HLSL in Vulkan
There and Back Again
Shrinking and Legalizing Vulkan Shaders with spirv-opt

Greg Fischer, LunarG
January 2017
Goals of Talk

- What is spirv-opt?
- What is the status of SPIR-V size?
- What is HLSL Legalization?
- How to engage spirv-opt?
SPIR-V

- Binary shader intermediate representation (IR) for Vulkan (and other APIs)
- SPIR-V is to Vulkan as DX Byte Code is to DirectX
- Primarily generated from high-level shader languages GLSL and HLSL
- Disassembler, Assembler, Validator also available
**spirv-opt**

- Open Source ([github.com/KhronosGroup/SPIRV-Tools](https://github.com/KhronosGroup/SPIRV-Tools))
- Collaboration between Google and LunarG with support from Valve. Additional contributions from Mesa and Roblox.
- SPIR-V -> “Optimized” SPIR-V
- Goals include
  - Reduced SPIR-V size
  - “Legalized” SPIR-V from HLSL
- Utilizes classic, platform-independent optimization techniques
- First announced at SIGGRAPH, August 2017
spirv-opt: Reducing SPIR-V Size

- Optimizations include
  - Function Call Inlining
  - Constant Folding & Propagation
  - Arithmetic Simplification
  - Local Store/Load Elimination
  - Dead Code Elimination
  - Common Subexpression Elimination
  - Debug Information Stripping

- Optimized SPIR-V now within 10% of DX Byte Code size
**HLSL and fxc Equivalence**

- HLSL shaders *must* be “legalized”
- HLSL contains constructs that are not directly supported by graphics hardware
- Optimization techniques can be used to put shader code in a form directly supported by graphics hardware
- fxc is Microsoft’s optimizing shader compiler
- fxc: HLSL -> Highly optimized and "legalized" DX Byte Code
- Current graphics hardware and drivers assume HLSL shaders will have certain optimizations done
- spirv-opt therefore needs to be "equivalent" to fxc in optimization capability when porting DX/HLSL to Vulkan
Some HLSL features not directly supported in Vulkan (or DX)
- Opaque structure members (eg. textures, samplers)
- Local Structured Buffers

glslangValidator issues warning for problematic constructs:
“WARNING: AST will form illegal SPIR-V; need to transform to
legalize”

These constructs are "optimized" away by spirv-opt through
function call inlining, dead control flow elimination, value
propagation
Opaque Struct Example

```c
struct os {
    SamplerState o_s2D;
    Texture2D o_tex;
};

Texture2D tex;
SamplerState s2D;

float4 osCall(os s, float2 f2)
{
    return s.o_tex.Sample(s.o_s2D, f2);
}

float4 main() : SV_TARGET0
{
    os s;
    s.o_tex = tex;
    s.o_s2D = s2D;
    return osCall(s, float2(0.2, 0.3));
}
```
Opaque Struct Example: Optimized

Texture2D tex;
SamplerState s2D;

float4 main() : SV_TARGET0
{
    return tex.Sample(s2D, float2(0.2, 0.3));
}
Local Structured Buffer Example

```cpp
struct S {
    float4 f;
};

RWStructuredBuffer<S> gRWSBuffer;

float4 main() : A {
    RWStructuredBuffer<S> t;
    t = gRWSBuffer;
    return t[0].f;
}
```
Local Structured Buffer Example: Optimized

```c
struct S {
    float4 f;
};

RWStructuredBuffer<S> gRWSBuffer;

float4 main() : A {
    return gRWSBuffer[0].f;
}
```
HLSL may contain dead texture, sampler, buffer references

Such dead refs cause Vulkan Validation Layer complaints:
“VALIDATION ERROR 0x3d Descriptor set 0x41bde encountered the following validation error at vkCmdDrawIndexed() time: Descriptor in binding #33 at global descriptor index 10 is being used in draw but has not been updated.”

