Using SPIR-V in practice with SPIRV-Cross

Hans-Kristian Arntzen
Engineer, ARM

Contents

- **Moving to offline compilation of SPIR-V**
- **Creating pipeline layouts with SPIRV-Cross**
  - Descriptor sets
  - Push constants
  - Multipass input attachments
- **Making SPIR-V portable to other graphics APIs**
- **Debugging complex shaders with your C++ debugger of choice**

KHRONOS GROUP

Offline Compilation to SPIR-V

- Shader compilation can be part of your build system
- Catching compilation bugs in build time is always a plus
- Strict, mature GLSL frontends available
  - glslang: https://github.com/KhronosGroup/glslang
  - shaderc: https://github.com/google/shaderc
- Full freedom for other languages in the future

```
# Makefile rules

FRAG_SHADERS := $(wildcard *.frag)
SPIRV_FILES :=
$(FRAG_SHADERS:.frag=.frag.spv)

shaders: $(SPIRV_FILES)

%.frag.spv: %.frag
    glslc –o $@ $< $(GLSL_FLAGS) –std=310es
```

Vulkan Pipeline Layouts

- Need to know the “function signature” of our shaders

```
pipelineInfo.layout = <layout goes here>;
vkCreateGraphicsPipelines(..., &pipelineInfo, ..., &pipeline);
```

The Contents of a Pipeline Layout

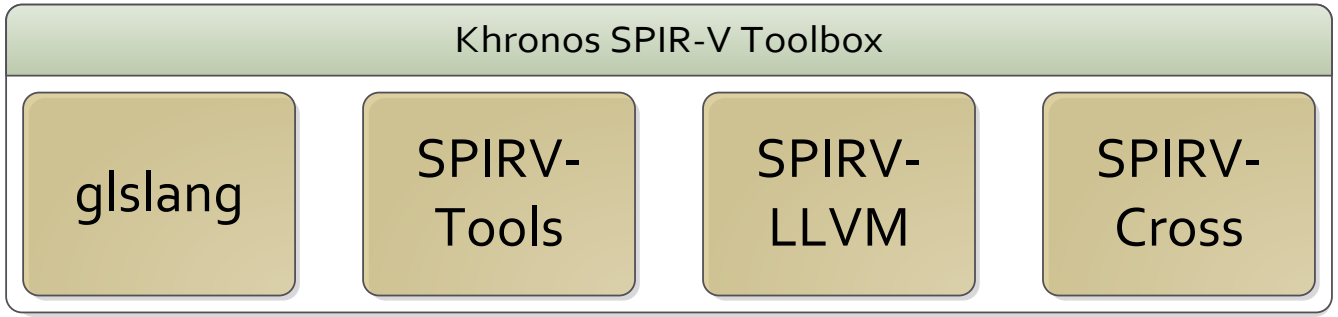

```
layout(set = 0, binding = 1) uniform UBO {
          mat4 MVP;
};
layout(set = 1, binding = 2) uniform sampler2D uTexture;
layout(push_constant) uniform PushConstants {
          vec4 FastConstant;
} constants;
```

- 16 bytes of push constant space
- Two descriptor sets
- Set #0 has one UBO at binding #1
- Set #1 has one combined image sampler at binding #2

- **Need to figure this out automatically, or write every layout by hand**
  - Latter is fine for tiny applications
  - Vulkan does not provide reflection here, after all, this is vendor neutral information

Introducing SPIRV-Cross

- SPIRV-Cross is a new tool hosted by Khronos
  - https://github.com/KhronosGroup/SPIRV-Cross
- Extensive reflection
- Decompilation to high level languages



Reflecting Uniforms and Samplers

- SPIRV-Cross has a simple API to retrieve resources

```
using namespace spirv_cross;

vector<uint32_t> spirv_binary = load_spirv_file();
Compiler comp(move(spirv_binary));

// The SPIR-V is now parsed, and we can perform reflection on it.
ShaderResources resources = comp.get_shader_resources();

for (auto &u : resources.uniform_buffers)
{
        uint32_t set = comp.get_decoration(u.id, spv::DecorationDescriptorSet);
        uint32_t binding = comp.get_decoration(u.id, spv::DecorationBinding);
        printf("Found UBO %s at set = %u, binding = %u!\n",
               u.name.c_str(), set, binding);
}
```

