codeplay

THE HETEROGENEOUS SYSTEMS EXPERTS

An Introduction to SPIR-V

Neil Henning

Principal Software Engineer, Vulkan & SPIR-V

Game Developers Conference – March 2016

What is SPIR-V?

SPIR-V at its heart is:

- A binary intermediate representation
- That is cross vendor
- And cross API
- And supports graphics & compute

Why does Vulkan use SPIR-V?

Because:

- It separates shader source from vendor implementations
- Allows for advances in shader languages to happen asynchronously from Vulkan
- Small performance improvement too!

Simple example

Lets take a simple fragment shader:

```
#version 450

layout(location = 0)
    out vec4 out_colour;

void main() {
 out_colour = vec4(0.4, 0.4, 0.8, 1.0);
}
```

Simple example

Lets take a simple fragment shader:

```
#version 450

layout(location = 0)
    out vec4 out_colour;

void main() {
 out_colour = vec4(0.4, 0.4, 0.8, 1.0);
}
```

```
0302 2307 0000 0100 0100 0800 0e00 0000
0000 0000 1100 0200 0100 0000 0b00 0600
0100 0000 474c 534c 2e73 7464 2e34 3530
0000 0000 0e00 0300 0000 0000 0100 0000
0f00 0600 0400 0000 0400 0000 6d61 696e
0000 0000 0900 0000 1000 0300 0400 0000
0700 0000 0300 0300 0200 0000 8c00 0000
0500 0400 0400 0000 6d61 696e 0000 0000
0500 0600 0900 0000 676c 5f46 7261 6743
6f6c 6f72 0000 0000 1300 0200 0200 0000
2100 0300 0300 0000 0200 0000 1600 0300
0600 0000 2000 0000 1700 0400 0700 0000
0600 0000 0400 0000 2000 0400 0800 0000
0300 0000 0700 0000 3b00 0400 0800 0000
0900 0000 0300 0000 2b00 0400 0600 0000
0a00 0000 cdcc cc3e 2b00 0400 0600 0000
0b00 0000 cdcc 4c3f 2b00 0400 0600 0000
0c00 0000 0000 803f 2c00 0700 0700 0000
0d00 0000 0a00 0000 0a00 0000 0b00 0000
0c00 0000 3600 0500 0200 0000 0400 0000
0000 0000 0300 0000 f800 0200 0500 0000
3e00 0300 0900 0000 0d00 0000 fd00 0100
3800 0100
```

Simple example

```
; SPIR-V
; Version: 1.0
; Generator: Khronos Glslang Reference Front End; 1
; Bound: 14
; Schema: 0
                OpCapability Shader
          %1 = OpExtInstImport "GLSL.std.450"
                OpMemoryModel Logical GLSL450
                OpEntryPoint Fragment %4 "main" %9
                OpExecutionMode %4 OriginUpperLeft
                OpSource GLSL 450
                OpName %4 "main“
                OpName %9 “out_colour“
                OpDecorate %9 Location 0
          %2 = OpTypeVoid
          %3 = OpTypeFunction %2
          %6 = OpTypeFloat 32
          %7 = OpTypeVector %6 4
          %8 = OpTypePointer Output %7
          %9 = OpVariable %8 Output
         %10 = OpConstant %6 0.4
         %11 = OpConstant %6 0.8
         %12 = OpConstant %6 1
         %13 = OpConstantComposite %7 %10 %10 %11 %12
          %4 = OpFunction %2 None %3
          %5 = OpLabel
                OpStore %9 %13
                OpReturn
                OpFunctionEnd
```

```
0302 2307 0000 0100 0100 0800 0e00 0000
0000 0000 1100 0200 0100 0000 0b00 0600
0100 0000 474c 534c 2e73 7464 2e34 3530
0000 0000 0e00 0300 0000 0000 0100 0000
0f00 0600 0400 0000 0400 0000 6d61 696e
0000 0000 0900 0000 1000 0300 0400 0000
0700 0000 0300 0300 0200 0000 8c00 0000
0500 0400 0400 0000 6d61 696e 0000 0000
0500 0600 0900 0000 676c 5f46 7261 6743
6f6c 6f72 0000 0000 1300 0200 0200 0000
2100 0300 0300 0000 0200 0000 1600 0300
0600 0000 2000 0000 1700 0400 0700 0000
0600 0000 0400 0000 2000 0400 0800 0000
0300 0000 0700 0000 3b00 0400 0800 0000
0900 0000 0300 0000 2b00 0400 0600 0000
0a00 0000 cdcc cc3e 2b00 0400 0600 0000
0b00 0000 cdcc 4c3f 2b00 0400 0600 0000
0c00 0000 0000 803f 2c00 0700 0700 0000
0d00 0000 0a00 0000 0a00 0000 0b00 0000
0c00 0000 3600 0500 0200 0000 0400 0000
0000 0000 0300 0000 f800 0200 0500 0000
3e00 0300 0900 0000 0d00 0000 fd00 0100
3800 0100
```

Simple example

```
; SPIR-V
; Version: 1.0
; Generator: Khronos Glslang Reference Front End; 1
; Bound: 14
; Schema: 0
                OpCapability Shader
          %1 = OpExtInstImport "GLSL.std.450"
                OpMemoryModel Logical GLSL450
                OpEntryPoint Fragment %4 "main" %9
                OpExecutionMode %4 OriginUpperLeft
                OpSource GLSL 450
                OpName %4 "main“
                OpName %9 “out_colour“
                OpDecorate %9 Location 0
          %2 = OpTypeVoid
          %3 = OpTypeFunction %2
          %6 = OpTypeFloat 32
          %7 = OpTypeVector %6 4
          %8 = OpTypePointer Output %7
          %9 = OpVariable %8 Output
         %10 = OpConstant %6 0.4
         %11 = OpConstant %6 0.8
         %12 = OpConstant %6 1
         %13 = OpConstantComposite %7 %10 %10 %11 %12
          %4 = OpFunction %2 None %3
          %5 = OpLabel
                OpStore %9 %13
                OpReturn
                OpFunctionEnd
```

```
#version 450

layout(location = 0)
    out vec4 out_colour;

void main() {
 out_colour = vec4(0.4, 0.4, 0.8, 1.0);
}
```

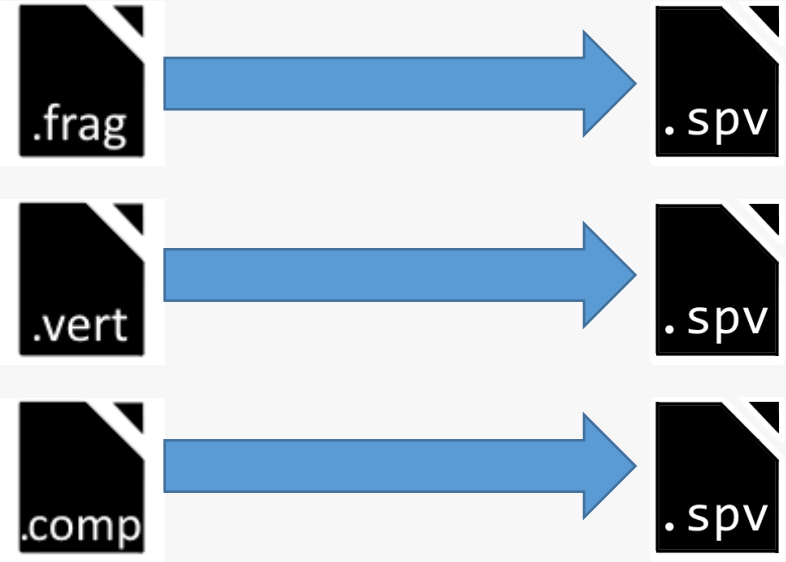
SPIR-V Tools

Khronos have open sourced two projects on GitHub:

