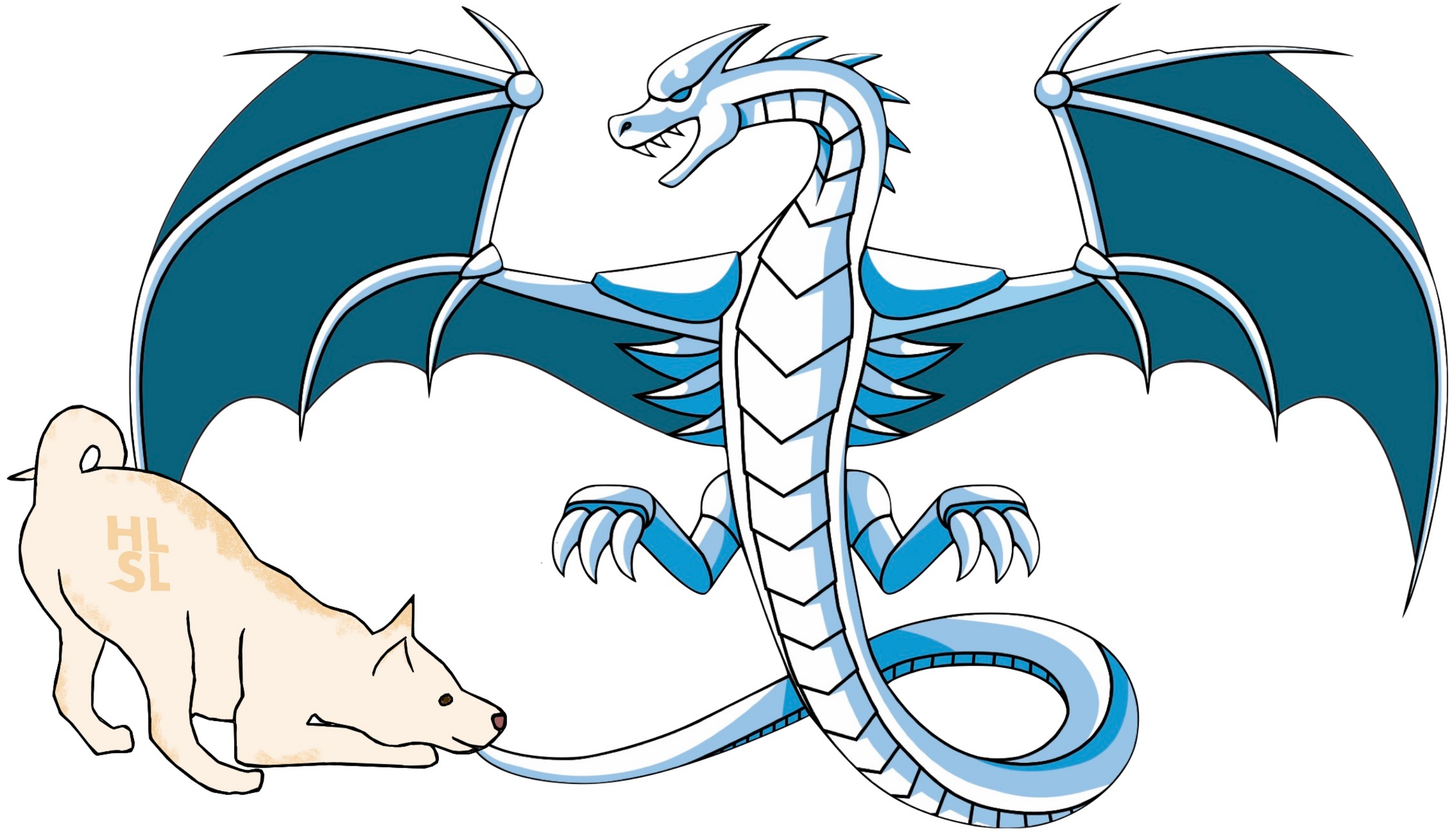
A Better HLSL Compiler



Previously:
LLVM Dev 2024

- TL;DR: Document your stuff
- https://www.youtube.com/watch?v=sVq5khCXkbw

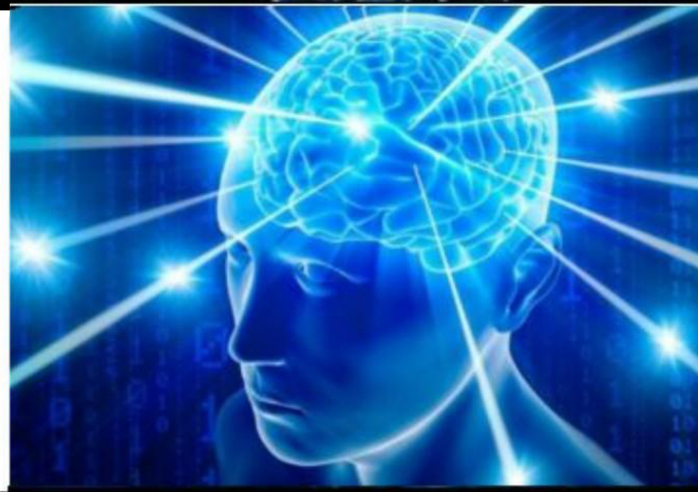
“Test your stuff”

Chris Bieneman

Write code

Compile code

Test code



The End

How we’re testing HLSL

What problem are we trying to solve?

How does the offload-test-suite help?

Where has our team integrated it to our process?

What value are we already seeing from it?

HLSL in Clang

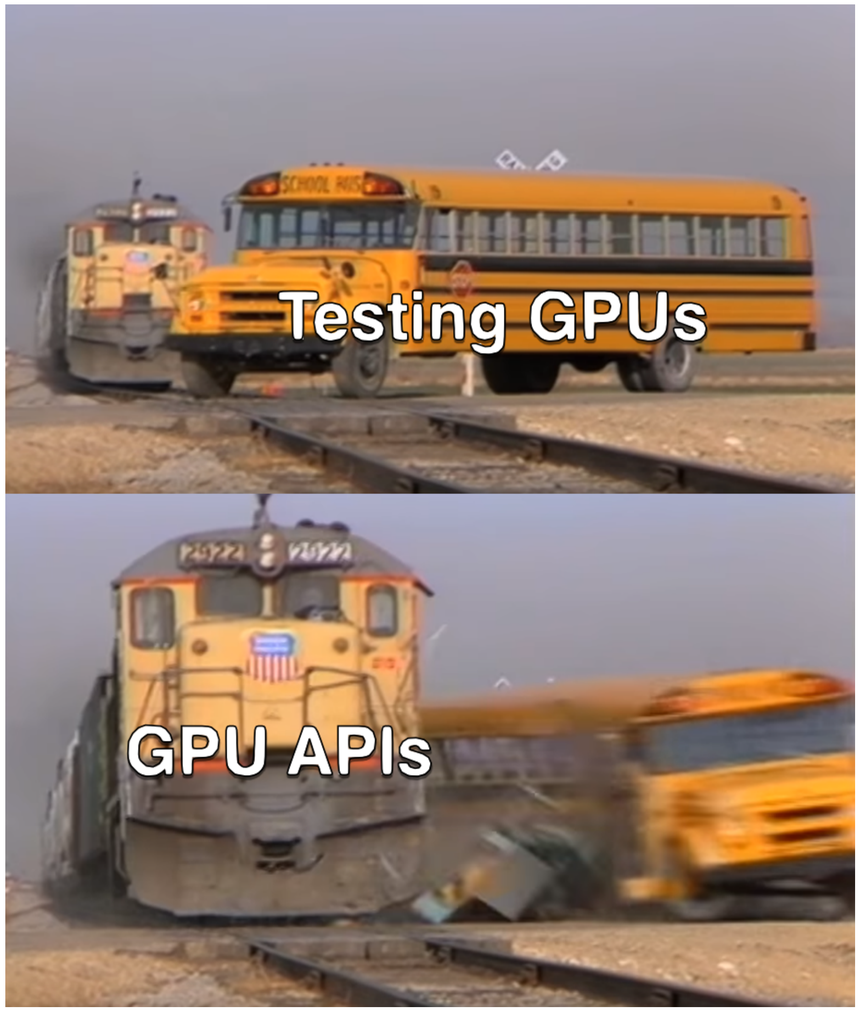
- Adding HLSL to Clang is the _first_ step
- Migrate existing users from DXC to Clang
  - HLSL users have decades of legacy code and tooling
  - Clang needs a high degree of source compatibility
  - Comparable or better usability

Testing as Substitute for Specification

- “correct” behavior is implementation defined
- We need a way to test and compare DXC and Clang
- Needs to support a variety of hardware and software environments

What makes this hard?

