DirectX 12 엔진 개발

유영천
Microsoft MVP
Visual Studio and Development Technologies

Twitter: @dgtman
Blog : http://megayuchi.wordpress.com

DirectX?

- MS의 Graphics API
- 대응되는 비 Windows계열 API로 OpenGL이 있음.
- 대부분의 Windows 게임과 100%의 XBOX, Wndows Phone게임이 DirectX를 사용
- 게임 수로 보면 DX9 >>>> DX11 >> DX12

DirectX 12?

- DirectX 12 == Direct 3D12
  - Direct 2D, Direct Write를 포함하지 않음.
  - D3D외의 기능은 D3D11 on D3D12를 사용하면 됨.

- 성능 최우선!
  - CPU 성능이 정체되었다.
  - GPU 성능은 정체되지 않았지만 가격이..그리고 데스크탑 점유율이 낮아지고 있다.
  - 현세대 하드웨어만으로 성능 향상을 올릴 수 없을까?

성능 개선을 위한 포인트

- CPU ->Draw Call -> GPU 처리의 과정을 줄임.
- 완전한 비동기 렌더링
- 각종 State들을 한방에 처리 ->
  - OMSet…RSSet…PSSet…VSSet… -> ID3D12PipenlineState 한 개로.
- DirectX runtime과 드라이버에서 해주던 일들을 Application레벨로 빼냄.
  - 하지만 그 결과는…..-_-;;;;;;;;

결론부터 얘기하겠습니다.

…

DirectX 12

F1머신. 드라이버 실력이 받쳐주지 못하면 출발도 못함.

DirectX 9/11

운전 보조 장치가 붙어있는 스포츠카

DirectX 9/11이 스포츠카라고?

- 암묵적인(implicit) 처리가 많기 때문에 DirectX runtime과 드라이버에서 최적화시킬 여지도 많음. 실제로 상당한 최적화가 이루어져있다.
- 지난 십 여년간 GPU회사들이 자사의 드라이버를 엄청나게 최적화 시켰음.
- Single-Thread로만 렌더링을 해도 GPU점유율 100%에 도달할 수 있는 것은 runtime 최적화와 Driver최적화 덕분.

- DirectX 12에선 이 보조장치들의 지원을 받지 못함.

DirectX 12가 필요한가?

- 이미 120프레임 이상으로 성능이 충분히 나오고 있다면 굳이 DX12로 바꿀 필요는 없다.

- Multi-Thread 렌더링을 이미 사용하고 있고 빈틈없는 최적화를 할 자신이 있다면...시도해 볼만 하다.
- GPU병목이 걸려있고 더 이상 최적화 여지가 없어 보인다.
- 45프레임 나오는 게임을 어떻게든 60프레임에 맞추고 싶다.

DirectX 9/11 -> DirectX 12

- Resource Binding
- Command List & Command Queue
- State변경 -> ID3D12PipelineState
- Shader 코드는 그대로 사용 가능.
- D3DX? DirectXTex? -> 그런거 없음. 텍스쳐 파일 로딩함수도 직접 만들어야함.
- 여기까지 하면 돌아는 간다.-_-

Resource Binding

DX9/11 –> DX12로 포팅작업을 한다면 가장 많은 시간을 들이게 될...(대부분 여기서 포기한다는데 500원 건다)

Resource Binding

- Vertex Buffer
- Index Buffer
- Texture
- Constant Buffer
- Unorderd Acceess Buffer (for Compute Shader)
- Sampler

렌더링을 위해 이러한 Resource들을 Graphics Pipeline에 bind한다.

용어정리

- Buffer – ID3D12Resource , D3D에서 사용하는 CPU/GPU 메모리 블록.
- CBV – Constant Buffer View
  - Buffer를 ConstantBuffer로서 Shader에서 사용하기 위한 자료구조.
- SRV – Shader Resource View
  - Buffer를 Texture로서 Shader에서 사용하기 위한 자료구조.
- UAV – Unordered Access View
  - Buffer를 Compute Shader에서 read/write하기 위한 자료구조

Descriptor

- Resource의 정보를 기술한 메모리 블록
- CBV,SRV,UAV를 생성하면 그 결과로 이 Descriptor를 얻는다.
- Texture(SRV)인지, Constant Buffer(CBV)인...
- 32-64bytes 사이즈.(GPU마다 다름.)
- 객체가 아니다. 해제할 필요 없다.
- 내부적으로 GPU Memory, CPU Memory pair로 구성됨.
- D3D12_GPU_DESCRIPTOR_HANDLE, D3D12_CPU_DESCRIPTOR_HANDLE로 표현되며 사실상 포인터.

Descriptor Heap

- Descriptor로 사용할 Memory배열
- ID3D12DescriptorHeap로 구현되어 있다.
- CPU측 메모리, GPU측 메모리 pair로 구성되어있다.
- CBV,SRV는 이 Descriptor Heap의 CPU/GPU 메모리에 생성(write)된다.
- Descriptor Heap의 CPU측 메모리에 write, Shader에선 GPU메모리에서 read한다.

Root Signature – ID3D12RootSignature

- 어떤(Texture, Contant Buffer, Sampler등) Resource가 어떻게 Pipeline에 bind 될지를 정의
- Resource binding 설정을 기술한 일종의 템플릿이다.

Descriptor Table

- Descriptor의 논리적 배열
- Descriptor Heap의 임의의 위치가 Descriptor Table에 맵핑된다.

TR Matrix	Bones Matrix	Light Cube	Shadow Map

pCommandList->SetGraphicsRootDescriptorTable(0, gpuHeap);

FaceGroup 0	material	diffuse	mask	toon

pCommandList->SetGraphicsRootDescriptorTable(1, gpuHeap);

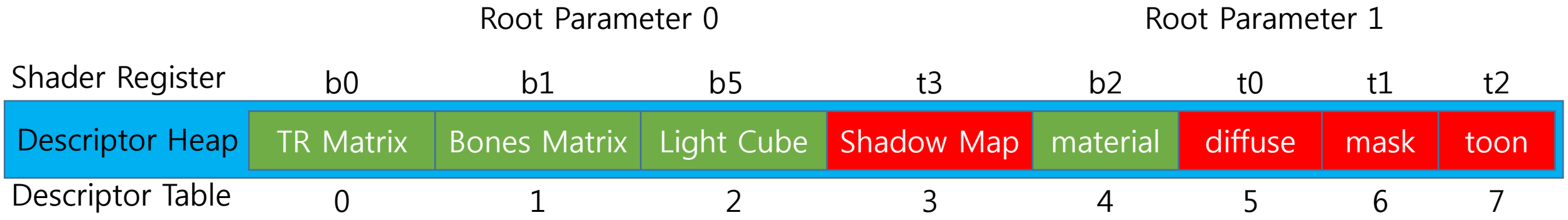
FaceGroup 1	material	diffuse	mask	toon

pCommandList->SetGraphicsRootDescriptorTable(1, gpuHeap);