Stepping it up with Push Constants

- **SPIRV-Cross can figure out which push constant elements are in use**
  - Push constant blocks are typically shared across the various stages
  - Only parts of the push constant block are referenced in a single stage

```
layout(push_constant) uniform PushConstants {
          mat4 MVPInVertex;
          vec4 ColorInFragment;
} constants;

FragColor = constants.ColorInFragment; // Fragment only uses element #1.
```

```
uint32_t id = resources.push_constant_buffers[0].id;
vector<BufferRange> ranges = comp.get_active_buffer_ranges(id);
for (auto &range : ranges)
{
          printf("Accessing member #%u, offset %u, size %u\n",
                 range.index, range.offset, range.range);
}

// Possible to get names for struct members as well ☺
```

Subpass Input Attachments

- **Subpass attachments are similar to regular images**
  - Set
  - Binding
  - Input attachment index

```
layout(set = 0, binding = 0, input_attachment_index = 0) uniform subpassInput uAlbedo;
layout(set = 0, binding = 1, input_attachment_index = 1) uniform subpassInput uNormal;

vec4 lastColor = subpassLoad(uLastPass);
```

```
for (auto &attachment : resources.subpass_inputs)
{
        // ...
}
```

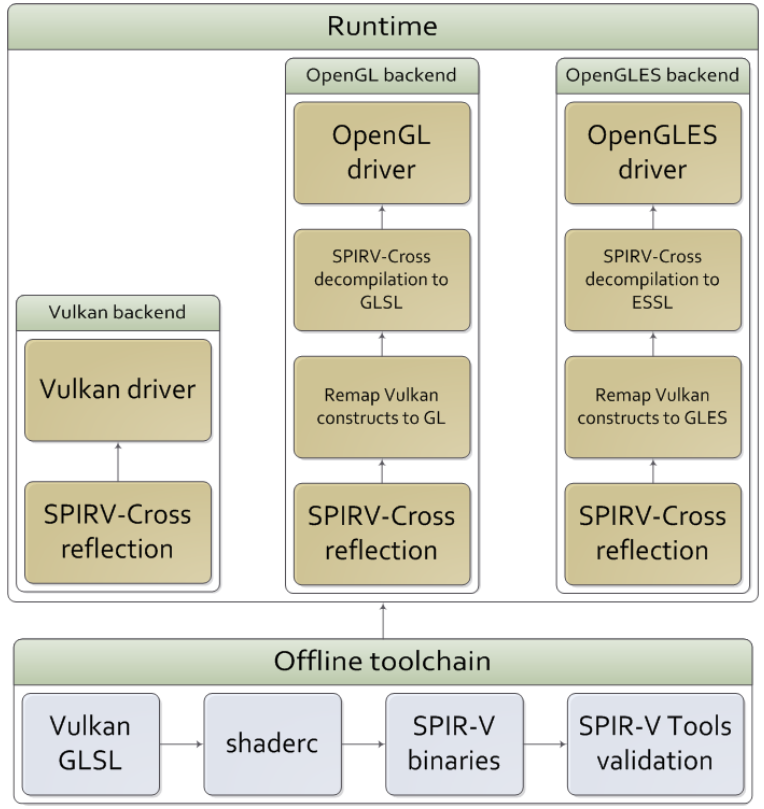
Taking SPIR-V Beyond Vulkan

- **SPIR-V is a great format to rally around**
  - Makes sense to be able to use it in older graphics APIs as well
- **Will take some time before exclusive Vulkan support is mainstream**
- **How to make use of Vulkan features while being compatible?**
  - Push constants
  - Subpass
  - Descriptor sets
- **Without tools, Vulkan features will be harder to take advantage of**

KHRONOS GROUP

GL + GLES + Vulkan Pipeline

- Implemented in our internal demo engine
- Write shaders in Vulkan GLSL
- Use Vulkan features directly
- No need for platform #ifdefs
- Can target mobile and desktop GL from same SPIR-V binary