- glslang compiler (GLSL -> SPIR-V) & SPIR-V remapper
  https://github.com/KhronosGroup/glslang
- SPIR-V tools (assembler, disassembler, validator)
  https://github.com/KhronosGroup/SPIRV-Tools

SPIR-V Tools - glslang

glslang takes GLSL shaders, turns them into SPIR-V




```
./glslang -V -o our_shader.spv our_shader.frag
```

```
./glslang -V -o our_shader.spv our_shader.vert
```

```
./glslang -V -o our_shader.spv our_shader.comp
```

SPIR-V Tools - glslang

glslang takes GLSL shaders, turns them into SPIR-V

```
./glslang –V –o our_shader.spv our_shader.frag
```

SPIR-V Tools - glslang

glslang takes GLSL shaders, turns them into SPIR-V

```
./glslang -V -o our_shader.spv our_shader.frag
```

-V to output SPIR-V binary

SPIR-V Tools - spirv-dis

spirv-dis takes SPIR-V binaries, turns them into a textual form

```
./spirv-dis –o our_shader.spvasm our_shader.spv
```

SPIR-V Tools - spirv-dis

```
; SPIR-V
; Version: 1.0
; Generator: Khronos Glslang Reference Front End; 1
; Bound: 14
; Schema: 0
                OpCapability Shader
          %1 = OpExtInstImport "GLSL.std.450"
               OpMemoryModel Logical GLSL450
               OpEntryPoint Fragment %4 "main" %9
               OpExecutionMode %4 OriginUpperLeft
               OpSource GLSL 450
               OpName %4 "main“
               OpName %9 “out_colour“
               OpDecorate %9 Location 0
          %2 = OpTypeVoid
          %3 = OpTypeFunction %2
          %6 = OpTypeFloat 32
          %7 = OpTypeVector %6 4
          %8 = OpTypePointer Output %7
          %9 = OpVariable %8 Output
         %10 = OpConstant %6 0.4
         %11 = OpConstant %6 0.8
         %12 = OpConstant %6 1
         %13 = OpConstantComposite %7 %10 %10 %11 %12
          %4 = OpFunction %2 None %3
          %5 = OpLabel
               OpStore %9 %13
               OpReturn
               OpFunctionEnd
```

This is the SPIR-V assembly output (as we seen earlier)

SPIR-V Tools - spirv-dis

```
; SPIR-V
; Version: 1.0
; Generator: Khronos Glslang Reference Front End; 1
; Bound: 14
; Schema: 0
                OpCapability Shader
          %1 = OpExtInstImport "GLSL.std.450"
               OpMemoryModel Logical GLSL450
               OpEntryPoint Fragment %4 "main" %9
               OpExecutionMode %4 OriginUpperLeft
               OpSource GLSL 450
               OpName %4 "main“
               OpName %9 “out_colour“
               OpDecorate %9 Location 0
          %2 = OpTypeVoid
          %3 = OpTypeFunction %2
          %6 = OpTypeFloat 32
          %7 = OpTypeVector %6 4
          %8 = OpTypePointer Output %7
          %9 = OpVariable %8 Output
         %10 = OpConstant %6 0.4
         %11 = OpConstant %6 0.8
         %12 = OpConstant %6 1
         %13 = OpConstantComposite %7 %10 %10 %11 %12
          %4 = OpFunction %2 None %3
          %5 = OpLabel
               OpStore %9 %13
               OpReturn
               OpFunctionEnd
```

This is the SPIR-V assembly output (as we seen earlier)

```
./spirv-dis our_shader.spv
```

If –o <name>.spvasm was not provided, spirv-dis will output to the command line with syntax colouring

SPIR-V Tools - spirv-as

spirv-as takes SPIR-V assembly, turns it into SPIR-V binaries

```
./spirv-as –o our_shader.spv our_shader.spvasm
```

SPIR-V Tools - spirv-as

The spirv-as tool has a cool feature – virtual IDs:

- Any ID %<number> can be written as %<alpha char><alpha char | number>*

SPIR-V Tools - spirv-as

The spirv-as tool has a cool feature – virtual IDs:

- Any ID %<number> can be written as %<alpha char><alpha char | number>*
- Then when spirv-as turns the SPIR-V assembly into a SPIR-V binary, it turns all virtual IDs into actual numerical IDs

SPIR-V Tools - spirv-as

Before:

```
          OpCapability Shader
          OpMemoryModel Logical GLSL450
          OpEntryPoint Fragment %func "main" %color
          OpExecutionMode %func OriginUpperLeft
          OpDecorate %color Location 0
    %void = OpTypeVoid
       %3 = OpTypeFunction %void
   %float = OpTypeFloat 32
    %vec4 = OpTypeVector %float 4
       %8 = OpTypePointer Output %vec4
   %color = OpVariable %8 Output
      %rg = OpConstant %float 0.4
       %b = OpConstant %float 0.8
       %a = OpConstant %float 1
%constant = OpConstantComposite %vec4 %rg %rg %b %a
    %func = OpFunction %void None %3
       %5 = OpLabel
          OpStore %color %constant
          OpReturn
          OpFunctionEnd
```

SPIR-V Tools - spirv-as

Before:

```
          OpCapability Shader
          OpMemoryModel Logical GLSL450
          OpEntryPoint Fragment %func "main" %color
          OpExecutionMode %func OriginUpperLeft
          OpDecorate %color Location 0
   %void = OpTypeVoid
      %3 = OpTypeFunction %void
  %float = OpTypeFloat 32
   %vec4 = OpTypeVector %float 4
      %8 = OpTypePointer Output %vec4
  %color = OpVariable %8 Output
     %rg = OpConstant %float 0.4
      %b = OpConstant %float 0.8
      %a = OpConstant %float 1
%constant = OpConstantComposite %vec4 %rg %rg %b %a
   %func = OpFunction %void None %3
      %5 = OpLabel
          OpStore %color %constant
          OpReturn
          OpFunctionEnd
```

After:

```
          OpCapability Shader
          OpMemoryModel Logical GLSL450
          OpEntryPoint Fragment %4 "main" %9
          OpExecutionMode %4 OriginUpperLeft
          OpDecorate %9 Location 0
   %2 = OpTypeVoid
   %3 = OpTypeFunction %2
   %6 = OpTypeFloat 32
   %7 = OpTypeVector %6 4
   %8 = OpTypePointer Output %7
   %9 = OpVariable %8 Output
  %10 = OpConstant %6 0.4
  %11 = OpConstant %6 0.8
  %12 = OpConstant %6 1
  %13 = OpConstantComposite %7 %10 %10 %11 %12
   %4 = OpFunction %2 None %3
   %5 = OpLabel
          OpStore %9 %13
          OpReturn
          OpFunctionEnd
```

SPIR-V Tools – spirv-val

spirv-val validates a SPIR-V binary

```
./spirv-val our_shader.spv
```

SPIR-V Tools – spirv-val

```
      OpCapability Shader
      OpMemoryModel Logical GLSL450
      OpEntryPoint Fragment %1 "main" %2
      OpExecutionMode %1 OriginUpperLeft
 %3 = OpTypeVoid
 %4 = OpTypeFunction %3
 %5 = OpTypeFloat 32
 %6 = OpTypeInt 32 0
 %7 = OpTypeVector %5 4
 %8 = OpTypePointer Output %7
 %2 = OpVariable %8 Output
 %9 = OpConstant %5 0.4
%10 = OpConstant %5 0.8
%11 = OpConstant %6 1
%12 = OpConstantComposite %7 %9 %9 %10 %11
 %1 = OpFunction %3 None %4
%13 = OpLabel
      OpStore %2 %12
      OpReturn
      OpFunctionEnd
```

Imagine you've written a fragment shader, in SPIR-V by hand

SPIR-V Tools – spirv-val

You then assemble it to SPIR-V:

```
./spirv-as –o our_shader.spv our_shader.spvasm
```

Load that into your game, and it crashes ☹

Always validate your SPIR-V after assembling!

