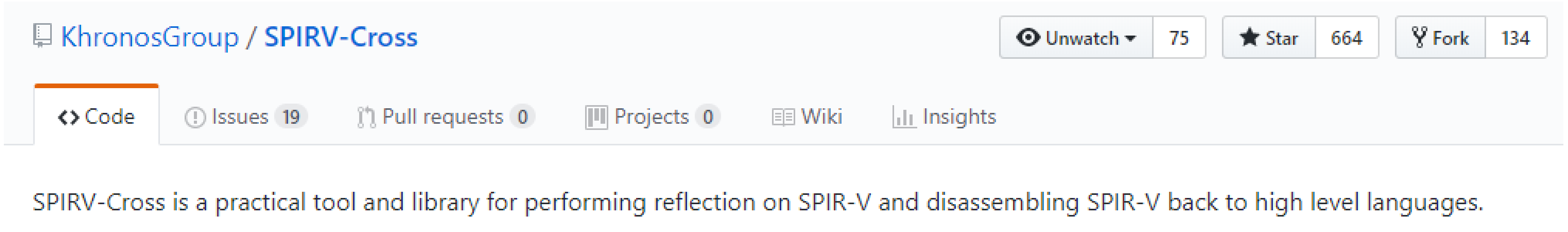
SPIRV-Cross:
Taking SPIR-V to the next level

Hans-Kristian Arntzen

May 20th 2019 – Vulkanised 2019

The gist of SPIRV-Cross


KhronosGroup / 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

arm

SPIRV-Cross

The war stories

- Hans-Kristian Arntzen
- 2018-05-22 – Vulkanise 2018

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 ;)

```
OpCapability Shader
          %1 = OpExtInstImport "GLSL.std.450"
               OpMemoryModel Logical GLSL450
               OpEntryPoint Fragment %main "main" %UV %_entryPointOutput
               OpExecutionMode %main OriginUpperLeft
               OpSource HLSL 500
               OpName %main "main"
               OpName %MyTexture "MyTexture"
               OpName %MySampler "MySampler"
               OpName %UV "UV"
               OpName %_entryPointOutput "@entryPointOutput"
               OpDecorate %MyTexture DescriptorSet 0
               OpDecorate %MyTexture Binding 0
               OpDecorate %MySampler DescriptorSet 0
               OpDecorate %MySampler Binding 1
               OpDecorate %UV Location 0
               OpDecorate %_entryPointOutput Location 0
       %void = OpTypeVoid
          %3 = OpTypeFunction %void
      %float = OpTypeFloat 32
    %v2float = OpTypeVector %float 2
    %v4float = OpTypeVector %float 4
         %14 = OpTypeImage %float 2D 0 0 0 1 Unknown
%_ptr_UniformConstant_14 = OpTypePointer UniformConstant %14
  %MyTexture = OpVariable %_ptr_UniformConstant_14 UniformConstant
                   mpler
                   8 = OpTypePointer UniformConstant %18
                   le %_ptr_UniformConstant_18 UniformConstant
                   mpledImage %14
                   pTypePointer Input %v2float
                   le %_ptr_Input_v2float Input
                   OpTypePointer Output %v4float
                   Variable %_ptr_Output_v4float Output
                   on %void None %3

                   v2float %UV
                   14 %MyTexture
                   18 %MySampler
                   dImage %22 %39 %40
         %43 = OpImageSampleImplicitLod %v4float %41 %31
               OpStore %_entryPointOutput %43
               OpReturn
               OpFunctionEnd
```

Target all the things!

ESSL 1.0 / GL2

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

uniform highp sampler2D MyTextureMySampler;

varying highp vec2 UV;

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

Modern GLSL / ESSL

```
#version 450

uniform sampler2D MyTextureMySampler;

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

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

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);
}
```

HLSL SM 5.0+ (D3D11+)

```
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;
}
```

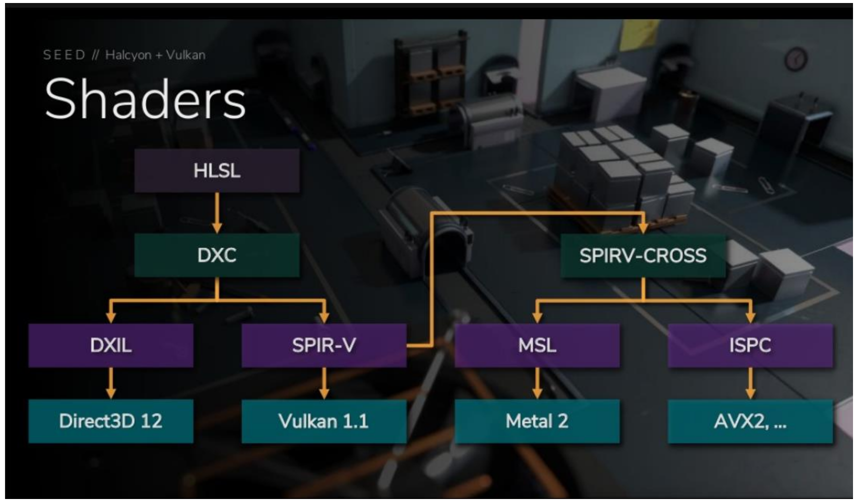
HLSL SM 3.0 (D3D9)

