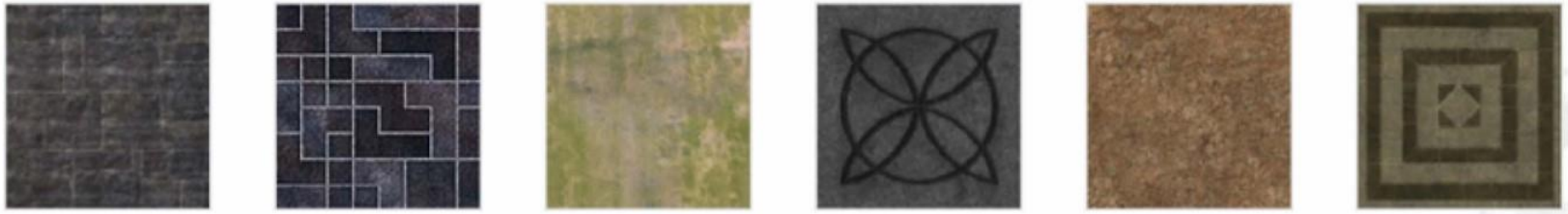
Game Programming with DirectX

텍스쳐(Texture)

텍스쳐(Texture)

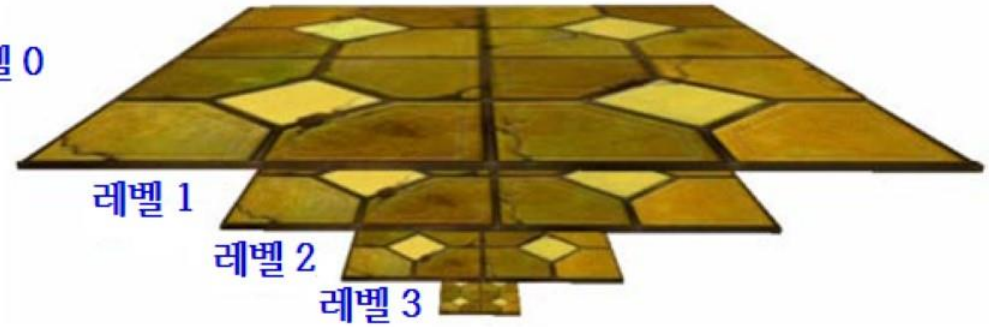
- **텍스쳐(Texture)**
  - 실세계의 객체들의 표면은 완전하지 않음, 균일하지 않음, 동일하지 않음
  - 세밀한(질감을 갖는) 표면들을 완전하게 모델링하는 것은 불가능(왜?)



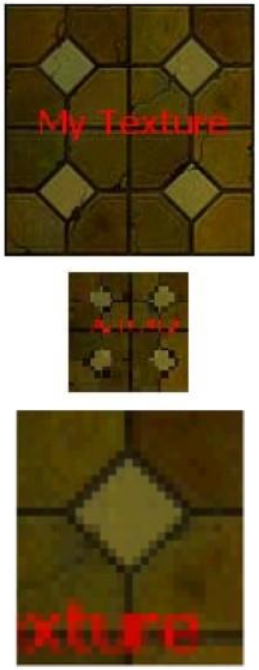
- 텍스쳐링(Texturing, Texture Mapping)
  - 표면을 모델링하지 않고 표면에 이미지(Texture)를 그리는 방법
- 텍스쳐 리소스
  - 텍셀(Texel) = 텍스쳐의 픽셀(Texture + Pixel)
  - 텍스쳐 = 텍셀(Texel)을 저장하기 위한 구조화된 데이터의 집합
  - 쉐이더가 텍스쳐를 읽을 때 기본적으로 샘플러(Sampler)가 필요함
  - 텍스쳐는 크기가 지정된 구조화된 리소스
- 텍스쳐의 형식(Type)
  - 1차원(1D), 2차원(2D), 3차원(3D)
  - 텍스쳐 큐브(Texture Cube)는 6개의 2차원 텍스쳐 배열
  - 각 차원의 텍스쳐는 밉맵(Mipmap)을 가질 수 있음(Level-Of-Detail)
  - Direct3D는 텍스쳐 배열과 다중샘플 텍스쳐를 제공

텍스쳐(Texture)

- **밉맵(MIP Map)**
  - MIP(Mulum In Parvo): 라틴어로 "Much in Small"이란 의미
  - 밉맵은 해상도가 연속적으로 절반인 순서화된 일련의 텍스쳐들을 의미
  - 밉맵을 동적으로 생성할 수도 있지만 텍스쳐 로드(On/Off-line)시 생성할 수 있음
  - 밉맵 생성시 고수준의 저해상도 이미지를 생성하기 위해 필터링 알고리즘 사용



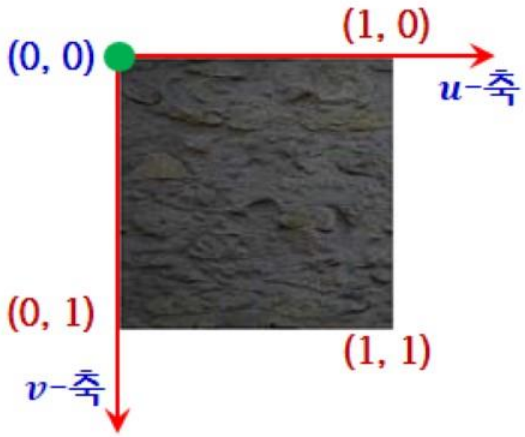
  - **LOD(Level Of Detail)**
    카메라까지 거리에 따라 자동적으로 밉맵 레벨 선택
    밉맵 레벨의 텍스쳐를 사용하여 텍스쳐 매핑



- **확대/축소의 문제**
  - 다각형의 크기가 64x64, 텍스쳐 크기 128x128
    단순히 축소하면 세밀함을 잃어 버림
  - 다각형의 크기가 521x512, 텍스쳐 크기 128x128
    단순 확대를 하면 계단 현상(Aliasing) 발생
  - 정교한 알고리즘을 사용하여 확대/축소의 문제 해결
    실시간으로 확대/축소 알고리즘을 사용하면(?)

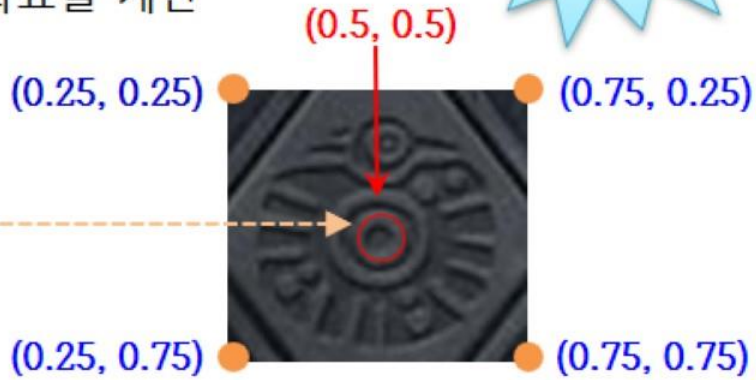
텍스쳐(Texture)

- **텍스쳐 좌표(Texture Coordinates)**
  - 텍스쳐 좌표는 0.0 ~ 1.0의 정규화된 좌표계를 사용
  - 래스터라이저는 보간으로 각 픽셀의 텍스쳐 좌표를 계산

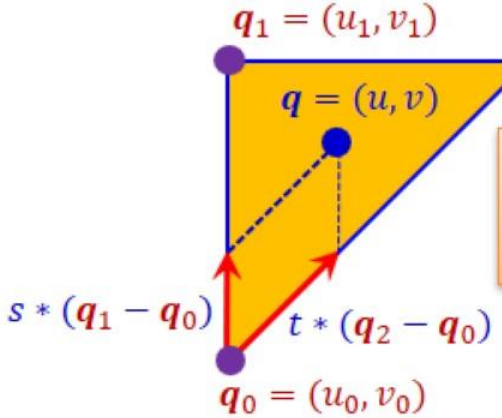


텍스쳐 이미지

샘플링



u = 텍스쳐(픽셀) x-좌표 ÷ 이미지 가로 크기
v = 텍스쳐(픽셀) y-좌표 ÷ 이미지 세로 크기

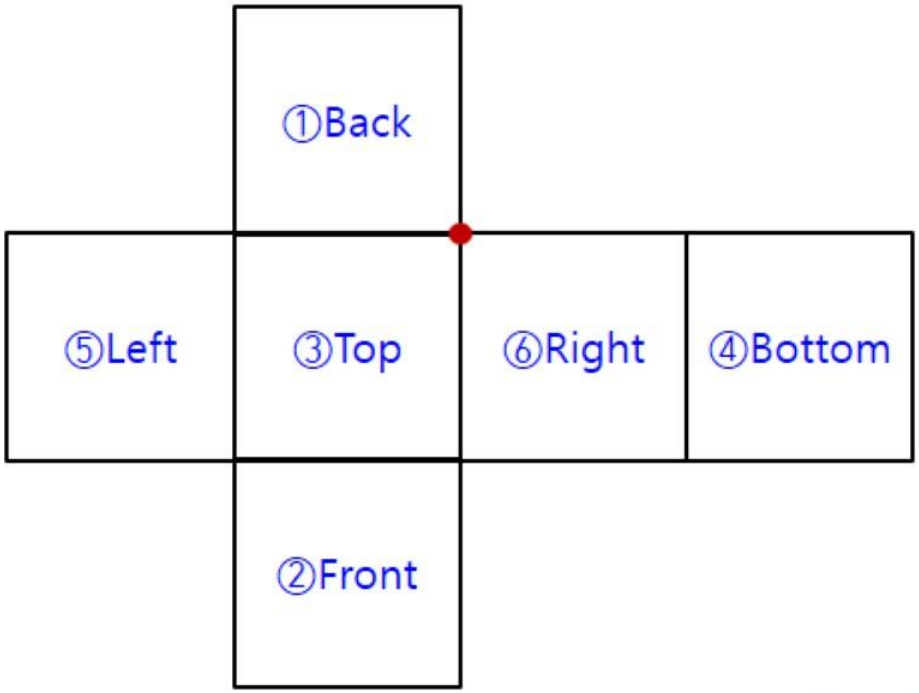


$$q = (u, v)$$
$$= q_0 + t * (q_2 - q_0) + s * (q_1 - q_0)$$
$$s \geq 0, t \geq 0, s + t \leq 1.0$$

```
class CTexturedVertex
{
    XMFLOAT3 m_xmf3Position;
    XMFLOAT2 m_xmf2Texture;
};
```

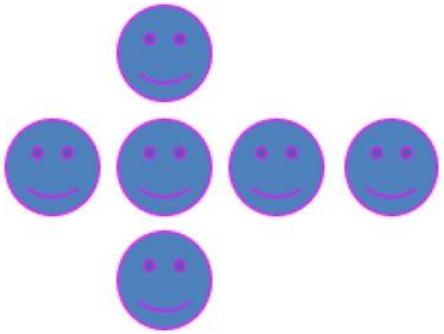
텍스쳐(Texture)

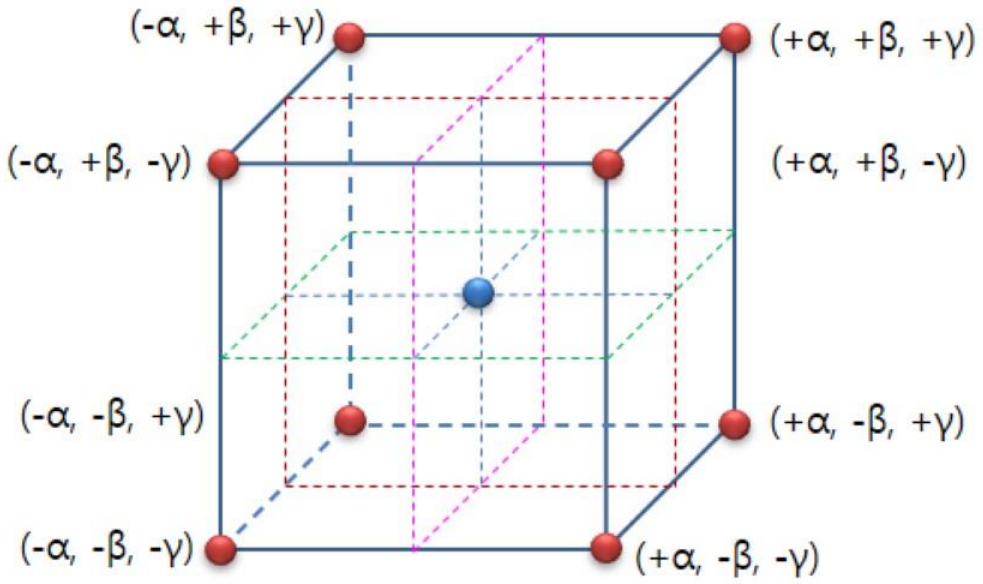
- 텍스쳐 좌표(Texture Coordinates)



D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST
Index Buffer: 36개 인덱스
사각형 6개: 삼각형 12개

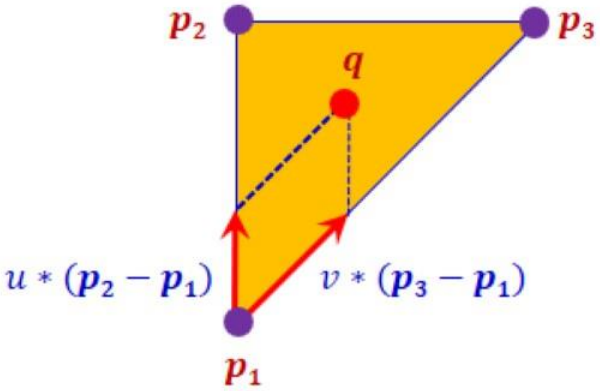
텍스쳐 매핑: 정점의 개수는?
8개(?): 가능한가?
24개: 사각형 6개, 6 x 4





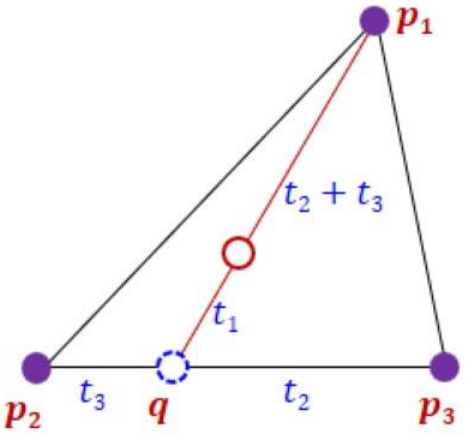
텍스쳐(Texture)

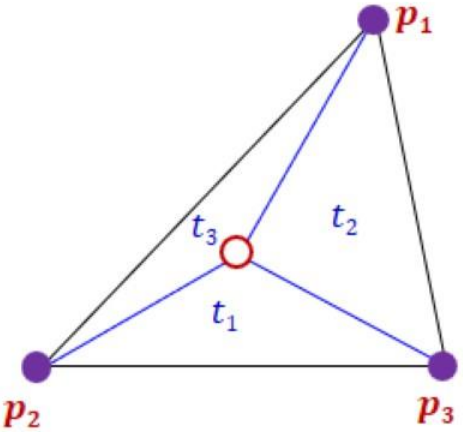
- **중심 좌표계(Barycentric coordinates)**



중심 좌표계는 삼각형의 꼭지점으로 삼각형 내부의 한 점을 정의한다.
중심 좌표계는 삼각형 내부의 한 점을 정의하기 위하여 세 개의 실수 (t_1, t_2, t_3)를 사용한다($t_1 + t_2 + t_3 = 1.0$).
실제로 중심 좌표계는 두 개의 실수를 사용한다(왜?).

$$
\begin{aligned}
\boldsymbol{q} &= \boldsymbol{p}_1 + u*(\boldsymbol{p}_2 - \boldsymbol{p}_1) + v*(\boldsymbol{p}_3 - \boldsymbol{p}_1) \\
&= \boldsymbol{p}_1 + u*\boldsymbol{p}_2 - u*\boldsymbol{p}_1 + v*\boldsymbol{p}_3 - v*\boldsymbol{p}_1 \\
&= (1-u-v)*\boldsymbol{p}_1 + u*\boldsymbol{p}_2 + v*\boldsymbol{p}_3 \\
&= w*\boldsymbol{p}_1 + u*\boldsymbol{p}_2 + v*\boldsymbol{p}_3 \\
u \geq 0, v \geq 0, u+v \leq 1.0, w &= 1-u-v
\end{aligned}
$$





삼각형 $(\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3)$에 대하여 중심 좌표계 (u, v)를 사용하여 점 $\boldsymbol{q}$를 표현한다고 하자.
u는 점 $\boldsymbol{p}_2$의 점 $\boldsymbol{q}$에 대한 가중치를 나타낸다.
v는 점 $\boldsymbol{p}_3$의 점 $\boldsymbol{q}$에 대한 가중치를 나타낸다.
$\boldsymbol{p}_1$의 점 $\boldsymbol{q}$에 대한 가중치는 $w = (1.0 - u - v)$이다.

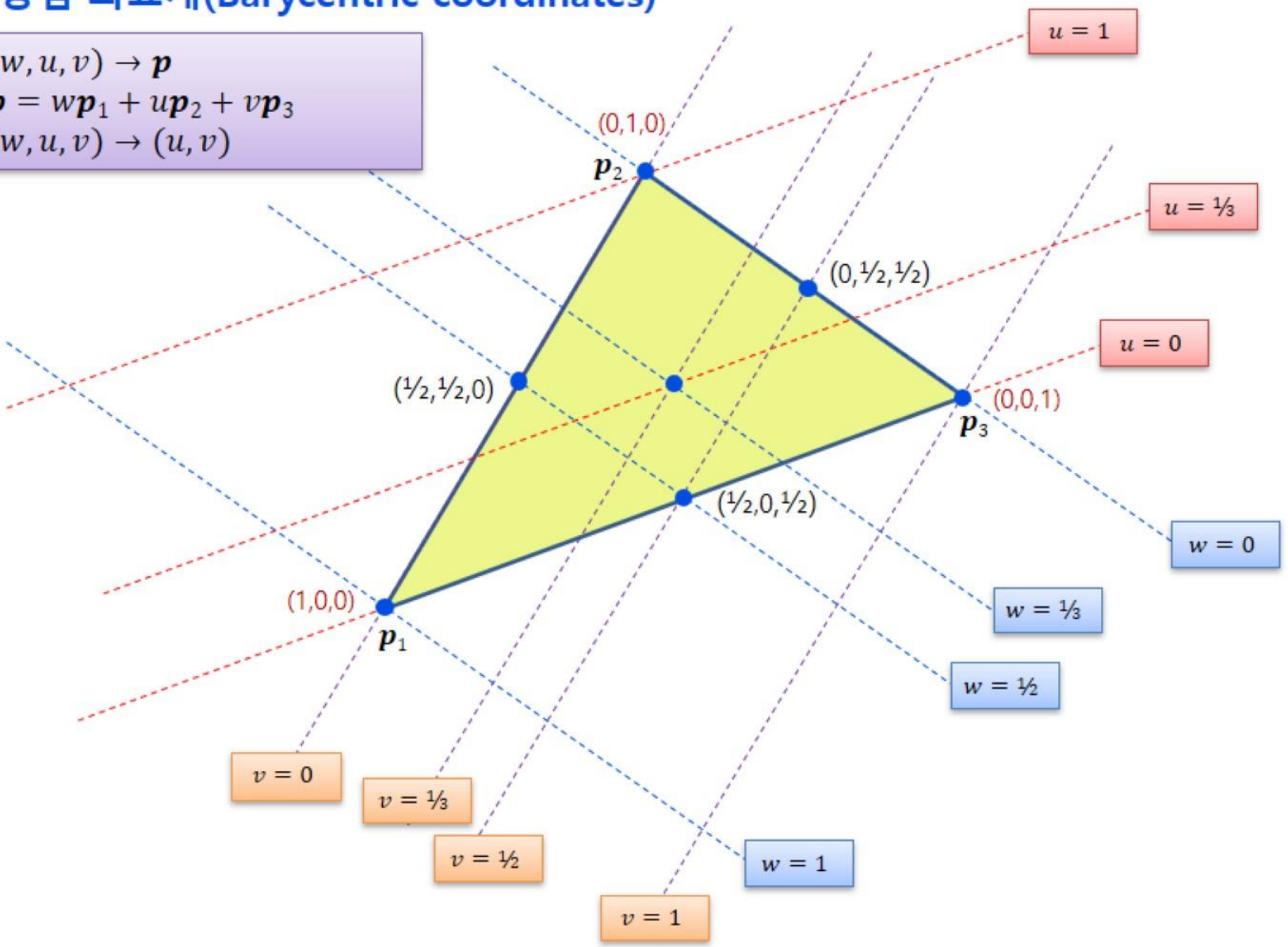
$$\boldsymbol{q} = \boldsymbol{p}_1 + u(\boldsymbol{p}_2 - \boldsymbol{p}_1) + v(\boldsymbol{p}_3 - \boldsymbol{p}_1)$$

$(u = 0)$인 경우 점 $\boldsymbol{q}$는 선분 $\boldsymbol{p}_1\boldsymbol{p}_3$ 위에 있다. 이때 $(v = ½)$인 경우 점 $\boldsymbol{q}$는 선분 $\boldsymbol{p}_1\boldsymbol{p}_3$의 중점이다.
$(v = 0)$인 경우 점 $\boldsymbol{q}$는 선분 $\boldsymbol{p}_1\boldsymbol{p}_2$ 위에 있다. 이때 $(u = ½)$인 경우 점 $\boldsymbol{q}$는 선분 $\boldsymbol{p}_1\boldsymbol{p}_2$의 중점이다.
$(u = ½, v = ½)$인 경우 점 $\boldsymbol{q}$는 선분 $\boldsymbol{p}_2\boldsymbol{p}_3$ 의 중점이다.
$(u = ⅓, v = ⅓, w = ⅓)$인 경우 점 $\boldsymbol{q}$는 삼각형의 무게중심이 된다.

텍스쳐(Texture)

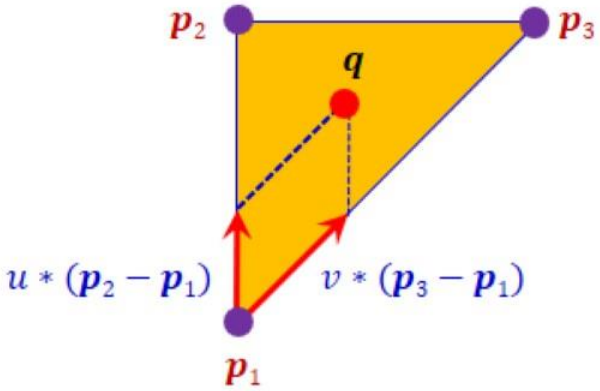
- 중심 좌표계(Barycentric coordinates)

$(w, u, v) \rightarrow \boldsymbol{p}$
$\boldsymbol{p} = w\boldsymbol{p}_1 + u\boldsymbol{p}_2 + v\boldsymbol{p}_3$
$(w, u, v) \rightarrow (u, v)$



텍스쳐(Texture)

- **중심 좌표계(Barycentric coordinates)**



$$q = p_1 + u(p_2 - p_1) + v(p_3 - p_1)$$
$$= (1 - u - v)p_1 + up_2 + vp_3$$

```
class CTexturedVertex
{
    float x, y, z;
    float tu, tv;
};
```

```
class CTexturedVertex
{
    XMFLOAT3 m_xmf3Position;
    XMFLOAT2 m_xmf2Texture;
};
```

중심 좌표계는 삼각형의 꼭지점(정점)의 좌표를 사용하여 삼각형 내부의 점을 표현한다.
삼각형의 각 꼭지점(정점)은 위치 벡터 뿐 아니라 다른 정보들을 포함할 수 있다.
이러한 다른 정보들은 정점의 색상, 텍스쳐 좌표, 법선 벡터, 접선 벡터 등이 될 수 있다.
중심 좌표 (w, u, v)로 정의되는 내부의 점에 대한 이러한 정보들은 세 개의 정점의 정보들로 표현할 수 있다.
삼각형 내부의 한 점에 대한 색상, 텍스쳐 좌표, 법선 벡터 등의 정보를 (w, u, v)로 보간할 수 있다.

위치 벡터의 보간:

$$q.x = w * p_1.x + u * p_2.x + v * p_3.x$$
$$q.y = w * p_1.y + u * p_2.y + v * p_3.y$$
$$q.z = w * p_1.z + u * p_2.z + v * p_3.z$$

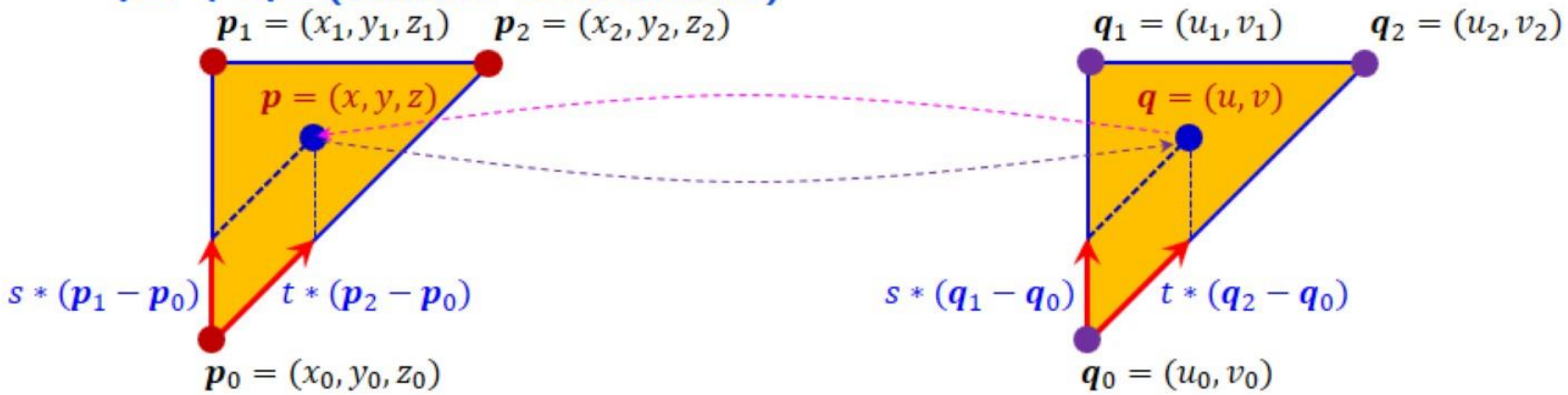
텍스쳐 좌표의 보간:

$$q.tu = w * p_1.tu + u * p_2.tu + v * p_3.tu$$
$$q.tv = w * p_1.tv + u * p_2.tv + v * p_3.tv$$

색상의 보간과 법선 벡터의 보간 등도 같은 방식으로 할 수 있다.