- Defining common abstractions across ecosystems is hard
- Different hardware devices can be wildly different in how they execute
- Runtime APIs have very different abstraction decisions
- No single runtime works flawlessly everywhere (even the cross-platform ones)



Other approaches

- Existing infrastructure like llvm-test-suite is designed for C/C++ code
  - Builds driven with CMake
  - Requires multiple configurations to change testing parameters
- CUDA/HIP/OpenMP have device unit testing too
- Google’s Amber project
  - Very shader focused
  - Not part of LLVM

Goals

- Simple workflow for writing single file tests
  - Also should be possible to write larger more complex tests!
- No complex custom grammar
- Test multiple compilers, languages and runtimes from a single configuration
  - HLSL goal: DXC and Clang generate functionally equivalent programs
- Utilize LLVM infrastructure and methodologies

Simple Example

- Tests use split-file
- Shader source, data to operate on and validation

```
#--- source.hlsl
RWBuffer<int> In : register(u0);
RWBuffer<int> Out : register(u1);

[numthreads(8,1,1)]
void CSMain(uint3 TID : SV_GroupThreadID) {
  Out[TID.x] = In[TID.x];
}

//--- pipeline.yaml

---
Shaders:
  - Stage: Compute
    Entry: CSMain
    DispatchSize: [1, 1, 1]
Buffers:
  - Name: In
    Format: Int32
    Data: [ 1, 2, 3, 4, 5, 6, 7, 8]
  - Name: Out
    Format: Int32
    Data: [ 9, 10, 11, 12, 13, 14, 15, 16]
DescriptorSets:
  - Resources:
    - Name: In
      Kind: RWBuffer
      DirectXBinding:
        Register: 0
        Space: 0
      VulkanBinding:
        Binding: 0
    - Name: Out
      Kind: RWBuffer
      DirectXBinding:
        Register: 1
        Space: 0
      VulkanBinding:
        Binding: 1
...
#--- end

# RUN: split-file %s %t
# RUN: %dxc_target -T cs_6_0 -E CSMain -Fo %t.o %t/source.hlsl
# RUN: %offloader %t/pipeline.yaml %t.o | FileCheck %s

# CHECK: Name: In
# CHECK: Format: Int32
# CHECK: Data: [ 1, 2, 3, 4, 5, 6, 7, 8 ]
# CHECK: Name: Out
# CHECK: Format: Int32
# CHECK: Data: [ 1, 2, 3, 4, 5, 6, 7, 8 ]
```

Simple Example

```
#--- source.hlsl
RWBuffer<int> In : register(u0);
RWBuffer<int> Out : register(u1);

[numthreads(8,1,1)]
void CSMain(uint3 TID : SV_GroupThreadID) {
  Out[TID.x] = In[TID.x];
}

//--- pipeline.yaml

---
Shaders:
  - Stage: Compute
    Entry: CSMain
    DispatchSize: [1, 1, 1]
Buffers:
  - Name: In
    Format: Int32
    Data: [ 1, 2, 3, 4, 5, 6, 7, 8]
  - Name: Out
    Format: Int32
    Data: [ 9, 10, 11, 12, 13, 14, 15, 16]
DescriptorSets:
  - Resources:
    - Name: In
      Kind: RWBuffer
      DirectXBinding:
        Register: 0
        Space: 0
      VulkanBinding:
        Binding: 0
    - Name: Out
      Kind: RWBuffer
      DirectXBinding:
        Register: 1
        Space: 0
      VulkanBinding:
        Binding: 1
...
#--- end

# RUN: split-file %s %t
# RUN: %dxc_target -T cs_6_0 -E CSMain -Fo %t.o %t/source.hlsl
# RUN: %offloader %t/pipeline.yaml %t.o | FileCheck %s

# CHECK: Name: In
# CHECK: Format: Int32
# CHECK: Data: [ 1, 2, 3, 4, 5, 6, 7, 8 ]
# CHECK: Name: Out
# CHECK: Format: Int32
# CHECK: Data: [ 1, 2, 3, 4, 5, 6, 7, 8 ]
```

Simple Example

```
RWBuffer<int> In : register(u0);
RWBuffer<int> Out : register(u1);

[numthreads(8,1,1)]
void CSMain(uint3 TID : SV_GroupThreadID) {
  Out[TID.x] = In[TID.x];
}
```

```
#--- source.hlsl
RWBuffer<int> In : register(u0);
RWBuffer<int> Out : register(u1);

[numthreads(8,1,1)]
void CSMain(uint3 TID : SV_GroupThreadID) {
  Out[TID.x] = In[TID.x];
}

//--- pipeline.yaml

---
Shaders:
  - Stage: Compute
    Entry: CSMain
    DispatchSize: [1, 1, 1]
Buffers:
  - Name: In
    Format: Int32
    Data: [ 1, 2, 3, 4, 5, 6, 7, 8]
  - Name: Out
    Format: Int32
    Data: [ 9, 10, 11, 12, 13, 14, 15, 16]
DescriptorSets:
  - Resources:
    - Name: In
      Kind: RWBuffer
      DirectXBinding:
        Register: 0
        Space: 0
      VulkanBinding:
        Binding: 0
    - Name: Out
      Kind: RWBuffer
      DirectXBinding:
        Register: 1
        Space: 0
      VulkanBinding:
        Binding: 1
...
#--- end

# RUN: split-file %s %t
# RUN: %dxc_target -T cs_6_0 -E CSMain -Fo %t.o %t/source.hlsl
# RUN: %offloader %t/pipeline.yaml %t.o | FileCheck %s

# CHECK: Name: In
# CHECK: Format: Int32
# CHECK: Data: [ 1, 2, 3, 4, 5, 6, 7, 8 ]
# CHECK: Name: Out
# CHECK: Format: Int32
# CHECK: Data: [ 1, 2, 3, 4, 5, 6, 7, 8 ]
```

Simple Example

```
#--- source.hlsl
RWBuffer<int> In : register(u0);
RWBuffer<int> Out : register(u1);

[numthreads(8,1,1)]
void CSMain(uint3 TID : SV_GroupThreadID) {
  Out[TID.x] = In[TID.x];
}

//--- pipeline.yaml

---
Shaders:
  - Stage: Compute
    Entry: CSMain
    DispatchSize: [1, 1, 1]
Buffers:
  - Name: In
    Format: Int32
    Data: [ 1, 2, 3, 4, 5, 6, 7, 8]
  - Name: Out
    Format: Int32
    Data: [ 9, 10, 11, 12, 13, 14, 15, 16]
DescriptorSets:
  - Resources:
    - Name: In
      Kind: RWBuffer
      DirectXBinding:
        Register: 0
        Space: 0
      VulkanBinding:
        Binding: 0
    - Name: Out
      Kind: RWBuffer
      DirectXBinding:
        Register: 1
        Space: 0
      VulkanBinding:
        Binding: 1
...
#--- end

# RUN: split-file %s %t
# RUN: %dxc_target -T cs_6_0 -E CSMain -Fo %t.o %t/source.hlsl
# RUN: %offloader %t/pipeline.yaml %t.o | FileCheck %s

# CHECK: Name: In
# CHECK: Format: Int32
# CHECK: Data: [ 1, 2, 3, 4, 5, 6, 7, 8 ]
# CHECK: Name: Out
# CHECK: Format: Int32
# CHECK: Data: [ 1, 2, 3, 4, 5, 6, 7, 8 ]
```

Simple Example

```yaml
---
Shaders:
  - Stage: Compute
    Entry: CSMain
    DispatchSize: [1, 1, 1]
Buffers:
  - Name: In
    Format: Int32
    Data: [ 1, 2, 3, 4, 5, 6, 7, 8]
  - Name: Out
    Format: Int32
    Data: [ 9, 10, 11, 12, 13, 14, 15, 16]
DescriptorSets:
  - Resources:
    - Name: In
      Kind: RWBuffer
      DirectXBinding:
        Register: 0
        Space: 0
      VulkanBinding:
        Binding: 0
    - Name: Out
      Kind: RWBuffer
      DirectXBinding:
        Register: 1
        Space: 0
      VulkanBinding:
        Binding: 1
...
```

```
#--- source.hlsl
RWBuffer<int> In : register(u0);
RWBuffer<int> Out : register(u1);

[numthreads(8,1,1)]
void CSMain(uint3 TID : SV_GroupThreadID) {
  Out[TID.x] = In[TID.x];
}

//--- pipeline.yaml

---
Shaders:
  - Stage: Compute
    Entry: CSMain
    DispatchSize: [1, 1, 1]
Buffers:
  - Name: In
    Format: Int32
    Data: [ 1, 2, 3, 4, 5, 6, 7, 8]
  - Name: Out
    Format: Int32
    Data: [ 9, 10, 11, 12, 13, 14, 15, 16]
DescriptorSets:
  - Resources:
    - Name: In
      Kind: RWBuffer
      DirectXBinding:
        Register: 0
        Space: 0
      VulkanBinding:
        Binding: 0
    - Name: Out
      Kind: RWBuffer
      DirectXBinding:
        Register: 1
        Space: 0
      VulkanBinding:
        Binding: 1
...
#--- end

# RUN: split-file %s %t
# RUN: %dxc_target -T cs_6_0 -E CSMain -Fo %t.o %t/source.hlsl
# RUN: %offloader %t/pipeline.yaml %t.o | FileCheck %s

# CHECK: Name: In
# CHECK: Format: Int32
# CHECK: Data: [ 1, 2, 3, 4, 5, 6, 7, 8 ]
# CHECK: Name: Out
# CHECK: Format: Int32
# CHECK: Data: [ 1, 2, 3, 4, 5, 6, 7, 8 ]
```

