Root Signatures & Shader ISA

Wessel de Groot

Principal Optimization Engineer

Nixxes

Intro

- What is a root signature?
- Descriptor tables
- Root descriptor
- Root constants
- Bindless

Disclaimer

- DirectX12
- AMD Shader ISA
  - RX 7900 XTX
  - I’m not an AMD representative
- Things may vary on other vendors

Crash Course

- Descriptor describes an object to the GPU
- Many different types
  - Shader Resource Views (SRVs)
    - Texture2D, TextureCube, Buffer, StructuredBuffer etc.
  - Unordered Access Views (UAVs)
    - RWTexture, RWBuffer, RWStructuredBuffer
  - Constant Buffer Views (CBVs)
  - Samplers
  - Render Targets
  - Depth Stencil
- Allocated from a descriptor heap

Crash Course

- Registers divided into scalar & vector
  - Scalar register holds the same value for each thread in a wave
    - s0, s1…
  - Vector register varies per thread in a wave
    - v0, v1…
- A scalar/vector register is 4 bytes in size
- Addressing multiple registers using s[#:#]
  - s[4:7]
- Memory loads are async

Root Signature

- User defined
- Links resources to the shader
  - Where to find them
  - How to access them
- Compiled into the shader
- Three different types
  - Descriptor Tables
  - Root Descriptors
  - Root Constants
- Different performance trade-offs

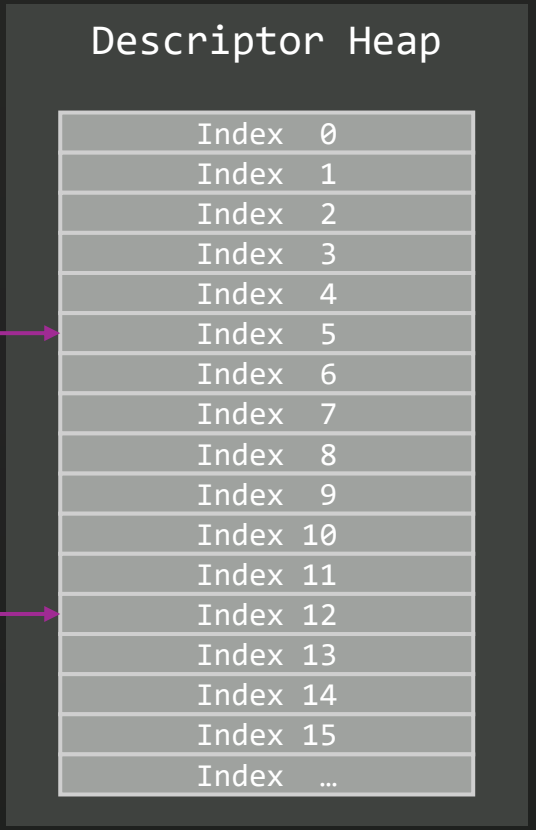
Descriptor Table

- An array of descriptors
- View on top of the bound descriptor heap
- Offset and length

Descriptor Table

Descriptor Table Start

Descriptor Table Size



Descriptor Table

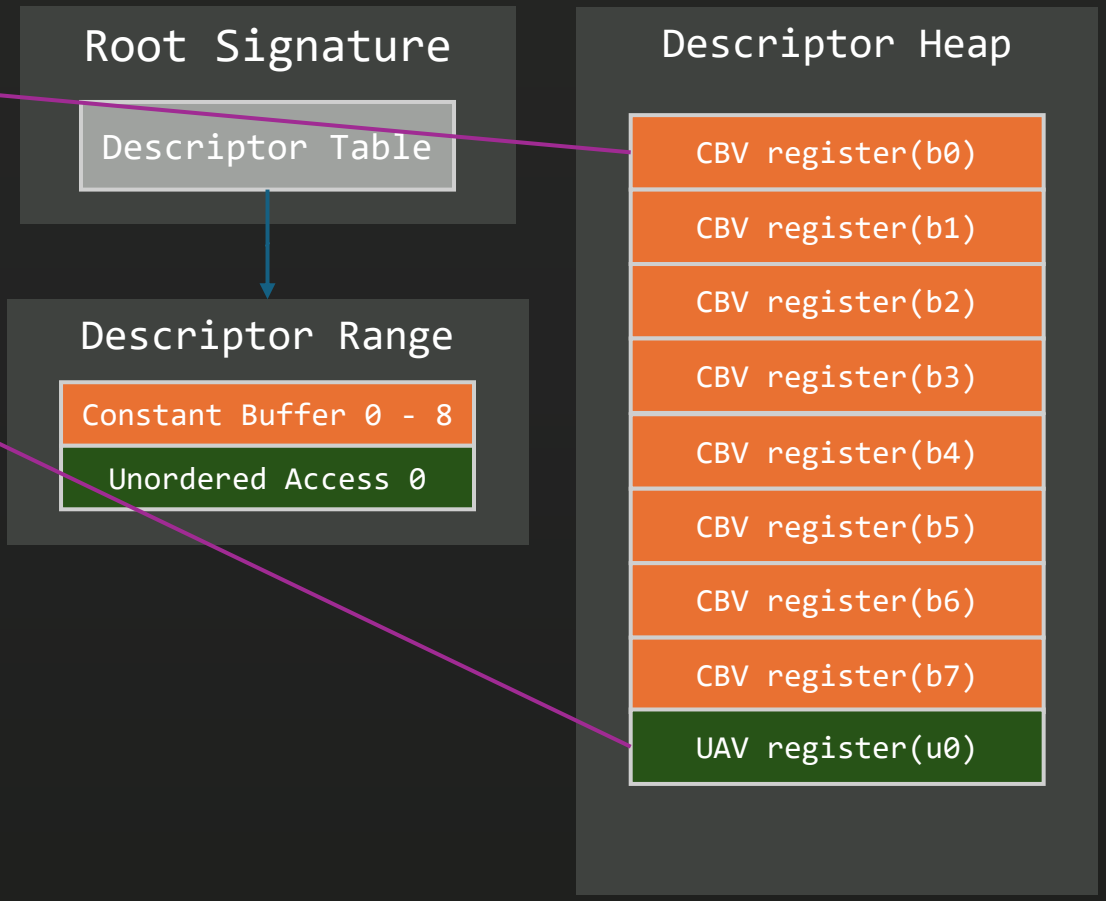
- One or more ranges per table
- Range per descriptor type
  - Constant buffer view
  - Shader resource view
  - Unordered access view
- Multiple tables per root signature

