Rendering Hitman with DirectX 12

Jonas Meyer
Lead Render Programmer, Io-Interactive

Agenda

- 1 Hitman frame
- DirectX 12 Implementation
- DirectX 12 vs. DirectX 11 Performance

Glacier

- No precomputation
  - Fast iteration ☺
- Dynamic time of day
  - Fixed on level startup
- Probe based reflections
  - Generated on level load
- Probes also used for ambient
- Tile Deferred

1 Frame

- 3500 Draw Calls
- 8000 Instances

G-Buffer

Light Macro Tiles

Light Tiles

Probes

- Reflections
- Ambient

Lighting

Dark Lights

Transparent

Atmospheric Scattering

Post-fx

DirectX 12 Goals

- Goals:
  - Improve CPU Performance
  - Improve GPU Performace with Async compute
- Not a rewrite:
  - Still supporting DirectX 11

Temp Allocator

- DX12 requires lots of temporary resources
  - Need a fast, multithreaded allocator
  - Ours is similar to cgyrling[0]
    - Large locked allocator maintains blocks
      - 1 Per resource type
    - Small lock free allocators claim blocks of resources
      - 1 Per thread per resource type
    - Fences control when blocks can be reused

[0] http://www.gdcvault.com/play/1022186/Parallelizing-the-Naughty-Dog-Engine

Temp Resource types

- Upload Memory
  - Constant buffers
- Descriptors
  - CBV
  - UAV
  - SRV

Root signature

```
CD3DX12_DESCRIPTOR_RANGE R[10];
R[0].Init(D3D12_DESCRIPTOR_RANGE_TYPE_SRV, 18, 0, 0); // 0-18
R[1].Init(D3D12_DESCRIPTOR_RANGE_TYPE_CBV, 8, 0, 0);
R[2].Init(D3D12_DESCRIPTOR_RANGE_TYPE_SRV, 18, 0, 0);
R[3].Init(D3D12_DESCRIPTOR_RANGE_TYPE_CBV, 8, 0, 0);
R[4].Init(D3D12_DESCRIPTOR_RANGE_TYPE_SRV, 18, 0, 0);
R[5].Init(D3D12_DESCRIPTOR_RANGE_TYPE_CBV, 8, 0, 0);
R[6].Init(D3D12_DESCRIPTOR_RANGE_TYPE_SRV, 18, 0, 0);
R[7].Init(D3D12_DESCRIPTOR_RANGE_TYPE_CBV, 8, 0, 0);
R[8].Init(D3D12_DESCRIPTOR_RANGE_TYPE_SRV, 17, 15); // 19-31
R[9].Init(D3D12_DESCRIPTOR_RANGE_TYPE_SAMPLER, 16, 0, 0);

CD3DX12_ROOT_PARAMETER Slot[10];
Slot[0].InitAsDescriptorTable(1, &R[0], D3D12_SHADER_VISIBILITY_PIXEL);
Slot[1].InitAsDescriptorTable(1, &R[1], D3D12_SHADER_VISIBILITY_PIXEL);
Slot[2].InitAsDescriptorTable(1, &R[2], D3D12_SHADER_VISIBILITY_VERTEX);
Slot[3].InitAsDescriptorTable(1, &R[3], D3D12_SHADER_VISIBILITY_VERTEX);
Slot[4].InitAsDescriptorTable(1, &R[4], D3D12_SHADER_VISIBILITY_HULL);
Slot[5].InitAsDescriptorTable(1, &R[5], D3D12_SHADER_VISIBILITY_HULL);
Slot[6].InitAsDescriptorTable(1, &R[6], D3D12_SHADER_VISIBILITY_DOMAIN);
Slot[7].InitAsDescriptorTable(1, &R[7], D3D12_SHADER_VISIBILITY_DOMAIN);
Slot[8].InitAsDescriptorTable(1, &R[8], D3D12_SHADER_VISIBILITY_ALL);
Slot[9].InitAsDescriptorTable(1, &R[9], D3D12_SHADER_VISIBILITY_ALL);
```

