
D3D11詳解
新しいDirectXがもたらす
イノベーションとは？
NVIDIA
風間 隆行
NVIDIA

D3D11 Overview

- D3D11はD3D10に多くの改良追加が施されたバージョン
  - 追加
    - Compute Shader
    - Tessellation (Domain/Hull Shader)
    - Multithreading (Immediate/Deferred context)
    - Dynamic Shader Linkage
  - 改良
    - Shader Model 5
    - Texture
    - Large memory
    - Conservative oDepth
    - Stream output
    - Read-only depth
    - …etc

D3D11に移行する７つの理由

- マーケティング面
  1. 対応OS
  2. 対応ハードウェア
- プログラミング面
  3. 移行のしやすさ
  4. 自由度と管理のしやすさの両立
  5. パフォーマンス
- アーティスティック面
  6. 制限の少なさ
  7. 表現の豊富さ

D3D11に移行する７つの理由

- マーケティング面
  1. 対応OS
  2. 対応ハードウェア
- プログラミング面
  3. 移行のしやすさ
  4. 自由度と管理のしやすさの両立
  5. パフォーマンス
- アーティスティック面
  6. 制限の少なさ
  7. 表現の豊富さ

Windows7 & Vista

D3D11に移行する７つの理由

- マーケティング面
  1. 対応OS
  2. 対応ハードウェア

D3D11
D3D10.1
D3D10
D3D9.3
D3D9.2
D3D9.1

FeatureLevel

- プログラミング面
  3. 移行のしやすさ
  4. 自由度と管理のしやすさの両立
  5. パフォーマンス
- アーティスティック面
  6. 制限の少なさ
  7. 表現の豊富さ

D3D11に移行する７つの理由

- マーケティング面
  1. 対応OS
  2. 対応ハードウェア
- プログラミング面
  3. 移行のしやすさ
  4. 自由度と管理のしやすさの両立
  5. パフォーマンス
- アーティスティック面
  6. 制限の少なさ
  7. 表現の豊富さ

D3D10 → D3D11
ID3D10 → ID3D11
D3DX10 → D3DX11

D3DCompiler
Device & Context
Flag & Structure

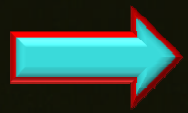
D3D11に移行する７つの理由

- マーケティング面
  1. 対応OS
  2. 対応ハードウェア
- プログラミング面
  3. 移行のしやすさ
  4. 自由度と管理のしやすさの両立
  5. パフォーマンス
- アーティスティック面
  6. 制限の少なさ
  7. 表現の豊富さ

ComputeShader
Multithread
DynamicShaderLinkage
Unordered Accesses

D3D11に移行する７つの理由

- マーケティング面
  1. 対応OS
  2. 対応ハードウェア
- プログラミング面
  3. 移行のしやすさ
  4. 自由度と管理のしやすさの両立
  5. パフォーマンス
- アーティスティック面
  6. 制限の少なさ
  7. 表現の豊富さ



ReadOnly Depth
Conservative oDepth
PS Input Coverage
SO Improvement

D3D11に移行する７つの理由

- マーケティング面
  1. 対応OS
  2. 対応ハードウェア
- プログラミング面
  3. 移行のしやすさ
  4. 自由度と管理のしやすさの両立
  5. パフォーマンス
- アーティスティック面
  6. 制限の少なさ
  7. 表現の豊富さ

Large Memory > 4GB
16K Texture Limit

D3D11に移行する７つの理由

- マーケティング面
  1. 対応OS
  2. 対応ハードウェア
- プログラミング面
  3. 移行のしやすさ
  4. 自由度と管理のしやすさの両立
  5. パフォーマンス
- アーティスティック面
  6. 制限の少なさ
  7. 表現の豊富さ