Validate the SPIR-V:

```
./spirv-val our_shader.spv
    error: 61: OpConstantComposite Constituent <id> '11's type does not match
        Result Type <id> '7's vector element type.
```

SPIR-V Tools – spirv-val

```
      OpCapability Shader
      OpMemoryModel Logical GLSL450
      OpEntryPoint Fragment %1 "main" %2
      OpExecutionMode %1 OriginUpperLeft
 %3 = OpTypeVoid
 %4 = OpTypeFunction %3
 %5 = OpTypeFloat 32
 %6 = OpTypeInt 32 0
 %7 = OpTypeVector %5 4
 %8 = OpTypePointer Output %7
 %2 = OpVariable %8 Output
 %9 = OpConstant %5 0.4
%10 = OpConstant %5 0.8
%11 = OpConstant %6 1
%12 = OpConstantComposite %7 %9 %9 %10 %11
 %1 = OpFunction %3 None %4
%13 = OpLabel
      OpStore %2 %12
      OpReturn
      OpFunctionEnd
```

```
error: 61: OpConstantComposite
Constituent <id> '11's type does
not match Result Type <id> '7's
vector element type.
```

SPIR-V Tools – spirv-val

```
      OpCapability Shader
      OpMemoryModel Logical GLSL450
      OpEntryPoint Fragment %1 "main" %2
      OpExecutionMode %1 OriginUpperLeft
 %3 = OpTypeVoid
 %4 = OpTypeFunction %3
 %5 = OpTypeFloat 32
 %6 = OpTypeInt 32 0
 %7 = OpTypeVector %5 4
 %8 = OpTypePointer Output %7
 %2 = OpVariable %8 Output
 %9 = OpConstant %5 0
%10 = OpConstant %5 0
%11 = OpConstant %6 1
%12 = OpConstantComposite %7 %9 %9 %10 %11
 %1 = OpFunction %3 None %4
%13 = OpLabel
      OpStore %2 %12
      OpReturn
      OpFunctionEnd
```

```
error: 61: OpConstantComposite
Constituent <id> '11's type does
not match Result Type <id> '7's
vector element type.
```

- %11 is of type %6

SPIR-V Tools – spirv-val

```
      OpCapability Shader
      OpMemoryModel Logical GLSL450
      OpEntryPoint Fragment %1 "main" %2
      OpExecutionMode %1 OriginUpperLeft
 %3 = OpTypeVoid
 %4 = OpTypeFunction %3
 %5 = OpTypeFloat 32
 %6 = OpTypeInt 32 0
 %7 = OpTypeVector %5 4
 %8 = OpTypePointer Output %7
 %2 = OpVariable %8 Output
 %9 = OpConstant %5 0.4
%10 = OpConstant %5 0.8
%11 = OpConstant %6 1
%12 = OpConstantComposite %7 %9 %9 %10 %11
 %1 = OpFunction %3 None %4
%13 = OpLabel
      OpStore %2 %12
      OpReturn
      OpFunctionEnd
```

```
error: 61: OpConstantComposite
Constituent <id> '11's type does
not match Result Type <id> '7's
vector element type.
```

- %11 is of type %6
- %6 is a 32 bit unsigned integer

SPIR-V Tools – spirv-val

```
      OpCapability Shader
      OpMemoryModel Logical GLSL450
      OpEntryPoint Fragment %1 "main" %2
      OpExecutionMode %1 OriginUpperLeft
 %3 = OpTypeVoid
 %4 = OpTypeFunction %3
 %5 = OpTypeFloat 32
 %6 = OpTypeInt 32 0
 %7 = OpTypeVector %5 4
 %8 = OpTypePointer Output %7
 %2 = OpVariable %8 Output
 %9 = OpConstant %5 0.4
%10 = OpConstant %5 0.8
%11 = OpConstant %6 1
%12 = OpConstantComposite %7 %9 %9 %10 %11
 %1 = OpFunction %3 None %4
%13 = OpLabel
      OpStore %2 %12
      OpReturn
      OpFunctionEnd
```

```
error: 61: OpConstantComposite
Constituent <id> '11's type does
not match Result Type <id> '7's
vector element type.
```

- %11 is of type %6
- %6 is a 32 bit unsigned integer
- %12 is of type %7

SPIR-V Tools – spirv-val

```
      OpCapability Shader
      OpMemoryModel Logical GLSL450
      OpEntryPoint Fragment %1 "main" %2
      OpExecutionMode %1 OriginUpperLeft
 %3 = OpTypeVoid
 %4 = OpTypeFunction %3
 %5 = OpTypeFloat 32
 %6 = OpTypeInt 32 0
 %7 = OpTypeVector %5 4
 %8 = OpTypePointer Output %7
 %2 = OpVariable %8 Output
 %9 = OpConstant %5 0.4
%10 = OpConstant %5 0.8
%11 = OpConstant %6 1
%12 = OpConstantComposite %7 %9 %9 %10 %11
 %1 = OpFunction %3 None %4
%13 = OpLabel
      OpStore %2 %12
      OpReturn
      OpFunctionEnd
```

```
error: 61: OpConstantComposite
Constituent <id> '11's type does
not match Result Type <id> '7's
vector element type.
```

- %11 is of type %6
- %6 is a 32 bit unsigned integer
- %12 is of type %7
- %7 is a vector of 4 32 bit floating point

SPIR-V Tools – spirv-val

```
      OpCapability Shader
      OpMemoryModel Logical GLSL450
      OpEntryPoint Fragment %1 "main" %2
      OpExecutionMode %1 OriginUpperLeft
 %3 = OpTypeVoid
 %4 = OpTypeFunction %3
 %5 = OpTypeFloat 32
 %6 = OpTypeInt 32 0
 %7 = OpTypeVector %5 4
 %8 = OpTypePointer Output %7
 %2 = OpVariable %8 Output
 %9 = OpConstant %5 0.4
%10 = OpConstant %5 0.8
%11 = OpConstant %6 1
%12 = OpConstantComposite %7 %9 %9 %10 %11
 %1 = OpFunction %3 None %4
%13 = OpLabel
      OpStore %2 %12
      OpReturn
      OpFunctionEnd
```

```
error: 61: OpConstantComposite
Constituent <id> '11's type does
not match Result Type <id> '7's
vector element type.
```

- %11 is of type %6
- %6 is a 32 bit unsigned integer
- %12 is of type %7
- %7 is a vector of 4 32 bit floating point
- We are mixing types when creating the OpConstantComposite!