Simple Example

```
#--- source.hlsl
RWBuffer<int> In : register(u0);
RWBuffer<int> Out : register(u1);

[numthreads(8,1,1)]
void CSMain(uint3 TID : SV_GroupThreadID) {
  Out[TID.x] = In[TID.x];
}

//--- pipeline.yaml

---
Shaders:
  - Stage: Compute
    Entry: CSMain
    DispatchSize: [1, 1, 1]
Buffers:
  - Name: In
    Format: Int32
    Data: [ 1, 2, 3, 4, 5, 6, 7, 8]
  - Name: Out
    Format: Int32
    Data: [ 9, 10, 11, 12, 13, 14, 15, 16]
DescriptorSets:
  - Resources:
    - Name: In
      Kind: RWBuffer
      DirectXBinding:
        Register: 0
        Space: 0
      VulkanBinding:
        Binding: 0
    - Name: Out
      Kind: RWBuffer
      DirectXBinding:
        Register: 1
        Space: 0
      VulkanBinding:
        Binding: 1
...
#--- end

# RUN: split-file %s %t
# RUN: %dxc_target -T cs_6_0 -E CSMain -Fo %t.o %t/source.hlsl
# RUN: %offloader %t/pipeline.yaml %t.o | FileCheck %s

# CHECK: Name: In
# CHECK: Format: Int32
# CHECK: Data: [ 1, 2, 3, 4, 5, 6, 7, 8 ]
# CHECK: Name: Out
# CHECK: Format: Int32
# CHECK: Data: [ 1, 2, 3, 4, 5, 6, 7, 8 ]
```

Simple Example

```
# RUN: split-file %s %t
# RUN: %dxc_target -T cs_6_0 -E CSMain -Fo %t.o %t/source.hlsl
# RUN: %offloader %t/pipeline.yaml %t.o | FileCheck %s

# CHECK: Name: In
# CHECK: Format: Int32
# CHECK: Data: [ 1, 2, 3, 4, 5, 6, 7, 8 ]
# CHECK: Name: Out
# CHECK: Format: Int32
# CHECK: Data: [ 1, 2, 3, 4, 5, 6, 7, 8 ]
```

```
#--- source.hlsl
RWBuffer<int> In : register(u0);
RWBuffer<int> Out : register(u1);

[numthreads(8,1,1)]
void CSMain(uint3 TID : SV_GroupThreadID) {
  Out[TID.x] = In[TID.x];
}

//--- pipeline.yaml

---
Shaders:
  - Stage: Compute
    Entry: CSMain
    DispatchSize: [1, 1, 1]
Buffers:
  - Name: In
    Format: Int32
    Data: [ 1, 2, 3, 4, 5, 6, 7, 8]
  - Name: Out
    Format: Int32
    Data: [ 9, 10, 11, 12, 13, 14, 15, 16]
DescriptorSets:
  - Resources:
    - Name: In
      Kind: RWBuffer
      DirectXBinding:
        Register: 0
        Space: 0
      VulkanBinding:
        Binding: 0
    - Name: Out
      Kind: RWBuffer
      DirectXBinding:
        Register: 1
        Space: 0
      VulkanBinding:
        Binding: 1
...
#--- end

# RUN: split-file %s %t
# RUN: %dxc_target -T cs_6_0 -E CSMain -Fo %t.o %t/source.hlsl
# RUN: %offloader %t/pipeline.yaml %t.o | FileCheck %s

# CHECK: Name: In
# CHECK: Format: Int32
# CHECK: Data: [ 1, 2, 3, 4, 5, 6, 7, 8 ]
# CHECK: Name: Out
# CHECK: Format: Int32
# CHECK: Data: [ 1, 2, 3, 4, 5, 6, 7, 8 ]
```

Simple Example

- Clean abstraction for simple things
- Simple tests are simple to write
- And we wrote a lot of them!

```
#--- source.hlsl
RWBuffer<int> In : register(u0);
RWBuffer<int> Out : register(u1);

[numthreads(8,1,1)]
void CSMain(uint3 TID : SV_GroupThreadID) {
  Out[TID.x] = In[TID.x];
}

//--- pipeline.yaml

---
Shaders:
  - Stage: Compute
    Entry: CSMain
    DispatchSize: [1, 1, 1]
Buffers:
  - Name: In
    Format: Int32
    Data: [ 1, 2, 3, 4, 5, 6, 7, 8]
  - Name: Out
    Format: Int32
    Data: [ 9, 10, 11, 12, 13, 14, 15, 16]
DescriptorSets:
  - Resources:
    - Name: In
      Kind: RWBuffer
      DirectXBinding:
        Register: 0
        Space: 0
      VulkanBinding:
        Binding: 0
    - Name: Out
      Kind: RWBuffer
      DirectXBinding:
        Register: 1
        Space: 0
      VulkanBinding:
        Binding: 1
...
#--- end

# RUN: split-file %s %t
# RUN: %dxc_target -T cs_6_0 -E CSMain -Fo %t.o %t/source.hlsl
# RUN: %offloader %t/pipeline.yaml %t.o | FileCheck %s

# CHECK: Name: In
# CHECK: Format: Int32
# CHECK: Data: [ 1, 2, 3, 4, 5, 6, 7, 8 ]
# CHECK: Name: Out
# CHECK: Format: Int32
# CHECK: Data: [ 1, 2, 3, 4, 5, 6, 7, 8 ]
```

A little more complicated…

```
#--- source.hlsl
StructuredBuffer<float4> X : register(t0);
StructuredBuffer<float4> Y : register(t1);

RWStructuredBuffer<float4> Out : register(u2);


[numthreads(1,1,1)]
void main() {
  // Only accepts vectors of length 3
  Out[0] = float4(cross(X[0].xyz, Y[0].xyz), 0);
  Out[1] = float4(cross(X[1].xyz, Y[1].xyz), 0);
  Out[2] = float4(cross(X[2].xyz, Y[2].xyz), 0);
  Out[3] = float4(cross(float3(1, 0, 0), float3(0, 1, 0)), 0);
}
//--- pipeline.yaml

---
Shaders:
  - Stage: Compute
    Entry: main
    DispatchSize: [1, 1, 1]
Buffers:
  - Name: X
    Format: Float32
    Stride: 16
    Data: [ 1, 0, 0, 0, 2, 3, 4, 0, -1.25, -2.5, -3, 0 ] # Every 4th value is filler
  - Name: Y
    Format: Float32
    Stride: 16
    Data: [ 0, 1, 0, 0, 4, 6, 8, 0, 4.25, 5, 6.75, 0 ] # Every 4th value is filler
  - Name: Out
    Format: Float32
    Stride: 16
    FillSize: 64
  - Name: ExpectedOut
    Format: Float32
    Stride: 16
    Data: [ 0, 0, 1, 0, 0, 0, 0, 0, -1.875, -4.3125, 4.375, 0, 0, 0, 1, 0 ] # Every 4th value is filler
Results:
  - Result: Test0
    Rule: BufferFloatEpsilon
    Epsilon: 0.0008
    Actual: Out
    Expected: ExpectedOut
DescriptorSets:
  - Resources:
    - Name: X
      Kind: StructuredBuffer
      DirectXBinding:
        Register: 0
        Space: 0
      VulkanBinding:
        Binding: 0
    - Name: Y
      Kind: StructuredBuffer
      DirectXBinding:
        Register: 1
        Space: 0
      VulkanBinding:
        Binding: 1
    - Name: Out
      Kind: RWStructuredBuffer
      DirectXBinding:
        Register: 2
        Space: 0
      VulkanBinding:
        Binding: 2
#--- end

# RUN: split-file %s %t
# RUN: %dxc_target -HV 202x -T cs_6_5 -Fo %t.o %t/source.hlsl
# RUN: %offloader %t/pipeline.yaml %t.o
```

