
KHRONOS® GROUP GDC

GDC 2019
#KhronosDevDay

# DEVELOPER DAY

*Gaijin's HLSL to Spir-V Compiler Stack Evolution*

*Tiemo Jung, Gaijin Entertainment*

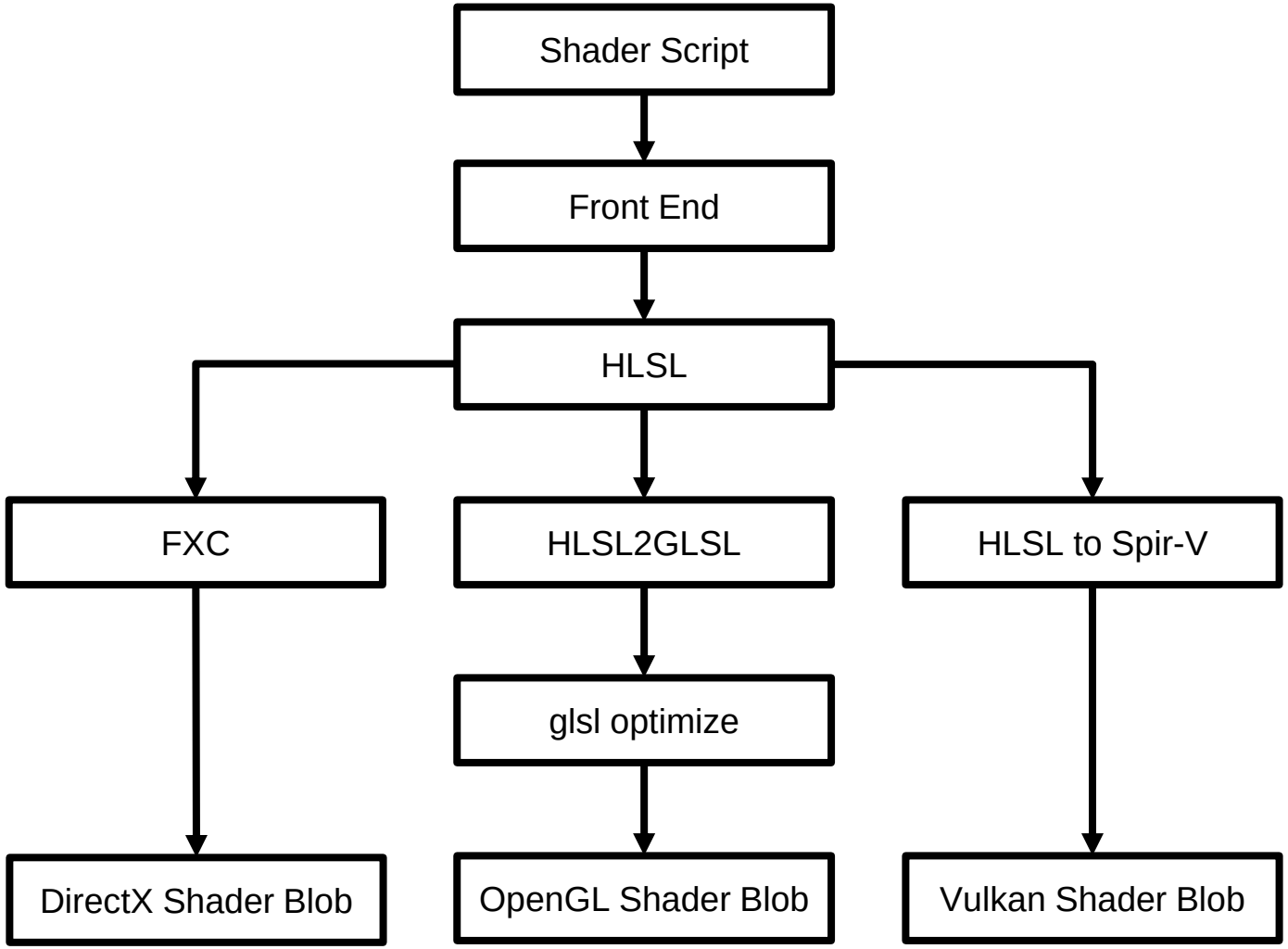
KHRONOS® GROUP

GDC

GDC 2019

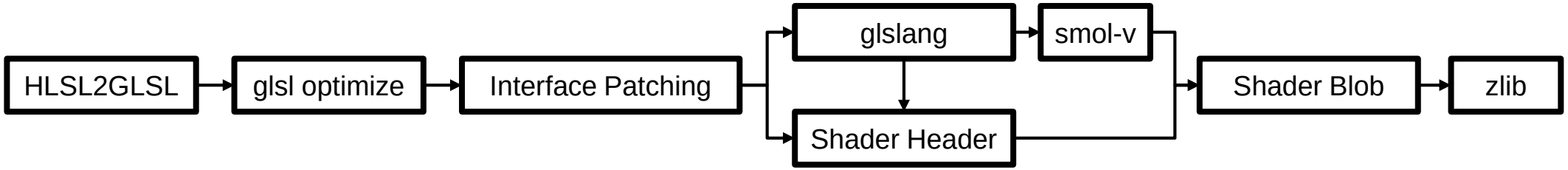
#KhronosDevDay

# Basic Compiler Pipeline



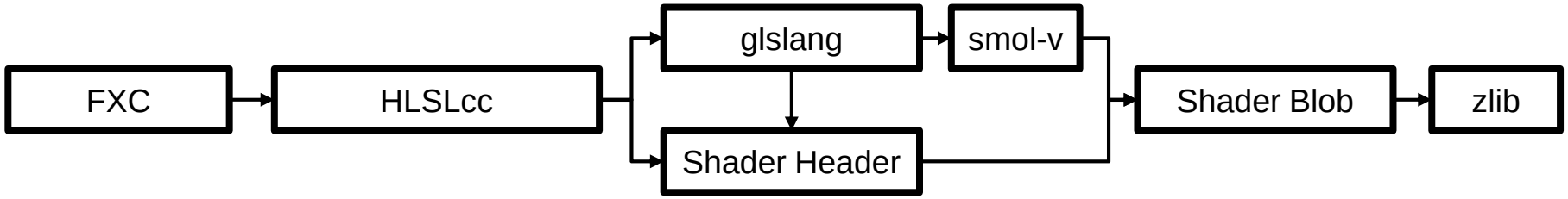
# Compiler: HLSL2GLSL

- **HLSL2GLSL and glsl optimize**
  - Taken from OpenGL shader compiler
  - Inherited required hacks and workarounds in shaders for HLSL2GLSL to work properly
- **Interface Patching**
  - Pattern matching to find interface variables in GLSL source
  - Generated global uniform block out of loose uniforms
  - Added necessary layout attributes for Vulkan compatibility