Root signature

- **Per Stage**
  - **18 SRVs**
  - **8 CBVs**

```
CD3DX12_DESCRIPTOR_RANGE R[10];
R[0].Init(D3D12_DESCRIPTOR_RANGE_TYPE_SRV, 18, 0, 0); // 0-18
R[1].Init(D3D12_DESCRIPTOR_RANGE_TYPE_CBV, 8, 0, 0);
R[2].Init(D3D12_DESCRIPTOR_RANGE_TYPE_SRV, 18, 0, 0);
R[3].Init(D3D12_DESCRIPTOR_RANGE_TYPE_CBV, 8, 0, 0);
R[4].Init(D3D12_DESCRIPTOR_RANGE_TYPE_SRV, 18, 0, 0);
R[5].Init(D3D12_DESCRIPTOR_RANGE_TYPE_CBV, 8, 0, 0);
R[6].Init(D3D12_DESCRIPTOR_RANGE_TYPE_SRV, 18, 0, 0);
R[7].Init(D3D12_DESCRIPTOR_RANGE_TYPE_CBV, 8, 0, 0);
R[8].Init(D3D12_DESCRIPTOR_RANGE_TYPE_SRV, 17, 15); // 19-31
R[9].Init(D3D12_DESCRIPTOR_RANGE_TYPE_SAMPLER, 16, 0, 0);

CD3DX12_ROOT_PARAMETER Slot[10];
Slot[0].InitAsDescriptorTable(1, &R[0], D3D12_SHADER_VISIBILITY_PIXEL);
Slot[1].InitAsDescriptorTable(1, &R[1], D3D12_SHADER_VISIBILITY_PIXEL);
Slot[2].InitAsDescriptorTable(1, &R[2], D3D12_SHADER_VISIBILITY_VERTEX);
Slot[3].InitAsDescriptorTable(1, &R[3], D3D12_SHADER_VISIBILITY_VERTEX);
Slot[4].InitAsDescriptorTable(1, &R[4], D3D12_SHADER_VISIBILITY_HULL);
Slot[5].InitAsDescriptorTable(1, &R[5], D3D12_SHADER_VISIBILITY_HULL);
Slot[6].InitAsDescriptorTable(1, &R[6], D3D12_SHADER_VISIBILITY_DOMAIN);
Slot[7].InitAsDescriptorTable(1, &R[7], D3D12_SHADER_VISIBILITY_DOMAIN);
Slot[8].InitAsDescriptorTable(1, &R[8], D3D12_SHADER_VISIBILITY_ALL);
Slot[9].InitAsDescriptorTable(1, &R[9], D3D12_SHADER_VISIBILITY_ALL);
```

Root signature

- Per Stage
  - 18 SRVs
  - 8 CBVs
- **15 shared SRVs**

```
CD3DX12_DESCRIPTOR_RANGE R[10];
R[0].Init(D3D12_DESCRIPTOR_RANGE_TYPE_SRV, 18, 0, 0); // 0-18
R[1].Init(D3D12_DESCRIPTOR_RANGE_TYPE_CBV, 8, 0, 0);
R[2].Init(D3D12_DESCRIPTOR_RANGE_TYPE_SRV, 18, 0, 0);
R[3].Init(D3D12_DESCRIPTOR_RANGE_TYPE_CBV, 8, 0, 0);
R[4].Init(D3D12_DESCRIPTOR_RANGE_TYPE_SRV, 18, 0, 0);
R[5].Init(D3D12_DESCRIPTOR_RANGE_TYPE_CBV, 8, 0, 0);
R[6].Init(D3D12_DESCRIPTOR_RANGE_TYPE_SRV, 18, 0, 0);
R[7].Init(D3D12_DESCRIPTOR_RANGE_TYPE_CBV, 8, 0, 0);
R[8].Init(D3D12_DESCRIPTOR_RANGE_TYPE_SRV, 17, 15); // 19-31
R[9].Init(D3D12_DESCRIPTOR_RANGE_TYPE_SAMPLER, 16, 0, 0);

CD3DX12_ROOT_PARAMETER Slot[10];
Slot[0].InitAsDescriptorTable(1, &R[0], D3D12_SHADER_VISIBILITY_PIXEL);
Slot[1].InitAsDescriptorTable(1, &R[1], D3D12_SHADER_VISIBILITY_PIXEL);
Slot[2].InitAsDescriptorTable(1, &R[2], D3D12_SHADER_VISIBILITY_VERTEX);
Slot[3].InitAsDescriptorTable(1, &R[3], D3D12_SHADER_VISIBILITY_VERTEX);
Slot[4].InitAsDescriptorTable(1, &R[4], D3D12_SHADER_VISIBILITY_HULL);
Slot[5].InitAsDescriptorTable(1, &R[5], D3D12_SHADER_VISIBILITY_HULL);
Slot[6].InitAsDescriptorTable(1, &R[6], D3D12_SHADER_VISIBILITY_DOMAIN);
Slot[7].InitAsDescriptorTable(1, &R[7], D3D12_SHADER_VISIBILITY_DOMAIN);
Slot[8].InitAsDescriptorTable(1, &R[8], D3D12_SHADER_VISIBILITY_ALL);
Slot[9].InitAsDescriptorTable(1, &R[9], D3D12_SHADER_VISIBILITY_ALL);
```

