



DirectX 11 소개

지오지엔엔터테인먼트
조진현
<http://vsts2010.net>

아무도 예상하지 못했습니다!!!

멀티 콘텐츠 시대



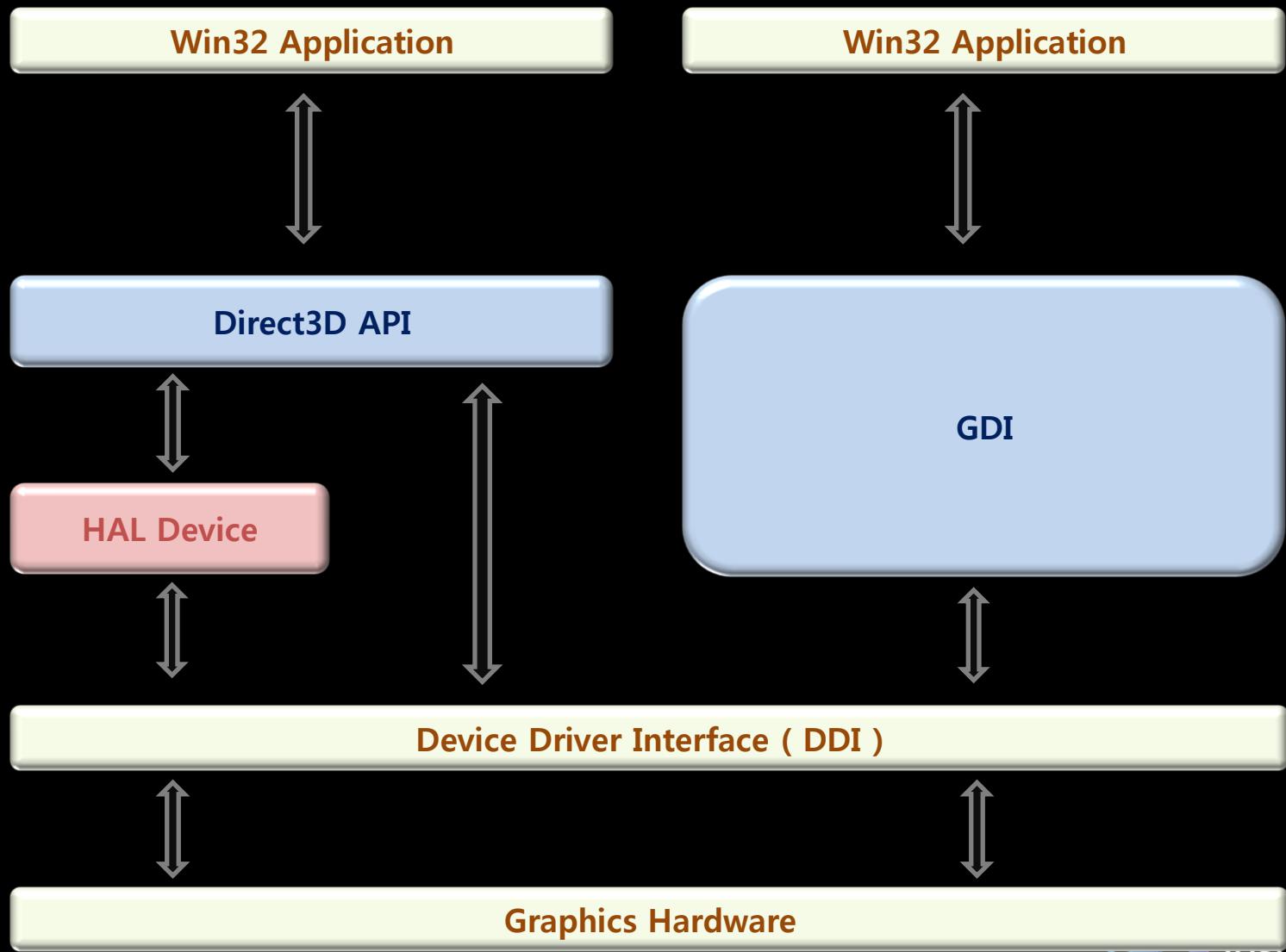
DirectX 도 자유로울 수 없다!!!

~ DX9

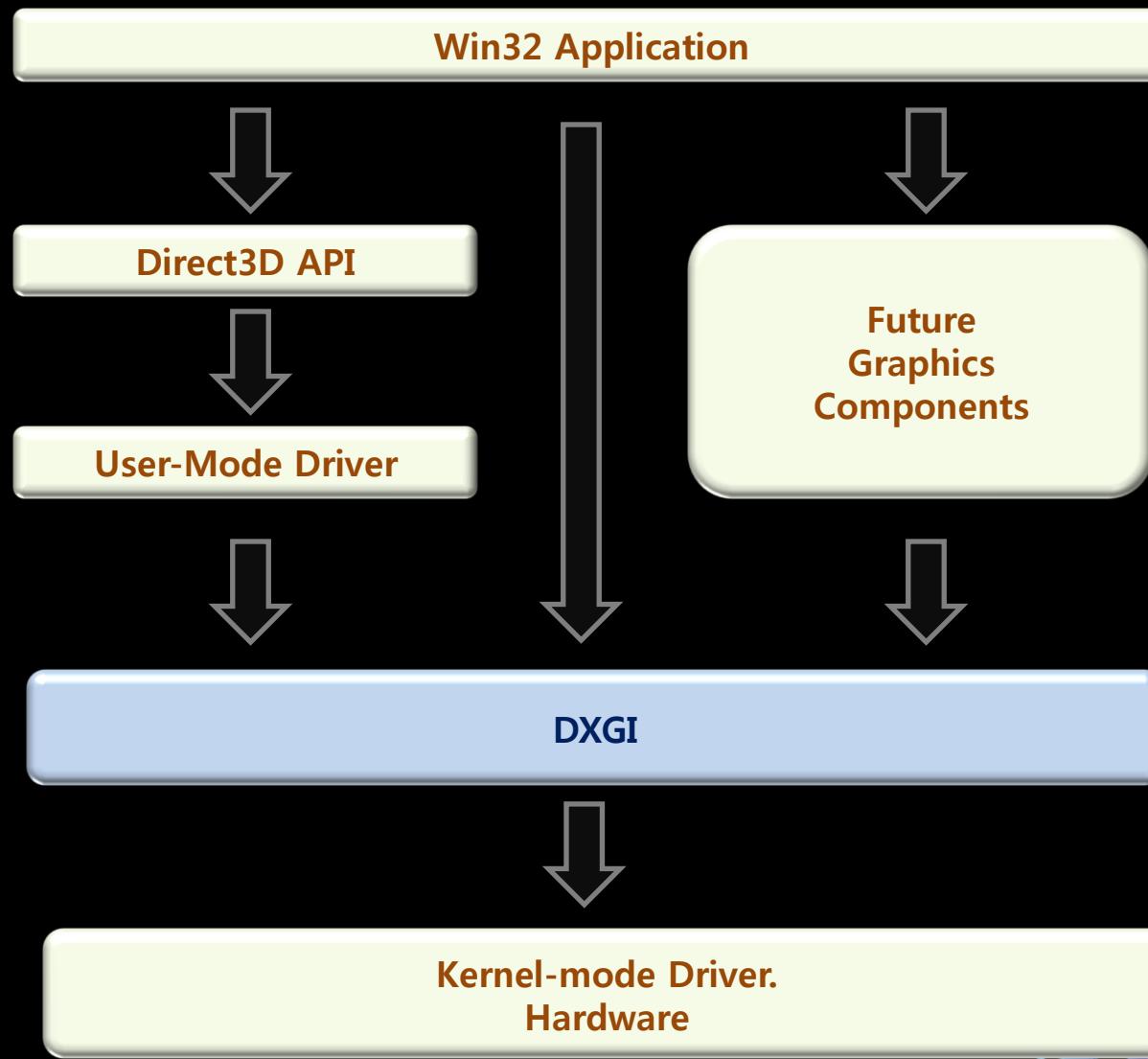
과거 세대

DX10 ~

현 세대



더 이상 게임만을 위한 DirectX 가 아닙니다.



이런 멋진 개념으로 등장!!



DirectX 10 의 성공적인 데뷔?



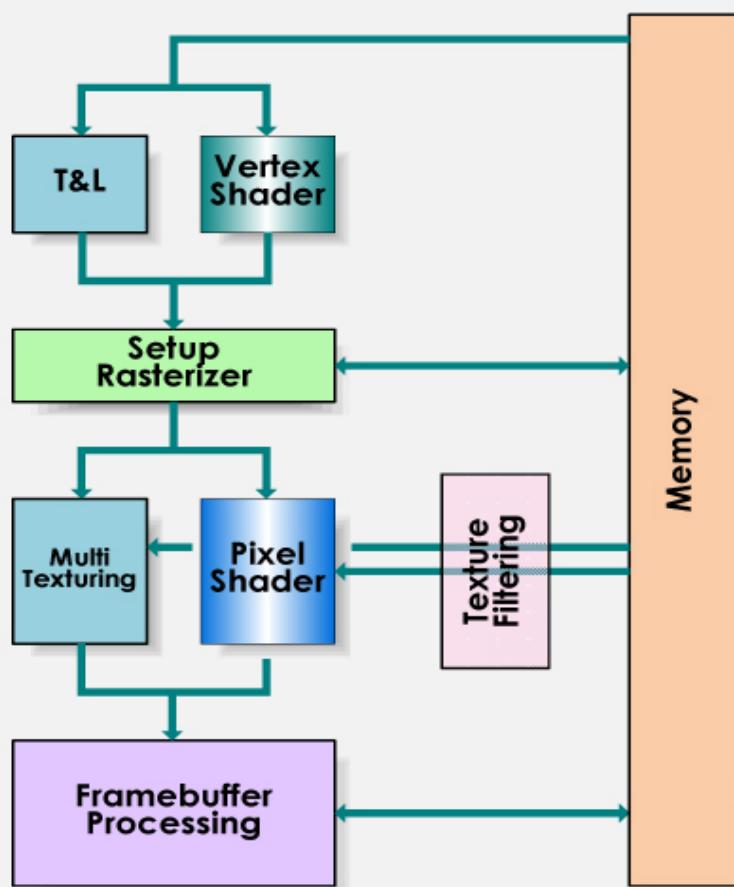


그래서 ?
어쩌라고?

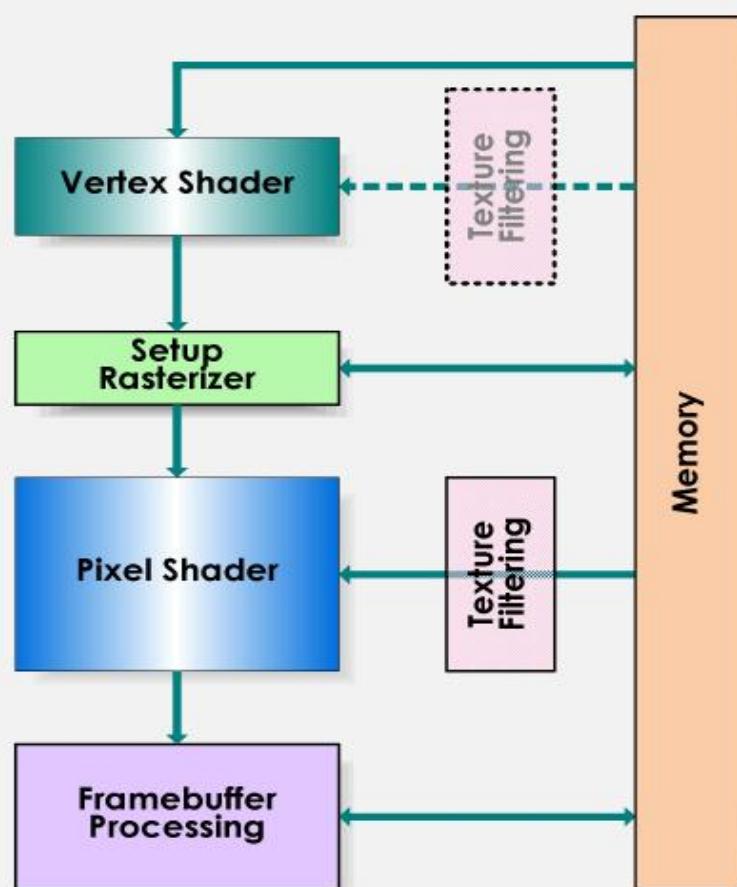
무관심!!!

파이프라인의 변화

Direct3D8 Logical Pipeline



Direct3D9 Logical Pipeline

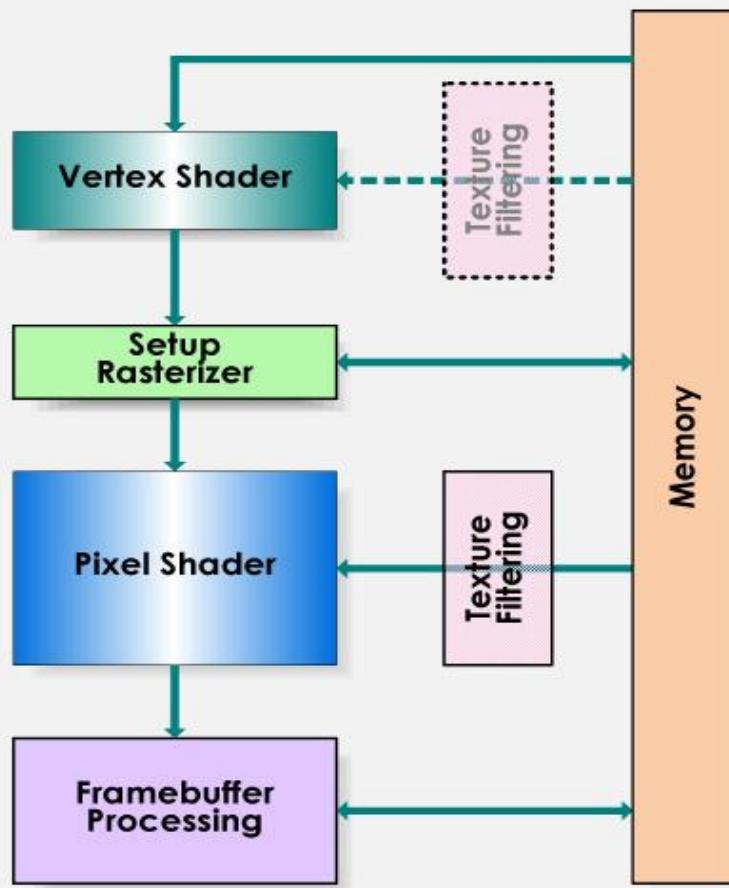


착각?

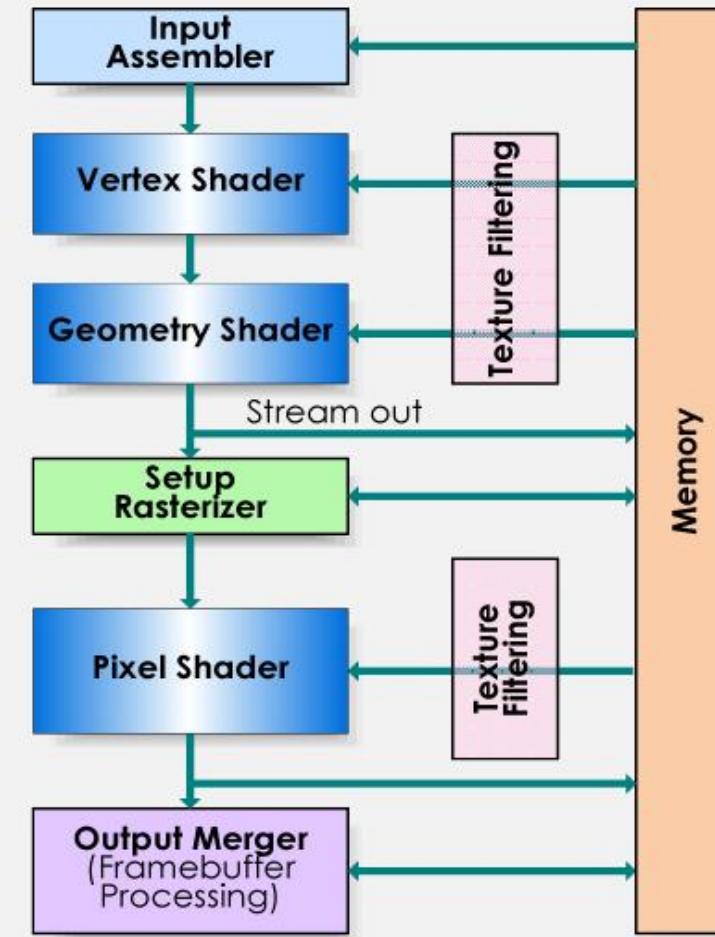
Version up == 단순 포팅 작업



Direct3D9 Logical Pipeline



Direct3D10 Logical Pipeline



DIRECTX11

DirectX 11 소개

Tessellation



텍스쳐의 힘?