SPIR-V Tools – spirv-remap

Another Khronos provided tool is spirv-remap:

```
./spirv-remap -i our_shader.spv --map all -o .
```

It takes a collection of input SPIR-V binaries and modifies them such that similar opcodes will use the same IDs

SPIR-V Tools – spirv-remap

Another Khronos provided tool is spirv-remap:

```
./spirv-remap -i our_shader.spv --map all -o .
```

-i one or more input SPIR-V binaries

SPIR-V Tools – spirv-remap

Another Khronos provided tool is spirv-remap:

```
./spirv-remap -i our_shader.spv --map all -o .
```

--map all enables the remapping

SPIR-V Tools – spirv-remap

Another Khronos provided tool is spirv-remap:

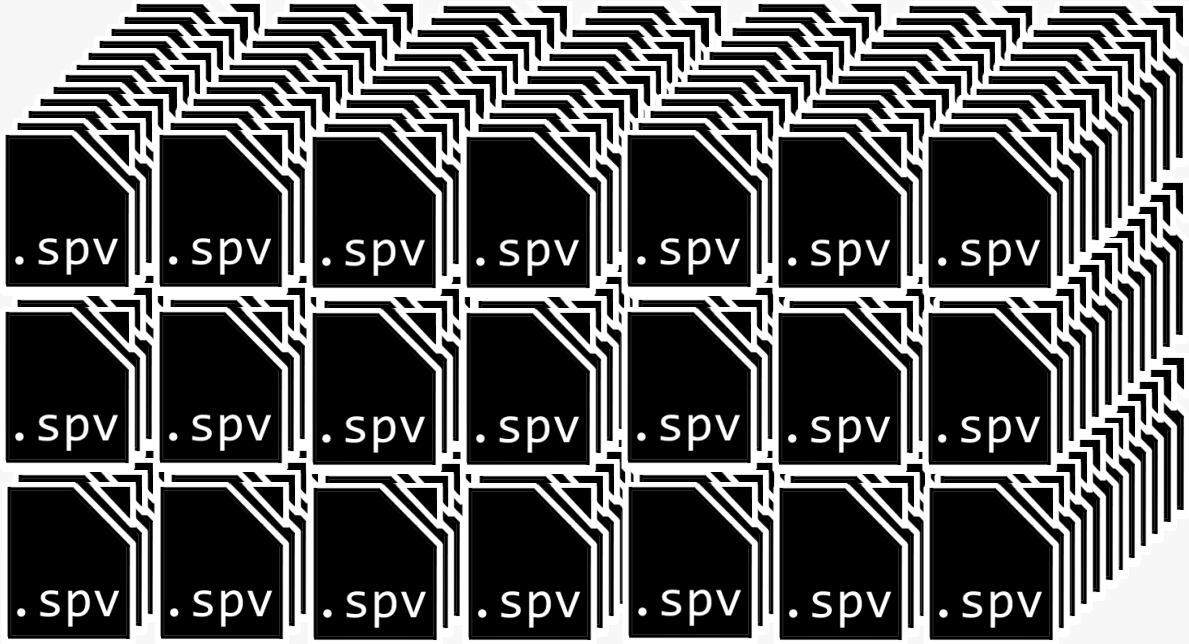
```
./spirv-remap -i our_shader.spv --map all -o .
```

-o <folder> outputs all remapping SPIR-V binaries into a folder

SPIR-V Tools – spirv-remap

Why is this useful?

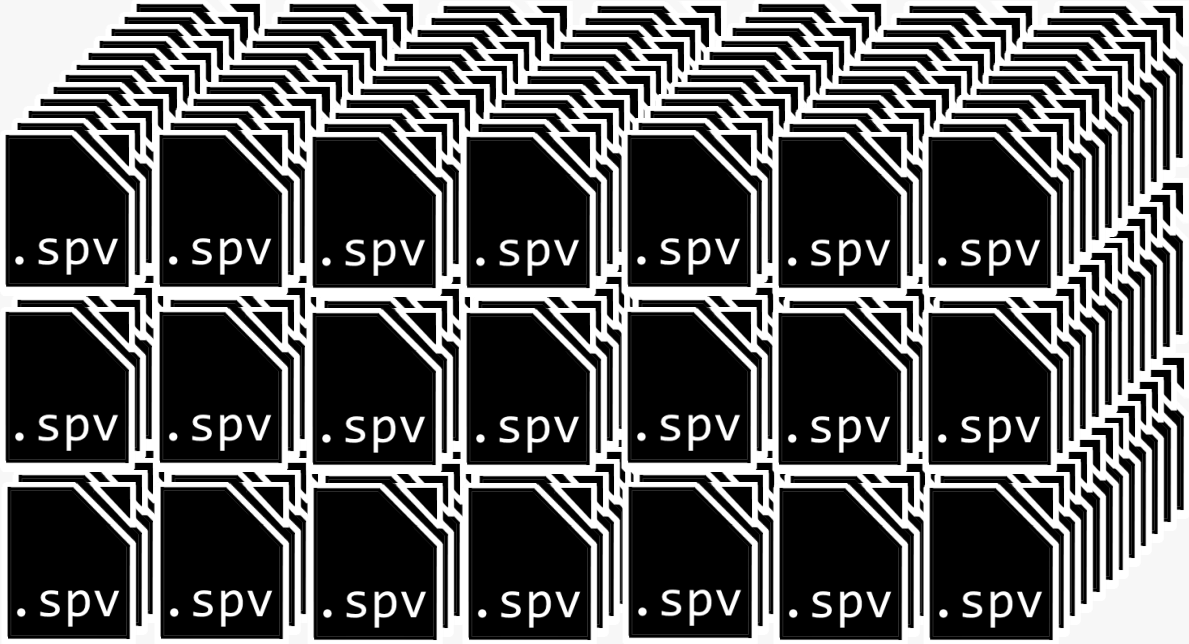
- Imagine we had a ton of SPIR-V binaries



SPIR-V Tools – spirv-remap

Why is this useful?

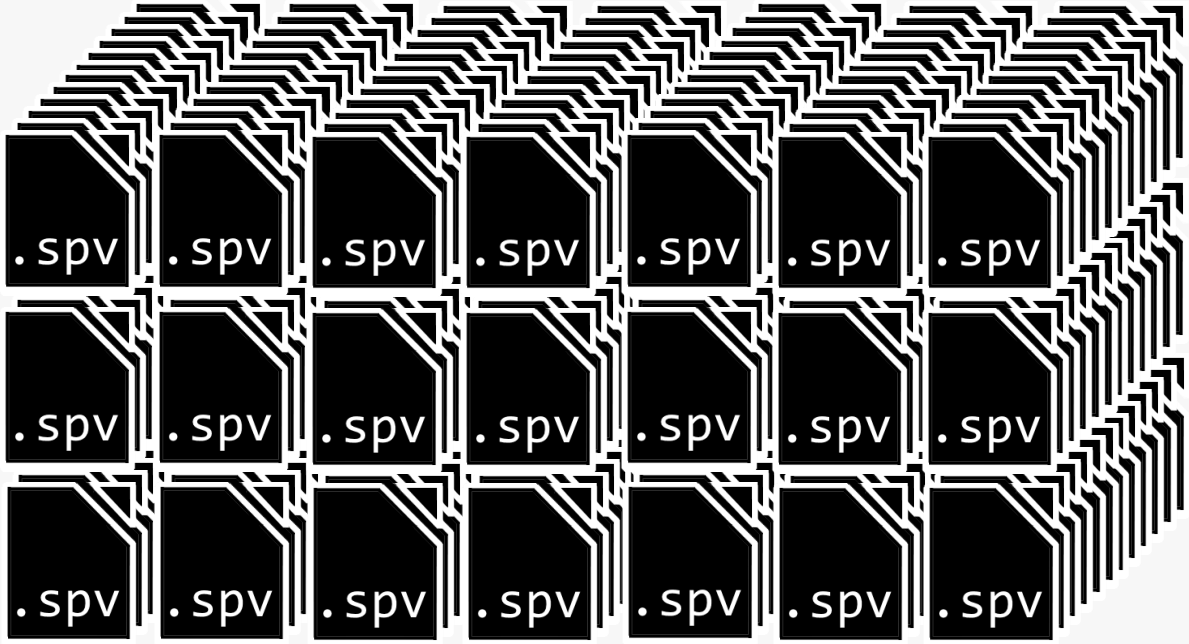
- Imagine we had a ton of SPIR-V binaries
- We'd compress these before shipping them with our game



SPIR-V Tools – spirv-remap

Why is this useful?