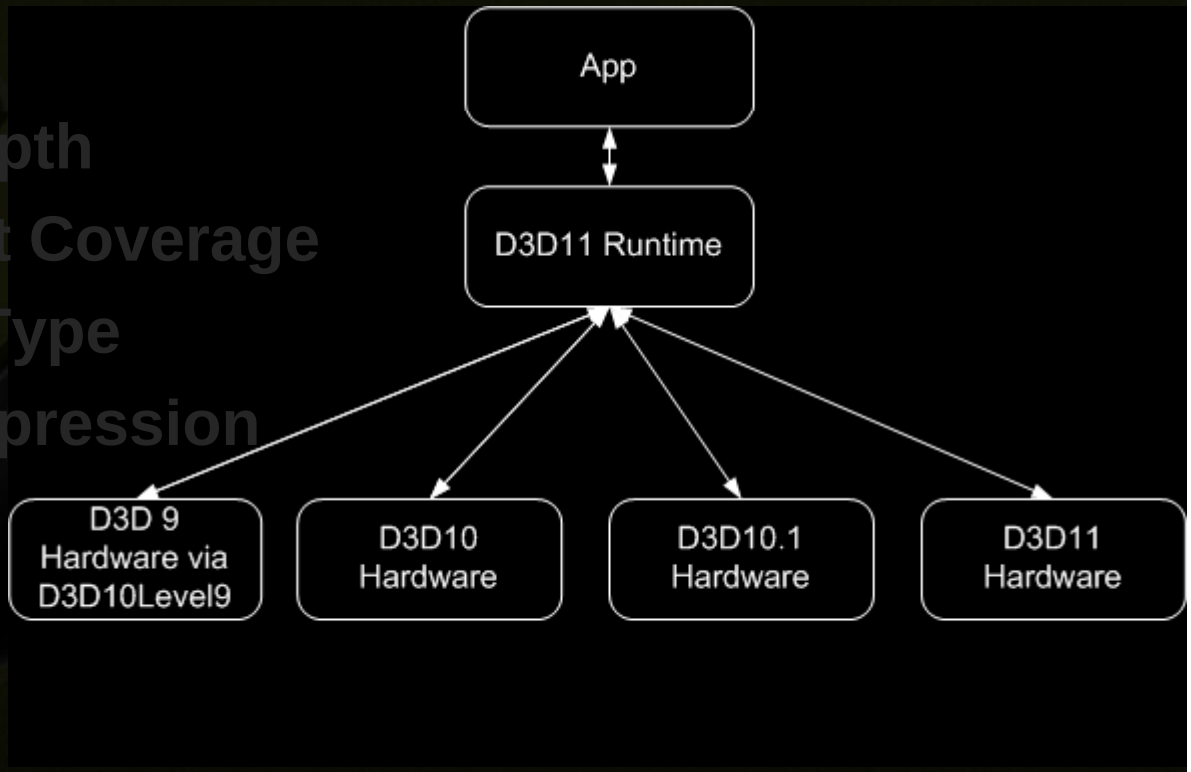
Tessellation
New TextureCompression

機能詳細

- FeatureLevel
- MultiThread
- Dynamic Shader Linkage
- Read Only Depth
- Conservative oDepth
- Pixel Shader Input Coverage
- Pixel Shader Unordered Accesses
- New Texture Compression

機能詳細：FeatureLevel

- **FeatureLevel**
- MultiThread
- Dynamic Shader Linkage
- Read Only Depth
- Conservative oDepth
- Pixel Shader Input Coverage
- New View/Buffer Type
- New Texture Compression



機能詳細：FeatureLevel

```
HRESULT hr = E_FAIL;
D3D_FEATURE_LEVEL FeatureLevel;

hr = D3D11CreateDevice(NULL, D3D_DRIVER_TYPE_HARDWARE, NULL, 0, NULL, 0,
                       D3D11_SDK_VERSION, NULL, &FeatureLevel, NULL);

if (FAILED(hr))
{
        return hr;
}
```

- D3D11CreateDeviceもしくはD3D11CreateDeviceAndSwapChainでFeatureLevelを得る
- 関数に渡す ppDeviceにはNULLを指定
- Deviceを作成後に ID3DDevice::GetFeatureLevel()で確認することも出来る

```
D3D_FeatureLevel = {    D3D_FEATURE_LEVEL_11_0,
                        D3D_FEATURE_LEVEL_10_1,
                        D3D_FEATURE_LEVEL_10_0,
                        D3D_FEATURE_LEVEL_9_3,
                        D3D_FEATURE_LEVEL_9_2,
                        D3D_FEATURE_LEVEL_9_1       }
```

機能詳細：FeatureLevel

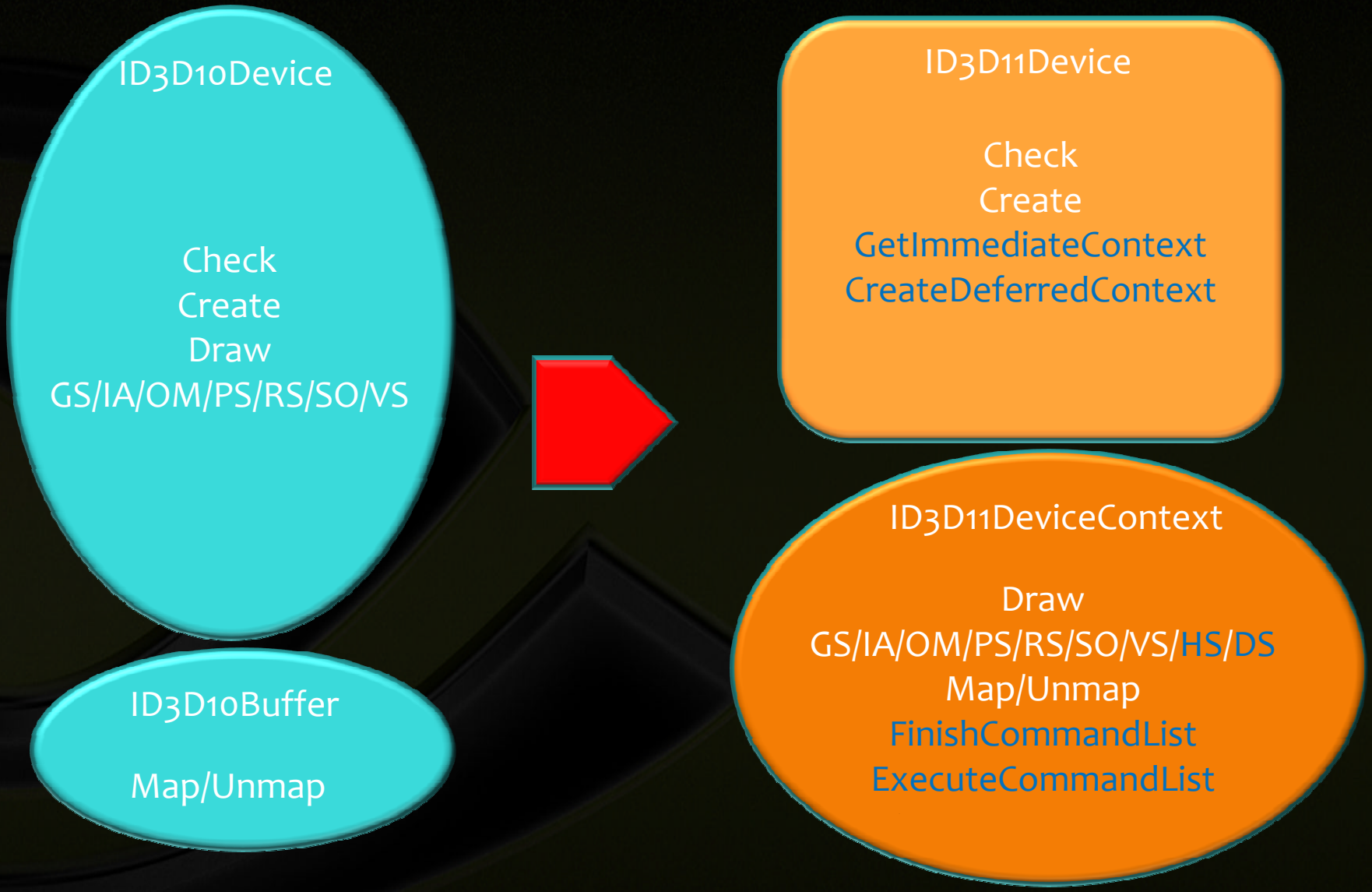
	11_0	10_1	10_0	9_3
ShaderModel	5.0	4.X	4.0	2.0(4_0_level_9_3)
Geometry Shader	Yes	Yes	Yes	No
Compute Shader	Yes	Optional	Optional	No
Tessellation	Yes	No	No	No
New TexCompression	Yes	No	No	No
MRT	8	8	8	4
Texture Limit	16K	8K	8K	8K