텍스쳐(Texture)

- 텍스쳐 좌표(Texture Coordinates)



$$p(s,t) = p_0 + t * (p_2 - p_0) + s * (p_1 - p_0)$$
$$s \geq 0, t \geq 0, s + t \leq 1.0$$

$$q = (u,v) = q_0 + t * (q_2 - q_0) + s * (q_1 - q_0)$$
$$s \geq 0, t \geq 0, s + t \leq 1.0$$

$$(u,v) = (u_0,v_0) + t * ((u_2,v_2) - (u_0,v_0)) + s * ((u_1,v_1) - (u_0,v_0))$$
$$(u,v) - (u_0,v_0) = t * (u_2 - u_0, v_2 - v_0) + s * (u_1 - u_0, v_1 - v_0)$$
$$(u - u_0, v - v_0) = s * (u_1 - u_0, v_1 - v_0) + t * (u_2 - u_0, v_2 - v_0)$$
$$(u - u_0, v - v_0) = (s * (u_1 - u_0) + t * (u_2 - u_0), s * (v_1 - v_0) + t * (v_2 - v_0))$$
$$(u - u_0, v - v_0) = (s,t) \begin{bmatrix} u_1 - u_0 & v_1 - v_0 \\ u_2 - u_0 & v_2 - v_0 \end{bmatrix}$$
$$(s,t) = (u - u_0, v - v_0) \begin{bmatrix} u_1 - u_0 & v_1 - v_0 \\ u_2 - u_0 & v_2 - v_0 \end{bmatrix}^{-1}$$
$$(s,t) = \frac{1}{(u_1 - u_0)(v_2 - v_0) - (u_2 - u_0)(v_1 - v_0)} (u - u_0, v - v_0) \begin{bmatrix} v_2 - v_0 & -(v_1 - v_0) \\ -(u_2 - u_0) & u_1 - u_0 \end{bmatrix}$$
$$s = f(u,v)$$
$$t = g(u,v)$$

$$p(s,t) = p_0 + t * (p_2 - p_0) + s * (p_1 - p_0)$$
$$= p_0 + g(u,v) * (p_2 - p_0) + f(u,v) * (p_1 - p_0) = p(u,v)$$
$$s \geq 0, t \geq 0, s + t \leq 1.0$$

텍스쳐(Texture)

- **텍스쳐 좌표(Texture Coordinates)**

$$(s,t) = (f(u,v), g(u,v))$$

$$= \frac{1}{(u_1-u_0)(v_2-v_0)-(u_2-u_0)(v_1-v_0)}(u-u_0, v-v_0)\begin{bmatrix} v_2-v_0 & -(v_1-v_0) \\ -(u_2-u_0) & u_1-u_0 \end{bmatrix}$$

$$= \frac{(u-u_0, v-v_0)\begin{bmatrix} v_2-v_0 & v_0-v_1 \\ u_0-u_2 & u_1-u_0 \end{bmatrix}}{(u_1-u_0)(v_2-v_0)-(u_2-u_0)(v_1-v_0)}$$

$$= \frac{\big((u-u_0)(v_2-v_0)+(v-v_0)(u_0-u_2), (u-u_0)(v_0-v_1)+(v-v_0)(u_1-u_0)\big)}{(u_1-u_0)(v_2-v_0)-(u_2-u_0)(v_1-v_0)}$$

$$f(u,v) = \frac{(u-u_0)(v_2-v_0)+(v-v_0)(u_0-u_2)}{(u_1-u_0)(v_2-v_0)-(u_2-u_0)(v_1-v_0)}$$

$$g(u,v) = \frac{(u-u_0)(v_0-v_1)+(v-v_0)(u_1-u_0)}{(u_1-u_0)(v_2-v_0)-(u_2-u_0)(v_1-v_0)}$$

$$\frac{\partial f(u,v)}{\partial u} = \frac{(v_2-v_0)}{(u_1-u_0)(v_2-v_0)-(u_2-u_0)(v_1-v_0)}$$

$$\frac{\partial g(u,v)}{\partial u} = \frac{(v_0-v_1)}{(u_1-u_0)(v_2-v_0)-(u_2-u_0)(v_1-v_0)}$$

$$\frac{\partial f(u,v)}{\partial v} = \frac{(u_0-u_2)}{(u_1-u_0)(v_2-v_0)-(u_2-u_0)(v_1-v_0)}$$

$$\frac{\partial g(u,v)}{\partial v} = \frac{(u_1-u_0)}{(u_1-u_0)(v_2-v_0)-(u_2-u_0)(v_1-v_0)}$$

$$\frac{\partial \boldsymbol{p}}{\partial u} = \frac{\partial \boldsymbol{p}}{\partial f}\frac{\partial f}{\partial u} + \frac{\partial \boldsymbol{p}}{\partial g}\frac{\partial g}{\partial u} = (\boldsymbol{p}_1-\boldsymbol{p}_0)\frac{\partial f}{\partial u} + (\boldsymbol{p}_2-\boldsymbol{p}_0)\frac{\partial g}{\partial u}$$

$$= \frac{(\boldsymbol{p}_1-\boldsymbol{p}_0)(v_2-v_0)+(\boldsymbol{p}_2-\boldsymbol{p}_0)(v_0-v_1)}{(u_1-u_0)(v_2-v_0)-(u_2-u_0)(v_1-v_0)}$$

$$= \frac{(\boldsymbol{p}_1-\boldsymbol{p}_0)(v_2-v_0)-(\boldsymbol{p}_2-\boldsymbol{p}_0)(v_1-v_0)}{(u_1-u_0)(v_2-v_0)-(u_2-u_0)(v_1-v_0)}$$

$$\frac{\partial \boldsymbol{p}}{\partial v} = \frac{\partial \boldsymbol{p}}{\partial f}\frac{\partial f}{\partial v} + \frac{\partial \boldsymbol{p}}{\partial g}\frac{\partial g}{\partial v} = (\boldsymbol{p}_1-\boldsymbol{p}_0)\frac{\partial f}{\partial v} + (\boldsymbol{p}_2-\boldsymbol{p}_0)\frac{\partial g}{\partial v}$$

$$= \frac{(\boldsymbol{p}_1-\boldsymbol{p}_0)(u_0-u_2)+(\boldsymbol{p}_2-\boldsymbol{p}_0)(u_1-u_0)}{(u_1-u_0)(v_2-v_0)-(u_2-u_0)(v_1-v_0)}$$

$$= \frac{(\boldsymbol{p}_2-\boldsymbol{p}_0)(u_1-u_0)-(\boldsymbol{p}_1-\boldsymbol{p}_0)(u_2-u_0)}{(u_1-u_0)(v_2-v_0)-(u_2-u_0)(v_1-v_0)}$$

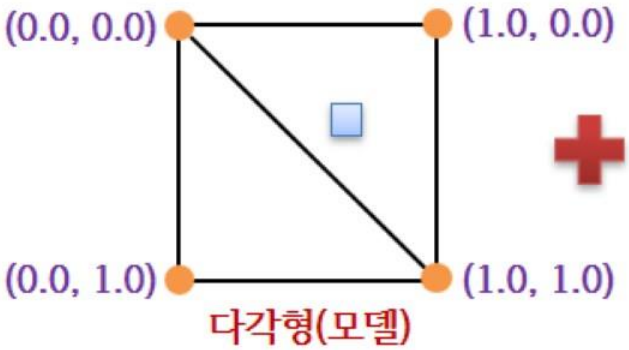
$$\boldsymbol{p}(s,t) = \boldsymbol{p}_0 + t*(\boldsymbol{p}_2-\boldsymbol{p}_0) + s*(\boldsymbol{p}_1-\boldsymbol{p}_0)$$

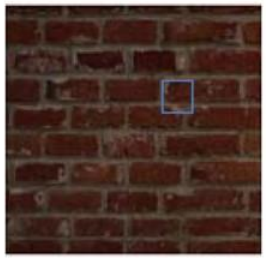
$$= \boldsymbol{p}_0 + g(u,v)*(\boldsymbol{p}_2-\boldsymbol{p}_0) + f(u,v)*(\boldsymbol{p}_1-\boldsymbol{p}_0) = \boldsymbol{p}(u,v)$$

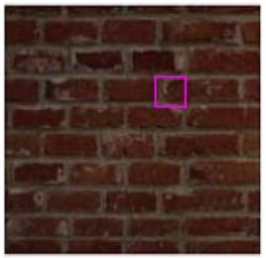
$$s \geq 0, t \geq 0, s+t \leq 1.0$$

텍스쳐(Texture)

- **텍스쳐 좌표(Texture Coordinates)**




텍스쳐 이미지


다각형(모델)

보간된 텍스쳐 좌표 (u, v) ➡ 픽셀 좌표 (U, V) = (u*Width, v*Height), 어떤 텍셀(색상)?

렌더링되는 사각형의 크기는 카메라까지의 거리에 따라 변함
- 사각형의 크기(픽셀) = 텍스쳐의 크기(픽셀)
- 사각형의 크기(픽셀) ≤ 텍스쳐의 크기(픽셀)
- 사각형의 크기(픽셀) ≥ 텍스쳐의 크기(픽셀)

밉맵(Mip Map) 레벨 ➪ 텍스쳐 이미지 결정

확대(Magnification)
축소(Minification)


1 : 1

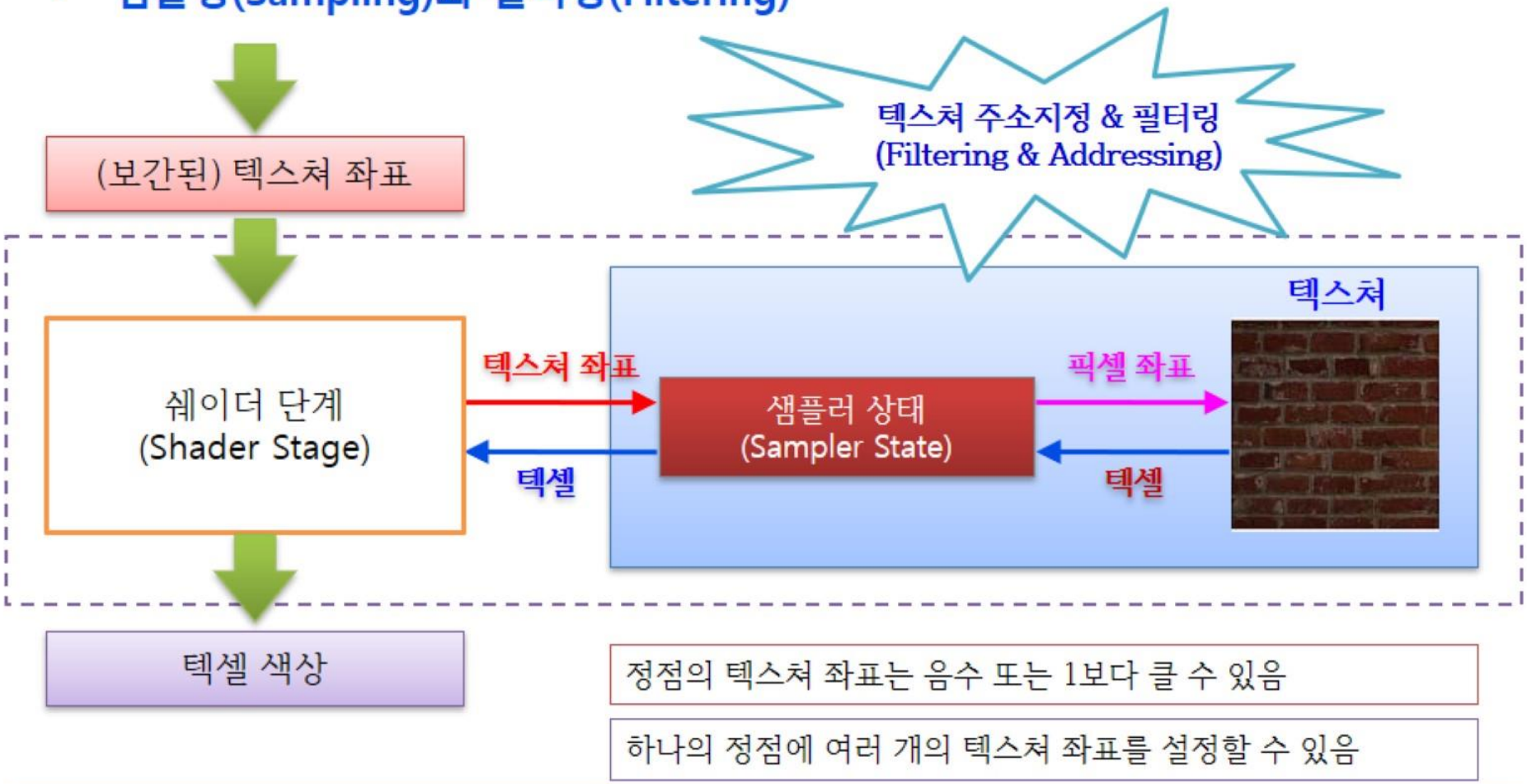

축소


확대

보간(Interpolation)
1.밉맵 레벨 결정
2.텍셀 결정

텍스쳐(Texture)

- 샘플링(Sampling)과 필터링(Filtering)



정점의 텍스쳐 좌표는 음수 또는 1보다 클 수 있음

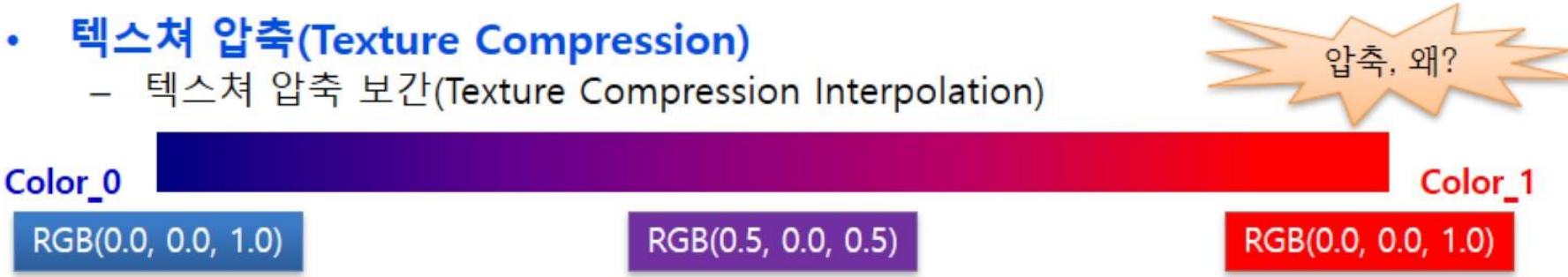
하나의 정점에 여러 개의 텍스쳐 좌표를 설정할 수 있음

프리미티브를 구성하는 각 픽셀에 대한 다음의 과정을 **샘플링(Sampling)**과 **필터링(Filtering)**이라고 한다.

① 각 텍스쳐 좌표에 대한 보간된 텍스쳐 좌표를 계산
② 텍스쳐 이미지에 대한 픽셀(텍셀) 좌표를 계산
③ 텍스쳐 이미지의 픽셀(텍셀)을 반환 (어떤 텍셀을 선택하고 어떻게 색상을 계산할 것인가: 필터링)

텍스쳐(Texture)

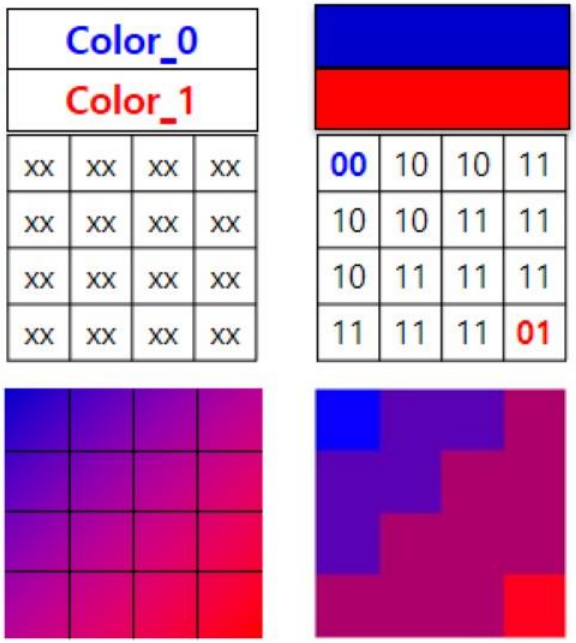
- **텍스쳐 압축(Texture Compression)**
  - 텍스쳐 압축 보간(Texture Compression Interpolation)




```
nColorShift = (Color_1 – Color_0) / (nGradientWidth – 1);
for (i = 0; i < nGradientWidth + 1; i++) Color[i] = Color_0 + (nColorShift * i);
```

  - Color_0, Color_1, i (i=0, …, nGradientWidth-1)

- **블럭 압축(Block Compression)**
  - 텍스쳐를 4x4 텍셀 블럭으로 분할
  - 각 텍셀은 2-비트로 표현(저장): 0~3 인덱스
  - 각 블럭은 2개의 색상을 가짐(4개가 아님)



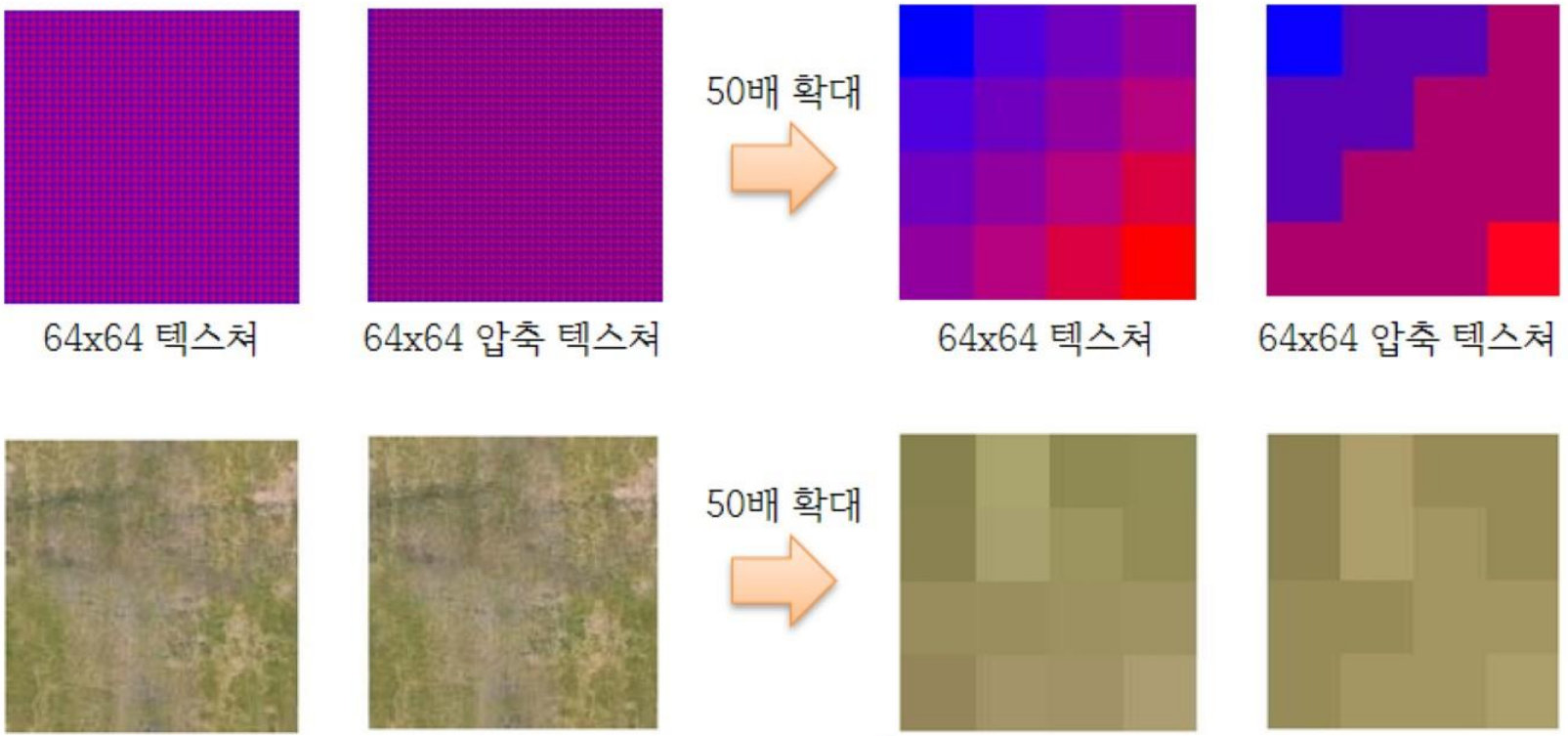
00: Color_0
01: Color_1
10: (Color_0 * ⅔) + (Color_1 * ⅓)
11: (Color_0 * ⅓) + (Color_1 * ⅔)

선형보간으로 계산

  - 텍셀(실수 4개 = 16 Byte)
    16개 텍셀 = 16*16 Byte = 256 Byte
  - 1개 블럭: 4 Byte + 16*2 Byte = 36 Byte

텍스쳐(Texture)

- 블럭 압축(Block Compression)



DXGI_FORMAT_BC1_UNORM

가로x세로	비트/픽셀	바이트 수
128x128	32	65,536
4096x4096	32	**67,108,864**

가로x세로	블럭 수	바이트 수
128x128	1,024 (32x32)	8,192
4096x4096	1,048,576 (1024x1024)	**8,388,608**

텍스쳐(Texture)

- **압축(Compressed) 텍스쳐 형식**

```
typedef enum DXGI_FORMAT {
    DXGI_FORMAT_BC1_TYPELESS,
    DXGI_FORMAT_BC1_UNORM,
    DXGI_FORMAT_BC1_UNORM_SRGB,
    DXGI_FORMAT_BC2_TYPELESS,
    DXGI_FORMAT_BC2_UNORM,
    DXGI_FORMAT_BC2_UNORM_SRGB,
    DXGI_FORMAT_BC3_TYPELESS,
    DXGI_FORMAT_BC3_UNORM,
    DXGI_FORMAT_BC3_UNORM_SRGB,
    DXGI_FORMAT_BC4_TYPELESS,
    DXGI_FORMAT_BC4_UNORM,
    DXGI_FORMAT_BC4_SNORM,
    DXGI_FORMAT_BC5_TYPELESS,
    DXGI_FORMAT_BC5_UNORM,
    DXGI_FORMAT_BC5_SNORM,
    DXGI_FORMAT_BC6H_TYPELESS,
    DXGI_FORMAT_BC6H_UF16,
    DXGI_FORMAT_BC6H_SF16,
    DXGI_FORMAT_BC7_TYPELESS,
    DXGI_FORMAT_BC7_UNORM,
    DXGI_FORMAT_BC7_UNORM_SRGB,
} DXGI_FORMAT
```

DXGI_FORMAT_BC1_	RGB(5:6:5), A(1-비트) 또는 알파 없음
DXGI_FORMAT_BC2_	RGB(5:6:5), A(4-비트)
DXGI_FORMAT_BC3_	RGB(5:6:5), A(8-비트)
DXGI_FORMAT_BC4_	1 색상(8) 채널(Grayscale 이미지)
DXGI_FORMAT_BC5_	2 색상(8:8) 채널
DXGI_FORMAT_BC6_	HDR(High Dynamic Range) 이미지
DXGI_FORMAT_BC7_	고화질 RGBA 압축(법선 맵 압축에 사용)

DXGI_FORMAT_BC1_UNORM

Color_0(5:6:5)			
Color_1(5:6:5)			
xx	xx	xx	xx
xx	xx	xx	xx
xx	xx	xx	xx
xx	xx	xx	xx

알파 없는 텍스쳐

00: Color_0
01: Color_1
10: (Color_0 * ⅔) + (Color_1 * ⅓)
11: (Color_0 * ⅓) + (Color_1 * ⅔)

00: Color_0
01: Color_1
10: (Color_0 * ½) + (Color_1 * ½)
11: 0 (투명)

밉맵 텍스쳐의 크기는 4의 배수로 만들어 짐(메모리 패딩(Padding)이 있을 수 있음)

압축 텍스쳐의 크기(가로, 세로)는 4의 배수, 쉐이더 단계의 입력으로만 사용, GPU가 자동적으로 압축 해제

텍스쳐(Texture)

- **압축(Compressed) 텍스쳐 형식**

```
typedef enum DXGI_FORMAT {
    DXGI_FORMAT_BC1_TYPELESS,
    DXGI_FORMAT_BC1_UNORM,
    DXGI_FORMAT_BC1_UNORM_SRGB,
    DXGI_FORMAT_BC2_TYPELESS,
    DXGI_FORMAT_BC2_UNORM,
    DXGI_FORMAT_BC2_UNORM_SRGB,
    DXGI_FORMAT_BC3_TYPELESS,
```