Shader

```
cbuffer input : register(b0)
{
    uint value;
};

RWStructuredBuffer<uint> output : register(u0);

[numthreads(32,1,1)]
void main(uint index : SV_DispatchThreadID)
{
    output[index] = value;
}
```

Descriptor Table

```
cbuffer input : register(b0)
{
    uint value;
};

RWStructuredBuffer<uint> output : register(u0);

[numthreads(32,1,1)]
void main(uint index : SV_DispatchThreadID)
{
    output[index] = value;
}
```



Descriptor Table

```
s_version                 UC_VERSION_GFX11 | UC_VERSION_W32_BIT
s_getpc_b64               s[4:5]
s_set_inst_prefetch_distance  0x0003
s_mov_b32                 s3, s5
s_load_b128               s[4:7], s[2:3], null
s_load_b128               s[0:3], s[2:3], 0x000100
s_waitcnt                 lgkmcnt(0)
s_buffer_load_b32         s4, s[4:7], null
v_and_b32                 v0, lit(0x000003ff), v0
v_lshl_add_u32            v0, s16, 5, v0
s_waitcnt                 lgkmcnt(0)
v_mov_b32                 v1, s4
buffer_store_b32          v1, v0, s[0:3], 0 idxen glc
s_nop                     0x0000
s_sendmsg                 sendmsg(MSG_DEALLOC_VGPRS)
s_endpgm
```

Descriptor Table

```
s_getpc_b64           s[4:5]
s_mov_b32             s3, s5
s_load_b128           s[4:7], s[2:3], null
s_load_b128           s[0:3], s[2:3], 0x000100
s_waitcnt             lgkmcnt(0)
s_buffer_load_b32     s4, s[4:7], null
v_and_b32             v0, lit(0x000003ff), v0
v_lshl_add_u32        v0, s16, 5, v0
s_waitcnt             lgkmcnt(0)
v_mov_b32             v1, s4
buffer_store_b32      v1, v0, s[0:3], 0 idxen glc
```

Descriptor Table

```
cbuffer input : register(b0)
{
    uint value;
};

RWStructuredBuffer<uint> output : register(u0);

[numthreads(32,1,1)]
void main(uint index : SV_DispatchThreadID)
{
    output[index] = value;
}
```

```
s_getpc_b64          s[4:5]
s_mov_b32            s3, s5
s_load_b128          s[4:7], s[2:3], null
s_load_b128          s[0:3], s[2:3], 0x000100
s_waitcnt            lgkmcnt(0)
s_buffer_load_b32    s4, s[4:7], null
v_and_b32            v0, lit(0x000003ff), v0
v_lshl_add_u32       v0, s16, 5, v0
s_waitcnt            lgkmcnt(0)
v_mov_b32            v1, s4
buffer_store_b32     v1, v0, s[0:3], 0 idxen glc
```

Descriptor Table

```
cbuffer input : register(b0)
{
    uint value;
};

RWStructuredBuffer<uint> output : register(u0);

[numthreads(32,1,1)]
void main(uint index : SV_DispatchThreadID)
{
    output[index] = value;
}
```

```
s_getpc_b64        s[4:5]
s_mov_b32          s3, s5
s_load_b128        s[4:7], s[2:3], null
s_load_b128        s[0:3], s[2:3], 0x000100
```