etc…

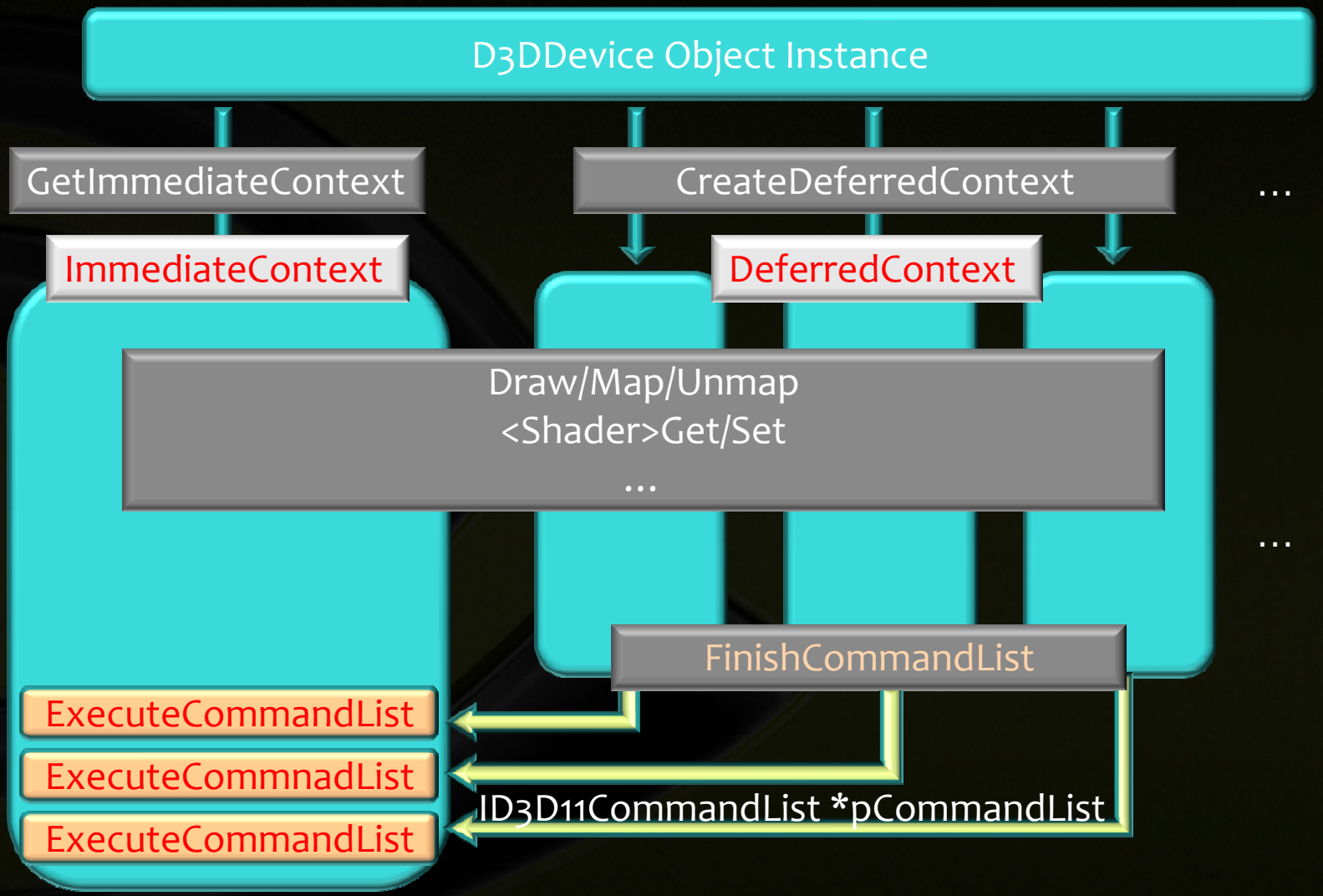
機能詳細：MultiThread

- FeatureLevel
- **MultiThread**
- Dynamic Shader Linkage
- Read Only Depth
- Conservative oDepth
- Pixel Shader Input Coverage
- New View/Buffer Type
- New Texture Compression