Root signature

- Per Stage
  - 18 SRVs
  - 8 CBVs
- 15 shared SRVs
- **16 shared samplers**

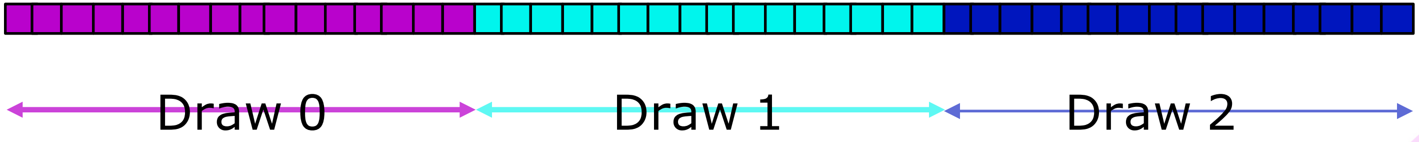
```
CD3DX12_DESCRIPTOR_RANGE R[10];
R[0].Init(D3D12_DESCRIPTOR_RANGE_TYPE_SRV, 18, 0, 0); // 0-18
R[1].Init(D3D12_DESCRIPTOR_RANGE_TYPE_CBV, 8, 0, 0);
R[2].Init(D3D12_DESCRIPTOR_RANGE_TYPE_SRV, 18, 0, 0);
R[3].Init(D3D12_DESCRIPTOR_RANGE_TYPE_CBV, 8, 0, 0);
R[4].Init(D3D12_DESCRIPTOR_RANGE_TYPE_SRV, 18, 0, 0);
R[5].Init(D3D12_DESCRIPTOR_RANGE_TYPE_CBV, 8, 0, 0);
R[6].Init(D3D12_DESCRIPTOR_RANGE_TYPE_SRV, 18, 0, 0);
R[7].Init(D3D12_DESCRIPTOR_RANGE_TYPE_CBV, 8, 0, 0);
R[8].Init(D3D12_DESCRIPTOR_RANGE_TYPE_SRV, 17, 15); // 19-31
R[9].Init(D3D12_DESCRIPTOR_RANGE_TYPE_SAMPLER, 16, 0, 0);

CD3DX12_ROOT_PARAMETER Slot[10];
Slot[0].InitAsDescriptorTable(1, &R[0], D3D12_SHADER_VISIBILITY_PIXEL);
Slot[1].InitAsDescriptorTable(1, &R[1], D3D12_SHADER_VISIBILITY_PIXEL);
Slot[2].InitAsDescriptorTable(1, &R[2], D3D12_SHADER_VISIBILITY_VERTEX);
Slot[3].InitAsDescriptorTable(1, &R[3], D3D12_SHADER_VISIBILITY_VERTEX);
Slot[4].InitAsDescriptorTable(1, &R[4], D3D12_SHADER_VISIBILITY_HULL);
Slot[5].InitAsDescriptorTable(1, &R[5], D3D12_SHADER_VISIBILITY_HULL);
Slot[6].InitAsDescriptorTable(1, &R[6], D3D12_SHADER_VISIBILITY_DOMAIN);
Slot[7].InitAsDescriptorTable(1, &R[7], D3D12_SHADER_VISIBILITY_DOMAIN);
Slot[8].InitAsDescriptorTable(1, &R[8], D3D12_SHADER_VISIBILITY_ALL);
Slot[9].InitAsDescriptorTable(1, &R[9], D3D12_SHADER_VISIBILITY_ALL);
```