Texturing 의 한계



(a) An untessellated character



(b) Character rendered with GPU tessellation

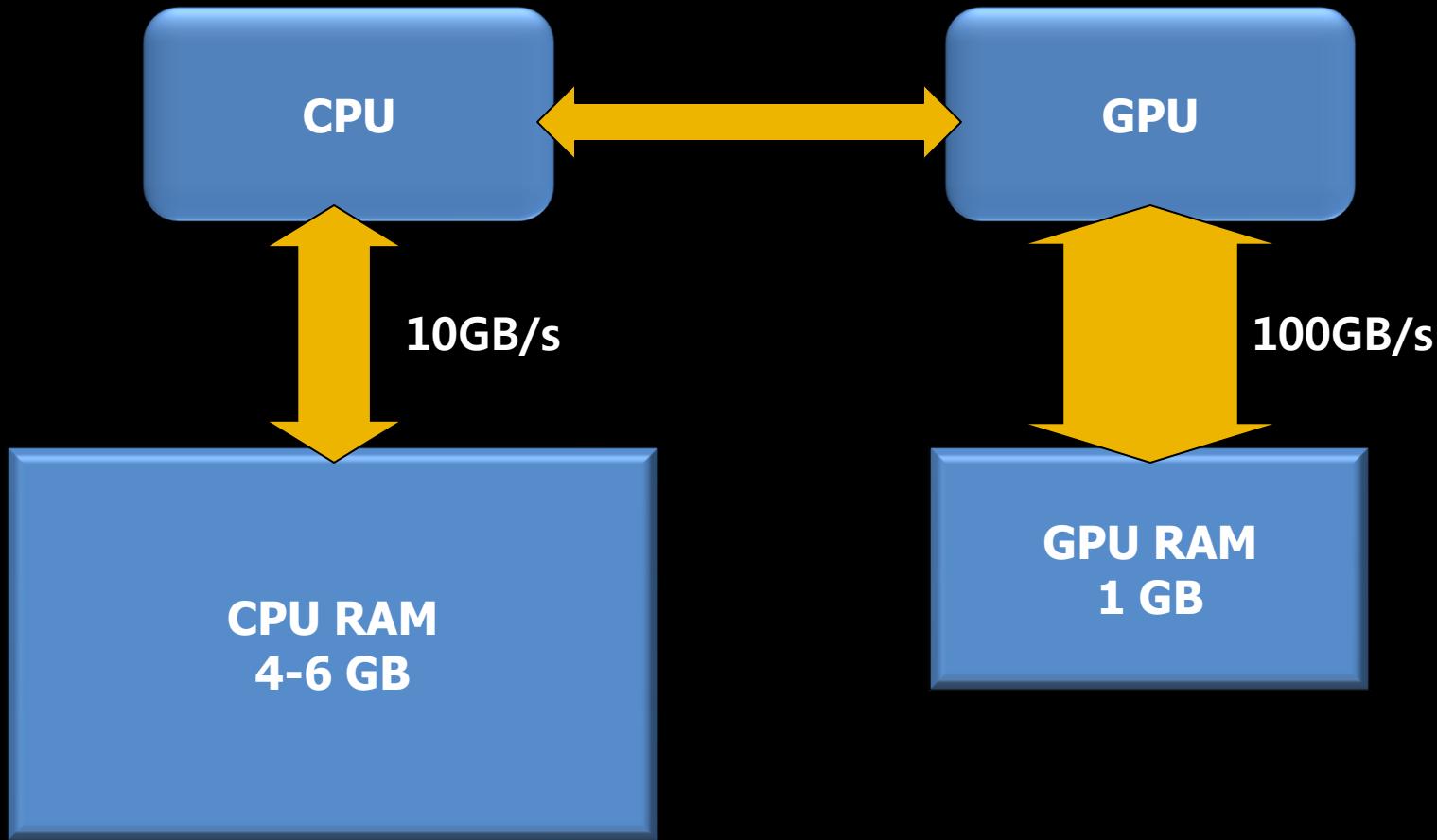


(c) An untessellated character close-up



(d) Tessellated character close-up

왜 게임에서는 폴리곤 갯수를
증가시키기 어려운가요?

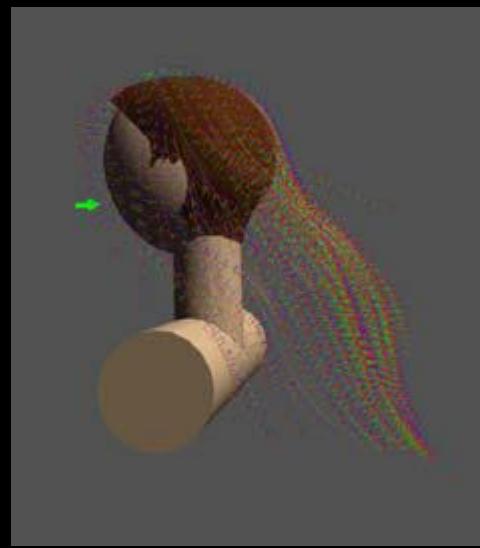
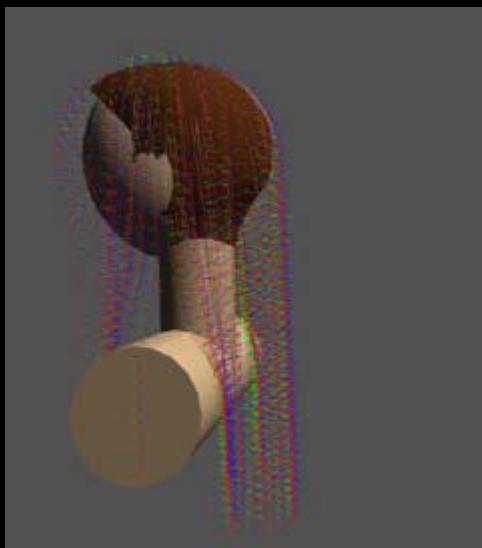
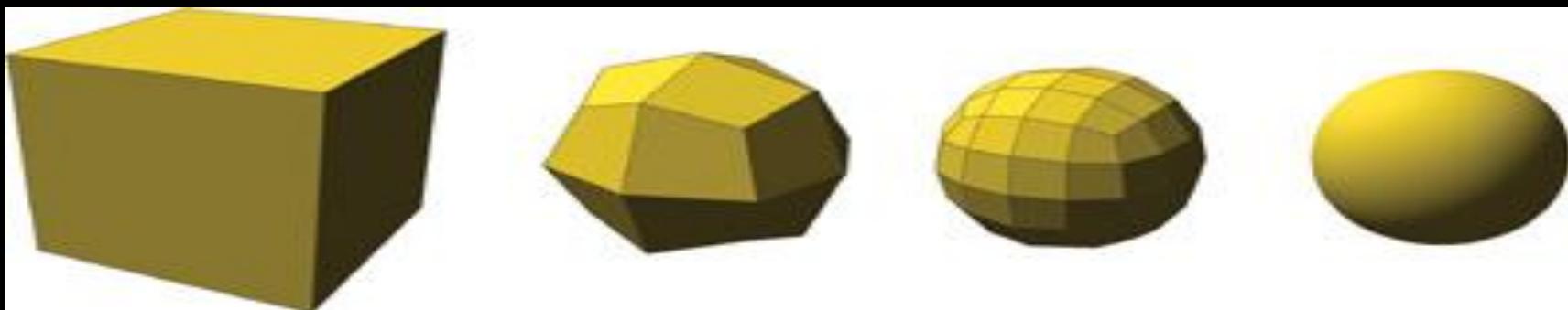


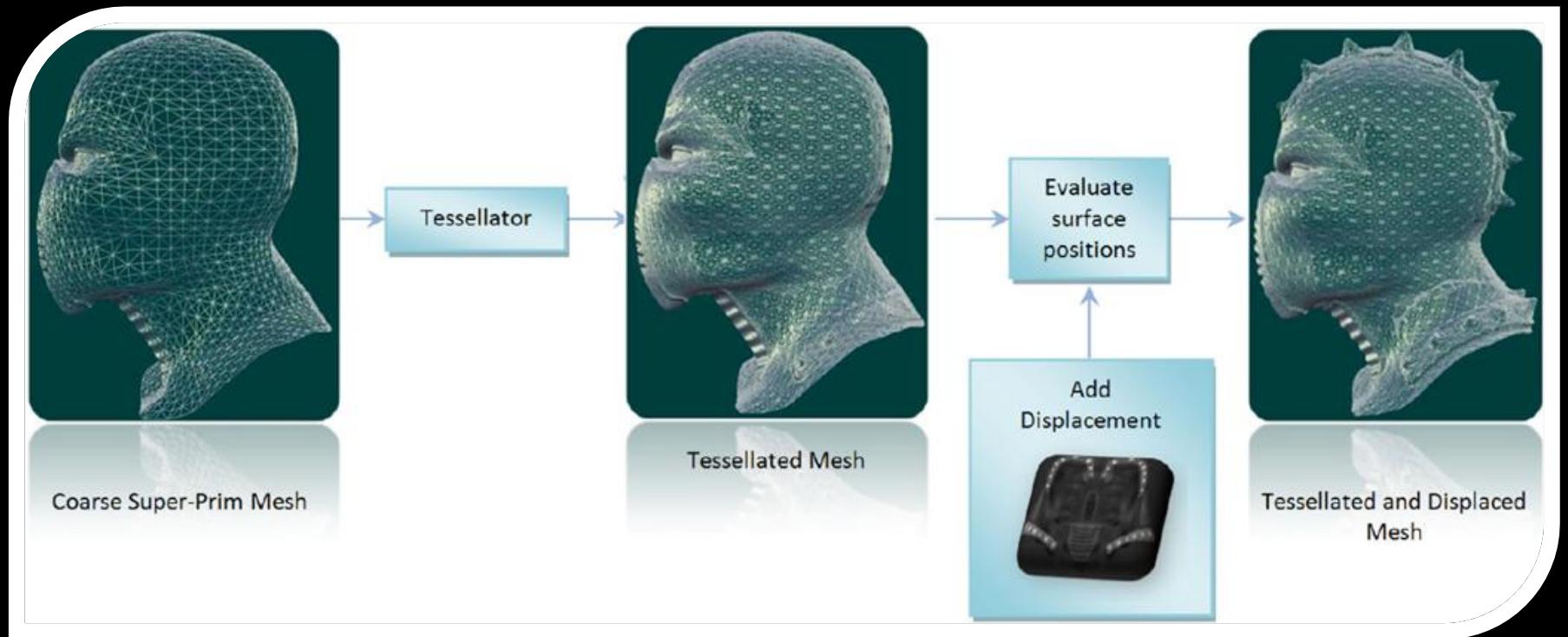
Tessellation 의 이점은 ?

GPU 의 프로세싱 능력 활용!

메모리 절약!

대역폭 감소!