機能詳細：MultiThread



機能詳細：MultiThread



機能詳解：MultiThread

contextの作成　　Main Thread、初期化時

```
pd3dDevice->GetImmediateContext(&MyImmediateContext);          // 一つだけ!

for (i = 0;  i < iNumThread;  ++i) {                           // 好きなだけ
   pd3dDevice->CreateDeferredContext(0,  &MyDeferredContext[i]);
   thread[i] = _beginthreadx( …. );
}
```

CommandListの作成　　各Thread (注: MyImmediateContextも使用可)

```
MyDeferredContext[id]->ClearRenderTargetView(pRTV,  ClearColor);
                      …. // コマンド (Draw, Map/Unmap, Shaders … )
MyDeferredContext[id]->FinishCommandList(FALSE, &MyCommandList[id]);
SetEvent(hEvent[id]);
```

実行　　Main Thread

```
WaitForMultipleObjects(iNumThread,  hEvent, TRUE, INFINITE);
for (i = 0;  i < iNumThread;  ++i) {
   MyImmediateContext->ExecuteCommandList(MyCommandList[i],  FALSE);
   MyCommandList[i]->Release();   // or SAFE_RELEASE
}
```

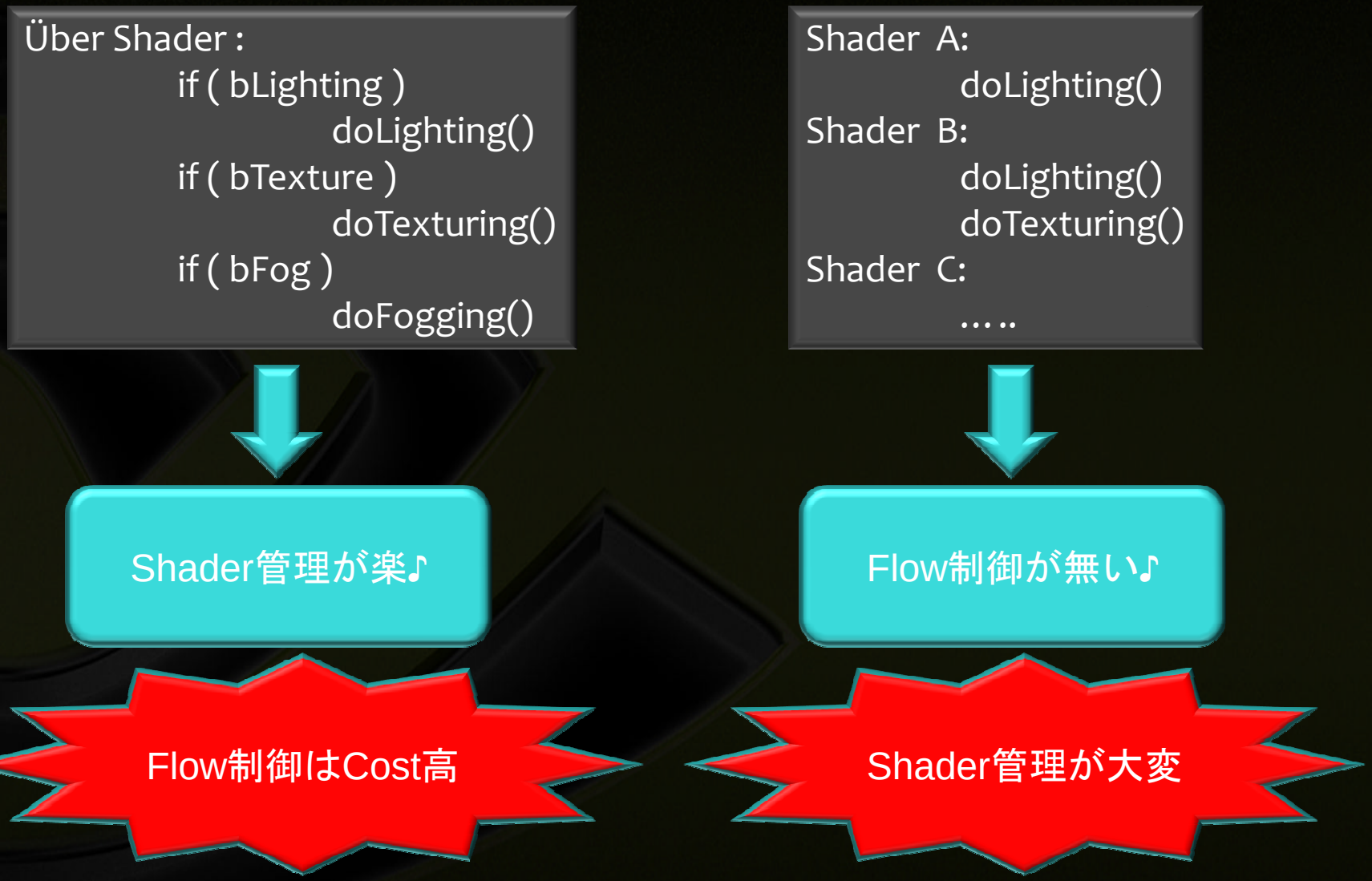
機能詳解：MultiThread

- ID3DDeviceとID3DDeviceContextに分離
- ImmediateContext一つとDeferredContextを複数
- 各ContextはThreadSafe
- Map/Unmap, CopyResource, UpdateSubresourceがDeviceContext側に移ったことによりリソースマネージメントがより柔軟に

