Shaders and SPIR-V
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:  

- Both are 0-based

<table>
<thead>
<tr>
<th>GLSL Shader in Vulkan/SPIR-V</th>
<th>GLSL Shader in OpenGL</th>
</tr>
</thead>
<tbody>
<tr>
<td>gl_VertexIndex, gl_InstanceIndex</td>
<td>gl_VertexID, gl_InstanceID</td>
</tr>
</tbody>
</table>

`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:

```
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

**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

Note: our sample code doesn’t use this.
Vulkan: Shaders’ use of Layouts for Uniform Variables

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

---

VkShaderModuleCreateInfo( )
vkCreateShaderModule( )

device
code[ ] (u_int32_t)
codeSize (in bytes)
shaderModuleCreateFlags

---

Oregon State University
Computer Graphics
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.
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`

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

Then type `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
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 **0x07230203**

So, if you use the Linux command `od -x` on the .spv file, like this:

```
  od -x sample-vert.spv
```

the magic number shows up like this:

```
00000000 0203 0723 0000 0001 000a 0008 007e 0000
00000200 0000 0000 0011 0002 0001 0000 000b 0006
...`

“`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

```c
#include <stdio.h>
#include <string.h>

# 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'\n", 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\n", 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 );
  . . .
```
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;
Vulkan: Creating a Pipeline

- VkGraphicsPipelineCreateInfo
  - Shader stages
    - VkShaderModule
    - VkSpecializationInfo
  - VertexInput State
    - VkVertexInputBindingDescription
    - VkVertexInputAttributeDescription
  - InputAssembly State
    - VkPipelineInputAssemblyStateCreateInfo
  - Tesselation State
  - Viewport State
    - VkViewportStateCreateInfo
      - Viewport
        - x, y, w, h
        - minDepth, maxDepth
      - Scissor
        - offset, extent
  - Rasterization State
    - VkPipelineRasterizationStateCreateInfo
      - Topology
      - cullMode, polygonMode, frontFace, lineWidth
  - MultiSample State
  - DepthStencil State
    - VkPipelineDepthStencilStateCreateInfo
      - depthTestEnable, depthWriteEnable, depthCompareOp, stencilTestEnable, stencilOpStateFront, stencilOpStateBack
  - ColorBlend State
    - VkPipelineColorBlendStateCreateInfo
      - blendEnable, srcColorBlendFactor, dstColorBlendFactor, colorBlendOp, srcAlphaBlendFactor, dstAlphaBlendFactor, alphaBlendOp, colorWriteMask
  - ColorBlend Attachment State
  - Dynamic State
    - VkPipelineDynamicStateCreateInfo
  - Pipeline layout
  - RenderPass
  - VkPipelineBasePipelineHandle
  - VkPipelineBasePipelineIndex

- vkCreateGraphicsPipeline()
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

```
#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;

void main()
{
    mat4 PVM = Matrices.uProjectionMatrix * Matrices.uViewMatrix * Matrices.uModelMatrix;
    gl_Position = PVM * vec4( aVertex, 1.0 );
    vNormal = Matrices.uNormalMatrix * aNormal;
    vColor = aColor;
    vTexCoord = aTexCoord;
}
```

---

This is the SPIR-V Assembly, Part I

```
Capability Shader

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 "vVertex"
Name 48 "vNormal"
Name 53 "vColor"
Name 57 "aColor"
Name 61 "vTexCoord"
Name 63 "aTexCoord"
Name 65 "lightBuf"
MemberName 65(lightBuf) 0 "uLightPos"
Name 67 "Light"
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

```assembly
define void main() {
  
  mat4 PVM = Matrices.uProjectionMatrix * Matrices.uViewMatrix * Matrices.uModelMatrix;
  gl_Position = PVM * vec4(aVertex, 1.0);

  vNormal = Matrices.uNormalMatrix * aNormal;
  vColor = aColor;
  vTexCoord = aTexCoord;

  lightBuf = vec4 uLightPos;

  light = vec4 vTexCoord;

  lightBuf = lightBuf * vec4(1.0);

  gl_Position = PVM * vec4(aVertex, 1.0);