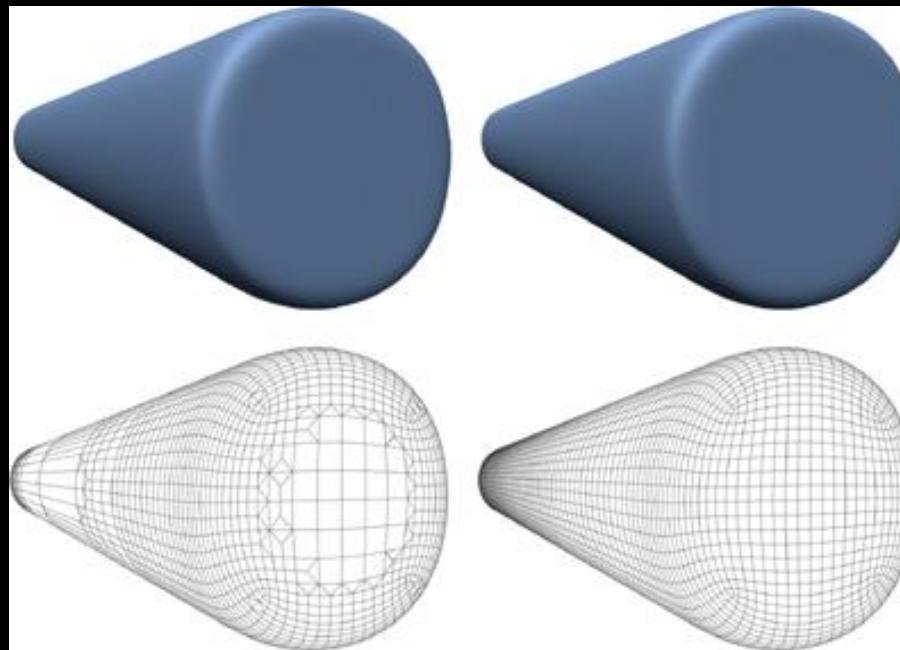


Tessellation 을 위해 필요한 정보

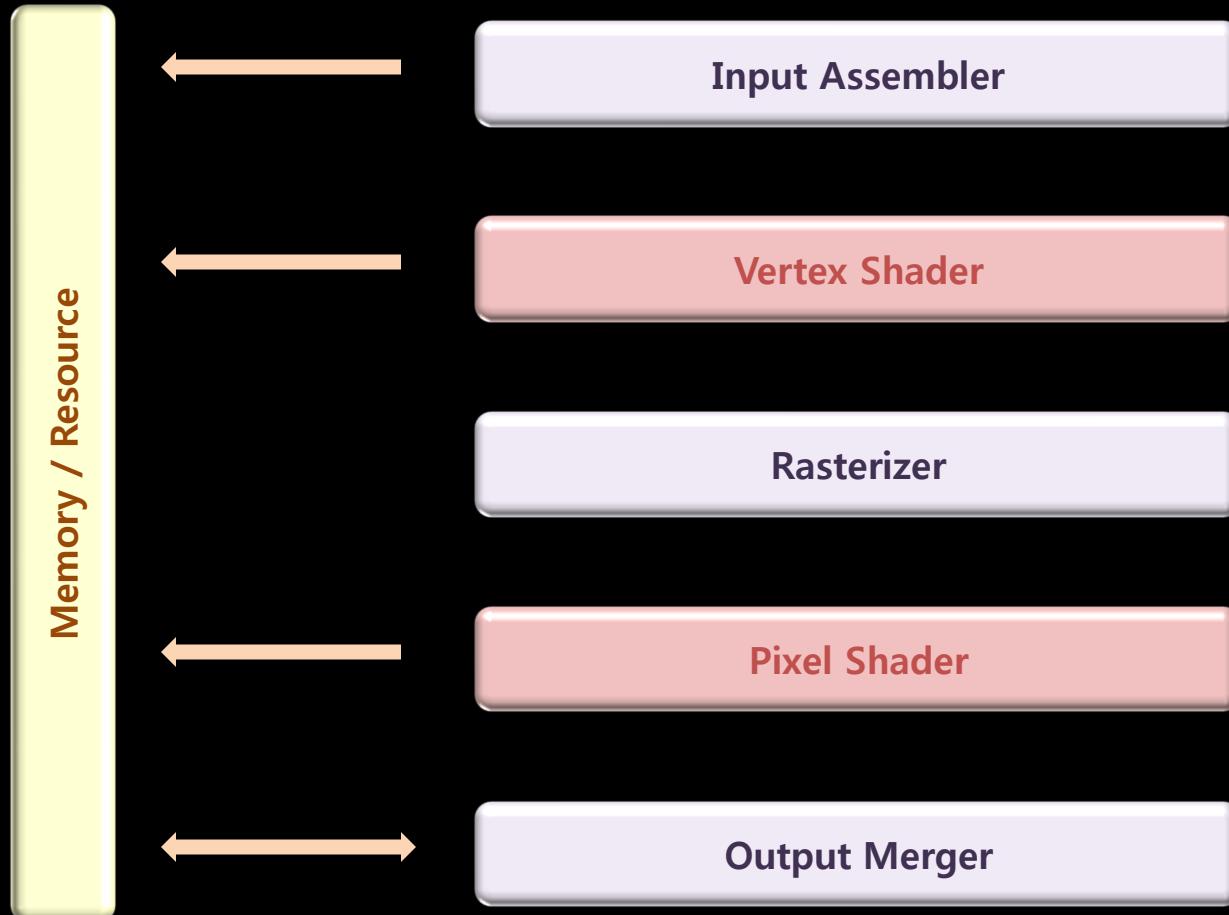
Control Points

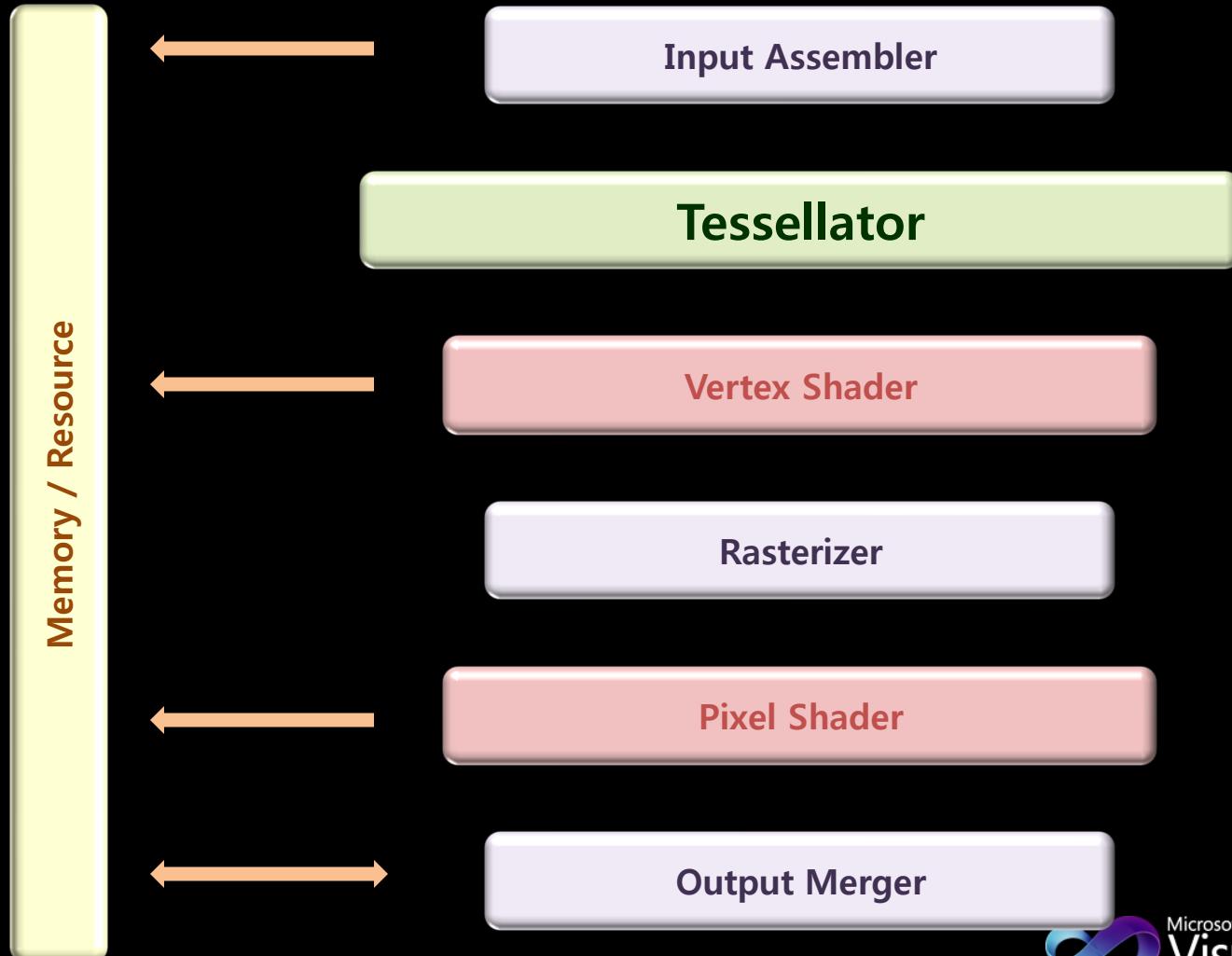
Tessellation Factors

DirectX9 에서의 Tessellation.

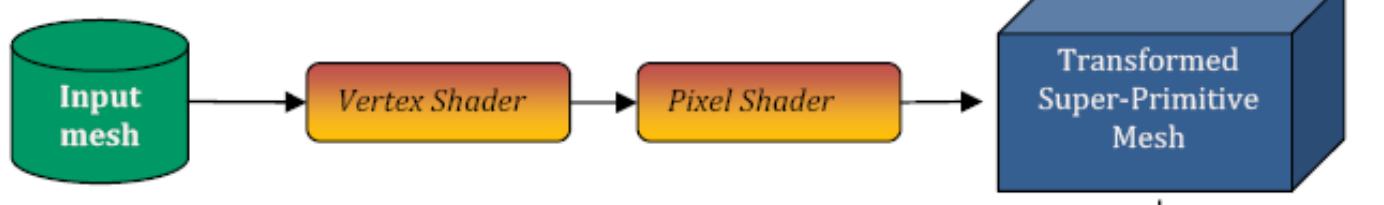


Adaptive Tessellation.

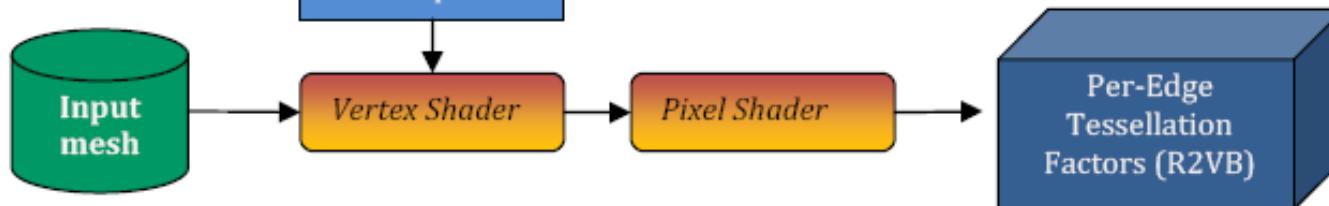




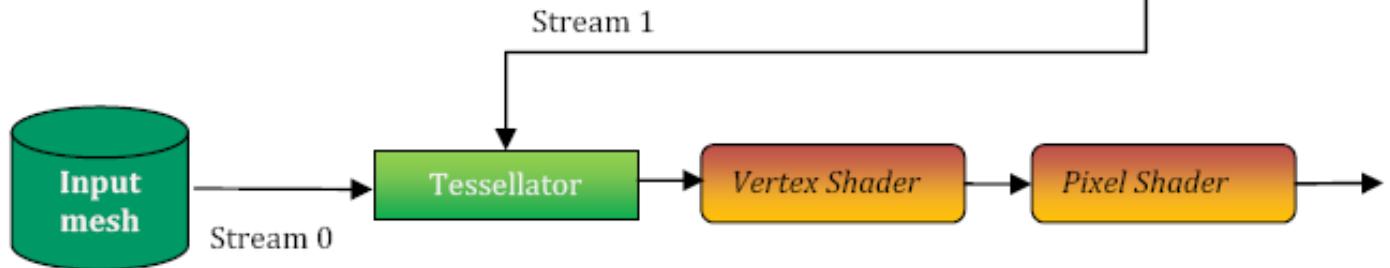
Pass 1:
Animate and transform input mesh



Pass 2:
Compute per-edge tessellation factors



Pass 3:
Render tessellated mesh



1 Pass

```

struct VsTfmInput
{
    float4 vPosOS : POSITION0;
    float2 vTexCoord: TEXCOORD0;
    float3 vNormal : NORMAL0;
};

struct VsTfmOutput
{
    float4 vPos      : POSITION;
    float fVertexID : TEXCOORD0;
    float3 vPosCS   : TEXCOORD1;
};

//-----

VsTfmOutput VSRenderTransformed( VsTfmInput i )
{
    VsTfmOutput o;
    int nVertexID    = floor( i.vPosOS.w );
    int nTextureWidth = g_vTformVertsMapSize.x;

    // Compute row and column of the position in 2D texture
    float2 vPos    = float2( nVertexID % nTextureWidth, nVertexID / nTextureWidth );
    vPos /= g_vTformVertsMapSize.xy;
    vPos.y = 1.0 - vPos.y;
    vPos    = vPos * 2 - 1.0; // Move to [-1; 1] range

    o.vPos = float4( vPos.xy, 0, 1 );

    // Propagate the vertex ID to the pixel shader:
    o.fVertexID = i.vPosOS.w;

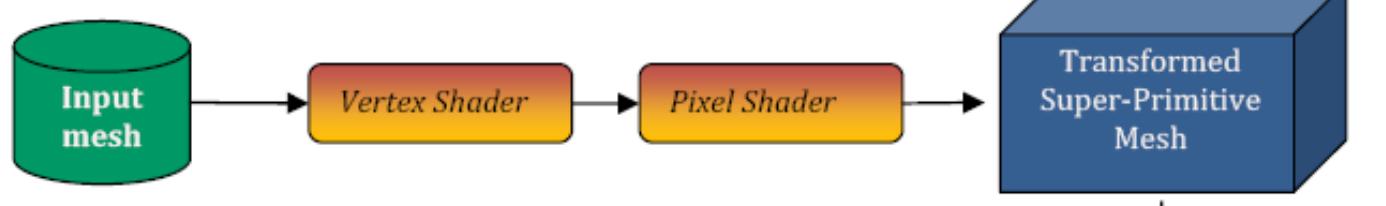
    // Transform vertex position to screen-space
    float4 vPositionCS = mul( float4( i.vPosOS.xyz, 1 ), g_mWorldView );
    vPositionCS /= vPositionCS.w;

    o.vPosCS = vPositionCS.xyz;

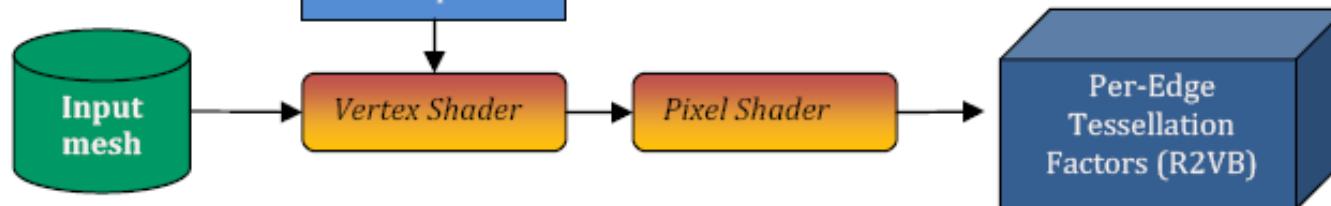
    return o;
} // End of VsTfmOutput VSRenderTransformed(..)

```

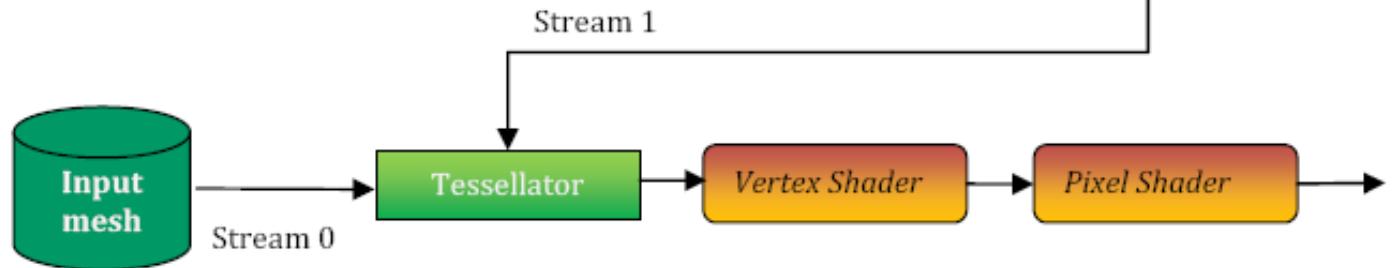
Pass 1:
Animate and transform input mesh



Pass 2:
Compute per-edge tessellation factors



Pass 3:
Render tessellated mesh



2 Pass

```

struct VsTFInput
{
    float4 vPositionOS : POSITION0; // (xyz) - camera-space position, w - vertex ID
};

struct VsTFOOutput
{
    float4 vPositionCS : POSITION;
    float fTessFactor : TEXCOORD0;
};

//-----

VsTFOOutput VSCalculateTFs( VsTFInput i )
{
    VsTFOOutput o;

    // Current vertex ID:
    int nCurrentVertID = (int)( i.vPositionOS.w );

    // Determine the ID of the edge neighbor's vertex (remember that this is done with non-indexed primitives, so
    // basically if the current vertex is v0 ,then we have edges: v0->v1, v1->v2, v2->v0

    // Note: currently (Jan 08) MS HLSL compiler has a bug in MOD ('%') computation.
    // Had to implement MOD myself because MS HLSL compiler didn't do it right - apparently using nCurrentVertID % 3
    // produced over 17 instructions and _wrong_ results. So did it myself [NT]. MS was notified
    int nCurrentVertEdgeID = nCurrentVertID - 3 * floor( (float)nCurrentVertID / (float) 3);

    int nEdgeVert0ID = nCurrentVertID; // this works if current vertex is v0 or v1
    int nEdgeVert1ID = nCurrentVertID + 1;

    if ( nCurrentVertEdgeID == 0 )
    {
        nEdgeVert0ID = nCurrentVertID + 1;
    }
    else if ( nCurrentVertEdgeID == 1 )
    {
        nEdgeVert0ID = nCurrentVertID + 1;
    }
    else if ( nCurrentVertEdgeID == 2 ) // In case of v2 we need to wrap around to v0
    {
        nEdgeVert0ID = nCurrentVertID - 2;
    }
}