FaceGroup 2	material	diffuse	mask	toon

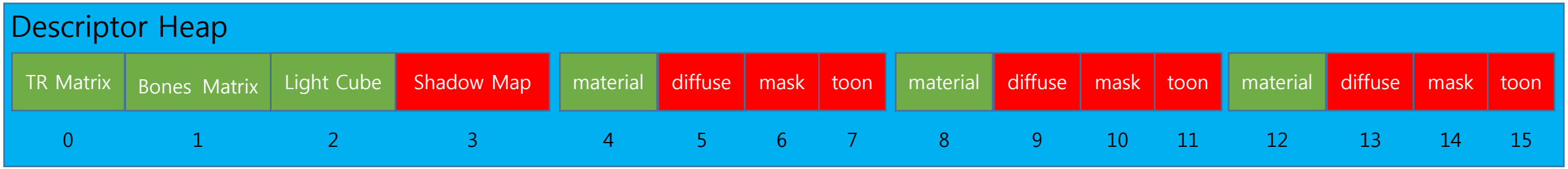
pCommandList->SetGraphicsRootDescriptorTable(1, gpuHeap);

```
CD3DX12_DESCRIPTOR_RANGE rangesPerObj[3];
rangesPerObj[0].Init(D3D12_DESCRIPTOR_RANGE_TYPE_CBV, 2, 0);// b0 : default , b1 : bones
rangesPerObj[1].Init(D3D12_DESCRIPTOR_RANGE_TYPE_CBV, 1, 5);// b5 : Light Cube
rangesPerObj[2].Init(D3D12_DESCRIPTOR_RANGE_TYPE_SRV, 1, 3);// t3 : shadow

CD3DX12_DESCRIPTOR_RANGE rangesPerFacegroup[2];
rangesPerFacegroup[0].Init(D3D12_DESCRIPTOR_RANGE_TYPE_CBV, 1, 2);// b2 : material
rangesPerFacegroup[1].Init(D3D12_DESCRIPTOR_RANGE_TYPE_SRV, 3, 0);// t0 : diffuse , t1 : mask , t2 : toon ta
ble

CD3DX12_ROOT_PARAMETER rootParameters[2];
rootParameters[0].InitAsDescriptorTable(_countof(rangesPerObj), rangesPerObj, D3D12_SHADER_VISIBILITY_ALL);
rootParameters[1].InitAsDescriptorTable(_countof(rangesPerFacegroup), rangesPerFacegroup, D3D12_SHADER_VISIB
ILITY_ALL);
```



필요 Descriptor개수 : 16개

```
D3D12_GPU_DESCRIPTOR_HANDLE gpuHeap = pDescriptorHeap->GetGPUDescriptorHandleForHeapStart();
pCommandList->IASetVertexBuffer(...);
pCommandList->SetGraphicsRootDescriptorTable(0, gpuHeap);
for (i=0; i<3; i++)
{
        pCommandList->SetGraphicsRootDescriptorTable(1, gpuHeap);
        pCommandList->IASetIndexBuffer(...);
        pCommandList->DrawIndexedInstanced(...);
        gpuHeap.Offset(4, DescriptorSize);

}
```

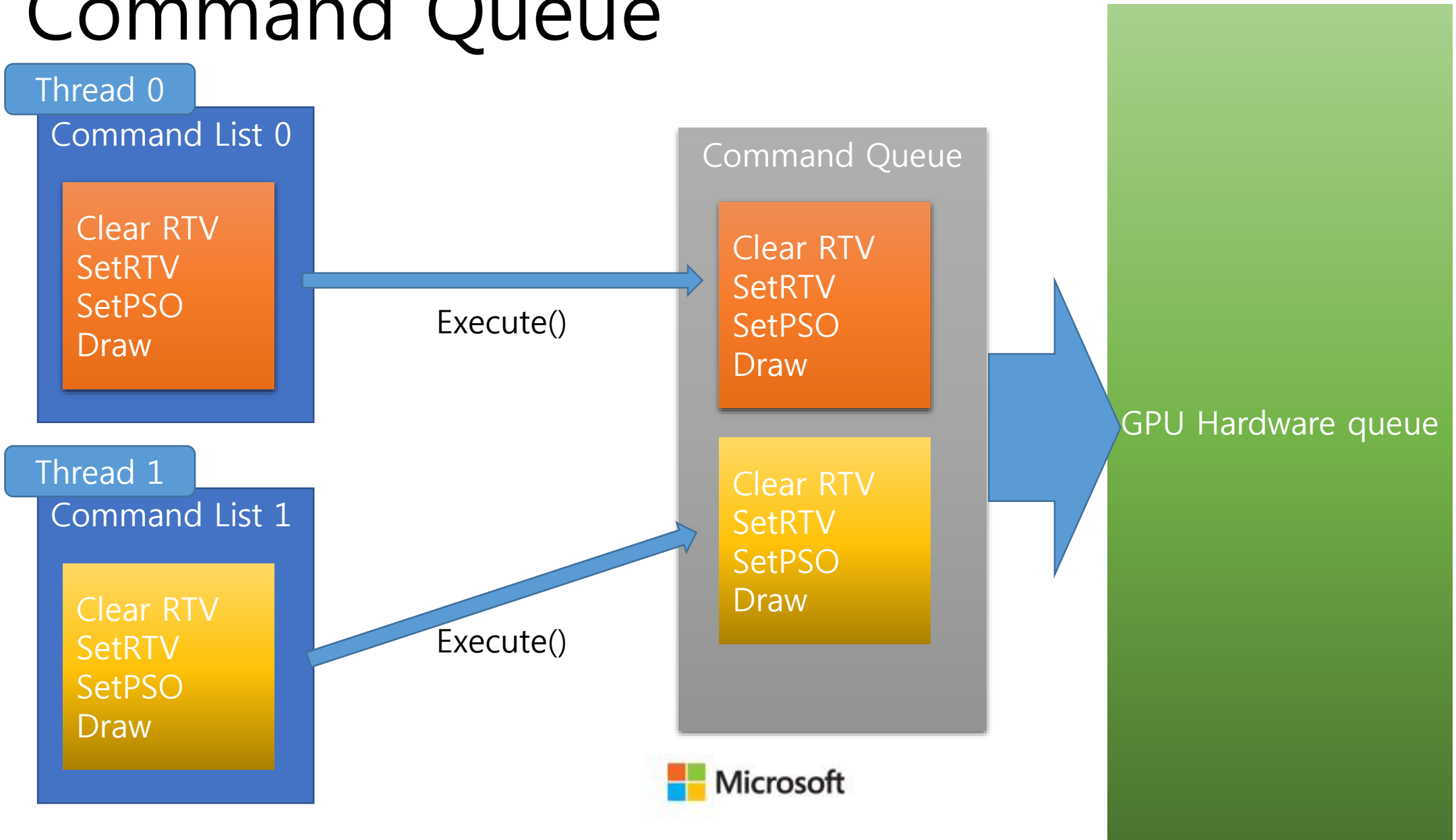


Object Face Group 0 Face Group 1 Face Group 2

Command List & Command Queue

- 비동기 렌더링을 위한 디자인
- Graphics Command를 Command List 에 Recording해서
- Command Queue에 전송. (이 시점에 GPU Queue로 전송)
- 1개의 Command List만으로도 처리는 가능. -> 성능 안나옴
- 멀티스레드로 여러 개의 Command List를 동시에 recording하고 각각의 스레드가 독립적으로 Execute하는 것을 권장.

Command Queue



Pipeline State – ID3D12PipelineState

- Blend State
- Depth State
- Render Target format
- Shaders…

이 모든 상태들을 하나로 묶어서 처리.

Shader 하나, Blend상태 하나 바꾸려고 해도 ID3D12PipelineState를 통째로 바꿔야함.

Shader 폭발에 이은 Pipeline State 폭발!

Pipeline State폭발의 슬픈 예

```
psoDesc.pRootSignature = m_pRootSignature;

D3D12_DEPTH_STENCIL_DESC    depthDescList[DEPTH_TYPE_NUM] = {};
SetDepthTypeDesc(depthDescList,_countof(depthDescList));

for (DWORD depth_type=0; depth_type<DEPTH_TYPE_NUM; depth_type++)
{
    psoDesc.DepthStencilState = depthDescList[depth_type];

    for (DWORD blend_index=0; blend_index<BLEND_TYPE_NUM; blend_index++)
    {
        psoDesc.BlendState = blendDesc[blend_index];

        for (DWORD shader_type=0; shader_type<VL_SHADER_TYPE_NUM; shader_type++)
        {
            for (DWORD param=0; param<SHADER_PARAMETER_COMBO_NUM; param++)
            {
                if (IsPhysique(param))
                {
                    psoDesc.InputLayout = { layoutVL_Physique, _countof(layoutVL_Physique) };
                }
                else
                {
                    psoDesc.InputLayout = { layoutVL, _countof(layoutVL) };
                }

                psoDesc.VS = CD3DX12_SHADER_BYTECODE(m_pVS[shader_type][param]->pCodeBuffer,m_pVS[shader_type][param]->dwCodeSize);
                psoDesc.PS = CD3DX12_SHADER_BYTECODE(m_pPS[shader_type][param]->pCodeBuffer,m_pPS[shader_type][param]->dwCodeSize);

                if (FAILED(pResourceManager->CreateGraphicsPipelineState(&psoDesc, &m_pPipelineState[depth_type][blend_index][shader_type][param])
                    __debugbreak();
            }
        }
    }
}
```

DX12로 기본적인 렌더링을 하기 위한 지식은 이 정도로...