- **Shader Header**
  - Small header with SPIR-V shader hash and resource slot usage mask
  - Evolved over time to meet the needs of the renderer



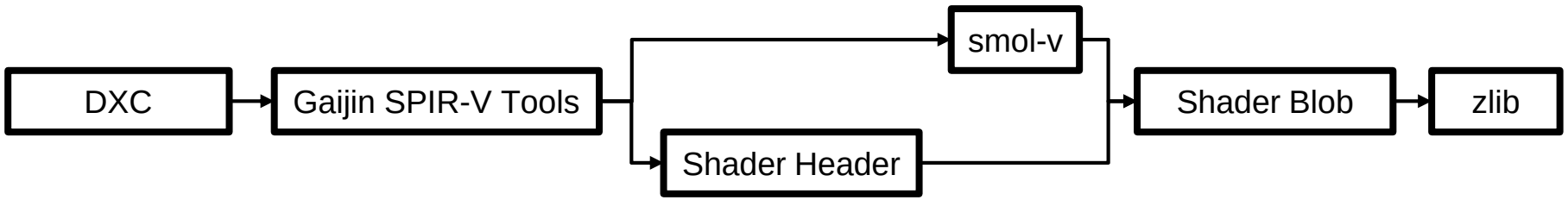
# Compiler: HLSLcc

- **FXC**
  - Still most commonly used to compile HLSL
  - No longer any hacks in shaders because of HLSL2GLS
  - Decent optimized DXBC
  - Stuck with shader model 5
- **HLSLcc**
  - Started with James Jones version
  - Moved over to Unity version
  - Both versions needed fixes to work properly with our shaders
  - Extended both to fit our needs
  - In DXBC instructions dictate how registers are interpreted, this lead to guessing and awkward shader code with lots of bit casts



