SPIRV-Cross: Taking SPIR-V to the next level

Hans-Kristian Arntzen
May 20th 2019 - Vulkanised 2019
The gist of SPIRV-Cross

SPIRV-Cross is a practical tool and library for performing reflection on SPIR-V and disassembling SPIR-V back to high level languages.
Last year at Vulkanised …

• The war stories
• Lots of technical detail
• Ranting is fun! :D
• Not trying to repeat
What happened?

- Went independent to work on SPIRV-Cross and other things
  - Been developing it since late 2015
- SPIRV-Cross development now funded by Valve 😊
- Committed to keep working on SPIRV-Cross going forward
Compile to SPIR-V

• Three major compilers
  - Glslang (Vulkan GLSL)
  - Glslang (HLSL)
  - DXC (HLSL)

• Emit SPIR-V directly from shader graph?

• Make your own language?
  - Scopes by Leonard Ritter (@paniq)

Texture2D MyTexture : register(t0);
SamplerState MySampler : register(s1);

float4 main(float2 UV : TEXCOORD0) : SV_Target
{
    return MyTexture.Sample(MySampler, UV);
}

Write in your favorite language once ...
I like GLSL though ;)
Target all the things!

```glsl
#version 100
precision mediump float;
precision highp int;

uniform highp sampler2D MyTextureMySampler;

varying highp vec2 UV;

void main()
{
  gl_FragData[0] = texture2D(MyTextureMySampler, UV);
}
```

```glsl
ESSL 1.0 / GL2

#version 450

uniform sampler2D MyTextureMySampler;

layout(location = 0) in vec2 UV;
layout(location = 0) out vec4 _epOutput;

void main()
{
  _epOutput = texture(MyTextureMySampler, UV);
}
```

```glsl
Modern GLSL / ESSL

#version 450

layout(set = 0, binding = 0) uniform texture2D MyTexture;
layout(set = 0, binding = 1) uniform sampler MySampler;

layout(location = 0) in vec2 UV;
layout(location = 0) out vec4 _entryPointOutput;

void main()
{
  _entryPointOutput = texture(sampler2D(MyTexture, MySampler), UV);
}
```

```glsl
Vulkan GLSL

#version 450

layout(set = 0, binding = 0) uniform texture2D MyTexture;
layout(set = 0, binding = 1) uniform sampler MySampler;

layout(location = 0) in vec2 UV;
layout(location = 0) out vec4 _entryPointOutput;

void main()
{
  _entryPointOutput = texture(sampler2D(MyTexture, MySampler), UV);
}
```
Texture2D<float4> MyTexture : register(t0);
SamplerState MySampler : register(s1);

static float2 UV;
static float4 _entryPointOutput;

struct SPIRV_Cross_Input
{
  float2 UV : TEXCOORD0;
};

struct SPIRV_Cross_Output
{
  float4 _entryPointOutput : SV_Target0;
};

float4 _main(float2 UV_1)
{
  return MyTexture.Sample(MySampler, UV_1);
}

void frag_main()
{
  float2 UV_1 = UV;
  float2 param = UV_1;
  _entryPointOutput = _main(param);
}

SPIRV_Cross_Output main(SPIRV_Cross_Input stage_input)
{
  UV = stage_input.UV;
  frag_main();
  SPIRV_Cross_Output stage_output;
  stage_output._entryPointOutput = _entryPointOutput;
  return stage_output;
}
```
#include <metal_stdlib>
#include <simd/simd.h>

using namespace metal;

struct main0_out
{
    float4 _entryPointOutput [[color(0)]];
};

struct main0_in
{
    float2 UV [[user(locn0)]];
};

fragment main0_out main0(main0_in in [[stage_in]],
    texture2d<float> MyTexture [[texture(0)]],
    sampler MySampler [[sampler(1)]])
{
    main0_out out = {};
    out._entryPointOutput = MyTexture.sample(MySampler, in.UV);
    return out;
}
```
The new wave of shader cross compilation

Andre Weissflog @RohOfWoe - Apr 14

diagnostic output for my WIP all-in-one shader-code-generator for sokol gfx.h, complete with SPIR-V disassembly :)

(built on top of glslang, SPIRV-Tools, SPIRV-Cross, getopt, fmtlib and pysting)

Stephan @stephanheigl - May 10

New blog post: Shader Cross-Compilation. How we built a simple SPIR-V based toolchain for HLSL to GLSL/MSL cross-compilation.
stephanheigl.github.io/posts/shader-c-

Special thanks to @Themaister for SPIR-V Cross

Robert Konrad @robertconrad

Replying to @shugomirnov @tibaradzic and 2 others

It is based on SPIRV-Cross nowadays, I switched to that one after contributing my HLSL code to it. It's by far the most mature open source shader cross-compilation thing around I think.