  vNormal = Matrices.uNormalMatrix * aNormal;
  vColor = aColor;
  vTexCoord = aTexCoord;
}
```

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(aVertex) Location 0
Decorate 48(vNormal) Location 0
Decorate 53(aNormal) Location 1
Decorate 56(vColor) Location 1
Decorate 57(aColor) Location 2
Decorate 61(vTexCoord) Location 2
Decorate 63(aTexCoord) Location 3
MemberDecorate 65(lightBuf) 0 Offset 0
Decorate 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 8 12
14:            TypePointer Uniform 13(matBuf)
15(Matrices):   14(ptr) Variable Uniform
16:            TypeInt 32 1
17:            16(int) Constant 2
18:            TypePointer Uniform 8
21:            16(int) Constant 1
25:            16(int) Constant 0
29:            TypeInt 32 0
30:            29(int) Constant 1
31:            TypeArray 6(float) 30
32(gl_PerVertex):  TypeStruct 7(fvec4) 6(float) 31
33:            TypePointer Output 32(gl_PerVertex)
34:            33(ptr) Variable Output
36:            TypePointer Input 11(fvec3)
37(aVertex):    36(ptr) Variable Input
39:            6(float) Constant 1065353216
45:            TypePointer Output 7(fvec4)
47:            TypePointer Output 11(fvec3)
48(vNormal):    47(ptr) Variable Output
49:            16(int) Constant 3
This is the SPIR-V Assembly, Part III

```spirv
#version 400
#extension GL_ARB_separate_shader_objects : enable
#extension GL_ARB_shading_language_420pack : enable
layout( std140, set = 0, binding = 0 ) uniform mat4Buf
{
    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;

void
main()
{
    mat4 PVM = Matrices.uProjectionMatrix * Matrices.uViewMatrix * Matrices.uModelMatrix;
    gl_Position = PVM * vec4( aVertex, 1. );

    vNormal = Matrices.uNormalMatrix * aNormal;
    vColor = aColor;
    vTexCoord = aTexCoord;
}

50:   TypePointer Uniform 12
53(aNormal):   36(ptr) Variable Input
56(vColor):   47(ptr) Variable Output
57(aColor):   36(ptr) Variable Input
59:   TypeVector 6(float) 2
60:   TypePointer Output 59(fvec2)
61(vTexCoord):   60(ptr) Variable Output
62:   TypePointer Input 59(fvec2)
63(aTexCoord):   62(ptr) Variable Input
65(lightBuf):   TypeStruct 7(fvec4)
66:   TypePointer Uniform 65(lightBuf)
67(Light):   66(ptr) Variable Uniform
4(main):   2 Function None 3
5:   Label
10(PVM):   9(ptr) Variable Function
19:   18(ptr) AccessChain 15(Matrices) 17
20:   8 Load 19
22:   18(ptr) AccessChain 15(Matrices) 21
23:   8 Load 22
24:   8 MatrixTimesMatrix 20 23
26:   18(ptr) AccessChain 15(Matrices) 25
27:   8 Load 26
28:   8 MatrixTimesMatrix 24 27
    Store 10(PVM) 28
35:   8 Load 10(PVM)
38:   11(fvec3) Load 37(aVertex)
40:   6(float) CompositeExtract 38 0
41:   6(float) CompositeExtract 38 1
42:   6(float) CompositeExtract 38 2
43:   7(fvec4) CompositeConstruct 40 41 42 39
44:   7(fvec4) MatrixTimesVector 35 43
46:   45(ptr) AccessChain 34 25
    Store 46 44
51:   50(ptr) AccessChain 15(Matrices) 49
52:   12 Load 51
54:   11(fvec3) Load 53(aNormal)
55:   11(fvec3) MatrixTimesVector 52 54
    Store 48(vNormal) 55
58:   11(fvec3) Load 57(aColor)
    Store 56(vColor) 58
64:   59(fvec2) Load 63(aTexCoord)
    Store 61(vTexCoord) 64
    Return
    FunctionEnd
```
### SPIR-V: Printing the Configuration