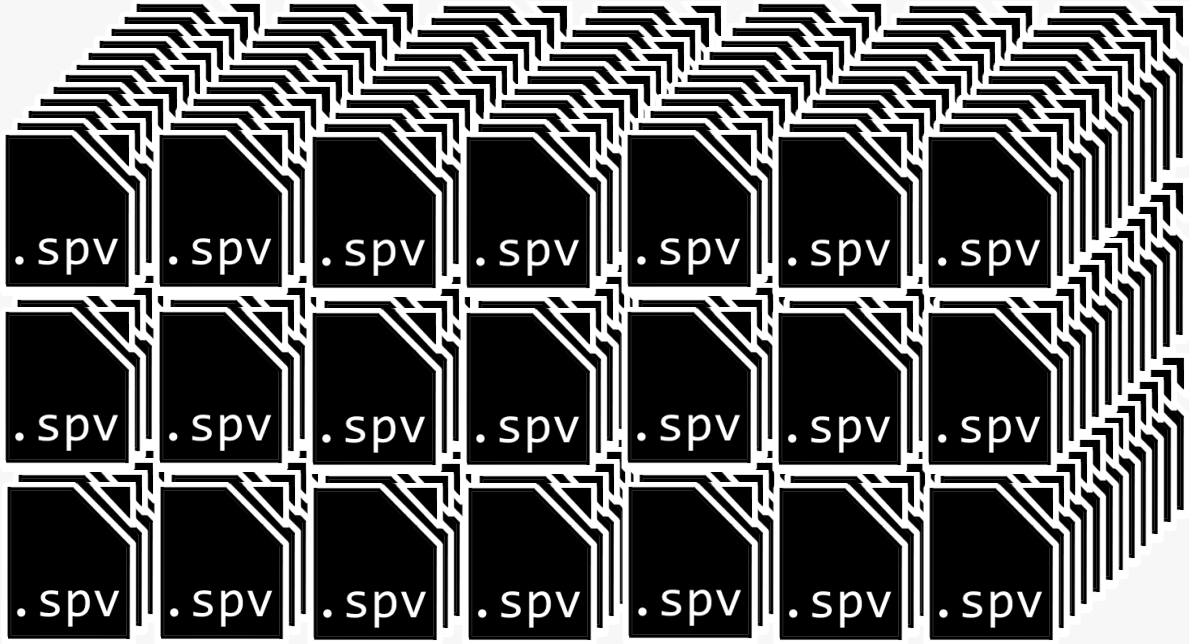
- Imagine we had a ton of SPIR-V binaries
- We'd compress these before shipping them with our game
- If we can make similar opcodes have similar IDs across all our files



SPIR-V Tools – spirv-remap

Why is this useful?

- Imagine we had a ton of SPIR-V binaries
- We'd compress these before shipping them with our game
- If we can make similar opcodes have similar IDs across all our files
- We'll compress our SPIR-V binaries substantially more

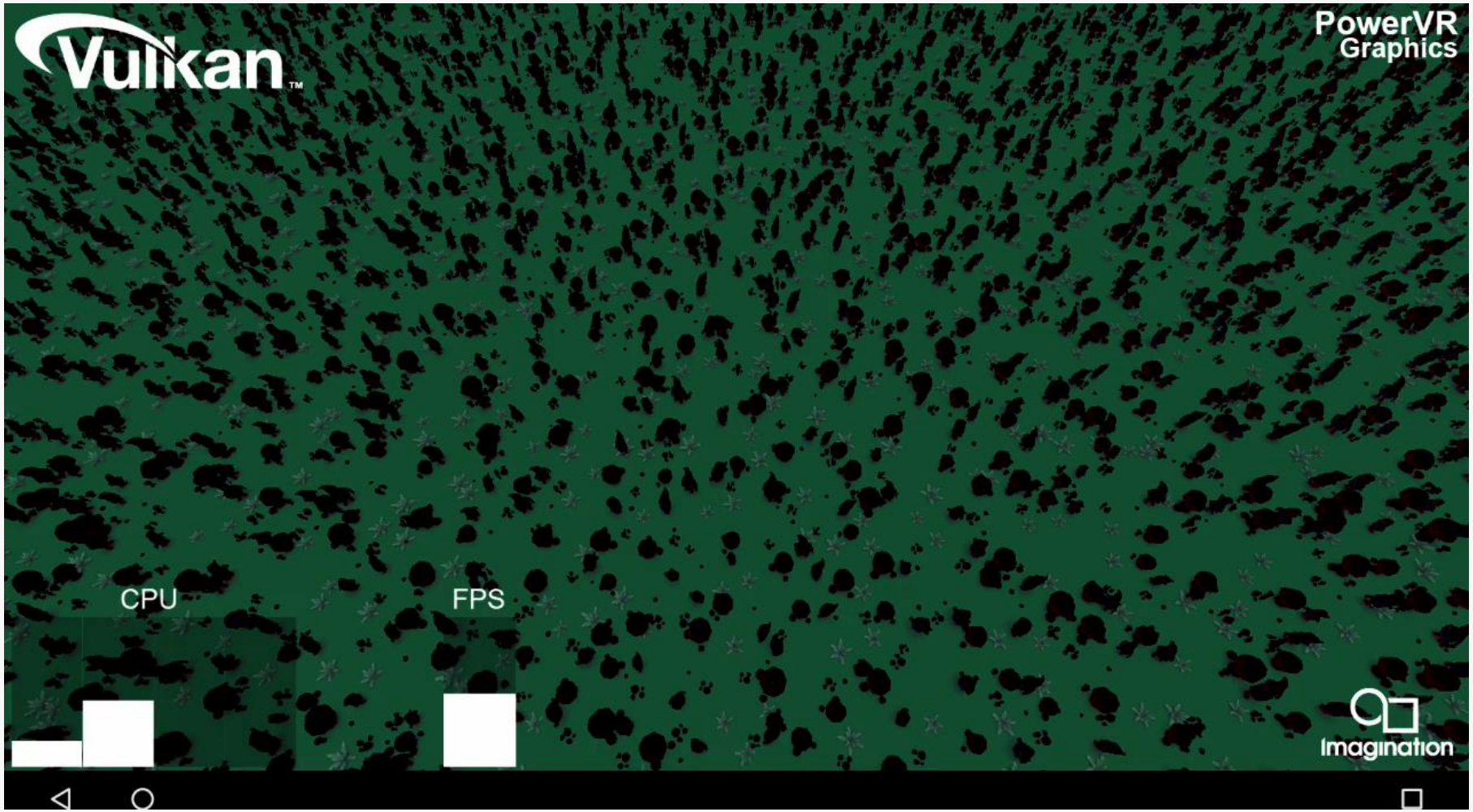


Example Time

Three examples of where these tools will be useful:

- Broken shader
- Unoptimal SPIR-V
- More unoptimal SPIR-V

Broken shader



Broken shader

Check that the shader you wrote matches your intent

```
#version 450

layout(set = 0, binding = 0) uniform sampler2D tex;

layout(location = 0) in lowp vec2 uvs;

layout(location = 1) in lowp float light_dot_norm;

layout(location = 0) out lowp vec4 out_colour;

void main() {
    lowp vec3 ambient = vec3(0.1, 0.0, 0.15);
    out_colour = vec4(ambient +
        light_dot_norm * texture(tex, uvs).rgb, 1.0);
}
```