Descriptor Heap

0B	CBV register(b0)
32B	CBV register(b1)
64B	CBV register(b2)
96B	CBV register(b3)
128B	CBV register(b4)
160B	CBV register(b5)
192B	CBV register(b6)
224B	CBV register(b7)
256B	UAV register(u0)

Descriptor Table

```
cbuffer input : register(b0)
{
    uint value;
};

RWStructuredBuffer<uint> output : register(u0);

[numthreads(32,1,1)]
void main(uint index : SV_DispatchThreadID)
{
    output[index] = value;
}
```

```
s_getpc_b64          s[4:5]
s_mov_b32            s3, s5
s_load_b128          s[4:7], s[2:3], null
s_load_b128          s[0:3], s[2:3], 0x000100
s_waitcnt            lgkmcnt(0)
s_buffer_load_b32    s4, s[4:7], null
v_and_b32            v0, lit(0x000003ff), v0
v_lshl_add_u32       v0, s16, 5, v0
s_waitcnt            lgkmcnt(0)
v_mov_b32            v1, s4
buffer_store_b32     v1, v0, s[0:3], 0 idxen glc
```

Descriptor Table

```
cbuffer input : register(b0)
{
    uint value;
};

RWStructuredBuffer<uint> output : register(u0);

[numthreads(32,1,1)]
void main(uint index : SV_DispatchThreadID)
{
    output[index] = value;
}
```

```
s_getpc_b64          s[4:5]
s_mov_b32            s3, s5
s_load_b128          s[4:7], s[2:3], null
s_load_b128          s[0:3], s[2:3], 0x000100
s_waitcnt            lgkmcnt(0)
s_buffer_load_b32    s4, s[4:7], null
v_and_b32            v0, lit(0x000003ff), v0
v_lshl_add_u32       v0, s16, 5, v0
s_waitcnt            lgkmcnt(0)
v_mov_b32            v1, s4
buffer_store_b32     v1, v0, s[0:3], 0 idxen glc
```

Descriptor Table

```
cbuffer input : register(b0)
{
    uint value;
};

RWStructuredBuffer<uint> output : register(u0);

[numthreads(32,1,1)]
void main(uint index : SV_DispatchThreadID)
{
    output[index] = value;
}
```

```
s_getpc_b64          s[4:5]
s_mov_b32            s3, s5
s_load_b128          s[4:7], s[2:3], null
s_load_b128          s[0:3], s[2:3], 0x000100
s_waitcnt            lgkmcnt(0)
s_buffer_load_b32    s4, s[4:7], null
v_and_b32            v0, lit(0x000003ff), v0
v_lshl_add_u32       v0, s16, 5, v0
s_waitcnt            lgkmcnt(0)
v_mov_b32            v1, s4
buffer_store_b32     v1, v0, s[0:3], 0 idxen glc
```

Descriptor Table

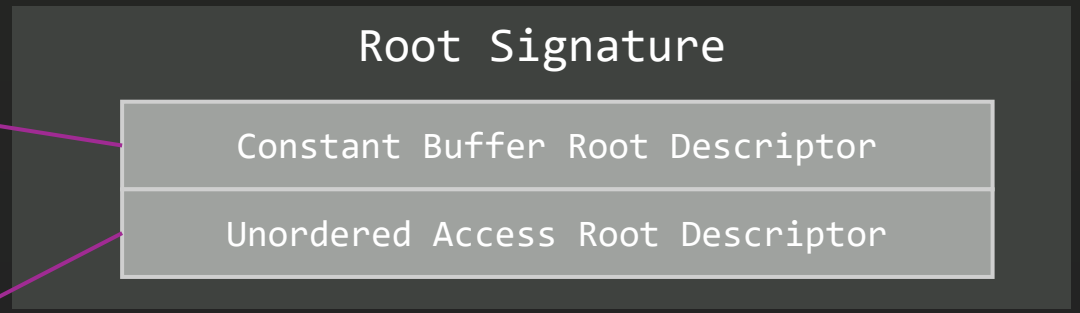
- A level of indirection
  - Read the descriptor from the descriptor heap
  - Use the actual descriptor

Root Descriptor

- Descriptor embedded into the root signature
- Kind of, bound using a gpu virtual address
- Limited to buffers
  - Constant buffers
  - Raw or structured uav / srv
- Defines one resource
- No out bounds checking!
  - We only have an address, no size

Root Descriptor

```
cbuffer input : register(b0)
{
    uint value;
};

RWStructuredBuffer<uint> output : register(u0);

[numthreads(32,1,1)]
void main(uint index : SV_DispatchThreadID)
{
    output[index] = value;
}
```