Descriptor burn

- Per draw descriptor usage:
  - 36 for SRV,
  - 16 for CBV
- 520k Descriptors for a 10k draw frame
  - Writing that many descriptors is slow
  - Requires multiple descriptor heaps

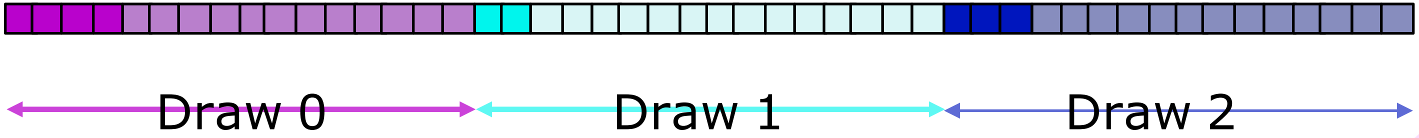
Descriptor burn

- Example:
  - SRV descriptors, one stage, three draw calls
- Naïve way



Descriptor burn

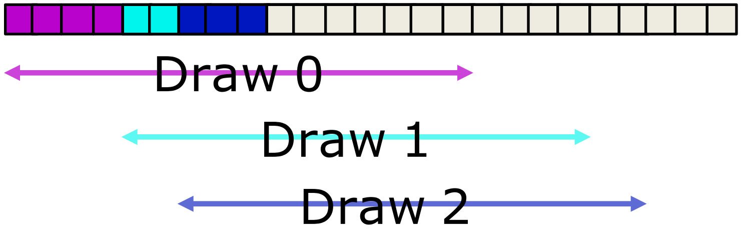
- Example:
  - SRV descriptors, one stage, three draw calls
- Naïve way



- Observation: Not all entries are used

Descriptor burn

- Solution: Allow overlap
  - Only put in descriptor actually used by shader
  - Restricts Descriptor heap type
  - Pad with Null descriptors
    - Only on submit



Pipeline State Objects

- Our interface is still DX11 based
  - Programmers prefer this
- PSOs handled internally
- Store an array in with the Pixel Shader
  - State is hashed into 128bit key
  - Every object has a runtime unique id
    - Assigned & deduplicated on creation
    - Makes the hashing a no-op

Pipeline State Objects

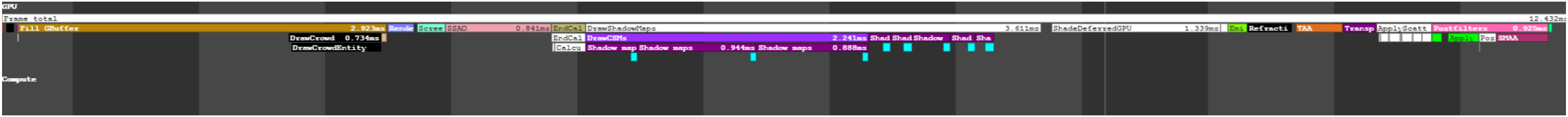
```
struct SPipelineStateObjectHash
{
    union
    {
        struct
        {
            uint64 VertexShader : RENDER_SHADER_BITS;                       // 12      4k
            uint64 PixelShader : RENDER_SHADER_BITS;                        // 12      4k
            uint64 InputLayout : RENDER_INPUT_LAYOUT_BITS;                  // 6       64
            uint64 RasterizerState : RENDER_RASTERIZER_STATE_BITS;          // 7      128
            uint64 BlendStateState : RENDER_BLEND_STATE_BITS;               // 8      256
            uint64 DepthStencilState : RENDER_DEPTHSTENCIL_STATE_BITS;      // 8      256
            uint64 RenderTargetFormat : RENDER_TARGET_FORMAT_BITS;          // 4       16
            uint64 Topology : RENDER_TOPOLOGY_BITS;                         // 3        8
            uint64 DomainShader : RENDER_SHADER_BITS;                       // 12      4k
            uint64 HullShader : RENDER_SHADER_BITS;                         // 12      4k
            uint64 _Pad : 128 - 84;                                         // = 84
        };
        struct
        {
            uint64 nHash0;
            uint64 nHash1;
        };
    };
};
```