```

```

// Compute the fetch coordinates to fetch transformed positions for these two vertices to compute their edge statistics:

int    nTextureWidth      = g_vTformVertsMapSize.x;

// Vertex0: nCurrentVertID, compute row and column of the position in 2D texture:
float2 vVert0Coords   = float2( nCurrentVertID % nTextureWidth, nCurrentVertID / nTextureWidth );
vVert0Coords /= g_vTformVertsMapSize.xy;

// Vertex1: nEdgeVert0ID, compute row and column of the position in 2D texture:
float2 vVert1Coords   = float2( nEdgeVert0ID % nTextureWidth, nEdgeVert0ID / nTextureWidth );
vVert1Coords /= g_vTformVertsMapSize.xy;

// Fetch transformed positions for these IDs:
float4 vVert0Pos = tex2Dlod( sTformVerts, float4( vVert0Coords, 0, 0 ) );
float4 vVert1Pos = tex2Dlod( sTformVerts, float4( vVert1Coords, 0, 0 ) );

// Swap vertices to make sure that we have the same edge direction regardless of their triangle order (based on vertex ID):
if ( vVert0Pos.w > vVert1Pos.w )
{
    float4 vTmpVert = vVert0Pos;
    vVert0Pos = vVert1Pos;
    vVert1Pos = vTmpVert;
}

// Use the distance from the camera to determine the tessellation factor:
float fEdgeDepth = 0.5 * ( ( vVert1Pos.z ) + ( vVert0Pos.z ) );

float fTessFactor = clamp( fEdgeDepth / g_fMaxCameraDistance, 0, 1 ); // map to 0-1 range
fTessFactor = (1 - fTessFactor) * 15;                                // map to 0-15 range and invert it (higher to lower)

const float fMinTessFactor = 1.0;
const float fMaxTessFactor = 14.99999;

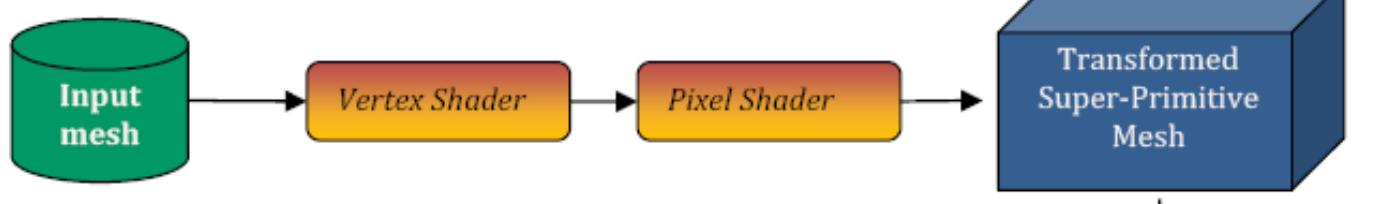
o.fTessFactor = clamp( fTessFactor, fMinTessFactor, fMaxTessFactor ); // clamp to the correct range

// Compute output position for rendering into a tessellation factors texture
int nVertexID = floor( i.vPositionOS.w );

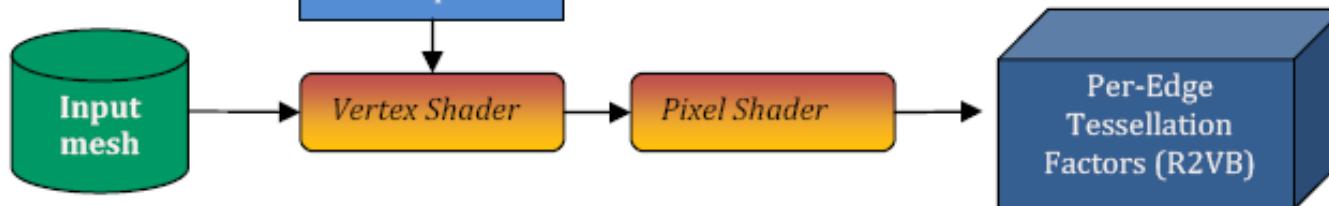
// Compute row and column of the position in 2D texture
float2 vOutputPos   = float2( nVertexID % nTextureWidth, nVertexID / nTextureWidth );
vOutputPos /= g_vTformVertsMapSize.xy;
vOutputPos.y = 1.0 - vOutputPos.y;
vOutputPos   = vOutputPos * 2 - 1.0; // Move to [-1; 1] range

```

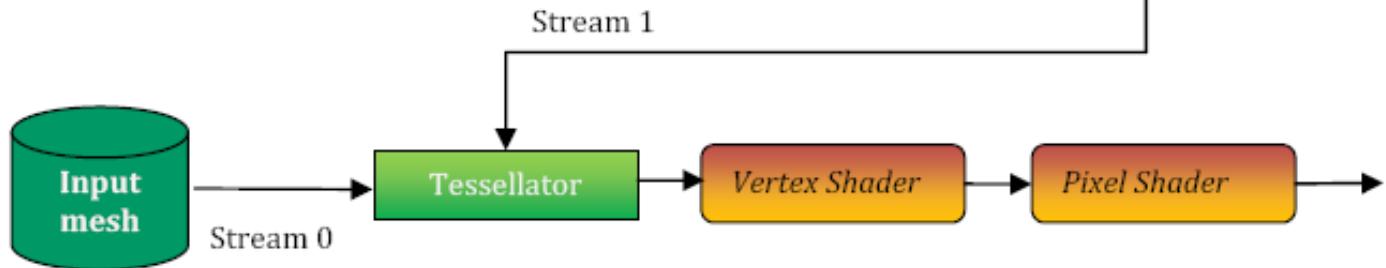
Pass 1:
Animate and transform input mesh



Pass 2:
Compute per-edge tessellation factors



Pass 3:
Render tessellated mesh



3 Pass

```
struct VsInputTessellated
{
    // Barycentric weights for tessellated vertex from the super-primitive (low-resolution input) mesh
    // I.e. each tessellated vertex quantities are computed via barycentric interpolation from the
    // three super-primitive input vertices
    float3 vBarycentric: BLENDWEIGHT0;

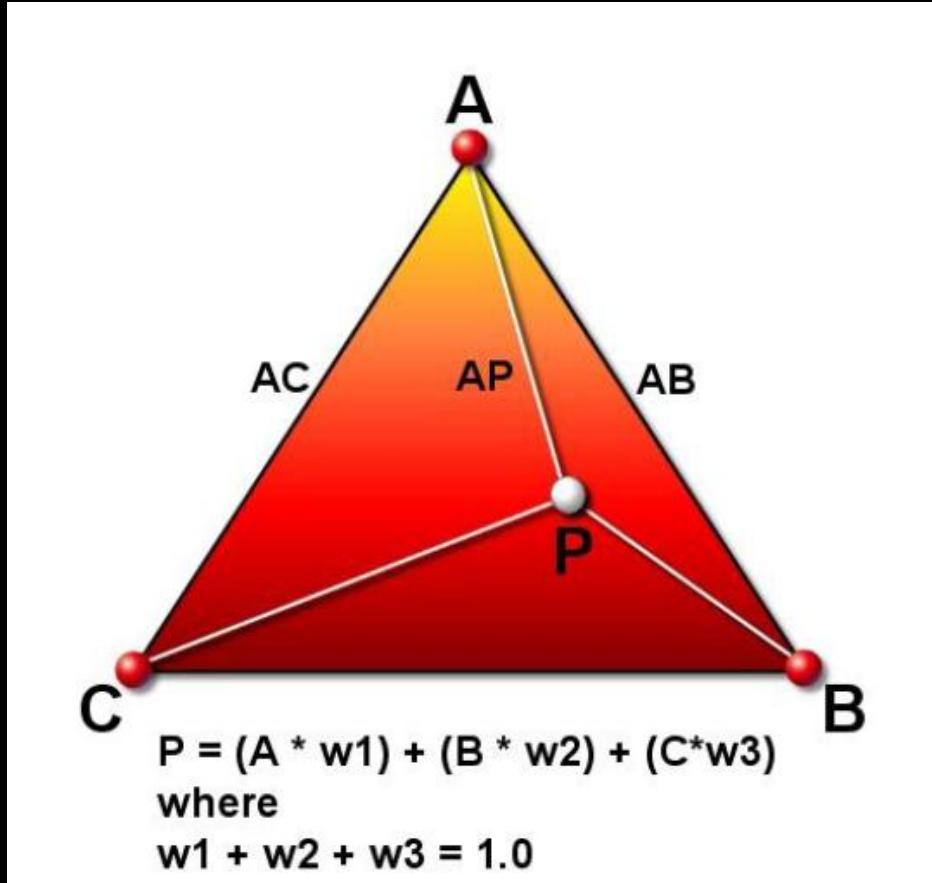
    // Data from superprim vertex 0:
    float4 vPositionVert0 : POSITION0;
    float2 vTexCoordVert0 : TEXCOORD0;
    float3 vNormalVert0 : NORMAL0;

    // Data from superprim vertex 1:
    float4 vPositionVert1 : POSITION4;
    float2 vTexCoordVert1 : TEXCOORD4;
    float3 vNormalVert1 : NORMAL4;

    // Data from superprim vertex 2:
    float4 vPositionVert2 : POSITION8;
    float2 vTexCoordVert2 : TEXCOORD8;
    float3 vNormalVert2 : NORMAL8;
};

struct VsOutputTessellated
{
    float4 vPosCS : POSITION;
    float2 vTexCoord : TEXCOORD0;
    float3 vNormalWS : TEXCOORD1;
    float3 vPositionWS : TEXCOORD2;
};
```

BaryCentric Coordinate



C weight = Area of APB /Area of ABC

```

//-----
// Render tessellated mesh without displacement
//-----
VsOutputTessellated VSRenderTessellated( VsInputTessellated i )
{
    VsOutputTessellated o;

    // Compute new position based on the barycentric coordinates:
    float3 vPosTessOS = i.vPositionVert0.xyz * i.vBarycentric.x + i.vPositionVert1.xyz * i.vBarycentric.y +
        i.vPositionVert2.xyz * i.vBarycentric.z;

    // Output world-space position:
    o.vPositionWS = vPosTessOS;

    // Compute new tangent space basis vectors for the tessellated vertex:
    o.vNormalWS = i.vNormalVert0.xyz * i.vBarycentric.x + i.vNormalVert1.xyz * i.vBarycentric.y +
        i.vNormalVert2.xyz * i.vBarycentric.z;

    // Compute new texture coordinates based on the barycentric coordinates:
    o.vTexCoord = i.vTexCoordVert0.xy * i.vBarycentric.x + i.vTexCoordVert1.xy * i.vBarycentric.y +
        i.vTexCoordVert2.xy * i.vBarycentric.z;

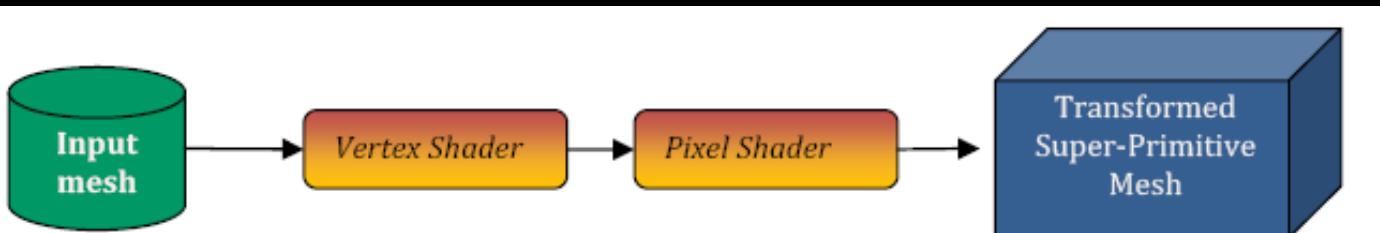
    // The model we exported is from Maya, so the texture space needs to be adjusted accordingly:
    o.vTexCoord = float2( o.vTexCoord.x, 1 - o.vTexCoord.y );

    // Transform position to screen-space:
    o.vPosCS = mul( float4( vPosTessOS, 1.0 ), g_mWorldViewProjection );

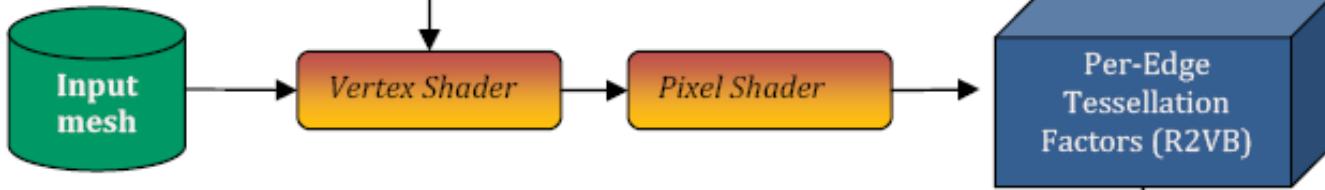
    return o;
} // End of VsOutputTessellated VSRenderTessellated(..)

```

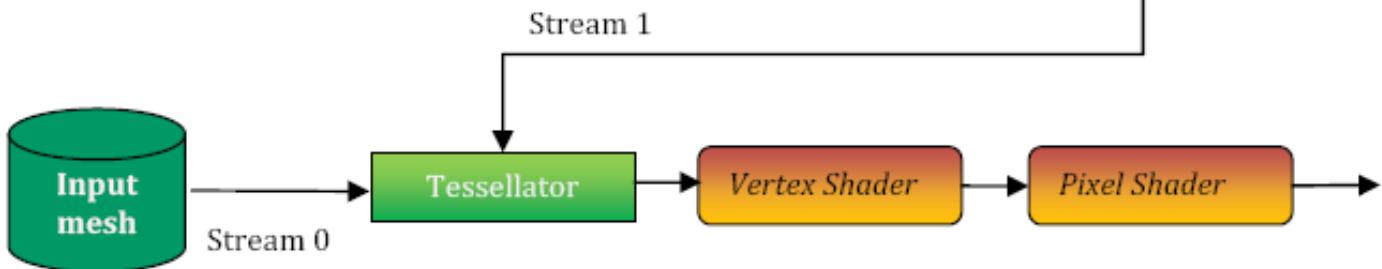
Pass 1:
Animate and transform input mesh

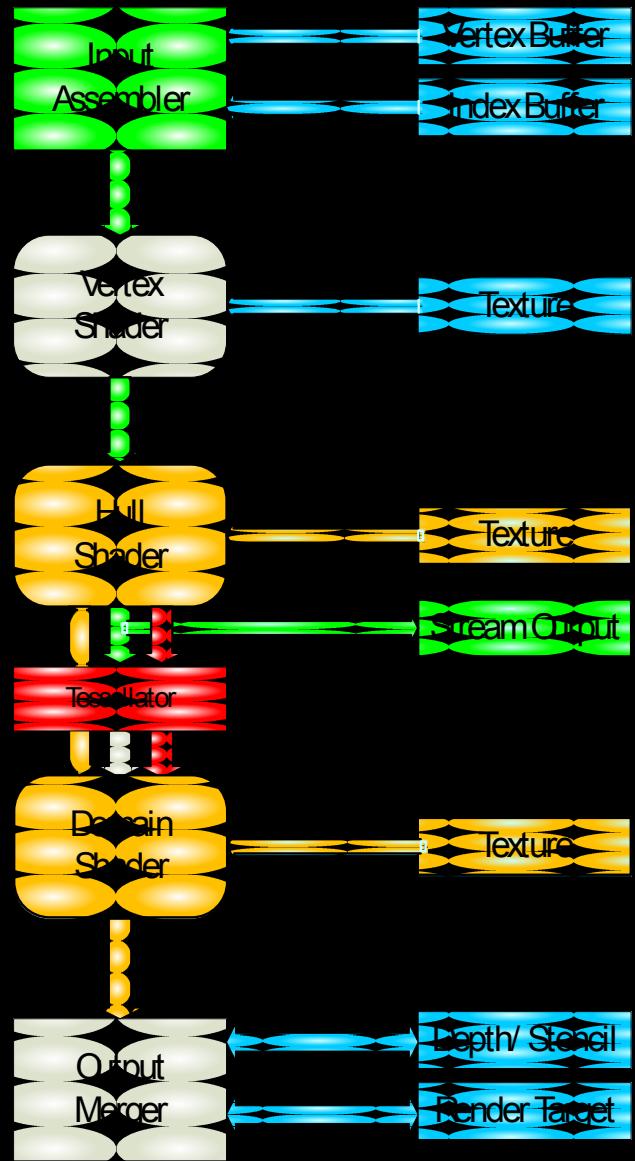


Pass 2:
Compute per-edge tessellation factors



Pass 3:
Render tessellated mesh





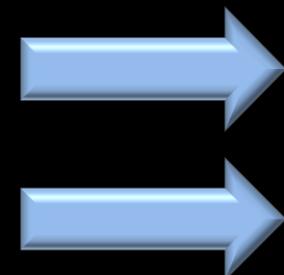
VertexShader



**patch control points
from the VS.**



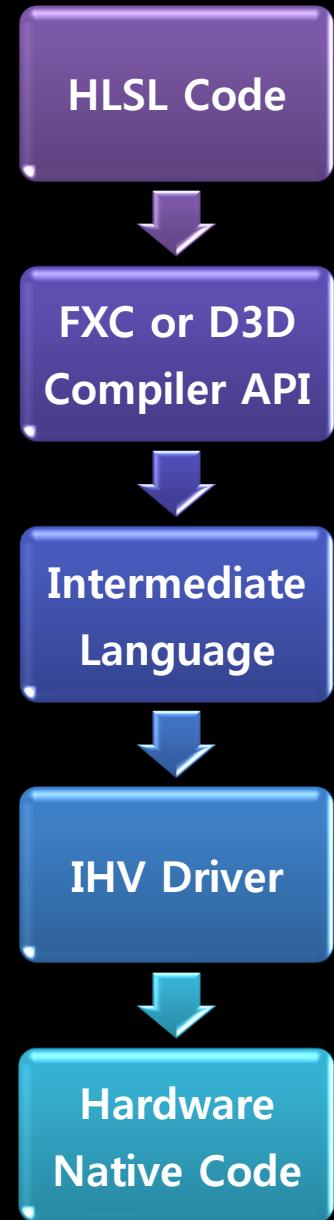
Control Points



**Patch Constant Data
(Tessellation Factors 포함)**

Compilation Steps

- Compiler (fxc or library) generates target-specific instructions (IL) from shader
- Different instruction sets for different generations of hardware
- Shader IL is highly optimized



Tessellator

Hull Shader
Output Control Points



Tessellator Stage
Output Texture Coordinates



Vertex Position



DirectX11 에서의 Tessellation

1 PASS

메모리 및 대역폭 절약

효율성 극대화 (ex>병렬 처리)

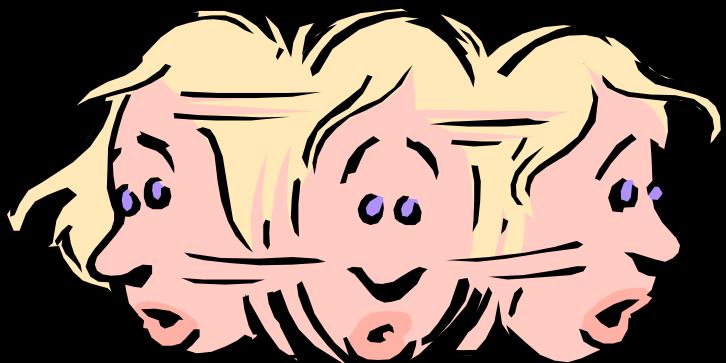
DirectX 11 소개

Tessellation

Compute Shader



GPGPU?
DirectCompute?
ComputeShader?