Debugging

Debugging

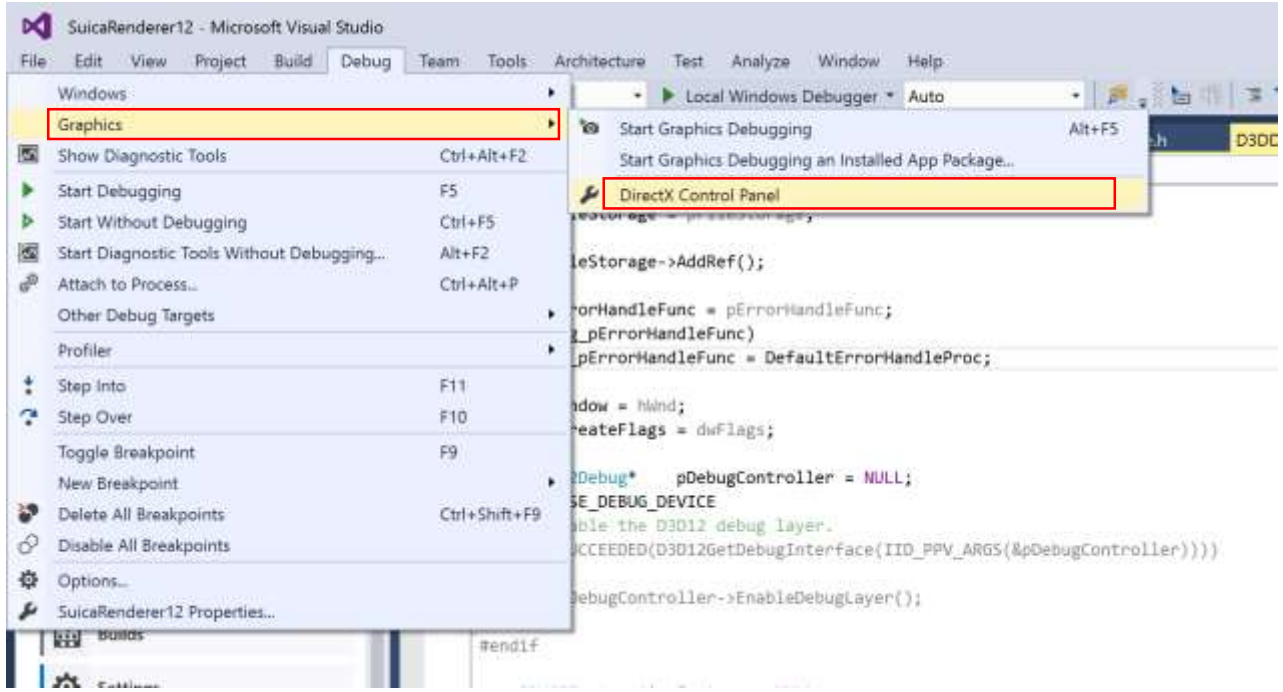
- 비동기 렌더링 특성상 API에 잘못된 파라미터를 전달해도 그 즉시 알기 어려움.
- DX12 runtime에 아직... 버그가 있는 것으로 보임.
- 각 업체별 GPU드라이버에도 아직 버그가 있는 것으로 보임.
- 그렇지만 대부분의 경우(99%) 프로그래머의 실수이다.
- 오동작하는것 같다면 내가 뭘 잘못했는지 먼저 체크하라.

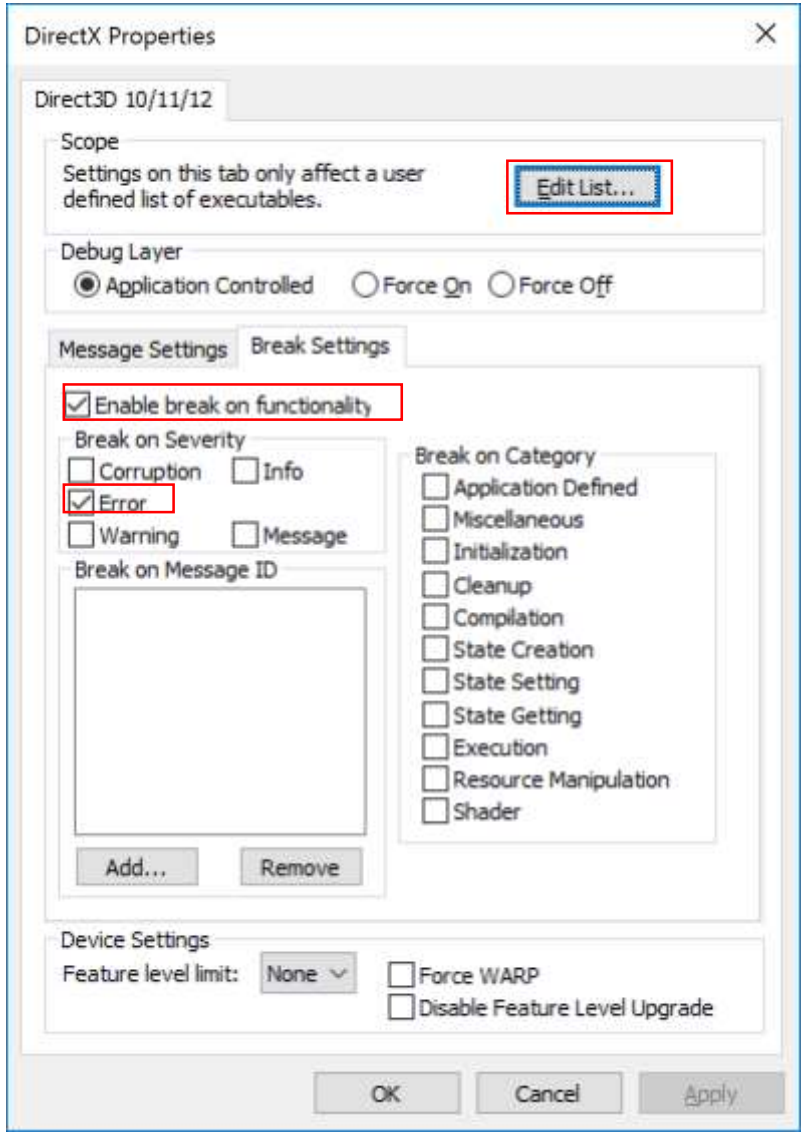
D3D12 Debug Layer의 사용

- 전달된 파라미터의 유효성, Resource Barrier의 상태 등등 프로그래머의 실수를 미리 잡아준다.
- 에러가 아닌데 에러라고 판별할 가능성도 없지는 않으나 반드시 무시하지 말고 체크할 것.

```
ID3D12Debug*pDebugController = NULL;

// Enable the D3D12 debug layer.
if (SUCCEEDED(D3D12GetDebugInterface(IID_PPV_ARGS(&pDebugController))))
{
        pDebugController->EnableDebugLayer();
}
```