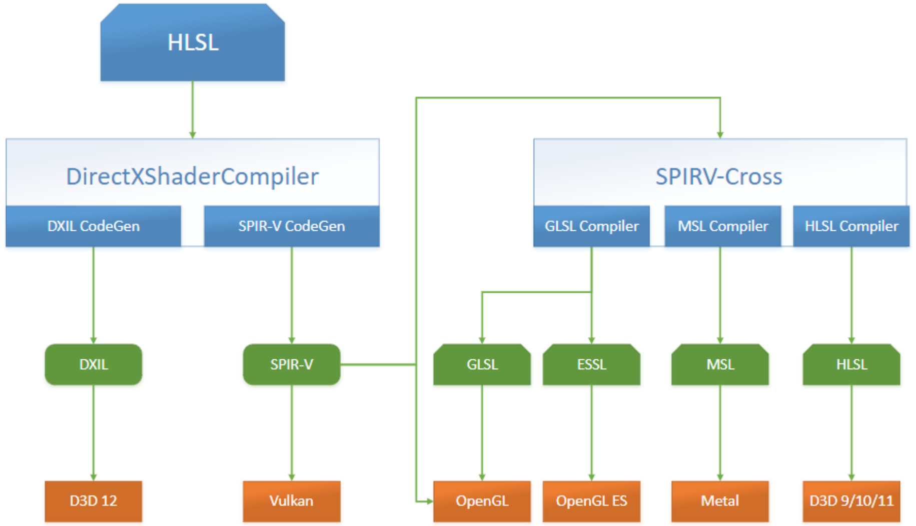
```
uniform sampler2D SPIRV_Cross_CombinedMyTextureMySampler;

static float2 UV;
static float4 _entryPointOutput;

struct SPIRV_Cross_Input
{
    float2 UV : TEXCOORD0;
};

struct SPIRV_Cross_Output
{
    float4 _entryPointOutput : COLOR0;
};

float4 _main(float2 UV_1)
{
    return tex2D(SPIRV_Cross_CombinedMyTextureMySampler, 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 = float4(_entryPointOutput);
    return stage_output;
}
```

