DirectX™ 12 Case Studies

Holger Gruen Senior DevTech Engineer, 3/1/2017

Agenda

- Introduction
- DX12 in The Division from Massive Entertainment
- DX12 in Anvil Next Engine from Ubisoft
- DX12 in Hitman from IO Interactive
- DX12 in 'Game AAA'
- AfterMath Preview
- Nsight VSE & DirectX12 Games
- Q&A

Agenda

- Introduction
- DX12 in The Division from Massive Entertainment
- DX12 in Anvil Next Engine from Ubisoft
- DX12 in Hitman from IO Interactive
- DX12 in 'Game AAA'
- AfterMath Preview
- Nsight VSE & DirectX12 Games
- Q&A

Introduction

- DirectX 12 is here to stay
  - Games do now support DX12 & many engines are transitioning to DX12
- DirectX 12 makes 3D programming more complex
  - see DX12 Do's & Don'ts in developer section on NVIDIA.com
- Goal for this talk is to …
  - Hear what talented developers have done to cope with DX12
  - See what developers want to share when asked to describe their DX12 story
  - Gain insights for your own DX11 to DX12 transition

Thanks & Credits

- **Carl Johan Lejdfors** *Technical Director* &
  Daniel Wesslen *Render Architect* - **Massive**
- **Jonas Meyer** *Lead Render Programmer* **&**
  Anders Wang Kristensen *Render Programmer* - **Io-Interactive**
- **Tiago Rodrigues** 3D Programmer - **Ubisoft Montreal**

Before we really start …

- Things we'll be hearing about a lot
  - Memory Managment
  - Barriers
  - Pipeline State Objects
  - Root Signature and Shader Bindings
  - Multiple Queues
  - Multi threading

If you get a chance check out the DX12 presentation from Monday's 'The Advanced Graphics Techniques tutorial'

Agenda

- Introduction
- DX12 in The Division from Massive Entertainment
- DX12 in Anvil Next Engine from Ubisoft
- DX12 in Hitman from IO Interactive
- DX12 in 'Game AAA'
- AfterMath Preview
- Nsight VSE & DirectX12 Games
- Q&A

TOM CLANCY'S THE DIVISION from MASSIVE

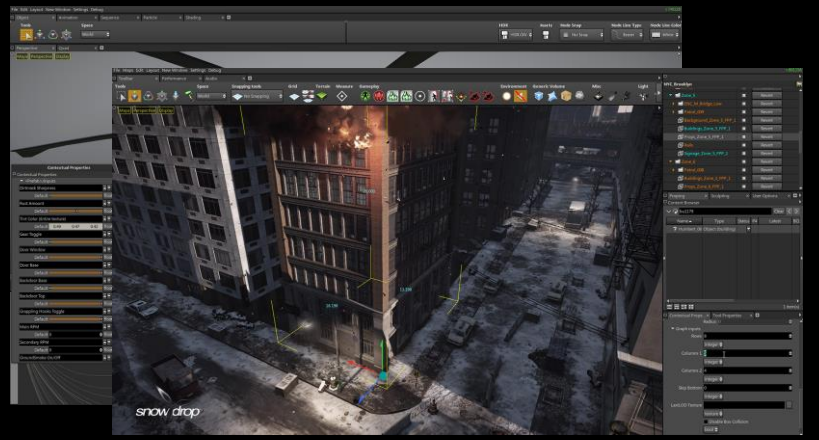
MASSIVE ENTERTAINMENT | A UBISOFT STUDIO

- **Snowdrop Engine** snow drop
  - Developed in-house to support The Division
  - Scalable & multi-threaded
  - Has a strong focus on great performance and fast iteration times

- **Tom Clancy's The Division**
  - An always online, coop game in a modern day setting

TOM CLANCY'S THE DIVISION from MASSIVE ENTERTAINMENT | A UBISOFT STUDIO

- **Snowdrop Engine** snow drop
  - Developed in-house to support The Division
  - Scalable & multi-threaded
  - Has a strong focus on great performance and fast iteration times

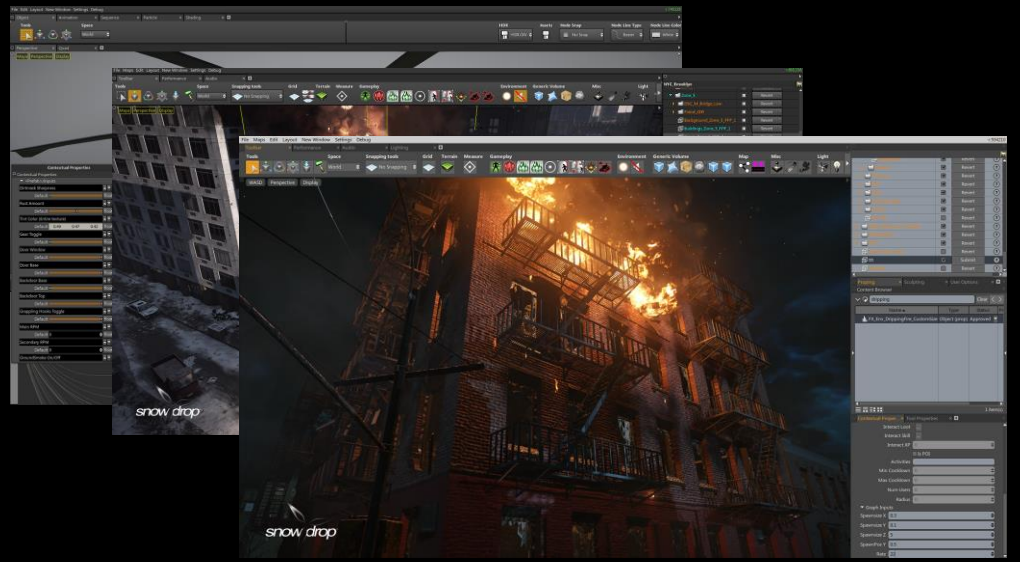


- **Tom Clancy's The Division**
  - An always online, coop game in a modern day setting

TOM CLANCY'S THE DIVISION from MASSIVE

MASSIVE ENTERTAINMENT | A UBISOFT STUDIO

- **Snowdrop Engine** snow drop
  - Developed in-house to support The Division
  - Scalable & multi-threaded
  - Has a strong focus on great performance and fast iteration times
- **Tom Clancy's The Division**
  - An always online, coop game in a modern day setting

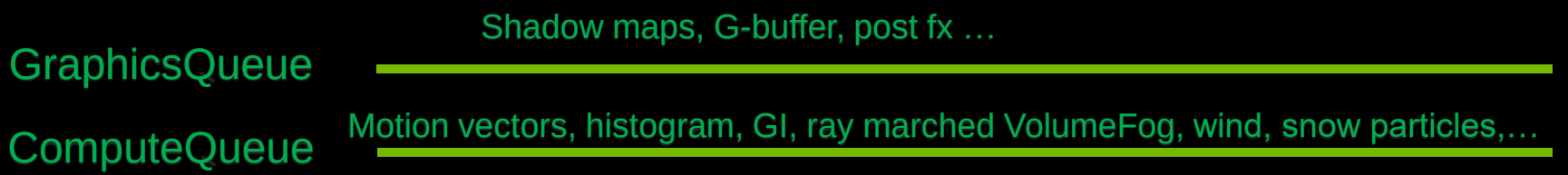


The Division DX12 - Agenda

- Asynchronous Queues
- Memory Management
- Pipeline State Objects
- Shader Model 5.1 Resource Binding
- Multi threading
- Miscellaneous