Broken shader

Compile the GLSL of our fragment shader to SPIR-V:

```
./glslang –V –o our_shader.spv our_shader.frag
```

Then validate the SPIR-V:

```
./spirv-val our_shader.spv
```

If the validator succeeds, we then disassemble the SPIR-V:

```
./spirv-dis –o our_shader.spvasm our_shader.spv
```

Broken shader

```
; SPIR-V
; Version: 1.0
; Generator: Khronos Glslang Reference Front End; 1
; Bound: 39
; Schema: 0
                OpCapability Shader
          %1 = OpExtInstImport "GLSL.std.450"
                OpMemoryModel Logical GLSL450
                OpEntryPoint Fragment %4 "main" %28 %19 %16
                OpExecutionMode %4 OriginLowerLeft
                OpSource GLSL 450
                OpName %4 "main"
                OpName %9 "ambient"
                OpName %16 "out_colour"
                OpName %19 "light_dot_norm"
                OpName %24 "tex"
                OpName %28 "uvs"
                OpDecorate %16 Location 0
                OpDecorate %19 Location 1
                OpDecorate %24 DescriptorSet 0
                OpDecorate %24 Binding 0
                OpDecorate %28 Location 0
          %2 = OpTypeVoid
          %3 = OpTypeFunction %2
          %6 = OpTypeFloat 32
          %7 = OpTypeVector %6 3
          %8 = OpTypePointer Function %7
         %14 = OpTypeVector %6 4
                ...
```

Broken shader

```
; SPIR-V
; Version: 1.0
; Generator: Khronos Glslang Reference Front End; 1
; Bound: 39
; Schema: 0
          OpCapability Shader
     %1 = OpExtInstImport "GLSL.std.450"
          OpMemoryModel Logical GLSL450
          OpEntryPoint Fragment %4 "main" %28 %19 %16
          OpExecutionMode %4 OriginLowerLeft
          OpSource GLSL 450
          OpName %4 "main"
          OpName %9 "ambient"
          OpName %16 "out_colour"
          OpName %19 "light_dot_norm"
          OpName %24 "tex"
          OpName %28 "uvs"
          OpDecorate %16 Location 0
          OpDecorate %19 Location 1
          OpDecorate %24 DescriptorSet 0
          OpDecorate %24 Binding 0
          OpDecorate %28 Location 0
     %2 = OpTypeVoid
     %3 = OpTypeFunction %2
     %6 = OpTypeFloat 32
     %7 = OpTypeVector %6 3
     %8 = OpTypePointer Function %7
    %14 = OpTypeVector %6 4
          ...
```

```
          ...
   %15 = OpTypePointer Output %14
   %18 = OpTypePointer Input %6
   %21 = OpTypeImage %6 2D 0 0 0 1 Unknown
   %22 = OpTypeSampledImage %21
   %23 = OpTypePointer UniformConstant %22
   %26 = OpTypeVector %6 2
   %27 = OpTypePointer Input %26
   %10 = OpConstant %6 0.1
   %11 = OpConstant %6 0
   %12 = OpConstant %6 0.15
   %13 = OpConstantComposite %7 %10 %11 %12
   %16 = OpVariable %15 Output
   %19 = OpVariable %18 Input
   %24 = OpVariable %23 UniformConstant
   %28 = OpVariable %27 Input
   %34 = OpConstant %6 1
          ...
```

Broken shader

```
      ...
 %4 = OpFunction %2 None %3
 %5 = OpLabel
 %9 = OpVariable %8 Function
      OpStore %9 %13
%17 = OpLoad %7 %9
%20 = OpLoad %6 %19
%25 = OpLoad %22 %24
%29 = OpLoad %26 %28
%30 = OpImageSampleImplicitLod %14 %25 %29
%31 = OpVectorShuffle %7 %30 %30 0 1 2
%32 = OpVectorTimesScalar %7 %31 %20
%33 = OpFSub %7 %17 %32
%35 = OpCompositeExtract %6 %33 0
%36 = OpCompositeExtract %6 %33 1
%37 = OpCompositeExtract %6 %33 2
%38 = OpCompositeConstruct %14 %35 %36 %37 %34
      OpStore %16 %38
      OpReturn
      OpFunctionEnd
```

```
void main() {
    lowp vec3 ambient =
        vec3(0.1, 0.0, 0.15);
    out_colour = vec4(ambient +
        light_dot_norm *
         texture(tex, uvs).rgb, 1.0);
}
```

Broken shader

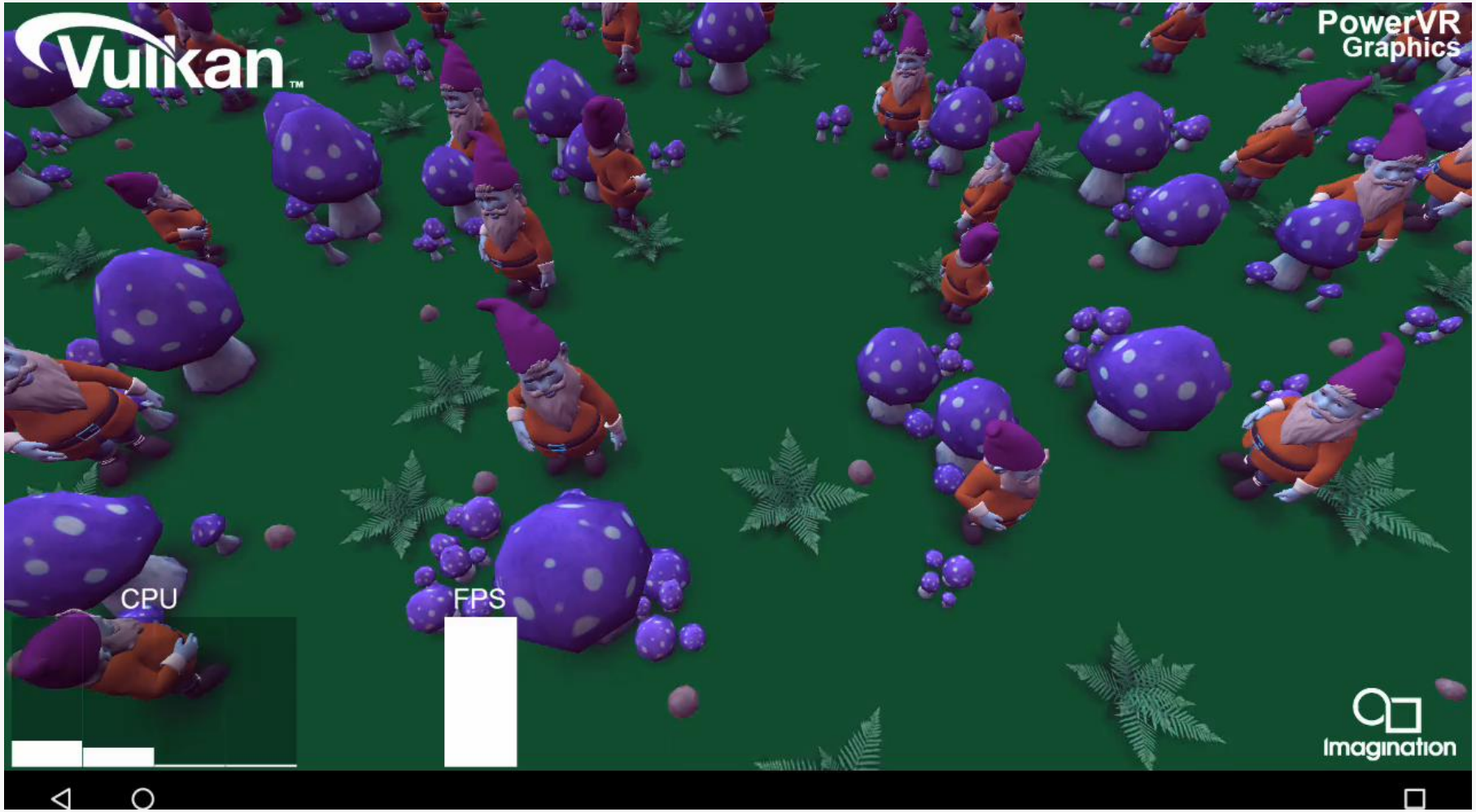
```
      ...
 %4 = OpFunction %2 None %3
 %5 = OpLabel
 %9 = OpVariable %8 Function
      OpStore %9 %13
%17 = OpLoad %7 %9
%20 = OpLoad %6 %19
%25 = OpLoad %22 %24
%29 = OpLoad %26 %28
%30 = OpImageSampleImplicitLod %14 %25 %29
%31 = OpVectorShuffle %7 %30 %30 0 1 2
%32 = OpVectorTimesScalar %7 %31 %20
%33 = OpFSub %7 %17 %32
%35 = OpCompositeExtract %6 %33 0
%36 = OpCompositeExtract %6 %33 1
%37 = OpCompositeExtract %6 %33 2
%38 = OpCompositeConstruct %14 %35 %36 %37 %34
      OpStore %16 %38
      OpReturn
      OpFunctionEnd
```

```
void main() {
    lowp vec3 ambient =
        vec3(0.1, 0.0, 0.15);
    out_colour = vec4(ambient +
        light_dot_norm *
         texture(tex, uvs).rgb, 1.0);
}
```