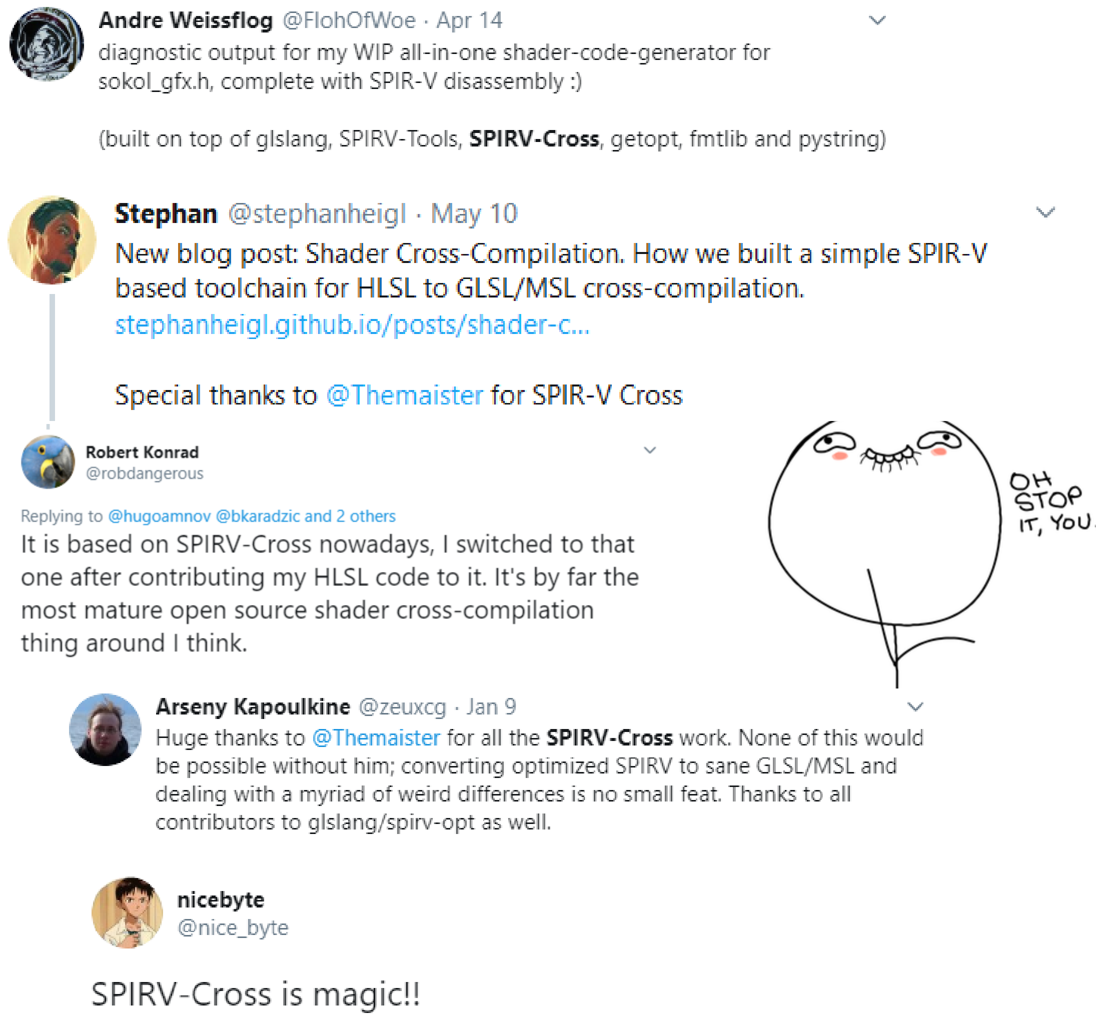
Metal 1.x/2.x

```
#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 @FlohOfWoe · 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 pystring)

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

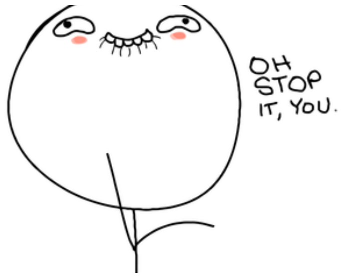
Robert Konrad
@robdangerous

Replying to @hugoamnov @bkaradzic 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.

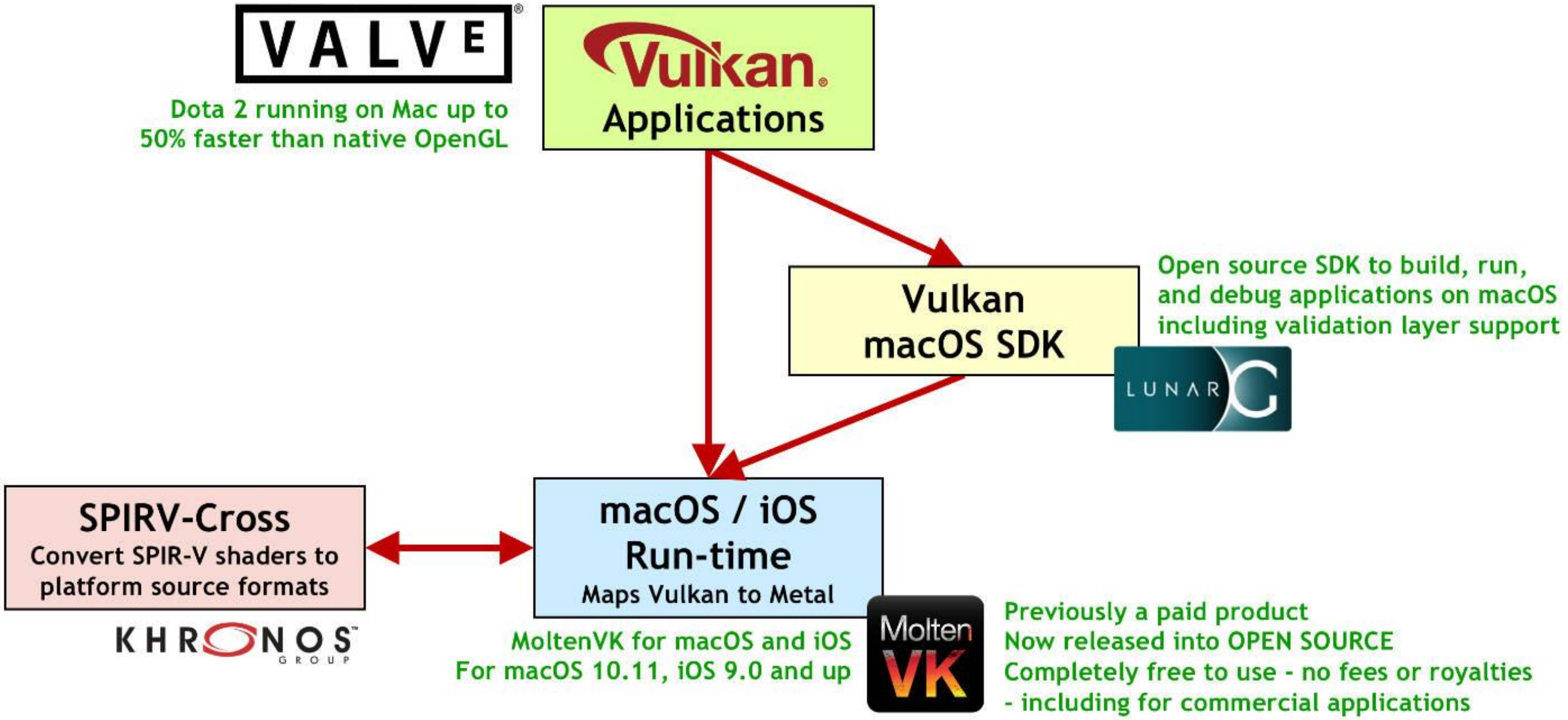
Arseny Kapoulkine @zeuxcg · Jan 9
Huge thanks to @Themaister for all the **SPIRV-Cross** work. None of this would be possible without him; converting optimized SPIRV to sane GLSL/MSL and dealing with a myriad of weird differences is no small feat. Thanks to all contributors to glslang/spirv-opt as well.

nicebyte
@nice_byte

SPIRV-Cross is magic!!



Vulkan portability initiative

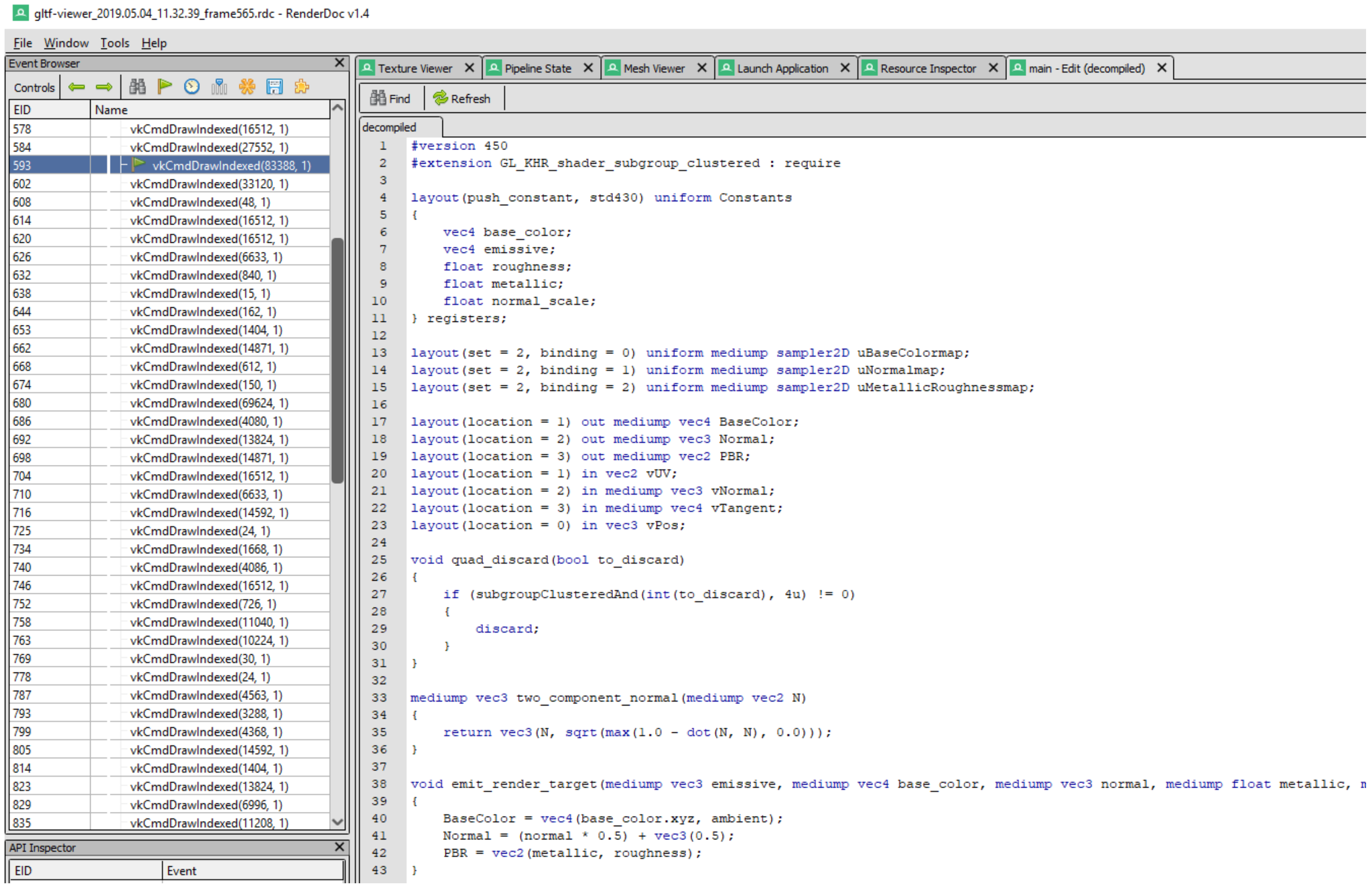

VALVE
Dota 2 running on Mac up to 50% faster than native OpenGL