Multithreading

- Want to submit command list before they are finished
  - Allows more parallelism
  - Async Command Lists
  - Not available in DirectX 12
- Easy to emulate
  - Push all Command Lists into a queue
  - Submit in order as they finish

Async Compute

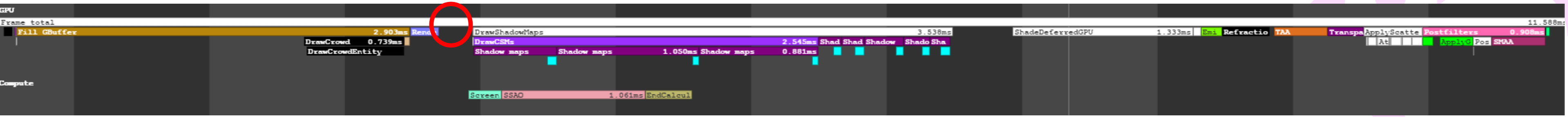
- Overlap independent work



- SSSAA
- SSAO
- Light Tile Calculations

Async Compute

- Graphics Queue: Write Fence
- Graphics Queue: Render Shadows

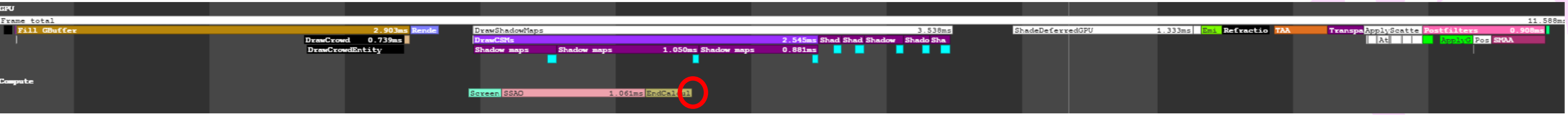


Async Compute

- Graphics Queue: Write Fence
- Graphics Queue: Render Shadows
- Compute Queue: Wait for Fence
- Compute Queue: Execute Async work

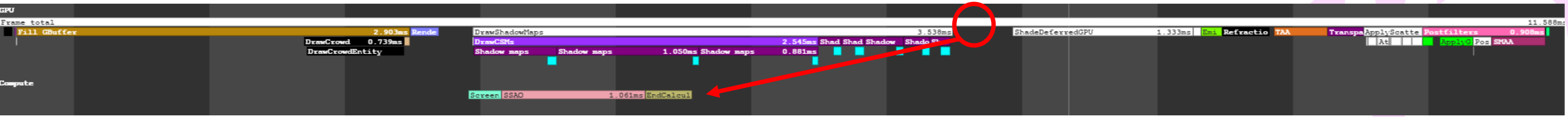
Async Compute

- Graphics Queue: Write Fence
- Graphics Queue: Render Shadows
- Compute Queue: Wait for Fence
- Compute Queue: Execute Async work
- Compute Queue: Write Fence



Async Compute

- Graphics Queue: Write Fence
- Graphics Queue: Render Shadows
- Compute Queue: Wait for Fence
- Compute Queue: Execute Async work
- Compute Queue: Write Fence
- Graphics Queue: Wait for Fence



Async Compute

- Win of 5-10% on AMD
- No difference on Nvidia
    - Working with Nvidia to get this fixed
- Hard to tune.
    - Too much async work can make it a penalty
    - PC has lots of configurations

Resource Transitions

- D3D12 Transitions are complicated
  - We dont want to have to worry too much about that when writing code
- We annotate render code with transitions
  - Simplified version of D3D12 Transitions
  - Only two transitions
    - To View defined state
      - UAV for UAVS
      - RTV for RTVS
      - DSV for DSV
    - To Read
  - One exception per resource
  - Subresource implied by view

```
#define SUBRESOURCE_TRANSITION_SRV(pDeviceContext, pSRV) ...
#define SUBRESOURCE_TRANSITION_RTV(pDeviceContext, pRTV) ...
#define SUBRESOURCE_TRANSITION_RTV_READ(pDeviceContext, pRTV) ...
#define SUBRESOURCE_TRANSITION_DSV(pDeviceContext, pDSV) ...
#define SUBRESOURCE_TRANSITION_DSV_READ(pDeviceContext, pDSV) ...
#define SUBRESOURCE_TRANSITION_UAV(pDeviceContext, pUAV) ...
#define SUBRESOURCE_TRANSITION_UAV_READ(pDeviceContext, pUAV) ...
```

