Vulkan Shader Stages

Shader stages

Vulkan: GLSL Differences from OpenGL

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

Vulkan: Creating a Pipeline

Vulkan: GLSL Differences from OpenGL

Shader combinations of separate texture data and samplers:

- `rootSignature = sampler2D(s, uST);
  vec4 rgba = texture(rootSignature, vST);`

Descriptor Sets:
- `layout(set=0, binding=0);`

Specialization Constants:
- `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:
- `layout(local_size_x=8, local_size_y=16);
  gl_WorkGroupSize.z is still as it was`
Vulkan: Shaders' use of Layouts for Uniform Variables

```c
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

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:

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-V has evolved into a truly open API standard that is fully defined by Khronos with active support from the shader and kernel community—called SPIR-V.
- SPIR-V is the first open standard, cross-API inter-language for shader, representing parallel compute and graphics, and it is comprised of the cross-architecture specifications of both OpenCL 2.2 and Vulkan 1.2, 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 2.3, as well as the features of OpenCL 1.2, but with a language that is optimized for Vulkan.
- SPIR-V 1.1, released in parallel with OpenCL 2.2, now supports all the features of OpenCL 1.2, including OpenCL 2.1, adding compiler and runtime features like native modules, kernel and workgroup creation, and named kernels. SPIR-V 1.1 also enhances the expressiveness of kernel programs by supporting named barriers, subgroup execution, and program scope barriers.
- SPIR-V is catalyzing a revolution in the language compiler ecosystem—It can split the compiler chain across multiple vendors’ products, modeling high-level language front-ends to end programs on a standardized, open-source intermediate language source compiler, which can then be translated to any target back-end, improving kernel reliability and portability across multiple hardware representations.
- For developers, using SPIR-V means that kernels can be compiled directly to SPIR-V without having to deal with vendor-specific binary formats.

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

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

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

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

1. Type word bash
2. Pick one:
   - Can get to your personal folders
   - Does not have make
   - Cannot get to your personal folders
   - Does have make

Windows: gleslangValidate.exe
Linux: setenv LD_LIBRARY_PATH /usr/local/common/gcc-6.3.0/lib64/
Running glslangValidator.exe

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
```

You can also run SPIR-V from a Linux Shell

You can also run SPIR-V from a Linux Shell

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

Specify the output file

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

Also, if you care, the `.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 . . .

How do you know if SPIR-V compiled successfully?

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

```gl
#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