# Compiler: DXC

- **DXC**
  - Replacement for FXC targeting DX12
  - Open source
  - Includes SPIR-V output (spiregg - maintained by google)
  - Modern shader features like shader model 6 and ray tracing
- **Gaijin SPIR-V Tools**
  - More later
- **zlib**
  - Will be replaced with zstd or lz4



# Atomic Counter: Embedded

- **Embedded at the top of the storage buffer**
- **Easy to implement in HLSLcc compiler pipeline**
- **Much more complex in SPIR-V**
  - Clone and modify structure type
  - Patch load, store, access chain and atomic operations with new target or new indices
  - Remove counter buffer and any left over types
- **Easy to implement in renderer**
  - Allocate more memory to store counter
  - Have two views, with and without counter
- **Wastes memory because of padding requirements (see minStorageBufferOffsetAlignment)**
- **Limited to one counter per buffer**

```
buffer StorageBufferWithCounter {
  int counter; // embedded by shader compiler
  StorageBufferElement data[];
};
```

# Atomic Counter: Global Indexed

- Arbitrary amount of counters per buffer
- Counters are tightly packed in one global buffer
- Reset to 0 is just handing out an unused counter index and reset old counters when it is convenient
- Requires 2 resource slots if used
- Not implemented yet
- Performance characteristics yet unknown

```
uniform CounterIndexStore {
  int indexForCounter;
};
buffer CounterStore {
  int counterStore[];
};
buffer DataStore {
  StorageBufferElement data[];
};
void main() {
  int index = atomicAdd(counterStore[indexForCounter], 1);
  data[index] = cumputeData();
}
```

# Pipeline Layout: Fixed

- No need to manage layouts
- Easy to implement in renderer and compiler
- Waste of descriptor memory
- Does not scale with growing and changing requirements
- Easy to run into HW limits even if never actually used
- Not suitable for compute, as layouts vary a lot

| Set | Binding | Usage |
|---|---|---|
| 0 | 0 | Uniform Buffer Vertex Shader |
| | 1 | Uniform Buffer Fragment Shader |
| 1 | 0 - 3 | Sampled Textures Vertex Shader |
| | 4 - 19 | Sampled Textures Fragment Shader |

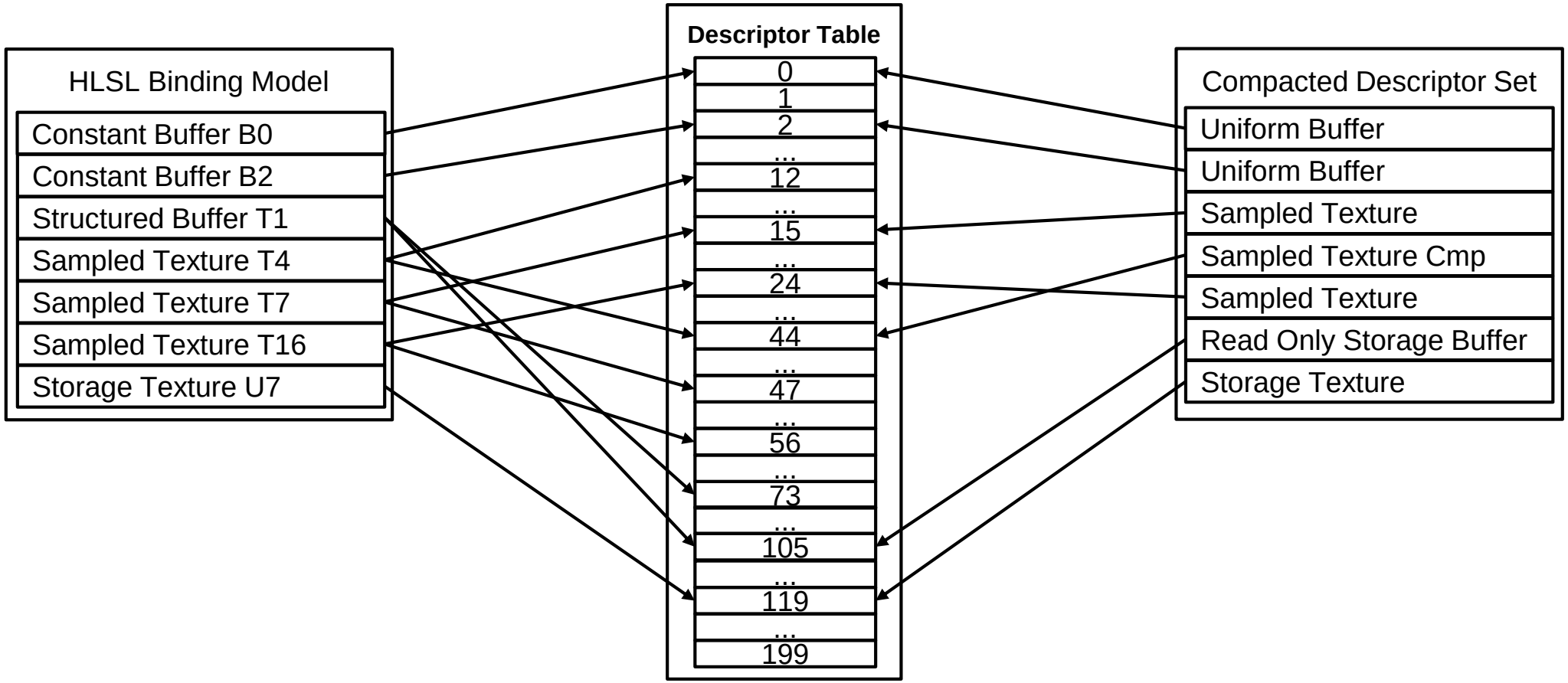
| Set | Binding | Usage |
|---|---|---|
| 0 | 0 | Uniform Buffer Vertex Shader |
| | 1 | Uniform Buffer Fragment Shader |
| 1 | 0 - 15 | Sampled Textures Fragment Shader |
| 2 | 0 - 3 | Sampled Textures Vertex Shader |
| 3 | 0 - 7 | Storage Images Fragment Shader |