These are optimized away by spirv-opt dead code/branch elimination
Dead Sample Example

Texture2D tex0;
SamplerState s0;

Texture2D tex1;
SamplerState s1;

float4 main() : SV_TARGET0
{
    if (true)
        return tex0.Sample(s0, float2(0.2, 0.3));
    else
        return tex1.Sample(s1, float2(0.2, 0.3));
}
Dead Sample Example: Optimized

Texture2D tex0;
SamplerState s0;

float4 main() : SV_TARGET0
{
    return tex0.Sample(s0, float2(0.2,0.3));
}

spirv-opt Usage - Direct

- Command line
  - --legalize-hlsl
  - -Os: Optimize for size
  - --<pass> --<pass> ... (see --help)
- API interface
  - Optimizer::RegisterLegalizationPasses()
  - Optimizer::RegisterSizePasses()
**spirv-opt Usage - Indirect through Frontends**

- `glslangValidator`
  - Khronos GLSL/HLSL FE
  - For legacy reasons, glslang does not require SPIRV-Tools to build
  - Legalization by default if built with SPIRV-Tools; can be disabled (see README)
  - `-Os`: Optimize for size if built with SPIRV-Tools
  - LunarG Vulkan SDK edition built with SPIRV-Tools
  - github.com/KhronosGroup/glslang
spirv-opt Usage - Indirect through Frontends

- **dxc**
  - Microsoft Open Source HLSL FE / SPIR-V generator
  - Default Legalization, Optimization
  - Can be disabled (see README)
  - [github.com/Microsoft/DirectXShaderCompiler/wiki/SPIR-V-CodeGen](https://github.com/Microsoft/DirectXShaderCompiler/wiki/SPIR-V-CodeGen)

- **glslc**
  - Wrapper for glslangValidator and SPIRV-Tools
  - Legalization always on for HLSL
  - `-Os` for size optimization
  - [github.com/google/shaderc/tree/master/glslc](https://github.com/google/shaderc/tree/master/glslc)

GLSL/HLSL

[glslangValidator](https://github.com/Microsoft/DirectXShaderCompiler/wiki/SPIR-V-CodeGen) / **dxc** / **glslc**

(Optimized/Legalized) SPIR-V

Vulkan
Acknowledgements

- Google (David Neto, Steven Perron, Alan Baker, Diego Novillo, John Kessenich, Lei Zhang)
- Valve (Pierre-Loup Griffais, Dan Ginsburg)
- Mesa (Pierre Moreau)
- Roblox (Arseny Kapoulkine)
From HLSL to Vulkan®

DR. MATTHÄUS G. CHAJDAS, AMD
Most games have large HLSL shader libraries
Using those with Vulkan requires some work

STATE OF THE UNION
WHERE ARE WE?
Vulkan has a different binding model from Direct3D 11/12
- Direct3D 11 is all about slots and named bindings
- Direct3D 12 has root signatures, typed descriptor heaps etc.

Vulkan has descriptor sets

```
Texture2D t : register (t0) ⇔ layout (set = X, binding = Y) texture2D t;
```
OTHER DIFFERENCES
GLSL VS HLSL

- **No 1:1 mapping** of resource types
  - UAV ↔ Image?
  - Some things have no 100% equivalent, like read-only structured buffers
- **Samplers are not objects** you can pass around
- Bindless is very different
- Descriptor remapping is generally the main problem
  - Partition descriptor ranges?
  - I/O remapper?
  - Engine needs to be somehow aware of both
- Some functionality is **missing** in HLSL
  - Push constants
  - Input attachments
  - Specialization constants
The big difference is the binding model

Specific syntax to help you out – attributes “[[vk::binding]]” etc.

Probably want to wrap those in macros so FXC doesn’t see this

This is going to be the majority of your porting effort!

```cpp
struct S {
    float2 f;
};

[[vk::binding(1)]] StructuredBuffer<S> buffer1;
[[vk::binding(3, 2)]] StructuredBuffer<S> buffer3;

[[vk::input_attachment_index(4)]]
Texture2D<float4> attach;

[[vk::constant_id(13)]] const int ci = 11;
[[vk::push_constant]] cbuffer pcBuf { int a; };

[[vk::location(7)]] float4
main([[vk::location(8)]] float4 input: A) : B
{
    return input + attach.Load(float2(0.5)); // * a;
}
```
Your compile pipeline should **end** in SPIR-V Opt

- Required for **legal** SPIR-V from HLSL – mostly for passing around opaque objects (samplers, etc.)
- Various optimizations are available
  - Anything **reducing output size** is generally safe – and also helps the backend
  - Prefer `[unroll]` to forced-unroll
  - Remap identifiers etc. is beneficial
  - Turning on **all options** is not a good idea 😊

You should also use it for GLSL

- Significant savings on shipping titles on the code size end
  - Still need to apply compression
  - Consider **domain-specific compression** like SMOL-V
Originally, there used to be a significant “Vulkan tax”
- Compilers not used to seeing SPIR-V
- GLSLang compile output sometimes rather naïve ...