DXGI_FORMAT_BC3_UNORM

```
if( alpha_0 > alpha_1 ) {
    // 6 interpolated alpha values.
    alpha_2 = 6/7*alpha_0 + 1/7*alpha_1; //010
    alpha_3 = 5/7*alpha_0 + 2/7*alpha_1; //011
    alpha_4 = 4/7*alpha_0 + 3/7*alpha_1; //100
    alpha_5 = 3/7*alpha_0 + 4/7*alpha_1; //101
    alpha_6 = 2/7*alpha_0 + 5/7*alpha_1; //110
    alpha_7 = 1/7*alpha_0 + 6/7*alpha_1; //111
} else {
    // 4 interpolated alpha values.
    alpha_2 = 4/5*alpha_0 + 1/5*alpha_1; //010
    alpha_3 = 3/5*alpha_0 + 2/5*alpha_1; //011
    alpha_4 = 2/5*alpha_0 + 3/5*alpha_1; //100
    alpha_5 = 1/5*alpha_0 + 4/5*alpha_1; //101
    alpha_6 = 0; //110
    alpha_7 = 255; //111
}
```

DXGI_FORMAT_BC1_	RGB(5:6:5), A(1-비트)
DXGI_FORMAT_BC2_	RGB(5:6:5), A(4-비트)
DXGI_FORMAT_BC3_	RGB(5:6:5), A(8-비트)
DXGI_FORMAT_BC4_	1 색상(8) 채널(Grayscale 이미지)
DXGI_FORMAT_BC5_	2 색상(8:8) 채널
DXGI_FORMAT_BC6_	HDR(High Dynamic Range) 이미지
DXGI_FORMAT_BC7_	고화질 RGBA 압축

DXGI_FORMAT_BC2_UNORM

DXGI_FORMAT_BC3_UNORM

압축 텍스쳐의 크기는 4의 배수, 쉐이더 단계의 입력으로만 사용, GPU가 압축 해제

텍스쳐(Texture)

- **DDS 텍스쳐 파일 생성**
  - DDS(Direct Draw Surface) 형식 파일 생성
    밉맵(MipMap)과 텍스쳐 압축 기능을 제공하는 형식
  - DirectXTex-master
    https://github.com/Microsoft/DirectXTex

Visual Studio 2015 Image Editor
Visual Studio 2017 Image Editor

① (~₩DirectXTex-master₩DirectXTex_Desktop_2017.sln)을 열고 솔루션 빌드
② (~₩DirectXTex-master₩TexConv₩Bin₩...₩TexConv.exe)파일을 C:₩Windows에 복사
③ 명령 프롬프트 실행(⊞+x)
④ texconv [options] "파일 이름(경로)"↵

```
d:↵
cd Working₩LabProjects₩Assets↵
texconv brick01.jpg↵
texconv brick02.jpg brick03.jpg↵
```

brick01.dds brick02.dds brick03.dds

texconv –m <mip-levels> -f <**format**> -c <color-key> files↵

texconv –f BC1_UNORM brick01.jpg↵

C:₩Program Files (x86)₩Microsoft DirectX SDK (June 2010)₩Utilities₩bin₩x64₩DxTex.exe
C:₩Program Files (x86)₩Microsoft DirectX SDK (June 2010)₩Utilities₩bin₩x64₩TexConv.exe

  - 텍스쳐 파일 읽기
    - DDS 텍스쳐 파일(.dds)

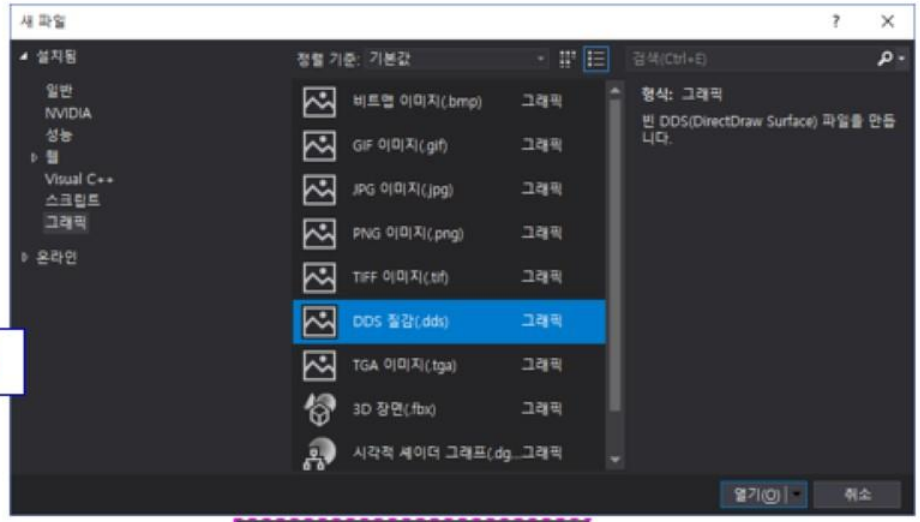
~₩DirectXTex-master₩DDSTextureLoader₩DDSTextureLoader12.h
~₩DirectXTex-master₩DDSTextureLoader₩DDSTextureLoader12.cpp

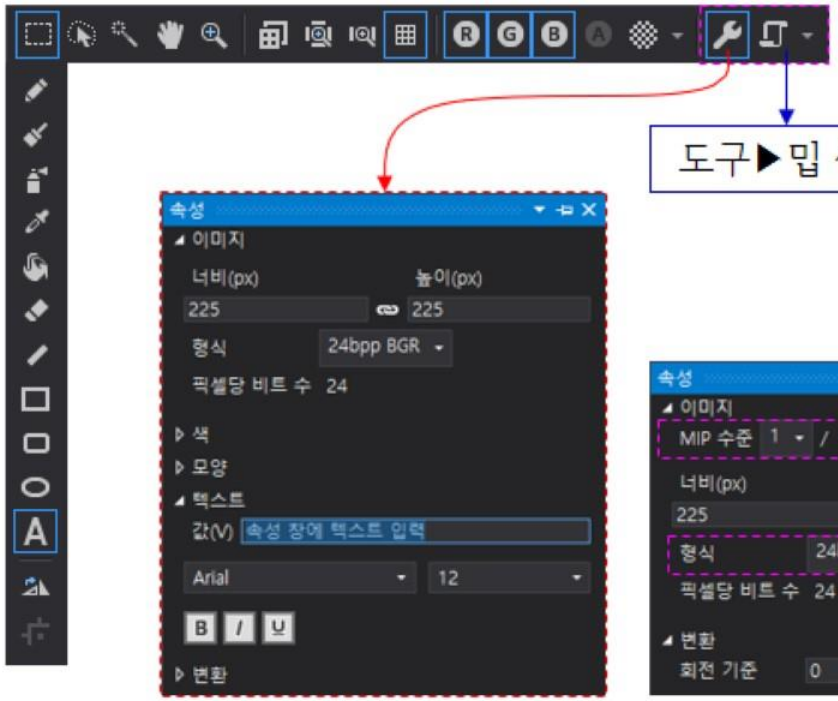
    - WIC 텍스쳐 파일(.bmp, .jpg, .png, ...)

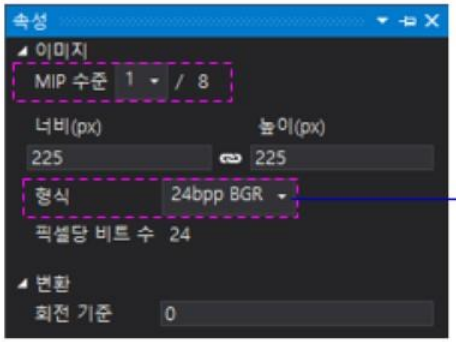
~₩DirectXTex-master₩WICTextureLoader₩WICTextureLoader12.h
~₩DirectXTex-master₩WICTextureLoader₩WICTextureLoader12.cpp

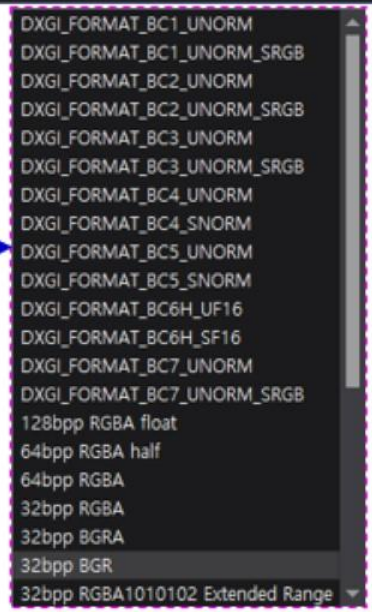
텍스쳐(Texture)

- **DDS 텍스쳐 파일 생성**
  - DDS(Direct Draw Surface) 형식 파일 생성
    Visual Studio 2017 Image Editor
    - 파일▶새로 만들기▶그래픽▶파일…
    - 파일▶열기▶그래픽▶파일…









  - 파일▶다른 이름으로 [파일 이름] 저장…
    파일 형식(T): DirectDraw 표면(*.dds)

텍스쳐(Texture)

- **텍스쳐 리소스 생성**
  - 프로젝트에 "DDSTextureLoader12.h"와 "DDSTextureLoader12.cpp" 추가
  - DDS(Direct Draw Surface) 텍스쳐 파일 읽기

DirectX-Graphics-Samples-master₩Libraries₩D3DX12

#include "d3dx12.h"

```
HRESULT LoadDDSTextureFromFile(
    ID3D12Device *pd3dDevice, //디바이스 인터페이스 포인터
    wchar_t *pszFileName, //파일 이름(경로)
    ID3D12Resource **pd3dTexture, //텍스쳐 리소스(픽셀 데이터를 채우지 않음)
    std::unique_ptr<uint8_t[]>& ddsData, //텍스쳐 데이터(픽셀)
    std::vector<D3D12_SUBRESOURCE_DATA>& subresources, //서브리소스 데이터
    size_t maxsize = 0,
    DDS_ALPHA_MODE *pAlphaMode = nullptr, //알파 모드
    bool *pbIsCubeMap = nullptr //텍스쳐가 큐브 텍스쳐인가
);
```

```
HRESULT LoadDDSTextureFromFileEx(
    ID3D12Device *pd3dDevice, //디바이스 인터페이스 포인터
    wchar_t *pszFileName, //파일 이름(경로)
    size_t maxsize,
    D3D12_RESOURCE_FLAGS resFlags,
    unsigned int loadFlags
    ID3D12Resource **pd3dTexture, //텍스쳐 리소스
    std::unique_ptr<uint8_t[]>& ddsData, //텍스쳐 데이터(픽셀)
    std::vector<D3D12_SUBRESOURCE_DATA>& subresources, //서브리소스 데이터
    DDS_ALPHA_MODE *pAlphaMode = nullptr, //알파 모드
    bool *pbIsCubeMap = nullptr //텍스쳐가 큐브 텍스쳐인가
);
```

텍스쳐(Texture)

- **텍스쳐 리소스 생성**
  - DDS(Direct Draw Surface) 텍스쳐 리소스 생성

```
ID3D12Resource *CreateTextureResourceFromFile(ID3D12Device *pd3dDevice,
ID3D12GraphicsCommandList *pd3dCommandList, wchar_t *pszFileName, D3D12_RESOURCE_STATES
d3dResourceStates, ID3D12Resource **ppd3dUploadBuffer)
{
   ID3D12Resource *pd3dTexture = NULL;
   std::unique_ptr<uint8_t[]> ddsData;
   std::vector<D3D12_SUBRESOURCE_DATA> vSubresources;
   DDS_ALPHA_MODE ddsAlphaMode = DDS_ALPHA_MODE_UNKNOWN;
   bool bIsCubeMap = false;

   HRESULT hResult = DirectX::LoadDDSTextureFromFileEx(pd3dDevice, pszFileName, 0,
D3D12_RESOURCE_FLAG_NONE, DDS_LOADER_DEFAULT, &pd3dTexture, ddsData, vSubresources,
&ddsAlphaMode, &bIsCubeMap);

   D3D12_HEAP_PROPERTIES d3dHeapPropertiesDesc;
   ::ZeroMemory(&d3dHeapPropertiesDesc, sizeof(D3D12_HEAP_PROPERTIES));
   d3dHeapPropertiesDesc.Type = D3D12_HEAP_TYPE_UPLOAD;
   d3dHeapPropertiesDesc.CPUPageProperty = D3D12_CPU_PAGE_PROPERTY_UNKNOWN;
   d3dHeapPropertiesDesc.MemoryPoolPreference = D3D12_MEMORY_POOL_UNKNOWN;
   d3dHeapPropertiesDesc.CreationNodeMask = 1;
   d3dHeapPropertiesDesc.VisibleNodeMask = 1;

   UINT nSubResources = (UINT)vSubresources.size();
   UINT64 nBytes = GetRequiredIntermediateSize(pd3dTexture, 0, nSubResources);
```

텍스쳐(Texture)

- **텍스쳐 리소스 생성**

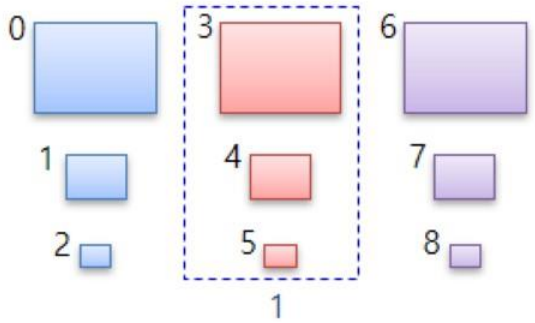
```
D3D12_RESOURCE_DESC d3dResourceDesc;
::ZeroMemory(&d3dResourceDesc, sizeof(D3D12_RESOURCE_DESC));
d3dResourceDesc.Dimension = D3D12_RESOURCE_DIMENSION_BUFFER;
d3dResourceDesc.Width = nBytes;
d3dResourceDesc.Height = 1;
d3dResourceDesc.DepthOrArraySize = d3dUploadResourceDesc.MipLevels = 1;
d3dResourceDesc.Format = DXGI_FORMAT_UNKNOWN;
d3dResourceDesc.SampleDesc.Count = 1;
d3dResourceDesc.Layout = D3D12_TEXTURE_LAYOUT_ROW_MAJOR;
d3dResourceDesc.Flags = D3D12_RESOURCE_FLAG_NONE;
pd3dDevice->CreateCommittedResource(&d3dHeapPropertiesDesc, D3D12_HEAP_FLAG_NONE,
&d3dResourceDesc, D3D12_RESOURCE_STATE_GENERIC_READ, NULL, IID_PPV_ARGS(&ppd3dUploadBuffer));
::UpdateSubresources(pd3dCommandList, pd3dTexture, *ppd3dUploadBuffer, 0, 0, nSubResources,
&vSubresources[0]);

D3D12_RESOURCE_BARRIER d3dResourceBarrier;
d3dResourceBarrier.Type = D3D12_RESOURCE_BARRIER_TYPE_TRANSITION;
d3dResourceBarrier.Flags = D3D12_RESOURCE_BARRIER_FLAG_NONE;
d3dResourceBarrier.Transition.pResource = pd3dTexture;
d3dResourceBarrier.Transition.StateBefore = D3D12_RESOURCE_STATE_COPY_DEST;
d3dResourceBarrier.Transition.StateAfter = d3dResourceStates;
d3dResourceBarrier.Transition.Subresource = D3D12_RESOURCE_BARRIER_ALL_SUBRESOURCES;
pd3dCommandList->ResourceBarrier(1, &d3dResourceBarrier);

return(pd3dTexture);
}
```

업로드 힙에는 텍스쳐 리소스를 생성할 수 없음

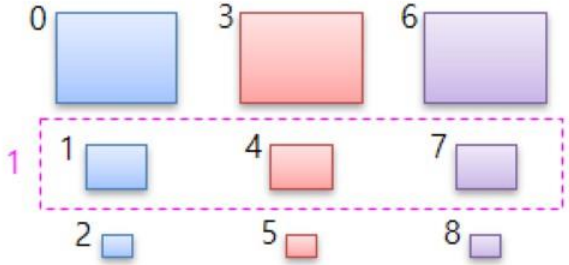
픽셀 데이터를 업로드 힙으로 복사하고 업로드 힙을 텍스쳐 리소스로 복사함

텍스쳐(Texture)

- **서브리소스(Subresource)**
  - 텍스쳐 리소스의 부분 집합(텍스쳐는 서브리소스들의 집합(배열)으로 취급)
  - 각 서브리소스는 인덱스(서브리소스 인덱스)로 접근할 수 있음
  - 버퍼(Buffer)는 하나의 서브리소스로 구성됨(서브리소스 인덱스가 필요없음)
  - 텍스쳐는 텍스쳐의 유형에 따라 서브리소스의 개수가 달라짐
    텍스쳐 배열의 크기와 밉맵 레벨에 따라 달라짐
  - 밉맵 텍스쳐는 하나의 서브리소스로 취급됨



배열 슬라이스(Array Slice)



밉 슬라이스(Mip Slice)

서브리소스 인덱스 = MipSlice + (ArraySlice * MipLevels)

```
UINT D3D12CalcSubresource(
   UINT MipSlice, //밉맵 레벨 인덱스, 0: 가장 세밀한 밉맵 레벨
   UINT ArraySlice, //배열 레벨 인덱스, 3D 텍스쳐에 0를 사용
   UINT PlaneSlice, //평면 레벨 인덱스(깊이-스텐실 버퍼, YCbCr), RGB 색상의 표현에는 사용하지 않음(0)
   UINT MipLevels, //밉맵 레벨의 개수
   UINT ArraySize //배열 원소의 개수
);
```

MipSlice + (ArraySlice * MipLevels) + (PlaneSlice * MipLevels * ArraySize)

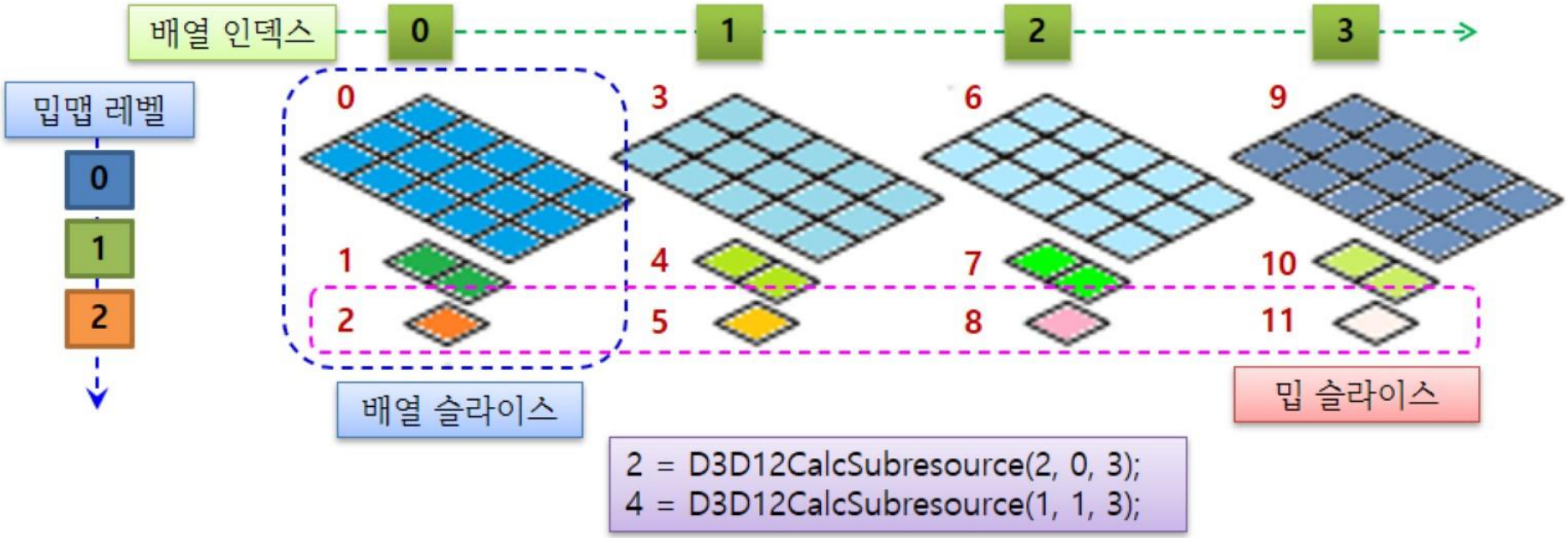
텍스쳐(Texture)

- **서브리소스(Subresource)**

```
UINT D3D12CalcSubresource(
    UINT MipSlice, //밉맵 레벨 인덱스, 0: 가장 세밀한 밉맵 레벨
    UINT ArraySlice, //배열 레벨 인덱스, 3D 텍스쳐에 0를 사용
    UINT PlaneSlice, //평면 레벨 인덱스(깊이-스텐실 버퍼, YCbCr), RGB 색상의 표현에는 사용하지 않음(0)
    UINT MipLevels, //밉맵 레벨의 개수
    UINT ArraySize //배열 원소의 개수
);
```

텍스쳐 배열의 서브 리소스 인덱스를 계산
인덱스 = MipSlice + (ArraySlice * MipLevels)

MipSlice	서브 텍스쳐의 배열 인덱스, 0(첫 번째, 가장 세밀한 서브 텍스쳐 또는 밉맵 레벨)
ArraySlice	텍스쳐의 배열에서 사용할 첫 번째 텍스쳐의 인덱스
MipLevels	사용할 밉맵 레벨(또는 서브 텍스쳐)의 개수



텍스쳐(Texture)

- 쉐이더 리소스 뷰(Shader Resource View)

```
void ID3D12Device::CreateShaderResourceView(
    ID3D12Resource *pResource,
    D3D12_SHADER_RESOURCE_VIEW_DESC *pDesc,
    D3D12_CPU_DESCRIPTOR_HANDLE DestDescriptor
);
```

```
typedef enum D3D12_SRV_DIMENSION {
    D3D12_SRV_DIMENSION_UNKNOWN,
    D3D12_SRV_DIMENSION_BUFFER,
    D3D12_SRV_DIMENSION_TEXTURE1D,
    D3D12_SRV_DIMENSION_TEXTURE1DARRAY,
    D3D12_SRV_DIMENSION_TEXTURE2D,
    D3D12_SRV_DIMENSION_TEXTURE2DARRAY,
    D3D12_SRV_DIMENSION_TEXTURE2DMS,
    D3D12_SRV_DIMENSION_TEXTURE2DMSARRAY,
    D3D12_SRV_DIMENSION_TEXTURE3D,
    D3D12_SRV_DIMENSION_TEXTURECUBE,
    D3D12_SRV_DIMENSION_TEXTURECUBEARRAY
} D3D12_SRV_DIMENSION;
```

```
typedef struct D3D12_SHADER_RESOURCE_VIEW_DESC {
    DXGI_FORMAT Format;
    D3D12_SRV_DIMENSION ViewDimension;
    UINT Shader4ComponentMapping;
    union {
        D3D12_BUFFER_SRV B
        D3D12_TEX1D_SRV Tex
        D3D12_TEX1D_ARRAY_
        D3D12_TEX2D_SRV Te
        D3D12_TEX2D_ARRAY_
        D3D12_TEX2DMS_SRV
        D3D12_TEX2DMS_ARR
        D3D12_TEX3D_SRV Tex
        D3D12_TEXCUBE_SRV TextureCube;
        D3D12_TEXCUBE_ARRAY_SRV TextureCu
    };
} D3D12_SHADER_RESOURCE_VIEW_DESC;
```

```
typedef enum D3D12_SHADE
    D3D12_SHADER_COMPON
    D3D12_SHADER_COMPON
    D3D12_SHADER_COMPON
    D3D12_SHADER_COMPONENT_MAPPING_FROM_MEMORY_COMPONENT_3,
    D3D12_SHADER_COMPONENT_MAPPING_FORCE_VALUE_0,
    D3D12_SHADER_COMPONENT_MAPPING_FORCE_VALUE_1
} D3D12_SHADER_COMPONENT_MAPPING;
```

```
typedef struct D3D12_BUFFER_SRV {
    UINT64 FirstElement;
    UINT NumElements;
    UINT StructureByteStride;
    D3D12_BUFFER_SRV_FLA
} D3D12_BUFFER_SRV;
```

```
typedef struct D3D12_TEX2D_SRV {
    UINT MostDetailedMip;
    UINT MipLevels;
    UINT PlaneSlice;
    FLOAT ResourceMinLODClamp;
} D3D12_TEX2D_SRV;
```

텍스쳐(Texture)