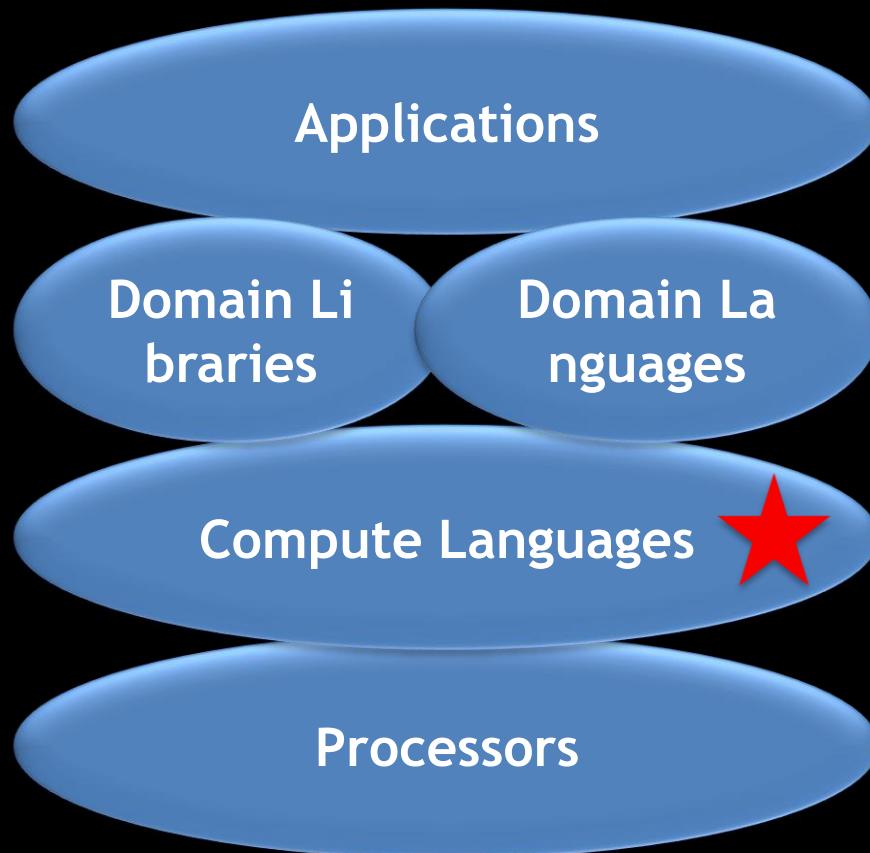


GPGPU

**General-Purpose computing
on Graphics Processing Unit**

DirectCompute

GPGPU 를 위한 MS의 API



Media playback or processing, media UI, recognition, etc. Technical

Accelerator, Brook+, Rapidmind, Ct
MKL, ACML, cuFFT, D3DX, etc.

DirectCompute, CUDA, CAL, OpenCL, L
RB Native, etc.

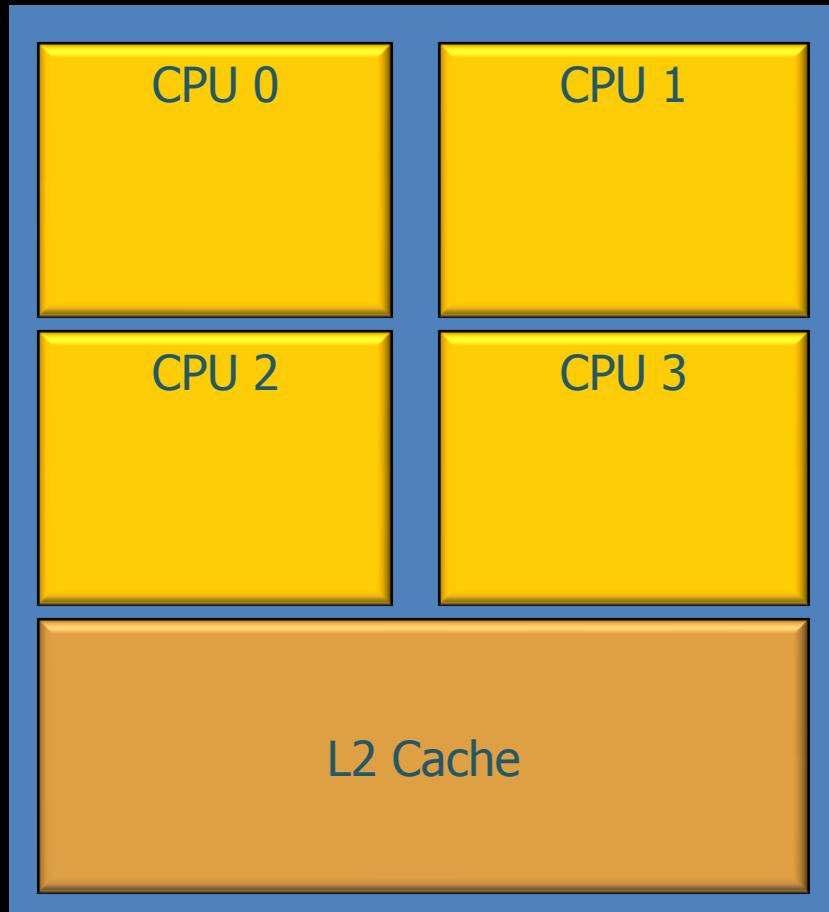
CPU, GPU, Larrabee
nVidia, Intel, AMD, S3, etc.

ComputeShader

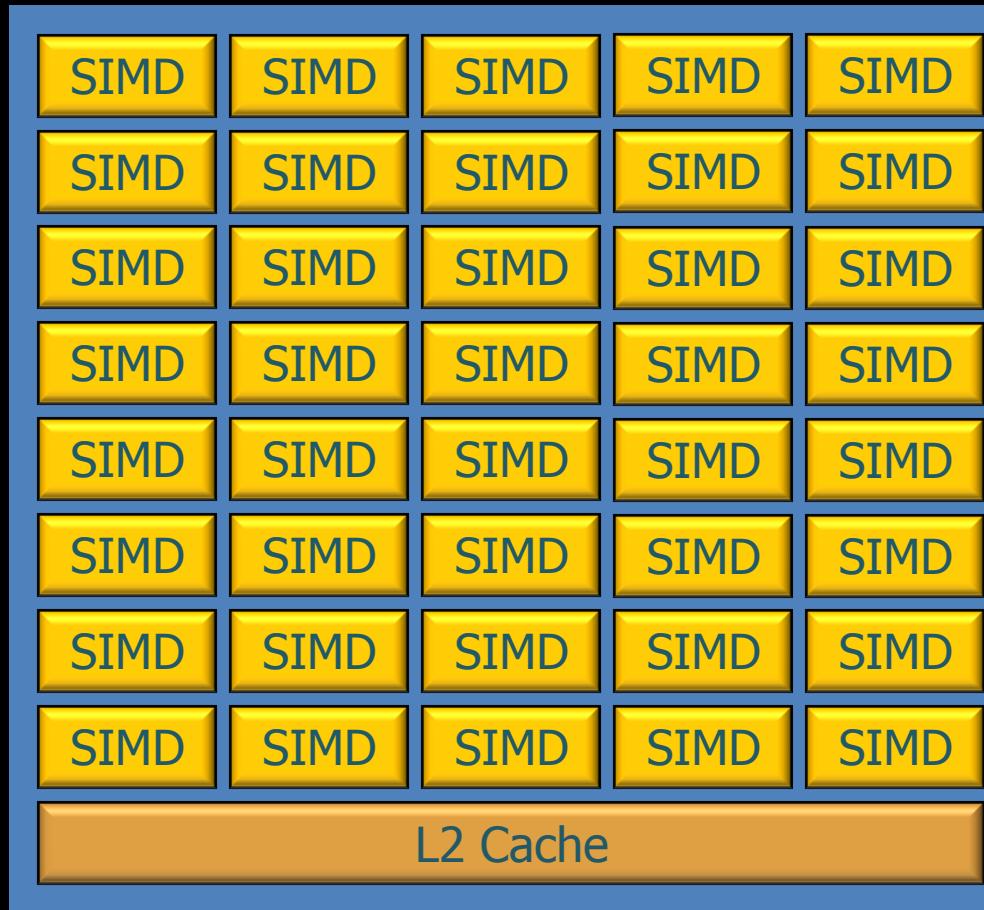
DirectCompute 를 구현하기 위한 HLSL

CPU vs GPU





4 Cores
4 float wide SIMD
3GHz
48-96GFlops
2x HyperThreaded
64kB \$L1/core
20GB/s to Memory
\$200
200W



32 Cores

32 Float wide

1GHz

1TeraFlop

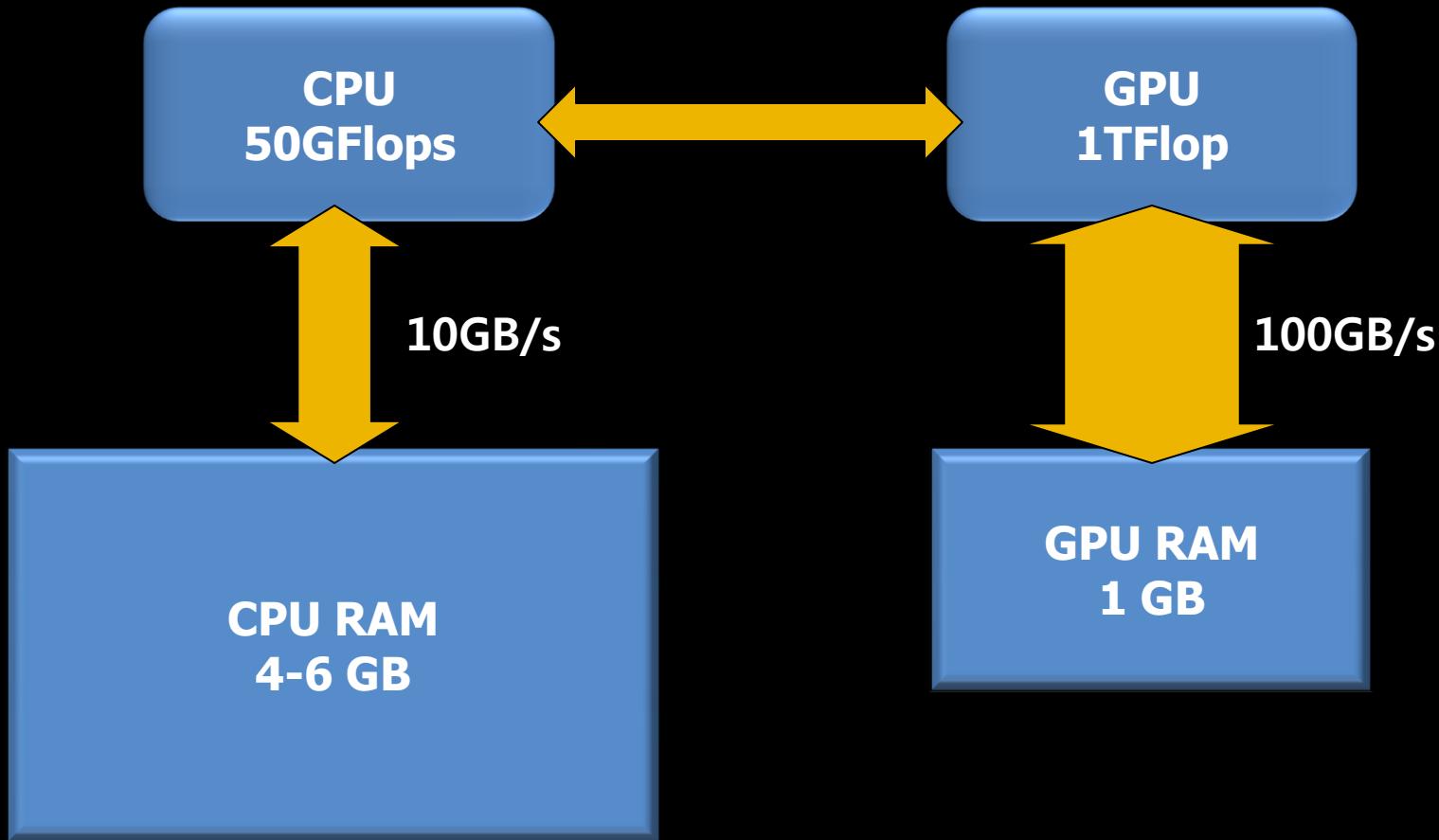
32x "HyperThreaded"

64kB \$L1/Core

150GB/s to Mem

\$200,

200W



막강한 GPU 활용을 위해

Tessellation 님이 등장

GPGPU 님이 등장

DirectCompute 사용하기



DirectCompute 초기화

```
hr = D3D11CreateDevice
(
    NULL,                                // default gfx adapter
    D3D_DRIVER_TYPE_HARDWARE, // use hw
    NULL,        // not sw rasterizer
    uCreationFlags, // Debug, Threaded, etc.
    NULL,        // feature levels
    0,           // size of above
    D3D11_SDK_VERSION, // SDK version
    ppDeviceOut,     // D3D Device
    &FeatureLevelOut, // of actual device
    ppContextOut ); // subunit of device
);
```

Example HLSL code

```
#define BLOCK_SIZE 256
StructuredBuffer gBuf1;
StructuredBuffer gBuf2;
RWStructuredBuffer gBufOut;

[numthreads(BLOCK_SIZE,1,1)]
void VectorAdd( uint3 id: SV_DispatchThreadID )
{
    gBufOut[id] = gBuf1[id] + gBuf2[id];
}
```

HLSL 컴파일.