Arseny Kapoulkine @zeuxcg - Jan 9

Huge thanks to @Themaister for all the SPIR-V-Cross work. None of this would be possible without him; converting optimized SPIRv to sane GLSL/MSL and dealing with a myriad of weird differences is no small feat. Thanks to all contributors to glslang/spirv-opt as well.

nicebyte @nice_byte

SPIRV-Cross is magic!!
Vulkan portability initiative

Vulkan Applications

Open source SDK to build, run, and debug applications on macOS including validation layer support

Vulkan macOS SDK

macOS / iOS Run-time
Maps Vulkan to Metal

MoltenVK for macOS and iOS
For macOS 10.11, iOS 9.0 and up

Previously a paid product
Now released into OPEN SOURCE
Completely free to use - no fees or royalties - including for commercial applications

SPIRV-Cross
Convert SPIR-V shaders to platform source formats

Dota 2 running on Mac up to 50% faster than native OpenGL
Cross compilation to Vulkan GLSL is useful

- De-optimizer can be useful
  - SPIR-V -> Vulkan GLSL -> glslang
  - De-optimizes aggressive SSA to classic Load/Store
  - Has helped isolate driver bugs
- Debugging is very important
- Vulkan applications are captured with SPIR-V
The shader debugging cycle - RenderDoc

Decompile -> Edit -> Recompile -> See results
You don’t want to edit SPIR-V assembly 😊
Assembly horror

SPIR-V is not something you write by hand ...
Explore SPIRV-Cross output online

http://shader-playground.timjones.io/
The goal of SPIRV-Cross

- Enable SPIR-V to be the de-facto standard shader format
  - Ecosystem problem, not specification problem
- Portable shader pipelines are tedious and painful
  - HLSL
  - GLSL / ESSL / WebGL
  - MSL
- SPIR-V deserves better than being just the thing you throw into Vulkan
- Wider SPIR-V use drives better toolchains
  - Validation
  - Optimizers
  - GLSL/HLSL compilers targeting SPIR-V
  - New languages targeting SPIR-V, e.g. Scopes
  - Encourages new tooling around SPIR-V
- SPIR-V all the things!
Responsiveness in tooling projects

- Tools like SPIRV-Cross won’t ever be 100% complete and perfect
  - Too many edge cases
  - New extensions released all the time

- Compensate by being as responsive as possible
  - Quick bug fixes
  - Quick response and review of pull requests

- Don’t let issue count spiral out of control
  - Every issue gets visibility and is not lost

- Build trust with users
  - High confidence that reported issues will be fixed quickly
  - More likely that issues will actually be reported

- Easy to reproduce bugs
  - Standalone SPIR-V files reproduce > 90% of the time
  - Compiler projects are generally very lucky here

Don’t hesitate to file issues or feature requests
Commits over time

There aren’t any open pull requests.

You could search all of GitHub or try an advanced search.
GLSL backend

- Keeping up with latest additions to Vulkan GLSL
- Recent advanced additions to Vulkan GLSL
  - 8/16-bit arithmetic and storage support
  - Subgroup operations
  - Scalar block layout
  - Buffer reference (look ma’, pointers in GLSL!)
  - Bindless (nonuniformEXT qualifier)
- Advanced vendor extensions
  - VK_NV_ray_tracing support contributed by NVIDIA
  - All the AMD specific Vulkan GLSL extensions contributed by AMD
- A lot small fixes and tweaks to codegen which affects all backends
HLSL backend

• HLSL has been fairly quiet
  - Mostly minor bug fixes

• Little need for developers to target HLSL?
  - Most developers write in HLSL to begin with

• Mostly simple shaders?
  - Rare that bugs would be found?

• Still missing geometry shaders and tesellation support
  - One day ... 😊
The state of CPU targets

- An experimental C++ backend was added very early on
  - Relied on GLM for GLSL math
  - Never went anywhere
  - Deprecated/discontinued

- Intel picked up the torch

- SPIRV-Cross fork targeting ISPC!
  - No more terrible performance
  - Vectorized compute shaders on CPU
  - Subgroup threads map to vector lanes, just like GPUs
  - https://github.com/GameTechDev/SPIRV-Cross
Metal remains the most impactful backend

- A ton of work has gone into Metal backend support
- Only practical way to target MSL from a cross compiler with open source tools (I think?)
- Portability initiative
- Special thanks for Chip Davis for a lot of excellent contributions to the Metal backend over the last year
Metal tessellation