A little more complicated…

```
StructuredBuffer<float4> X : register(t0);
StructuredBuffer<float4> Y : register(t1);

RWStructuredBuffer<float4> Out : register(u2);


[numthreads(1,1,1)]
void main() {
  // Only accepts vectors of length 3
  Out[0] = float4(cross(X[0].xyz, Y[0].xyz), 0);
  Out[1] = float4(cross(X[1].xyz, Y[1].xyz), 0);
  Out[2] = float4(cross(X[2].xyz, Y[2].xyz), 0);
  Out[3] = float4(cross(float3(1, 0, 0), float3(0, 1, 0)), 0);
}
```

```
#--- source.hlsl
StructuredBuffer<float4> X : register(t0);
StructuredBuffer<float4> Y : register(t1);

RWStructuredBuffer<float4> Out : register(u2);


[numthreads(1,1,1)]
void main() {
  // Only accepts vectors of length 3
  Out[0] = float4(cross(X[0].xyz, Y[0].xyz), 0);
  Out[1] = float4(cross(X[1].xyz, Y[1].xyz), 0);
  Out[2] = float4(cross(X[2].xyz, Y[2].xyz), 0);
  Out[3] = float4(cross(float3(1, 0, 0), float3(0, 1, 0)), 0);
}
//--- pipeline.yaml

---
Shaders:
  - Stage: Compute
    Entry: main
    DispatchSize: [1, 1, 1]
Buffers:
  - Name: X
    Format: Float32
    Stride: 16
    Data: [ 1, 0, 0, 0, 2, 3, 4, 0, -1.25, -2.5, -3, 0 ] # Every 4th value is filler
  - Name: Y
    Format: Float32
    Stride: 16
    Data: [ 0, 1, 0, 0, 4, 6, 8, 0, 4.25, 5, 6.75, 0 ] # Every 4th value is filler
  - Name: Out
    Format: Float32
    Stride: 16
    FillSize: 64
  - Name: ExpectedOut
    Format: Float32
    Stride: 16
    Data: [ 0, 0, 1, 0, 0, 0, 0, 0, -1.875, -4.3125, 4.375, 0, 0, 0, 1, 0 ] # Every 4th value is filler
Results:
  - Result: Test0
    Rule: BufferFloatEpsilon
    Epsilon: 0.0008
    Actual: Out
    Expected: ExpectedOut
DescriptorSets:
  - Resources:
    - Name: X
      Kind: StructuredBuffer
      DirectXBinding:
        Register: 0
        Space: 0
      VulkanBinding:
        Binding: 0
    - Name: Y
      Kind: StructuredBuffer
      DirectXBinding:
        Register: 1
        Space: 0
      VulkanBinding:
        Binding: 1
    - Name: Out
      Kind: RWStructuredBuffer
      DirectXBinding:
        Register: 2
        Space: 0
      VulkanBinding:
        Binding: 2
#--- end

# RUN: split-file %s %t
# RUN: %dxc_target -HV 202x -T cs_6_5 -Fo %t.o %t/source.hlsl
# RUN: %offloader %t/pipeline.yaml %t.o
```

A little more complicated…

```
#--- source.hlsl
StructuredBuffer<float4> X : register(t0);
StructuredBuffer<float4> Y : register(t1);

RWStructuredBuffer<float4> Out : register(u2);


[numthreads(1,1,1)]
void main() {
  // Only accepts vectors of length 3
  Out[0] = float4(cross(X[0].xyz, Y[0].xyz), 0);
  Out[1] = float4(cross(X[1].xyz, Y[1].xyz), 0);
  Out[2] = float4(cross(X[2].xyz, Y[2].xyz), 0);
  Out[3] = float4(cross(float3(1, 0, 0), float3(0, 1, 0)), 0);
}
//--- pipeline.yaml

---
Shaders:
  - Stage: Compute
    Entry: main
    DispatchSize: [1, 1, 1]
Buffers:
  - Name: X
    Format: Float32
    Stride: 16
    Data: [ 1, 0, 0, 0, 2, 3, 4, 0, -1.25, -2.5, -3, 0 ] # Every 4th value is filler
  - Name: Y
    Format: Float32
    Stride: 16
    Data: [ 0, 1, 0, 0, 4, 6, 8, 0, 4.25, 5, 6.75, 0 ] # Every 4th value is filler
  - Name: Out
    Format: Float32
    Stride: 16
    FillSize: 64
  - Name: ExpectedOut
    Format: Float32
    Stride: 16
    Data: [ 0, 0, 1, 0, 0, 0, 0, 0, -1.875, -4.3125, 4.375, 0, 0, 0, 1, 0 ] # Every 4th value is filler
Results:
  - Result: Test0
    Rule: BufferFloatEpsilon
    Epsilon: 0.0008
    Actual: Out
    Expected: ExpectedOut
DescriptorSets:
  - Resources:
    - Name: X
      Kind: StructuredBuffer
      DirectXBinding:
        Register: 0
        Space: 0
      VulkanBinding:
        Binding: 0
    - Name: Y
      Kind: StructuredBuffer
      DirectXBinding:
        Register: 1
        Space: 0
      VulkanBinding:
        Binding: 1
    - Name: Out
      Kind: RWStructuredBuffer
      DirectXBinding:
        Register: 2
        Space: 0
      VulkanBinding:
        Binding: 2
#--- end

# RUN: split-file %s %t
# RUN: %dxc_target -HV 202x -T cs_6_5 -Fo %t.o %t/source.hlsl
# RUN: %offloader %t/pipeline.yaml %t.o
```

A little more complicated…

```
Buffers:
  - Name: X
    Format: Float32
    Stride: 16
    Data: [ 1, 0, 0, 0, 2, 3, 4, 0, -1.25, -2.5, -3, 0 ] # Every 4th value is filler
  - Name: Y
    Format: Float32
    Stride: 16
    Data: [ 0, 1, 0, 0, 4, 6, 8, 0, 4.25, 5, 6.75, 0 ] # Every 4th value is filler
  - Name: Out
    Format: Float32
    Stride: 16
    FillSize: 64
  - Name: ExpectedOut
    Format: Float32
    Stride: 16
    Data: [ 0, 0, 1, 0, 0, 0, 0, 0, -1.875, -4.3125, 4.375, 0, 0, 0, 1, 0 ] # Every 4th value is filler
Results:
  - Result: Test0
    Rule: BufferFloatEpsilon
    Epsilon: 0.0008
    Actual: Out
    Expected: ExpectedOut
```

```
#--- source.hlsl
StructuredBuffer<float4> X : register(t0);
StructuredBuffer<float4> Y : register(t1);

RWStructuredBuffer<float4> Out : register(u2);


[numthreads(1,1,1)]
void main() {
  // Only accepts vectors of length 3
  Out[0] = float4(cross(X[0].xyz, Y[0].xyz), 0);
  Out[1] = float4(cross(X[1].xyz, Y[1].xyz), 0);
  Out[2] = float4(cross(X[2].xyz, Y[2].xyz), 0);
  Out[3] = float4(cross(float3(1, 0, 0), float3(0, 1, 0)), 0);
}
//--- pipeline.yaml

---
Shaders:
  - Stage: Compute
    Entry: main
    DispatchSize: [1, 1, 1]
Buffers:
  - Name: X
    Format: Float32
    Stride: 16
    Data: [ 1, 0, 0, 0, 2, 3, 4, 0, -1.25, -2.5, -3, 0 ] # Every 4th value is filler
  - Name: Y
    Format: Float32
    Stride: 16
    Data: [ 0, 1, 0, 0, 4, 6, 8, 0, 4.25, 5, 6.75, 0 ] # Every 4th value is filler
  - Name: Out
    Format: Float32
    Stride: 16
    FillSize: 64
  - Name: ExpectedOut
    Format: Float32
    Stride: 16
    Data: [ 0, 0, 1, 0, 0, 0, 0, 0, -1.875, -4.3125, 4.375, 0, 0, 0, 1, 0 ] # Every 4th value is filler
Results:
  - Result: Test0
    Rule: BufferFloatEpsilon
    Epsilon: 0.0008
    Actual: Out
    Expected: ExpectedOut
DescriptorSets:
  - Resources:
    - Name: X
      Kind: StructuredBuffer
      DirectXBinding:
        Register: 0
        Space: 0
      VulkanBinding:
        Binding: 0
    - Name: Y
      Kind: StructuredBuffer
      DirectXBinding:
        Register: 1
        Space: 0
      VulkanBinding:
        Binding: 1
    - Name: Out
      Kind: RWStructuredBuffer
      DirectXBinding:
        Register: 2
        Space: 0
      VulkanBinding:
        Binding: 2
#--- end

# RUN: split-file %s %t
# RUN: %dxc_target -HV 202x -T cs_6_5 -Fo %t.o %t/source.hlsl
# RUN: %offloader %t/pipeline.yaml %t.o
```

A little more complicated…

```
#--- source.hlsl
StructuredBuffer<float4> X : register(t0);
StructuredBuffer<float4> Y : register(t1);

RWStructuredBuffer<float4> Out : register(u2);


[numthreads(1,1,1)]
void main() {
  // Only accepts vectors of length 3
  Out[0] = float4(cross(X[0].xyz, Y[0].xyz), 0);
  Out[1] = float4(cross(X[1].xyz, Y[1].xyz), 0);
  Out[2] = float4(cross(X[2].xyz, Y[2].xyz), 0);
  Out[3] = float4(cross(float3(1, 0, 0), float3(0, 1, 0)), 0);
}
//--- pipeline.yaml

---
Shaders:
  - Stage: Compute
    Entry: main
    DispatchSize: [1, 1, 1]
Buffers:
  - Name: X
    Format: Float32
    Stride: 16
    Data: [ 1, 0, 0, 0, 2, 3, 4, 0, -1.25, -2.5, -3, 0 ] # Every 4th value is filler
  - Name: Y
    Format: Float32
    Stride: 16
    Data: [ 0, 1, 0, 0, 4, 6, 8, 0, 4.25, 5, 6.75, 0 ] # Every 4th value is filler
  - Name: Out
    Format: Float32
    Stride: 16
    FillSize: 64
  - Name: ExpectedOut
    Format: Float32
    Stride: 16
    Data: [ 0, 0, 1, 0, 0, 0, 0, 0, -1.875, -4.3125, 4.375, 0, 0, 0, 1, 0 ] # Every 4th value is filler
Results:
  - Result: Test0
    Rule: BufferFloatEpsilon
    Epsilon: 0.0008
    Actual: Out
    Expected: ExpectedOut
DescriptorSets:
  - Resources:
    - Name: X
      Kind: StructuredBuffer
      DirectXBinding:
        Register: 0
        Space: 0
      VulkanBinding:
        Binding: 0
    - Name: Y
      Kind: StructuredBuffer
      DirectXBinding:
        Register: 1
        Space: 0
      VulkanBinding:
        Binding: 1
    - Name: Out
      Kind: RWStructuredBuffer
      DirectXBinding:
        Register: 2
        Space: 0
      VulkanBinding:
        Binding: 2
#--- end

# RUN: split-file %s %t
# RUN: %dxc_target -HV 202x -T cs_6_5 -Fo %t.o %t/source.hlsl
# RUN: %offloader %t/pipeline.yaml %t.o
```