Resource Transitions

- We only allow transitions on one thread
  - No resource state patching
  - Batching & optimization of changes becomes simple

Resource Transitions

- Slow when gpu bound? Check your transitions
  - Dont do unecessary transitions
- Use COMMON to upload
  - VB, IB, Read only Textures
- Never use COMMON or GENERIC_READ for
  - Render Targets
  - UAVs

Memory Budget

- You should care about memory budget
- Can change dynamically
- If you fail to follow, Windows will enforce
  - Resources will be pushed out of video memory
- No Resource Priorities in DX12
  - They exists for the driver
  - Usually this is enough
  - We had problems with UAVs being pushed to system memory
  - Maybe we'll be able to set priorities in the future?

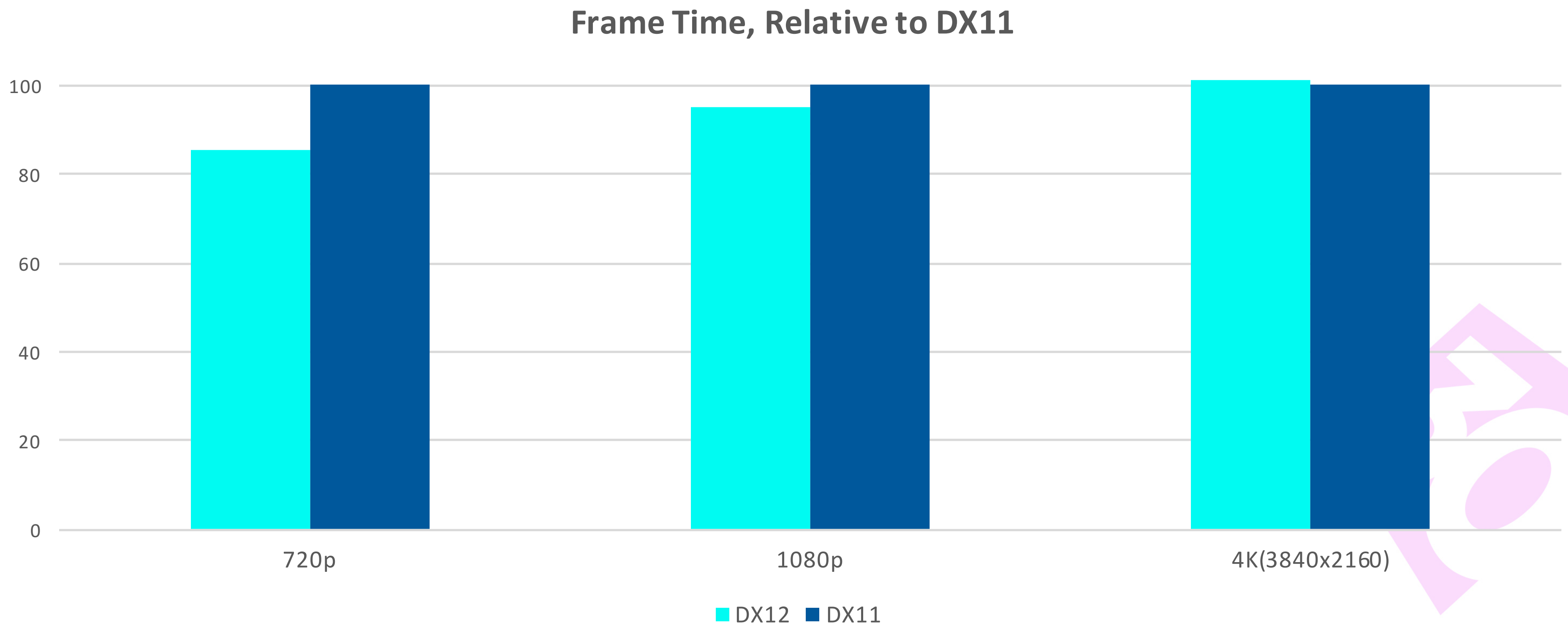
MakeResident & Evict

- The official guide line is:
  - Use MakeResident & Evict to ensure you are within the memory budget
- Evict
  - Makes a ressource unusable
  - Lazy, Never blocks
  - But budget updated immediately
- MakeResident
  - Makes an Evicted resource usable
  - Synchronous
  - Time proportional to size of resource