The Division DX12 : Asynchronous Queues

- Compute Queue
  - Nice cross vendor speedup
    - On average 5%
    - Asynchronous workload mostly resolution-independent (tuned for 1080p)
    - Diminishing returns as resolution increases



The Division DX12 : Asynchronous Queues

- The engine uses 3 Copy Queues

CopyQueue1	High frequency copies from upload to default heap
CopyQueue2	Asynchronous streaming of mip map data
CopyQueue3	Data initialization during resource creation

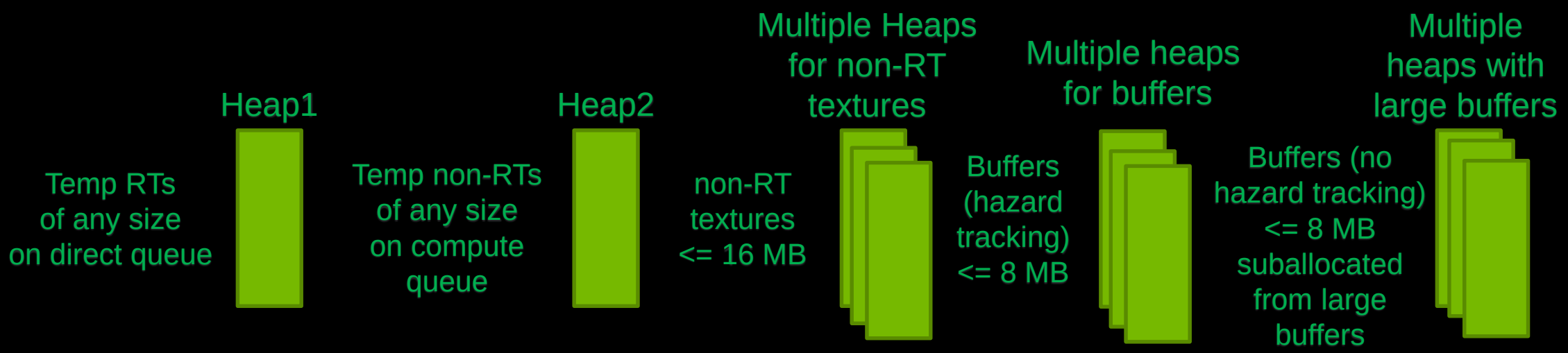
- Multiple copy queues ease engine thread synchronization

The Division DX12 : Memory Managment

- After DX12 bring-up, MemoryManagment improvements increased performance most
- One type of CB accidentally allocated as committed resource
  - Caused memory fragmentation => intermittent stuttering
  - Sub-allocation from a larger heap (as intended) improved performance by ~15%
  - Used GPUView to detect this
    - Not overcommitted on memory
    - Saw a huge amount of copies from outside the game process

The Division DX12 : Memory Managment

- Tuning which resources to allocate from which heap added >2% perf
  - Included adjusting rules for where to allocate placed resources



The Division DX12 : Memory Managment

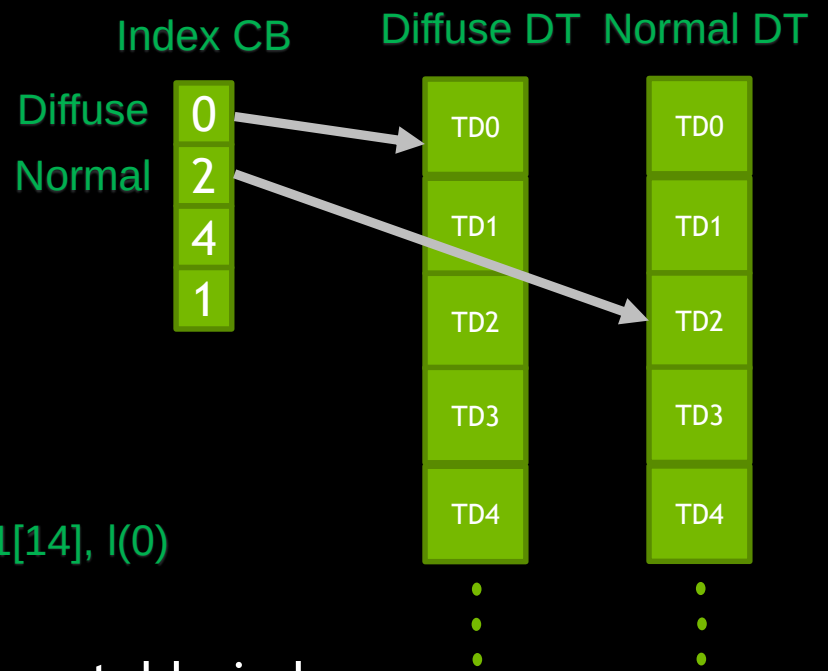
- Typically frequently-updated-and-rarely-read buffers are placed on the upload heap
  - Constant Buffers
  - Dynamic VBs, …
- Turns out this strategy is not optimal for The Division
  - Copying data from the an upload to a default heap generates a nice speedup > 1%

The Division DX12 : Pipeline State Objects

- Luckily “Shader State Object” concept was already used by the engine
  - Mapped nicely to DX12 PSO after some small extension work
  - Most PSOs get pre-created when the game starts and during streaming
  - Support to skip rendering objects for ‘missing’ PSOs
- Ended up restructuring mesh queuing and rendering to reduce # of PSO changes
  - Unifying all equivalent artists created render states
  - Saved >= ~15% CPU time (Better batching)

The Division DX12 : SM 5.1 Resource Binding

- Unbounded descriptor tables simplify many things
  - Allows to store all local material indices in one CB
  - Makes mesh rendering very efficient on the CPU
  - Only VB,IB and a single root CBV changed per PSO change




```
sample_l r5.w, r7.xyxx, t3[r5.w].yzwx, s1[14], l(0)
```

- During streaming updated textures are placed at the same table index
  - Descriptor heap with texture descriptors triple buffered to prevent race conditions

The Division DX12 : Multi threading

- DX12 finally allows multi-threaded submission and recording
  - One thread per queue type (Direct, Compute, Copy)
    - recording more complex command lists and submitting work
  - Command list recording runs on all cores
    - 43 tasks – able to run on as many threads

The Division DX12 : Miscellaneous

- SM 5.1 and /all_resources_bound Shader compiler flag improve perf by ~1.0-1.5%
  - No change in shader code necessary
  - Enables less conservative code generation for texture accesses
  - Not new: check https://blogs.msdn.microsoft.com/marcelolr/2016/08/19/understanding-all_resources_bound-in-hlsl/

```
Texture2D<float4> t0;
uint count;

float4 main() : SV_Target {
  float4 result = 0;
  for (uint i = 0; i < count; ++i) {
    result += t0[float2(0,0)];
  }
  return result;
}
```

w/o /all_resources_bound

```
mov r0.xyzw, l(0,0,0,0)
mov r1.x, l(0)
loop
  uge r1.y, r1.x, CB0[0][0].x
  breakc_nz r1.y
  ld r2.xyzw, l(0, 0, 0, 0), T0[0].xyzw
  add r0.xyzw, r0.xyzw, r2.xyzw
  iadd r1.x, r1.x, l(1)
endloop
mov o0.xyzw, r0.xyzw
```

with /all_resources_bound

```
ld r0.xyzw, l(0, 0, 0, 0), T0[0].xyzw
mov r1.xyzw, l(0,0,0,0)
mov r2.x, l(0)
loop
  uge r2.y, r2.x, CB0[0][0].x
  breakc_nz r2.y
  add r1.xyzw, r0.xyzw, r1.xyzw
  iadd r2.x, r2.x, l(1)
endloop
mov o0.xyzw, r1.xyzw
```