Be on the lookout
- Temporary/local arrays or arrays in general
- Function calls – specifically passing large objects around
- Annotations not getting translated correctly

Overall, we’re really close these days
- And if we aren’t, use the tools
- SPIR-V opt, dis is your friend
- IHV tools to inspect generated ISA
SUMMARY
WHERE ARE WE?

- Games **are shipping** on Vulkan with tons of HLSL source
- **Single HLSL source** for both Vulkan and D3D is possible
- Going forward, you can use DXC supporting both APIs – more on this in a minute!
DISCLAIMER & ATTRIBUTION

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

ATTRIBUTION

© 2018 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Other names are for informational purposes only and may be trademarks of their respective owners.
Overview

• Brief History
  - There and back again

• Development
  - Compilation
  - Legalization
  - Optimization

• What’s Here
  - SM6.0 Wave Ops
  - Semantics
  - Reflection

• What’s Coming
  - Descriptor Indexing
  - Extensions

• Credits
  - Appreciating the blood, sweat, and tears
Brief History

• There...
  - David Neto (Google) initiated the Spiregg project
    - Google met with Microsoft DXC team in early 2017
  - HLSL is a language without a spec
    - “I did not know that was legal...” - Baldur Karlsson (RenderDoc)
    - Microsoft plans to evolve the HLSL rapidly once DXC is stable
    - Keeping up with parser could prove difficult...so leverage it!

• ...and back again
  - Spiregg at Google contributes and maintains a SPIR-V backend for DXC
    - Open source under LLVM license and hosted on Github
  - Spiregg leaned on glslang’s progress to get started
    - Both projects collaborate to maintain parity
  - Spiregg works closely with SPIRV-Tools/spirv-opt team
    - DXC/SPIRV leans on spirv-opt for legalization
  - Outreach to community and IHVs has provided valuable feedback
Development

• Compilation
  - Parsing HLSL to generalized SPIR-V
  - *Front end* process that happens in DXC

• Legalization
  - Transforming generalized SPIR-V to Vulkan dialect
  - *Back end* process that happens in SPIRV-Tools
    - spirv-opt transform passes
  - See Greg’s talk for more details

• Optimization
  - Transforms Vulkan SPIR-V to be more performant
  - Controversial transform
    - Loop unrolling
      - Must be done for legalization, will follow up about performance
    - Will follow up at SIGGRAPH
Development

- Spiregg has been very busy!
  - Lei Zhang (antiagainst)
  - Ehsan Nasiri (ehsannas)
Legalization

- We are aware that some generated SPIR-V is problematic for consumption
  - Complex HLSL opaque object resolution
  - Structs within structs containing opaque objects
  - First time legalization has been attempted for HLSL in Vulkan
    - We’re learning very valuable lessons
- Working with IHVs to address this as quickly as possible
  - Bug fixes to DXC
  - Updated drivers
What’s Here

• Full SM 5.1 Support
  - Bugs (and features not in Vulkan) not withstanding
    - Please bugs report on Github!
    - Let us know if we’re missing anything!

• SM 6.0 Wave Ops
  - Landed with Vulkan 1.1

• Other Highlights
  - Global variable collected under $Globals cbuffer
    - Working on support to assing $Globals to a specific register
  - SPV_KHR_shader_draw_parameters implemented to explore extension workflow

• Semantics and Counter Buffers!
  - Two SPIR-V Extensions
  - One upcoming Vulkan Extension to support SPIR-V Extensions

• Reflection
  - Reflecting SPIR-V data at runtime
What’s Here: Semantics and Counter Buffers

• Two SPIR-V extensions
  - SPV_GOOGLE_decorate_string
    - OpDecorateStringGOOGLE - decorates variable with string
    - OpMemberDecorateStringGOOGLE - decorate struct member with string
  - SPV_GOOGLE_hlsl_functionality1
    - HlslCounterBufferGOOGLE - link a counter buffer to a UAV resource that has an
      associated counter
    - HlslSemanticGOOGLE - decorate an input or output variable id with a string
      representing semantic as defined in the HLSL source