DirectX Control Panel





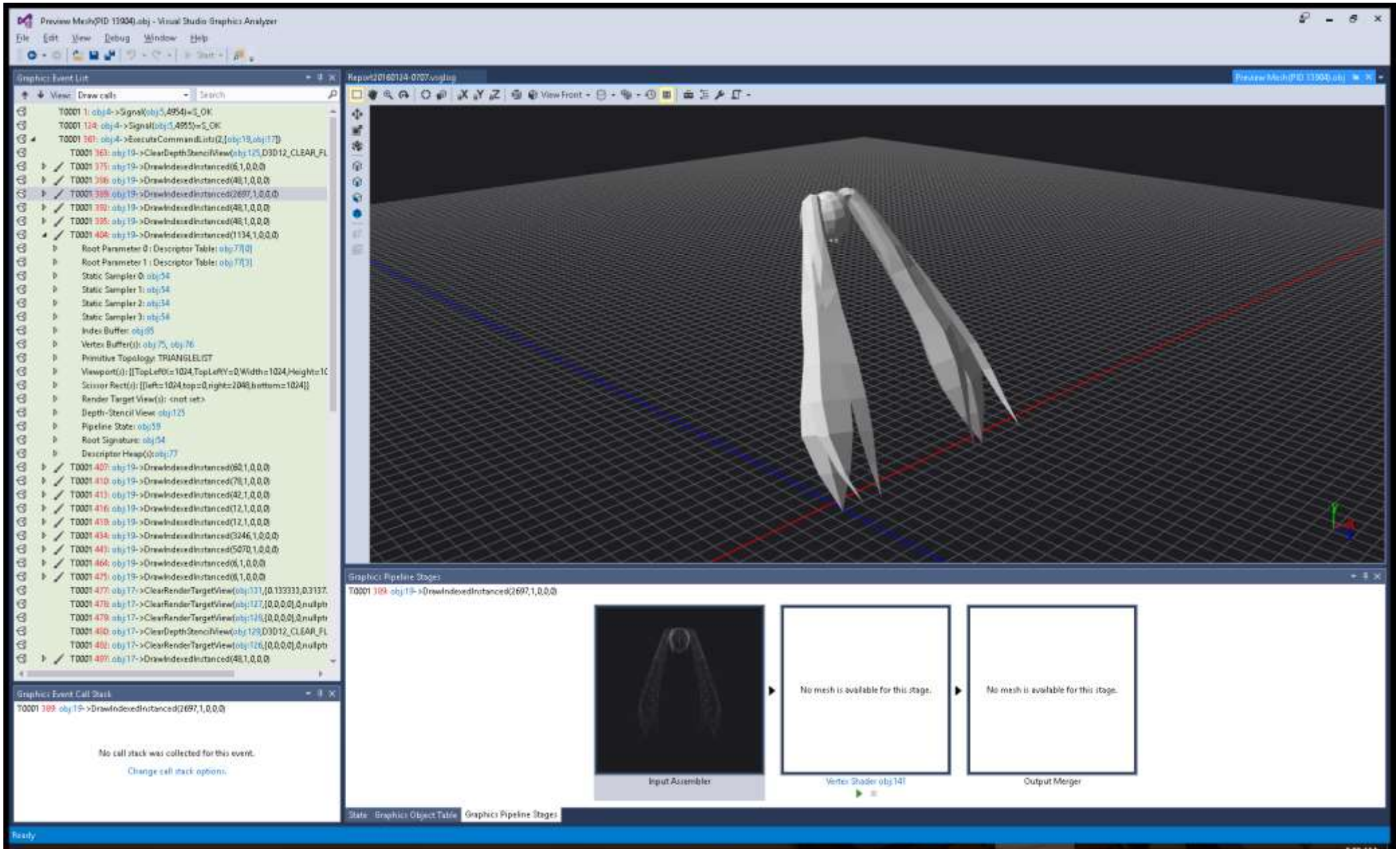
exe파일명을 등록시켜 놓으면 해당 app에서 Error나 Warning 발생시 브레이크를 걸 수 있다.

Microsoft

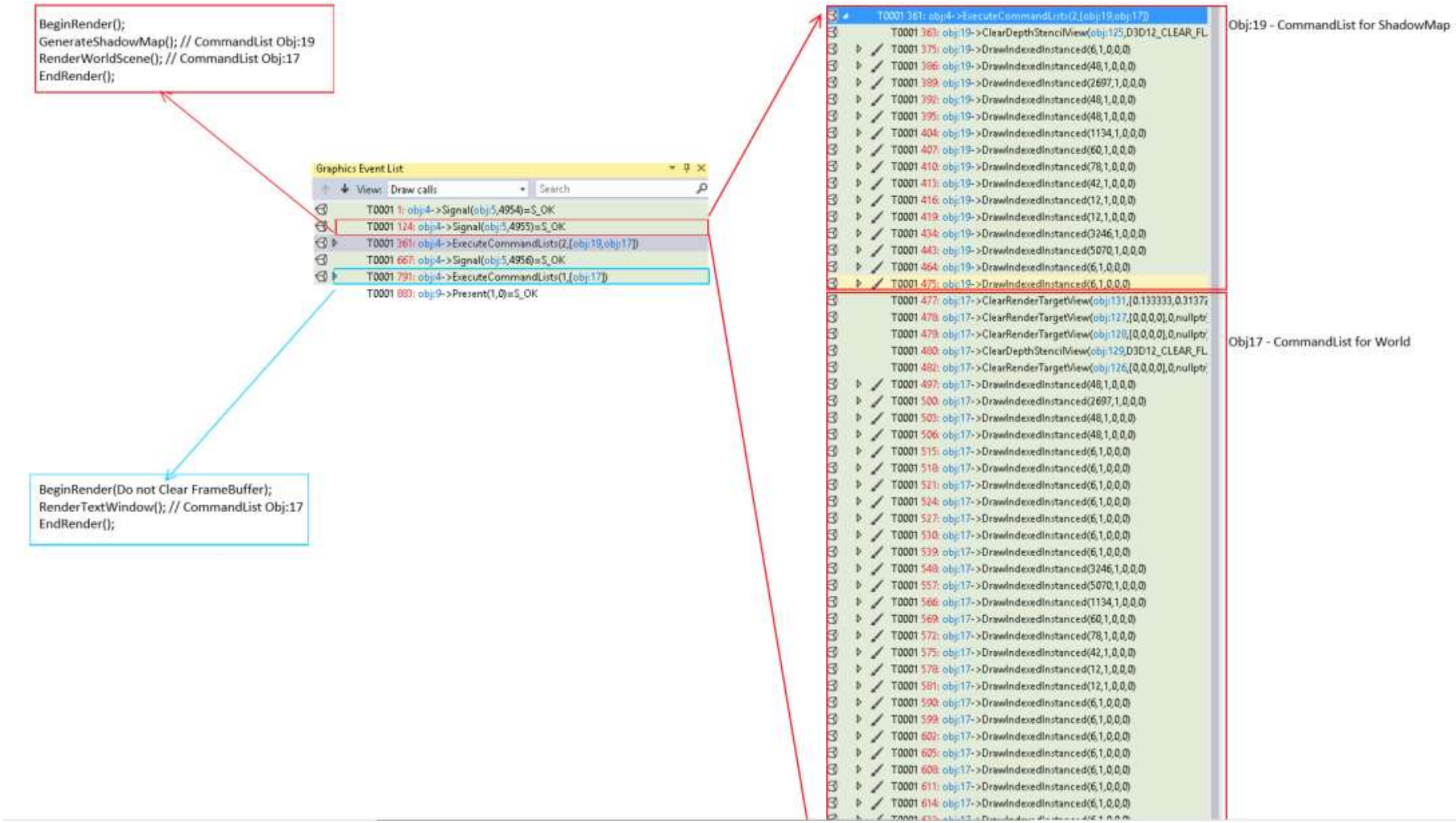
Visual Studio Graphics Debugger

- 프레임을 캡쳐해서 렌더링 과정을 보여줌.
- Pipeline에 바인드된 D3DResouce를 추적할 수 있다.
- Descriptor Table에 내용을 볼 수 있는 것이 큰 장점.
- 딱 잘라 말하건데 이 툴 없이는 개발 불가능.
- ALT+F5로 디버거 작동.