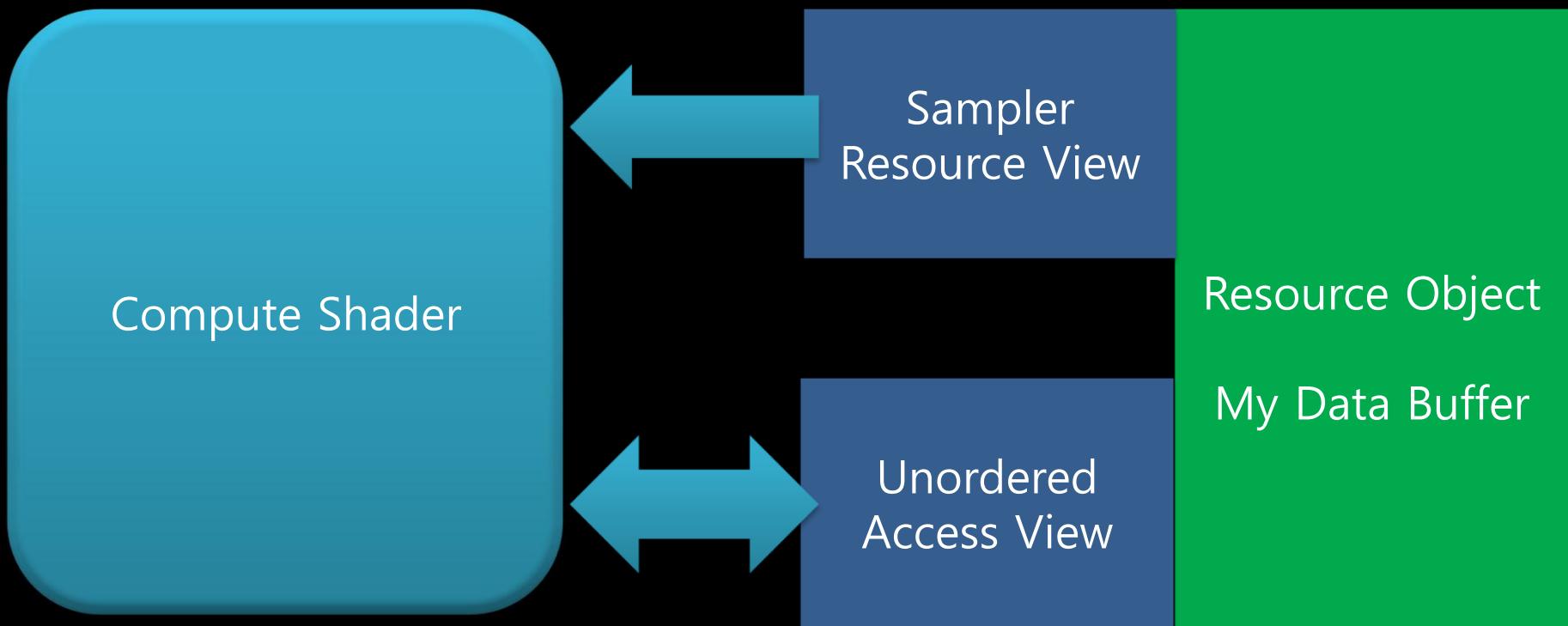
```
hr = D3DX11CompileFromFile(  
    "myCode.hlsl", // path to .hlsl file  
    NULL,  
    NULL,  
    "VectorAdd", // entry point  
    pProfile,  
    NULL,           // Flags  
    NULL,  
    NULL,  
    &pBlob,          // compiled shader  
    &pErrorBlob, // error log  
    NULL );
```

CS 생성 및 바인딩.

```
pD3D->CreateComputeShader(  
    pBlob->GetBufferPointer(),  
    pBlob->GetBufferSize(),  
    NULL,  
    &pMyShader );           // hw fmt
```

```
pD3D->CSSetShader(  
    pMyShader, NULL, 0 );
```

Resources



GPU 에 데이터 입력을 위한 버퍼 만들기.

```
D3D11_BUFFER_DESC descBuf;  
ZeroMemory( &descBuf, sizeof(descBuf) );  
desc.BindFlags = D3D11_BIND_UNORDERED_ACCESS;  
desc.StructureByteStride = uElementSize;  
desc.ByteWidth = uElementSize * uCount;  
desc.MiscFlags = D3D11_RESOURCE_MISC_BUFFER_STRUCTURED;  
  
pD3D->CreateBuffer( &desc, pInput, ppBuffer );
```

CS를 위한 리소스.

- Read/Write Buffers and Textures
- Structured Buffer
- Byte Address Buffer
- Unordered Access Buffer or Texture
- Append and Consume Buffer

리소스에 대한 접근

- Access By Byte Offset
- Access By Index

```
Texture2D<float4> myTexture;  
float4 myVar = myTexture[pos];
```

- Access By Mips Member

```
float4 myColor = myTexture.mips[0][float2(x,y)];
```

Atomic Functions

- InterlockedAdd
- InterlockedMin
- InterlockedMax
- InterlockedOr
- InterlockedAnd
- InterlockedXor
- InterlockedCompareStore
- InterlockedCompareExchange
- InterlockedExchange

View 설정하기

```
D3D11_UNORDERED_ACCESS_VIEW_DESC desc;  
ZeroMemory( &desc, sizeof(desc) );  
desc.ViewDimension = D3D11_UAV_DIMENSION_BUFFER;  
desc.Buffer.FirstElement = 0;  
desc.Format = DXGI_FORMAT_UNKNOWN;  
desc.Buffer.NumElements = uCount;  
  
pD3D->CreateUnorderedAccessView(  
    pBuffer,                      // Buffer view is into  
    &desc,                         // above data  
    &pMyUAV );                     // result
```

실행.

```
pD3D->CSSetUnorderedAccessViews(
```

```
    0,
```

```
    1,
```

```
&pMyUAV,
```

```
NULL );
```

```
pD3D->Dispatch( GrpsX, GrpsY, GrpsZ );
```

pDev11->Dispatch(3, 2, 1);

[numthreads(4, 4, 1)]
void MyCS(...)

00	01	02	03	00	01	02	03	00	01	02	03
10	11	12	13	10	11	12	13	10	11	12	13
20	21	22	23	20	21	22	23	20	21	22	23
30	31	32	33	30	31	32	33	30	31	32	33
00	01	02	03	00	01	02	03	00	01	02	03
10	11	12	13	10	11	12	13	10	11	12	13
20	21	22	23	20	21	22	23	20	21	22	23
30	31	32	33	30	31	32	33	30	31	32	33

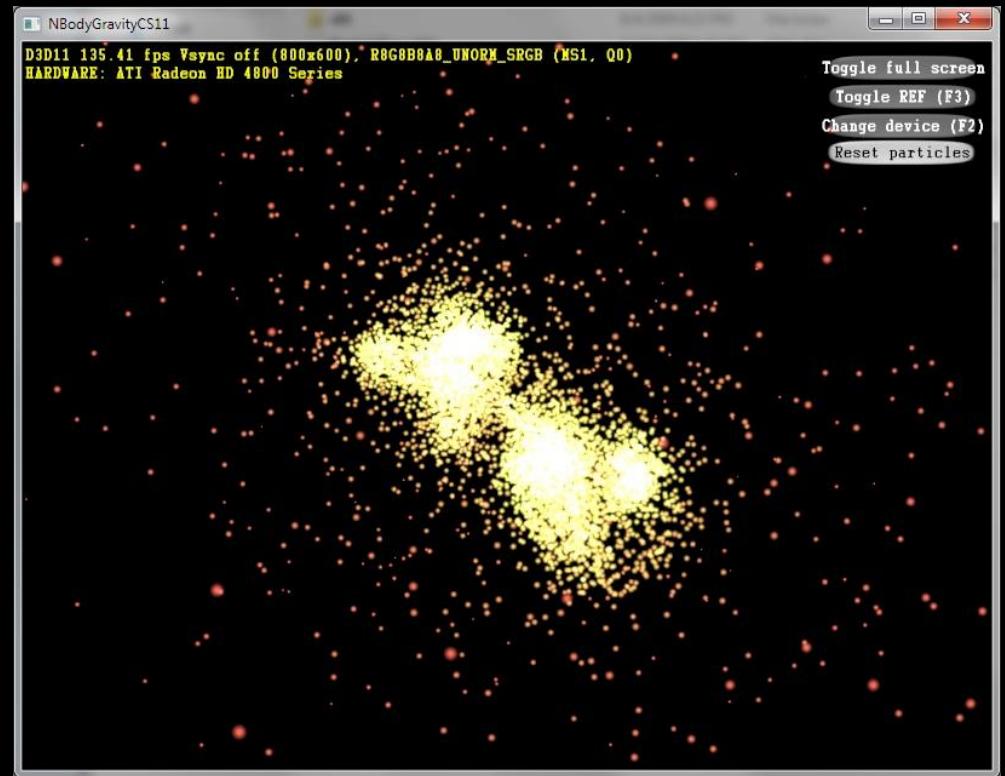
CPU에서 액세스 할 수 있는 버퍼를 생성

```
D3D11_BUFFER_DESC desc;  
ZeroMemory( &desc, sizeof(desc) );  
desc.CPUAccessFlags = D3D11_CPU_ACCESS_READ;  
desc.Usage = D3D11_USAGE_STAGING;  
desc.BindFlags = 0;  
desc.MiscFlags = 0;  
pD3D->CreateBuffer( &desc, NULL, &StagingBuf );
```

결과를 CPU 로 전송.

```
pD3D->CopyResource( debugbuf, pBuffer );
```

demo



pdcos

결과.

N-Body Demo App:

AMD Phenom II X4 940 3GHz + Radeon HD 5850

CPU 13.7GFlops Multicore SSE, not cache-aware

GPU 537GFlops DirectCompute

Intel Xeon E5410 2.33GHz + Radeon HD 5870

CPU 25.5GFlops Multicore SSE, not cache-aware

GPU 722GFlops DirectCompute

Feature	Compute Shader Model 4.0	Compute Shader Model 5.0	Benefits
Thread Dispatch	2D	3D	Replace multiple 2D thread arrays with a single 3D array
Thread Limit	768	1024	Execute up to 33% more simultaneous threads for better performance
Thread Group Shared Memory	16 kB	32 kB	Double the memory available for inter-thread communication
Shared Memory Access	256 byte write-only region	Full 32 kB read/write access	More efficient shared memory I/O
Atomic Operations	Not supported	Supported	Allows each thread to operate on protected memory locations – greatly simplifies implementation of CPU-based algorithms on GPU
Double Precision	Not supported	Supported	Allows 64-bit floating point precision for computations
Append/Consume Buffers	Not supported	Supported	Useful for building and accessing data in list or stack form
Unordered Access Views Bound to Compute Shader	1	8	Allows each compute shader to access multiple buffers for more flexibility
Unordered Access Views Bound to Pixel Shader	Not supported	8	Allows pixel shaders to access compute shader data for interoperability with rendering pipeline
Gather4	Not supported	Supported	Fetch adjacent data values from memory up to 4x faster

DirectX 11 소개

Tessellation

Compute Shader

Multi-Thread Rendering

게임에서의 멀티 코어 활용

Rendering

Resource loading

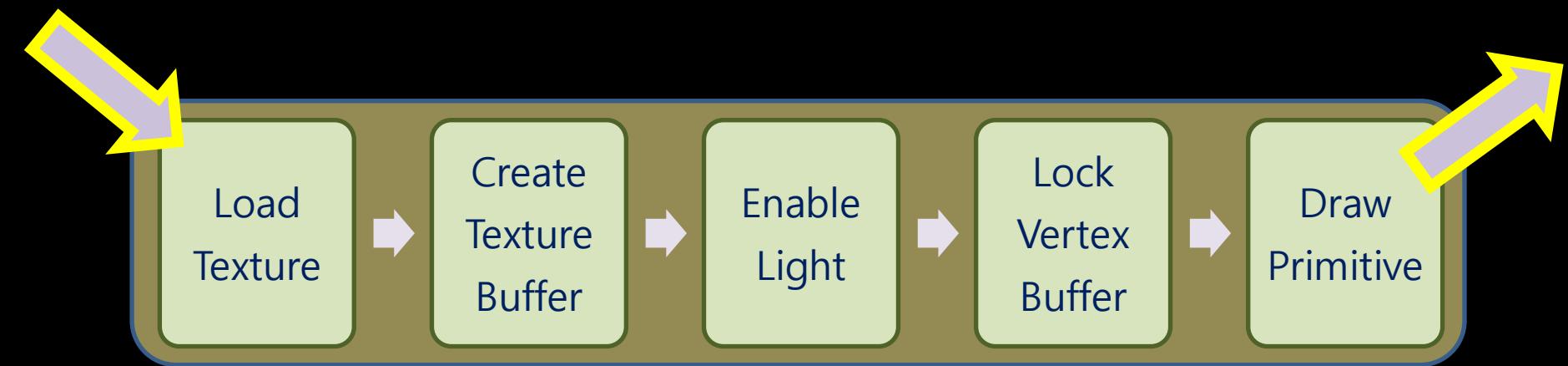
AI / Physics

Entity loading

각자의 길을 가다



기준의 렌더링 방법.



DirectX11 API 디자인의 목표

Asynchronous Resource resource loading & modification.

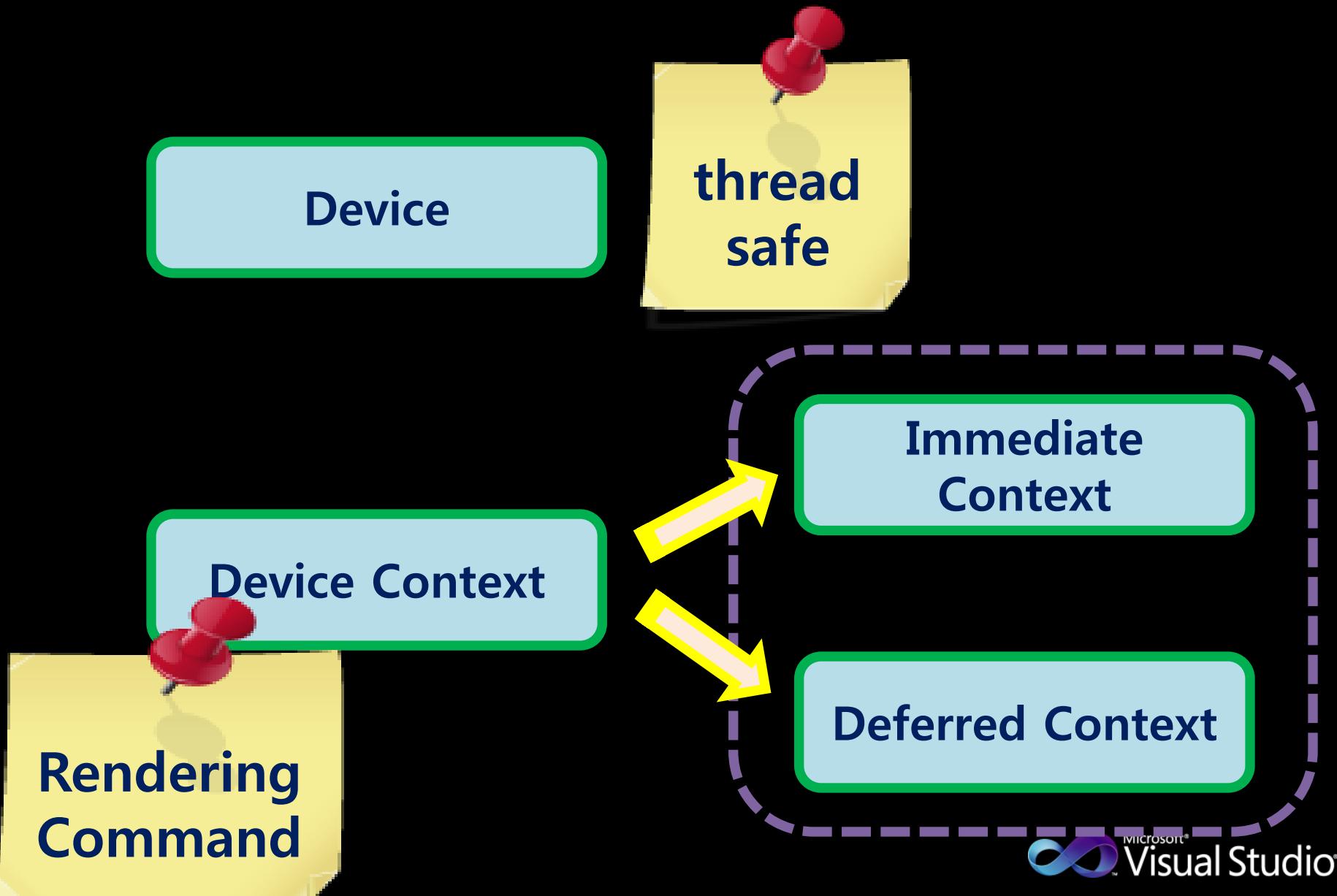
- ☞ Device와 driver 간의 오버헤드 감소.

