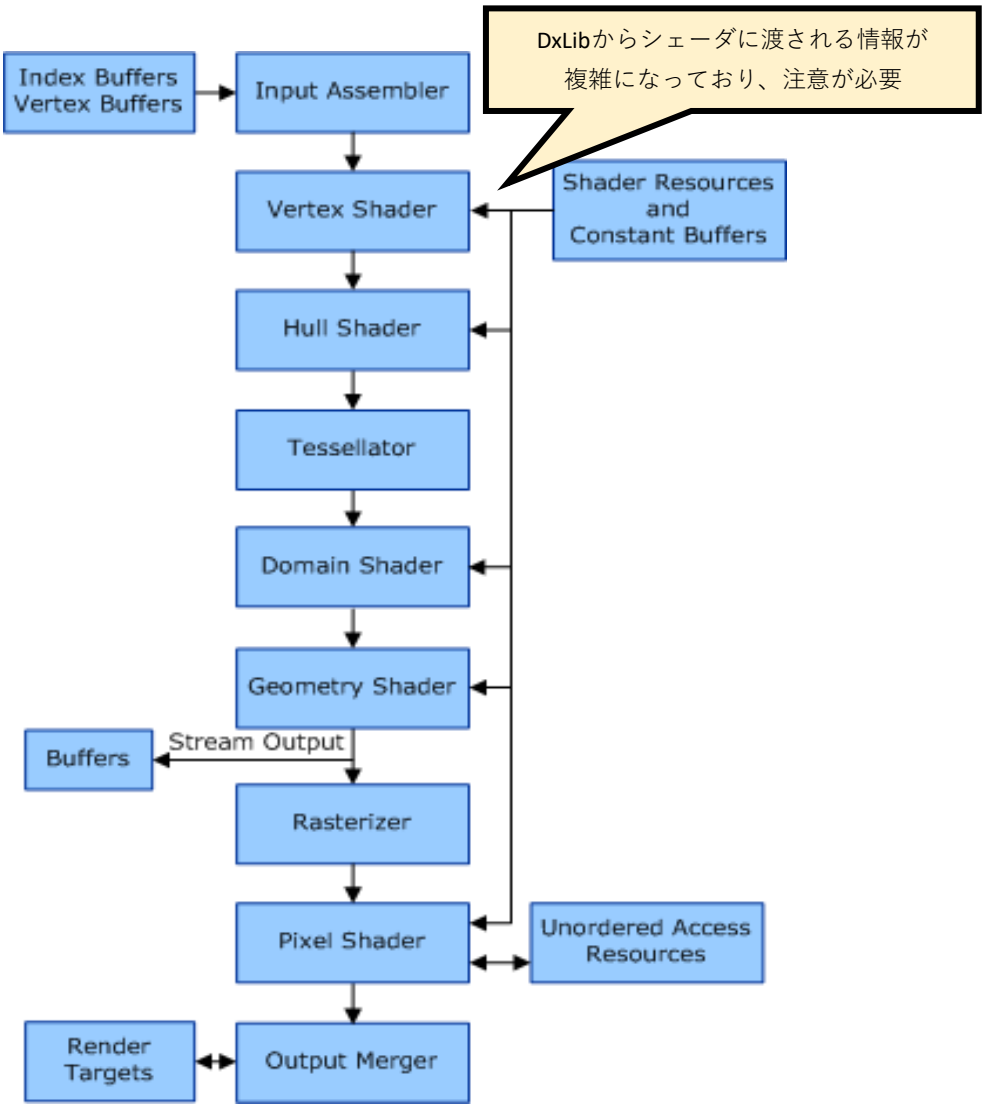
DxLibのシェーダ

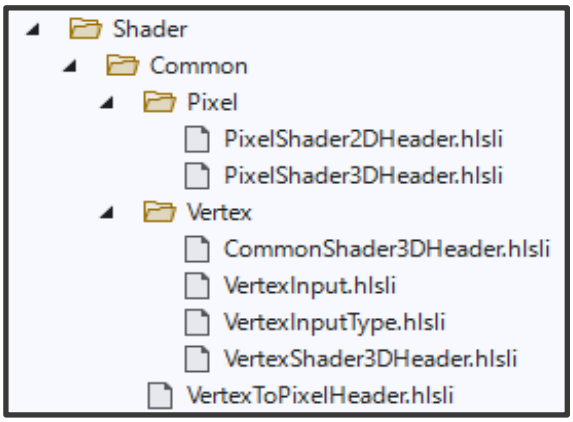
オリジナルシェーダを使用するにあたり、DxLibの仕様には注意が必要です。
グラフィックフローの最初に渡される情報は、DxLibの仕様で決められており、
DirextXのバージョンによって、大きく異なります。
ゲームアーキテクチャでは、DX9は古い、DX12は難し過ぎることから、
DirextX 11をDxLibで使っていきます。

公式サイトでも上手くまとめられている状態ではありませんので、
Teams開発情報サポートでも公開されている、
「３ＤゲームのTips
　03_DXライブラリで3Dシェーダーの導入が分からない人向けのサンプル」
参照しましょう。



ピクセルシェーダを使うための準備

配布プロジェクトの下記場所のファイルをHLSLから include することで
ある程度は複雑さが解消されます。



今回、ポストエフェクトで使用していく関数、

DrawPolygonIndexed2DToShader
(ポリゴンの頂点情報を元にオリジナルピクセルシェーダを２Ｄ的に使用)

こちらは、HLSLの記述の最初で下記ファイルを include しておくと良いです。

#include "../Common/Pixel/PixelShader2DHeader.hlsli"

[内容]

```
struct PS_INPUT
{
    float4 svPos    : SV_POSITION;
    float4 diffuse  : COLOR0;           //拡散反射の色
    float2 uv       : TEXCOORD0;        //UV値
    float2 suv      : TEXCOORD1;        //サブテクスチャのUV値
};

Texture2D tex : register(t0);                   //テクスチャ
SamplerState texSampler : register(s0);         //サンプラー
```

どんな頂点シェーダが使用されているか不明なため、
ピクセルシェーダに、何の情報がどの順番で渡されるか、わかりにくい。
概ね、上記の情報、セマンティクスで事足りる。

これを include することで、
ピクセルシェーダ内の main 関数の引数で渡されている情報(PS_INPUT)が
明確になり、ピクセルシェーダを複数種類作る際にも共通化でき、
個別に定義する必要がなくなる。

```
float4 main(PS_INPUT PSInput) : SV_TARGET
{
    ・・・
```

定数バッファの使用について

DxLib側でも定数バッファを使用しています。

DxLibと、オリジナルシェーダで使用する私たちの定数バッファの
使用番号(スロット)が混在しないように、

ピクセスシェーダは、『 4番目 』から使用するように。

```
// 定数バッファ：スロット4番目(b4と書く)
cbuffer cbParam : register(b4)
{
    float g_none;
}
```

頂点シェーダは、『 7番目 』から使用するように。

```
// 定数バッファ：スロット7番目
cbuffer cbParam : register(b7)
{
    float g_time;
    float2 g_uv_scale;
    float dummy;
}
```