The silly compiler has subtracted from the ambient, instead of adding to it!*

* This bug was added for the purposes of this talk, no compiler is this dumb

Broken shader



Unoptimal SPIR-V

```
 %4 = OpFunction %2 None %3
 %5 = OpLabel
 %9 = OpVariable %8 Function
      OpStore %9 %13
%17 = OpLoad %7 %9
%20 = OpLoad %6 %19
%25 = OpLoad %22 %24
%29 = OpLoad %26 %28
%30 = OpImageSampleImplicitLod %14 %25 %29
%31 = OpVectorShuffle %7 %30 %30 0 1 2
%32 = OpVectorTimesScalar %7 %31 %20
%33 = OpFAdd %7 %17 %32
%35 = OpCompositeExtract %6 %33 0
%36 = OpCompositeExtract %6 %33 1
%37 = OpCompositeExtract %6 %33 2
%38 = OpCompositeConstruct %14 %35 %36 %37 %34
      OpStore %16 %38
      OpReturn
      OpFunctionEnd
```

Here is the same fragment shader we just used in the broken shader example

There is unoptimal code in here!

Unoptimal SPIR-V

Creating a variable

```
 %4 = OpFunction %2 None %3
 %5 = OpLabel
 %9 = OpVariable %8 Function
      OpStore %9 %13
%17 = OpLoad %7 %9
%20 = OpLoad %6 %19
%25 = OpLoad %22 %24
%29 = OpLoad %26 %28
%30 = OpImageSampleImplicitLod %14 %25 %29
%31 = OpVectorShuffle %7 %30 %30 0 1 2
%32 = OpVectorTimesScalar %7 %31 %20
%33 = OpFAdd %7 %17 %32
%35 = OpCompositeExtract %6 %33 0
%36 = OpCompositeExtract %6 %33 1
%37 = OpCompositeExtract %6 %33 2
%38 = OpCompositeConstruct %14 %35 %36 %37 %34
      OpStore %16 %38
      OpReturn
      OpFunctionEnd
```

Unoptimal SPIR-V

```
 %4 = OpFunction %2 None %3
 %5 = OpLabel
 %9 = OpVariable %8 Function
      OpStore %9 %13
%17 = OpLoad %7 %9
%20 = OpLoad %6 %19
%25 = OpLoad %22 %24
%29 = OpLoad %26 %28
%30 = OpImageSampleImplicitLod %14 %25 %29
%31 = OpVectorShuffle %7 %30 %30 0 1 2
%32 = OpVectorTimesScalar %7 %31 %20
%33 = OpFAdd %7 %17 %32
%35 = OpCompositeExtract %6 %33 0
%36 = OpCompositeExtract %6 %33 1
%37 = OpCompositeExtract %6 %33 2
%38 = OpCompositeConstruct %14 %35 %36 %37 %34
      OpStore %16 %38
      OpReturn
      OpFunctionEnd
```

Creating a variable

Storing a constant vector to it

Unoptimal SPIR-V

```
 %4 = OpFunction %2 None %3
 %5 = OpLabel
 %9 = OpVariable %8 Function
      OpStore %9 %13
%17 = OpLoad %7 %9
%20 = OpLoad %6 %19
%25 = OpLoad %22 %24
%29 = OpLoad %26 %28
%30 = OpImageSampleImplicitLod %14 %25 %29
%31 = OpVectorShuffle %7 %30 %30 0 1 2
%32 = OpVectorTimesScalar %7 %31 %20
%33 = OpFAdd %7 %17 %32
%35 = OpCompositeExtract %6 %33 0
%36 = OpCompositeExtract %6 %33 1
%37 = OpCompositeExtract %6 %33 2
%38 = OpCompositeConstruct %14 %35 %36 %37 %34
      OpStore %16 %38
      OpReturn
      OpFunctionEnd
```

Creating a variable

Storing a constant vector to it

Immediately reloading that variable!

Unoptimal SPIR-V

```
 %4 = OpFunction %2 None %3
 %5 = OpLabel
 %9 = OpVariable %8 Function
      OpStore %9 %13
%17 = OpLoad %7 %9
%20 = OpLoad %6 %19
%25 = OpLoad %22 %24
%29 = OpLoad %26 %28
%30 = OpImageSampleImplicitLod %14 %25 %29
%31 = OpVectorShuffle %7 %30 %30 0 1 2
%32 = OpVectorTimesScalar %7 %31 %20
%33 = OpFAdd %7 %17 %32
%35 = OpCompositeExtract %6 %33 0
%36 = OpCompositeExtract %6 %33 1
%37 = OpCompositeExtract %6 %33 2
%38 = OpCompositeConstruct %14 %35 %36 %37 %34
      OpStore %16 %38
      OpReturn
      OpFunctionEnd
```

Unoptimal SPIR-V

```
%4 = OpFunction %2 None %3
%5 = OpLabel
```

```
%20 = OpLoad %6 %19
%25 = OpLoad %22 %24
%29 = OpLoad %26 %28
%30 = OpImageSampleImplicitLod %14 %25 %29
%31 = OpVectorShuffle %7 %30 %30 0 1 2
%32 = OpVectorTimesScalar %7 %31 %20
%33 = OpFAdd %7 %17 %32
%35 = OpCompositeExtract %6 %33 0
%36 = OpCompositeExtract %6 %33 1
%37 = OpCompositeExtract %6 %33 2
%38 = OpCompositeConstruct %14 %35 %36 %37 %34
      OpStore %16 %38
      OpReturn
      OpFunctionEnd
```

Instead we can remove the variable, remove the store, and remove the load

Unoptimal SPIR-V

```
%4 = OpFunction %2 None %3
%5 = OpLabel

%20 = OpLoad %6 %19
%25 = OpLoad %22 %24
%29 = OpLoad %26 %28
%30 = OpImageSampleImplicitLod %14 %25 %29
%31 = OpVectorShuffle %7 %30 %30 0 1 2
%32 = OpVectorTimesScalar %7 %31 %20
%33 = OpFAdd %7 %13 %32
%35 = OpCompositeExtract %6 %33 0
%36 = OpCompositeExtract %6 %33 1
%37 = OpCompositeExtract %6 %33 2
%38 = OpCompositeConstruct %14 %35 %36 %37 %34
      OpStore %16 %38
      OpReturn
      OpFunctionEnd
```

Instead we can remove the variable, remove the store, and remove the load

And change the add opcode to use the constant instead!