A little more complicated…

```
# RUN: split-file %s %t
# RUN: %dxc_target -HV 202x -T cs_6_5 -Fo %t.o %t/source.hlsl
# RUN: %offloader %t/pipeline.yaml %t.o
```

```
#--- source.hlsl
StructuredBuffer<float4> X : register(t0);
StructuredBuffer<float4> Y : register(t1);

RWStructuredBuffer<float4> Out : register(u2);


[numthreads(1,1,1)]
void main() {
  // Only accepts vectors of length 3
  Out[0] = float4(cross(X[0].xyz, Y[0].xyz), 0);
  Out[1] = float4(cross(X[1].xyz, Y[1].xyz), 0);
  Out[2] = float4(cross(X[2].xyz, Y[2].xyz), 0);
  Out[3] = float4(cross(float3(1, 0, 0), float3(0, 1, 0)), 0);
}
//--- pipeline.yaml

---
Shaders:
  - Stage: Compute
    Entry: main
    DispatchSize: [1, 1, 1]
Buffers:
  - Name: X
    Format: Float32
    Stride: 16
    Data: [ 1, 0, 0, 0, 2, 3, 4, 0, -1.25, -2.5, -3, 0 ] # Every 4th value is filler
  - Name: Y
    Format: Float32
    Stride: 16
    Data: [ 0, 1, 0, 0, 4, 6, 8, 0, 4.25, 5, 6.75, 0 ] # Every 4th value is filler
  - Name: Out
    Format: Float32
    Stride: 16
    FillSize: 64
  - Name: ExpectedOut
    Format: Float32
    Stride: 16
    Data: [ 0, 0, 1, 0, 0, 0, 0, 0, -1.875, -4.3125, 4.375, 0, 0, 0, 1, 0 ] # Every 4th value is filler
Results:
  - Result: Test0
    Rule: BufferFloatEpsilon
    Epsilon: 0.0008
    Actual: Out
    Expected: ExpectedOut
DescriptorSets:
  - Resources:
    - Name: X
      Kind: StructuredBuffer
      DirectXBinding:
        Register: 0
        Space: 0
      VulkanBinding:
        Binding: 0
    - Name: Y
      Kind: StructuredBuffer
      DirectXBinding:
        Register: 1
        Space: 0
      VulkanBinding:
        Binding: 1
    - Name: Out
      Kind: RWStructuredBuffer
      DirectXBinding:
        Register: 2
        Space: 0
      VulkanBinding:
        Binding: 2
#--- end

# RUN: split-file %s %t
# RUN: %dxc_target -HV 202x -T cs_6_5 -Fo %t.o %t/source.hlsl
# RUN: %offloader %t/pipeline.yaml %t.o
```

Also supports graphics!

```
#--- vertex.hlsl
struct PSInput {
    float4 position : SV_POSITION;
    float4 color : COLOR;
};

PSInput main(float4 position : POSITION, float4 color : COLOR) {
    PSInput result;
    result.position = position;
    result.color = color;
    return result;
}
#--- pixel.hlsl
struct PSInput {
    float4 position : SV_POSITION;
    float4 color : COLOR;
};

float4 main(PSInput input) : SV_TARGET {
    return input.color;
}
#--- pipeline.yaml
---
Shaders:
  - Stage: Vertex
    Entry: main
  - Stage: Pixel
    Entry: main
Buffers:
  - Name: VertexData
    Format: Float32
    Stride: 28 # 32 bytes per vertex
    Data: [ 0.0, 0.25, 0.0, 1.0, 0.0, 0.0, 1.0,
            0.25, -0.25, 0.0, 0.0, 1.0, 0.0, 1.0,
           -0.25, -0.25, 0.0, 0.0, 0.0, 1.0, 1.0 ]
  - Name: Output
    Format: Float32
    Channels: 4
    ZeroInitSize: 1048576 # 256x256 @ 16 bytes per pixel
    OutputProps:
      Height: 256
      Width: 256
      Depth: 16
Bindings:
  VertexBuffer: VertexData
  VertexAttributes:
    - Format: Float32
      Channels: 3
      Offset: 0
      Name: POSITION
    - Format: Float32
      Channels: 4
      Offset: 12
      Name: COLOR
  RenderTarget: Output
DescriptorSets: []
...
#--- rules.yaml
---
- Type: PixelPercent
  Val: 0.2 # No more than 0.2% of pixels may be visibly different.
...
#--- end

# UNSUPPORTED: Clang
# REQUIRES: goldenimage

# RUN: split-file %s %t
# RUN: %dxc_target -T vs_6_0 -Fo %t-vertex.o %t/vertex.hlsl
# RUN: %dxc_target -T ps_6_0 -Fo %t-pixel.o %t/pixel.hlsl
# RUN: %offloader %t/pipeline.yaml %t-vertex.o %t-pixel.o -r Output -o %t/Output.png
# RUN: imgdiff %t/Output.png %goldenimage_dir/hlsl/Graphics/SimpleTriangle.png -rules %t/rules.yaml
```

Also supports graphics!

```
#--- vertex.hlsl
struct PSInput {
    float4 position : SV_POSITION;
    float4 color : COLOR;
};

PSInput main(float4 position : POSITION, float4 color : COLOR) {
    PSInput result;
    result.position = position;
    result.color = color;
    return result;
}
#--- pixel.hlsl
struct PSInput {
    float4 position : SV_POSITION;
    float4 color : COLOR;
};

float4 main(PSInput input) : SV_TARGET {
    return input.color;
}
```

```
#--- vertex.hlsl
struct PSInput {
    float4 position : SV_POSITION;
    float4 color : COLOR;
};

PSInput main(float4 position : POSITION, float4 color : COLOR) {
    PSInput result;
    result.position = position;
    result.color = color;
    return result;
}
#--- pixel.hlsl
struct PSInput {
    float4 position : SV_POSITION;
    float4 color : COLOR;
};

float4 main(PSInput input) : SV_TARGET {
    return input.color;
}
#--- pipeline.yaml
---
Shaders:
  - Stage: Vertex
    Entry: main
  - Stage: Pixel
    Entry: main
Buffers:
  - Name: VertexData
    Format: Float32
    Stride: 28 # 32 bytes per vertex
    Data: [ 0.0, 0.25, 0.0, 1.0, 0.0, 0.0, 1.0,
            0.25, -0.25, 0.0, 0.0, 1.0, 0.0, 1.0,
           -0.25, -0.25, 0.0, 0.0, 0.0, 1.0, 1.0 ]
  - Name: Output
    Format: Float32
    Channels: 4
    ZeroInitSize: 1048576 # 256x256 @ 16 bytes per pixel
    OutputProps:
      Height: 256
      Width: 256
      Depth: 16
Bindings:
  VertexBuffer: VertexData
  VertexAttributes:
    - Format: Float32
      Channels: 3
      Offset: 0
      Name: POSITION
    - Format: Float32
      Channels: 4
      Offset: 12
      Name: COLOR
  RenderTarget: Output
DescriptorSets: []
...
#--- rules.yaml
---
- Type: PixelPercent
  Val: 0.2 # No more than 0.2% of pixels may be visibly different.
...
#--- end

# UNSUPPORTED: Clang
# REQUIRES: goldenimage

# RUN: split-file %s %t
# RUN: %dxc_target -T vs_6_0 -Fo %t-vertex.o %t/vertex.hlsl
# RUN: %dxc_target -T ps_6_0 -Fo %t-pixel.o %t/pixel.hlsl
# RUN: %offloader %t/pipeline.yaml %t-vertex.o %t-pixel.o -r Output -o %t/Output.png
# RUN: imgdiff %t/Output.png %goldenimage_dir/hlsl/Graphics/SimpleTriangle.png -rules %t/rules.yaml
```