機能詳細：Dynamic Shader Linkage

- FeatureLevel
- MultiThread
- **Dynamic Shader Linkage**
- Read Only Depth
- Conservative oDepth
- Pixel Shader Input Coverage
- New View/Buffer Type
- New Texture Compression

機能詳解：Dynamic Shader Link



機能詳解：Dynamic Shader Link

- Shaderの管理が楽で、かつFlow制御をなくすには独自のシェーダジェネレータ等を作るしかなかった

- DX11ではHLSLにInterfaceとClassが導入される
  - Shader側　　Interface & Class
  - Application側　　ClassLinkage & New Reflection

分岐コストの無いÜber Shaderが作成可能に！

機能詳解：Dynamic Shader Link

```
interface  iLight  {
  float4  Calculate(…);
};

class  cAmbient  :  iLight {
  float4   m_Ambient;
  float4  Calculate(…) {
    return   m_Ambient;
  }
};

class   cDirectional  : iLight {
    float4   m_Dir;
    float4   m_Col;
    float4   Calculate(…) {
      float   ratio = satulate(dot(…));
      return   m_Col * ratio;
};
```

```
iLight   g_Lights[4];
cbuffer  cbData  : register( b0 ) {
  cAmbient     g_Ambient
  cDirectional   g_Directional0;
  cDirectional   g_Directional1;
  cDirectional   g_Directional2; }
float  accumlateLights(…) {
  …
  for (uint   i =  0;  i  <  g_NumLights; … ) {
    col += g_Lights[i].Calculate(….);
  }
…
}
```

呼ばれるか決定されます

機能詳解：Dynamic Shader Link

```
cbuffer cbData : register( b0 ) {
   cAmbient      g_Ambient
   cDirectional  g_Directional0;
   cDirectional  g_Directional1;
   cDirectional  g_Directional2;
 }
```

Shaderで定義した
Instance

lication側でClass Instance
mber変数を順番に定義
する必要がある

継承した場合
に注意

```
class cA : iBase {
   float4   valueA;
};
class cB : cA {
   float4   valueB;
};
cBuffer cbData : register(b0) {
   cA    g_A0;
   cB    g_B0;
   cB    g_B1;
};
```

```
Struct CB_PS_DATA {
   D3DXVECTOR4  m_valueA_A0;
   D3DXVECTOR4  m_valueA_B0;
   D3DXVECTOR4  m_valueB_B0;
   D3DXVECTOR4  m_valueA_B1;
   D3DXVECTOR4  m_valueB_B1;
};
```

```
p);

CTOR4( .... );
pMyPSData->m_Direction0 = D3DXVECTOR4(....);
...
MyContext->Unmap( ... );
```

機能詳解：Dynamic Shader Link

Linkage Objectの作成

```
pd3dDevice->CreateClassLinkage( &classLinkage);
```

Shaderの作成

```
pd3dDevice->Create○○Shader(…, classLinkage, … );
```

ル化するためのオブジェクト

Reflectionの取得

```
D3DReflect(ShaderBuffer, ShaderSize, …,  &ref);
```

めCreate○○Shader()に渡される

Interface数の

でシェーダデータにアク

めにReflectionを得る

```
col += g_Lights[i].Calculate(….);
```

Shader内での iLightの呼び出しが
実際にどのクラスインスタンスで呼ばれるかはApplicationサイドで与える

Linkage Array

```
num );
```

にあるInterfaceの数を
取得します

Interface Slot

ights[4]：４つ

```
Slot(o);
```

Interface Slotのoffsetを取得します
これはLinkage Arrayでの

Class Instanceの取得

```
classLinl
```

```
stance)
```

Class Instaceを名前引きで取得します

InterfaceにClass Instaceをset

```
LinkArra[   ]
```

Interfaceがどの実Instanceと結びつく
定します

```
myContext->○○SetShader(Shader, LinkArray, num );
```

機能詳解：Dynamic Shader Link

Dynamic Linkなんて
Performance大丈夫？？

```
interface Color {
   float4 Get();
};

class Red : Color {
   float4 Get() { return float4(1, 0, 0, 1); }
};

class Blue : Color {
   float4 Get() { return float4(0, 0, 1, 1); }
};

Color color;

float4 main() : SV_Target
{
   return color.Get();
}
```

```
ps_5_0
dcl_globalFlags refactoringAllowed
dcl_function_body fb0
dcl_function_body fb1
dcl_function_table ft0 = {fb0}
dcl_function_table ft1 = {fb1}
dcl_interface fp0[1][1] = {ft0, ft1}
dcl_output o0.xyzw
dcl_temps 1
fcall fp0[0][0]
mov o0.xz, r0.xxyx
mov o0.yw, l(0,0,0,1.000000)
ret
label fb0
mov r0.xy, l(0,1.000000,0,0)
ret
label fb1
mov r0.xy, l(1.000000,0,0,0)
ret
```

間接function call

機能詳細：Read Only Depth

- FeatureLevel
- MultiThread
- Dynamic Shader Linkage
- **Read Only Depth**
- Conservative oDepth
- Pixel Shader Input Coverage
- New View/Buffer Type
- New Texture Compression

機能詳細：Read Only Depth

ShaderでDepth/StencilをReadしたいし、DepthTestも行いたい
例: Soft Particle

ところが・・・

DX10ではRead/Write Hazard回避の為に
同一場所にRead/WriteのResource設定をしていた場合、
Read側がunboundされていた