- **쉐이더 리소스 뷰(Shader Resource View)**

```
typedef struct D3D12_SHADER_RESOURCE_VIEW_DESC {
    DXGI_FORMAT Format; //뷰가 리소스를 보는 형식
    D3D12_SRV_DIMENSION ViewDimension;
    UINT Shader4ComponentMapping;
    union {
        D3D12_BUFFER_SRV Buffer;
        D3D12_TEX1D_SRV Texture1D;
        D3D12_TEX1D_ARRAY_SRV Texture1DArray;
        D3D12_TEX2D_SRV Texture2D;
        D3D12_TEX2D_ARRAY_SRV Texture2DArray;
        D3D12_TEX2DMS_SRV Texture2DMS;
        D3D12_TEX2DMS_ARRAY_SRV Texture2DMSArray;
        D3D12_TEX3D_SRV Texture3D;
        D3D12_TEXCUBE_SRV TextureCube;
        D3D12_TEXCUBE_ARRAY_SRV TextureCubeArray;
    };
} D3D12_SHADER_RESOURCE_VIEW_DESC;
```

```
typedef enum D3D12_SRV_DIMENSION {
    D3D12_SRV_DIMENSION_UNKNOWN,
    D3D12_SRV_DIMENSION_BUFFER,
    D3D12_SRV_DIMENSION_TEXTURE1D,
    D3D12_SRV_DIMENSION_TEXTURE1DARRAY,
    D3D12_SRV_DIMENSION_TEXTURE2D,
    D3D12_SRV_DIMENSION_TEXTURE2DARRAY,
    D3D12_SRV_DIMENSION_TEXTURE2DMS,
    D3D12_SRV_DIMENSION_TEXTURE2DMSARRAY,
    D3D12_SRV_DIMENSION_TEXTURE3D,
    D3D12_SRV_DIMENSION_TEXTURECUBE,
    D3D12_SRV_DIMENSION_TEXTURECUBEARRAY
} D3D12_SRV_DIMENSION;
//뷰의 리소스 유형
```

```
typedef enum D3D12_BUFFER_SRV_FLAGS {
    D3D12_BUFFER_SRV_FLAG_NONE,
    D3D12_BUFFER_SRV_FLAG_RAW
} D3D12_BUFFER_SRV_FLAGS;
```

```
typedef struct D3D12_TEX2D_SRV {
    UINT MostDetailedMip; //0~MipLevels
    UINT MipLevels; //밉맵 레벨 수, -1
    UINT PlaneSlice; //평면 슬라이스 번호(인덱스)
    FLOAT ResourceMinLODClamp; //최소 밉맵 레벨
} D3D12_TEX2D_SRV;
```

```
typedef struct D3D12_BUFFER_SRV {
    UINT64 FirstElement; //첫번째 원소(인덱스)
    UINT NumElements; //원소의 개수
    UINT StructureByteStride; //구조체 원소의 크기(바이트)
    D3D12_BUFFER_SRV_FLAGS Flags;
} D3D12_BUFFER_SRV;
```

텍스쳐(Texture)

- 쉐이더 리소스 뷰(Shader Resource View)

```
typedef struct D3D12_SHADER_RESOURCE_VIEW_DESC {
    DXGI_FORMAT Format;
    D3D12_SRV_DIMENSION ViewDimension;
    UINT Shader4ComponentMapping; //D3D12_DEFAULT_SHADER_4_COMPONENT_MAPPING
    union {
        D3D12_BUFFER_SRV Buffer;
        D3D12_TEX1D_SRV Texture1D;
        D3D12_TEX1D_ARRAY_SRV Texture1DArray;
        D3D12_TEX2D_SRV Texture2D;
        D3D12_TEX2D_ARRAY_SRV Texture2DArray;
        D3D12_TEX2DMS_SRV Texture2DMS;
        D3D12_TEX2DMS_ARRAY_SRV Texture2DMSArray;
        D3D12_TEX3D_SRV Texture3D;
        D3D12_TEXCUBE_SRV TextureCube;
        D3D12_TEXCUBE_ARRAY_SRV TextureCubeArray;
    };
} D3D12_SHADER_RESOURCE_VIEW_DESC;
```

D3D12_ENCODE_SHADER_4_COMPONENT_MAPPING(Src0,Src1,Src2,Src3)

```
#define D3D12_DEFAULT_SHADER_4_COMPONENT_MAPPING D3D12_ENCODE_SHADER_4_COMPONENT_MAPPING(0,1,2,3)
```

```
typedef enum D3D12_SHADER_COMPONENT_MAPPING {
    D3D12_SHADER_COMPONENT_MAPPING_FROM_MEMORY_COMPONENT_0, //요소 0(Red)
    D3D12_SHADER_COMPONENT_MAPPING_FROM_MEMORY_COMPONENT_1, //요소 1(Green)
    D3D12_SHADER_COMPONENT_MAPPING_FROM_MEMORY_COMPONENT_2, //요소 2(Blue)
    D3D12_SHADER_COMPONENT_MAPPING_FROM_MEMORY_COMPONENT_3, //요소 3(Alpha)
    D3D12_SHADER_COMPONENT_MAPPING_FORCE_VALUE_0, //0
    D3D12_SHADER_COMPONENT_MAPPING_FORCE_VALUE_1 //1
} D3D12_SHADER_COMPONENT_MAPPING;
```

텍스쳐(Texture)

- 쉐이더 리소스(텍스쳐) 뷰 생성

```
m_ppd3dTextureUploadBuffers = new ID3D12Resource*[4];
m_ppd3dTextures = new ID3D12Resource*[4];
m_ppd3dTextures[0] = ::CreateTextureResourceFromFile(pd3dDevice, pd3dCommandList,
L"../Assets/Image/Miscellaneous/Stones.dds", &m_ppd3dTextureUploadBuffers[0]);
...
D3D12_DESCRIPTOR_HEAP_DESC d3dDescriptorHeapDesc;
d3dDescriptorHeapDesc.NumDescriptors = m_nGameObjects + 4; //CBVs + SRVs
d3dDescriptorHeapDesc.Type = D3D12_DESCRIPTOR_HEAP_TYPE_CBV_SRV_UAV;
d3dDescriptorHeapDesc.Flags = D3D12_DESCRIPTOR_HEAP_FLAG_SHADER_VISIBLE;
d3dDescriptorHeapDesc.NodeMask = 0;
pd3dDevice->CreateDescriptorHeap(&d3dDescriptorHeapDesc, IID_PPV_ARGS(&m_pd3dCbvSrvHeap));
D3D12_CPU_DESCRIPTOR_HANDLE d3dSrvCPUHandle = m_pd3dCbvSrvHeap-
>GetCPUDescriptorHandleForHeapStart();
d3dSrvCPUHandle.ptr = d3dSrvCPUHandle.ptr + (::gnCbvSrvIncrementSize * m_nGameObjects);
D3D12_SHADER_RESOURCE_VIEW_DESC d3dSRVDesc;
for (int i = 0; i < 4; i++) {
   D3D12_RESOURCE_DESC d3dTextureResourceDesc = m_ppd3dTextures[i]->GetDesc();
   d3dSRVDesc.Format = d3dTextureResourceDesc.Format;
   d3dSRVDesc.ViewDimension = D3D12_SRV_DIMENSION_TEXTURE2D;
   d3dSRVDesc.Shader4ComponentMapping = D3D12_DEFAULT_SHADER_4_COMPONENT_MAPPING;
   d3dSRVDesc.Texture2D.MostDetailedMip = d3dSRVDesc.Texture2D.PlaneSlice = 0;
   d3dSRVDesc.Texture2D.MipLevels = 1;
   d3dSRVDesc.Texture2D.ResourceMinLODClamp = 0.0f;
   pd3dDevice->CreateShaderResourceView(m_ppd3dTextures[i], &d3dSRVDesc, d3dSrvCPUHandle);
   d3dSrvCPUHandle.ptr += ::gnCbvSrvIncrementSize;
}
```

텍스쳐(Texture)

- **루트 시그너쳐(Root Signature)**

```
typedef struct D3D12_ROOT_PARAMETER {
    D3D12_ROOT_PARAMETER_TYPE ParameterType;
    union {
        D3D12_ROOT_DESCRIPTOR_TABLE DescriptorTable;
        D3D12_ROOT_CONSTANTS Constants; //상수 배열
        D3D12_ROOT_DESCRIPTOR Descriptor; //서술자
    };
    D3D12_SHADER_VISIBILITY ShaderVisibility;
} D3D12_ROOT_PARAMETER;
```

```
typedef enum D3D12_ROOT_PARAMETER_TYPE {
    D3D12_ROOT_PARAMETER_TYPE_DESCRIPTOR_TABLE,
    D3D12_ROOT_PARAMETER_TYPE_32BIT_CONSTANTS,
    D3D12_ROOT_PARAMETER_TYPE_CBV,
    D3D12_ROOT_PARAMETER_TYPE_SRV,
    D3D12_ROOT_PARAMETER_TYPE_UAV
} D3D12_ROOT_PARAMETER_TYPE;
```

```
typedef enum D3D12_SHADER_VISIBILITY {
    D3D12_SHADER_VISIBILITY_ALL,
    D3D12_SHADER_VISIBILITY_VERTEX,
    D3D12_SHADER_VISIBILITY_HULL,
    D3D12_SHADER_VISIBILITY_DOMAIN,
    D3D12_SHADER_VISIBILITY_GEOMETRY,
    D3D12_SHADER_VISIBILITY_PIXEL
} D3D12_SHADER_VISIBILITY;
```

```
typedef struct D3D12_ROOT_CONSTANTS {
    UINT ShaderRegister; //레지스터 번호
    UINT RegisterSpace; //레지스터 공간
    UINT Num32BitValues; //32-비트 상수의 개수
} D3D12_ROOT_CONSTANTS;
```

```
typedef struct D3D12_ROOT_DESCRIPTOR {
    UINT ShaderRegister; //레지스터 번호
    UINT RegisterSpace; //레지스터 공간
} D3D12_ROOT_DESCRIPTOR;
```

```
typedef struct D3D12_ROOT_DESCRIPTOR_TABLE {
    UINT NumDescriptorRanges;
    D3D12_DESCRIPTOR_RANGE *pRanges;
} D3D12_ROOT_DESCRIPTOR_TABLE;
```

```
typedef struct D3D12_DESCRIPTOR_RANGE {
    D3D12_DESCRIPTOR_RANGE_TYPE RangeType;
    UINT NumDescriptors; //서술자의 개수
    UINT BaseShaderRegister; //레지스터 번호
    UINT RegisterSpace; //레지스터 공간
    UINT OffsetInDescriptorsFromTableStart;
} D3D12_DESCRIPTOR_RANGE;
```

```
typedef enum D3D12_DESCRIPTOR_RANGE_TYPE {
    D3D12_DESCRIPTOR_RANGE_TYPE_SRV,
    D3D12_DESCRIPTOR_RANGE_TYPE_UAV,
    D3D12_DESCRIPTOR_RANGE_TYPE_CBV,
    D3D12_DESCRIPTOR_RANGE_TYPE_SAMPLER
} D3D12_DESCRIPTOR_RANGE_TYPE;
```

텍스쳐(Texture)

- **루트 시그너쳐(Root Signature)**

```
typedef struct D3D12_ROOT_PARAMETER {
    D3D12_ROOT_PARAMETER_TYPE ParameterType;
    union {
        D3D12_ROOT_DESCRIPTOR_TABLE DescriptorTable;
        D3D12_ROOT_CONSTANTS Constants; //상수 배열
        D3D12_ROOT_DESCRIPTOR Descriptor; //서술자
    };
    D3D12_SHADER_VISIBILITY ShaderVisibility;
} D3D12_ROOT_PARAMETER;
```

```
typedef struct D3D12_ROOT_DESCRIPTOR_TABLE {
    UINT NumDescriptorRanges;
    D3D12_DESCRIPTOR_RANGE *pRanges;
} D3D12_ROOT_DESCRIPTOR_TABLE;
```

```
typedef struct D3D12_DESCRIPTOR_RANGE {
    D3D12_DESCRIPTOR_RANGE_TYPE RangeType;
    UINT NumDescriptors; //서술자의 개수
    UINT BaseShaderRegister; //레지스터 번호
    UINT RegisterSpace; //레지스터 공간
    UINT OffsetInDescriptorsFromTableStart;
} D3D12_DESCRIPTOR_RANGE;
```

```
D3D12_DESCRIPTOR_RANGE pd3dDescriptorRanges[1];
pd3dDescriptorRanges[0].NumDescriptors = 1;
pd3dDescriptorRanges[0].BaseShaderRegister = 2; //Game Object
pd3dDescriptorRanges[0].RegisterSpace = 0;
pd3dDescriptorRanges[0].RangeType = D3D12_DESCRIPTOR_RANGE_TYPE_CBV;
pd3dDescriptorRanges[0].OffsetInDescriptorsFromTableStart = 0;
...
pd3dRootParameters[2].ParameterType = D3D12_ROOT_PARAMETER_TYPE_DESCRIPTOR_TABLE;
pd3dRootParameters[2].DescriptorTable.NumDescriptorRanges = 1;
pd3dRootParameters[2].DescriptorTable.pDescriptorRanges = &pd3dDescriptorRanges[0];
pd3dRootParameters[2].ShaderVisibility = D3D12_SHADER_VISIBILITY_ALL;
```

```
cbuffer cbGameObjectInfo : register(b2) {
    matrix gmtxGameObject : packoffset(c0);
};
```

텍스쳐(Texture)

- **루트 시그너쳐(Root Signature)**

```
void ID3D12GraphicsCommandList::SetGraphicsRootDescriptorTable(
    [in] UINT RootParameterIndex, //루트 파라메터 인덱스
    [in] D3D12_GPU_DESCRIPTOR_HANDLE BaseDescriptor //서술자 핸들(서술자 힙의 인덱스)
);
```

```
ID3D12DescriptorHeap *m_pd3dCbvDescriptorHeap;
pd3dDevice->CreateDescriptorHeap(..., (void **)&m_pd3dCbvSrvDescriptorHeap);
```

```
ID3D12Resource *m_pd3dcbGameObject;
pd3dDevice->CreateCommittedResource(..., &m_pd3dcbGameObject);
m_pd3dcbGameObject->Map(0, NULL, (void **)&m_pcbMappedGameObject);
```

```
D3D12_CPU_DESCRIPTOR_HANDLE d3dCbvCPUDescriptorHandle = m_pd3dCbvDescriptorHeap-
>GetCPUDescriptorHandleForHeapStart();
D3D12_CONSTANT_BUFFER_VIEW_DESC d3dcbvDesc;
d3dcbvDesc.SizeInBytes = ((sizeof(CB_GAMEOBJECT_INFO) + 255) & ~255);
d3dcbvDesc.BufferLocation = m_pd3dcbGameObject->GetGPUVirtualAddress();

pd3dDevice->CreateConstantBufferView(&d3dcbvDesc, d3dCbvCPUDescriptorHandle);
```

```
XMStoreFloat4x4(&m_pcbMappedGameObject->m_xmf4x4World, ...);
```

```
D3D12_GPU_DESCRIPTOR_HANDLE d3dCbvGPUDescriptorHandle = m_pd3dCbvDescriptorHeap-
>GetGPUDescriptorHandleForHeapStart();
pd3dCommandList->SetGraphicsRootDescriptorTable(2, d3dCbvGPUDescriptorHandle);
```

텍스쳐(Texture)

- **루트 시그너쳐(Root Signature)**

```
void ID3D12GraphicsCommandList::SetGraphicsRootDescriptorTable(
    [in] UINT RootParameterIndex, //루트 파라메터 인덱스
    [in] D3D12_GPU_DESCRIPTOR_HANDLE BaseDescriptor //서술자 핸들(서술자 힙의 인덱스)
);
```

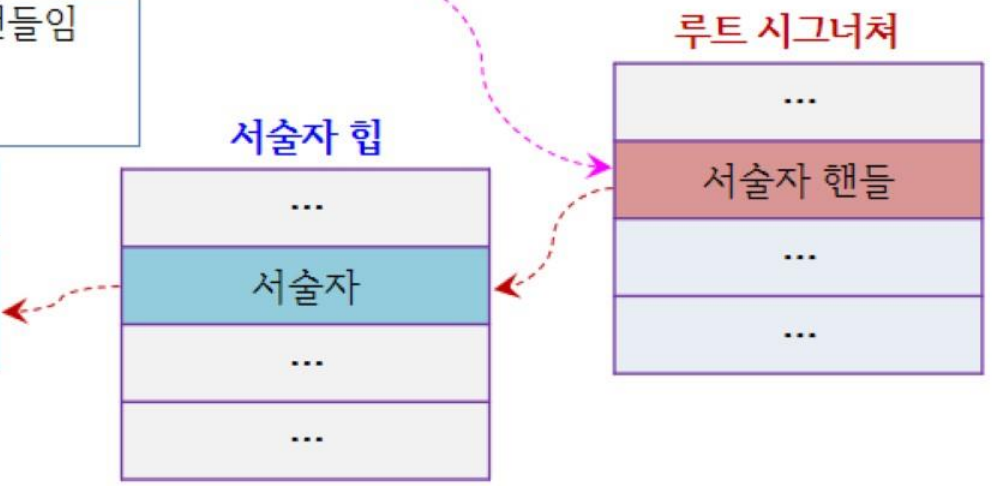
```
pd3dDevice->CreateDescriptorHeap(..., (void **)&m_pd3dCbvDescriptorHeap);
```

```
pd3dDevice->CreateCommittedResource(..., &m_pd3dcbGameObject);
```

```
pd3dDevice->CreateConstantBufferView(..., d3dCbvCPUDescriptorHandle);
```

```
pd3dCommandList->SetGraphicsRootDescriptorTable(2, d3dCbvGPUDescriptorHandle);
```

루트 시그너쳐(테이블) 원소의 값은 리소스의 서술자 핸들임
서술자 핸들은 서술자 힙의 주소(인덱스)를 의미함
리소스에 쉐이더 변수의 값을 저장함




```
ID3D12Resource *m_pd3dcbGameObject;
```

16개 실수

```
cbuffer cbGameObjectInfo : register(b2) {
    matrix gmtxGameObject : packoffset(c0);
};
```

```
ID3D12DescriptorHeap *m_pd3dCbvDescriptorHeap;
```

텍스쳐(Texture)

- **서술자 힙(ID3D12DescriptorHeap)**
  - 리소스를 서술하는 서술자들을 저장하는 연속적인 메모리 영역(배열)

```
HRESULT ID3D12Device::CreateDescriptorHeap(
    D3D12_DESCRIPTOR_HEAP_DESC *pDescriptorHeapDesc,
    REFIID riid, //__uuidof(ID3D12DescriptorHeap)
    void **ppvHeap //ID3D12DescriptorHeap
);
```

```
typedef struct D3D12_DESCRIPTOR_HEAP_DESC {
    D3D12_DESCRIPTOR_HEAP_TYPE Type;
    UINT NumDescriptors; //서술자의 개수
    D3D12_DESCRIPTOR_HEAP_FLAGS Flags;
    UINT NodeMask; //단일 GPU: 0
} D3D12_DESCRIPTOR_HEAP_DESC;
```

```
typedef enum D3D12_DESCRIPTOR_HEAP_TYPE {
    D3D12_DESCRIPTOR_HEAP_TYPE_CBV_SRV_UAV,
    D3D12_DESCRIPTOR_HEAP_TYPE_SAMPLER,
    D3D12_DESCRIPTOR_HEAP_TYPE_RTV,
    D3D12_DESCRIPTOR_HEAP_TYPE_DSV
} D3D12_DESCRIPTOR_HEAP_TYPE;
```

```
typedef enum D3D12_DESCRIPTOR_HEAP_FLAGS {
    D3D12_DESCRIPTOR_HEAP_FLAG_NONE,
    D3D12_DESCRIPTOR_HEAP_FLAG_SHADER_VISIBLE
} D3D12_DESCRIPTOR_HEAP_FLAGS;
```

```
D3D12_DESCRIPTOR_HEAP_DESC ID3D12DescriptorHeap::GetDesc();
```

  - 디바이스(어댑터)마다 서술자 유형별 메모리 크기가 다름(32~64 바이트)
    서술자 힙을 사용하려면 서술자 배열의 원소의 크기를 알아야 함

```
UINT ID3D12Device::GetDescriptorHandleIncrementSize(
    [in] D3D12_DESCRIPTOR_HEAP_TYPE DescriptorHeapType
);
```

텍스쳐(Texture)

- **루트 시그너쳐 생성**
  - 텍스쳐(쉐이더 리소스 뷰) 루트 파라메터는 반드시 서술자 테이블을 사용하여 생성

```
ID3D12RootSignature *pd3dGraphicsRootSignature = NULL;

D3D12_DESCRIPTOR_RANGE pd3dDescriptorRanges[2];
...
pd3dDescriptorRanges[1].RangeType = D3D12_DESCRIPTOR_RANGE_TYPE_SRV;
pd3dDescriptorRanges[1].NumDescriptors = 1; //Texture2D : register(t0);
pd3dDescriptorRanges[1].BaseShaderRegister = 0; //t0
pd3dDescriptorRanges[1].RegisterSpace = 0;
pd3dDescriptorRanges[1].OffsetInDescriptorsFromTableStart = 0;
...
D3D12_ROOT_PARAMETER pd3dRootParameters[4];
...
pd3dRootParameters[3].ParameterType = D3D12_ROOT_PARAMETER_TYPE_DESCRIPTOR_TABLE;
pd3dRootParameters[3].DescriptorTable.NumDescriptorRanges = 1;
pd3dRootParameters[3].DescriptorTable.pDescriptorRanges = &pd3dDescriptorRanges[1];
pd3dRootParameters[3].ShaderVisibility = D3D12_SHADER_VISIBILITY_PIXEL;
...
D3D12SerializeRootSignature(&d3dRootSignatureDesc, D3D_ROOT_SIGNATURE_VERSION_1, ...);
pd3dDevice->CreateRootSignature(0, ..., (void **)&pd3dGraphicsRootSignature);
```

```
Texture2D gtxtTexture : register(t0);
float4 PSTextured(VS_TEXTURED_OUTPUT input) : SV_TARGET
{
    return(gtxtTexture.Sample(gWrapSamplerState, input.texCoord));
}
```

텍스쳐(Texture)

- 쉐이더 리소스 뷰 연결

```
void ID3D12GraphicsCommandList::SetGraphicsRootDescriptorTable(
   [in] UINT RootParameterIndex, //루트 파라메터 인덱스
   [in] D3D12_GPU_DESCRIPTOR_HANDLE BaseDescriptor //서술자의 주소(핸들)
);
```

```
D3D12_GPU_DESCRIPTOR_HANDLE ID3D12DescriptorHeap::GetGPUDescriptorHandleForHeapStart();
```

```
class CGameObject
{
   XMFLOAT4X4 m_xmf4x4World;
   D3D12_GPU_DESCRIPTOR_HANDLE m_d3dCbvGPUHandle;

   CMesh *m_pMesh;

   CMaterial *m_pMaterial = NULL;
};
```

```
class CMaterial
{
   ...
   CShader *m_pShader = NULL;
   ...
   CTexture *m_pTexture = NULL;
};
```

```
class CTexture
{
   ID3D12Resource *m_pd3dTexture = NULL;
   UINT m_nRootParameterIndex = -1;
   D3D12_GPU_DESCRIPTOR_HANDLE m_d3dSrvGpuHandle;

   void UpdateShaderVariables(ID3D12GraphicsCommandList *pd3dCommandList);
};
```

텍스쳐(Texture)

- 쉐이더 리소스 뷰 연결

```
void CGameObject::Render(ID3D12GraphicsCommandList *pd3dCommandList, CCamera *pCamera)
{
   OnPrepareRender();
   if (m_pMaterial)
   {
      if (m_pMaterial->m_pShader)
      {
         m_pMaterial->m_pShader->Render(pd3dCommandList, pCamera);
         m_pMaterial->m_pShader->UpdateShaderVariable(pd3dCommandList, &m_xmf4x4World);
      }
      m_pMaterial->UpdateShaderVariables(pd3dCommandList);
   }
   pd3dCommandList->SetGraphicsRootDescriptorTable(2, m_d3dCbvGPUHandle);
   if (m_pMesh) m_pMesh->Render(pd3dCommandList);
}
```

```
void CMaterial::UpdateShaderVariables(ID3D12GraphicsCommandList *pd3dCommandList)
{
   ...
   if (m_pTexture) UpdateShaderVariables(pd3dCommandList);
}
```

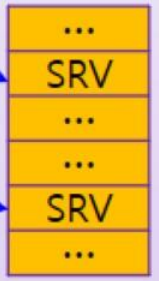
```
void CTexture::UpdateShaderVariables(ID3D12GraphicsCommandList *pd3dCommandList)
{
   pd3dCommandList->SetGraphicsRootDescriptorTable(m_nRootParameterIndex, m_d3dSrvGpuHandle);
}
```

텍스쳐(Texture)

- **루트 시그너쳐(Root Signature)**

```
D3D12_DESCRIPTOR_RANGE pd3dDescriptorRanges[2];
pd3dDescriptorRanges[0].RangeType = D3D12_DESCRIPTOR_RANGE_TYPE_SRV;
pd3dDescriptorRanges[0].NumDescriptors = 1;
pd3dDescriptorRanges[0].BaseShaderRegister = 0; //Texture: t0
pd3dDescriptorRanges[0].RegisterSpace = 0;
pd3dDescriptorRanges[0].OffsetInDescriptorsFromTableStart = 0;
pd3dDescriptorRanges[1].RangeType = D3D12_DESCRIPTOR_RANGE_TYPE_SRV;
pd3dDescriptorRanges[1].NumDescriptors = 1;
pd3dDescriptorRanges[1].BaseShaderRegister = 1; //Texture: t1
pd3dDescriptorRanges[1].RegisterSpace = 0;
pd3dDescriptorRanges[1].OffsetInDescriptorsFromTableStart = 0;

D3D12_ROOT_PARAMETER pd3dRootParameters[6];
...
pd3dRootParameters[2].ParameterType = D3D12_ROOT_PARAMETER_TYPE_DESCRIPTOR_TABLE;
pd3dRootParameters[2].DescriptorTable.NumDescriptorRanges = 1;
pd3dRootParameters[2].DescriptorTable.pDescriptorRanges = &pd3dDescriptorRanges[0];
pd3dRootParameters[2].ShaderVisibility = D3D12_SHADER_VISIBILITY_PIXEL;

pd3dRootParameters[3].ParameterType = D3D12_ROOT_PARAMETER_TYPE_DESCRIPTOR_TABLE;
pd3dRootParameters[3].DescriptorTable.NumDescriptorRanges = 1;
pd3dRootParameters[3].DescriptorTable.pDescriptorRanges = &pd3dDescriptorRanges[1];
pd3dRootParameters[3].ShaderVisibility = D3D12_SHADER_VISIBILITY_PIXEL;
```

서술자 힙



m_d3dSrvAlbedoGpuHandle

m_d3dSrvEmissionGpuHandle

```
Texture2D gtxtAlbedoTexture : register(t0);
Texture2D gtxtEmissionTexture : register(t1);
```

