29(int) Constant 0

29:    TypeArray 6(float) 30

30:    TypeArray 6(float) 0

31:    TypeArray 6(float) 31

32(gl_PerVertex) TypeStruct 7(fvec4) 31

33:    TypePointer Output 32(gl_PerVertex)

34:    33(ptr) Variable Output

36:    TypePointer Input 11(fvec3)

37(Vectors) 36(ptr) Variable Input

39:    6(fvec2) Constant 0

40:    16(int) Constant 3

41:    16(int) Constant 3

Oregon State University Computer Graphics
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:

```
glslic.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:

```
glslic.exe  --target-env=vulkan  -DNUMPONTS=4  sample-vert.vert  -o  sample-vert.spv
```

glslic is included in your Sample .zip file