DX11なら！

Depth/Stencil BufferをOM(Output Merger)で
”Read-Only”としてBound可能に

DirectX RuntimeのSystemがNo Hazardである事を認識できるので
Shader Input とDepth-Testが同時に行えるようになりました

機能詳細：Read Only Depth

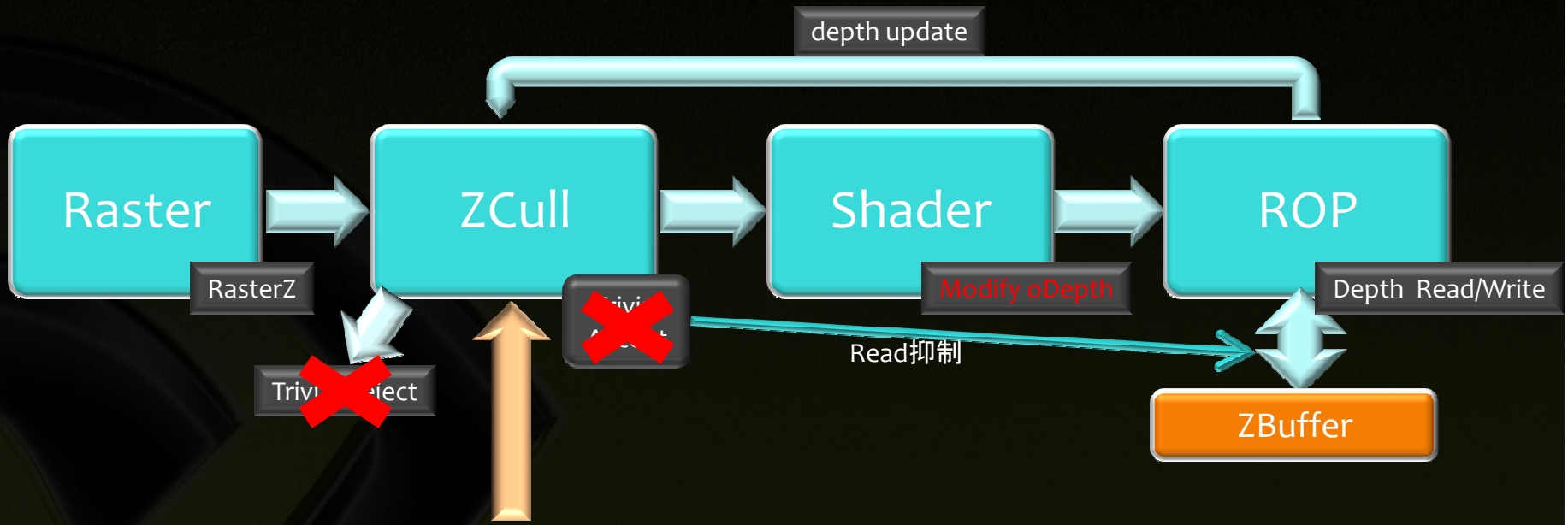
```
#define   D3D11_DSV_FLAG_READ_ONLY_DEPTH          0x1;
#define D3D11_DSV_FLAG_READ_ONLY_STENCIL          0x2;
```

```
typdef  struct  D3D11_DEPTH_STENCIL_VIEW_DESC
{
    DXGI_FORMAT                         Format;
    D3D11_DSV_DIMENSION                 ViewDimension;
    DWORD                               Flags;
    union
    {
        D3D11_TEX1D_DSV                 Texture1D;
        D3D11_TEX1D_ARRAY_DSV           Texture1DArray;
        D3D11_TEX2D_DSV                 Texture2D;
        D3D11_TEX2D_ARRAY_DSV           Texture2DArray;
        D3D11_TEX2DMS_DSV               Texture2DMS;
        D3D11_TEX2DMS_ARRAY_DSV         Texture2DMSArray;
     };
}  D3D11_DEPTH_STENCIL_VIEW_DESC ;
```

機能詳細：Conservative oDepth

- FeatureLevel
- MultiThread
- Dynamic Shader Linkage
- Read Only Depth
- **Conservative oDepth**
- Pixel Shader Input Coverage
- New View/Buffer Type
- New Texture Compression

機能詳細： Conservative oDepth



RasterZがShaderの出力Depthと違う
ZCull段階での比較が出来なくなる
常にShaderが動き、常にDepth ReadがROPで発生

Performance Down!!!

ところがShaderがZ値を書き換えると・・・

機能詳細： Conservative oDepth

新しいoutput System Valueが導入
SV_DepthGreaterEqual
SV_DepthLessEqual

```
struct  PS_OUTPUT {
    float4   color  : SV_TARGET;
    float    depth : SV_DepthGreaterEqual;
};
```

LESS or LESS_EQUALの場合

RasterZ以下のDepthを
出力することをShaderが保障

RasterZ以上のDepthを
出力することをShaderが保障

Depth値
Shader Output Depth
Raster Output

RasterZは明らかにAcceptなのでoDepthもAccept

RasterZは明らかにRejectなのでoDepthもReject

Depth-Test方向:

LESS or LESS_EQUAL

出力先	SV_Depth	: ZCullは動かない
	SV_DepthGreaterEqual	: Trivial Reject可能
	SV_DepthLessEqual	: Trivial Accept可能

GREATER or GREATER_EUQAL

出力先	SV_Depth	: ZCullは動かない
	SV_DepthGreaterEqual	: Trivial Accept可能
	SV_DepthLessEqual	: Trivial Reject可能

機能詳細：PixelShader Input Coverage

- FeatureLevel
- MultiThread
- Dynamic Shader Linkage
- Read Only Depth
- Conservative oDepth
- **Pixel Shader Input Coverage**
- New View/Buffer Type
- New Texture Compression