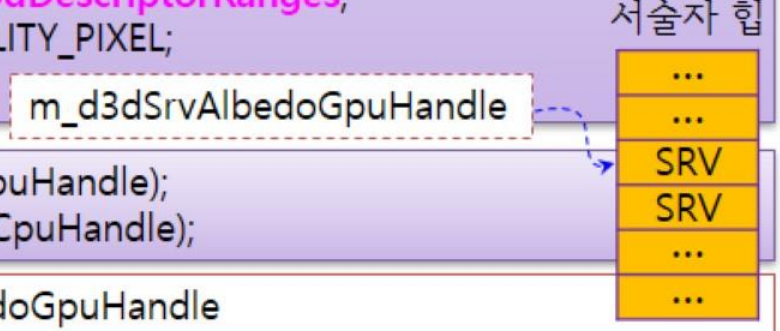
```
pd3dCommandList->SetGraphicsRootDescriptorTable(2, m_d3dSrvAlbedoGpuHandle);
pd3dCommandList->SetGraphicsRootDescriptorTable(3, m_d3dSrvEmissionGpuHandle);
```

텍스쳐(Texture)

- **루트 시그너쳐(Root Signature)**

```
D3D12_DESCRIPTOR_RANGE pd3dDescriptorRanges[2];
pd3dDescriptorRanges[0].RangeType = D3D12_DESCRIPTOR_RANGE_TYPE_SRV;
pd3dDescriptorRanges[0].NumDescriptors = 1;
pd3dDescriptorRanges[0].BaseShaderRegister = 0; //Texture: t0
pd3dDescriptorRanges[0].RegisterSpace = 0;
pd3dDescriptorRanges[0].OffsetInDescriptorsFromTableStart = 0;
pd3dDescriptorRanges[1].RangeType = D3D12_DESCRIPTOR_RANGE_TYPE_SRV;
pd3dDescriptorRanges[1].NumDescriptors = 1;
pd3dDescriptorRanges[1].BaseShaderRegister = 1; //Texture: t1
pd3dDescriptorRanges[1].RegisterSpace = 0;
pd3dDescriptorRanges[1].OffsetInDescriptorsFromTableStart = D3D12_DESCRIPTOR_RANGE_OFFSET_APPEND;

D3D12_ROOT_PARAMETER pd3dRootParameters[6];
...
pd3dRootParameters[2].ParameterType = D3D12_ROOT_PARAMETER_TYPE_DESCRIPTOR_TABLE;
pd3dRootParameters[2].DescriptorTable.NumDescriptorRanges = 2;
pd3dRootParameters[2].DescriptorTable.pDescriptorRanges = pd3dDescriptorRanges;
pd3dRootParameters[2].ShaderVisibility = D3D12_SHADER_VISIBILITY_PIXEL;
...
```

```
Texture2D gtxtAlbedoTexture : register(t0);
Texture2D gtxtEmissionTexture : register(t1);
```




```
pd3dDevice->CreateShaderResourceView(..., m_d3dSrvAlbedoCpuHandle);
pd3dDevice->CreateShaderResourceView(..., m_d3dSrvEmissionCpuHandle);
```

서술자 힙의 연속된 2개의 서술자의 시작 GPU 주소: m_d3dSrvAlbedoGpuHandle

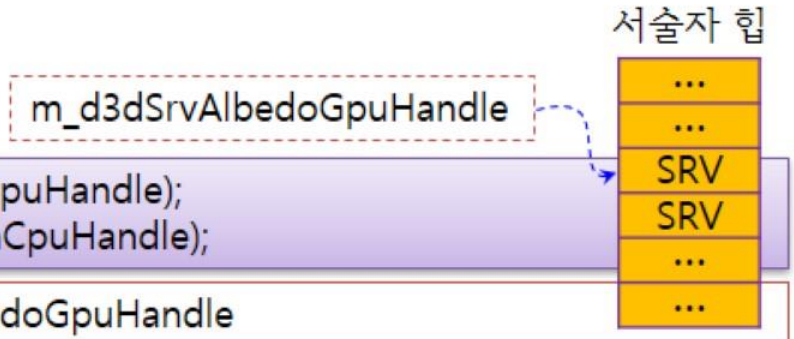
```
pd3dCommandList->SetGraphicsRootDescriptorTable(2, m_d3dSrvAlbedoGpuHandle);
```

텍스쳐(Texture)

- **루트 시그너쳐(Root Signature)**

```
D3D12_DESCRIPTOR_RANGE pd3dDescriptorRanges[1];
pd3dDescriptorRanges[0].RangeType = D3D12_DESCRIPTOR_RANGE_TYPE_SRV;
pd3dDescriptorRanges[0].NumDescriptors = 2;
pd3dDescriptorRanges[0].BaseShaderRegister = 0; //Texture: t0~t1
pd3dDescriptorRanges[0].RegisterSpace = 0;
pd3dDescriptorRanges[0].OffsetInDescriptorsFromTableStart = 0;

D3D12_ROOT_PARAMETER pd3dRootParameters[6];
...
pd3dRootParameters[2].ParameterType = D3D12_ROOT_PARAMETER_TYPE_DESCRIPTOR_TABLE;
pd3dRootParameters[2].DescriptorTable.NumDescriptorRanges = 1;
pd3dRootParameters[2].DescriptorTable.pDescriptorRanges = pd3dDescriptorRanges;
pd3dRootParameters[2].ShaderVisibility = D3D12_SHADER_VISIBILITY_PIXEL;
...
```

```
Texture2D gtxtAlbedoTexture : register(t0);
Texture2D gtxtEmissionTexture : register(t1);
```




```
pd3dDevice->CreateShaderResourceView(..., m_d3dSrvAlbedoCpuHandle);
pd3dDevice->CreateShaderResourceView(..., m_d3dSrvEmissionCpuHandle);
```

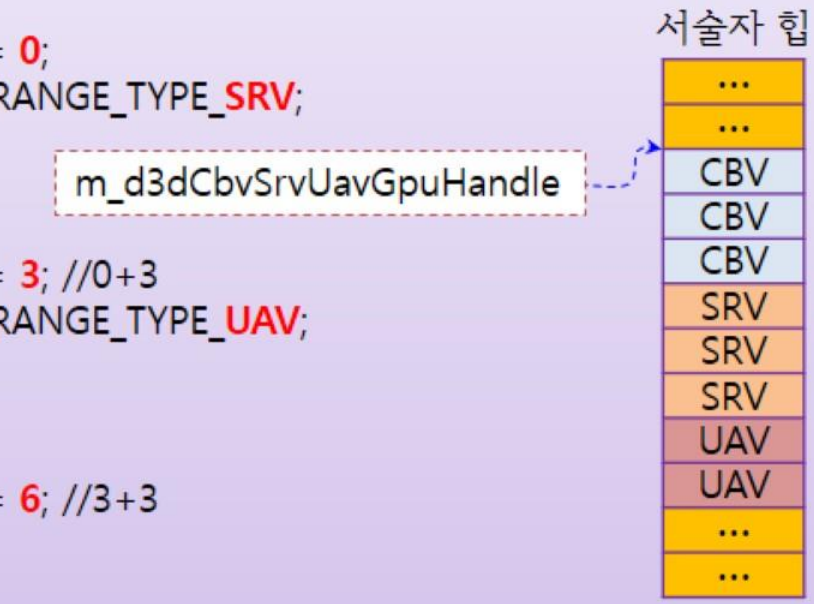
서술자 힙의 연속된 2개의 서술자의 시작 GPU 주소: m_d3dSrvAlbedoGpuHandle

```
pd3dCommandList->SetGraphicsRootDescriptorTable(2, m_d3dSrvAlbedoGpuHandle);
```

텍스쳐(Texture)

- **루트 시그너쳐(Root Signature)**

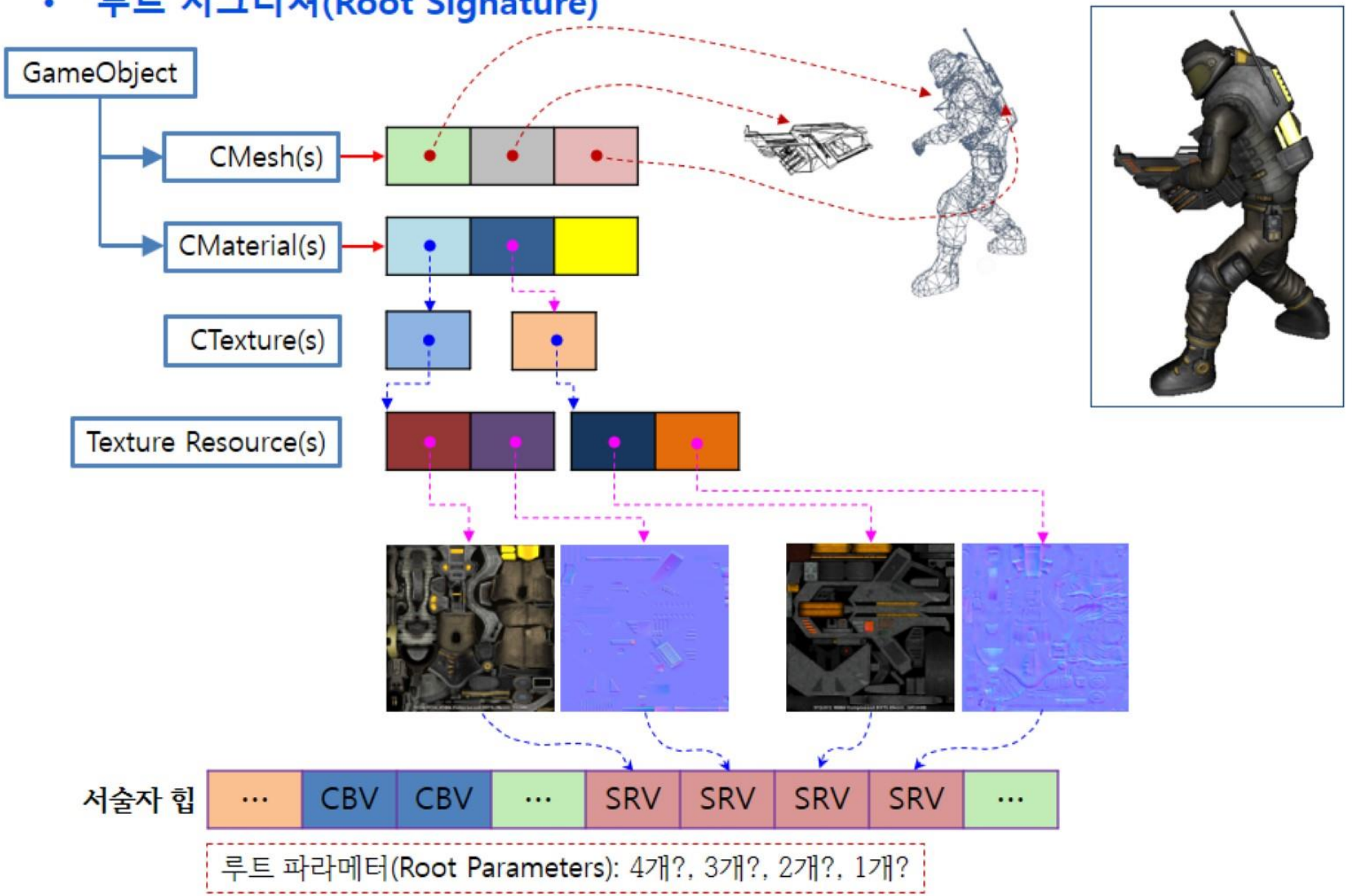
```
D3D12_DESCRIPTOR_RANGE pd3dDescriptorRanges[3];
pd3dDescriptorRanges[0].RangeType = D3D12_DESCRIPTOR_RANGE_TYPE_CBV;
pd3dDescriptorRanges[0].NumDescriptors = 3;
pd3dDescriptorRanges[0].BaseShaderRegister = 0; //b0~b2
pd3dDescriptorRanges[0].RegisterSpace = 0;
pd3dDescriptorRanges[0].OffsetInDescriptorsFromTableStart = 0;
pd3dDescriptorRanges[1].RangeType = D3D12_DESCRIPTOR_RANGE_TYPE_SRV;
pd3dDescriptorRanges[1].NumDescriptors = 3;
pd3dDescriptorRanges[1].BaseShaderRegister = 0; //t0~t2
pd3dDescriptorRanges[1].RegisterSpace = 0;
pd3dDescriptorRanges[1].OffsetInDescriptorsFromTableStart = 3; //0+3
pd3dDescriptorRanges[2].RangeType = D3D12_DESCRIPTOR_RANGE_TYPE_UAV;
pd3dDescriptorRanges[2].NumDescriptors = 2;
pd3dDescriptorRanges[2].BaseShaderRegister = 0; //u0~u1
pd3dDescriptorRanges[2].RegisterSpace = 0;
pd3dDescriptorRanges[2].OffsetInDescriptorsFromTableStart = 6; //3+3

D3D12_ROOT_PARAMETER pd3dRootParameters[6];
...
pd3dRootParameters[2].ParameterType = D3D12_ROOT_PARAMETER_TYPE_DESCRIPTOR_TABLE;
pd3dRootParameters[2].DescriptorTable.NumDescriptorRanges = 3;
pd3dRootParameters[2].DescriptorTable.pDescriptorRanges = pd3dDescriptorRanges;
pd3dRootParameters[2].ShaderVisibility = D3D12_SHADER_VISIBILITY_ALL;
...
```




```
pd3dCommandList->SetGraphicsRootDescriptorTable(2, m_d3dCbvSrvUavGpuHandle);
```

텍스쳐(Texture)

- 루트 시그너쳐(Root Signature)



텍스쳐(Texture)

- **서술자 힙(ID3D12DescriptorHeap)**
  - 리소스를 서술하는 서술자들을 저장하는 연속적인 메모리 영역(배열)

```
HRESULT ID3D12Device::CreateDescriptorHeap(
    D3D12_DESCRIPTOR_HEAP_DESC *pDescriptorHeapDesc,
    REFIID riid, //__uuidof(ID3D12DescriptorHeap)
    void **ppvHeap //ID3D12DescriptorHeap
);
```

```
typedef struct D3D12_DESCRIPTOR_HEAP_DESC {
    D3D12_DESCRIPTOR_HEAP_TYPE Type;
    UINT NumDescriptors; //서술자의 개수
    D3D12_DESCRIPTOR_HEAP_FLAGS Flags;
    UINT NodeMask; //단일 GPU: 0
} D3D12_DESCRIPTOR_HEAP_DESC;
```

```
typedef enum D3D12_DESCRIPTOR_HEAP_TYPE {
    D3D12_DESCRIPTOR_HEAP_TYPE_CBV_SRV_UAV,
    D3D12_DESCRIPTOR_HEAP_TYPE_SAMPLER,
    D3D12_DESCRIPTOR_HEAP_TYPE_RTV,
    D3D12_DESCRIPTOR_HEAP_TYPE_DSV
} D3D12_DESCRIPTOR_HEAP_TYPE;
```

```
typedef enum D3D12_DESCRIPTOR_HEAP_FLAGS {
    D3D12_DESCRIPTOR_HEAP_FLAG_NONE,
    D3D12_DESCRIPTOR_HEAP_FLAG_SHADER_VISIBLE
} D3D12_DESCRIPTOR_HEAP_FLAGS;
```

```
D3D12_DESCRIPTOR_HEAP_DESC ID3D12DescriptorHeap::GetDesc();
```

  - 디바이스(어댑터)마다 서술자 유형별 메모리 크기가 다름(32~64 바이트)
    서술자 힙을 사용하려면 서술자 배열의 원소의 크기를 알아야 함

```
UINT ID3D12Device::GetDescriptorHandleIncrementSize(
    [in] D3D12_DESCRIPTOR_HEAP_TYPE DescriptorHeapType
);
```

텍스쳐(Texture)

- 샘플러 상태(Sampler State) 생성

```
void ID3D12Device::CreateSampler(
    D3D12_SAMPLER_DESC *pDesc,
    D3D12_CPU_DESCRIPTOR_HANDLE DestDescriptor
);
```

```
typedef struct D3D12_SAMPLER_DESC {
    D3D12_FILTER Filter;
    D3D12_TEXTURE_ADDRESS_MODE AddressU;
    D3D12_TEXTURE_ADDRESS_MODE AddressV;
    D3D12_TEXTURE_ADDRESS_MODE AddressW;
    FLOAT MipLODBias;
    UINT MaxAnisotropy;
    D3D12_COMPARISON_FUNC ComparisonFunc;
    FLOAT BorderColor[4];
    FLOAT MinLOD;
    FLOAT MaxLOD;
} D3D12_SAMPLER_DESC;
```

```
typedef enum D3D12_FILTER {
    D3D12_FILTER_MIN_MAG_MIP_POINT,
    D3D12_FILTER_MIN_MAG_POINT_MIP_LINEAR,
    D3D12_FILTER_MIN_POINT_MAG_LINEAR_MIP_POINT,
    D3D12_FILTER_MIN_POINT_MAG_MIP_LINEAR,
    D3D12_FILTER_MIN_LINEAR_MAG_MIP_POINT,
    D3D12_FILTER_MIN_LINEAR_MAG_POINT_MIP_LINEA
    D3D12_FILTER_MIN_MAG_LINEAR_MIP_POINT,
    D3D12_FILTER_MIN_MAG_MIP_LINEAR,
    D3D12_FILTER_ANISOTROPIC,
    D3D12_FILTER_COMPARISON_MIN_MAG_MIP_POINT
    D3D12_FILTER_COMPARISON_MIN_MAG_POINT_MIP
    ...
    D3D12_FILTER_COMPARISON_ANISOTROPIC,
    D3D12_FILTER_MINIMUM_MIN_MAG_MIP_POINT,
    ...
    D3D12_FILTER_MAXIMUM_MIN_MAG_MIP_POINT,
    ...
    D3D12_FILTER_MAXIMUM_ANISOTROPIC
} D3D12_FILTER;
```

```
typ
    D3D12_TEXTURE_ADDRESS_MODE_WRAP,
    D3D12_TEXTURE_ADDRESS_MODE_MIRROR,
    D3D12_TEXTURE_ADDRESS_MODE_CLAMP,
    D3D12_TEXTURE_ADDRESS_MODE_BORDER ,
    D3D12_TEXTURE_ADDRESS_MODE_MIRROR_ONCE
} D3D12_TEXTURE_ADDRESS_MODE;
```

```
    D3D12_COMPARISON_FUNC_NEVER,
    D3D12_COMPARISON_FUNC_LESS,
    D3D12_COMPARISON_FUNC_EQUAL,
    D3D12_COMPARISON_FUNC_LESS_EQUAL,
    D3D12_COMPARISON_FUNC_GREATER,
    D3D12_COMPARISON_FUNC_NOT_EQUAL,
    D3D12_COMPARISON_FUNC_GREATER_EQUAL,
    D3D12_COMPARISON_FUNC_ALWAYS
} D3D12_COMPARISON_FUNC;
```

텍스쳐(Texture)

- **샘플러(Sampler) 객체**

```
void ID3D12Device::CreateSampler(
    D3D12_SAMPLER_DESC *pDesc,
    D3D12_CPU_DESCRIPTOR_HANDLE DestDescriptor
);
```

정점의 텍스쳐 좌표는 0보다 작거나 1보다 클 수 있음

```
typedef struct D3D12_SAMPLER_DESC {
    D3D12_FILTER Filter;
    D3D12_TEXTURE_ADDRESS_MODE AddressU;
    D3D12_TEXTURE_ADDRESS_MODE AddressV;
    D3D12_TEXTURE_ADDRESS_MODE AddressW;
    FLOAT MipLODBias;
    UINT MaxAnisotropy;
    D3D12_COMPARISON_FUNC ComparisonFunc;
    FLOAT BorderColor[4];
    FLOAT MinLOD;
    FLOAT MaxLOD;
} D3D12_SAMPLER_DESC;
```

Filter	샘플링할 때 사용할 필터링 방법, **D3D12_FILTER_MIN_MAG_MIP_LINEAR**
AddressU	0~1을 벗어난 u-좌표의 해결 방법, **D3D12_TEXTURE_ADDRESS_MODE_CLAMP**
AddressV	0~1을 벗어난 v-좌표의 해결 방법, **D3D12_TEXTURE_ADDRESS_MODE_CLAMP**
AddressW	0~1을 벗어난 w-좌표의 해결 방법, **D3D12_TEXTURE_ADDRESS_MODE_CLAMP**
MipLODBias	Direct3D가 계산한 LOD(밉맵) 레벨에 이 값을 더한 밉맵 레벨에서 샘플링, **0**
MaxAnisotropy	필터가 _ANISOTROPIC(비등방)일 때 사용되는 레벨 값(1~16), **1**
ComparisonFunc	샘플링한 값을 비교하기 위한 함수, **D3D12_COMPARISON_FUNC_NEVER**
BorderColor[4]	경계 색상(0~1), D3D12_TEXTURE_ADDRESS_MODE_BORDER, **float4(1.0f,1.0f,1.0f,1.0f)**
MinLOD	최소 밉맵 레벨, 밉맵 레벨 0이 가장 크고 세밀한 밉맵, **-FLT_MAX**
MaxLOD	최대 밉맵 레벨, MinLOD 보다 크거나 같아야 함, **FLT_MAX**

텍스쳐(Texture)

- **D3D12_FILTER**

D3D12_FILTER_MIN_MAG_MIP_POINT	축소, 확대, 밉맵: 점 샘플링(Point Sampling)
_MIN_MAG_POINT_MIP_LINEAR	축소, 확대: 점 샘플링, 밉맵: 선형 보간
_MIN_POINT_MAG_LINEAR_MIP_POINT	축소: 점 샘플링, 확대: 선형 보간, 밉맵: 점 샘플링
_MIN_POINT_MAG_MIP_LINEAR	축소: 점 샘플링, 확대, 밉맵: 선형 보간
_MIN_LINEAR_MAG_MIP_POINT	축소: 선형 보간, 확대, 밉맵: 점 샘플링
_MIN_LINEAR_MAG_POINT_MIP_LINEAR	축소: 선형 보간, 확대: 점 샘플링, 밉맵: 선형 보간
_MIN_MAG_LINEAR_MIP_POINT	축소, 확대: 선형 보간, 밉맵: 점 샘플링
_MIN_MAG_MIP_LINEAR	축소, 확대, 밉맵: 선형 보간
_ANISOTROPIC	축소, 확대, 밉맵: 비등방 보간(Anisotropic Interpolation)
_COMPARISON_MIN_MAG_MIP_POINT	축소, 확대, 밉맵: 점 샘플링. 결과를 비교
_MIN_MAG_POINT_MIP_LINEAR	축소, 확대: 점 샘플링, 밉맵: 선형 보간. 결과를 비교
_MIN_POINT_MAG_LINEAR_MIP_POINT	축소: 점 샘플링, 확대: 선형 보간, 밉맵: 점 샘플링. 결과를 비교
_MIN_POINT_MAG_MIP_LINEAR	축소: 점 샘플링, 확대, 밉맵: 선형 보간. 결과를 비교
_MIN_LINEAR_MAG_MIP_POINT	축소: 선형 보간, 확대, 밉맵: 점 샘플링. 결과를 비교
_MIN_LINEAR_MAG_POINT_MIP_LINEAR	축소: 선형 보간, 확대: 점 샘플링, 밉맵: 선형 보간. 결과를 비교
_MIN_MAG_LINEAR_MIP_POINT	축소, 확대: 선형 보간, 밉맵: 점 샘플링. 결과를 비교
_MIN_MAG_MIP_LINEAR	축소, 확대, 밉맵: 선형 보간. 결과를 비교
_ANISOTROPIC	축소, 확대, 밉맵: 비등방 보간, 결과를 비교
_MINIMUM_MIN_MAG_MIP_POINT	점 샘플링, 텍셀들의 최소값을 반환
_MAXIMUM_MIN_MAG_MIP_POINT	점 샘플링, 텍셀들의 최대값을 반환

텍스쳐(Texture)

- **텍스쳐 필터링(Texture Filtering)**
  - 필터링은 하나 이상의 텍셀을 읽고 결합하여 하나의 색상을 생성하는 과정임
  - 필터링은 픽셀 UV 좌표(실수)로 텍스쳐의 텍셀을 샘플링하는 방법에 영향을 줌
  - 비교 필터링(Comparison Filtering)은 샘플링된 각 텍셀을 비교값과 비교함

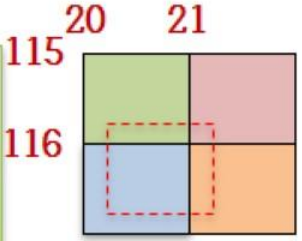
점 필터링(Point Filtering)

텍셀 좌표에 가장 가까운 텍셀을 사용, 실제로는 필터링을 하지 않는 의미

$(u,v) \Rightarrow (U,V) = (int(u * ImageWidth + 0.5), int(v * ImageHeight + 0.5))$

확대: 여러 픽셀에 대하여 (U,V)가 같아지므로 같은 텍셀 반복(계단 현상)

축소: 픽셀 좌표가 바뀔 때 많은 텍셀 정보가 없어짐(중요 정보 없어짐)



$(U,V) = (20.15, 115.75)$

선형 필터링(Bilinear Filtering)

$(u,v) \Rightarrow (U,V) = (u * ImageWidth, v * ImageHeight)$

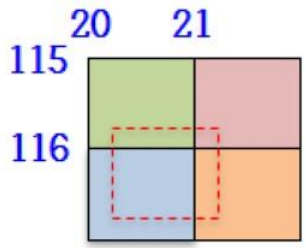
$(P,Q) = (int(u * ImageWidth), int(v * ImageHeight))$