Subpasses in OpenGL

- **The subpass attachment is really just a texture read from gl_FragCoord**
  - Enables reading directly from tile memory on tiled architectures
  - Great for deferred rendering and programmable blending

```
// Vulkan GLSL
uniform subpassInput uAlbedo;
...
FragColor = accumulateLight(
                              subpassLoad(uAlbedo),
                              subpassLoad(uNormal).xyz,
                              subpassLoad(uDepth).x);

// Translated to GLSL in SPIRV-Cross
uniform sampler2D uAlbedo;
...
FragColor = accumulateLight(
                              texelFetch(uAlbedo, ivec2(gl_FragCoord.xy), 0),
                              texelFetch(uNormal, ivec2(gl_FragCoord.xy), 0).xyz,
                              texelFetch(uDepth,  ivec2(gl_FragCoord.xy), 0).x);
```

KHRONOS GROUP

Push Constants in OpenGL

- **Push constants bundle up old-style uniforms into buffer blocks**
  - Translates directly to uniform structs
  - Use reflection to stamp out a list of glUniform() calls

```
// Vulkan GLSL
layout(push_constant) uniform PushConstants {
          vec4 Material;
} constants;

FragColor = constants.Material;

// Translated to GLSL in SPIRV-Cross
struct PushConstants {
          vec4 Material;
};
uniform PushConstants constants;

FragColor = constants.Material;
```

Descriptor Sets in OpenGL

- OpenGL has a binding space per type
- Find some remapping scheme that fits your application
- SPIRV-Cross can tweak bindings before decompiling to GLSL

```
// Vulkan GLSL
layout(set = 1, binding = 1) uniform sampler2D uTexture;

// SPIRV-Cross
uint32_t newBinding = 4;
glsl.set_decoration(texture.id, spv::DecorationBinding, newBinding);
glsl.unset_decoration(texture.id, spv::DecorationDescriptorSet);
string glslSource = glsl.compile();

// GLSL
layout(binding = 4) uniform sampler2D uTexture;
```

gl_InstanceIndex in OpenGL

- **Vulkan adds the base instance to the instance ID**
  - GL does not ☹
  - Workaround is to have GL backend pass in the base index as a uniform

```
// Vulkan GLSL
layout(set = 0, binding = 0) uniform UBO {
          mat4 MVPs[MAX_INSTANCES];
};

gl_Position = MVPs[gl_InstanceIndex] * Position;

// GLSL through SPIRV-Cross
layout(binding = 0) uniform UBO {
          mat4 MVPs[MAX_INSTANCES];
};
uniform int SPIRV_Cross_BaseInstance; // Supplied by application

gl_Position = MVPs[(gl_InstanceID + SPIRV_Cross_BaseInstance)] * Position;
```

KHRONOS GROUP

Debugging Shaders in C++

- **If you have thought …**
  - "I wish I could assert() in a compute shader"
  - "I wish I could instrument a shader with logging"
  - "I wish I could use clang address sanitizer to debug out-of-bounds access"
  - "I want to reproduce a shader bug outside the driver"
  - "I want to run regression tests when optimizing a shader"
  - "I want to step through a compute thread in <insert C++ debugger here>"
- **… the C++ backend in SPIRV-Cross could be interesting**
- **Still a very experimental feature**
- **Hope to expand this further in the future**