Shadow map 생성 과정을 디버깅 하는 예 (https://goo.gl/nC5f80 참고)



Shadow map 생성 과정을 디버깅 하는 예 (https://goo.gl/nC5f80 참고)



Optimization

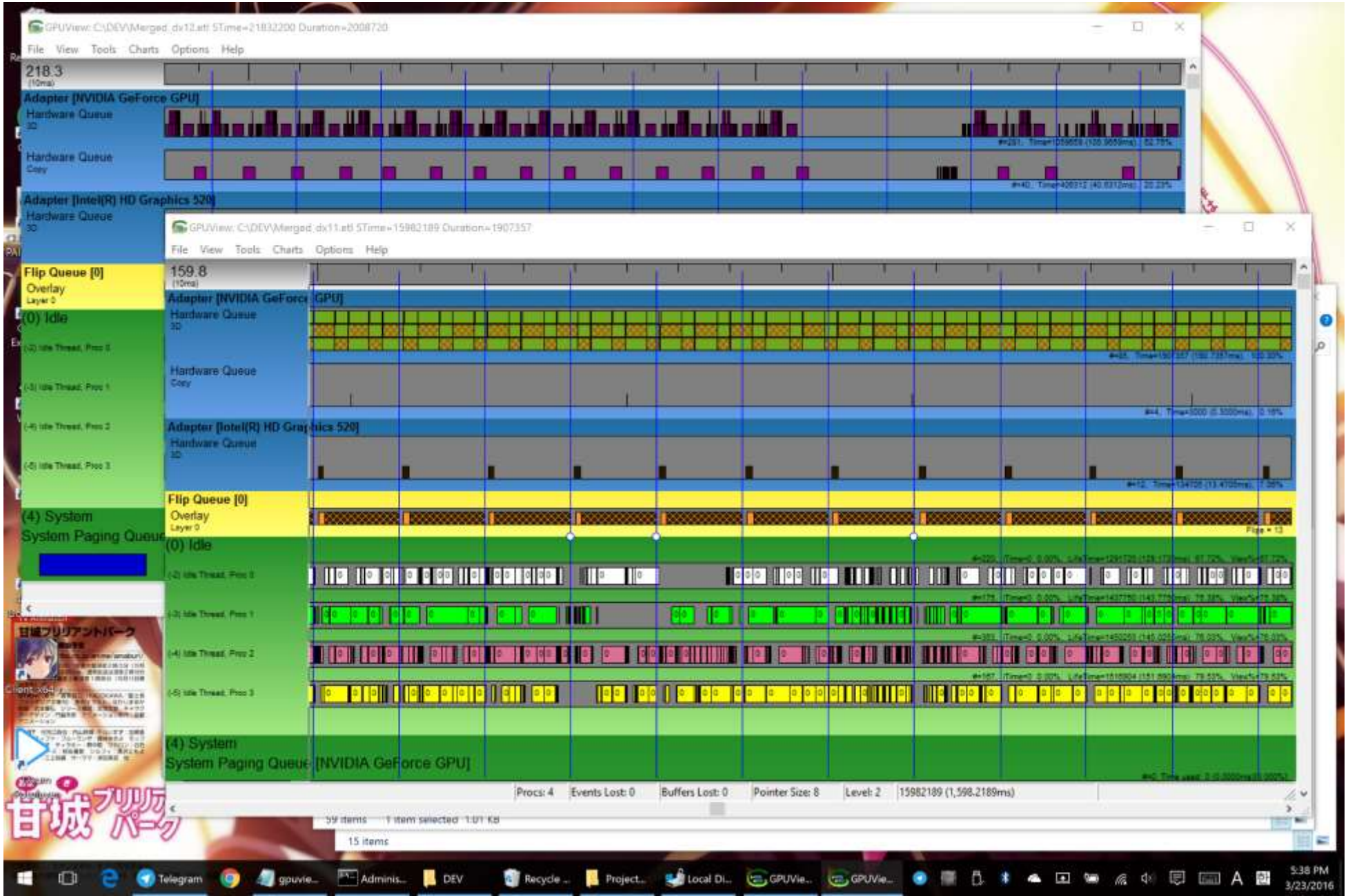
힘들게 포팅했는데 D3D11보다 느리네...

- 왜 느린가?
  - 앞서 설명했다. D3D11은 온갖 최적화 기법을 갖추고 있다.
  - 그에 비해 D3D12는 D3D11의 최적화 기법을 어플리케이션 레벨에서 구현해야한다.
  - 그러지 못하면 느리다...많이..
- 최적화의 시간 -> 최적화라 쓰고 다시 만든다고 읽는다.

GPUView

- Mircrosoft ADK의 Performance Toolkit에 포함된 로그 분석기
  1. ADK설치
  2. Admin권한으로 CMD을 열고
  - C:₩Program Files (x86)₩Windows Kits₩10₩Windows Performance Toolkit₩gpuview 폴더로 이동
  - >Log.cmd 로 logging시작
  - 다시 Log.cmd를 입력해서 logging종료
  - Merged.etl파일을 GPUView에서 로드

GPUView로 보는 D3D11 vs D3D12



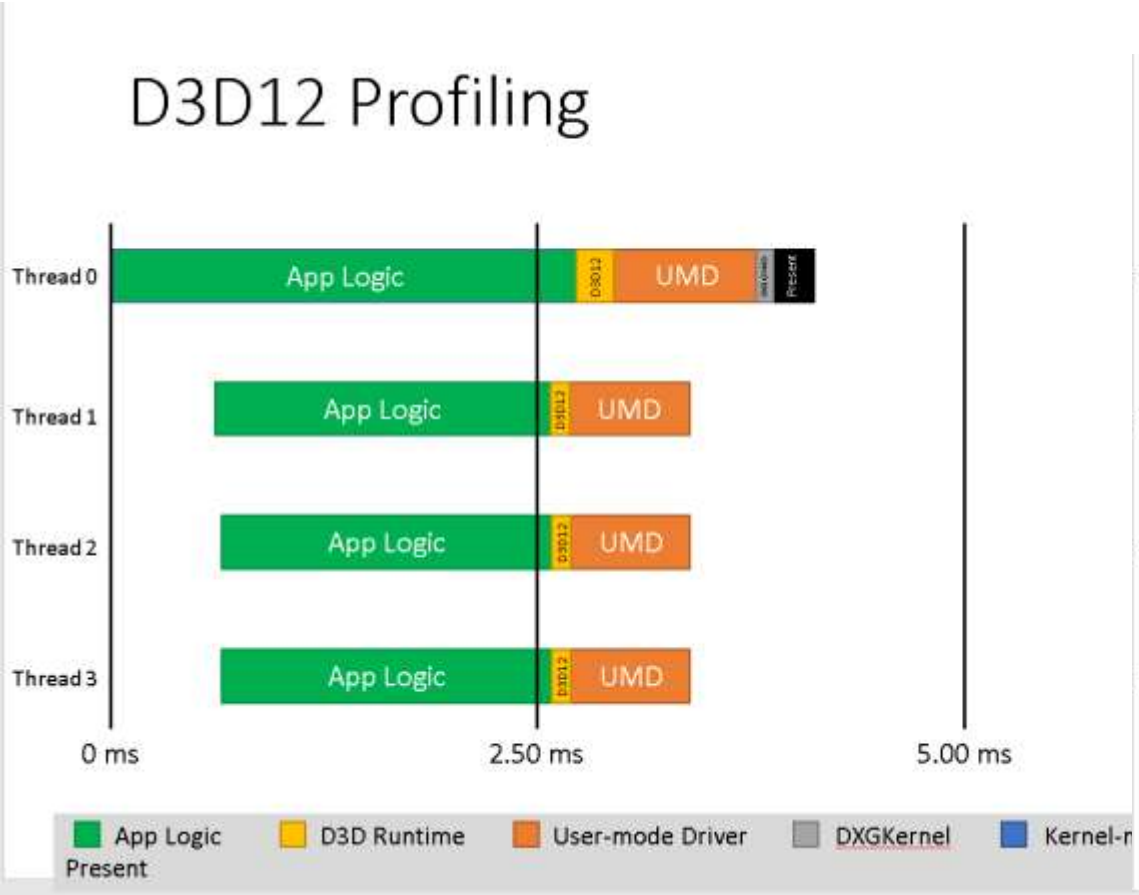
DirectX 11 , 과연 안정적인 스포츠카!!!