Vulkan Applications
Vulkan macOS SDK
Open source SDK to build, run, and debug applications on macOS including validation layer support
LUNARG
SPIRV-Cross
Convert SPIR-V shaders to platform source formats
KHRONOS GROUP
macOS / iOS Run-time
Maps Vulkan to Metal
MoltenVK for macOS and iOS
For macOS 10.11, iOS 9.0 and up
Molten VK
Previously a paid product
Now released into OPEN SOURCE
Completely free to use - no fees or royalties
- including for commercial applications

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 ☺


gltf-viewer_2019.05.04_11.32.39_frame565.rdc - RenderDoc v1.4

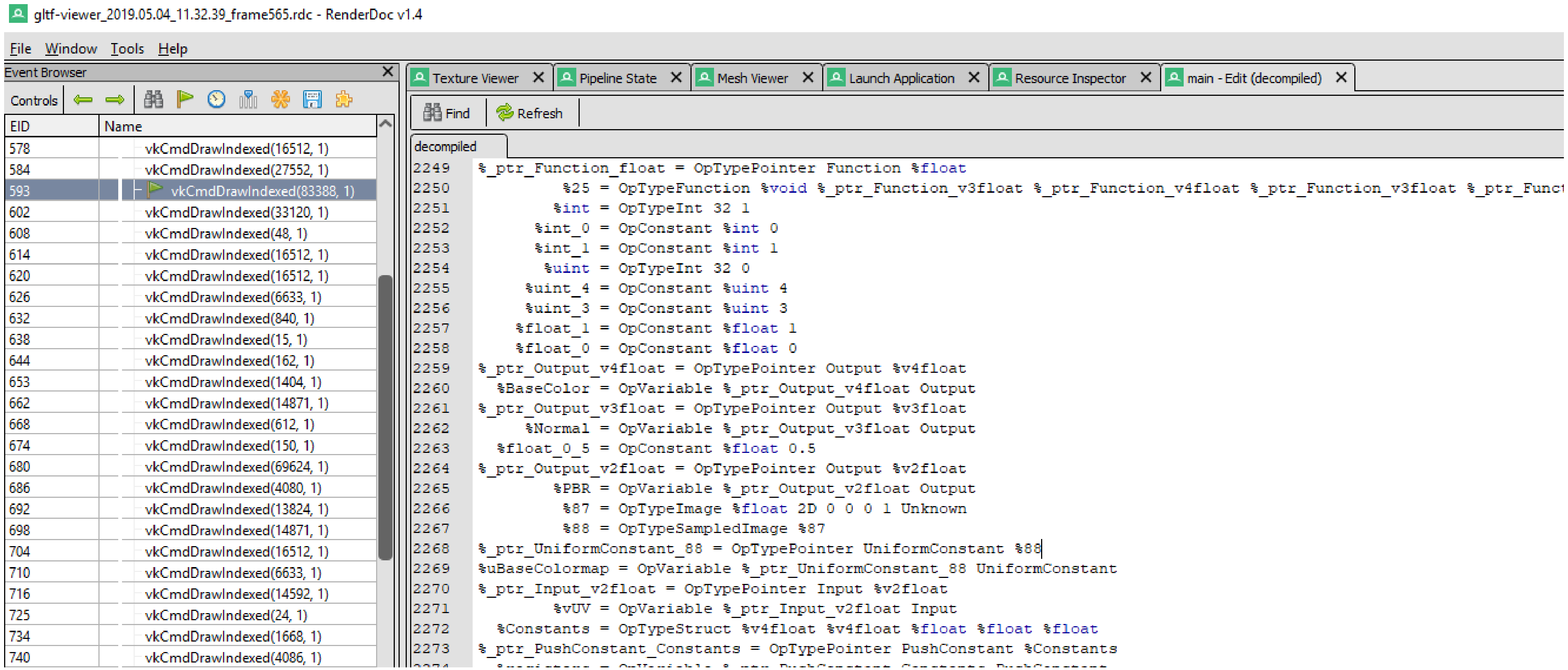
EID	Name
578	vkCmdDrawIndexed(16512, 1)
584	vkCmdDrawIndexed(27552, 1)
593	vkCmdDrawIndexed(83388, 1)
602	vkCmdDrawIndexed(33120, 1)
608	vkCmdDrawIndexed(48, 1)
614	vkCmdDrawIndexed(16512, 1)
620	vkCmdDrawIndexed(16512, 1)
626	vkCmdDrawIndexed(6633, 1)
632	vkCmdDrawIndexed(840, 1)