Basic Idea

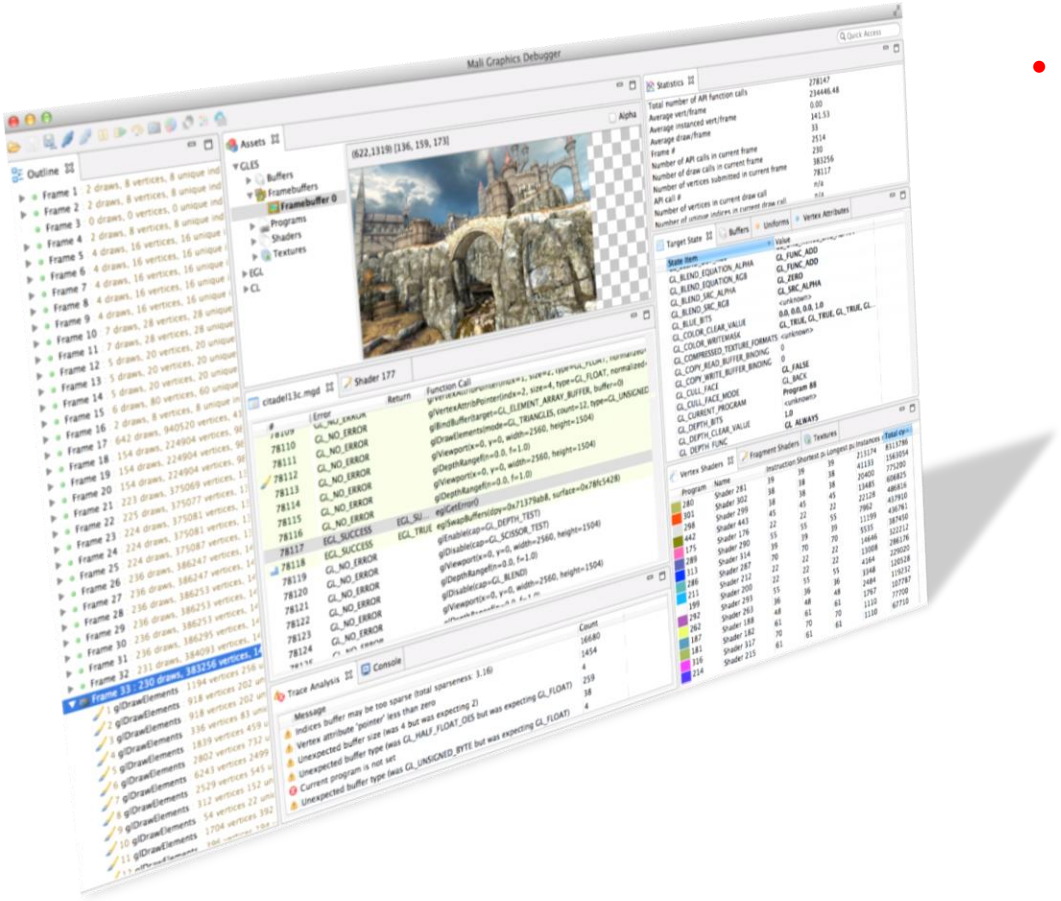
- **With GLM, C++ can be near GLSL compatible**
- **Reuse the GLSL backend to emit code which also works in C++**
  - Minor differences like references vs. in/out, etc
- **Add some scaffolding to redirect shader resources**
  - Easily done with macros, the actual C++ output is kept clean
- **The C++ output implements a simple C-compatible interface**
- **Add instrumentation to the C++ file as desired**
- **Compile C++ file to a dynamic library with debug symbols**
- **Instantiate from test program, bind buffers and invoke**
  - And have fun running shadertoy raymarchers at seconds per frame

On the Command Line


```
# Compile to SPIR-V
glslc –o test.spv test.comp

# Create C++ interface
spirv-cross --output test.cpp test.spv --cpp

# Add some instrumentation to the shader if you want
$EDITOR test.cpp

# Build library
g++ -o test.so –shared test.cpp –O0 –g –Iinclude/spirv_cross

# Run your test app
./<my app> --shader test.so
```