Code snippets from: https://blogs.msdn.microsoft.com/marcelolr/2016/08/19/understanding-all_resources_bound-in-hlsl/

Agenda

- Introduction
- DX12 in The Division from Massive Entertainment
- DX12 in Anvil Next Engine from Ubisoft
- DX12 in Hitman from IO Interactive
- DX12 in 'Game AAA'
- AfterMath Preview
- Nsight VSE & DirectX12 Games
- Q&A

Anvil Next Engine from Ubisoft

- Used in Assassin's creed series
- Initial ‘naïve’ port revealed a number of performance issues
  - Inefficient Barriers
  - Hitching on PSO creation
  - Memory over-commitment

This is a condensed version of **Tiago Rodrigues** talk “Moving to DirectX 12: Lessons Learned” – check out the full version!

Anvil Next Engine from Ubisoft

- Re-designed to get the best out of DX12

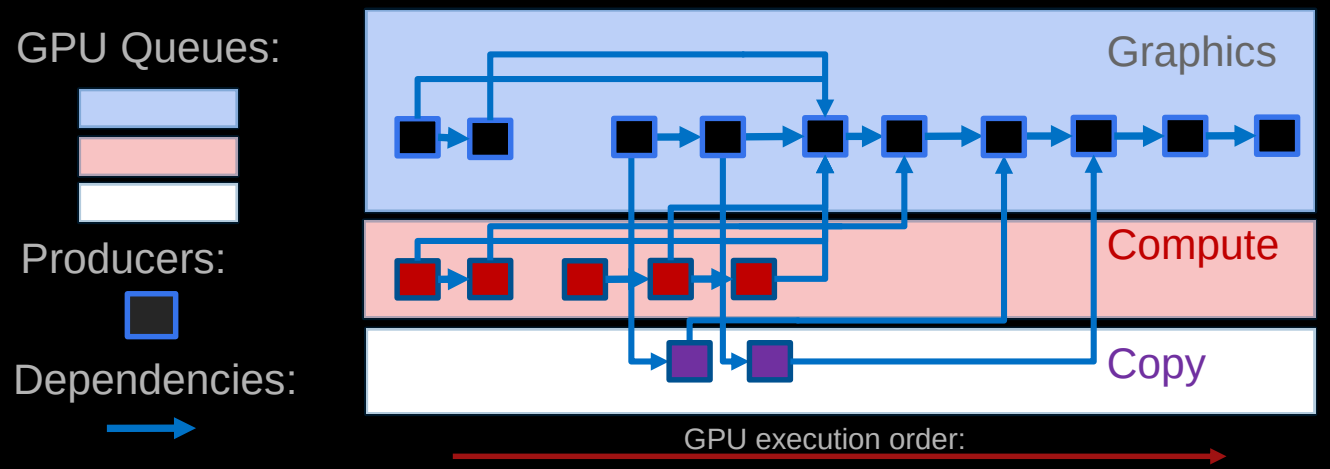
1. Minimize and batch resource barriers
2. Take full advantage of parallel CMD list recording
3. Use precompiled render state to minimize runtime work
4. Minimize memory footprint
5. Make use of the several GPU queues

Anvil Next Engine - Agenda

- Automatic Resource Tracking
- Barriers
- Shader Bindings
- Pipeline State

Anvil Next Engine – Resource Tracking

- Engine code explicitly defines a dependency graph
  - Each pass/producer defines which resources are needed and in which state

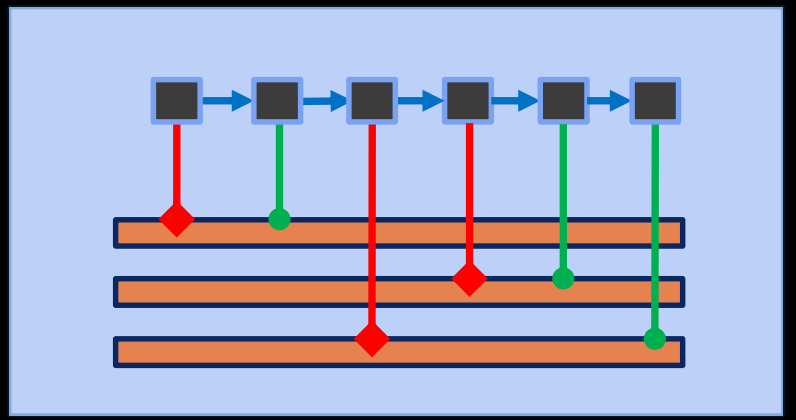


Anvil Next Engine – Resource Tracking

- Engine tracks resource dependencies automatically
  - Analyzes graph dependency graph between GPU producers & consumers

: producer

: resource



Explicit dependencies:

: resource write

: resource read

Derived dependencies:

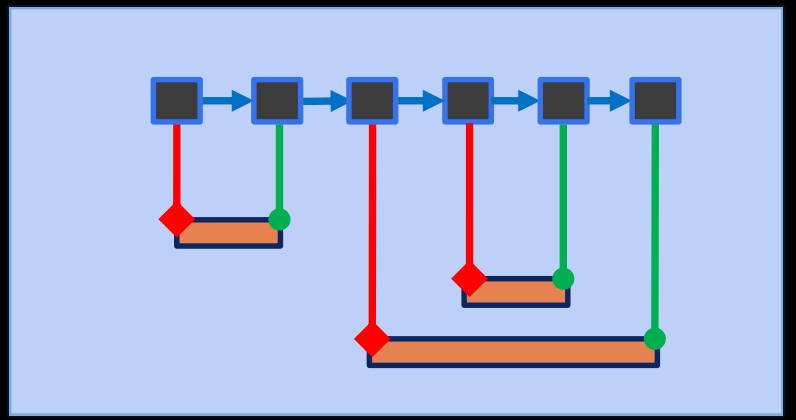
: execution order

Anvil Next Engine – Resource Tracking

- Engine tracks resource life times automatically

: producer

: resource



Explicit dependencies:

: resource write

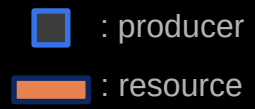
: resource read

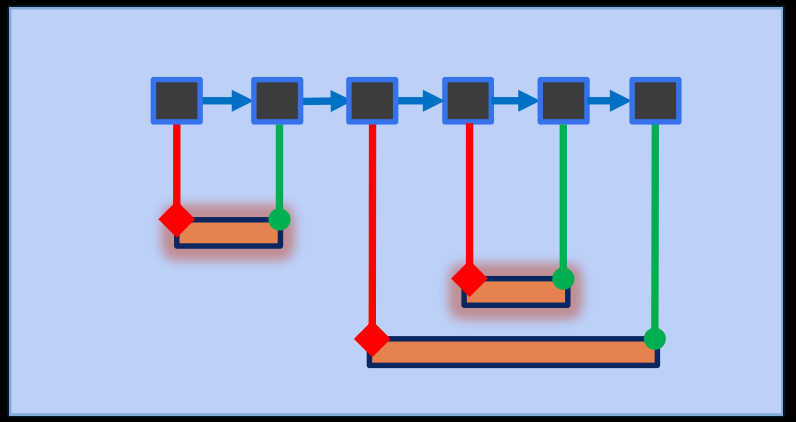
Derived dependencies:

: execution order

Anvil Next Engine – Resource Tracking

- Engine tracks resource life times automatically
- Engine uses life times to determine options for memory reuse (placed resources)





Explicit dependencies:

: resource write

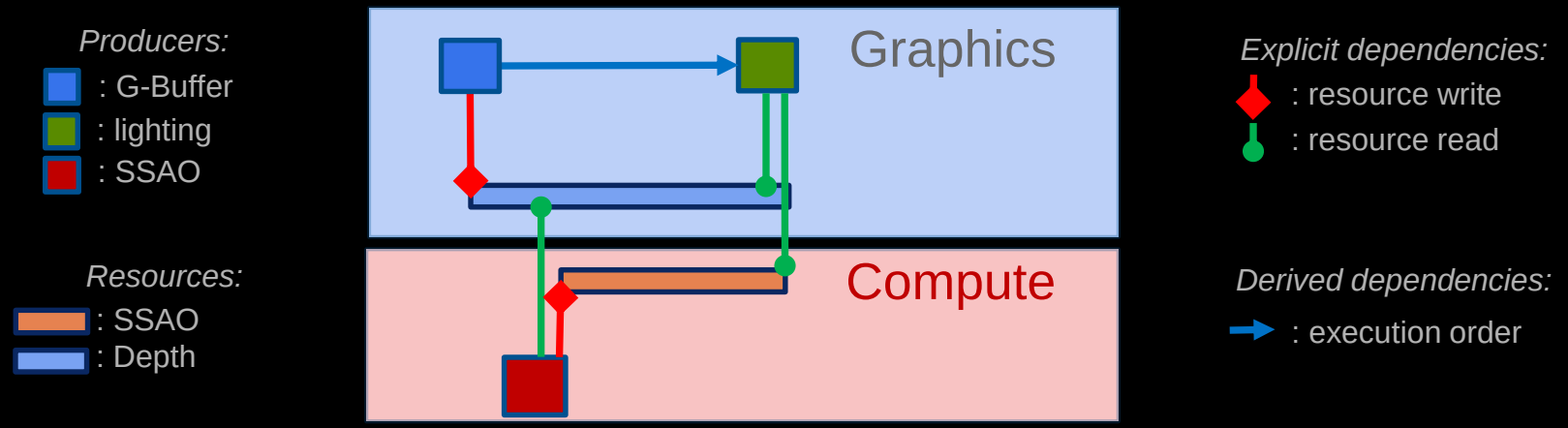
: resource read

Derived dependencies:

: execution order

Anvil Next Engine – Resource Tracking

- Engine tracks resource access synchronization automatically
  - SSAO buffer produced in compute, consumed in GFX queue

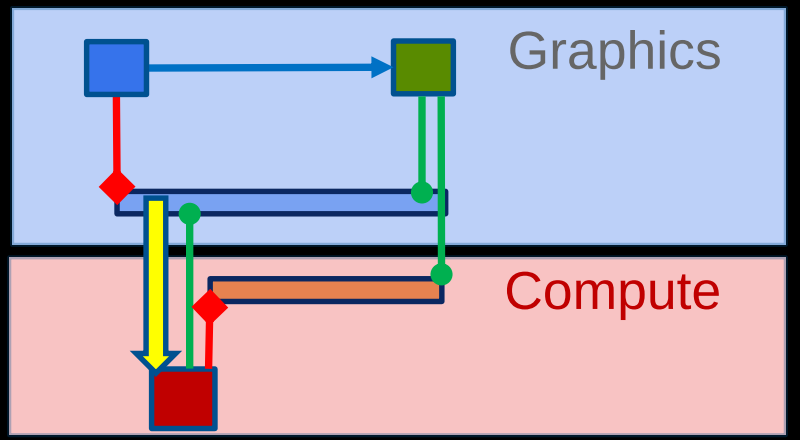


Anvil Next Engine – Resource Tracking

- Engine tracks resource access synchronization automatically
  - SSAO on compute queue must wait for G-Buffer rendering to finish

Producers:
: G-Buffer
: lighting
: SSAO

Resources:
: SSAO
: Depth



Explicit dependencies:
: resource write
: resource read

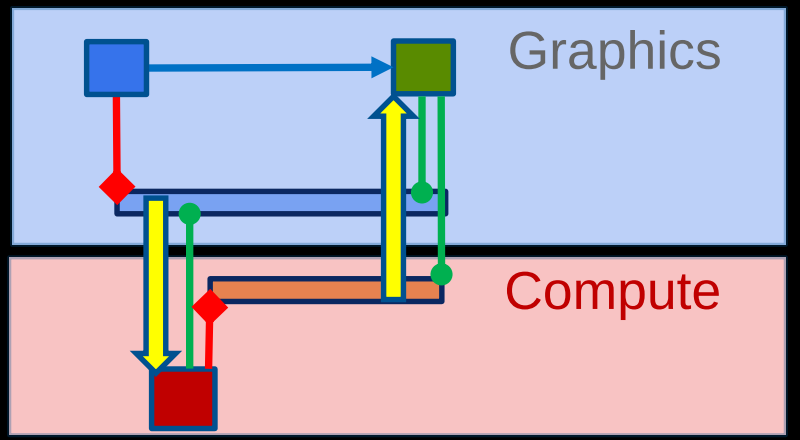
Derived dependencies:
: execution order
: auto fencing

Anvil Next Engine – Resource Tracking

- Engine tracks resource access synchronization automatically
  - Deferred lighting on GFX queue must wait for SSAO to finish