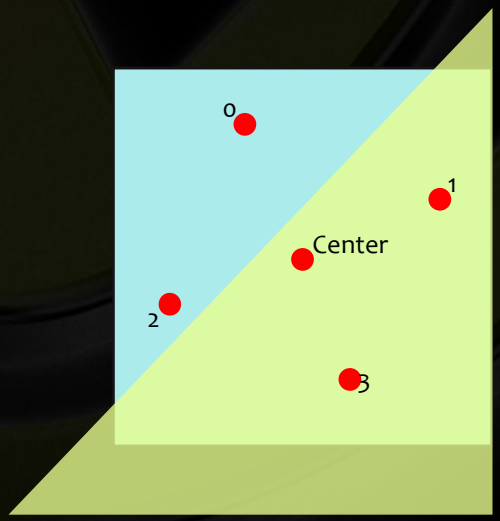
機能詳細：PixelShader Input Coverage

Pixel ShaderにSystem ValueとしてInput Coverageが導入されました

SV_Coverage：32bitスカラー

SV_Coverageはサンプル毎にPrimitiveがCoverしているかを示すbitで構成されます

例: MSAA 4xの場合



SV_Coverage := 0b1010

Use Case：Edge検出

```
IsEdge = (SV_Coverage != 0) &&
         (SV_Coverage != 0xf);
```

機能詳細：New View/Buffer Type

- FeatureLevel
- MultiThread
- Dynamic Shader Linkage
- Read Only Depth
- Conservative oDepth
- Pixel Shader Input Coverage
- **New View/Buffer Type**
- New Texture Compression

機能詳細：New View/Buffer Type

D3D11には新しいResource Viewが導入されます

- RTV ：Render Target View
- DSV ：Depth-Stencil View
- SRV ：Shader Resource View
- UAV ：Unordered Access View

PS5.0かCS5.0のみ

ID3D11Device::CreateUnorderedAccessView()で作成

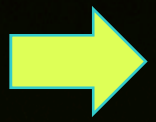
```
typedef struct D3D11_UNORDERED_ACCESS_VIEW_DESC {
    DXGI_FORMAT                 Format;
    D3D11_UAV_DIMENTIONS        ViewDimention;
    union {
        D3D11_BUFFER_UAV                Buffer;
        D3D11_TEX1D_UAV             Texture1D;
        D3D11_TEX1D_ARRAY_UAV       Texture1DArray;
        D3D11_TEX2D_UAV             Texture2D;
        D3D11_TEX2D_ARRAY_UAV       Texture2DArray;
        D3D11_TEX3D_UAV             Texture3D;
    };
} D3D11_UNORDERED_ACCESS_VIEW_DESC;
```

UAVは Pixel/Compute Shaderに Random Access可能なリソースを提供

機能詳細：New View/Buffer Type

UAVの導入にあわせて、Buffer/Texture CreationのBIND FLAGが拡張されます

```
typedef enum D3D11_BIND_FLAG
{
    D3D11_BIND_VERTEX_BUFFER        = 0x1L,
    D3D11_BIND_INDEX_BUFFER         = 0x2L,
    D3D11_BIND_CONSTANT_BUFFER      = 0x4L,
    D3D11_BIND_SHADER_RESOURCE      = 0x8L,
    D3D11_BIND_STREAM_OUTPUT        = 0x10L,
    D3D11_BIND_RENDER_TARGET        = 0x20L,
    D3D11_BIND_DEPTH_STENCIL        = 0x40L,
    D3D11_BIND_UNORDERED_ACCESS     = 0x80L,
} D3D11_BIND_FLAG;
```



D3D11_TEXTURE○○_DESC
D3D11_BUFFER_DESC

機能詳細：New View/Buffer Type

新たなBufferも追加されました

Raw Buffer

- 32bit aligned typeless array
-D3D11_RESOURCE_MISC_BUFFER_ALLOW_RAW_VIEWS
- ShaderResourceView or UnorderedResourceView

Structured Buffer

-Struct data access
- D3D11_RESOURCE_MISC_BUFFER_STRUCTURED
- ShaderResourceView or UnorderedResourceView

Append/Consume Buffer

-Bufferの終端をRead/WriteするためのBuffer
-Structured Bufferの特殊なもの
- D3D11_BUFFER_UAV_FLAG = D3D11_BUFFER_UAV_FLAG_APPEND
- UnorderedResourceViewのみ

機能詳細：New View/Buffer Type

Bufferに対してShaderでRead Writeが可能になりました

Buffer	RWBuffer
Texture1D	RWTexture1D
Texture1DArray	RWTexture1DArray
Texture2D	RWTexture2D
Texture2DArray	RWTexture2DArray
Texture3D	RWTexture3D
StructuredBuffer	RWStructuredBuffer
ByteAddressBuffer	RWByteAddressBuffer

operator[]を利用してアクセス

例:

```
RWBuffer                    rwb;

float4 PSMain() : SV_Target
{
    rwb[123] = 3;
    return  float4(rwb[1], 0, 0, 0);
}
```

Load()/Store()でアクセス
Atomic命令が使える

例:

```
RWByteAddressBuffer    rwbab;

float4  PSMain()  : SV_Target
{
    rwbab.Store(123, 3);
    return float4(rwbab.Load(1), 0, 0, 0);
}
```

Atomic操作
InterlockedAdd
InterlockedAnd/InterlockedOr/InterlockedXor
InterlockedCompareExchange
InterlockedCompareStore
InterlockedExchange
InterlockedMax/InterlockedMin