```bash
$ glslangValidator -c
```

<table>
<thead>
<tr>
<th>Limit</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaxLights</td>
<td>32</td>
</tr>
<tr>
<td>MaxClipPlanes</td>
<td>6</td>
</tr>
<tr>
<td>MaxTextureUnits</td>
<td>32</td>
</tr>
<tr>
<td>MaxTextureCoords</td>
<td>32</td>
</tr>
<tr>
<td>MaxVertexAttribs</td>
<td>64</td>
</tr>
<tr>
<td>MaxVertexAttribs</td>
<td>64</td>
</tr>
<tr>
<td>MaxUniformComponents</td>
<td>4096</td>
</tr>
<tr>
<td>MaxVaryingFloats</td>
<td>64</td>
</tr>
<tr>
<td>MaxVertexTextureImageUnits</td>
<td>32</td>
</tr>
<tr>
<td>MaxCombinedTextureImageUnits</td>
<td>80</td>
</tr>
<tr>
<td>MaxTextureUnits</td>
<td>32</td>
</tr>
<tr>
<td>MaxFragmentUniformComponents</td>
<td>4096</td>
</tr>
<tr>
<td>MaxDrawBuffers</td>
<td>32</td>
</tr>
<tr>
<td>MaxVertexUniformVectors</td>
<td>128</td>
</tr>
<tr>
<td>MaxVaryingVectors</td>
<td>8</td>
</tr>
<tr>
<td>MaxFragmentUniformVectors</td>
<td>16</td>
</tr>
<tr>
<td>MaxVertexOutputVectors</td>
<td>16</td>
</tr>
<tr>
<td>MaxFragmentInputVectors</td>
<td>15</td>
</tr>
<tr>
<td>MinProgramTexelOffset</td>
<td>-8</td>
</tr>
<tr>
<td>MaxProgramTexelOffset</td>
<td>7</td>
</tr>
<tr>
<td>MaxClipDistances</td>
<td>8</td>
</tr>
<tr>
<td>MaxComputeWorkGroupCountX</td>
<td>65535</td>
</tr>
<tr>
<td>MaxComputeWorkGroupCountY</td>
<td>65535</td>
</tr>
<tr>
<td>MaxComputeWorkGroupCountZ</td>
<td>65535</td>
</tr>
<tr>
<td>MaxComputeWorkGroupSizeX</td>
<td>1024</td>
</tr>
<tr>
<td>MaxComputeWorkGroupSizeY</td>
<td>1024</td>
</tr>
<tr>
<td>MaxComputeWorkGroupSizeZ</td>
<td>64</td>
</tr>
<tr>
<td>MaxComputeUniformComponents</td>
<td>1024</td>
</tr>
<tr>
<td>MaxComputeTextureImageUnits</td>
<td>16</td>
</tr>
<tr>
<td>MaxComputeImageUniforms</td>
<td>8</td>
</tr>
<tr>
<td>MaxComputeAtomicCounters</td>
<td>8</td>
</tr>
<tr>
<td>MaxComputeAtomicCounterBuffers</td>
<td>1</td>
</tr>
<tr>
<td>MaxVaryingComponents</td>
<td>60</td>
</tr>
<tr>
<td>MaxVertexOutputComponents</td>
<td>64</td>
</tr>
<tr>
<td>MaxGeometryInputComponents</td>
<td>64</td>
</tr>
<tr>
<td>MaxGeometryOutputComponents</td>
<td>128</td>
</tr>
<tr>
<td>MaxFragmentInputComponents</td>
<td>128</td>
</tr>
<tr>
<td>MaxImageUnits</td>
<td>8</td>
</tr>
<tr>
<td>MaxCombinedImageUnitsAndFragmentOutputs</td>
<td>8</td>
</tr>
<tr>
<td>MaxCombinedShaderOutputResources</td>
<td>8</td>
</tr>
<tr>
<td>MaxImageSamples</td>
<td>0</td>
</tr>
<tr>
<td>MaxVertexTextureUniforms</td>
<td>0</td>
</tr>
<tr>
<td>MaxTessControlImageUniforms</td>
<td>0</td>
</tr>
<tr>
<td>MaxTessEvaluationImageUniforms</td>
<td>0</td>
</tr>
<tr>
<td>MaxGeometryImageUniforms</td>
<td>0</td>
</tr>
<tr>
<td>MaxFragmentImageUniforms</td>
<td>81</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Limit</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaxGeometryTextureImageUnits</td>
<td>16</td>
</tr>
<tr>
<td>MaxGeometryOutputVertices</td>
<td>256</td>
</tr>
<tr>