# Pipeline Layout: Compacted (per Stage)

- Allows mix and match of resources as needed
- HW limits not a problem except on very limited targets
- Sharing of SPIR-V when shader only differs in DX resource binding slot
- Each unique descriptor set has its own descriptor heaps to avoid fragmentation
- Used for all shader stages uniformly
- Easy to extent
- Needs management of descriptor layouts

| Table Index | DX Mapping | Usage |
|---|---|---|
| 0 - 7 | b0 - b7 | Uniform Buffers |
| 8 - 39 | t0 - t31 | Sampled Images |
| 40 - 71 | t0 - t31 | Sampled Images With Comparison |
| 72 - 103 | t0 - t31 | Buffer Images |
| 104 - 135 | t0 - t31 | Read Only Storage Buffers |
| 136 - 143 | u0 - u7 | Storage Images |
| 144 - 151 | u0 - u7 | Storage Buffer Images |
| 152 - 159 | u0 - u7 | Storage Buffers |
| 160 - 167 | u0 - u7 | Storage Buffers With Counter |
| 168 - 199 | t0 - t31 | Raytrace Acceleration Structure |

# Pipeline Layout: Compacted (per Stage)



# Gaijin SPIR-V Tools: Rationale

- **Why own SPIR-V Tools?**
  - Most other SPIR-V tools and libraries are designed to be used for their special purpose and are closed off for other uses
  - We want a thin decoder for fast, easy and safe peeking at SPIR-V blobs
  - We also want a full suit to load, modify and store SPIR-V in our compiler chain
- **Bulk is auto generated with a Python script from the grammar JSON spec files, inspired by the Khronos SPIR-V Tools.**
  - Basic information to print and decode SPIR-V binary format
  - Pattern matching to generate AST node and properties for each SPIR-V instruction
    - Required some handcrafted overrides for a few instructions

# Gaijin SPIR-V Tools: Components

- **Decoder**
  - Basic SPIR-V binary format decoder
  - Converts word stream in properly typed commands to the user
  - Does not allocates any memory
- **Module Builder**
  - Handwritten
  - Interfaces with auto generated AST nodes and properties
  - In memory representation of a SPIR-V module
  - Provides basic tools to build, inspect and modify the SPIR-V module
- **Loader**
  - Bridge from Decoder to Module Builder
  - Auto generated

# Gaijin SPIR-V Tools: Components

- **Writer / Encoder**
  - Encodes Module Builder state into SPIR-V binary representation
  - Auto generated
- **Passes**
  - Map HLSL Semantics to location indices
  - Merge separate Samplers and Images to combined Sampled Images
  - Transform Buffer Counters
  - Descriptor packing and Shader Header compilation
  - Forking shaders with different extension implementations?
  - Target specific optimizations?