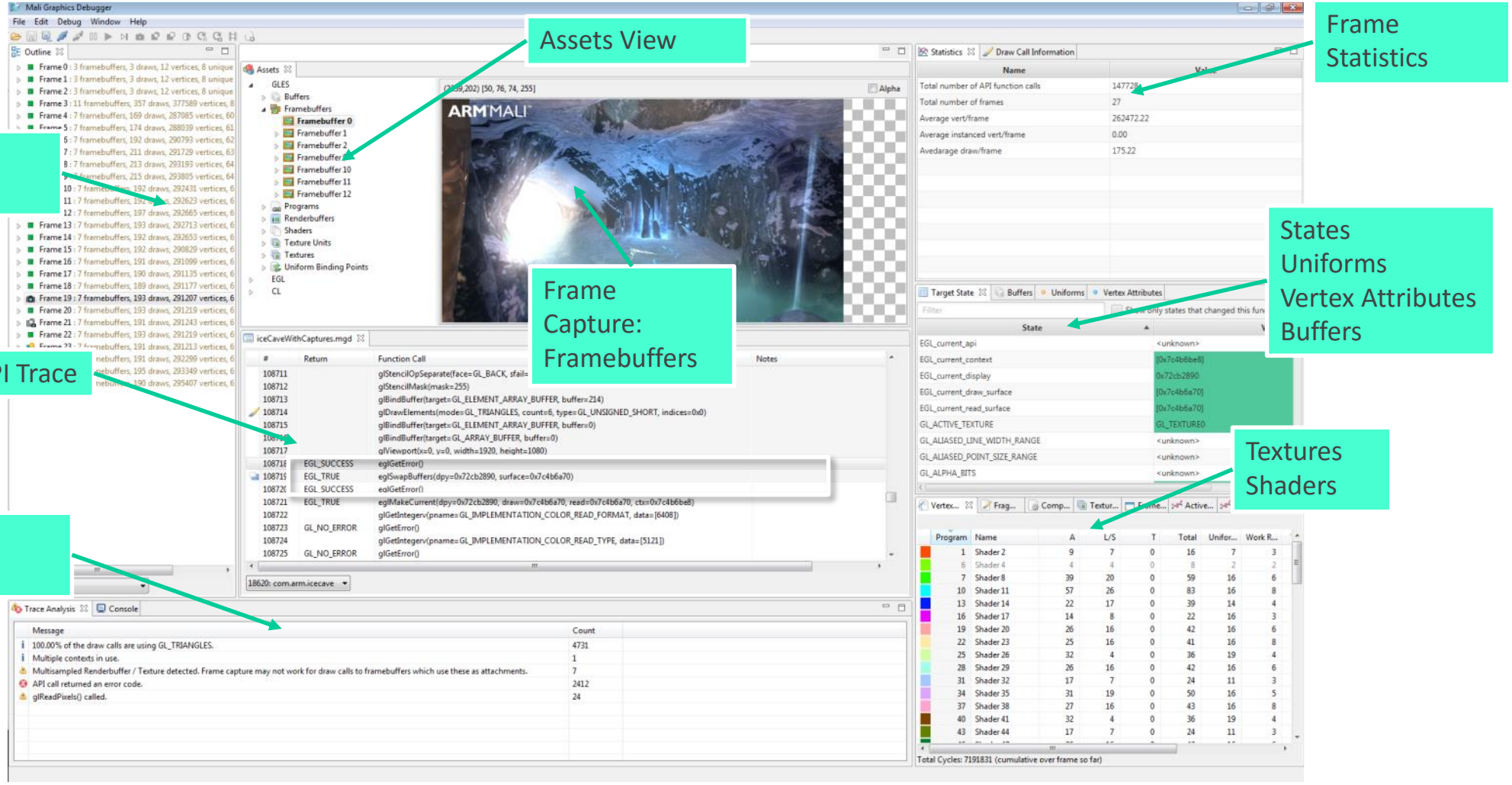
Another tool supporting Vulkan:

Mali Graphics Debugger is an advanced API tracer tool for Vulkan, OpenGL ES, EGL and OpenCL. It allows developers to trace their graphics and compute applications to debug issues and analyze the performance.



- **Vulkan Support**
  - Trace all the function calls in the SPEC.
  - Allows you to see exactly what calls compose your application.
  - Contact the Mali forums and we would love to get you setup. https://community.arm.com/groups/arm-mali-graphics

Investigation with the Mali Graphics Debugger



References

- **SPIRV-Cross**
  - https://github.com/KhronosGroup/SPIRV-Cross
- **Glslang**
  - https://github.com/KhronosGroup/glslang
- **Shaderc**
  - https://github.com/google/shaderc
- **SPIRV-Tools**
  - https://github.com/KhronosGroup/SPIRV-Tools
- **Mali Graphics Debugger**
  - http://malideveloper.arm.com/resources/tools/mali-graphics-debugger/