- Tessellation interface on Metal is very different from all previous APIs
  - «Think different»
- Vertex / tessellation control (Hull) must be emulated
  - Vertex shader with side effects for vertex stage
  - Compute kernel for control shaders
- Tessellator stage takes a GPU buffer of tessellation factors
- Supported in MoltenVK
  - Primary motivation seems to be running DXVK content
- Probably best to avoid tessellation if you can
```glsl
#version 450

layout(vertices = 4) out;

void main()
{
    gl_out[gl_InvocationID].gl_Position =
        gl_in[0].gl_Position +
        gl_in[1].gl_Position;

    if (gl_InvocationID == 0)
    {
        gl_TessLevelOuter[0] = 1.0;
        gl_TessLevelOuter[1] = 2.0;
        gl_TessLevelOuter[2] = 3.0;
        gl_TessLevelOuter[3] = 4.0;
        gl_TessLevelInner[0] = 5.0;
        gl_TessLevelInner[1] = 6.0;
    }
}

kernel void main0(/* A LOT OF STUFF */)
{
    // Write stage out to memory.
    device main0_out* gl_out =
        &spvOut[gl_PrimitiveID * 4];

    // Vertex -> Tess shenanigans.
    if (gl_InvocationID < spvIndirectParams[0])
        gl_in[gl_InvocationID] = in;
    threadgroup_barrier(mem_flags::mem_threadgroup);
    if (gl_InvocationID >= 4)
        return;

    // Shader
    gl_out[gl_InvocationID].gl_Position =
        gl_in[0].gl_Position +
        gl_in[1].gl_Position;
    if (gl_InvocationID == 0)
    {
        spvTessLevel[gl_PrimitiveID].
            edgeTessellationFactor[0] = half(1.0);
        // etc ...
    }
}
```

Pure pain

Metal horror story - Image view swizzling

- Gotta have some horror stories!
- VkImageView swizzle is not supported in Metal
  - Even GLES 3 supports this …
- Optional path to dynamically swizzle in shader …
  - Some games just need it
- Showing generated code here is a sin
  - Look up u32 swizzle code based on binding
  - Loop over all components and throw data into a switch block
- Try not to rely on VkImageView if targeting Metal
Metal indirect argument buffers

- Metal 2.0 feature
- Essentially VkDescriptorSet
- Less CPU overhead

```c
struct spvDescriptorSetBuffer0
{
    texture2d<float> MyTexture [[id(0)]];
    sampler MySampler [[id(1)]];
};

struct main0_out
{
    float4 _entryPointOutput [[color(0)]];
};

struct main0_in
{
    float2 UV [[user(locn0)]];
};

fragment main0_out main0(main0_in in [[stage_in]],
    constant spvDescriptorSetBuffer0& spvDescriptorSet0 [[buffer(0)]])
{
    main0_out out = {};
    out._entryPointOutput = spvDescriptorSet0.MyTexture.sample(spvDescriptorSet0.MySampler, in.UV);
    return out;
}
```
Variable pointer support

- **VK_KHR_variable_pointers**: OpenCL-on-Vulkan (clspv)
- **MSL has pointer support (C++ dialect)**

```c
struct foo
{
    int a;
};

struct bar
{
    int b;
};

device int* _24(device foo& a, device bar& b, thread uint3& gl_GlobalInvocationID)
{
    return (gl_GlobalInvocationID.x != 0u) ? &a.a : &b.b;
}

kernel void main0(device foo& x [[buffer(0)]], device bar& y [[buffer(1)]],
                  uint3 gl_GlobalInvocationID [[thread_position_in_grid]])
{
    device int* _34 = _24(x, y, gl_GlobalInvocationID);
    device int* _33 = _34;
    int _37 = x.a;
    *_33 = _37;
    y.b = _37 + _37;
}
```
Challenges in future shading models

- Pointers
- ...
- Pointers!?

- **Shader vs Kernel execution model**
  - OpenCL has full pointer support -> Physical pointers
  - Vulkan 1.1 -> Logical pointers

- **The shader model is inching towards pointer support**
  - SPV_KHR_variable_pointers -> Pointer to anything, but logical
  - SPV_EXT_physical_storage_buffer -> Bastard child of physical and logical

- GLSL and HLSL are too awkward to express all of this
- Flexible buffer packing rules keep me up at night
A new C API

• SPIRV-Cross’ interfaces are not API/ABI stable
  - C++ with lots of data structures flying around? Yeah …
  - Always intended SPIRV-Cross to be linked statically

• Spent a lot of time wrapping almost all of the C++ API in C
  - Committing to a stable API
  - … and stable ABI
  - Shared library support w/ so-versioning
  - Should be shippable in Linux distros

• Rust is all the rage right now
  - A Rust wrapper for the C API would be nice, if it does not exist already
Takeaways

- Make SPIR-V the main target for your shader pipeline
- Target all the things from SPIR-V

File issues on GitHub: [https://github.com/KhronosGroup/SPIRV-Cross](https://github.com/KhronosGroup/SPIRV-Cross)
  - Fixing bugs take priority
    - Need SPIR-V repro case
    - IP-sensitive repro cases can be arranged over e-mail
  - Feature requests
  - Missing target language functionality
  - Questions / support
Thanks!

Questions?

GitHub: Themaister / HansKristian-Work
Twitter: @themaister
E-mail: post@arntzen-software.no