Root Descriptor

```
s_pack_ll_b32_b16  s0, s3, 16
s_mov_b32     s3, lit(0x2003dfac)
s_mov_b32     s1, s0
s_mov_b32     s0, s2
s_movk_i32    s2, 0x1000
v_and_b32     v0, lit(0x000003ff), v0
s_buffer_load_b32  s0, s[0:3], null
s_pack_ll_b32_b16  s1, s5, 4
s_movk_i32    s2, 0xffff
s_mov_b32     s3, lit(0x20014fac)
v_lshl_add_u32  v0, s16, 5, v0
s_waitcnt     lgkmcnt(0)
v_mov_b32     v1, s0
s_mov_b32     s0, s4
buffer_store_b32  v1, v0, s[0:3], 0 idxen glc
```

Root Descriptor

```
cbuffer input : register(b0)
{
    uint value;
};

RWStructuredBuffer<uint> output : register(u0);

[numthreads(32,1,1)]
void main(uint index : SV_DispatchThreadID)
{
    output[index] = value;
}
```

```
s_pack_ll_b32_b16  s0, s3, 16
s_mov_b32      s3, lit(0x2003dfac)
s_mov_b32      s1, s0
s_mov_b32      s0, s2
s_movk_i32     s2, 0x1000
v_and_b32      v0, lit(0x000003ff), v0
s_buffer_load_b32  s0, s[0:3], null
s_pack_ll_b32_b16  s1, s5, 4
s_movk_i32     s2, 0xffff
s_mov_b32      s3, lit(0x20014fac)
v_lshl_add_u32  v0, s16, 5, v0
s_waitcnt      lgkmcnt(0)
v_mov_b32      v1, s0
s_mov_b32      s0, s4
buffer_store_b32  v1, v0, s[0:3], 0 idxen glc
```

Root Descriptor

```
cbuffer input : r
{
    uint value;
};

RWStructuredBuffe

[numthreads(32,1,
void main(uint in
{
    output[index]
}
```

Table 47. Buffer Resource Descriptor

Bits	Size	Name	Description
47:0	48	Base address	Byte address.
61:48	14	Stride	Bytes 0 to 16383
63:62	2	swizzle Enable	Swizzle AOS according to stride, index_stride and element_size; otherwise linear. 0: disabled 1: enabled with element_size = 4byte 2: Reserved 3: enabled with element_size = 16byte
95:64	32	Num_records	In units of stride if (stride >=1), else in bytes.
98:96	3	Dst_sel_x	Destination channel select: 0=0, 1=1, 4=R, 5=G, 6=B, 7=A
101:99	3	Dst_sel_y	
104:102	3	Dst_sel_z	
107:105	3	Dst_sel_w	
113:108	6	Format	Memory data type.
118:117	2	Index stride	0:8, 1:16, 2:32, or 3:64. Used for swizzled buffer addressing.
119	1	Add tid enable	Add thread ID to the index for to calculate the address.
123:122	2	Reserved	Set to zero.
125:124	2	OOB_SELECT	Out of bounds select.
127:126	2	Type	Value == 0 for buffer. Overlaps upper two bits of four-bit TYPE field in 128-bit V# resource.

Root Descriptor

```
cbuffer input : register(b0)
{
    uint value;
};

RWStructuredBuffer<uint> output : register(u0);

[numthreads(32,1,1)]
void main(uint index : SV_DispatchThreadID)
{
    output[index] = value;
}
```

```
s_pack_ll_b32_b16  s0, s3, 16
s_mov_b32      s3, lit(0x2003dfac)
s_mov_b32      s1, s0
s_mov_b32      s0, s2
s_movk_i32     s2, 0x1000
s_buffer_load_b32  s0, s[0:3], null
```

Table 47. Buffer Resource Descriptor

Bits	Size	Name	Description
47:0	48	Base address	Byte address.
61:48	14	Stride	Bytes 0 to 16383
63:62	2	swizzle Enable	Swizzle AOS according to stride, index_ otherwise linear. 0: disabled 1: enabled with element_size = 4byte 2: Reserved 3: enabled with element_size = 16byte
95:64	32	Num_records	In units of stride if (stride >=1), else in b
98:96	3	Dst_sel_x	Destination channel select: 0=0, 1=1, 4=R, 5=G, 6=B, 7=A
101:99	3	Dst_sel_y	

Root Descriptor

```
cbuffer input : register(b0)
{
    uint value;
};

RWStructuredBuffer<uint> output : register(u0);

[numthreads(32,1,1)]
void main(uint index : SV_DispatchThreadID)
{
    output[index] = value;
}
```

```
s_pack_ll_b32_b16  s0, s3, 16
s_mov_b32      s3, lit(0x2003dfac)
s_mov_b32      s1, s0
s_mov_b32      s0, s2
s_movk_i32     s2, 0x1000
v_and_b32      v0, lit(0x000003ff), v0
s_buffer_load_b32  s0, s[0:3], null
s_pack_ll_b32_b16  s1, s5, 4
s_movk_i32     s2, 0xffff
s_mov_b32      s3, lit(0x20014fac)
v_lshl_add_u32  v0, s16, 5, v0
s_waitcnt      lgkmcnt(0)
v_mov_b32      v1, s0
s_mov_b32      s0, s4
buffer_store_b32  v1, v0, s[0:3], 0 idxen glc
```

