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

Vulkan: GLSL Differences from OpenGL

<table>
<thead>
<tr>
<th>Shader combinations of separate texture data and samplers:</th>
</tr>
</thead>
<tbody>
<tr>
<td>uniform sampler s;</td>
</tr>
<tr>
<td>uniform texture2D t;</td>
</tr>
<tr>
<td>vec4 rgba = texture(  sampler2D( t, s ),  vST );</td>
</tr>
</tbody>
</table>

Descriptor Sets:

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

Push Constants:

layout( push_constant ) . . .  ;

Specialization Constants:

layout( constant_id = 3 ) const int N = 5;

- Can only use basic operators, declarations, and constructions
- Only for scalars, but a vector can be constructed from specialization constants

Specialization Constants for Compute Shaders:

layout( local_size_x_id = 8, local_size_y_id = 16 );

- gl_WorkGroupSize.z is still as it was

Vulkan: GLSL Differences from OpenGL

<table>
<thead>
<tr>
<th>These are gl_VertexIndex, gl_InstanceIndex</th>
</tr>
</thead>
<tbody>
<tr>
<td>In OpenGL, they broadcast to all color attachments</td>
</tr>
<tr>
<td>In Vulkan, they just broadcast to color attachment location #0</td>
</tr>
<tr>
<td>Best idea, don’t use it – explicitly declare output variables to have specific location numbers</td>
</tr>
</tbody>
</table>

Vulkan: Shaders’ use of Layouts for Uniform Variables

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

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 public 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 has now evolved into a cross-API standard, fully defined by Khronos with native support for shader and kernel features – called SPIR-V.
- SPIR-V is the first public standard, cross-API intermediate language for naturally representing parallel compute and graphics and is incorporated as part of the core specification of both OpenCL 1.1 and OpenGL 2.1 and the new Vulkan graphics and compute API.
- SPIR-V improves the runtime efficiency for OpenCL 1.2, 2.0, 2.1, 2.2 and Vulkan – including full tess control and graphics and smaller constructs not supported in LLVM. SPIR-V also supports OpenCL 1.1 and OpenGL 2.0 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 2.2, including init and finalizer function execution 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
- SPIR-V is catalyzing a revolution in the language compiler ecosystem – it can split the compiler chain across multiple vendor products, existing high-level language front-ends and programs in a standardized intermediate form to be repeated in Vulkan or OpenCL drivers. For instance, a VSG/Vulkan front-end can take high-level language programs and compile them to Vulkan SPIR-V, then pass the SPIR-V intermediate form to a Vulkan driver to be consumed as Vulkan shader modules or Vulkan kernels. The driver can then compile SPIR-V to native shader or kernel code.
- The ability to provide a consistent language front-end across different 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 architectures.

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

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

Running glslangValidator.exe

Picking the Bash shell allows you to get to your personal folders: cannot get to your personal folders

Pick one:
- Can get to your personal folders
- Does not have make
- Cannot get to your personal folders
- Does have make
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
- .tccs Tessellation Control Shader
- .tecs Tessellation Evaluation Shader
- .geom Geometry shader
- .frag Fragment shader
- .comp Compute shader

Specify the output file

Same as C/C++ -- the compiler gives you no nasty messages.

Also, if you care, legal .spv files have a magic number 0x07230203

So, if you do an od –x on the .spv file, the magic number looks like this:

0203 0723 ... 

How do you know if SPIR-V compiled successfully?

Reading a SPIR-V File into a Vulkan Shader Module

```
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 );

    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, IN &vsmci, PALLOCATOR, pShaderModule );
    fprintf( FpDebug, "Shader Module '%s' successfully loaded
" , filename.c_str() );
    delete [  ] code;
    return result;
}
```

Vulkan: Creating a Pipeline
You can also take a look at SPIR-V Assembly

This prints out the SPIR-V "assembly" to standard output. Other than neat interest, there is no graphics-programming reason to look at this.

glslangValidator.exe  -V  -H  sample.vert -o sample-vert.spv
SPIR-V: More Information

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

Installing bash on Windows

1. Open Settings.
2. Click on "Update & security".
3. Click on "Developer features".
4. Under "Use developer features", select the "Developer mode" option to setup the environment to install Bash.
5. On the message box, click "Yes" to turn on developer model.
6. After the necessary components install, you’ll need to restart your computer.
7. Once your computer reboots, open Control Panel.
8. Click on Programs.
9. Click on "Turn Windows features on or off".
10. Check the "Windows Subsystem for Linux (beta)" option.
11. Click "OK".
12. After the components install on your computer, click the "Restart now" button to complete the task.
13. Once the components installed on your computer, click the "Restart now" button to complete the task.
14. After your computer reboots, you will notice that Bash won’t appear in the "Recently added" list of apps, this is because Bash isn’t actually installed yet. Now that you have setup the necessary components, use the following steps to complete the installation of Bash.
15. Open Start, do a search for "bash.exe", and press Enter.
16. On the command prompt, type "y" and press Enter to download and install Bash from the Windows Store.
17. Then you’ll need to create a default UNIX user account. This account doesn’t have to be the same as your Windows account. Enter the username in the required field and press Enter (you can’t use the username "admin").
18. Close the "bash.exe" command prompt.
19. Now that you completed the installation and setup, you can open the Bash tool from the Start menu like you would with any other app.