Also supports graphics!

```
#--- vertex.hlsl
struct PSInput {
    float4 position : SV_POSITION;
    float4 color : COLOR;
};

PSInput main(float4 position : POSITION, float4 color : COLOR) {
    PSInput result;
    result.position = position;
    result.color = color;
    return result;
}
#--- pixel.hlsl
struct PSInput {
    float4 position : SV_POSITION;
    float4 color : COLOR;
};

float4 main(PSInput input) : SV_TARGET {
    return input.color;
}
#--- pipeline.yaml
---
Shaders:
  - Stage: Vertex
    Entry: main
  - Stage: Pixel
    Entry: main
Buffers:
  - Name: VertexData
    Format: Float32
    Stride: 28 # 32 bytes per vertex
    Data: [ 0.0, 0.25, 0.0, 1.0, 0.0, 0.0, 1.0,
            0.25, -0.25, 0.0, 0.0, 1.0, 0.0, 1.0,
           -0.25, -0.25, 0.0, 0.0, 0.0, 1.0, 1.0 ]
  - Name: Output
    Format: Float32
    Channels: 4
    ZeroInitSize: 1048576 # 256x256 @ 16 bytes per pixel
    OutputProps:
      Height: 256
      Width: 256
      Depth: 16
Bindings:
  VertexBuffer: VertexData
  VertexAttributes:
    - Format: Float32
      Channels: 3
      Offset: 0
      Name: POSITION
    - Format: Float32
      Channels: 4
      Offset: 12
      Name: COLOR
  RenderTarget: Output
DescriptorSets: []
...
#--- rules.yaml
---
- Type: PixelPercent
  Val: 0.2 # No more than 0.2% of pixels may be visibly different.
...
#--- end

# UNSUPPORTED: Clang
# REQUIRES: goldenimage

# RUN: split-file %s %t
# RUN: %dxc_target -T vs_6_0 -Fo %t-vertex.o %t/vertex.hlsl
# RUN: %dxc_target -T ps_6_0 -Fo %t-pixel.o %t/pixel.hlsl
# RUN: %offloader %t/pipeline.yaml %t-vertex.o %t-pixel.o -r Output -o %t/Output.png
# RUN: imgdiff %t/Output.png %goldenimage_dir/hlsl/Graphics/SimpleTriangle.png -rules %t/rules.yaml
```

Also supports graphics!

```yaml
---
Shaders:
  - Stage: Vertex
    Entry: main
  - Stage: Pixel
    Entry: main
Buffers:
  - Name: VertexData
    Format: Float32
    Stride: 28 # 32 bytes per vertex
    Data: [ 0.0, 0.25, 0.0, 1.0, 0.0, 0.0, 1.0,
             0.25, -0.25, 0.0, 0.0, 1.0, 0.0, 1.0,
            -0.25, -0.25, 0.0, 0.0, 0.0, 1.0, 1.0 ]
  - Name: Output
    Format: Float32
    Channels: 4
    ZeroInitSize: 1048576 # 256x256 @ 16 bytes per pixel
    OutputProps:
      Height: 256
      Width: 256
      Depth: 16
Bindings:
  VertexBuffer: VertexData
  VertexAttributes:
    - Format: Float32
      Channels: 3
      Offset: 0
      Name: POSITION
    - Format: Float32
      Channels: 4
      Offset: 12
      Name: COLOR
  RenderTarget: Output
DescriptorSets: []
...
```

```
#--- vertex.hlsl
struct PSInput {
    float4 position : SV_POSITION;
    float4 color : COLOR;
};

PSInput main(float4 position : POSITION, float4 color : COLOR) {
    PSInput result;
    result.position = position;
    result.color = color;
    return result;
}
#--- pixel.hlsl
struct PSInput {
    float4 position : SV_POSITION;
    float4 color : COLOR;
};

float4 main(PSInput input) : SV_TARGET {
    return input.color;
}
#--- pipeline.yaml
---
Shaders:
  - Stage: Vertex
    Entry: main
  - Stage: Pixel
    Entry: main
Buffers:
  - Name: VertexData
    Format: Float32
    Stride: 28 # 32 bytes per vertex
    Data: [ 0.0, 0.25, 0.0, 1.0, 0.0, 0.0, 1.0,
             0.25, -0.25, 0.0, 0.0, 1.0, 0.0, 1.0,
            -0.25, -0.25, 0.0, 0.0, 0.0, 1.0, 1.0 ]
  - Name: Output
    Format: Float32
    Channels: 4
    ZeroInitSize: 1048576 # 256x256 @ 16 bytes per pixel
    OutputProps:
      Height: 256
      Width: 256
      Depth: 16
Bindings:
  VertexBuffer: VertexData
  VertexAttributes:
    - Format: Float32
      Channels: 3
      Offset: 0
      Name: POSITION
    - Format: Float32
      Channels: 4
      Offset: 12
      Name: COLOR
  RenderTarget: Output
DescriptorSets: []
...
#--- rules.yaml
---
- Type: PixelPercent
  Val: 0.2 # No more than 0.2% of pixels may be visibly different.
...
#--- end

# UNSUPPORTED: Clang
# REQUIRES: goldenimage

# RUN: split-file %s %t
# RUN: %dxc_target -T vs_6_0 -Fo %t-vertex.o %t/vertex.hlsl
# RUN: %dxc_target -T ps_6_0 -Fo %t-pixel.o %t/pixel.hlsl
# RUN: %offloader %t/pipeline.yaml %t-vertex.o %t-pixel.o -r Output -o %t/Output.png
# RUN: imgdiff %t/Output.png %goldenimage_dir/hlsl/Graphics/SimpleTriangle.png -rules %t/rules.yaml
```

Also supports graphics!

```yaml
---
Shaders:
  - Stage: Vertex
    Entry: main
  - Stage: Pixel
    Entry: main
Buffers:
  - Name: VertexData
    Format: Float32
    Stride: 28 # 32 bytes per vertex
    Data: [ 0.0, 0.25, 0.0, 1.0, 0.0, 0.0, 1.0,
            0.25, -0.25, 0.0, 0.0, 1.0, 0.0, 1.0,
           -0.25, -0.25, 0.0, 0.0, 0.0, 1.0, 1.0 ]
  - Name: Output
    Format: Float32
    Channels: 4
    ZeroInitSize: 1048576 # 256x256 @ 16 bytes per pixel
    OutputProps:
      Height: 256
      Width: 256
      Depth: 16
Bindings:
  VertexBuffer: VertexData
  VertexAttributes:
    - Format: Float32
      Channels: 3
      Offset: 0
      Name: POSITION
    - Format: Float32
      Channels: 4
      Offset: 12
      Name: COLOR
  RenderTarget: Output
DescriptorSets: []
...
```

```
#--- vertex.hlsl
struct PSInput {
    float4 position : SV_POSITION;
    float4 color : COLOR;
};

PSInput main(float4 position : POSITION, float4 color : COLOR) {
    PSInput result;
    result.position = position;
    result.color = color;
    return result;
}
#--- pixel.hlsl
struct PSInput {
    float4 position : SV_POSITION;
    float4 color : COLOR;
};

float4 main(PSInput input) : SV_TARGET {
    return input.color;
}
#--- pipeline.yaml
---
Shaders:
  - Stage: Vertex
    Entry: main
  - Stage: Pixel
    Entry: main
Buffers:
  - Name: VertexData
    Format: Float32
    Stride: 28 # 32 bytes per vertex
    Data: [ 0.0, 0.25, 0.0, 1.0, 0.0, 0.0, 1.0,
            0.25, -0.25, 0.0, 0.0, 1.0, 0.0, 1.0,
           -0.25, -0.25, 0.0, 0.0, 0.0, 1.0, 1.0 ]
  - Name: Output
    Format: Float32
    Channels: 4
    ZeroInitSize: 1048576 # 256x256 @ 16 bytes per pixel
    OutputProps:
      Height: 256
      Width: 256
      Depth: 16
Bindings:
  VertexBuffer: VertexData
  VertexAttributes:
    - Format: Float32
      Channels: 3
      Offset: 0
      Name: POSITION
    - Format: Float32
      Channels: 4
      Offset: 12
      Name: COLOR
  RenderTarget: Output
DescriptorSets: []
...
#--- rules.yaml
---
- Type: PixelPercent
  Val: 0.2 # No more than 0.2% of pixels may be visibly different.
...
#--- end

# UNSUPPORTED: Clang
# REQUIRES: goldenimage

# RUN: split-file %s %t
# RUN: %dxc_target -T vs_6_0 -Fo %t-vertex.o %t/vertex.hlsl
# RUN: %dxc_target -T ps_6_0 -Fo %t-pixel.o %t/pixel.hlsl
# RUN: %offloader %t/pipeline.yaml %t-vertex.o %t-pixel.o -r Output -o %t/Output.png
# RUN: imgdiff %t/Output.png %goldenimage_dir/hlsl/Graphics/SimpleTriangle.png -rules %t/rules.yaml
```