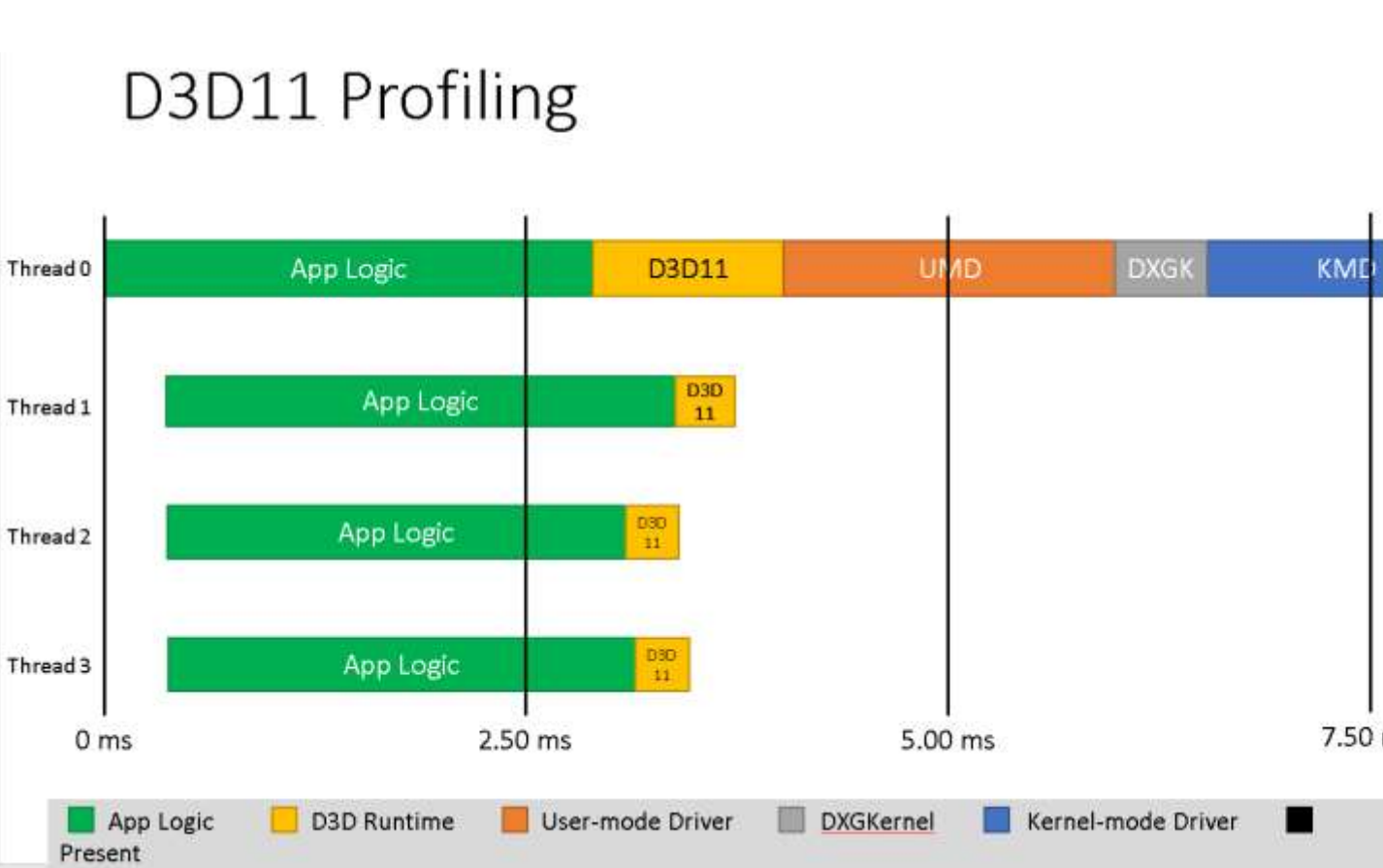
- Single-Thread Rendering
- 신경 쓴 것이라곤 GPU Memory에 최대한 올려놓고 안건드렸을 뿐 (avoid Map, Unmap)
- 심지어 D3DResource를 렌더링 중에 마구 변경해가며 재활용하고 있다. -> DX11은 Resource Renaming으로 이 문제를 해결한다
- 그럼에도 불구하고 DX11은 GPU Queue를 꽉꽉 채우며 최대한의 성능을 내주고 있다.

최적화할 수 있는 포인트는 뭐가 있을까?

- CPU -> Draw Call -> GPU까지의 과정이 짧아진건 명백한 사실.
  - 이 부분에선 확실히 성능향상이 있다.
- Command List 작성과 Execute의 적절한 배분.
  - 하나의 Command List에 몰아서 Command를 기록하고 마지막에 Execute 한번만 하면?
    - GPU가 펑펑 놀다가 마지막에 한번에 과부하를 받게 된다.
  - 여러 개의 Command List를 사용해서 Command기록과 Execute를 동시에 처리해야할 필요가 있다.
- Command List 작성과 Execute의 비동기 처리를 극대화하기 위해 Multi-Thread Rendering이 필요하다.

D3D12 runtime의 CPU처리가 훨씬 짧다.





From "Direct3D 12 API Preview" in Build 2014

D3D12 runtime의 CPU처리가 훨씬 짧다.

- 따라서 비슷한 정도의 GPU점유율이라면 D3D12 엔진이 더 빠를 것이다.
- GPU H/W Queue를 최대한 꽉 채우는 것이 성능 향상의 열쇠.

Execute호출 빈도 조절

- Execute()를 해야만 GPU H/W Queue로 Command전송
- D3D11엔진의 초기버전에선 Execute()를 present()직전에 한번만 호출했었다.
  - 그래서 Present 직전까지 GPU가 그냥 놀았다-_-;;;;;

Command List Pooling

- 여러 개의 Command List를 미리 할당해 둔다.
- Command List 하나당 N개의 오브젝트 렌더링에 대한 Command를 기록.
- N개에 도달하면 Execute(), 다음번 Command List에 계속해서 렌더링 Command 기록.
- 모든 오브젝트를 다 렌더링하거나 할당해 둔 Command List를 다 사용할 때까지 반복.