정수 텍셀 좌표: $(P,Q), (P+1,Q), (P,Q+1), (P+1,Q+1)$

$\Delta U = U - P, \Delta V = V - Q$

가중 평균

샘플 색상 = $T(P,Q) * (1-\Delta U) * (1-\Delta V) + T(P+1,Q) * \Delta U * (1-\Delta V) +$
$T(P,Q+1) * (1-\Delta U) * \Delta V + T(P+1,Q+1) * \Delta U * \Delta V$



$(U,V) = (20.15, 115.75)$

샘플 색상 = $T(20,115) * (1-0.15) * (1-0.75) +$
$T(21,115) * (0.15) * (1-0.75) +$
$T(20,116) * (1-0.15) * (0.75) +$
$T(21,116) * (0.15) * (0.75)$

텍스쳐(Texture)

- **텍스쳐 필터링(Texture Filtering)**

Kaiser–Bessel window

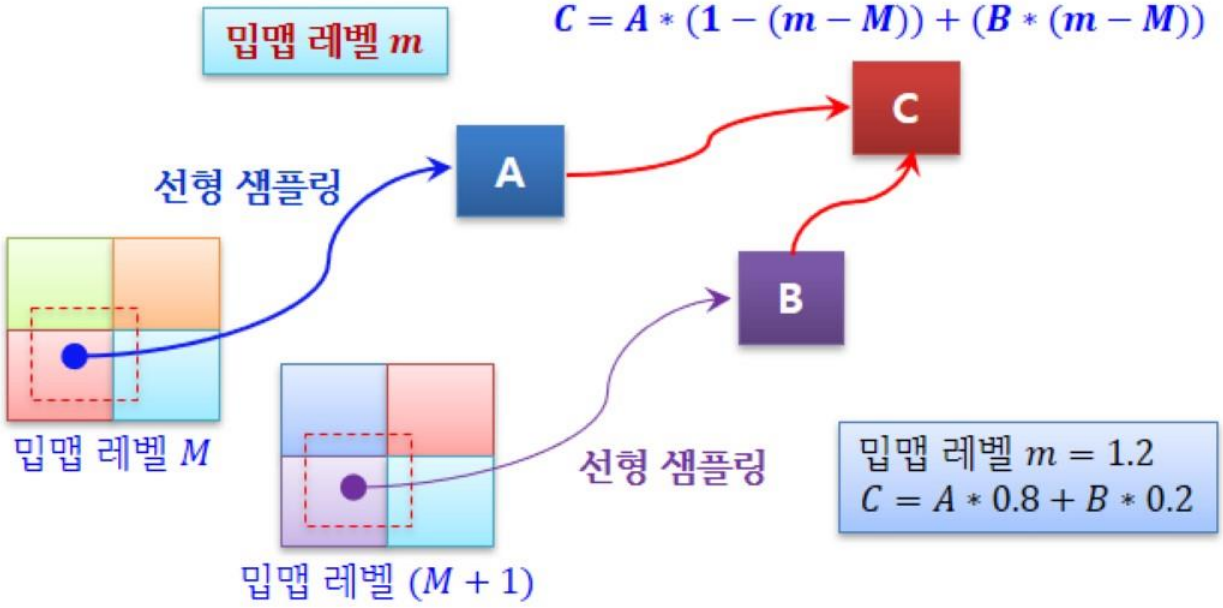
밉맵 필터링(Mip Map Filtering)

카메라까지의 거리에 따라 밉맵 레벨 계산: m(실수)
점 필터링: $M = int(m)$
선형 필터링(Trilinear Interpolation): 인접한 밉맵 레벨의 가중 평균 적용

부드러운 이동

밉맵 점 필터링

밉맵 선형 필터링

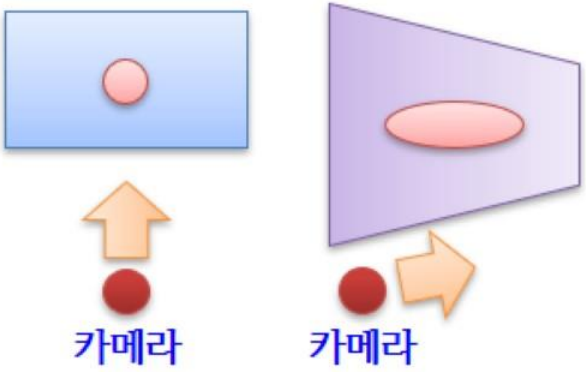


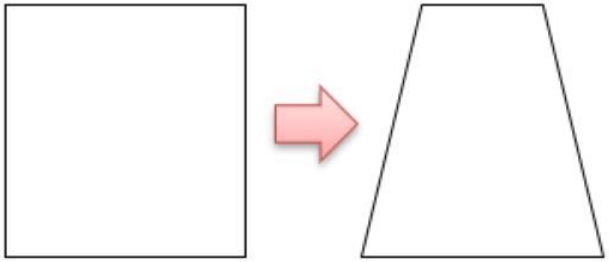
텍스쳐(Texture)

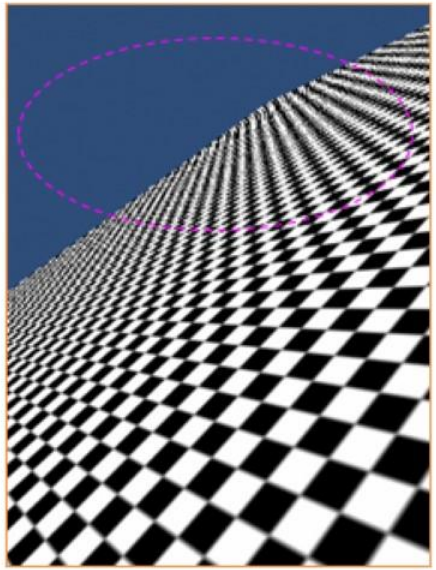
- **텍스쳐 필터링(Texture Filtering)**

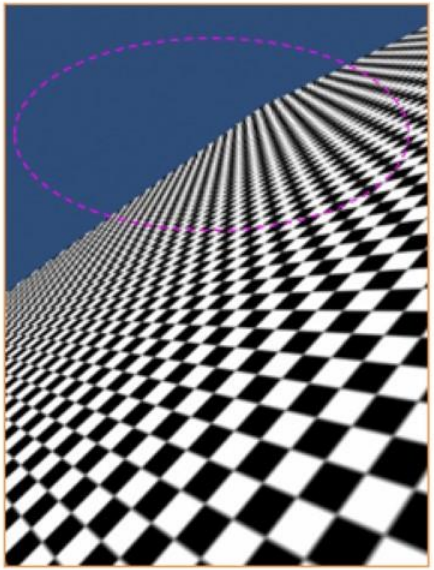
비등방 필터링(Anisotropic Filtering)

다각형의 법선 벡터와 카메라까지의 방향 벡터가 이루는 각이 커지면 원근 효과 때문에 텍스쳐가 늘어나게 된다(다각형이 한쪽 방향으로 늘어나서). 비등방 필터링은 이러한 경우에 텍스쳐가 한 방향으로 늘어나지 않도록 GPU에서 처리를 해준다(보통 1~16 단계).









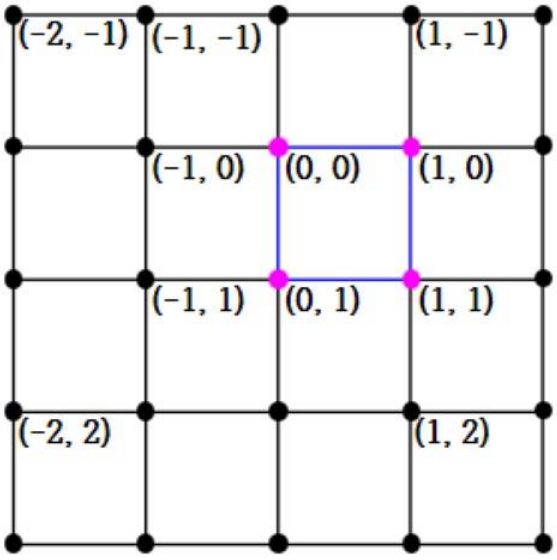
텍스쳐(Texture)

- **D3D12_TEXTURE_ADDRESS_MODE**

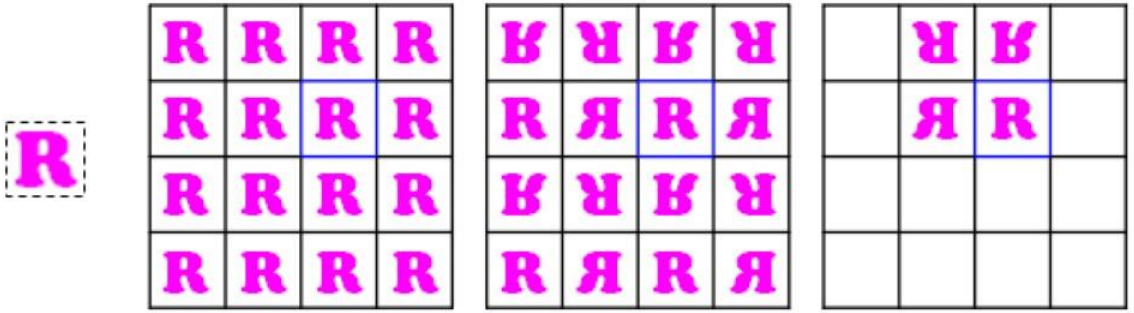
D3D12_TEXTURE_ADDRESS_MODE_WRAP	각 정수 위치에서 텍스쳐를 타일링(반복)
D3D12_TEXTURE_ADDRESS_MODE_MIRROR	각 정수 위치에서 텍스쳐를 플립(대칭 반복)
D3D12_TEXTURE_ADDRESS_MODE_CLAMP	0보다 작으면 0의 텍셀 색상, 1보다 크면 1의 텍셀 색상
D3D12_TEXTURE_ADDRESS_MODE_BORDER	0보다 작거나 1보다 크면 BorderColor[4] 색상을 사용
D3D12_TEXTURE_ADDRESS_MODE_MIRROR_ONCE	텍스쳐 좌표의 절대값을 사용, 1보다 크면 1의 텍셀 색상

D3D12_TEXTURE_ADDRESS_MODE_WRAP	$u = frac(u)$
D3D12_TEXTURE_ADDRESS_MODE_MIRROR	$if\ (floor(u) == 짝수)\ u = frac(u);\ else\ u = 1 - frac(u);$
D3D12_TEXTURE_ADDRESS_MODE_CLAMP	$if\ (u > 1)\ u = 1;\ else\ if\ (u < 0)\ u = 0;$
D3D12_TEXTURE_ADDRESS_MODE_BORDER	$if\ \big((u < 0)\ ||\ (u > 1)\big)\ return(BorderColor[]);$
D3D12_TEXTURE_ADDRESS_MODE_MIRROR_ONCE	$u = abs(u);\ if\ (u > 1)\ u = 1;$

$$frac(u) = \alpha \quad (u = floor(u) + \alpha, 0 \le \alpha < 1)$$



_WRAP	(1.2, 2.3) → (0.2, 0.3) (−0.2, 1.3) → (0.8, 0.3) (−1.2, −2.3) → (0.8, 0.7)
_MIRROR	(1.2, 2.3) → (0.8, 0.3) (−0.2, 1.3) → (0.2, 0.7) (−1.2, −2.3) → (0.8, 0.3)
_MIRROR_ONCE	(1.2, 2.3) → (1.0, 1.0) (−0.2, 1.3) → (0.2, 1.0) (−1.2, −2.3) → (1.0, 1.0)
_CLAMP	(1.2, 2.3) → (1.0, 1.0) (−0.2, 1.3) → (0.0, 1.0) (−1.2, −2.3) → (0.0, 0.0)



텍스쳐(Texture)

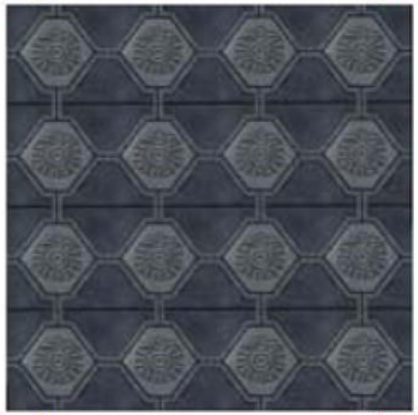
- D3D12_TEXTURE_ADDRESS_MODE

텍스쳐

(0, 0) (1, 0)

(0, 1) (1, 1)

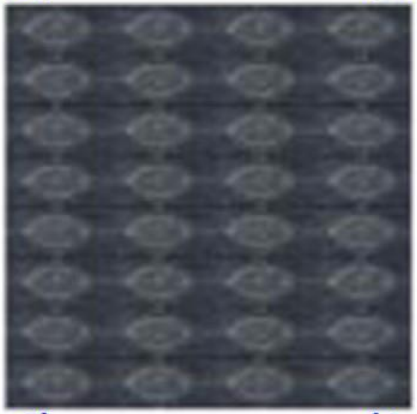
(0, 0) (4, 0)



(0, 4) (4, 4)

_WRAP

(0, 0) (4, 0)



(0, 8) (4, 8)

_WRAP

(0, 0) (1.4, 0)

(0, 1.4) (1.4, 1.4)

_CLAMP

텍스쳐

(0, 0) (1.3, 0)

(0, 1.3) (1.3, 1.3)

_BORDER

(0, 0) (2, 0)

(0, 2) (2, 2)

_MIRROR

텍스쳐(Texture)

- **루트 시그너쳐(Root Signature)**

```
HRESULT WINAPI D3D12SerializeRootSignature(
    D3D12_ROOT_SIGNATURE_DESC *pRootSignature,
    D3D_ROOT_SIGNATURE_VERSION Version,
    ID3DBlob **ppBlob, //직렬화된 루트 시그너쳐
    ID3DBlob **ppErrorBlob
);
```

```
LPVOID ID3DBlob::GetBufferPointer();
SIZE_T ID3DBlob::GetBufferSize();
```

```
HRESULT D3D12SerializeVersionedRootSignature(
    D3D12_VERSIONED_ROOT_SIGNATURE_DESC *pRootSignature,
    ID3DBlob **ppBlob,
    ID3DBlob **ppErrorBlob
);
```

```
typedef struct D3D12_VERSIONED_ROOT_SIGNATURE_DESC {
    D3D_ROOT_SIGNATURE_VERSION Version;
    union {
        D3D12_ROOT_SIGNATURE_DESC Desc_1_0;
        D3D12_ROOT_SIGNATURE_DESC1 Desc_1_1;
    };
} D3D12_VERSIONED_ROOT_SIGNATURE_DESC;
```

```
typedef enum D3D_ROOT_SIGNATURE_VERSION
    D3D_ROOT_SIGNATURE_VERSION_1,
    D3D_ROOT_SIGNATURE_VERSION_1_0,
    D3D_ROOT_SIGNATURE_VERSION_1_1
} D3D_ROOT_SIGNATURE_VERSION;
```

```
typedef struct D3D12_ROOT_SIGNATURE_DESC {
    UINT NumParameters; //루트 시그너쳐의 슬롯(파라메터) 개수
    D3D12_ROOT_PARAMETER *pParameters;
    UINT NumStaticSamplers; //정적 샘플러의 개수(2032개)
    D3D12_STATIC_SAMPLER_DESC *pStaticSamplers; //정적 샘플러 배열
    D3D12_ROOT_SIGNATURE_FLAGS Flags; //루트 시그너쳐 레이아웃을 위한 선택 사항
} D3D12_ROOT_SIGNATURE_DESC;
```

텍스쳐(Texture)

- 샘플러(Sampler) 객체

```
HRESULT WINAPI D3D12SerializeRootSignature(
    D3D12_ROOT_SIGNATURE_DESC *pRootSignature,
    D3D_ROOT_SIGNATURE_VERSION Version,
    ID3DBlob **ppBlob,
    ID3DBlob **ppErrorBlob
);
```

```
typedef struct D3D12_ROOT_SIGNATURE_DESC {
    UINT NumParameters;
    D3D12_ROOT_PARAMETER *pParameters;
    UINT NumStaticSamplers; //최대 2023개
    D3D12_STATIC_SAMPLER_DESC *pStaticSamplers;
    D3D12_ROOT_SIGNATURE_FLAGS Flags;
} D3D12_ROOT_SIGNATURE_DESC;
```

```
typedef struct D3D12_STATIC_SAMPLER_DESC {
    D3D12_FILTER Filter;
    D3D12_TEXTURE_ADDRESS_MODE AddressU;
    D3D12_TEXTURE_ADDRESS_MODE AddressV;
    D3D12_TEXTURE_ADDRESS_MODE AddressW;
    FLOAT MipLODBias;
    UINT MaxAnisotropy;
    D3D12_COMPARISON_FUNC ComparisonFunc;
    D3D12_STATIC_BORDER_COLOR BorderColor;
    FLOAT MinLOD;
    FLOAT MaxLOD;
    UINT ShaderRegister; //샘플러 레지스터 번호
    UINT RegisterSpace;
    D3D12_SHADER_VISIBILITY ShaderVisibility;
} D3D12_STATIC_SAMPLER_DESC;
```

```
typedef enum D3D12_STATIC_BORDER_COLOR {
    D3D12_STATIC_BORDER_COLOR_TRANSPARENT_BLACK,
    D3D12_STATIC_BORDER_COLOR_OPAQUE_BLACK,
    D3D12_STATIC_BORDER_COLOR_OPAQUE_WHITE
} D3D12_STATIC_BORDER_COLOR;
```

```
typedef enum D3D12_SHADER_VISIBILITY {
    D3D12_SHADER_VISIBILITY_ALL, //모든 쉐이더에서 사용
    D3D12_SHADER_VISIBILITY_VERTEX,
    D3D12_SHADER_VISIBILITY_HULL,
    D3D12_SHADER_VISIBILITY_DOMAIN,
    D3D12_SHADER_VISIBILITY_GEOMETRY,
    D3D12_SHADER_VISIBILITY_PIXEL
} D3D12_SHADER_VISIBILITY;
```

```
SamplerState gssSampler : register(s2, space3);
```

텍스쳐(Texture)

- 루트 시그너쳐 생성

```
...
D3D12_STATIC_SAMPLER_DESC pd3dSamplerDescs[2];
::ZeroMemory(&pd3dSamplerDescs, sizeof(D3D12_STATIC_SAMPLER_DESC) * 2);
pd3dSamplerDescs[0].Filter = D3D12_FILTER_MIN_MAG_MIP_LINEAR;
pd3dSamplerDescs[0].AddressU = D3D12_TEXTURE_ADDRESS_MODE_WRAP;
pd3dSamplerDescs[0].AddressV = D3D12_TEXTURE_ADDRESS_MODE_WRAP;
pd3dSamplerDescs[0].AddressW = D3D12_TEXTURE_ADDRESS_MODE_WRAP;
pd3dSamplerDescs[0].MaxAnisotropy = 1;
pd3dSamplerDescs[0].ComparisonFunc = D3D12_COMPARISON_FUNC_ALWAYS;
pd3dSamplerDescs[0].MaxLOD = D3D12_FLOAT32_MAX;
pd3dSamplerDescs[0].ShaderRegister = 0;
pd3dSamplerDescs[0].ShaderVisibility = D3D12_SHADER_VISIBILITY_PIXEL;
...
D3D12_ROOT_SIGNATURE_DESC d3dRootSignatureDesc;
...
d3dRootSignatureDesc.NumStaticSamplers = _countof(pd3dSamplerDescs);
d3dRootSignatureDesc.pStaticSamplers = pd3dSamplerDescs;
D3D12SerializeRootSignature(&d3dRootSignatureDesc, D3D_ROOT_SIGNATURE_VERSION_1, ...);
pd3dDevice->CreateRootSignature(0, ..., (void **)&pd3dGraphicsRootSignature);
```

```
Texture2D gtxtTexture : register(t0);
SamplerState gWrapSamplerState : register(s0);
float4 PSTextured(VS_TEXTURED_OUTPUT input) : SV_TARGET
{
    return(gtxtTexture.Sample(gWrapSamplerState, input.texCoord));
}
```

텍스쳐(Texture)

- **필터링(Filtering) 함수**

```
DXGI_FORMAT TextureObject.Sample(SamplerState S, float# Location[, int# Offset]);
DXGI_FORMAT TextureObject.SampleLevel(SamplerState S, float# Location, float LOD[, int# Offset]);
```

Texture1D, Texture1DArray, Texture2D, Texture2DArray, Texture3D, TextureCube, TextureCubeArray

```
float Texture.SampleCmp(SamplerComparisonState S, float# Location, float Value[, int# Offset]);
float Texture.SampleCmpLevelZero(SamplerComparisonState S, float# Location, float Value[, int# Offset]);
```

```
Texture2D gShadowMap : register(t1);
SamplerComparisonState ShadowSampler: register(s2);
...
float uShadow = gShadowMap.SampleCmpLevelZero(ShadowSampler, vModProjUV.xy, vModProjUV.z);
```

```
typedef enum D3D12_COMPARISON_FUNC {
    D3D12_COMPARISON_NEVER, //비교가 항상 실패
    D3D12_COMPARISON_LESS, //텍셀 값이 비교값보다 작으면 성공
    D3D12_COMPARISON_EQUAL,
    D3D12_COMPARISON_LESS_EQUAL,
    D3D12_COMPARISON_GREATER,
    D3D12_COMPARISON_NOT_EQUAL,
    D3D12_COMPARISON_GREATER_EQUAL,
    D3D12_COMPARISON_ALWAYS //비교가 항상 성공
} D3D12_COMPARISON_FUNC;
```

샘플링한 텍셀의 z-값을 비교값과 비교
비교의 결과에 성공하면 1을 반환
픽셀 쉐이더에서만 호출할 수 있음

다음의 텍셀 형식의 텍스쳐에 대해서 비교 필터링이 가능:
DXGI_FORMAT(R32_FLOAT_X8X24_TYPELESS, R32_FLOAT, R24_UNORM_X8_TYPELESS, R16_UNORM)

텍스쳐(Texture)

- **텍스쳐 변환(Transforming Texture)**
  - 2D 텍스쳐 좌표는 2D 좌표계로 표현됨
    텍스쳐 좌표를 변환(평행이동, 회전, 크기 변환)할 수 있음
    텍스쳐 좌표를 변환하면 텍스쳐 매핑의 결과가 달라짐
  - 텍스쳐 좌표의 크기 변환: 텍스쳐 좌표 (0, 1) → (0, 1) * 3 = (0, 3)
  - 텍스쳐 좌표의 평행이동 변환: 물, 구름 등이 움직이는 효과를 얻을 수 있음
  - 텍스쳐 좌표의 회전 변환: 입자(Particle) 효과에 유용

```
VS_TEXTURED_LIGHTING_OUTPUT VSTextureAnimate(VS_TEXTURED_LIGHTING_INPUT input)
{
   VS_TEXTURED_LIGHTING_OUTPUT output = (VS_TEXTURED_LIGHTING_OUTPUT)0;
   output.normalW = mul(input.normal, (float3x3)gmtxWorld);
   output.positionW = mul(float4(input.position, 1.0f), gmtxWorld).xyz;
   output.position = mul(mul(float4(output.positionW, 1.0f), gmtxView), gmtxProjection);
   output.uv0 = input.uv0;
   output.uv1 = mul(float4(input.uv1, 0.0f, 1.0f), gmtxTexture).xy;
   return(output);
}
```

```
cbuffer cbTextureMatrix : register(b2)
{
   matrix gmtxTexture : packoffset(c0);
};
```

```
float4 PSDetailTexturedLightingColor(VS_TEXTURED_LIGHTING_OUTPUT input) : SV_Target
{
   float4 cIllumination = Lighting(input.positionW, input.normalW);
   float4 cBaseTexColor = gtxtBase.Sample(gssDefault, input.uv0);
   float4 cDetailTexColor = gtxtDetail.Sample(gssDefaultDetail, input.uv1);
   return(saturate((cBaseTexColor * 0.5f) + (cDetailTexColor * 0.5f)) * cIllumination);
}
```

텍스쳐(Texture)

- **텍스쳐 배열(Texture Array)**

```
Texture2D gTreeTextures[4] : register(t4); //텍스쳐 레지스터 4~7 사용, 텍스쳐 크기와 형식이 다를 수 있음
SamplerState gSamplerState : register(s0);

float4 PS(VS_OUT input, uint primitiveID : SV_PrimitiveID) : SV_Target {
    float4 cColor = gTreeTextures[primitiveID % 4].Sample(gSamplerState, input.uv);
    ...
}
```

```
gTreeTextures[2].Sample(gSamplerState, input.uv); //OK
```

```
Texture2DArray gTreeTextureArray : register(t4); //텍스쳐 레지스터 4 사용, 텍스쳐 크기와 형식이 같아야 함
SamplerState gSamplerState : register(s0);

float4 PS(VS_OUT input, uint primitiveID : SV_PrimitiveID) : SV_Target {
    ...
    float3 uvw = float3(input.uv, (primitiveID % 4)); //w: 배열 인덱스
    float4 cColor = gTreeTextureArray.Sample(gSamplerState, uvw);
    ...
}
```

```
struct VS_OUT {
    float2 uv : TEXCOORD;
    ...
};
```

```
DXGI_FORMAT Object.Sample(
    sampler_state S,
    float# Location
    [, int# Offset]
);
```

Object	텍스쳐 객체 데이터 형
S	샘플러 상태 객체
Location	텍스쳐 좌표(텍스쳐 객체에 따라 다름)
Offset	텍스쳐 좌표 오프셋, 텍스쳐 좌표에 더해짐

텍스쳐 좌표 데이터 형

Texture1D	float
Texture1DArray, Texture2D	float2
Texture2DArray, Texture3D, TextureCube	**float3**
TextureCubeArray	float4

텍스쳐 좌표 오프셋 데이터 형

Texture1D, Texture1DArray	int
Texture2D, Texture2DArray	int2
Texture3D	int3
TextureCube, TextureCubeArray	x