638	vkCmdDrawIndexed(15, 1)
644	vkCmdDrawIndexed(162, 1)
653	vkCmdDrawIndexed(1404, 1)
662	vkCmdDrawIndexed(14871, 1)
668	vkCmdDrawIndexed(612, 1)
674	vkCmdDrawIndexed(150, 1)
680	vkCmdDrawIndexed(69624, 1)
686	vkCmdDrawIndexed(4080, 1)
692	vkCmdDrawIndexed(13824, 1)
698	vkCmdDrawIndexed(14871, 1)
704	vkCmdDrawIndexed(16512, 1)
710	vkCmdDrawIndexed(6633, 1)
716	vkCmdDrawIndexed(14592, 1)
725	vkCmdDrawIndexed(24, 1)
734	vkCmdDrawIndexed(1668, 1)
740	vkCmdDrawIndexed(4086, 1)
746	vkCmdDrawIndexed(16512, 1)
752	vkCmdDrawIndexed(726, 1)
758	vkCmdDrawIndexed(11040, 1)
763	vkCmdDrawIndexed(10224, 1)
769	vkCmdDrawIndexed(30, 1)
778	vkCmdDrawIndexed(24, 1)
787	vkCmdDrawIndexed(4563, 1)
793	vkCmdDrawIndexed(3288, 1)
799	vkCmdDrawIndexed(4368, 1)
805	vkCmdDrawIndexed(14592, 1)
814	vkCmdDrawIndexed(1404, 1)
823	vkCmdDrawIndexed(13824, 1)
829	vkCmdDrawIndexed(6996, 1)
835	vkCmdDrawIndexed(11208, 1)

EID	Event