• One Vulkan extension
  - WIP

• Opt-in Feature for DXC and glslang
  - -fspv-reflect for DXC
  - -hlsl_functionality1 for glslang
What’s Here: Semantics and Counter Buffers

```c
struct PSInput {
    float4 Position : SV_POSITION;
    float3 Normal : NORMAL;
    float3 Color : COLOR;
    float4 Alpha : OPACITY;
    float4 Scaling : SCALE;
    float2 TexCoord0 : TEXCOORD0;
    float2 TexCoord1 : TEXCOORD1;
    float2 TexCoord2 : TEXCOORD2;
};

struct PSOutput {
    float4 oColor0 : SV_TARGET0;
    float4 oColor1 : SV_TARGET1;
    float4 oColor2 : SV_TARGET2;
    float4 oColor3 : SV_TARGET3;
    float4 oColor4 : SV_TARGET4;
    float4 oColor5 : SV_TARGET5;
    float4 oColor6 : SV_TARGET6;
};
```

Copyright ©2018 The Khronos™ Group Inc. - Page 42
What’s Here: Semantics and Counter Buffers

- There’s just one thing…
  - New SPIR-V opcodes and decorations not yet consumable by drivers
  - Must be stripped before handing off the SPIR-V to the driver

- Fear not...batteries are included!
  - SPIRV-Reflect repo will have copy/paste snipped to strip reflection data
  - What’s SPIRV-Reflect?
What’s Here: Reflection

• SPIRV-Reflect
  - Small library C/C++
    - 2 files: 1 header, 1 source
  - Cort Stratton + Hai Nguyen
  - Send bugs and requests to Cort

• Reflected Data
  - Vertex attribute locations
    - Basic type info
  - Descriptor bindings and sets
    - Binding #, set#, descriptor type
  - Uniform, storage, push constants blocks
    - Relative offsets, absolute offsets, raw size, padded size, type info
  - HLSL resource types
    - CBV, SRV, UAV, Samplers
  - Semantics and Counter Buffers

I <3 BUGS!
What’s Here: Reflection
What’s Here: Reflection

```c
MINGW44/C/code/hsl/SPIRV-Reflect/texts/hsl

```
What’s Here: Reflection

- [https://github.com/chaoticbob/SPIRV-Reflect](https://github.com/chaoticbob/SPIRV-Reflect)
- Goes live this Friday (March 23, 2018)
- Please file issues on Github
- Work in progress to add reflection to SPIRV-Tools
  - Will deprecate SPIRV-Reflect
What’s Coming

• Descriptor Indexing!
  - NonUniformResourceIndex FTW!
  - “Coming soon”

• Extensions Support
  - Will likely be command line option
    - -fspv-extension=<ext-a> -fspv-extension=<ext-b>

• Moar SM 6.x
  - 64-bit integer
  - 16-bit scalars
  - Barycentrics

• Things being considered…but no firm conclusion
  - Inline cbuffer initialization
  - Root descriptors in HLSL source
  - Specifying extensions in the source like GLSL
Credits

• **Individuals** (apologies if I missed anyone)
  - Team #spiregg (DXC/SPIRV)
    - Lei Zhang (Google), Ehsan Nasiri (Google)
  - Guidance
    - David Neto (Google), John Kessenich (Google)
  - spirv-opt
    - Greg Fischer (LunarG), Diego Novillio (Google), Steven Perron (Google), Alan Baker (Google)
  - IHV Friends
    - Dr. Matthäus G. Chajdas (AMD), Nuno Subtil (NVIDIA), Piers Daniell (NVIDIA), Jason Ekstrand (Intel), Slawomir Grajewski (Intel)
  - Khronos Members
    - Neil Henning (Codeplay), Tobias Hector (Imagination), Dan Ginsburg (Valve)
  - Community Members
    - Graham Wihlidal (EA), Andrew Lauritzen (EA), Dan Baker (Oxide)

• **Companies**
  - AMD, Intel, LunarG, Microsoft, Nvidia, Valve
Thank You!