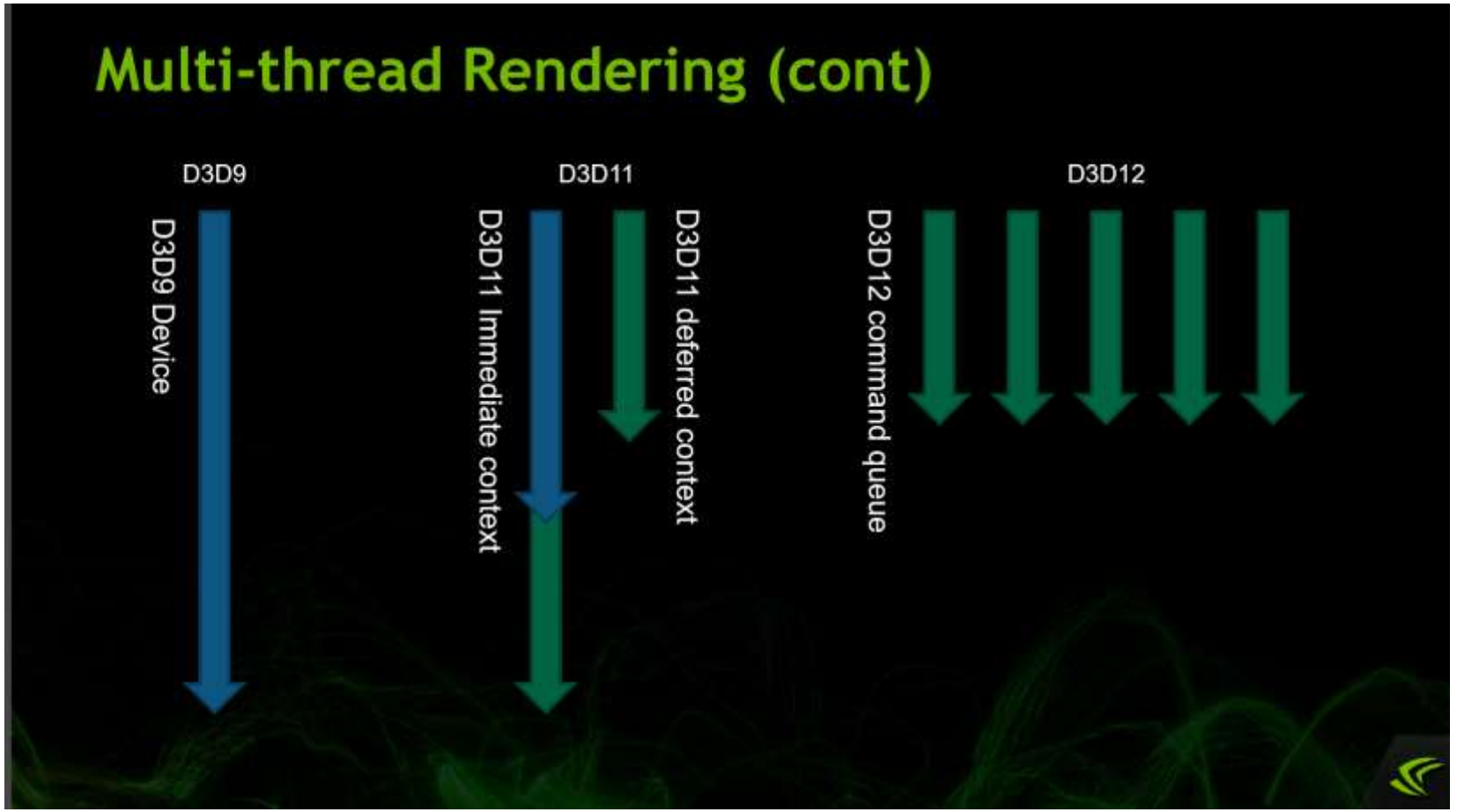
Command List Pooling

- 성능이 상당히 개선되었다.
- 그러나 아직도 부족하다.

Multi-Thread Rendering

- Multi-Thread로 Command 작성 시간을 최소화시킨다.
- 단순계산으로 n이란 시간이 걸린다면 4 thread를 사용하면 시간을 n/4로 줄일 수 있다.
- GPU Queue가 empty 되지 않도록 최선을 다한다.