機能詳細：New View/Buffer Type

- Unordered Access な Read/Write Bufferを使うことで
  - Order Independent Transparency (with Atomic)
  - Data Logging
  - and so on…

などが考えられる

機能詳細：New Texture Compression

- FeatureLevel
- MultiThread
- Dynamic Shader Linkage
- Read Only Depth
- Conservative oDepth
- Pixel Shader Input Coverage
- New View/Buffer Type
- **New Texture Compression**

機能詳細：New Texture Compression

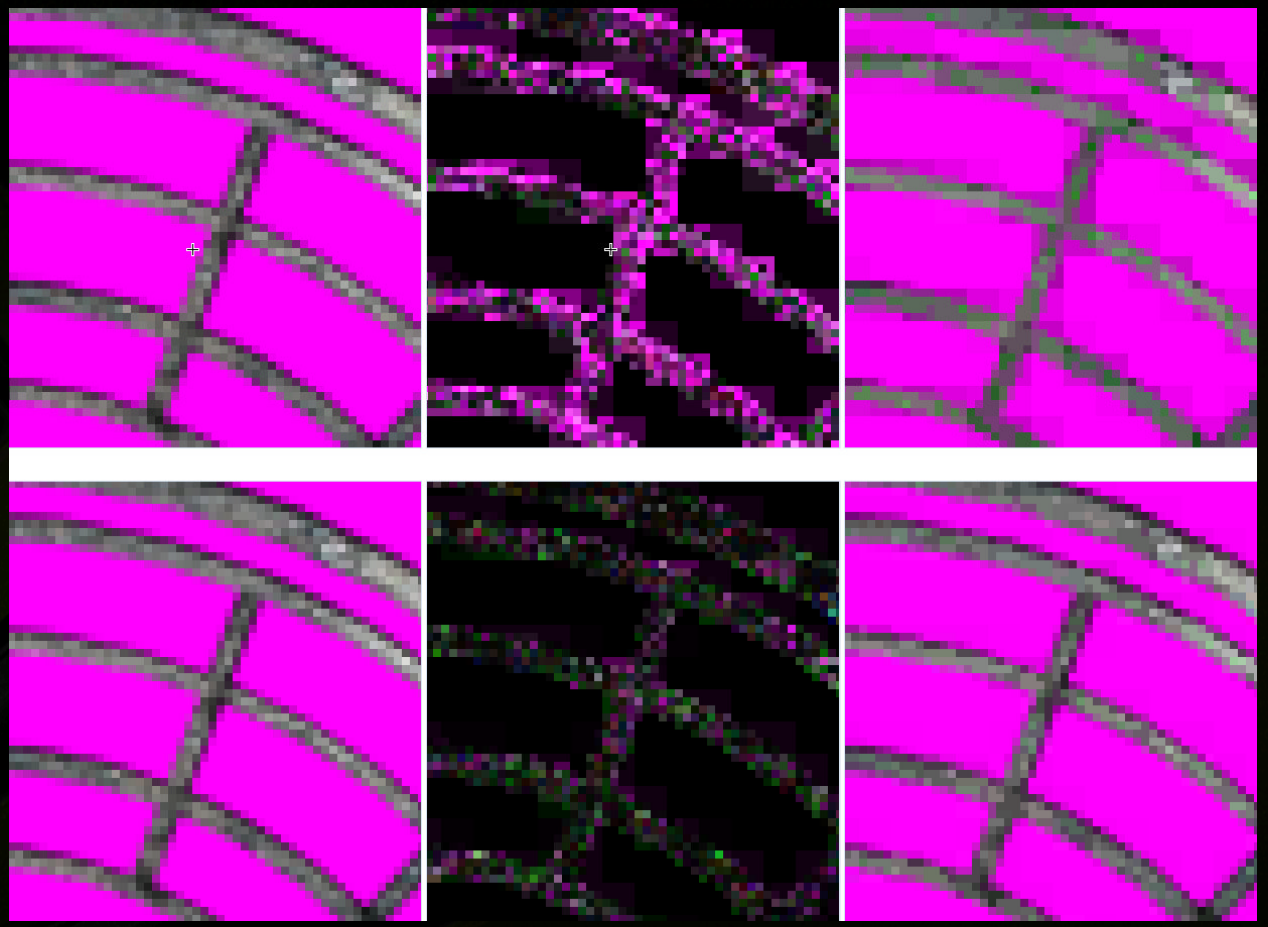
- 今までのD3DでのTextureの問題点
  1. Blockが非常に単純
     (Blockノイズが出るときがある)
  2. HDRが無い

機能詳細：New Texture Compression

- BC6 (BC6H)
  - High Dynamic Range
  - 6:1 compression
  - High quality
- BC7
  - LDR with Alpha
  - 3:1 compression(RGB) or 4:1 (RGBA)
  - High quality

Compression Blockも
Multiple blockを採用

機能詳細：New Texture Compression



NVISION08 “Introduction to the Direct3D 11 Pipeline”より

D3D11その他の機能

- Pixel Shader
  - Programmable Interpolation of Input
  - Centroid
- Texture
  - Improved Gather4
- Stream Out
  - Addressable Stream Out
  - 4 counts output
- SM5.0
  - Count bits, Find bits, Carry&Overflow, Bit reversal
  - Conditional Swap
- etc
  - Float point viewport
  - Per-resource mipmap clamp

他にもいろいろ！

Question?

Takayuki Kazama tkazama@nvidia.com

NVIDIA Corporation