More Unoptimal SPIR-V

Here's another fragment shader:

```
#version 450

layout(location = 0)
     out vec4 out_colour;


void main() {
  out_colour = vec4(
    sin(0.4), 0.4, 0.8, 1.0);
}
```

More Unoptimal SPIR-V

```
      OpCapability Shader
 %1 = OpExtInstImport "GLSL.std.450"
      OpMemoryModel Logical GLSL450
      OpEntryPoint Fragment %2 "main" %3
      OpExecutionMode %2 OriginUpperLeft
      OpDecorate %3 Location 0
 %4 = OpTypeVoid
 %5 = OpTypeFunction %4
 %6 = OpTypeFloat 32
 %7 = OpTypeVector %6 4
 %8 = OpTypePointer Output %7
 %3 = OpVariable %8 Output
 %9 = OpConstant %6 0.4
%10 = OpConstant %6 0.8
%11 = OpConstant %6 1
 %2 = OpFunction %4 None %5
%12 = OpLabel
%13 = OpExtInst %6 %1 Sin %9
%14 = OpCompositeConstruct %7 %13 %9 %10 %11
      OpStore %3 %14
      OpReturn
      OpFunctionEnd
```

Here’s another fragment shader:

```
#version 450

layout(location = 0)
     out vec4 out_colour;


void main() {
  out_colour = vec4(
    sin(0.4), 0.4, 0.8, 1.0);
}
```

More Unoptimal SPIR-V

```
      OpCapability Shader
 %1 = OpExtInstImport "GLSL.std.450"
      OpMemoryModel Logical GLSL450
      OpEntryPoint Fragment %2 "main" %3
      OpExecutionMode %2 OriginUpperLeft
      OpDecorate %3 Location 0
 %4 = OpTypeVoid
 %5 = OpTypeFunction %4
 %6 = OpTypeFloat 32
 %7 = OpTypeVector %6 4
 %8 = OpTypePointer Output %7
 %3 = OpVariable %8 Output
 %9 = OpConstant %6 0.4
%10 = OpConstant %6 0.8
%11 = OpConstant %6 1
 %2 = OpFunction %4 None %5
%12 = OpLabel
%13 = OpExtInst %6 %1 Sin %9
%14 = OpCompositeConstruct %7 %13 %9 %10 %11
      OpStore %3 %14
      OpReturn
      OpFunctionEnd
```

SPIR-V is perfectly valid – but unoptimal

- Computing sin(x), where x = 0.4
- This is known at compile time!
- We can replace the call to sin(x)!

More Unoptimal SPIR-V

```
      OpCapability Shader
  %1 = OpExtInstImport "GLSL.std.450"
      OpMemoryModel Logical GLSL450
      OpEntryPoint Fragment %2 "main" %3
      OpExecutionMode %2 OriginUpperLeft
      OpDecorate %3 Location 0
  %4 = OpTypeVoid
  %5 = OpTypeFunction %4
  %6 = OpTypeFloat 32
  %7 = OpTypeVector %6 4
  %8 = OpTypePointer Output %7
  %3 = OpVariable %8 Output
  %9 = OpConstant %6 0.4
 %10 = OpConstant %6 0.8
 %11 = OpConstant %6 1
%new = OpConstant %6 0.389418
  %2 = OpFunction %4 None %5
 %12 = OpLabel
 %13 = OpExtInst %6 %1 Sin %9
 %14 = OpCompositeConstruct %7 %13 %9 %10 %11
      OpStore %3 %14
      OpReturn
      OpFunctionEnd
```

Instead we can:

- Add a new constant = sin(0.4)

More Unoptimal SPIR-V

```
        OpCapability Shader
   %1 = OpExtInstImport "GLSL.std.450"
        OpMemoryModel Logical GLSL450
        OpEntryPoint Fragment %2 "main" %3
        OpExecutionMode %2 OriginUpperLeft
        OpDecorate %3 Location 0
   %4 = OpTypeVoid
   %5 = OpTypeFunction %4
   %6 = OpTypeFloat 32
   %7 = OpTypeVector %6 4
   %8 = OpTypePointer Output %7
   %3 = OpVariable %8 Output
   %9 = OpConstant %6 0.4
  %10 = OpConstant %6 0.8
  %11 = OpConstant %6 1
 %new = OpConstant %6 0.389418
   %2 = OpFunction %4 None %5
  %12 = OpLabel
  %13 = OpExtInst %6 %1 Sin %9
  %14 = OpCompositeConstruct %7 %13 %9 %10 %11
        OpStore %3 %14
        OpReturn
        OpFunctionEnd
```

Instead we can:

- Add a new constant = sin(0.4)
- Remove the original call to sin(x)

More Unoptimal SPIR-V

```
        OpCapability Shader
   %1 = OpExtInstImport "GLSL.std.450"
        OpMemoryModel Logical GLSL450
        OpEntryPoint Fragment %2 "main" %3
        OpExecutionMode %2 OriginUpperLeft
        OpDecorate %3 Location 0
   %4 = OpTypeVoid
   %5 = OpTypeFunction %4
   %6 = OpTypeFloat 32
   %7 = OpTypeVector %6 4
   %8 = OpTypePointer Output %7
   %3 = OpVariable %8 Output
   %9 = OpConstant %6 0.4
  %10 = OpConstant %6 0.8
  %11 = OpConstant %6 1
 %new = OpConstant %6 0.389418
   %2 = OpFunction %4 None %5
  %12 = OpLabel

  %14 = OpCompositeConstruct %7 %new %9 %10 %11
        OpStore %3 %14
        OpReturn
        OpFunctionEnd
```

Instead we can:

- Add a new constant = sin(0.4)
- Remove the original call to sin(x)
- Change the constant composite

More Unoptimal SPIR-V

```
        OpCapability Shader
  %1 = OpExtInstImport "GLSL.std.450"
        OpMemoryModel Logical GLSL450
        OpEntryPoint Fragment %2 "main" %3
        OpExecutionMode %2 OriginUpperLeft
        OpDecorate %3 Location 0
  %4 = OpTypeVoid
  %5 = OpTypeFunction %4
  %6 = OpTypeFloat 32
  %7 = OpTypeVector %6 4
  %8 = OpTypePointer Output %7
  %3 = OpVariable %8 Output
  %9 = OpConstant %6 0.4
 %10 = OpConstant %6 0.8
 %11 = OpConstant %6 1
%new = OpConstant %6 0.389418
  %2 = OpFunction %4 None %5
 %12 = OpLabel
 %14 = OpCompositeConstruct %7 %new %9 %10 %11
        OpStore %3 %14
        OpReturn
        OpFunctionEnd
```

Instead we can:

- Add a new constant = sin(0.4)
- Remove the original call to sin(x)
- Change the constant composite

Done!

Summary

We’ve covered a lot today:

- What SPIR-V is and why Vulkan uses it
- How to use glslang, spirv-dis, spirv-as, spirv-val and spirv-remap tools
- Some ‘out in the wild’ examples of these tools in action

We're Hiring!

codeplay.com/careers/

codeplay

THE HETEROGENEOUS SYSTEMS EXPERTS

Any Questions?

@sheredom