Root Descriptor

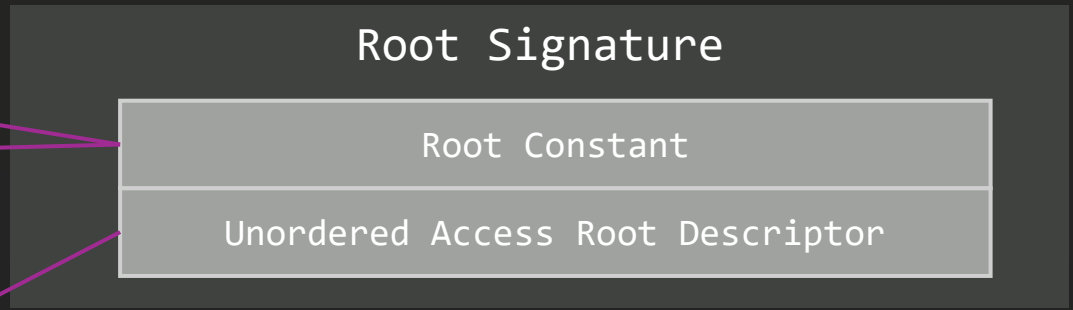
- No level of indirection
  - Only one wait for memory instead of two
- Descriptor composed using ALU
- Only the memory addressed passed to the shader
- Limit to how much can go into registers
- Static samplers can behave in the same way

Root Constant

- 32 bit value embedded into the root signature
- Appears as a constant buffer to the shader
  - Constant buffer padding still applies!
- No bounds checking either

Root Constant

```
cbuffer input : register(b0)
{
    uint value;
};

RWStructuredBuffer<uint> output : register(u0);

[numthreads(32,1,1)]
void main(uint index : SV_DispatchThreadID)
{
    output[index] = value;
}
```



Root Constant

```
v_dual_mov_b32  v1, s2
v_dual_and_b32  v0, lit(0x000003ff), v0
v_lshl_add_u32  v0, s16, 5, v0
s_pack_ll_b32_b16  s0, s4, 4
s_mov_b32       s1, s0
s_movk_i32      s2, 0xffff
s_mov_b32       s0, s3
s_mov_b32       s3, lit(0x20014fac)
buffer_store_b32  v1, v0, s[0:3], 0 idxen glc
```