Also supports graphics!

```
#--- vertex.hlsl
struct PSInput {
    float4 position : SV_POSITION;
    float4 color : COLOR;
};

PSInput main(float4 position : POSITION, float4 color : COLOR) {
    PSInput result;
    result.position = position;
    result.color = color;
    return result;
}
#--- pixel.hlsl
struct PSInput {
    float4 position : SV_POSITION;
    float4 color : COLOR;
};

float4 main(PSInput input) : SV_TARGET {
    return input.color;
}
#--- pipeline.yaml
---
Shaders:
  - Stage: Vertex
    Entry: main
  - Stage: Pixel
    Entry: main
Buffers:
  - Name: VertexData
    Format: Float32
    Stride: 28 # 32 bytes per vertex
    Data: [ 0.0, 0.25, 0.0, 1.0, 0.0, 0.0, 1.0,
            0.25, -0.25, 0.0, 0.0, 1.0, 0.0, 1.0,
           -0.25, -0.25, 0.0, 0.0, 0.0, 1.0, 1.0 ]
  - Name: Output
    Format: Float32
    Channels: 4
    ZeroInitSize: 1048576 # 256x256 @ 16 bytes per pixel
    OutputProps:
      Height: 256
      Width: 256
      Depth: 16
Bindings:
  VertexBuffer: VertexData
  VertexAttributes:
    - Format: Float32
      Channels: 3
      Offset: 0
      Name: POSITION
    - Format: Float32
      Channels: 4
      Offset: 12
      Name: COLOR
  RenderTarget: Output
DescriptorSets: []
...
#--- rules.yaml
---
- Type: PixelPercent
  Val: 0.2 # No more than 0.2% of pixels may be visibly different.
...
#--- end

# UNSUPPORTED: Clang
# REQUIRES: goldenimage

# RUN: split-file %s %t
# RUN: %dxc_target -T vs_6_0 -Fo %t-vertex.o %t/vertex.hlsl
# RUN: %dxc_target -T ps_6_0 -Fo %t-pixel.o %t/pixel.hlsl
# RUN: %offloader %t/pipeline.yaml %t-vertex.o %t-pixel.o -r Output -o %t/Output.png
# RUN: imgdiff %t/Output.png %goldenimage_dir/hlsl/Graphics/SimpleTriangle.png -rules %t/rules.yaml
```

Also supports graphics!

```
#--- rules.yaml
---
- Type: PixelPercent
  Val: 0.2 # No more than 0.2% of pixels may be visibly different.
...
#--- end

# UNSUPPORTED: Clang
# REQUIRES: goldenimage

# RUN: split-file %s %t
# RUN: %dxc_target -T vs_6_0 -Fo %t-vertex.o %t/vertex.hlsl
# RUN: %dxc_target -T ps_6_0 -Fo %t-pixel.o %t/pixel.hlsl
# RUN: %offloader %t/pipeline.yaml %t-vertex.o %t-pixel.o -r Output -o %t/Output.png
# RUN: imgdiff %t/Output.png %goldenimage_dir/hlsl/Graphics/SimpleTriangle.png -rules %t/rules.yaml
```

```
#--- vertex.hlsl
struct PSInput {
    float4 position : SV_POSITION;
    float4 color : COLOR;
};

PSInput main(float4 position : POSITION, float4 color : COLOR) {
    PSInput result;
    result.position = position;
    result.color = color;
    return result;
}
#--- pixel.hlsl
struct PSInput {
    float4 position : SV_POSITION;
    float4 color : COLOR;
};

float4 main(PSInput input) : SV_TARGET {
    return input.color;
}
#--- pipeline.yaml
---
Shaders:
  - Stage: Vertex
    Entry: main
  - Stage: Pixel
    Entry: main
Buffers:
  - Name: VertexData
    Format: Float32
    Stride: 28 # 32 bytes per vertex
    Data: [ 0.0, 0.25, 0.0, 1.0, 0.0, 0.0, 1.0,
            0.25, -0.25, 0.0, 0.0, 1.0, 0.0, 1.0,
           -0.25, -0.25, 0.0, 0.0, 0.0, 1.0, 1.0 ]
  - Name: Output
    Format: Float32
    Channels: 4
    ZeroInitSize: 1048576 # 256x256 @ 16 bytes per pixel
    OutputProps:
      Height: 256
      Width: 256
      Depth: 16
Bindings:
  VertexBuffer: VertexData
  VertexAttributes:
    - Format: Float32
      Channels: 3
      Offset: 0
      Name: POSITION
    - Format: Float32
      Channels: 4
      Offset: 12
      Name: COLOR
  RenderTarget: Output
DescriptorSets: []
...
#--- rules.yaml
---
- Type: PixelPercent
  Val: 0.2 # No more than 0.2% of pixels may be visibly different.
...
#--- end

# UNSUPPORTED: Clang
# REQUIRES: goldenimage

# RUN: split-file %s %t
# RUN: %dxc_target -T vs_6_0 -Fo %t-vertex.o %t/vertex.hlsl
# RUN: %dxc_target -T ps_6_0 -Fo %t-pixel.o %t/pixel.hlsl
# RUN: %offloader %t/pipeline.yaml %t-vertex.o %t-pixel.o -r Output -o %t/Output.png
# RUN: imgdiff %t/Output.png %goldenimage_dir/hlsl/Graphics/SimpleTriangle.png -rules %t/rules.yaml
```

Also supports graphics!

- Logic for fuzzy-matching buffers and images
  - Buffers: exact, epsilon and ULP
  - Images: CIELAB distance, RMS, flexible thresholds
- LIT features for API features
- Gives us a good basis for portability

```
#--- vertex.hlsl
struct PSInput {
    float4 position : SV_POSITION;
    float4 color : COLOR;
};

PSInput main(float4 position : POSITION, float4 color : COLOR) {
    PSInput result;
    result.position = position;
    result.color = color;
    return result;
}
#--- pixel.hlsl
struct PSInput {
    float4 position : SV_POSITION;
    float4 color : COLOR;
};

float4 main(PSInput input) : SV_TARGET {
    return input.color;
}
#--- pipeline.yaml
---
Shaders:
  - Stage: Vertex
    Entry: main
  - Stage: Pixel
    Entry: main
Buffers:
  - Name: VertexData
    Format: Float32
    Stride: 28 # 32 bytes per vertex
    Data: [ 0.0, 0.25, 0.0, 1.0, 0.0, 0.0, 1.0,
            0.25, -0.25, 0.0, 0.0, 1.0, 0.0, 1.0,
           -0.25, -0.25, 0.0, 0.0, 0.0, 1.0, 1.0 ]
  - Name: Output
    Format: Float32
    Channels: 4
    ZeroInitSize: 1048576 # 256x256 @ 16 bytes per pixel
    OutputProps:
      Height: 256
      Width: 256
      Depth: 16
Bindings:
  VertexBuffer: VertexData
  VertexAttributes:
    - Format: Float32
      Channels: 3
      Offset: 0
      Name: POSITION
    - Format: Float32
      Channels: 4
      Offset: 12
      Name: COLOR
  RenderTarget: Output
DescriptorSets: []
...
#--- rules.yaml
---
- Type: PixelPercent
  Val: 0.2 # No more than 0.2% of pixels may be visibly different.
...
#--- end

# UNSUPPORTED: Clang
# REQUIRES: goldenimage

# RUN: split-file %s %t
# RUN: %dxc_target -T vs_6_0 -Fo %t-vertex.o %t/vertex.hlsl
# RUN: %dxc_target -T ps_6_0 -Fo %t-pixel.o %t/pixel.hlsl
# RUN: %offloader %t/pipeline.yaml %t-vertex.o %t-pixel.o -r Output -o %t/Output.png
# RUN: imgdiff %t/Output.png %goldenimage_dir/hlsl/Graphics/SimpleTriangle.png -rules %t/rules.yaml
```

Growing set of hardware

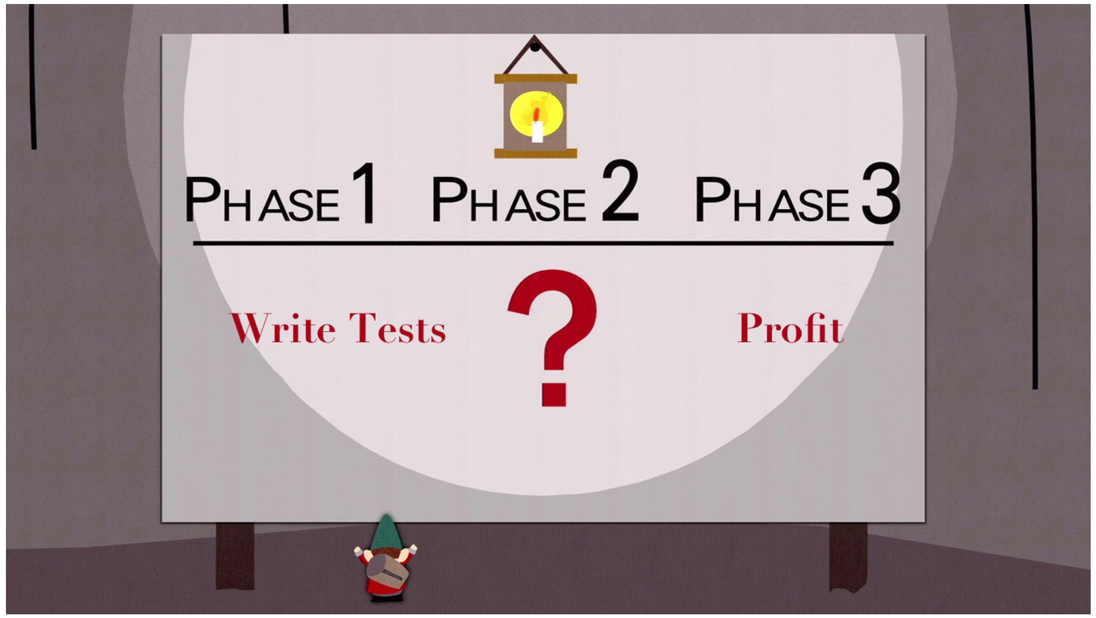
- Writing tests before implementing clang features
- Representative not exhaustive
  - Valuing cross-API and cross-hardware
- Configurations that we don’t have CI for
  - Vulkan on macOS
  - Linux, WSL (DX and VK)