<td>MaxGeometryTotalOutputComponents</td>
<td>1024</td>
</tr>
<tr>
<td>MaxGeometryUniformComponents</td>
<td>1024</td>
</tr>
<tr>
<td>MaxGeometryVaryingComponents</td>
<td>64</td>
</tr>
<tr>
<td>MaxTessControlInputComponents</td>
<td>128</td>
</tr>
<tr>
<td>MaxTessControlOutputComponents</td>
<td>128</td>
</tr>
<tr>
<td>MaxTessControlTextureImageUnits</td>
<td>16</td>
</tr>
<tr>
<td>MaxTessControlUniformComponents</td>
<td>1024</td>
</tr>
<tr>
<td>MaxTessControlTotalOutputComponents</td>
<td>4096</td>
</tr>
<tr>
<td>MaxTessEvaluationInputComponents</td>
<td>128</td>
</tr>
<tr>
<td>MaxTessEvaluationOutputComponents</td>
<td>128</td>
</tr>
<tr>
<td>MaxTessEvaluationTextureImageUnits</td>
<td>16</td>
</tr>
<tr>
<td>MaxTessEvaluationUniformComponents</td>
<td>1024</td>
</tr>
<tr>
<td>MaxTessPatchComponents</td>
<td>120</td>
</tr>
<tr>
<td>MaxPatchVertices</td>
<td>32</td>
</tr>
<tr>
<td>MaxTessGenLevel</td>
<td>64</td>
</tr>
<tr>
<td>MaxViewports</td>
<td>16</td>
</tr>
<tr>
<td>MaxVertexAtomicCounters</td>
<td>0</td>
</tr>
<tr>
<td>MaxTessControlAtomicCounters</td>
<td>0</td>
</tr>
<tr>
<td>MaxTessEvaluationAtomicCounters</td>
<td>0</td>
</tr>
<tr>
<td>MaxGeometryAtomicCounters</td>
<td>0</td>
</tr>
<tr>
<td>MaxTessPatchAtomicCounters</td>
<td>8</td>
</tr>
<tr>
<td>MaxCombinedAtomicCounters</td>
<td>8</td>
</tr>
<tr>
<td>MaxAtomicCounterBindings</td>
<td>1</td>
</tr>
<tr>
<td>MaxVertexAtomicCounterBuffers</td>
<td>0</td>
</tr>
<tr>
<td>MaxTessControlAtomicCounterBuffers</td>
<td>0</td>
</tr>
<tr>
<td>MaxTessEvaluationAtomicCounterBuffers</td>
<td>0</td>
</tr>
<tr>
<td>MaxGeometryAtomicCounterBuffers</td>
<td>0</td>
</tr>
<tr>
<td>MaxFragmentAtomicCounterBuffers</td>
<td>1</td>
</tr>
<tr>
<td>MaxCombinedAtomicCounterBuffers</td>
<td>1</td>
</tr>
<tr>
<td>MaxAtomicCounterBufferSize</td>
<td>16384</td>
</tr>
<tr>
<td>MaxTransformFeedbackBuffers</td>
<td>4</td>
</tr>
<tr>
<td>MaxTransformFeedbackInterleavedComponents</td>
<td>64</td>
</tr>
<tr>
<td>MaxCullDistances</td>
<td>8</td>
</tr>
<tr>
<td>MaxCombinedClipAndCullDistances</td>
<td>8</td>
</tr>
<tr>
<td>MaxSamples</td>
<td>4</td>
</tr>
<tr>
<td>nonInductiveForLoops</td>
<td>1</td>
</tr>
<tr>
<td>whileLoops</td>
<td>1</td>
</tr>
<tr>
<td>doWhileLoops</td>
<td>1</td>
</tr>
<tr>
<td>generalUniformIndexing</td>
<td>1</td>
</tr>
<tr>
<td>generalAttributeMatrixVectorIndexing</td>
<td>1</td>
</tr>
<tr>
<td>generalVaryingIndexing</td>
<td>1</td>
</tr>
<tr>
<td>generalSamplerIndexing</td>
<td>1</td>
</tr>
<tr>
<td>generalVariableIndexing</td>
<td>1</td>
</tr>
<tr>
<td>generalConstantMatrixVectorIndexing</td>
<td>1</td>
</tr>
<tr>
<td>generalConstantMatrixIndexing</td>
<td>1</td>
</tr>
</tbody>
</table>
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  -DNUMPONTS=4   sample-vert.vert  -o  sample-vert.spv
```

This causes:

```
#define NUMPONTS   4
```

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