Root Constant

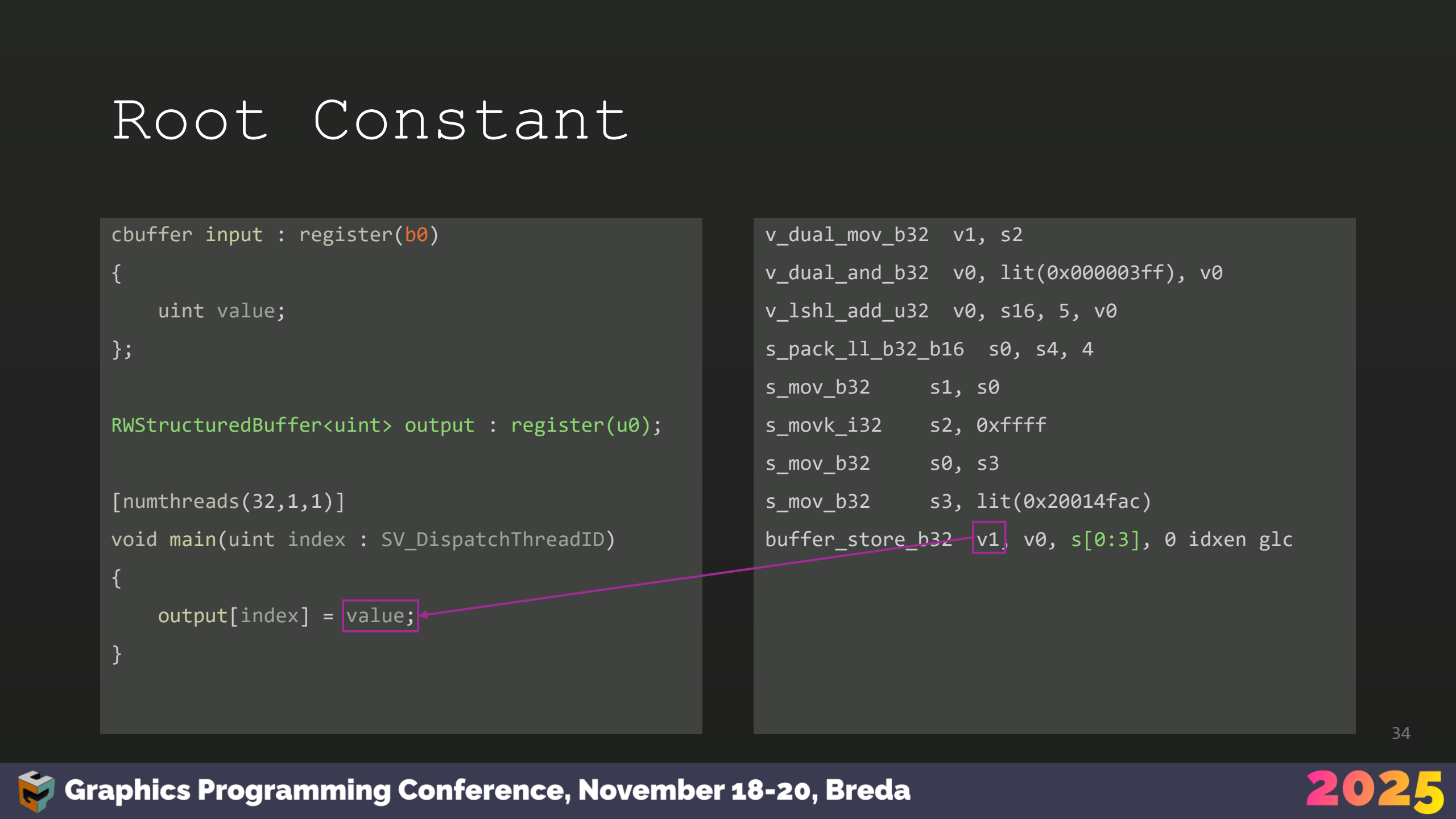
```
cbuffer input : register(b0)
{
    uint value;
};

RWStructuredBuffer<uint> output : register(u0);

[numthreads(32,1,1)]
void main(uint index : SV_DispatchThreadID)
{
    output[index] = value;
}
```

```
v_dual_mov_b32  v1, s2
v_dual_and_b32  v0, lit(0x000003ff), v0
v_lshl_add_u32  v0, s16, 5, v0
s_pack_ll_b32_b16  s0, s4, 4
s_mov_b32     s1, s0
s_movk_i32    s2, 0xffff
s_mov_b32     s0, s3
s_mov_b32     s3, lit(0x20014fac)
buffer_store_b32  v1, v0, s[0:3], 0 idxen glc
```

Root Constant

```
cbuffer input : register(b0)
{
    uint value;
};

RWStructuredBuffer<uint> output : register(u0);

[numthreads(32,1,1)]
void main(uint index : SV_DispatchThreadID)
{
    output[index] = value;
}
```

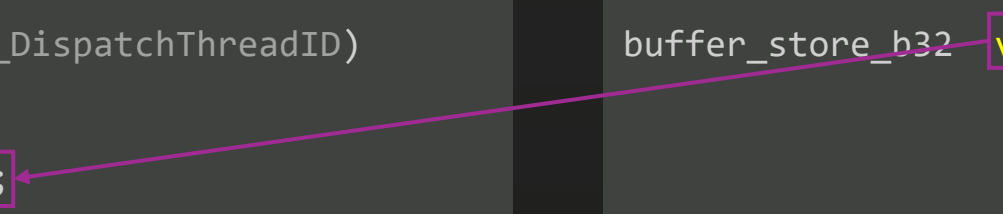
```
v_dual_mov_b32  v1, s2
v_dual_and_b32  v0, lit(0x000003ff), v0
v_lshl_add_u32  v0, s16, 5, v0
s_pack_ll_b32_b16  s0, s4, 4
s_mov_b32       s1, s0
s_movk_i32      s2, 0xffff
s_mov_b32       s0, s3
s_mov_b32       s3, lit(0x20014fac)
buffer_store_b32  v1, v0, s[0:3], 0 idxen glc
```



Root Constant

```
cbuffer input : register(b0)
{
    uint value;
};

RWStructuredBuffer<uint> output : register(u0);

[numthreads(32,1,1)]
void main(uint index : SV_DispatchThreadID)
{
    output[index] = value;
}
```

```
v_dual_mov_b32  v1, s2
v_dual_and_b32  v0, lit(0x000003ff), v0
v_lshl_add_u32  v0, s16, 5, v0
s_pack_ll_b32_b16  s0, s4, 4
s_mov_b32       s1, s0
s_movk_i32      s2, 0xffff
s_mov_b32       s0, s3
s_mov_b32       s3, lit(0x20014fac)
buffer_store_b32  v1, v0, s[0:3], 0 idxen glc
```



Root Constant

- Initialized before launching the shader
- No buffer load needed
- Limit to how much can go into registers