```glsl
#version 450
#extension GL_KHR_shader_subgroup_clustered : require

layout(push_constant, std430) uniform Constants
{
    vec4 base_color;
    vec4 emissive;
    float roughness;
    float metallic;
    float normal_scale;
} registers;

layout(set = 2, binding = 0) uniform mediump sampler2D uBaseColormap;
layout(set = 2, binding = 1) uniform mediump sampler2D uNormalmap;
layout(set = 2, binding = 2) uniform mediump sampler2D uMetallicRoughnessmap;

layout(location = 1) out mediump vec4 BaseColor;
layout(location = 2) out mediump vec3 Normal;
layout(location = 3) out mediump vec2 PBR;
layout(location = 1) in vec2 vUV;
layout(location = 2) in mediump vec3 vNormal;
layout(location = 3) in mediump vec4 vTangent;
layout(location = 0) in vec3 vPos;

void quad_discard(bool to_discard)
{
    if (subgroupClusteredAnd(int(to_discard), 4u) != 0)
    {
        discard;
    }
}

mediump vec3 two_component_normal(mediump vec2 N)
{
    return vec3(N, sqrt(max(1.0 - dot(N, N), 0.0)));
}

void emit_render_target(mediump vec3 emissive, mediump vec4 base_color, mediump vec3 normal, mediump float metallic, m
{
    BaseColor = vec4(base_color.xyz, ambient);
    Normal = (normal * 0.5) + vec3(0.5);
    PBR = vec2(metallic, roughness);
}
```

Assembly horror

SPIR-V is not something you write by hand ...

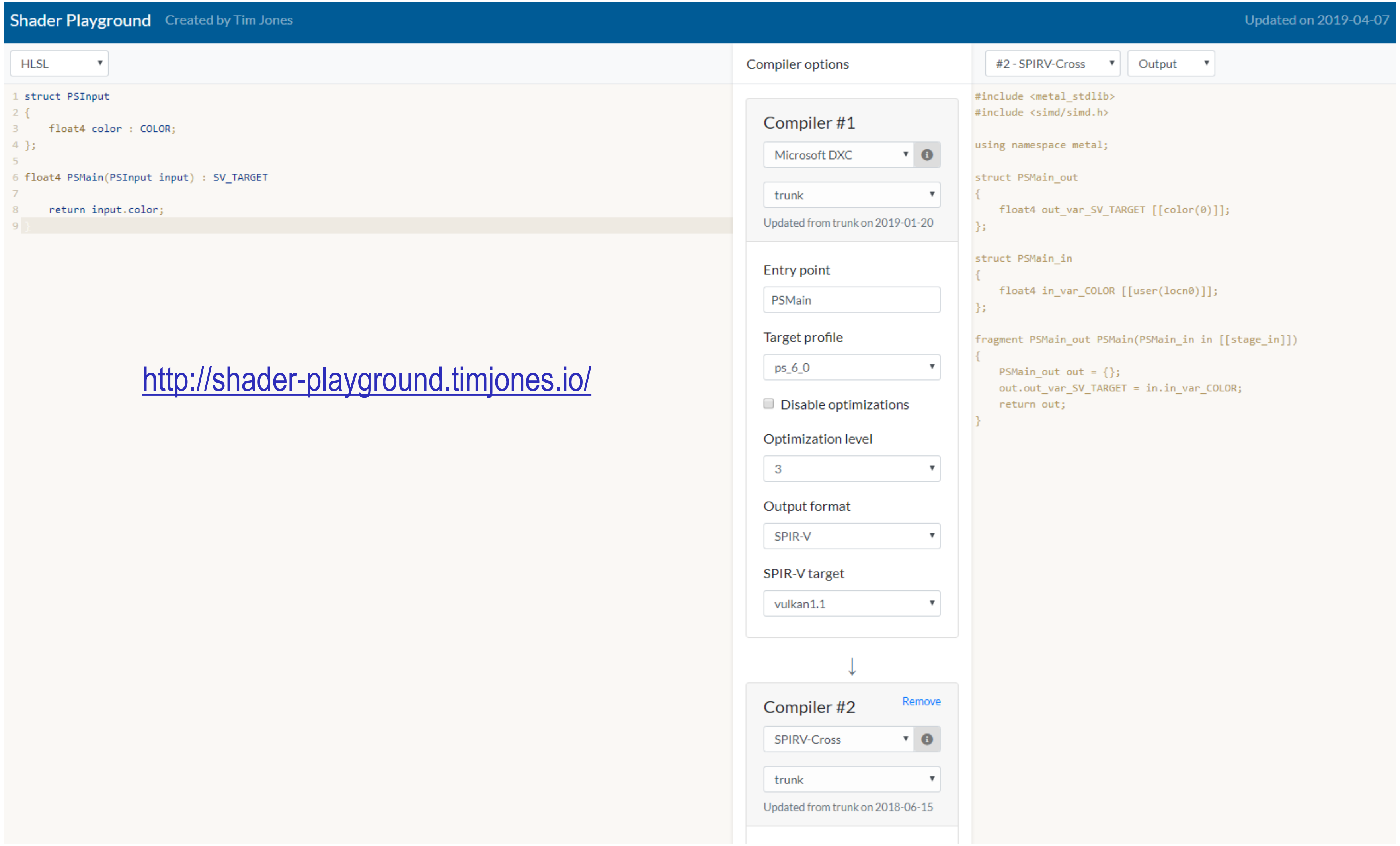


gltf-viewer_2019.05.04_11.32.39_frame565.rdc - RenderDoc v1.4

EID	Name
578	vkCmdDrawIndexed(16512, 1)
584	vkCmdDrawIndexed(27552, 1)
593	vkCmdDrawIndexed(83388, 1)
602	vkCmdDrawIndexed(33120, 1)
608	vkCmdDrawIndexed(48, 1)
614	vkCmdDrawIndexed(16512, 1)
620	vkCmdDrawIndexed(16512, 1)
626	vkCmdDrawIndexed(6633, 1)
632	vkCmdDrawIndexed(840, 1)
638	vkCmdDrawIndexed(15, 1)
644	vkCmdDrawIndexed(162, 1)
653	vkCmdDrawIndexed(1404, 1)
662	vkCmdDrawIndexed(14871, 1)
668	vkCmdDrawIndexed(612, 1)
674	vkCmdDrawIndexed(150, 1)
680	vkCmdDrawIndexed(69624, 1)
686	vkCmdDrawIndexed(4080, 1)
692	vkCmdDrawIndexed(13824, 1)
698	vkCmdDrawIndexed(14871, 1)
704	vkCmdDrawIndexed(16512, 1)
710	vkCmdDrawIndexed(6633, 1)
716	vkCmdDrawIndexed(14592, 1)
725	vkCmdDrawIndexed(24, 1)
734	vkCmdDrawIndexed(1668, 1)
740	vkCmdDrawIndexed(4086, 1)