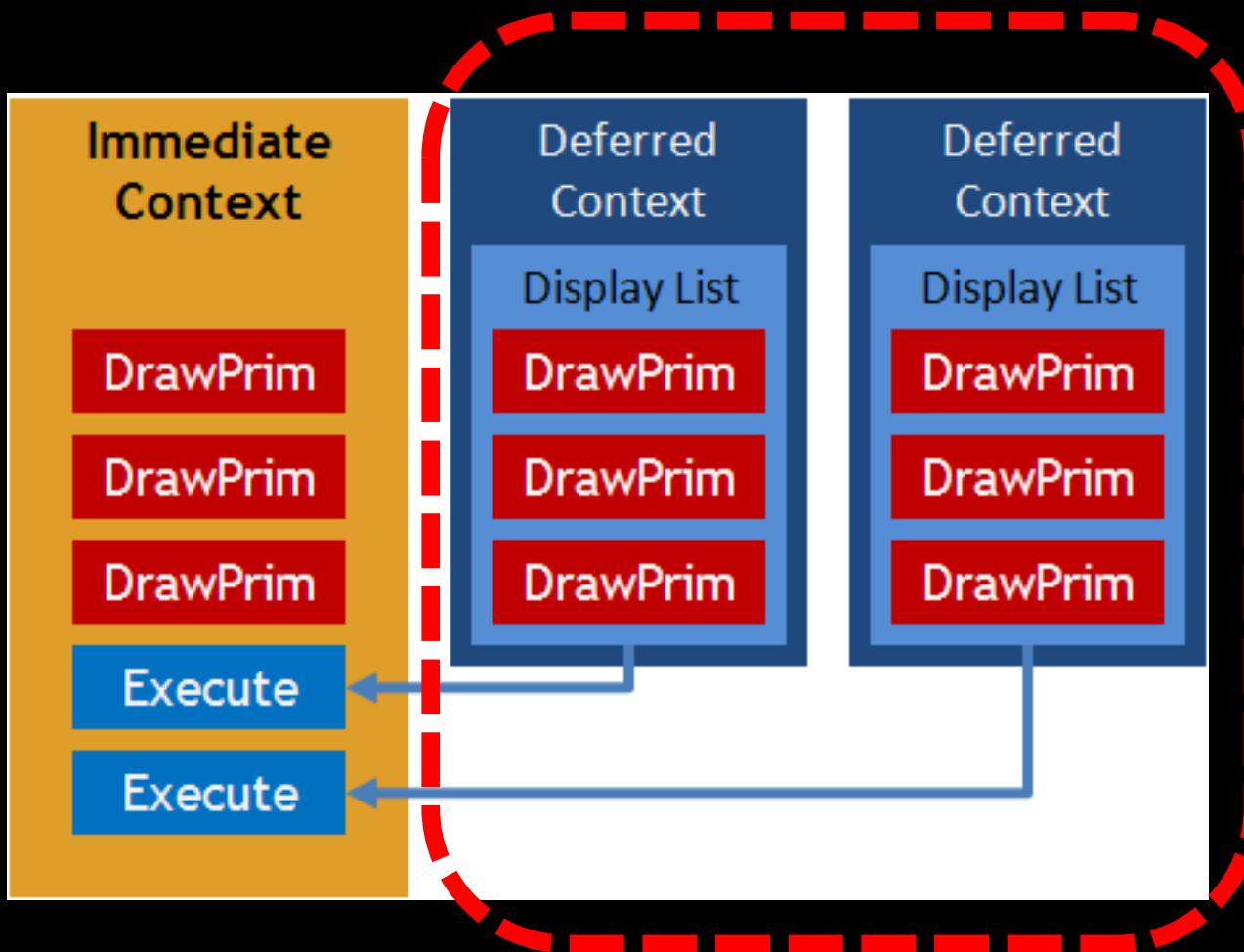
Multithreaded draw.

- ☞ 렌더링 작업의 여러 스레드들로 분산.

Display lists.

- ☞ 한번 기록한 커맨들의 재사용으로 인한 성능 향상.





```
int  
HANDLE*  
HANDLE*  
HANDLE*  
ID3D11DeviceContext**  
ID3D11CommandList**  
  
m_iRenderTargetCount;  
m_pDeferredThreadHandleArray;  
m_pBeginDeferredEventHandleArray;  
m_pEndDeferredEventHandleArray;  
m_ippDeferredContextPointerArray;  
m_ippCommandListPointerArray;
```

```

this->m_iRenderTargetCount = iThreadCount;
this->m_pDeferredThreadHandleArray = new HANDLE[ this->m_iRenderTargetCount ];
this->m_pBeginDeferredEventHandleArray = new HANDLE[ this->m_iRenderTargetCount ];
this->m_pEndDeferredEventHandleArray = new HANDLE[ this->m_iRenderTargetCount ];
this->m_ippDeferredContextPointerArray = new ID3D11DeviceContext*[ this->m_iRenderTargetCount ];
this->m_ippCommandListPointerArray = new ID3D11CommandList*[ this->m_iRenderTargetCount ];

HRESULT hr = E_FAIL;
ID3D11DeviceContext* ipResultPointer = 0x00;
for( int i = 0; i < this->m_iRenderTargetCount; ++i )
{
    //
    // DeferredContext 를 생성한다.
    //
    ipResultPointer = JinRenderUtil::CreateDeferredContext( this->m_ipGPU );
    this->m_ippDeferredContextPointerArray[ i ] = ipResultPointer;

    //
    // 스레드를 생성한다.
    // CREATE_SUSPENDED 옵션을 주어서 ResumeThread 에 전달될때까지 스레드 실행을 멈춘다.
    //
    this->m_pDeferredThreadHandleArray[ i ] = (HANDLE)_beginthreadex( NULL,
                                                                    0,
                                                                    DeferredProcForRenderPerScene,
                                                                    &i,
                                                                    CREATE_SUSPENDED,
                                                                    NULL );

    //
    // 스레드를 실행한다.
    //
    ResumeThread( this->m_pDeferredThreadHandleArray[ i ] );
}

```

```

unsigned int WINAPI DeferredProcForRenderPerScene( LPVOID lpParameter )
{
    HRESULT hr = E_FAIL;

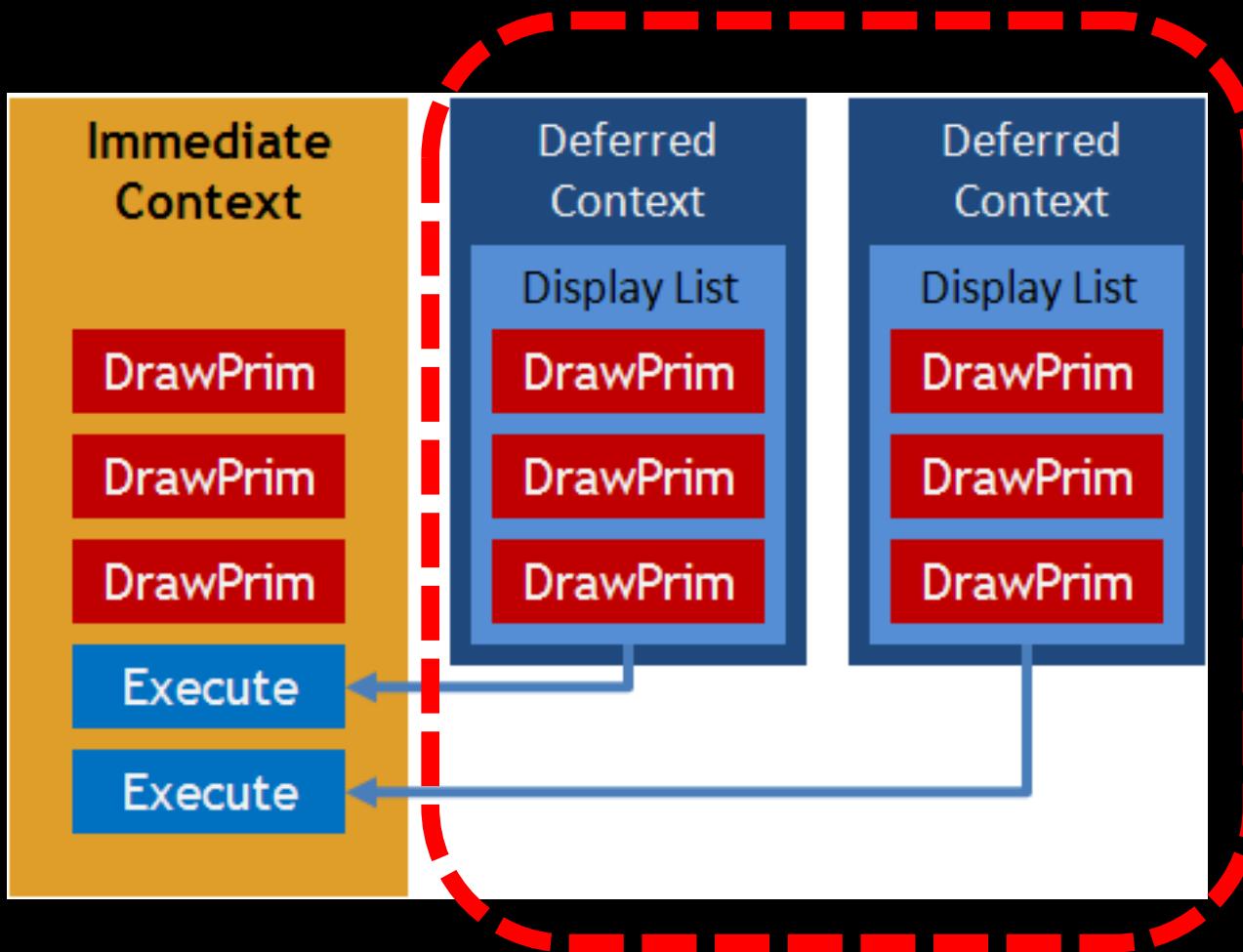
    const int iInstance = *( int* )lpParameter;
    ID3D11DeviceContext* ipDeferredContextPtr = Jin3D::GetInstance()->GetDeferredContext( iInstance );
    ID3D11CommandList* ipCommandListPtr = Jin3D::GetInstance()->GetCommandList( iInstance );

    for( ; ; )
    {
        // 해당 작업이 종료될때까지 기다린다.
        // ::WaitForSingleObject( Jin3D::GetInstance()->GetBeginDeferredEventHandle( iInstance ), INFINITE );

        // 렌더링 작업을 수행한다.
        // [REDACTED]
        // 커맨드들을 생성한다.
        // BOOL 플래그는 명령어들을 실행한 후
        // ImmediateContext 의 상태를 저장하고 복원할지를 결정하는 플래그이다.
        // 성능을 위해서 FALSE 를 사용한다.
        //
        hr = ipDeferredContextPtr->FinishCommandList( FALSE, &ipCommandListPtr );
        assert( hr == S_OK );

        // 커맨드 리스트들의 작업이 끝났음을 이벤트로 알린다.
        // ::SetEvent( Jin3D::GetInstance()->GetEndDeferredEventHandle( iInstance ) );
    }
}

```



```
for( int i = 0;      i < this->m_iThreadCount;      ++i )  
{      this->m_ipImmediateContext->ExecuteCommandList( this->m_ipCommandListPointerArray[ i ], TRUE );  
}
```

화면에 무엇인가를 나타내는 것!

Rendering Command를 생성하는 것!



이제는 이것이 멀티스레딩 방식으로 가능!



DirectX 도 점점 예쁘게 성장하고 있습니다!!!

참고 자료

- DirectX SDK Document & Sample
- GameFest 2008, 2009
- MSDN & ATI & NVIDIA web site
- Realtime-Rendering 책
- NVIDIA_Tessellation_GDC10
<http://microsoftpdc.com/Sessions/P09-16>
- http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter07.html

감사합니다..^^

Q & A


```

    mWVP;
    c4x4 mMW;

    sampler2D sDisplacement;

uniform float fDisplacementScale;
uniform float fDisplacementBias;

struct VsInput
{
    float4 vPosition : POSITION0;
    float2 vTexCoord : TEXCOORD0;
    float3 vNormal : NORMAL0;
};

struct VsOutput
{
    float4 vPosCS : POSITION;
    float2 vTexCoord : TEXCOORD0;
    float3 vPositionWS : TEXCOORD1;
    float3 vNormal : TEXCOORD2;
};

VsOutput VS( VsInput i )
{
    VsOutput o;

    o.vTexCoord = i.vTexCoord;
    o.vNormal = i.vNormal;

    // Sample displacement map:
    float fDisplacement = tex2Dlod( sDisplacement, float4( i.vTexCoord, 0, 0 ) ).r;
        fDisplacement = (fDisplacement * fDisplacementScale) + fDisplacementBias;

    o.vPositionWS = i.vPosition + (fDisplacement * i.vNormal);

    // Transform position to clip-space
    o.vPosCS = mul( mWVP, float4( o.vPositionWS, 1.0 ) );
    return o;
}

```

```

float4x4 mMVP;
sampler2D sDisplacement;
uniform float fDisplacementScale;
uniform float fDisplacementBias;

struct VsInput
{
    float3 vBarycentric : BLENDWEIGHT0;
    // Superprim vertex 0:
    float4 vPositionVert0 : POSITION0;
    float2 vTexCoordVert0 : TEXCOORD0;
    float3 vNormalVert0 : NORMAL0;
    // Superprim vertex 1:
    float4 vPositionVert1 : POSITION1;
    float2 vTexCoordVert1 : TEXCOORD1;
    float3 vNormalVert1 : NORMAL1;
    // Superprim vertex 2:
    float4 vPositionVert2 : POSITION2;
    float2 vTexCoordVert2 : TEXCOORD2;
    float3 vNormalVert2 : NORMAL2;
};

struct VsOutput
{
    float4 vPosCS : POSITION;
    float2 vTexCoord : TEXCOORD0;
    float3 vPositionWS : TEXCOORD1;
    float3 vNormal : TEXCOORD2;
};

sOutput VS( VsInput i )
{
    VsOutput o;

    // Compute new position based on the barycentric coordinates:
    float3 vPosTessOS = i.vPositionVert0.xyz * i.vBarycentric.x +
        i.vPositionVert1.xyz * i.vBarycentric.y +
        i.vPositionVert2.xyz * i.vBarycentric.z;

    // Compute new texture coordinates based on the barycentric coordinates:
    o.vTexCoord = i.vTexCoordVert0.xy * i.vBarycentric.x +
        i.vTexCoordVert1.xy * i.vBarycentric.y +
        i.vTexCoordVert2.xy * i.vBarycentric.z;

    // Compute new normal based on the barycentric coordinates:
    o.vNormal = i.vNormalVert0.xyz * i.vBarycentric.x +
        i.vNormalVert1.xyz * i.vBarycentric.y +
        i.vNormalVert2.xyz * i.vBarycentric.z;

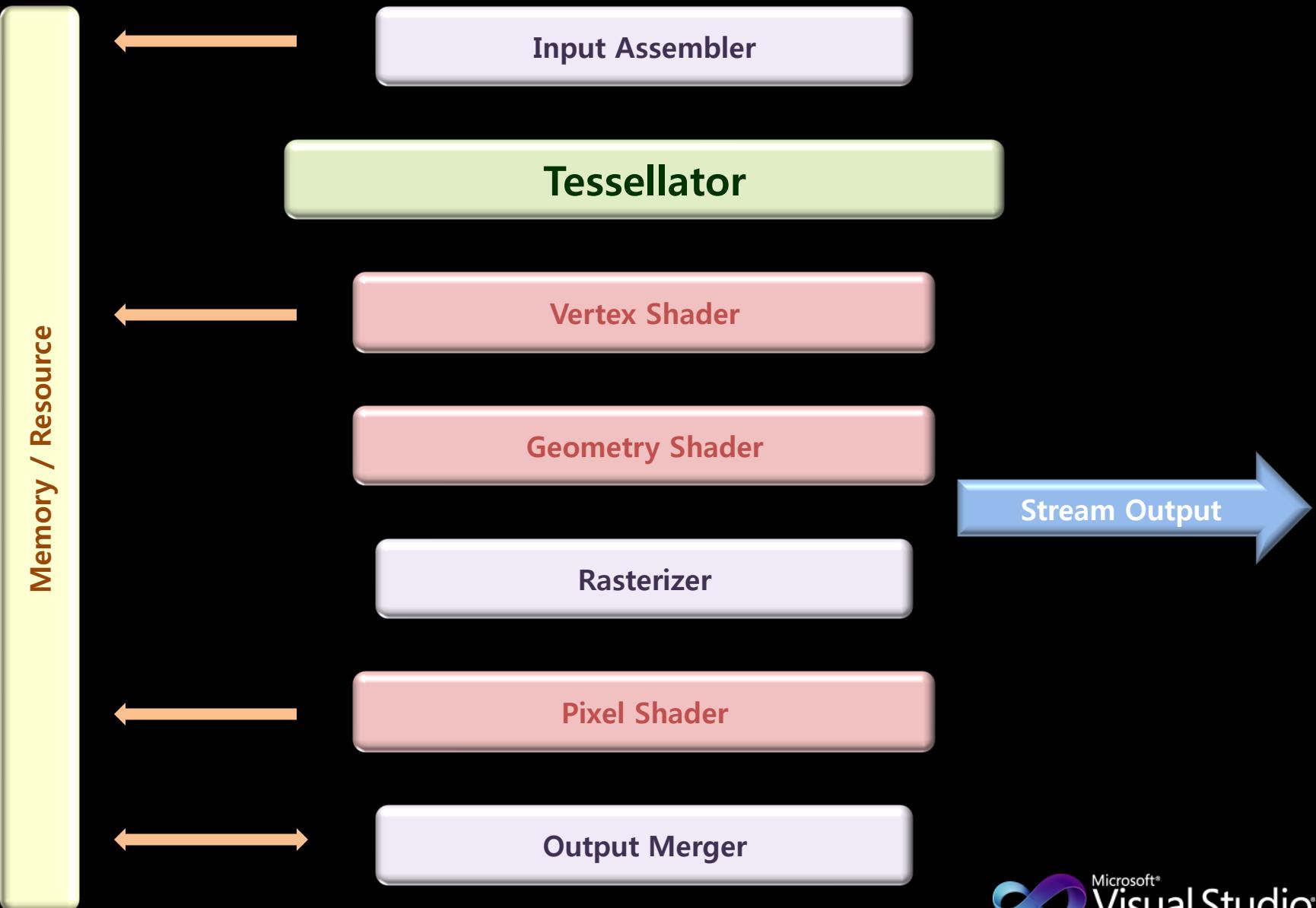
    // Sample displacement map:
    float fDisplacement = tex2Dlod( sDisplacement, float4( o.vTexCoord, 0, 0 ) ).r;
    fDisplacement = (fDisplacement * fDisplacementScale) + fDisplacementBias;

    o.vPositionWS = vPosTessOS + ( fDisplacement * o.vNormal);

    // Transform position to clip-space
    o.vPosCS = mul( mMVP, float4( o.vPositionWS, 1.0 ) );
    return o;
}