Bindless

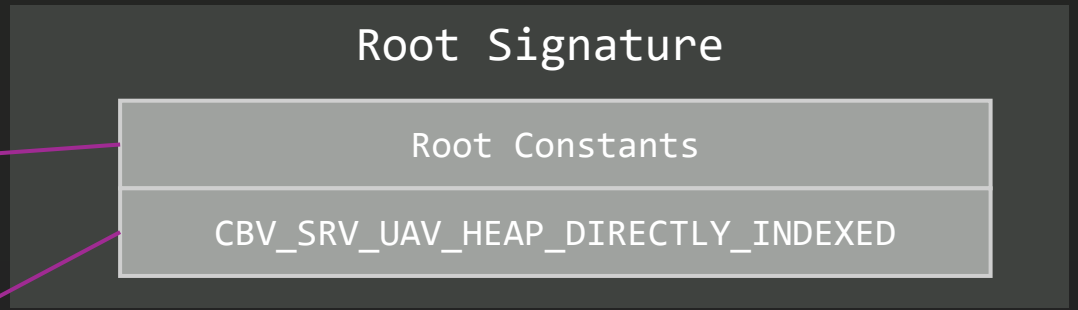
- Dynamically index the bound descriptor heap
- No mapping required, it’s just a flag
  - D3D12_ROOT_SIGNATURE_FLAG_CBV_SRV_UAV_HEAP_DIRECTLY_INDEXED
  - D3D12_ROOT_SIGNATURE_FLAG_SAMPLER_HEAP_DIRECTLY_INDEXED

Bindless

```
cbuffer input : register(b0)
{
  uint value;
  uint heap_index;
};

[numthreads(32,1,1)]
void main(uint index : SV_DispatchThreadID)
{
  RWStructuredBuffer<uint> output = ResourceDescriptorHeap[heap_index];
  output[index] = value;
}
```

Bindless

```
cbuffer input : register(b0)
{
  uint value;
  uint heap_index;
};

[numthreads(32,1,1)]
void main(uint index : SV_DispatchThreadID)
{
  RWStructuredBuffer<uint> output =
        ResourceDescriptorHeap[heap_index];
  output[index] = value;
}
```



Bindless

```
s_getpc_b64    s[10:11]
s_mov_b32      s8, s2
s_mov_b32      s9, s11
s_lshl_b32     s0, s4, 5
v_and_b32      v0, lit(0x000003ff), v0
s_load_b128    s[8:11], s[8:9], s0
v_lshl_add_u32  v0, s7, 6, v0
v_mov_b32      v1, s3
s_waitcnt      lgkmcnt(0)
buffer_store_b32  v1, v0, s[8:11], 0 idxen glc
```

Bindless

```
cbuffer input : register(b0)
{
  uint value;
  uint heap_index;
};

[numthreads(32,1,1)]
void main(uint index : SV_DispatchThreadID)
{
  RWStructuredBuffer<uint> output =
        ResourceDescriptorHeap[heap_index];
  output[index] = value;
}
```

```
s_getpc_b64    s[10:11]
s_mov_b32      s8, s2
s_mov_b32      s9, s11
s_lshl_b32     s0, s4, 5
v_and_b32      v0, lit(0x000003ff), v0
s_load_b128    s[8:11], s[8:9], s0
v_lshl_add_u32  v0, s7, 6, v0
v_mov_b32      v1, s3
s_waitcnt      lgkmcnt(0)
buffer_store_b32  v1, v0, s[8:11], 0 idxen glc
```

Bindless

```
cbuffer input : register(b0)
{
  uint value;
  uint heap_index;
};

[numthreads(32,1,1)]
void main(uint index : SV_DispatchThreadID)
{
  RWStructuredBuffer<uint> output =
        ResourceDescriptorHeap[heap_index];
  output[index] = value;
}
```

```
s_getpc_b64    s[10:11]
s_mov_b32      s8, s2
s_mov_b32      s9, s11
s_lshl_b32     s0, s4, 5
v_and_b32      v0, lit(0x000003ff), v0
s_load_b128    s[8:11], s[8:9], s0
v_lshl_add_u32  v0, s7, 6, v0
v_mov_b32      v1, s3
s_waitcnt      lgkmcnt(0)
buffer_store_b32  v1, v0, s[8:11], 0 idxen glc
```

Bindless

```
cbuffer input : register(b0)
{
  uint value;
  uint heap_index;
};

[numthreads(32,1,1)]
void main(uint index : SV_DispatchThreadID)
{
  RWStructuredBuffer<uint> output =
        ResourceDescriptorHeap[heap_index];
  output[index] = value;
}
```

```
s_getpc_b64    s[10:11]
s_mov_b32      s8, s2
s_mov_b32      s9, s11
s_lshl_b32     s0, s4, 5
v_and_b32      v0, lit(0x000003ff), v0
s_load_b128    s[8:11], s[8:9], s0
v_lshl_add_u32  v0, s7, 6, v0
v_mov_b32      v1, s3
s_waitcnt      lgkmcnt(0)
buffer_store_b32  v1, v0, s[8:11], 0 idxen glc
```