Producers:
: G-Buffer
: lighting
: SSAO

Resources:
: SSAO
: Depth

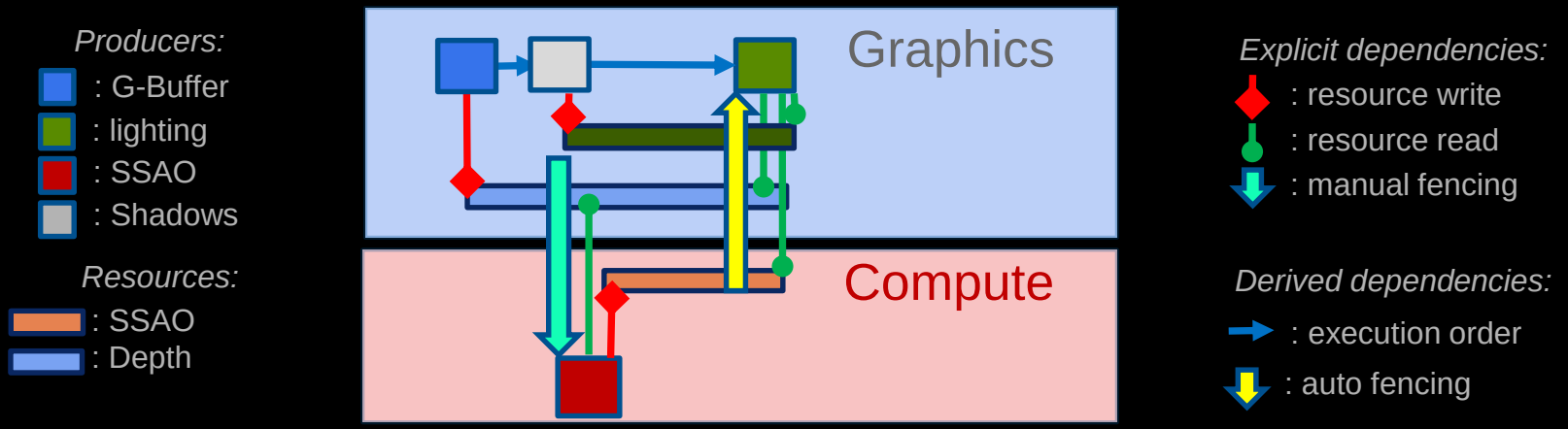


Explicit dependencies:
: resource write
: resource read

Derived dependencies:
: execution order
: auto fencing

Anvil Next Engine – Resource Tracking

- Engine tracks resource access synchronization automatically
  - User can add manual sync to better match workloads



Anvil Next Engine – Barriers

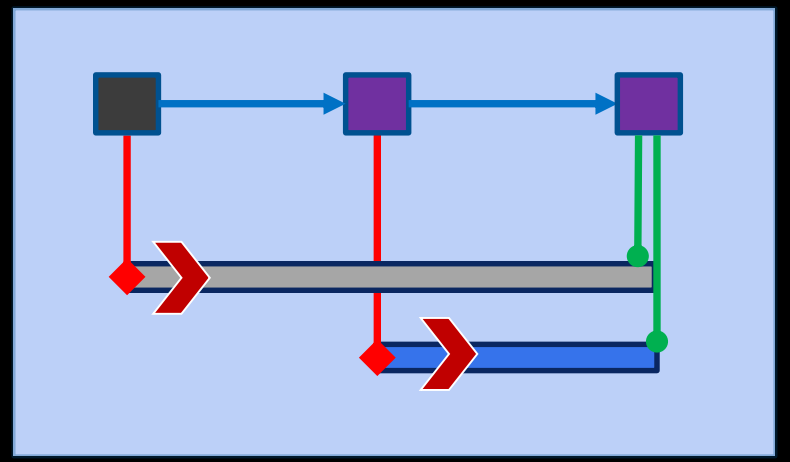
- Using producer resource dependencies
  - Batch transitions at producer boundaries
  - Determine minimal set of merged states
  - Auto split barriers

Anvil Next Engine – Barriers

- Barriers at producer boundaries

: Producer
: Depth Buffer
: Shadow Map

: resource write
: resource read



Barriers:

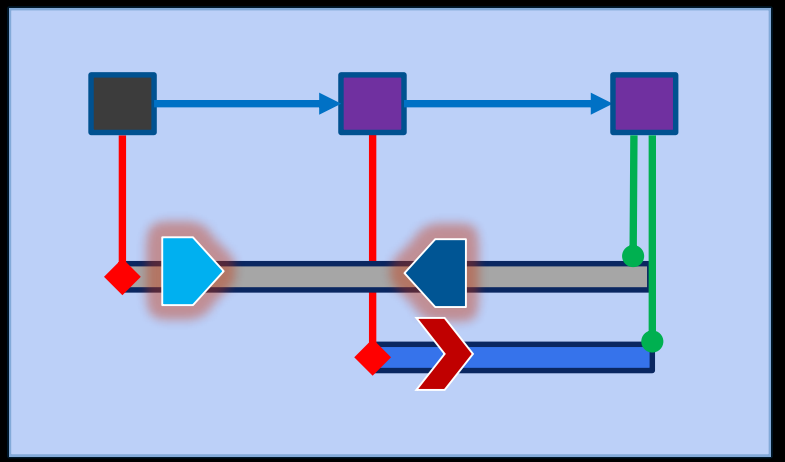
DepthWrite ->
PS Resource

Anvil Next Engine – Barriers

- Auto split barriers

: Producer
: Depth Buffer
: Shadow Map

: resource write
: resource read



Barriers:

Begin DepthWrite -> PS Resource

End DepthWrite -> PS Resource

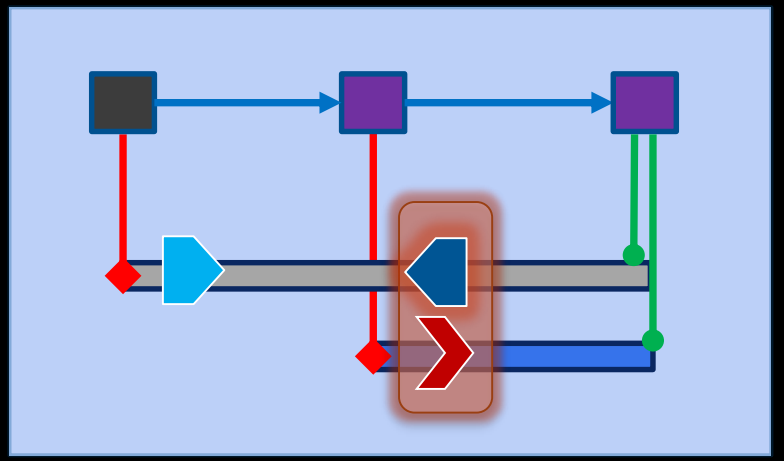
DepthWrite -> PS Resource

Anvil Next Engine – Barriers

- Group Barriers

: Producer
: Depth Buffer
: Shadow Map

: resource write
: resource read



Barriers:

Begin DepthWrite -> PS Resource

End DepthWrite -> PS Resource

DepthWrite -> PS Resource

Single call to ResourceBarrier()

Anvil Next Engine – Shader Bindings

- Re-architected to match DX12 binding model
- Engine class ShaderInputLayout maps to DX12 Root Signature
  - Hides root signature 1.0/1.1, tier restrictions etc.
- Engine class ShaderInputGroup maps to DX12 Descriptor Tables
  - Abstracts underlying API details like bind slots
  - ShaderInputGroup is the granularity of change
  - Each unique ShaderInputGroup gets compiled to an immutable Blob

Anvil Next Engine – Pipeline State

- Run-time PSO creation is expensive
- Engine thus supports two modes for PSO creation
  - Blob based PSOs for data driven material rendering code paths
    - Uses precompiled groups of state
    - Uses predefined state presets to restricts independent state changes
    - Opens the opportunity for load time/offline blob compile time optimization
  - Legacy mode for DX9-style changes in state (only used in legacy code rendering passes)
    - Late compilation (then cached)

Agenda

- Introduction
- DX12 in The Division from Massive Entertainment
- DX12 in Anvil Next Engine from Ubisoft
- DX12 in Hitman from IO Interactive
- DX12 in 'Game AAA'
- AfterMath Preview
- Nsight VSE & DirectX12 Games
- Q&A