D3D12에서의 Multi-Thread Rendering

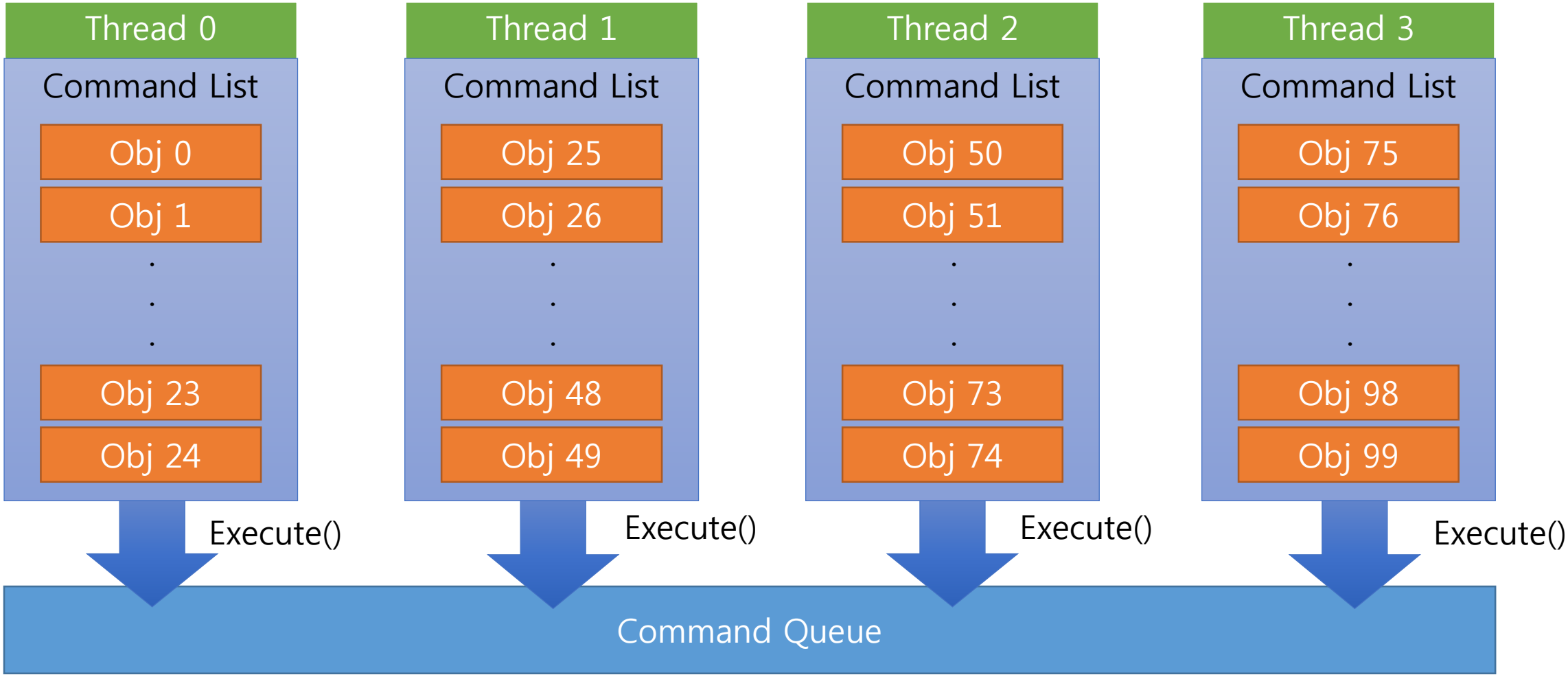


From "Approaching Minimum Overhead with Direct3D12" NVIDIA

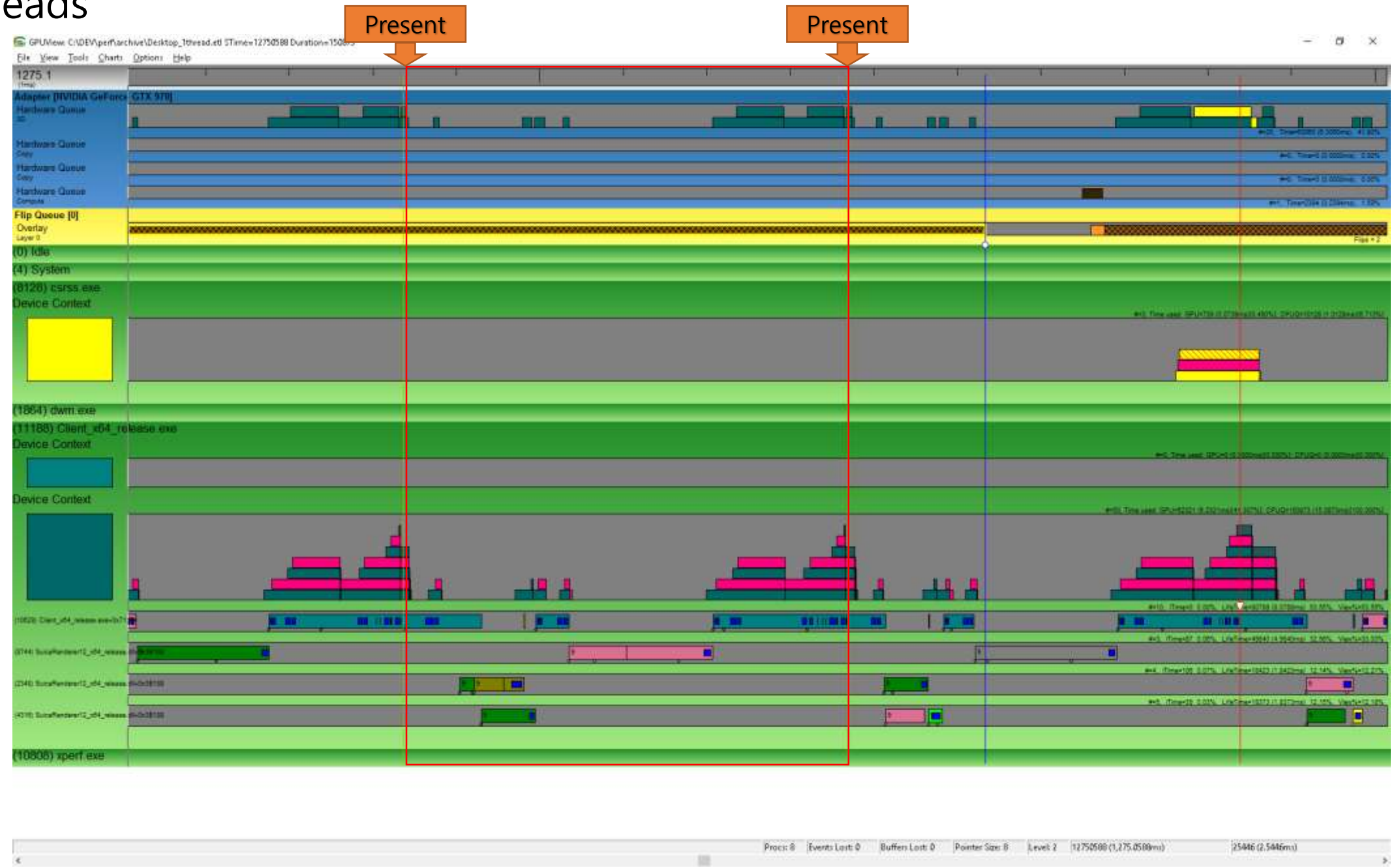
Multi-Thread Rendering 구현

- 스레드 1개당 Command List하나씩 배당
- 엔진 내의 Queue에 담긴 오브젝트들을 각 스레드에 균등하게 배분
- 각 스레드는 자신의 Command List에 오브젝트들의 렌더링 Command를 기록
- 마지막 오브젝트까지 처리하고 난 후 Execute()

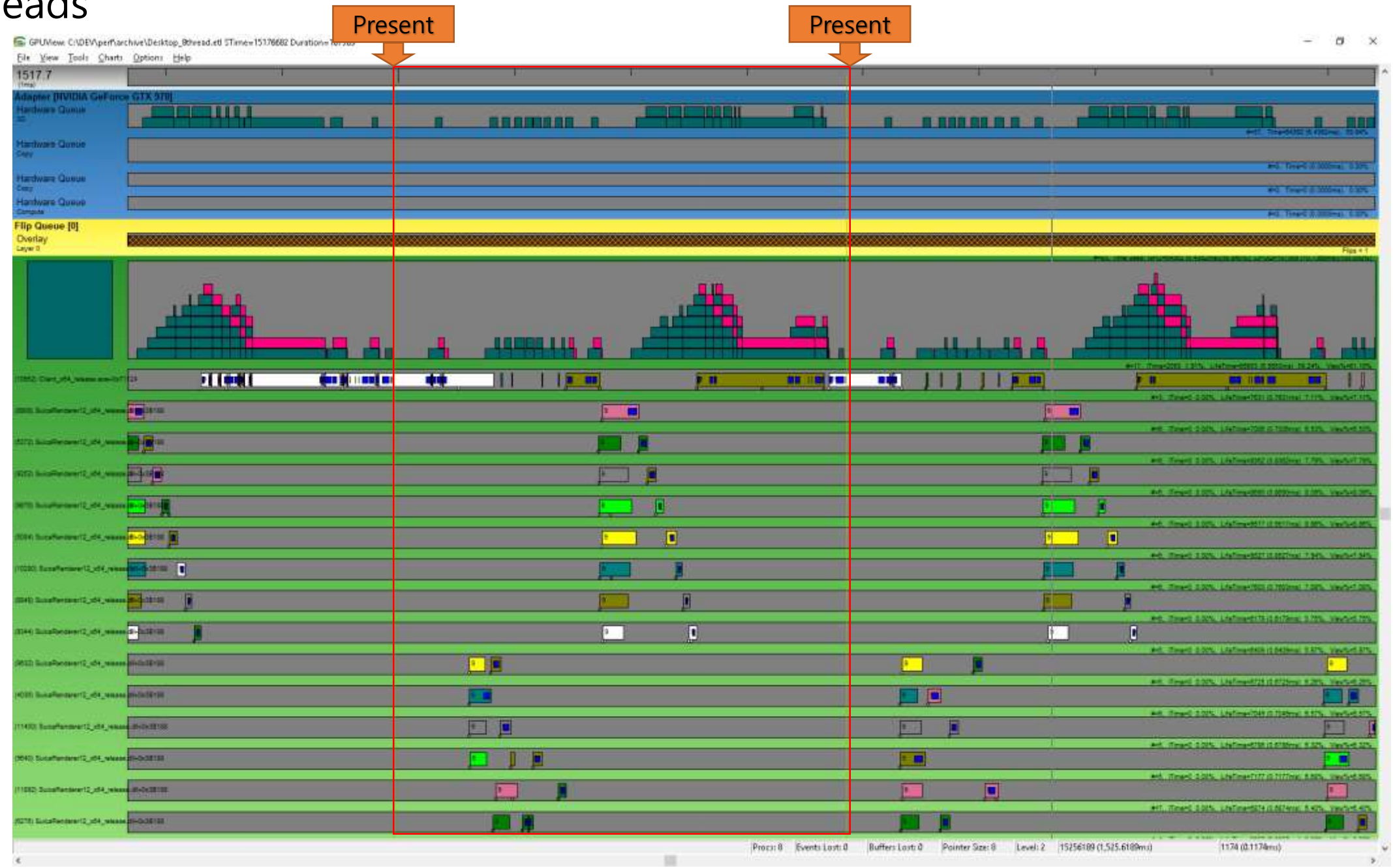
Multi-Thread Rendering 구현



1 Threads


Present
Present
File View Tools Charts Options Help
1275.1
Hardware Queue
Hardware Queue
Hardware Queue
Hardware Queue
Flip Queue [0]
Overlay
(0) Idle
(4) System
(8128) csrss.exe
Device Context
(1864) dwm.exe
(11188) Client_x64_release.exe
Device Context
Device Context
(10808) xperf.exe

8 Threads


Present
Present
File View Tools Charts Options Help
1517.7
Hardware Queue
Hardware Queue
Hardware Queue
Hardware Queue
Flip Queue [0]
Overlay

성능비교

(약을 팔진 않겠습니다..정직한 자료....)

@Surface Book i5 nvidia dGPU

Res. 3000x2000	D3D11	D3D12	
10 Characters	25 fps	28 fps	11%
100 Characters	16 fps	19 fps	16%

Demo

결론

- D3D12로 포팅만 하면 11보다 빠를거라고 생각하면 큰 착각.
- 45프레임 나오는 게임을 60프레임 만들고 싶을 때 사용한다.
- 실질적으로 기대할 수 있는 성능향상폭은 10% - 30%정도.
- 차세대 API는 다 비슷한데 그래도 D3D12가 가장 성능이 잘 나오는 편.
- DirectX Runtime, Driver의 업데이트로 성능 향상 가능성이 약간 있음.
- D3D12 API에 더 익숙해지면 조금 더 빨라지긴 할것임.

FAQ

- DirectX 12를 사용 가능한 OS및 디바이스는?
  - Windows 10 PC, XBOX ONE, Windows 10 Mobile (Snapdragon 808이상 장착한 폰에 한해서, Lumia950/950XL)
- Direct Write와 Direct2D를 쓰고 싶어요.
  - D3D11 on D2D를 사용하세요.
- D3D12 API는 Thread-safe한가요?
  - 네. Thread-safe합니다. 별도의 lock을 걸어줄 필요가 없습니다.

Reference

- yuchi dev D3D tag
- Approaching Minimum Overhead with Direct3D12
- **Direct3D 12 API Preview**
- Efficient Rendering with DirectX 12 on Intel Graphics
- DIRECTX ADVANCEMENTS IN THE MANY CORE ERA Getting the most out of the PC Platform
- Using GPUView to Understand your DirectX 11 Game