```
%_ptr_Function_float = OpTypePointer Function %float
         %25 = OpTypeFunction %void %_ptr_Function_v3float %_ptr_Function_v4float %_ptr_Function_v3float %_ptr_Funct
        %int = OpTypeInt 32 1
      %int_0 = OpConstant %int 0
      %int_1 = OpConstant %int 1
       %uint = OpTypeInt 32 0
     %uint_4 = OpConstant %uint 4
     %uint_3 = OpConstant %uint 3
    %float_1 = OpConstant %float 1
    %float_0 = OpConstant %float 0
%_ptr_Output_v4float = OpTypePointer Output %v4float
  %BaseColor = OpVariable %_ptr_Output_v4float Output
%_ptr_Output_v3float = OpTypePointer Output %v3float
     %Normal = OpVariable %_ptr_Output_v3float Output
  %float_0_5 = OpConstant %float 0.5
%_ptr_Output_v2float = OpTypePointer Output %v2float
        %PBR = OpVariable %_ptr_Output_v2float Output
         %87 = OpTypeImage %float 2D 0 0 0 1 Unknown
         %88 = OpTypeSampledImage %87
%_ptr_UniformConstant_88 = OpTypePointer UniformConstant %88
%uBaseColormap = OpVariable %_ptr_UniformConstant_88 UniformConstant
%_ptr_Input_v2float = OpTypePointer Input %v2float
        %vUV = OpVariable %_ptr_Input_v2float Input
  %Constants = OpTypeStruct %v4float %v4float %float %float %float
%_ptr_PushConstant_Constants = OpTypePointer PushConstant %Constants
```

Explore SPIRV-Cross output online



```hlsl
struct PSInput
{
    float4 color : COLOR;
};

float4 PSMain(PSInput input) : SV_TARGET

    return input.color;
}
```

Compiler options

Compiler #1
Microsoft DXC
trunk
Updated from trunk on 2019-01-20

Entry point: PSMain
Target profile: ps_6_0
☐ Disable optimizations
Optimization level: 3
Output format: SPIR-V
SPIR-V target: vulkan1.1

↓

Compiler #2 Remove
SPIRV-Cross
trunk
Updated from trunk on 2018-06-15

```
#include <metal_stdlib>
#include <simd/simd.h>

using namespace metal;

struct PSMain_out
{
    float4 out_var_SV_TARGET [[color(0)]];
};

struct PSMain_in
{
    float4 in_var_COLOR [[user(locn0)]];
};