Current Status

Testing Machine	DXC	Clang
Tier 1 Targets		
Windows DirectX12 Intel GPU	Windows D3D12 Intel DXC passing	Windows D3D12 Intel Clang passing
Windows DirectX12 Warp (x64 LKG)	Windows D3D12 Warp DXC passing	Windows D3D12 Warp Clang passing
Windows DirectX12 Warp (arm64 LKG)	Windows ARM64 D3D12 Warp DXC passing	Windows ARM64 D3D12 Warp Clang passing
Windows Vulkan Intel GPU	Windows Vulkan Intel DXC passing	Windows Vulkan Intel Clang passing
Tier 2 Targets		
macOS Apple M1	macOS Metal DXC passing	macOS Metal Clang passing
Experimental Targets		
Windows DirectX12 AMD GPU	Windows D3D12 AMD DXC failing	Windows D3D12 AMD Clang failing
Windows DirectX12 NVIDIA GPU	Windows D3D12 NVIDIA DXC failing	Windows D3D12 NVIDIA Clang failing
Windows DirectX12 Qualcomm GPU	Windows D3D12 QC DXC failing	Windows D3D12 QC Clang failing
Windows Vulkan AMD GPU	Windows Vulkan AMD DXC failing	Windows Vulkan AMD Clang failing
Windows Vulkan NVIDIA GPU	Windows Vulkan NVIDIA DXC failing	Windows Vulkan NVIDIA Clang failing
Windows Vulkan Qualcomm GPU	Windows Vulkan QC DXC failing	Windows Vulkan QC Clang failing

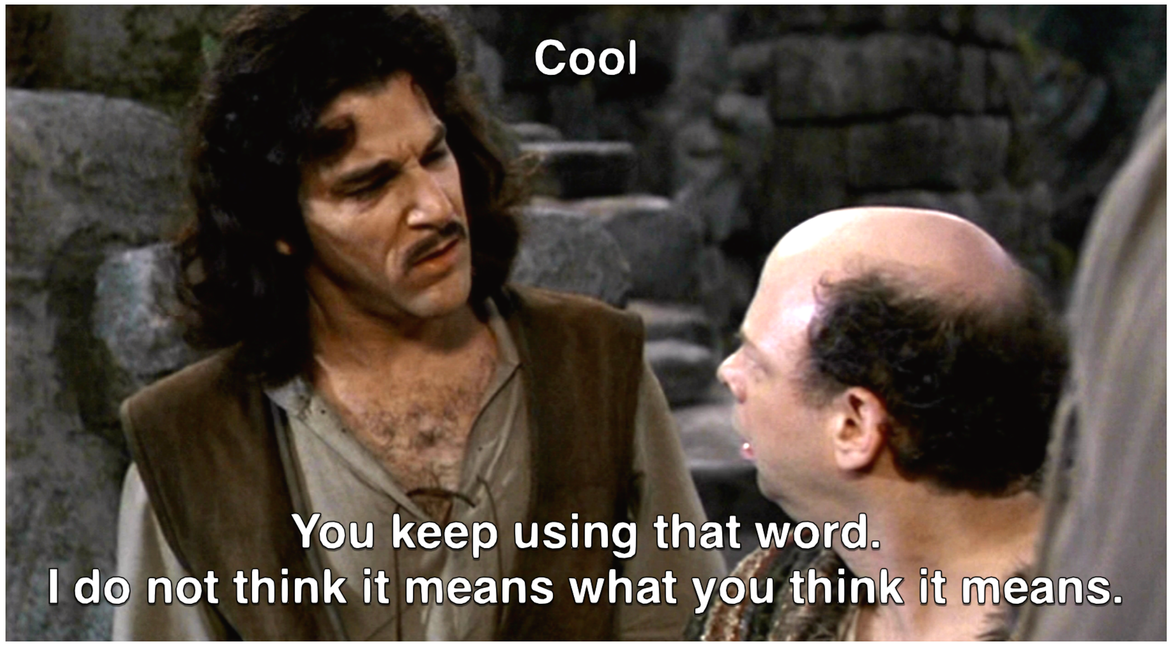
Measuring Success

- First 180 days of team use
  - Found 17 GPU driver bugs
  - Found 13 bugs in DXC
  - Found 26 bugs in Clang/LLVM
  - Found 4 bugs in SPIRV-Cross
  - Found 8 bugs in the Metal Shader Converter
- Learned some cool things!
  - Discovered a lot of un-specified behaviors

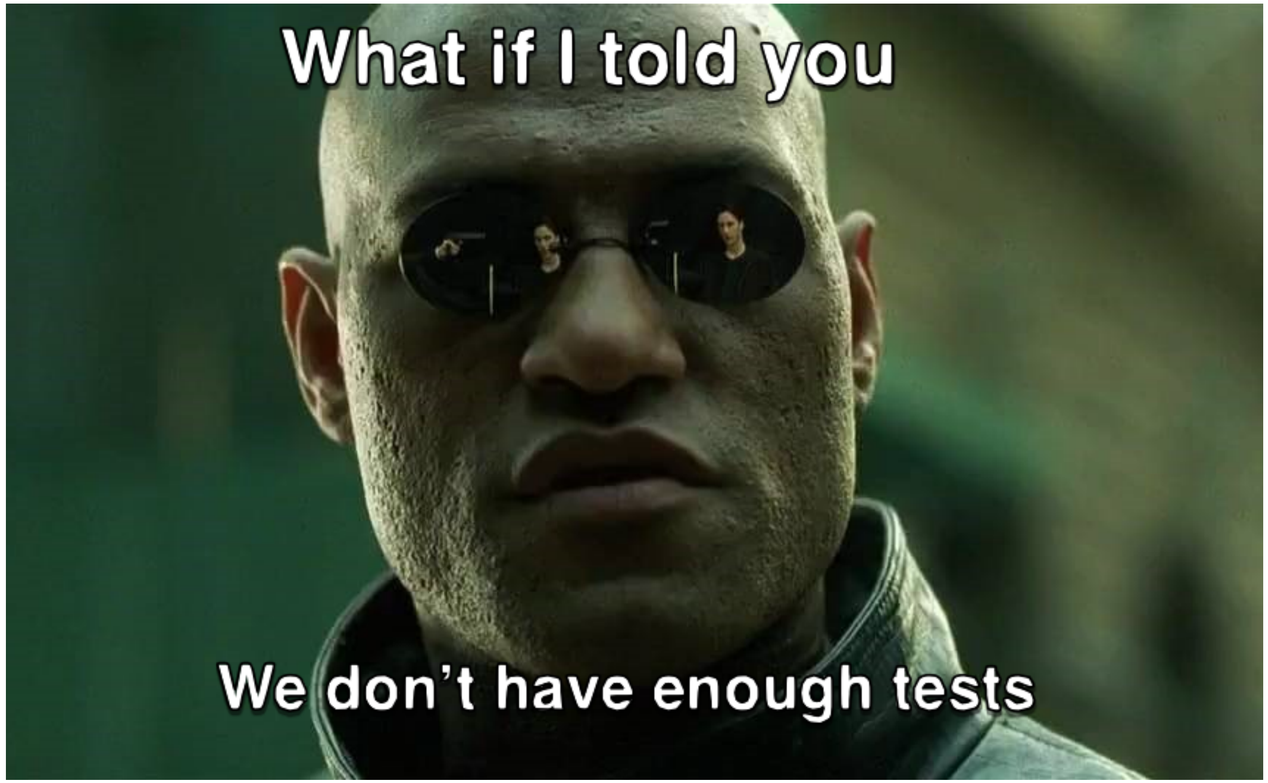


Cool things?!

- DXC failed to generate 16-bit float versions of some operations
  - Resulted in drivers not handling them
- GPU drivers don’t like treating resources as arrays of memory…
- tanh is _crazy_ imprecise on NV 50-series GPUs under DX but not VK
- It is _really_ hard to write portable GPU code…



Every bug we find is a testing gap!



Where simple doesn’t work

- Floating point behavior
  - Accuracy tolerances
  - NaN, INF, denorm
- Mismatched feature sets
- Mismatched UB sets
- Places where HLSL is just weird
- Complex scenarios

Key Takeaways

- It is amazing anyone ever ships GPU code
- More tests is good
  - Making it easier to write tests is always worthwhile
- Tests written by inexperienced engineers are extremely valuable
- We still have a lot of work to do
  - Bugs in tests, compilers, and drivers

Future Directions for HLSL

- Building out more API features
- Expanding our test matrix
- Using this framework to build a language conformance suite
- Pull these tests into Microsoft Hardware Lab Kit (HLK) test suite

Future Directions

- Compiler and runtime performance!
- Additional language and API support
  - I’ve written local tests for GLSL and MSL
  - Played with an OpenCL backend
- Cleanup lingering “HLSL-isms”

Get Involved

- Offload Test Suite
  - https://github.com/llvm/offload-test-suite/
- WG-HLSL
  - https://github.com/llvm/wg-hlsl
  - Find our weekly meetings on the LLVM community Calendar
- #hlsl on LLVM Discord
  - https://discord.gg/xS7Z362