텍스쳐(Texture)

- **텍스쳐 배열(Texture Array)**

```
Texture2D gTextures[4] : register(t4); //레지스터 4~7 사용(서술자 4개 필요), 텍스쳐 크기와 형식이 다를 수 있음
//루트 시그너쳐의 서술자와 호환될 수 있어야 함(서술자 개수가 배열의 크기보다 커야 함)
Texture2D<float4> gTextures[4] : register(t4);

Texture2D<float4> gTexture1 : register(t4, space0); //논리적 공간(space)
Texture2D<float4> gTextures2[4] : register(t10);
Texture2D<float4> gTextures3[4][3] : register(t10, space1);

Texture2D<float4> gTextures4[] : register(t20); //배열의 크기가 정해지지 않음(Unbounded Size)
//D3D12_DESCRIPTOR_RANGE.NumDescriptors = -1
//서술자 테이블의 마지막 원소만 바운드되지 않은 크기를 가질 수 있음
```

```
Texture2D<float4> gTextures2[4] : register(t10);
SamplerState gSamplers[7] : register(s0);

gTextures2[2].Sample(gSamplers[0], uv);
gTextures2[gnTextureIndex].Sample(gSamplers[gnSamplerIndex], ...); //Error
//텍스쳐 배열[]과 샘플러 배열[]의 인덱스는 기본적으로 하나의 Draw/Dispatch 호출동안 변할 수 없음

//텍스쳐 배열[]과 샘플러 배열[]의 인덱스를 변수로 사용하려면 NonUniformResourceIndex(…)를 사용
gTextures2[NonUniformResourceIndex(gnTextureIndex)].Sample(...);
```

```
cbuffer ... {
    uint gnTextureIndex;
    uint gnSamplerIndex;
}
```

```
Texture2DArray gTextureArray : register(t4); //레지스터 4 사용(서술자 1개), 텍스쳐 크기와 형식이 같아야 함
Texture2DArray<float4> gTextureArray : register(t4);
float3 uvw = float3(input.uv, gnTextureIndex); //w: 배열의 인덱스
gTextureArray.Sample(..., uvw); //인덱스를 변수로 자유롭게 사용할 수 있음
```

텍스쳐(Texture)

- **2D 텍스쳐 배열(Texture2DArray) 생성**
  - https://github.com/Microsoft/DirectXTex

DirectXTex_Desktop_2017.sln

DirectXTex-master\Texassemble\Bin\Desktop_2017\Win32\Debug\texassemble.exe

texassemble <**command**> <options> <**files**>

texassemble array –o texarray.dds -y tree01.bmp tree02.bmp

- cube
- volume
- **array**
- cubearray

입력 파일의 크기와 형식이 같아야 함
입력 파일은 밉맵 레벨이 1이어야 함(텍스쳐 배열의 밉맵을 생성하려면 texconv.exe를 사용)
출력 파일은 항상 dds 형식임

texassemble <command> <options> <**files**>

-r	wildcard filename search is recursive
-w <n>	width
-h <n>	height
-f <format>	format
-if <filter>	image filtering
-srgb{i|o}	sRGB {input, output}
-o <filename>	**output filename**
-y	overwrite existing output file (if any)
-sepalpha	resize alpha channel separately from color channels
-wrap, -mirror	texture addressing mode (wrap, mirror, or clamp)
-alpha	convert premultiplied alpha to straight alpha
-dx10	Force use of 'DX10' extended header
-nologo	suppress copyright message
-tonemap	apply a tonemap operator based on maximum luminance

텍스쳐(Texture)

- **2D 텍스쳐 배열(Texture2DArray) 생성**

```
ID3D12RootSignature *CScene::CreateGraphicsRootSignature(ID3D12Device *pd3dDevice)
{
    ...
    D3D12_DESCRIPTOR_RANGE pd3dDescriptorRanges[4];
    pd3dDescriptorRanges[0].RangeType = D3D12_DESCRIPTOR_RANGE_TYPE_CBV;
    ...
    pd3dDescriptorRanges[1].RangeType = D3D12_DESCRIPTOR_RANGE_TYPE_SRV;
    pd3dDescriptorRanges[1].NumDescriptors = 1;
    pd3dDescriptorRanges[1].BaseShaderRegister = 0; //Texture2D : register(t0);
    ...
    pd3dDescriptorRanges[2].RangeType = D3D12_DESCRIPTOR_RANGE_TYPE_SRV;
    pd3dDescriptorRanges[2].NumDescriptors = 1;
    pd3dDescriptorRanges[2].BaseShaderRegister = 1; //Texture2DArray gtxtTextures : register(t1);
    pd3dDescriptorRanges[2].RegisterSpace = 0;
    pd3dDescriptorRanges[2].OffsetInDescriptorsFromTableStart = 0;
    ...
    D3D12_ROOT_PARAMETER pd3dRootParameters[5];
    ...
    pd3dRootParameters[2].ParameterType = D3D12_ROOT_PARAMETER_TYPE_DESCRIPTOR_TABLE;
    pd3dRootParameters[2].DescriptorTable.NumDescriptorRanges = 1;
    pd3dRootParameters[2].DescriptorTable.pDescriptorRanges = &pd3dDescriptorRanges[2];
    pd3dRootParameters[2].ShaderVisibility = D3D12_SHADER_VISIBILITY_PIXEL;
    ...
    D3D12SerializeRootSignature(&d3dRootSignatureDesc, D3D_ROOT_SIGNATURE_VERSION_1, ...);
    pd3dDevice->CreateRootSignature(0, ..., (void **)&pd3dGraphicsRootSignature);
}
```

텍스쳐(Texture)

- **2D 텍스쳐 배열(Texture2DArray) 생성**

```
m_nTextures = 1;
m_ppd3dTextures = new ID3D12Resource*[m_nTextures];
m_ppd3dTextures[0] = ::CreateTextureResourceFromFile(pd3dDevice, pd3dCommandList, L"texarray.dds", ...);
...
D3D12_RESOURCE_DESC d3dTextureResourceDesc = m_ppd3dTextures[0]->GetDesc();
d3dSRVDesc.Format = d3dTextureResourceDesc.Format;
d3dSRVDesc.ViewDimension = D3D12_SRV_DIMENSION_TEXTURE2DARRAY;
d3dSRVDesc.Shader4ComponentMapping = D3D12_DEFAULT_SHADER_4_COMPONENT_MAPPING;
d3dSRVDesc.Texture2DArray.MostDetailedMip = 0;
d3dSRVDesc.Texture2DArray.MipLevels = d3dTextureResourceDesc.MipLevels;
d3dSRVDesc.Texture2DArray.FirstArraySlice = 0;
d3dSRVDesc.Texture2DArray.ArraySize = d3dTextureResourceDesc.DepthOrArraySize;
d3dSRVDesc.Texture2DArray.PlaneSlice = 0;
d3dSRVDesc.Texture2DArray.ResourceMinLODClamp = 0.0f;
pd3dDevice->CreateShaderResourceView(m_ppd3dTextures[0], &d3dSRVDesc, m_d3dSrvCPUHandle);
```

```
pd3dCommandList->SetGraphicsRootDescriptorTable(2, m_d3dSrvGPUHandle);
```

```
Texture2DArray gtxtTextures : register(t1);
SamplerState gWrapSamplerState : register(s0);
float4 PSTextured(VS_TEXTURED_OUTPUT input, uint primitiveID : SV_PrimitiveID) : SV_TARGET
{
   float3 uvw = float3(input.uv, primitiveID % 4);
   float4 cColor = gtxtTextures.Sample(gWrapSamplerState, uvw);
   return(cColor);
}
```

텍스쳐(Texture)

- 게임 객체의 자료 구조

```
class CGameObject {
   ...
   CMaterial *m_pMaterial;
};
```

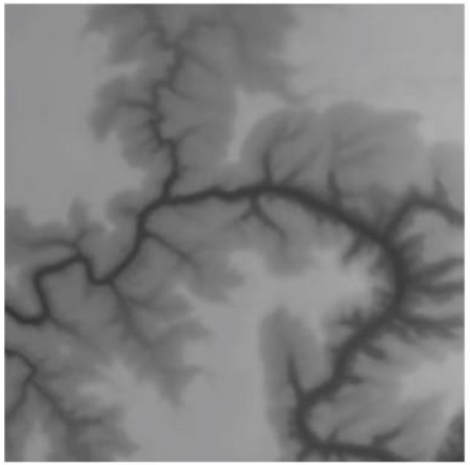
```
class CMaterial {
   CShader *m_pShader;
   CTexture *m_pTexture;
   XMFLOAT4m_xmf4Albedo;
   MATERIAL *m_pReflection;
};
```

```
struct MATERIAL {
   XMFLOAT4 m_xmf4Ambient;
   XMFLOAT4 m_xmf4Diffuse;
   XMFLOAT4 m_xmf4Specular;
   XMFLOAT4 m_xmf4Emissive;
};
```

```
class CTexture {
   UINT m_nTextureType;

   int m_nTextures = 0;
   _TCHAR(*m_ppstrTextureNames)[64] = NULL;
   ID3D12Resource** m_ppd3dTextures = NULL;
   ID3D12Resource** m_ppd3dTextureUploadBuffers;
   UINT* m_pnResourceTypes = NULL;
   DXGI_FORMAT* m_pdxgiBufferFormats = NULL;
   int* m_pnBufferElements = NULL;

   int m_nRootParameters = 0;
   UINT* m_pnRootParameterIndices = NULL;
   D3D12_GPU_DESCRIPTOR_HANDLE* m_pd3dSrvGpuDescriptorHandles = NULL;

   CTexture(int nTextureResources, UINT nResourceType, int nSamplers, int nRootParameters);
   void LoadTextureFromDDSFile(ID3D12Device* pd3dDevice, ID3D12GraphicsCommandList*
pd3dCommandList, wchar_t* pszFileName, UINT nResourceType, UINT nIndex);
   void SetRootParameterIndex(int nIndex, UINT nRootParameterIndex);
   void SetGpuDescriptorHandle(int nIndex, D3D12_GPU_DESCRIPTOR_HANDLE d3dSrvGpuDescriptorHandle);
   void UpdateShaderVariables(ID3D12GraphicsCommandList *pd3dCommandList);
};
```

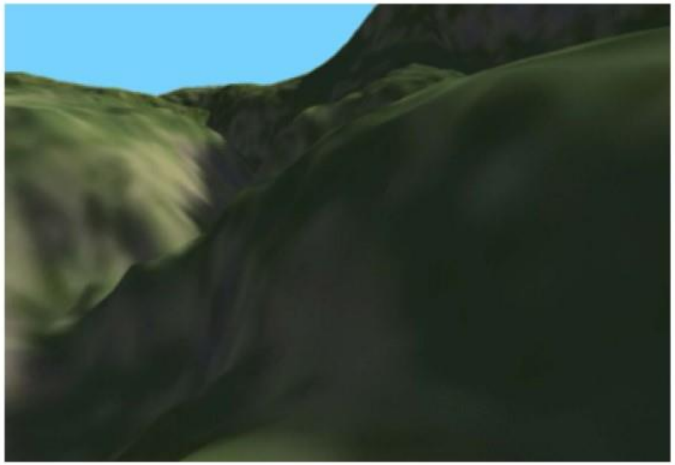
텍스쳐(Texture)

- 지형 텍스쳐 맵핑하기



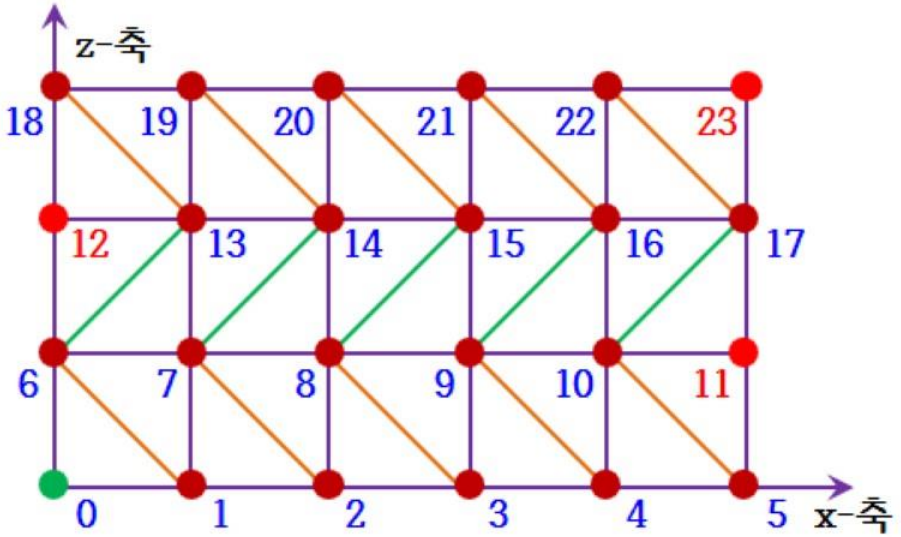
높이맵

지형 텍스쳐



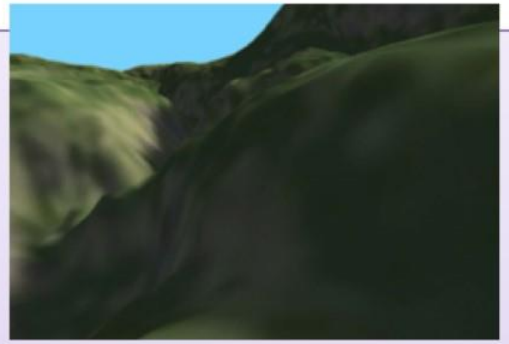
```
class CTexturedVertex
{
    XMFLOAT3 m_xmf3Position;
    ...
    XMFLOAT2 m_xmf2UV0;
};
```

```
u = x / float(ImageWidth-1);
v = (ImageHeight-1-z) / float(ImageHeight-1);
```



텍스쳐(Texture)

- 지형 텍스쳐 맵핑하기



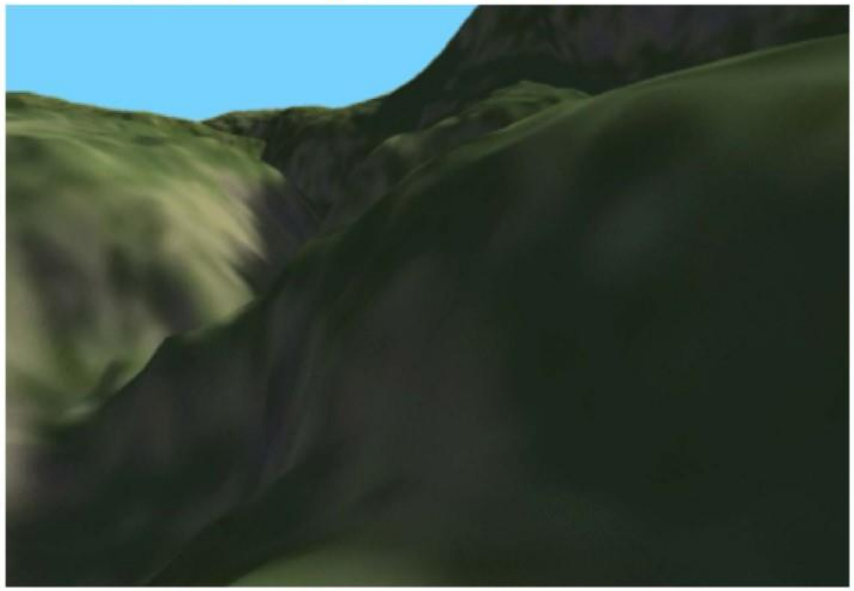
```
CHeightMapGridMesh::CHeightMapGridMesh(...)
{
   m_nVertices = nWidth * nLength;
   m_nStride = sizeof(CDiffusedTexturedVertex);
   m_nOffset = m_nSlot = 0;
   m_d3dPrimitiveTopology = D3D_PRIMITIVE_TOPOLOGY_TRIANGLESTRIP;
   m_nStride = sizeof(CDiffusedTexturedVertex);
   ...
   CDiffusedTexturedVertex *pVertices = new CDiffusedTexturedVertex[m_nVertices];
   CHeightMapImage *pHeightMapImage = (CHeightMapImage *)pContext;
   int cxHeightMap = pHeightMapImage->GetHeightMapWidth();
   int czHeightMap = pHeightMapImage->GetHeightMapLength();
   for (int i = 0, z = zStart; z < (zStart+nLength); z++)
   {
      for (int x = xStart; x < (xStart+nWidth); x++)
      {
         float fHeight = OnGetHeight(x, z, pContext);
         pVertices[i].m_xmf3Position = XMFLOAT3((x*m_xmf3Scale.x), fHeight, (z*m_xmf3Scale.z));
         pVertices[i].m_xmf3Color = ...;
         pVertices[i].m_xmf2UV0 = XMFLOAT2(float(x) / float(cxHeightMap - 1), float(czHeightMap - 1 - z)
/ float(czHeightMap - 1));
      }
   }
   ....
}
```

```
class CDiffusedTexturedVertex {
   XMFLOAT3 m_xmf3Position;
   XMFLOAT3 m_xmf3Color;
   XMFLOAT2 m_xmf2UV0;
};
```

```
class CNormalTexturedVertex {
   XMFLOAT3 m_xmf3Position;
   XMFLOAT3 m_xmf3Normal;
   XMFLOAT2 m_xmf2UV0;
};
```

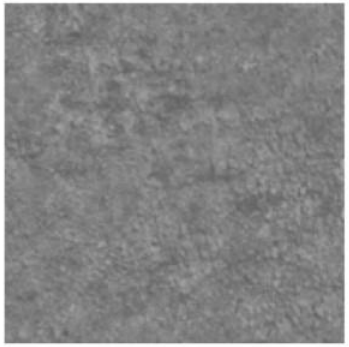
텍스쳐(Texture)

- 지형 텍스쳐 맵핑하기

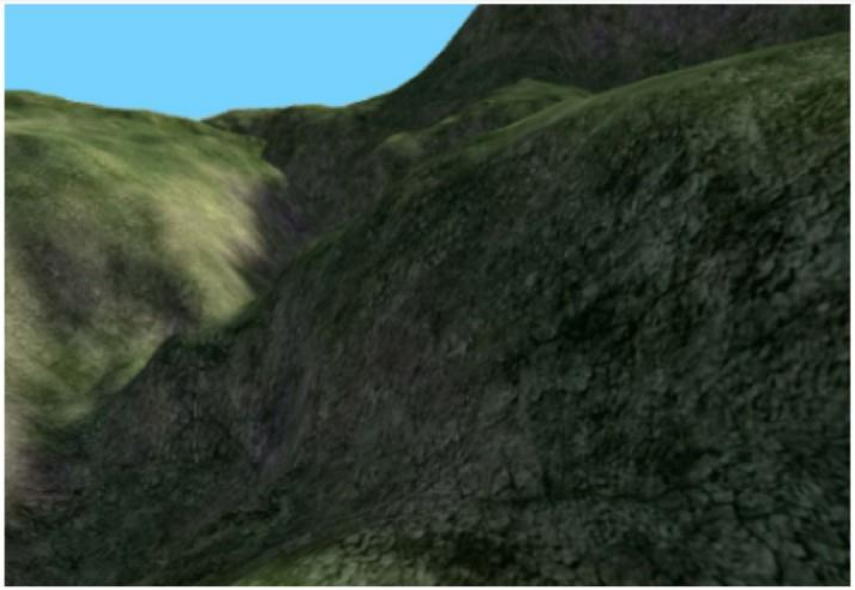


```
class CTexturedVertex {
    XMFLOAT3 m_xmf3Position;
    XMFLOAT3 m_xmf3Normal; //m_xmf3Color
    XMFLOAT2 m_xmf2UV0;
};
```

```
class CTexturedVertex {
    XMFLOAT3 m_xmf3Position;
    XMFLOAT3 m_xmf3Normal; //m_xmf3Color
    XMFLOAT2 m_xmf2UV0;
    XMFLOAT2 m_xmf2UV1;
};
```



디테일 텍스쳐



텍스쳐(Texture)

- 지형 텍스쳐 맵핑하기

```
class CDiffused2TexturedVertex {
    XMFLOAT3 m_xmf3Position;
    XMFLOAT3 m_xmf4Color;
    XMFLOAT2 m_xmf2UV0;
    XMFLOAT2 m_xmf2UV1;
};
```

```
CHeightMapGridMesh::CHeightMapGridMesh(...)
{
    m_nVertices = nWidth * nLength;
    m_nStride = sizeof(CDiffused2TexturedVertex);
    m_nOffset = m_nSlot = 0;
    m_d3dPrimitiveTopology = D3D_PRIMITIVE_TOPOLOGY_TRIANGLESTRIP;
    m_nStride = sizeof(CDiffusedTexturedVertex);
    ...
    CDiffusedTexturedVertex *pVertices = new CDiffusedTexturedVertex[m_nVertices];
    CHeightMapImage *pHeightMapImage = (CHeightMapImage *)pContext;
    int cxHeightMap = pHeightMapImage->GetHeightMapWidth();
    int czHeightMap = pHeightMapImage->GetHeightMapLength();
    for (int i = 0, z = zStart; z < (zStart+nLength); z++)
    {
        for (int x = xStart; x < (xStart+nWidth); x++)
        {
            pVertices[i].m_xmf3Position = XMFLOAT3((x*m_xmf3Scale.x), fHeight, (z*m_xmf3Scale.z));
            pVertices[i].m_xmf4Color = ...;
            pVertices[i].m_xmf2UV0 = XMFLOAT2(float(x) / float(cxHeightMap - 1), float(czHeightMap - 1 - z)
/ float(czHeightMap - 1));
            pVertices[i].m_xmf2UV1 = XMFLOAT2(float(x) / float(m_xmf3Scale.x*0.5f), float(z) /
float(m_xmf3Scale.z*0.5f));
        }
    }
    ....
}
```

텍스쳐(Texture)

- 지형 텍스쳐 맵핑하기

```
VS_TERRAIN_OUTPUT VSTerrain(VS_TERRAIN_INPUT input)
{
   VS_TERRAIN_OUTPUT output;
   ...
   output.uv0 = input.uv0;
   output.uv1 = input.uv1;
   return(output);
}
```

```
struct VS_TERRAIN_INPUT {
   float3 position : POSITION;
   float4 color : COLOR;
   float2 uv0 : TEXCOORD0;
   float2 uv1 : TEXCOORD1;
};
```

```
Texture2D gtxtTerrainBase : register(t4);
Texture2D gtxtTerrainDetail : register(t5);
```

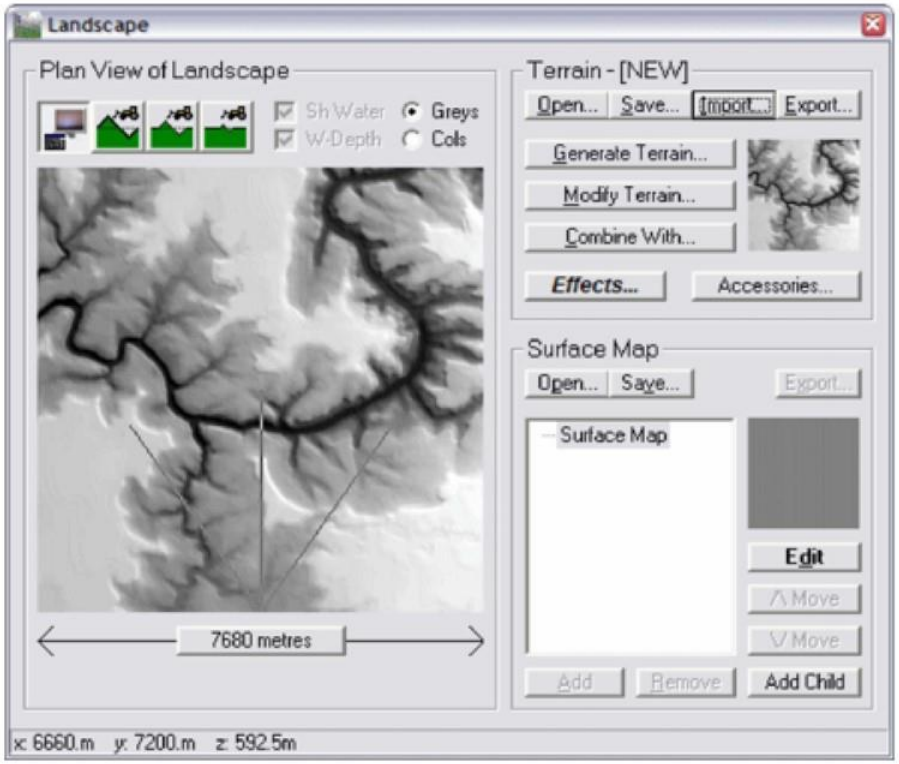
```
struct VS_TERRAIN_OUTPUT {
   float4 position : SV_POSITION;
   float4 color : COLOR;
   float2 uv0 : TEXCOORD0;
   float2 uv1 : TEXCOORD1;
};
```