fragment PSMain_out PSMain(PSMain_in in [[stage_in]])
{
    PSMain_out out = {};
    out.out_var_SV_TARGET = in.in_var_COLOR;
    return out;
}
```

http://shader-playground.timjones.io/

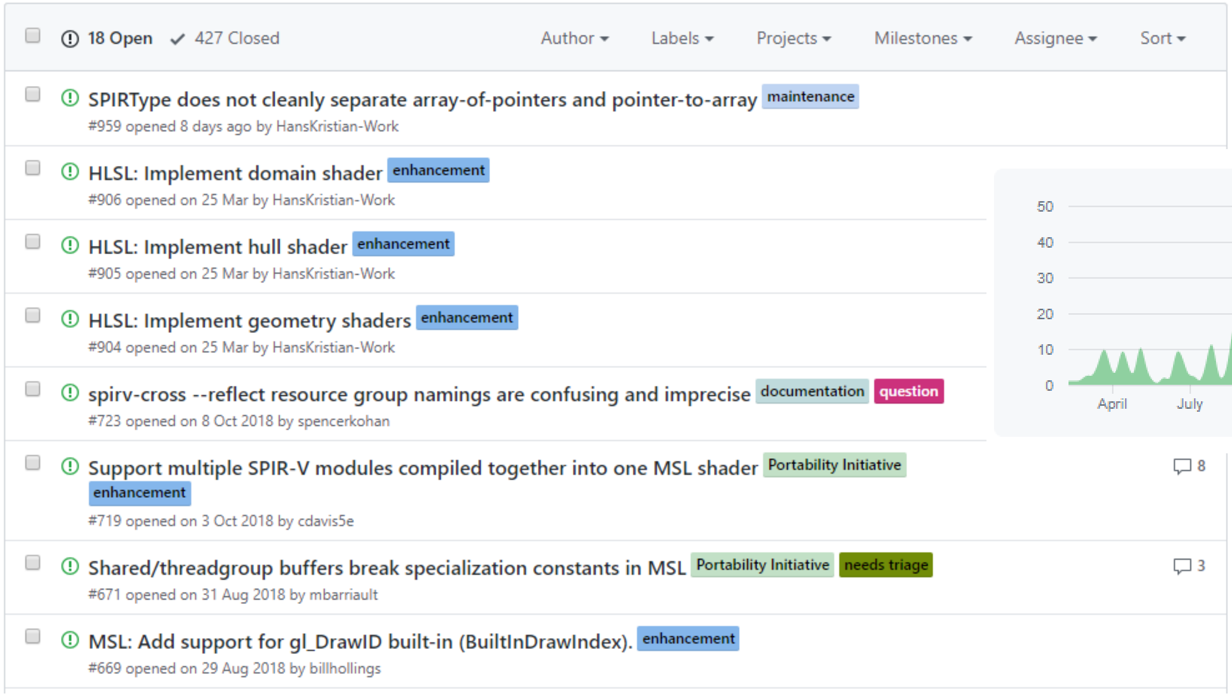
The goal of SPIRV-Cross

- **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

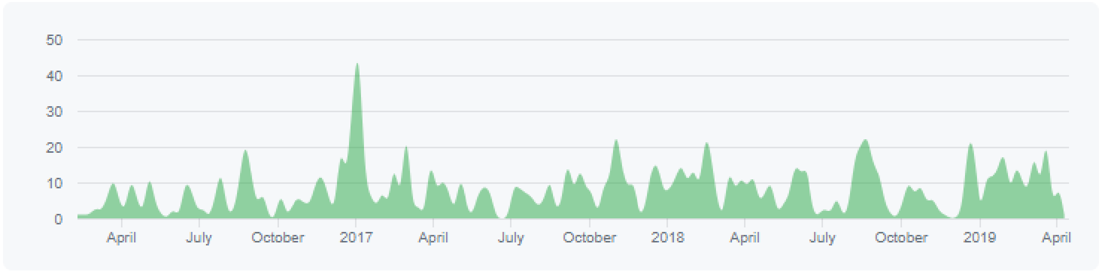
Don’t hesitate to file issues or feature requests

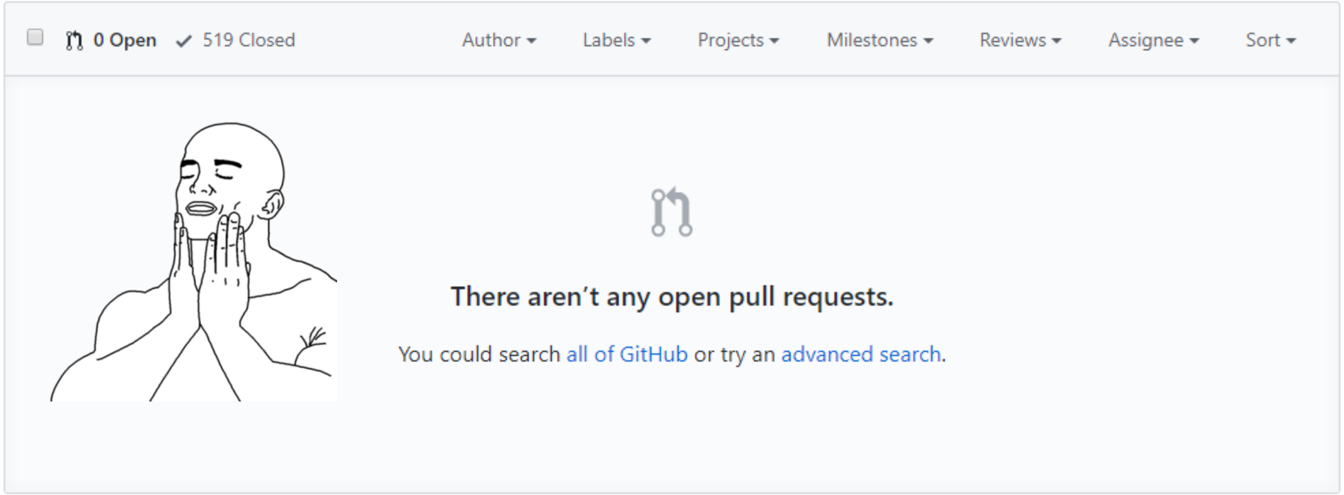


18 Open ✓ 427 Closed — Author ▾ Labels ▾ Projects ▾ Milestones ▾ Assignee ▾ Sort ▾

- SPIRType does not cleanly separate array-of-pointers and pointer-to-array `maintenance`
  #959 opened 8 days ago by HansKristian-Work
- HLSL: Implement domain shader `enhancement`
  #906 opened on 25 Mar by HansKristian-Work
- HLSL: Implement hull shader `enhancement`
  #905 opened on 25 Mar by HansKristian-Work
- HLSL: Implement geometry shaders `enhancement`
  #904 opened on 25 Mar by HansKristian-Work
- spirv-cross --reflect resource group namings are confusing and imprecise `documentation` `question`
  #723 opened on 8 Oct 2018 by spencerkohan
- Support multiple SPIR-V modules compiled together into one MSL shader `Portability Initiative` `enhancement` 8
  #719 opened on 3 Oct 2018 by cdavis5e
- Shared/threadgroup buffers break specialization constants in MSL `Portability Initiative` `needs triage` 3
  #671 opened on 31 Aug 2018 by mbarriault
- MSL: Add support for gl_DrawID built-in (BuiltInDrawIndex). `enhancement`
  #669 opened on 29 Aug 2018 by billhollings

Commits over time





0 Open ✓ 519 Closed — Author ▾ Labels ▾ Projects ▾ Milestones ▾ Reviews ▾ Assignee ▾ Sort ▾

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**

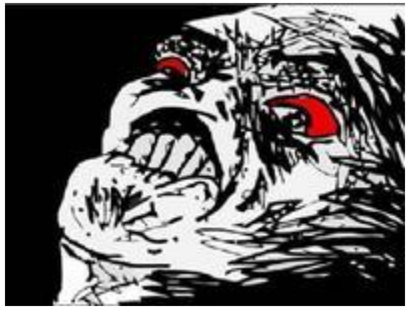
```
#version 450

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

```
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)

```
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 = 0;
    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**
  - 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