Hitman from IO Interactive

- About the game
  - Episodic Murder simulator
  - Released March 2016

Also Check:
http://twvideo01.ubm-us.net/o1/vault/gdc2016/Presentations/meyer_jonas_rendering_hitman_with.pdf

Hitman Renderer

- Deals with fully dynamic scenes
  - Exceptions are reflection and ambient probe generation during level load
- Tile Deferred lighting
  - Forward lit uses separate pass
  - Gate/Room(Portal/Cell) system used to cull lighting
- Shadows
  - 4 VSM Cascades, 4th is static
  - 4-8 Extra shadow maps

Hitman Scene Complexity

- A typical scene contains
  - ~100k Objects
  - 10k+ Light sources
  - up to 2k Lights visible per frame
  - up to 5-10k Draw Calls Per frame
  - up to 10-15K Draw Instances Per frame

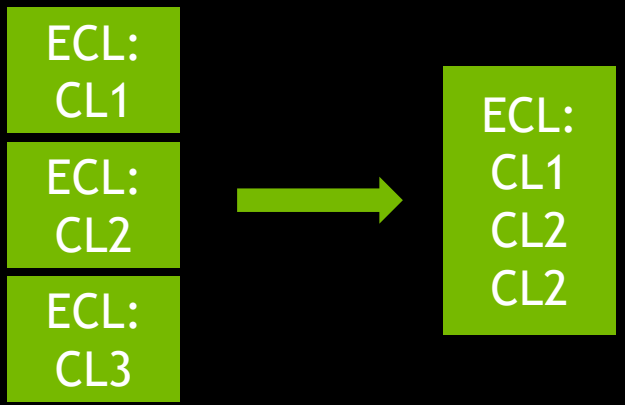
Hitman DX12 - Agenda

- CPU performance
- Memory Managment
- Allocations & Fences
- State Managment
- Resource Transitions
- Misc

Initial Problems : CPU performance

This time code could be profiled and fixed as it was engine code!

- Setting too many redundant descriptors
  - Made sure to only set descriptors that actually get used
- Sub-optimal batching
  - Ended up batching Resource Transitions and Command List submissions



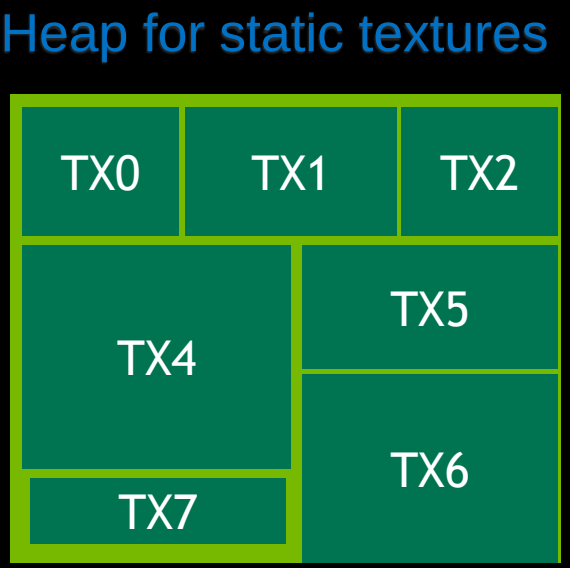
- Managed to eventually match the fastest driver through multi-threading

Initial Problems: DX12 Memory Management 1

- DX12 video memory consumption too high (vs DX11)
  - DX11 drivers are really good at moving memory to/from video memory
- Ended up implementing a system to page out (evict) resources
  - Used extremely simple LRU model
  - Lots of work that wasn't anticipated
  - Still not ideal / MakeResident blocks
  - Performs worse on DX12 especially on low vidmem cards

Initial Problems: DX12 Memory Management 2

- Implemented Render target memory reuse system (placed resources)
  - Introduced sub-allocation for static resources
    - Creating committed resources for everything will use tons of memory
- Memory savings on PC not as high as on consoles
  - Resource tiers prevent all memory to be reused for all kinds of resources

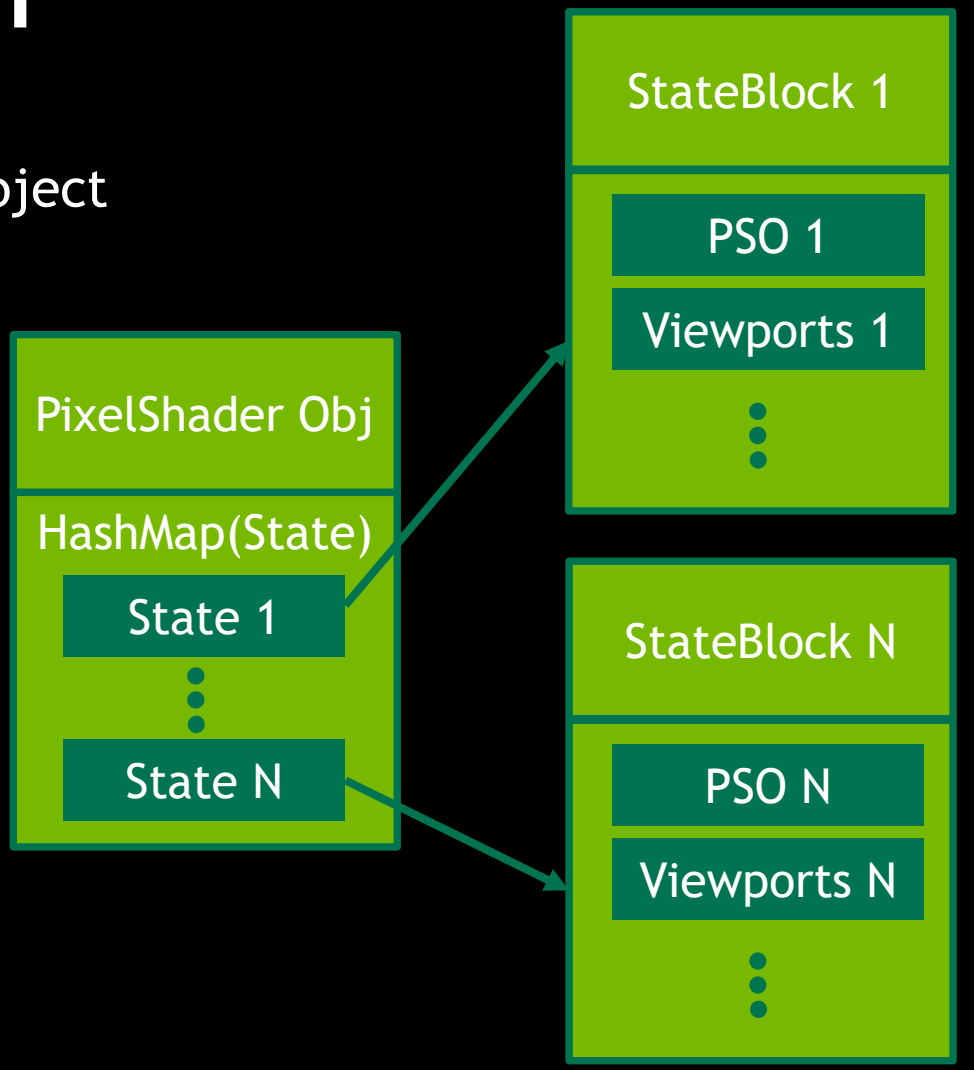


DX12 Resource Allocation & Fences

- Needed a super fast allocator for lockless allocation of frame resources
  - Lots of on-the-fly resource allocations during a frame
    - Descriptors
    - Upload heaps etc.
- Fences are expensive
  - Tried using them for fine grained reuse of resources
  - Ended up using one SignalFence to sync all resource reuse