```

DirectX10 에서의 Tessellation.



```

struct VsTfmInput
{
    float4 vPosOS : POSITION0; // (x,y,z) - 3D position, w - vertex ID (precomputed)
    float2 vTexCoord: TEXCOORD0;
    float3 vNormal : NORMAL0;
};
struct VsTfmOutput
{
    float4 vPos : POSITION;
    float fVertexID : TEXCOORD0;
    float3 vPosCS : TEXCOORD1;
};

VsTfmOutput VSRenderTransformedDisplaced( VsTfmInput i )
{
    VsTfmOutput o;
    int nVertexID = floor( i.vPosOS.w );
    int nTextureWidth = g_vTformVertsMapSize.x;

    // Compute row and column of the position in 2D texture
    float2 vPos = float2( nVertexID % nTextureWidth, nVertexID / nTextureWidth );
    vPos /= g_vTformVertsMapSize.xy;
    vPos.y = 1.0 - vPos.y;
    vPos = vPos * 2 - 1.0; // Move to [-1; 1] range

    o.vPos = float4( vPos.xy, 0, 1 );

    // Propagate the vertex ID to the pixel shader
    o.fVertexID = i.vPosOS.w;

    // Displace input vertex:
    float4 vPositionCS = float4( DisplaceVertex( i.vPosOS, i.vTexCoord, i.vNormal ), 1 );

    // Transform vertex position to screen-space
    vPositionCS = mul( vPositionCS, g_mWorldView );
    vPositionCS /= vPositionCS.w;

    o.vPosCS = vPositionCS.xyz;
}

```

```

struct VsTFInput
{
    float4 vPositionOS : POSITION0; // (xyz) - camera-space position, w - vertex ID
};

struct VsTFOOutput
{
    float4 vPositionCS : POSITION;
    float fTessFactor : TEXCOORD0;
};

//-----
VsTFOOutput VSCalculateTFs( VsTFInput i )
{
    VsTFOOutput o;

    // Current vertex ID:
    int nCurrentVertID = (int) ( i.vPositionOS.w );

    // Determine the ID of the edge neighbor's vertex (remember that this is done with
    // non-indexed primitives, so basically if the current vertex is v0 ,then we have
    // edges: v0->v1, v1->v2, v2->v0

    int nCurrentVertEdgeID = nCurrentVertID - 3 * floor( floatnCurrentVertID /
                                                float( 3 ) );

    int nEdgeVert0ID = nCurrentVertID;      // this works if current vertex is v0 or v1
    int nEdgeVert1ID = nCurrentVertID + 1;

    if ( nCurrentVertEdgeID == 0 )
    {
        nEdgeVert0ID = nCurrentVertID + 1;
    }
    else if ( nCurrentVertEdgeID == 1 )
    {
        nEdgeVert0ID = nCurrentVertID + 1;
    }
    else if ( nCurrentVertEdgeID == 2 ) // In case of v2 we need to wrap around to v0
    {
        nEdgeVert0ID = nCurrentVertID - 2;
    }

    // Compute the fetch coordinates to fetch transformed positions
    // for these two vertices to compute their edge statistics:
    int    nTextureWidth      = g_vTformVertsMapSize.x;

    // Vertex0: nCurrentVertID, compute row and column of the position in 2D texture:
    float2 vVert0Coords     = float2( nCurrentVertID % nTextureWidth,
                                         nCurrentVertID / nTextureWidth );
    vVert0Coords /= g_vTformVertsMapSize.xy;
}

```

```

// Vertex1: nEdgeVertOID, compute row and column of the position in 2D texture:
float2 vVert1Coords    = float2( nEdgeVertOID % nTextureWidth,
                                nEdgeVertOID / nTextureWidth );
vVert1Coords /= g_vTformVertsMapSize.xy;

// Fetch transformed positions for these IDs:
float4 vVert0Pos = tex2Dlod( stformVerts, float4( vVert0Coords, 0, 0 ) );
float4 vVert1Pos = tex2Dlod( stformVerts, float4( vVert1Coords, 0, 0 ) );

// Swap vertices to make sure that we have the same edge direction
// regardless of their triangle order (based on vertex ID):
if ( vVert0Pos.w > vVert1Pos.w )
{
    float4 vTmpVert = vVert0Pos;
    vVert0Pos = vVert1Pos;
    vVert1Pos = vTmpVert;
}

// Use the distance from the camera to determine the tessellation factor:
float fEdgeDepth = 0.5 * ( ( vVert1Pos.z ) + ( vVert0Pos.z ) );

float fTessFactor = clamp( fEdgeDepth / g_fMaxCameraDistance,
                           0, 1 ); // Map to 0-1 range
fTessFactor = (1 - fTessFactor) * 15; // map to 0-15 range and invert it
// (higher to lower)

const float fMinTessFactor = 1.0;
const float fMaxTessFactor = 14.99999;

// Clamp to the correct range
o.fTessFactor = clamp( fTessFactor, fMinTessFactor, fMaxTessFactor );

// Compute output position for rendering into a tessellation factors texture
int nVertexID = floor( i.vPositionOS.w );

// Compute row and column of the position in 2D texture
float2 vOutputPos = float2( nVertexID % nTextureWidth, nVertexID / nTextureWidth );
vOutputPos /= g_vTformVertsMapSize.xy;
vOutputPos.y = 1.0 - vOutputPos.y;
vOutputPos = vOutputPos * 2 - 1.0; // Move to [-1; 1] range

o.vPositionCS = float4( vOutputPos.xy, 0, 1 );

return o;
} // End of VsTfmOutput VSCalculateTessellations(..)

// Pixel shader for rendering the tessellation factor into an R2VB buffer.
struct PsTFInput
{
    float fTessFactor : TEXCOORD0;
};

//-----
float4 PSCalculateTessellations( PsTFInput i ): COLOR0
{
    return i.fTessFactor.xxxx;
}

//.....
technique CalculateTessellationFactors
{

```

무엇이 문제인가?

인덱스 기반의 렌더링 미지원
→ 별도의 VB 필요

더 많은 리소스 소모.
→ 뷰좌표를 저장하기 위한 텍스쳐 메모리

Multi Pass
→ 성능 ↓

DX11 샘플

```

VS_OUTPUT_HS_INPUT VS( VS_INPUT i )
{
    VS_OUTPUT_HS_INPUT Out;

    // Compute position in world space
    float4 vPositionWS = mul( i.inPositionOS.xyz, g_mWorld );

    // Compute denormalized light vector in world space
    float3 vLightWS = g_LightPosition.xyz - vPositionWS.xyz;
    // Need to invert Z for correct lighting
    vLightWS.z = -vLightWS.z;

    // Propagate texture coordinate through:
    Out.texCoord = i.inTexCoord * g_fBaseTextureRepeat.x;

    // Transform normal, tangent and binormal vectors from object
    // space to homogeneous projection space and normalize them
    float3 vNormalWS  = mul( i.vInNormalOS, (float3x3) g_mWorld );
    float3 vTangentWS = mul( i.vInTangentOS, (float3x3) g_mWorld );
    float3 vBinormalWS = mul( i.vInBinormalOS, (float3x3) g_mWorld );

    // Normalize tangent space vectors
    vNormalWS  = normalize( vNormalWS );
    vTangentWS = normalize( vTangentWS );
    vBinormalWS = normalize( vBinormalWS );

    // Output normal
    Out.vNormal = vNormalWS;

    // Calculate tangent basis
    float3x3 mWorldToTangent = float3x3( vTangentWS, vBinormalWS, vNormalWS );

    // Propagate the light vector (in tangent space)
    Out.vLightTS = mul( mWorldToTangent, vLightWS );

#if ADD_SPECULAR==1
    // Compute and output the world view vector (unnormalized)
    float3 vViewWS = g_vEye - vPositionWS;
    Out.vViewTS = mul( mWorldToTangent, vViewWS );
#endif
}

```

```
// Write world position
Out.vWorldPos = float3( vPositionWS.xyz );

#if DISTANCE_ADAPTIVE_TESSELLATION==1
// Min and max distance should be chosen according to scene quality requirements
const float fMinDistance = 20.0f;
const float fMaxDistance = 250.0f;

// Calculate distance between vertex and camera, and a vertex distance factor issued from it
float fDistance = distance( vPositionWS.xyz, g_vEye );
Out.fVertexDistanceFactor = 1.0 - clamp( ( ( fDistance - fMinDistance ) / ( fMaxDistance - fMinDistance ) ),
                                         0.0, 1.0 - g_vTessellationFactor.z/g_vTessellationFactor.x);
#endif

return Out;
}
```

```

//-----
// Hull shader
//-----
HS_CONSTANT_DATA_OUTPUT ConstantsHS( InputPatch<VS_OUTPUT_HS_INPUT, 3> p, uint PatchID : SV_PrimitiveID )
{
    HS_CONSTANT_DATA_OUTPUT output = (HS_CONSTANT_DATA_OUTPUT)0;
    float4 vEdgeTessellationFactors;

#if DENSITY_BASED_TESSELLATION==1

    // Retrieve edge density from edge density buffer (swizzle required to match vertex ordering)
    vEdgeTessellationFactors = g_DensityBuffer.Load( PatchID ).yzxw;

    // Multiply them by global tessellation factor
    vEdgeTessellationFactors *= g_vTessellationFactor.xyyy;

#else

    // Tessellation level fixed by variable
    vEdgeTessellationFactors = g_vTessellationFactor.xyyy;

#endif

#if DISTANCE_ADAPTIVE_TESSELLATION==1

    // Calculate edge scale factor from vertex scale factor: simply compute
    // average tess factor between the two vertices making up an edge
    float3 fScaleFactor;
    fScaleFactor.x = 0.5 * ( p[1].fVertexDistanceFactor + p[2].fVertexDistanceFactor );
    fScaleFactor.y = 0.5 * ( p[2].fVertexDistanceFactor + p[0].fVertexDistanceFactor );
    fScaleFactor.z = 0.5 * ( p[0].fVertexDistanceFactor + p[1].fVertexDistanceFactor );

    // Scale edge factors
    vEdgeTessellationFactors *= fScaleFactor.xyzx;

#endif

    // Assign tessellation levels
    output.Edges[0] = vEdgeTessellationFactors.x;
    output.Edges[1] = vEdgeTessellationFactors.y;
    output.Edges[2] = vEdgeTessellationFactors.z;
    output.Inside = vEdgeTessellationFactors.w;

    return output;
}

```

```

//-----
// Domain Shader
//-----
[domain("tri")]
DS_OUTPUT DS( HS_CONSTANT_DATA_OUTPUT input, float3 BarycentricCoordinates : SV_DomainLocation,
    const OutputPatch<HS_CONTROL_POINT_OUTPUT, 3> TrianglePatch )
{
    DS_OUTPUT output = (DS_OUTPUT)0;

    // Interpolate world space position with barycentric coordinates
    float3 vWorldPos = BarycentricCoordinates.x * TrianglePatch[0].vWorldPos +
        BarycentricCoordinates.y * TrianglePatch[1].vWorldPos +
        BarycentricCoordinates.z * TrianglePatch[2].vWorldPos;

    // Interpolate world space normal and renormalize it
    float3 vNormal = BarycentricCoordinates.x * TrianglePatch[0].vNormal +
        BarycentricCoordinates.y * TrianglePatch[1].vNormal +
        BarycentricCoordinates.z * TrianglePatch[2].vNormal;
    vNormal = normalize( vNormal );

    // Interpolate other inputs with barycentric coordinates
    output.texCoord = BarycentricCoordinates.x * TrianglePatch[0].texCoord +
        BarycentricCoordinates.y * TrianglePatch[1].texCoord +
        BarycentricCoordinates.z * TrianglePatch[2].texCoord;
    float3 vLightTS = BarycentricCoordinates.x * TrianglePatch[0].vLightTS +
        BarycentricCoordinates.y * TrianglePatch[1].vLightTS +
        BarycentricCoordinates.z * TrianglePatch[2].vLightTS;
}

```

```

// Calculate MIP level to fetch normal from
float fHeightMapMIPLevel = clamp( ( distance( vWorldPos, g_vEye ) - 100.0f ) / 100.0f, 0.0f, 6.0f);

// Sample normal and height map
float4 vNormalHeight = g_nmhTexture.SampleLevel( g_samLinear, output.texCoord, fHeightMapMIPLevel );

// Displace vertex along normal
vWorldPos += vNormal * ( g_vDetailTessellationHeightScale.x * ( vNormalHeight.w-1.0 ) );

// Transform world position with viewprojection matrix
output.vPosition = mul( float4( vWorldPos.xyz, 1.0 ), g_mViewProjection );

#if PERPIXEL_DIFFUSE_LIGHTING==1
    // Per-pixel lighting: pass tangent space light vector to pixel shader
    output.vLightTS = vLightTS;

    #if ADD_SPECULAR==1
        // Also pass tangent space view vector
        output.vViewTS = BarycentricCoordinates.x * TrianglePatch[0].vViewTS +
                        BarycentricCoordinates.y * TrianglePatch[1].vViewTS +
                        BarycentricCoordinates.z * TrianglePatch[2].vViewTS;
    #endif
#else
    // Per-vertex lighting
    float3 vNormalTS = normalize( vNormalHeight.xyz * 2.0 - 1.0 );
    vLightTS = normalize( vLightTS );
    output.vDiffuseColor = saturate( dot( vNormalTS, vLightTS ) ) * g_materialDiffuseColor;
#endif

#if DENSITY_BASED_TESSELLATION==1 && DEBUG_VIEW==1 && PERPIXEL_DIFFUSE_LIGHTING==1
    output.vDiffuseColor = BarycentricCoordinates.x*input.VertexDensity[0] +
                          BarycentricCoordinates.y*input.VertexDensity[1] +
                          BarycentricCoordinates.z*input.VertexDensity[2];
#endif

#if DEBUG_VIEW==2
    float fRed =      saturate( fHeightMapMIPLevel );
    float fGreen =     saturate( ( fHeightMapMIPLevel-1.0 ) / 2.0 );
    float fBlue =      saturate( ( fHeightMapMIPLevel-2.0 ) / 4.0 );
    output.vDiffuseColor = float4(fRed, fGreen, fBlue,0);
#endif

```