Bindless

```
cbuffer input : register(b0)
{
  uint value;
  uint heap_index;
};

[numthreads(32,1,1)]
void main(uint index : SV_DispatchThreadID)
{
  RWStructuredBuffer<uint> output =
        ResourceDescriptorHeap[heap_index];
  output[index] = value;
}
```

```
s_getpc_b64    s[10:11]
s_mov_b32      s8, s2
s_mov_b32      s9, s11
s_lshl_b32     s0, s4, 5
v_and_b32      v0, lit(0x000003ff), v0
s_load_b128    s[8:11], s[8:9], s0
v_lshl_add_u32  v0, s7, 6, v0
v_mov_b32      v1, s3
s_waitcnt      lgkmcnt(0)
buffer_store_b32  v1, v0, s[8:11], 0 idxen glc
```

Bindless

```
cbuffer input : register(b0)
{
  uint value;
  uint heap_index;
};

[numthreads(32,1,1)]
void main(uint index : SV_DispatchThreadID)
{
  RWStructuredBuffer<uint> output =
        ResourceDescriptorHeap[heap_index];
  output[index] = value;
}
```

```
s_getpc_b64    s[10:11]
s_mov_b32      s8, s2
s_mov_b32      s9, s11
s_lshl_b32     s0, s4, 5
v_and_b32      v0, lit(0x000003ff), v0
s_load_b128    s[8:11], s[8:9], s0
v_lshl_add_u32  v0, s7, 6, v0
v_mov_b32      v1, s3
s_waitcnt      lgkmcnt(0)
buffer_store_b32  v1, v0, s[8:11], 0 idxen glc
```

Bindless

- Behaves similarly as a descriptor table
- But without the added complexity

Summary

	Descriptor Table	Root Descriptor	Root Constant	Bindless	Static Samplers

Summary

	Descriptor Table	Root Descriptor	Root Constant	Bindless	Static Samplers
Resource Types	All	Untyped Buffers	Constant Buffers	All	Samplers*

Summary

	Descriptor Table	Root Descriptor	Root Constant	Bindless	Static Samplers
Resource Types	All	Untyped Buffers	Constant Buffers	All	Samplers
# of resources	1000000+ srv, cbv & uav 2048+ samplers	1 srv, cbv or uav	1 32-bit constant	1000000+ srv, cbv & uav 2048+ samplers	1

Summary

	Descriptor Table	Root Descriptor	Root Constant	Bindless	Static Samplers
Resource Types	All	Untyped Buffers	Constant Buffers	All	Samplers
# of resources	1000000+ srv, cbv & uav 2048+ samplers	1 srv, cbv or uav	1 32-bit constant	1000000+ srv, cbv & uav 2048+ samplers	1
Bounds Checking	Yes	No	No	Yes*	-

Summary

	Descriptor Table	Root Descriptor	Root Constant	Bindless	Static Samplers
Resource Types	All	Untyped Buffers	Constant Buffers	All	Samplers
# of resources	1000000+ srv, cbv & uav 2048+ samplers	1 srv, cbv or uav	1 32-bit constant	1000000+ srv, cbv & uav 2048+ samplers	1
Bounds Checking	Yes	No	No	Yes*	-
Trips to memory*	2	1	0	2	0

Summary

	Descriptor Table	Root Descriptor	Root Constant	Bindless	Static Samplers
Resource Types	All	Untyped Buffers	Constant Buffers	All	Samplers
# of resources	1000000+ srv, cbv & uav 2048+ samplers	1 srv, cbv or uav	1 32-bit constant	1000000+ srv, cbv & uav 2048+ samplers	1
Bounds Checking	Yes	No	No	Yes*	-
Trips to memory*	2	1	0	2	0
Cpu Setup	Complex	Simple	Simple	Simple	Simple

Summary

	Descriptor Table	Root Descriptor	Root Constant	Bindless	Static Samplers
Resource Types	All	Untyped Buffers	Constant Buffers	All	Samplers
# of resources	1000000+ srv, cbv & uav 2048+ samplers	1 srv, cbv or uav	1 32-bit constant	1000000+ srv, cbv & uav 2048+ samplers	1
Bounds Checking	Yes	No	No	Yes*	-
Trips to memory*	2	1	0	2	0
Cpu Setup	Complex	Simple	Simple	Simple	Simple
Root Signature Size	1 dword	2 dwords	1 dword	-	1 sampler slot

Questions?

Bonus

- `ID3D12PipelineState::GetPrivateData(WKPDID_CommentStringW, ...)`
  - Returns a string of the ISA of your pipeline on AMD

References

1. https://www.amd.com/content/dam/amd/en/documents/radeon-tech-docs/instruction-set-architectures/rdna3-shader-instruction-set-architecture-feb-2023_0.pdf
2. https://microsoft.github.io/DirectX-Specs/d3d/ResourceBinding.html
3. https://www.youtube.com/watch?v=Wbnw87tYqVg