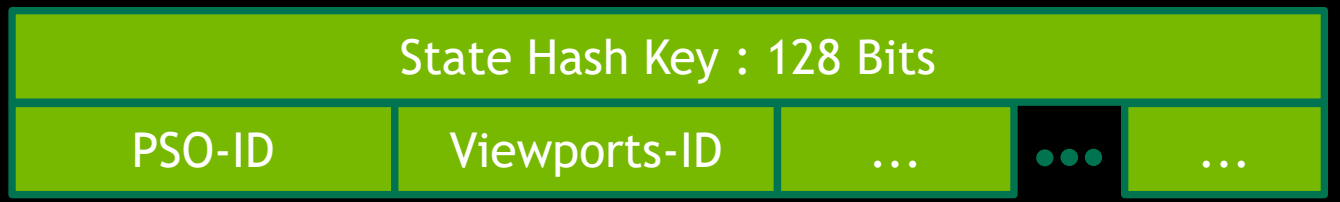
DX12 State Management 1

- Store PSO and other state with pixel shader object
  - Vast majority of pixel shaders only have a few permutations
  - Permutations accessible via hash
  - Removed sampler state objects from state management
    - Decided to use 16 fixed sampler states



DX12 State Management 2

- Create unique state hash for each state
  - Put all state blocks in pools with unique IDs
    - State blocks are: Rasterizer State, shaders etc.
    - Use block IDs as bits to construct a state hash

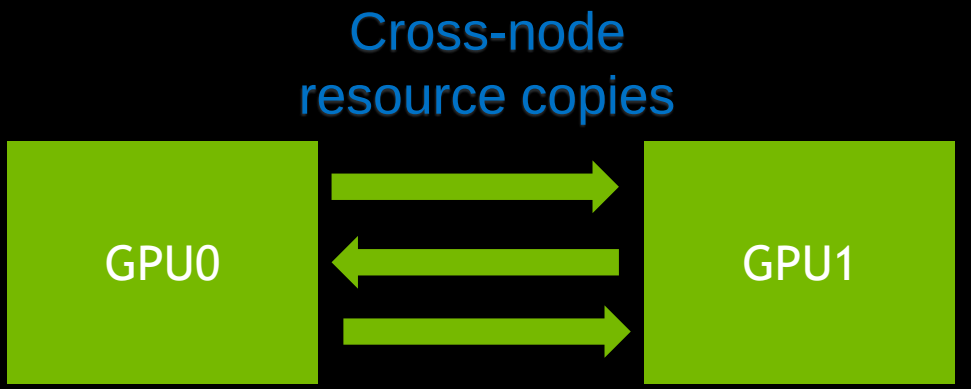


DX12 Resource Transitions

- Implemented simplified resource transition system
  - Assumes READ/SRV as the default state
  - Only supports transitions to RTV, UAV, DSV and back
  - Additional transitions needed only needed for UAV-UAV Barriers
- All transitions submitted by a main render thread
  - Main render thread can also record command lists and does all multi-thread syncs
  - Greatly simplifies code

DX12 Misc.

- Multi GPU
  - Only linked adapter mode
  - Manually copying of resources
  - Uses separate copy queue

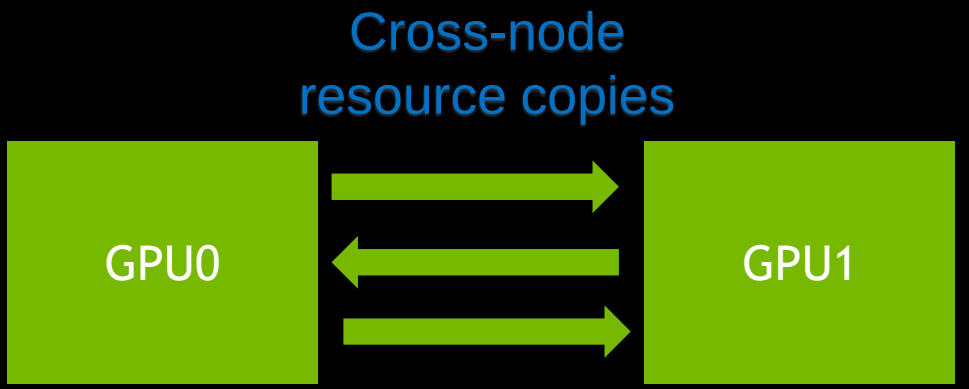


- Making DX12 as fast as DX11
  - When stuff was running 100% perfect still saw 5-10% performance delta
  - The only way this could be solved was with the help of IHV's.


GPUView: C:\Program Files (x86)\Windows Kits\10\Windows Performance Toolkit\gpuview\Merged.etl STime=22462897 Duration=6233723
File View Tools Charts Options Help
224.6
(10ms)
Adapter [NVIDIA GeForce GTX 980]
Hardware Queue
3D
Hardware Queue
3D
Hardware Queue
Copy
Hardware Queue
Copy
Hardware Queue
Copy
Hardware Queue
Copy
Hardware Queue
Copy
Hardware Queue
Copy
Hardware Queue
Compute_0
Hardware Queue
Compute_0
Hardware Queue
Compute_1
Hardware Queue
Compute_1

DX12 Misc.

- Multi GPU
  - Only linked adapter mode
  - Manually copying of resources
  - Uses separate copy queue



- Making DX12 as fast as DX11
  - When stuff was running 100% perfect still saw 5-10% performance delta
  - The only way this could be solved was with the help of IHV's.

Agenda

- Introduction
- DX12 in The Division from Massive Entertainment
- DX12 in Anvil Next Engine from Ubisoft
- DX12 in Hitman from IO Interactive
- DX12 in 'Game AAA'
- AfterMath Preview
- Nsight VSE & DirectX12 Games
- Q&A

DX12 in 'Game AAA' - Agenda

- Memory Managment
- Root Signatures
- Barriers

'Game AAA' – DX12 Memory Management

- Things learned
  - Explicit memory management is the key to great & consistent performance
  - LRU resources management strategy goes a long way
    - Keep resource in memory for a while after it has been used last
    - Bring in only when it has been evicted

'Game AAA' – Root Signature Tables 1

- Supporting resource binding tier 2
  - CB descriptors in tables must be unbound (set to 0) when not used
    - Nvidia drivers now support uncleared descriptors
  - Moved CBVs into the root to skip unbinding
    - CBVs are just a GPU address when used as root CBVs
    - No need to call CreateConstantBufferView()

'Game AAA' – Root Signature Tables 2

- Made sure to use optimal shader visibility flags for all RST entries
  - Avoid SHADER_VISIBILITY_ALL wherever possible