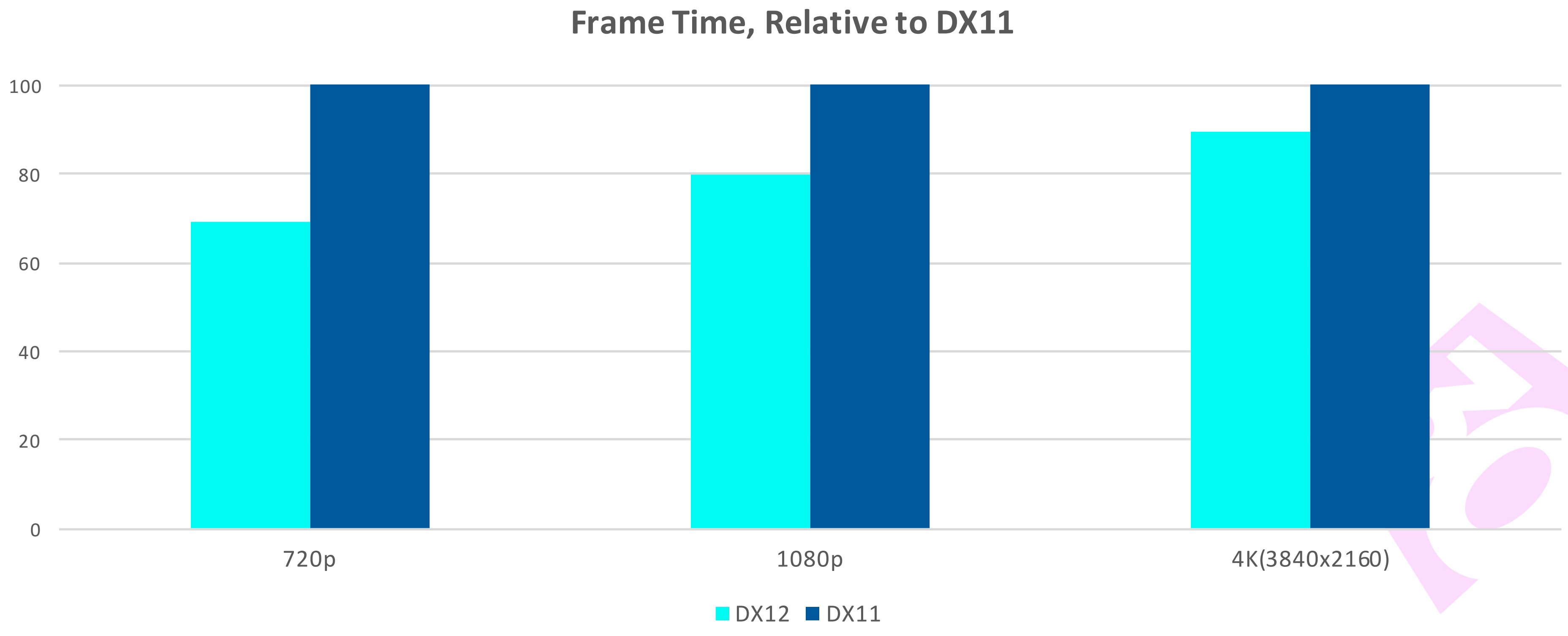
The MakeResident/Evict Rabbit Hole

- Complicated
- Hard to get right
- Easy to get wrong
- For Optimal Eviction
  - All resources are comitted resources
  - Wastes huge amount of memory (1gb!)
  - Comitted resources are 64kb aligned
- Compromise:
  - Resources >= 64KB -> Comitted
  - Resources < 64KB -> Suballocated in multiple heaps
  - VB/IB in system mem on low end hardware
  - Only Evict once per frame

D3D11 vs D3D12



D3D11 vs D3D12



Acknowledgements

- Anders Wang Kristensen
- Kasper Høy Nielsen
- Tim van Klooster
- Rune Lehard Hansen Stubbe

Questions?

- jonasm@ioi.dk

Thank you for listening