'Game AAA' - Root Signature Tables

```
...
CBV( b2, visibility = SHADER_VISIBILITY_ALL    ),
CBV( b0, visibility = SHADER_VISIBILITY_VERTEX ),
CBV( b1, visibility = SHADER_VISIBILITY_VERTEX ),
CBV( b0, visibility = SHADER_VISIBILITY_PIXEL  ),
CBV( b1, visibility = SHADER_VISIBILITY_PIXEL  ),
CBV( b0, visibility = SHADER_VISIBILITY_HULL   ),
CBV( b1, visibility = SHADER_VISIBILITY_HULL   ),
CBV( b0, visibility = SHADER_VISIBILITY_DOMAIN ),
CBV( b1, visibility = SHADER_VISIBILITY_DOMAIN ),

DescriptorTable( SRV(t0,     numDescriptors= 64), visibility = SHADER_VISIBILITY_VERTEX ),
DescriptorTable( Sampler(s0, numDescriptors= 16), visibility = SHADER_VISIBILITY_VERTEX ),
DescriptorTable( SRV(t0,     numDescriptors= 64), visibility = SHADER_VISIBILITY_PIXEL ),
DescriptorTable( Sampler(s0, numDescriptors= 16), visibility = SHADER_VISIBILITY_PIXEL ),
DescriptorTable( SRV(t0,     numDescriptors= 64), visibility = SHADER_VISIBILITY_HULL ),
DescriptorTable( Sampler(s0, numDescriptors= 16), visibility = SHADER_VISIBILITY_HULL ),
DescriptorTable( SRV(t0,     numDescriptors= 64), visibility = SHADER_VISIBILITY_DOMAIN ),
DescriptorTable( Sampler(s0, numDescriptors= 16), visibility = SHADER_VISIBILITY_DOMAIN )
...
```

'Game AAA' – Root Signature Tables 2

- Made sure to use optimal shader visibility flags for all RST entries
  - Avoid SHADER_VISIBILITY_ALL wherever possible
- Cache RST state in CPU memory to skip redundant binds helped CPU perf
- Mimizing RST changes turned out to be a winner
  - Changed to two layouts for the entire frame

'Game AAA' – Barriers

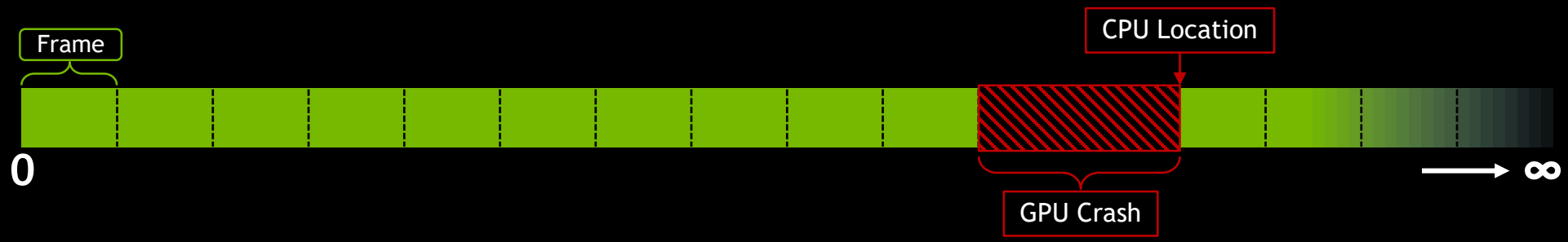
- Initial DX12 path had redudant barriers
  - Barriers were hidden in abstraction layers (triggered automatically)
    - Works most of the time
  - For specfic cases engine switches to explicit barrier management
    - NOP on DX11
- Deferred barriers were used to skip further redundancy and batch up barriers
  - Append barriers to a pending list
  - Wait until last moment to flush the list
    - Filter away redundancies

Agenda

- Introduction
- DX12 in The Division from Massive Entertainment
- DX12 in Anvil Next Engine from Ubisoft
- DX12 in Hitman from IO Interactive
- DX12 in 'Game AAA'
- AfterMath Preview
- Nsight VSE & DirectX12 Games
- Q&A

DEBUGGING GPU (CURRENTLY)

1. Crash detected based on error code from API (CPU)
2. Crash happened sometime in the last N frames of commands…
3. CPU call stack is likely a red-herring



Not useful for debugging!

See Alex Dunn’s talk about AfterMath on Thursday 3/2/2017 at 3:00 PM!

GPU DEBUGGING 101

Preventative
Changes timing
Development-use Only
Limited coverage

1st line of defense: MSFT Debug Layer

2nd line of defense: MSFT GPU-Based Validation

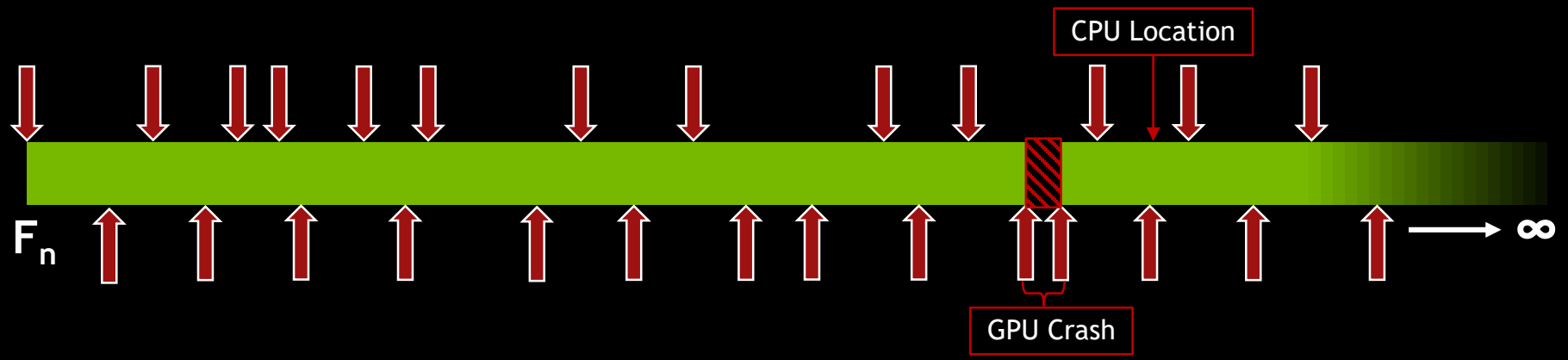
3rd line of defense:
- Catches issues that fall through
- Minimal impact
- Shippable

NVIDIA AFTERMATH

KO: Increase accuracy of GPU crash location

Idea:

- Inline user defined markers with the command stream
- GPU signals each marker once reached
- Last marker reached indicates GPU crash location



NVIDIA AFTERMATH

Deployment

- New tool to help diagnose GPU crashes (Header + DLL)
- **Extremely** flexible/simple API
- Currently compatible with; DX11 and DX12 – UWP and/or Windows 7+

Limitations

- Requires NVIDIA GeForce driver version 378.xx and above!
- Currently not compatible with D3D debug layers

Agenda

- Introduction
- DX12 in The Division from Massive Entertainment
- DX12 in Anvil Next Engine from Ubisoft
- DX12 in Hitman from IO Interactive
- DX12 in 'Game AAA'
- AfterMath Preview
- Nsight VSE & DirectX12 Games
- Q&A

Agenda

- Introduction
- DX12 in The Division from Massive Entertainment
- DX12 in Anvil Next Engine from Ubisoft
- DX12 in Hitman from IO Interactive
- DX12 in 'Game AAA'
- AfterMath Preview
- Nsight VSE & DirectX12 Games
- Q&A

Q&A