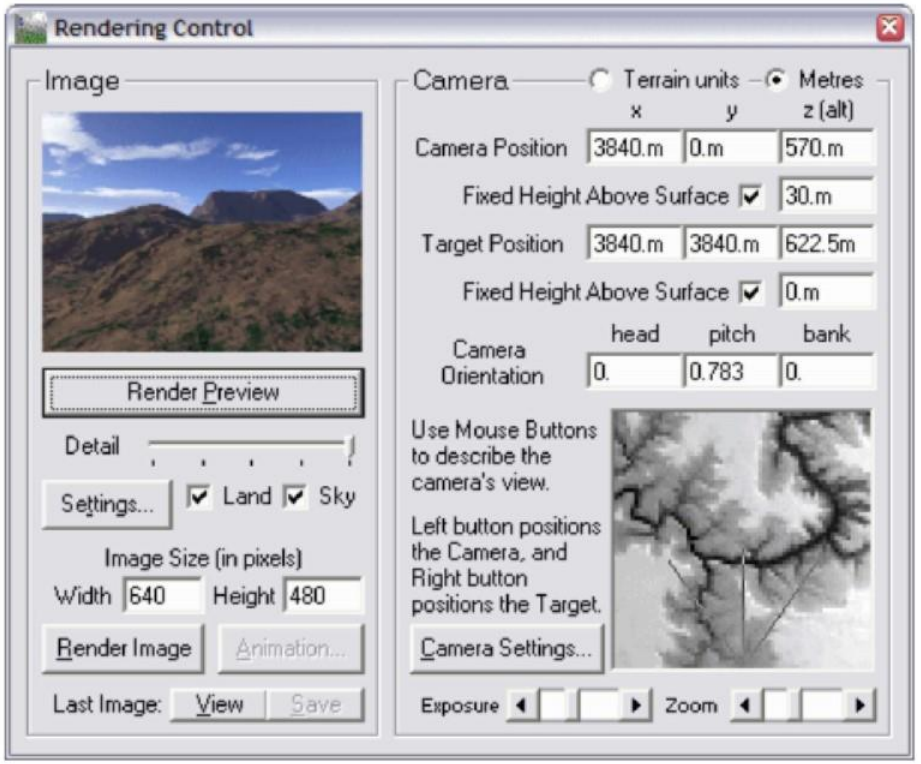
```
float4 PSTerrain(VS_TERRAIN_OUTPUT input) : SV_TARGET
{
   float4 cBaseTexColor = gtxtTerrainBase.Sample(gWrapSamplerState, input.uv0);
   float4 cDetailTexColor = gtxtTerrainDetail.Sample(gWrapSamplerState, input.uv1);
   float4 cColor = saturate(input.color * 0.5f + (cBaseTexColor + cDetailTexColor) * 0.5f);
   return(cColor);
}
```

```
float4 PSTerrain(VS_TERRAIN_OUTPUT input) : SV_TARGET
{
    float4 cIllumination = Lighting(input.positionW, input.normalW);
    float4 cBaseTexColor = gtxtTerrainBase.Sample(gWrapSamplerState, input.uv0);
    float4 cDetailTexColor = gtxtTerrainDetail.Sample(gWrapSamplerState, input.uv1);
    float4 cColor = saturate(cIllumination * 0.5f + (cBaseTexColor + cDetailTexColor) * 0.5f);
    return(cColor);
}
```

텍스쳐(Texture)

- **지형 텍스쳐 생성하기**
  - Terragen™
  - http://www.planetside.co.uk/terragen/





텍스쳐(Texture)

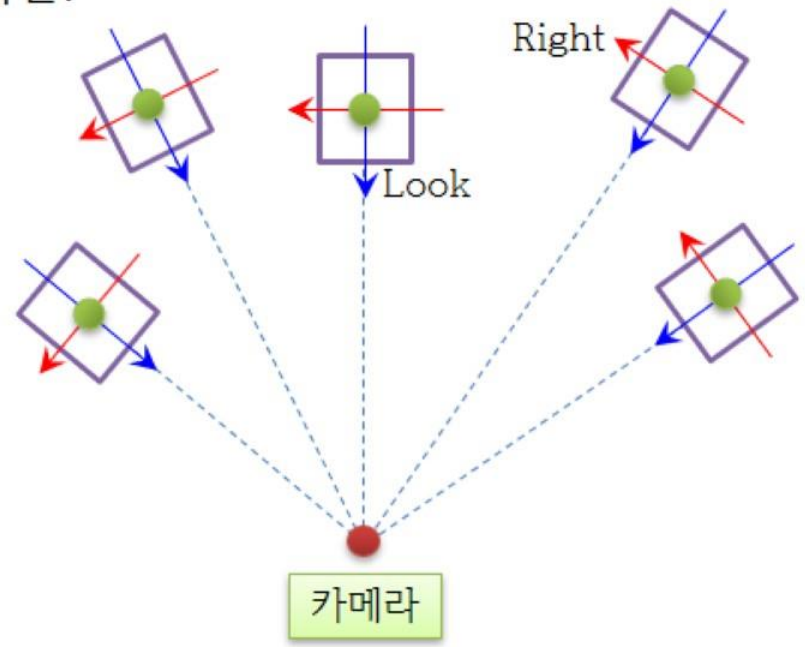
- **빌보드(Billboard)**
  - 나무(Tree), 광고판(Billboard), 전봇대(Pole)
    이러한 객체를 메쉬로 만들 수 있지만 객체가 아주 많다면 프레임 레이트가 낮아짐
    나무 모델(메쉬)은 다각형의 수가 상당히 많고 모델링이 어려움





텍스쳐(Texture)

- **빌보드(Billboard)**
  - 나무(Tree), 광고판(Billboard), 전봇대(Pole)
    메쉬를 사각형으로 만들고 텍스쳐 이미지를 매핑(투명 처리 또는 알파 블렌딩)
    카메라가 이동하면? 특히 나무 뒤로 이동하면?
    사각형이 항상 카메라를 향하도록 처리함



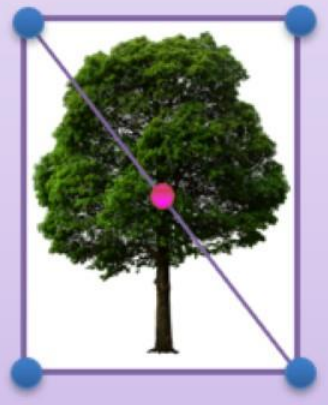
빌보드(사각형)의 중점(월드 좌표계) $P = (P_x, P_y, P_z)$
카메라의 위치(월드 좌표계) $C = (C_x, C_y, C_z)$
빌보드의 (Right, Up, Look) 벡터
Look = normalize(C – P)
Up = (0, 1, 0)
Right = Up x Look

텍스쳐(Texture)

- 사각형 메쉬(Rectangle Mesh)

```
CTexturedRectMesh::CTexturedRectMesh(ID3D12Device *pd3dDevice, ID3D12GraphicsCommandList
*pd3dCommandList, float fWidth, float fHeight, float fDepth, float fxPosition, float fyPosition, float
fzPosition) : CMesh(pd3dDevice, pd3dCommandList)
{
   m_nVertices = 6;
   m_nStride = sizeof(CTexturedVertex);
   m_d3dPrimitiveTopology = D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST;
   CTexturedVertex pVertices[6];
   float fx = (fWidth * 0.5f) + fxPosition, fy = (fHeight * 0.5f) + fyPosition, fz = (fDepth * 0.5f) + fzPosition;
   if (fWidth == 0.0f) {
      if (fxPosition > 0.0f) {
         pVertices[0] = CTexturedVertex(XMFLOAT3(fx, +fy, -fz), XMFLOAT2(1.0f, 0.0f));
         pVertices[1] = CTexturedVertex(XMFLOAT3(fx, -fy, -fz), XMFLOAT2(1.0f, 1.0f));
         pVertices[2] = CTexturedVertex(XMFLOAT3(fx, -fy, +fz), XMFLOAT2(0.0f, 1.0f));
         pVertices[3] = CTexturedVertex(XMFLOAT3(fx, -fy, +fz), XMFLOAT2(0.0f, 1.0f));
         pVertices[4] = CTexturedVertex(XMFLOAT3(fx, +fy, +fz), XMFLOAT2(0.0f, 0.0f));
         pVertices[5] = CTexturedVertex(XMFLOAT3(fx, +fy, -fz), XMFLOAT2(1.0f, 0.0f));
      }
      else {
         pVertices[0] = CTexturedVertex(XMFLOAT3(fx, +fy, +fz), XMFLOAT2(1.0f, 0.0f));
         pVertices[1] = CTexturedVertex(XMFLOAT3(fx, -fy, +fz), XMFLOAT2(1.0f, 1.0f));
         pVertices[2] = CTexturedVertex(XMFLOAT3(fx, -fy, -fz), XMFLOAT2(0.0f, 1.0f));
         pVertices[3] = CTexturedVertex(XMFLOAT3(fx, -fy, -fz), XMFLOAT2(0.0f, 1.0f));
         ...
      }
   }
```

class CTexturedRectMesh : public CMesh { ... }



텍스쳐(Texture)

- 사각형 메쉬(Rectangle Mesh)

```
else if (fHeight == 0.0f) {
   if (fyPosition > 0.0f) {
      pVertices[0] = CTexturedVertex(XMFLOAT3(+fx, fy, -fz), XMFLOAT2(1.0f, 0.0f));
      pVertices[1] = CTexturedVertex(XMFLOAT3(+fx, fy, +fz), XMFLOAT2(1.0f, 1.0f));
      pVertices[2] = CTexturedVertex(XMFLOAT3(-fx, fy, +fz), XMFLOAT2(0.0f, 1.0f));
      pVertices[3] = CTexturedVertex(XMFLOAT3(-fx, fy, +fz), XMFLOAT2(0.0f, 1.0f));
      pVertices[4] = CTexturedVertex(XMFLOAT3(-fx, fy, -fz), XMFLOAT2(0.0f, 0.0f));
      pVertices[5] = CTexturedVertex(XMFLOAT3(+fx, fy, -fz), XMFLOAT2(1.0f, 0.0f));
   }
   else {
      pVertices[0] = CTexturedVertex(XMFLOAT3(+fx, fy, +fz), XMFLOAT2(1.0f, 0.0f));
      pVertices[1] = CTexturedVertex(XMFLOAT3(+fx, fy, -fz), XMFLOAT2(1.0f, 1.0f));
      ...
   }
}
else if (fDepth == 0.0f) {
   if (fzPosition > 0.0f) {
      pVertices[0] = CTexturedVertex(XMFLOAT3(+fx, +fy, fz), XMFLOAT2(1.0f, 0.0f));
      pVertices[1] = CTexturedVertex(XMFLOAT3(+fx, -fy, fz), XMFLOAT2(1.0f, 1.0f));
      ...
   }
   else {
      pVertices[0] = CTexturedVertex(XMFLOAT3(-fx, +fy, fz), XMFLOAT2(1.0f, 0.0f));
      ...
   }
}
```

텍스쳐(Texture)

- 사각형 메쉬(Rectangle Mesh)

```
    else if (fHeight == 0.0f) {
        if (fyPosition > 0.0f) {
            pVertices[0] = CTexturedVertex(XMFLOAT3(+fx, fy, -fz), XMFLOAT2(1.0f, 0.0f));
            ...
        }
        else {
            pVertices[0] = CTexturedVertex(XMFLOAT3(+fx, fy, +fz), XMFLOAT2(1.0f, 0.0f));
            ...
        }
    }
    else if (fDepth == 0.0f) {
        if (fzPosition > 0.0f) {
            pVertices[0] = CTexturedVertex(XMFLOAT3(+fx, +fy, fz), XMFLOAT2(1.0f, 0.0f));
            ...
        }
        else {
            ...
        }
    }
    m_pd3dVertexBuffer = ::CreateBufferResource(pd3dDevice, pd3dCommandList, pVertices, m_nStride *
m_nVertices, D3D12_HEAP_TYPE_DEFAULT, D3D12_RESOURCE_STATE_VERTEX_AND_CONSTANT_BUFFER,
&m_pd3dVertexUploadBuffer);
    m_d3dVertexBufferView.BufferLocation = m_pd3dVertexBuffer->GetGPUVirtualAddress();
    m_d3dVertexBufferView.StrideInBytes = m_nStride;
    m_d3dVertexBufferView.SizeInBytes = m_nStride * m_nVertices;
}
```

텍스쳐(Texture)

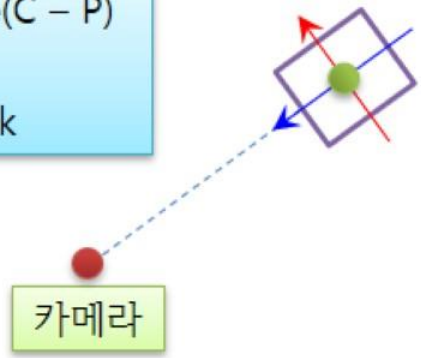
- 빌보드(Billboard)

```
void CBillboardObject::Animate(float fTimeElapsed, CCamera *pCamera)
{
    XMFLOAT3 xmf3CameraPosition = pCamera->GetPosition();
    SetLookAt(xmf3CameraPosition);
}
```

```
void CBillboardObject::SetLookAt(XMFLOAT3& xmf3Target)
{
    XMFLOAT3 xmf3Up(0.0f, 1.0f, 0.0f);
    XMFLOAT3 xmf3Position(m_xmf4x4World._41, m_xmf4x4World._42, m_xmf4x4World._43);
    XMFLOAT3 xmf3Look = Vector3::Subtract(xmf3Target, xmf3Position, true);
    XMFLOAT3 xmf3Right = Vector3::CrossProduct(xmf3Up, xmf3Look, true);
    m_xmf4x4World._11 = xmf3Right.x; m_xmf4x4World._12 = xmf3Right.y; m_xmf4x4World._13 = xmf3Right.z;
    m_xmf4x4World._21 = xmf3Up.x; m_xmf4x4World._22 = xmf3Up.y; m_xmf4x4World._23 = xmf3Up.z;
    m_xmf4x4World._31 = xmf3Look.x; m_xmf4x4World._32 = xmf3Look.y; m_xmf4x4World._33 = xmf3Look.z;
}
```

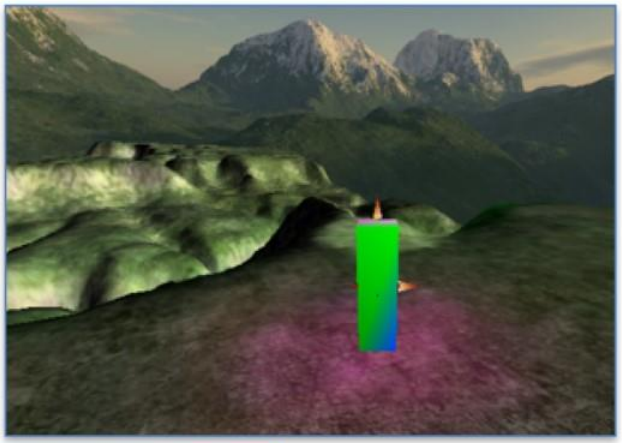
Right.x	Right.y	Right.z	0
Up.x	Up.y	Up.z	0
Look.x	Look.y	Look.z	0
Position.x	Position.y	Position.z	1

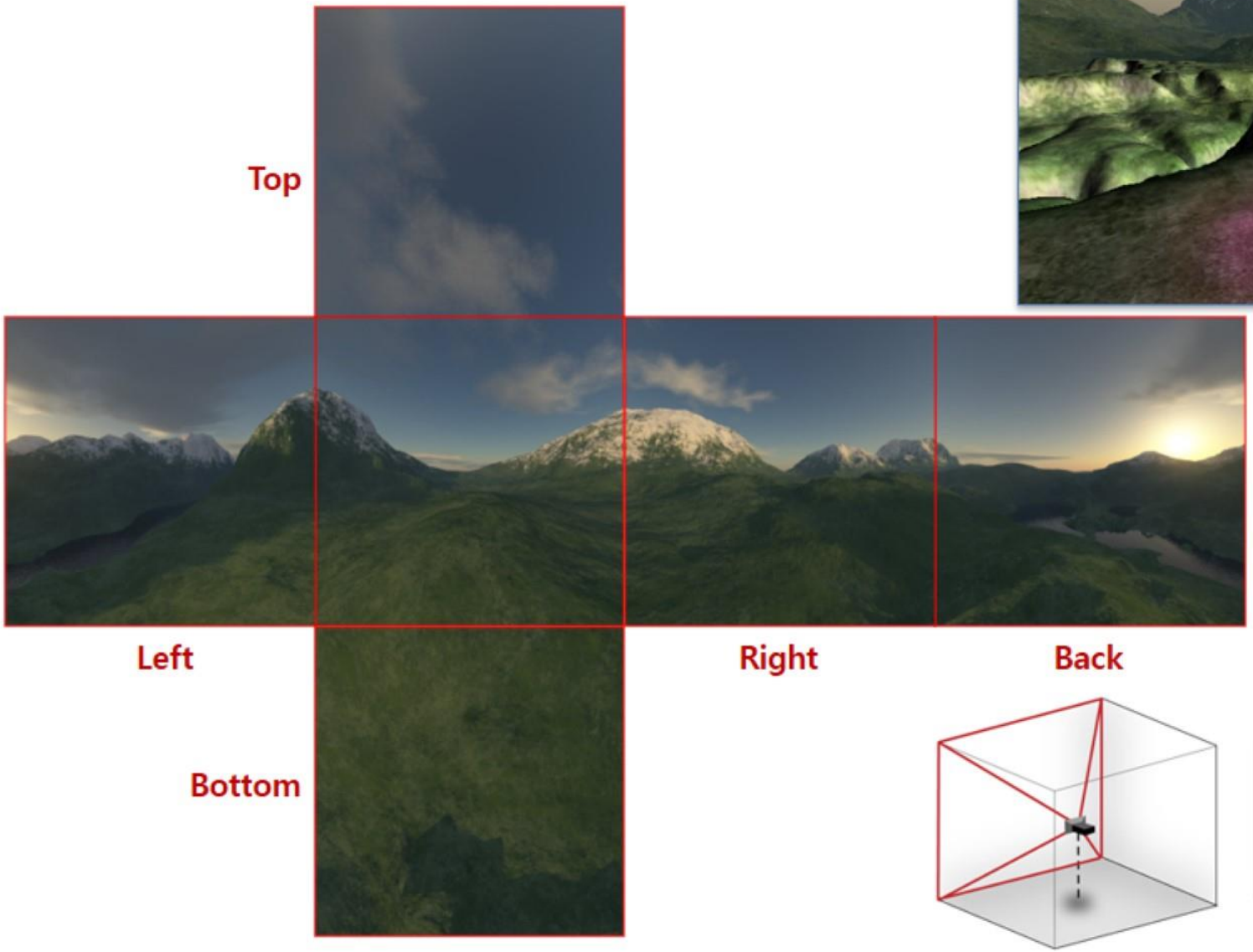
Look = normalize(C – P)
Up = (0, 1, 0)
Right = Up x Look

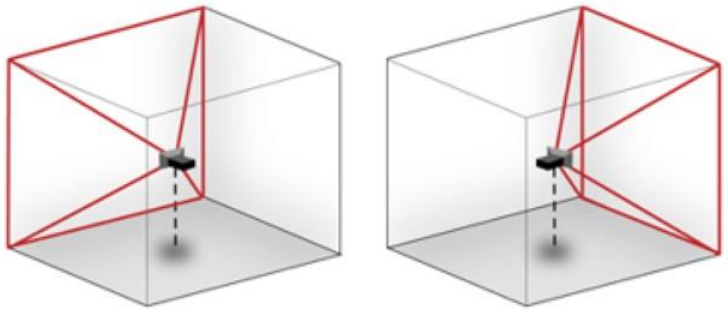


텍스쳐(Texture)

- 스카이 박스(Sky Box)







텍스쳐(Texture)

- 스카이 박스(Sky Box)

```
CSkyBox::CSkyBox(ID3D12Device *pd3dDevice, ID3D12GraphicsCommandList *pd3dCmdList) : CGameObject(6)
{
   m_ppMeshes[0] = new CTexturedRectMesh(pd3dDevice, pd3dCmdList, 20.0f, 20.0f, 0.0f, 0.0f, 0.0f, +10.0f);
   m_ppMeshes[1] = new CTexturedRectMesh(pd3dDevice, pd3dCmdList, 20.0f, 20.0f, 0.0f, 0.0f, 0.0f, -10.0f);
   m_ppMeshes[2] = new CTexturedRectMesh(pd3dDevice, pd3dCmdList, 0.0f, 20.0f, 20.0f, -10.0f, 0.0f, 0.0f);
   m_ppMeshes[3] = new CTexturedRectMesh(pd3dDevice, pd3dCmdList, 0.0f, 20.0f, 20.0f, +10.0f, 0.0f, 0.0f);
   m_ppMeshes[4] = new CTexturedRectMesh(pd3dDevice, pd3dCmdList, 20.0f, 0.0f, 20.0f, 0.0f, +10.0f, 0.0f);
   m_ppMeshes[5] = new CTexturedRectMesh(pd3dDevice, pd3dCmdList, 20.0f, 0.0f, 20.0f, 0.0f, -10.0f, 0.0f);

   m_nUploadBuffers = 6;
   m_ppd3dUploadBuffers = new ID3D12Resource*[m_nUploadBuffers];

   ID3D12Resource *pd3dTextures[6];
   pd3dTextures[0] = ::CreateTextureResourceFromFile(pd3dDevice, pd3dCmdList, L"SkyBox_Front_0.dds", ...);
   pd3dTextures[1] = ::CreateTextureResourceFromFile(pd3dDevice, pd3dCmdList, L"SkyBox_Back_0.dds", ...);
   pd3dTextures[2] = ::CreateTextureResourceFromFile(pd3dDevice, pd3dCmdList, L"SkyBox_Left_0.dds", ...);
   pd3dTextures[3] = ::CreateTextureResourceFromFile(pd3dDevice, pd3dCmdList, L"SkyBox_Right_0.dds", ...);
   pd3dTextures[4] = ::CreateTextureResourceFromFile(pd3dDevice, pd3dCmdList, L"SkyBox_Top_0.dds", ...);
   pd3dTextures[5] = ::CreateTextureResourceFromFile(pd3dDevice, pd3dCmdList, L"SkyBox_Bottom_0.dds", ...);

   CSkyBoxShader *pSkyBoxShader = new CSkyBoxShader();
   pSkyBoxShader->CreateShader(pd3dDevice, pd3dGraphicsRootSignature);
   pSkyBoxShader->CreateShaderVariables(pd3dDevice, pd3dCommandList);
   pSkyBoxShader->CreateCbvAndSrvDescriptorHeaps(pd3dDevice, pd3dCommandList, 1, 6);
   pSkyBoxShader->CreateConstantBufferViews(pd3dDevice, pd3dCommandList, 1, pSkyBoxShader-
>GetGameObjectsConstantBuffer(), 256);
```

```
class CSkyBox : public CGameObject { ... }
```

텍스쳐(Texture)

- 스카이 박스(Sky Box)

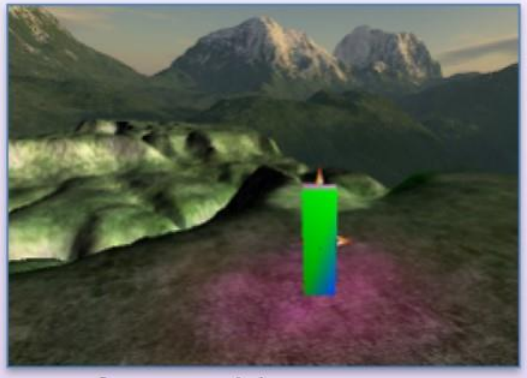
```
CTexture *pSkyBoxTexture = pSkyBoxShader->CreateShaderResourceViews(pd3dDevice,
pd3dCommandList, 6, ppd3dSkyBoxTextures, 2, false);

CMaterial *pSkyBoxMaterial = new CMaterial();
pSkyBoxMaterial->SetTexture(pSkyBoxTexture);

SetMaterial(pSkyBoxMaterial);

SetCbvGPUDescriptorHandle(pSkyBoxShader->GetGPUCbvDescriptorStartHandle());

SetShader(pSkyBoxShader);
}
```

```
Texture2D gtxtTexture : register(t0);
SamplerState gWrapSamplerState : register(s0);
```

```
ID3D12RootSignature *CScene::CreateGraphicsRootSignature(ID3D12Device *pd3dDevice)
{
    ...
    pd3dDescriptorRanges[1].RangeType = D3D12_DESCRIPTOR_RANGE_TYPE_SRV;
    pd3dDescriptorRanges[1].NumDescriptors = 1; //Texture2D gtxtTexture : register(t0);
    pd3dDescriptorRanges[1].BaseShaderRegister = 0; //t0
    ...
    pd3dRootParameters[2].ParameterType = D3D12_ROOT_PARAMETER_TYPE_DESCRIPTOR_TABLE;
    pd3dRootParameters[2].DescriptorTable.NumDescriptorRanges = 1;
    pd3dRootParameters[2].DescriptorTable.pDescriptorRanges = &pd3dDescriptorRanges[1];
    pd3dRootParameters[2].ShaderVisibility = D3D12_SHADER_VISIBILITY_PIXEL;
    ...
}
```

텍스쳐(Texture)

- 스카이 박스(Sky Box)



```
void CSkyBox::Render(ID3D12GraphicsCommandList *pd3dCommandList, CCamera *pCamera)
{
   XMFLOAT3 xmf3CameraPos = pCamera->GetPosition();
   SetPosition(xmf3CameraPos.x, xmf3CameraPos.y, xmf3CameraPos.z);

   OnPrepareRender();

   if (m_pMaterial && m_pMaterial->m_pShader)
   {
      m_pMaterial->m_pShader->Render(pd3dCommandList, pCamera);
      m_pMaterial->m_pShader->UpdateShaderVariable(pd3dCommandList, &m_xmf4x4World);
   }

   pd3dCommandList->SetGraphicsRootDescriptorTable(1, m_d3dCbvGPUDescriptorHandle);

   if (m_ppMeshes)
   {
      for (int i = 0; i < m_nMeshes; i++)
      {
         if (m_pMaterial && m_pMaterial->m_pTexture)
         {
            m_pMaterial->m_pTexture->UpdateShaderVariable(pd3dCommandList, i);
         }
         if (m_ppMeshes[i]) m_ppMeshes[i]->Render(pd3dCommandList);
      }
   }
}
```

텍스쳐(Texture)

- 스카이 박스(Sky Box)

```
D3D12_DEPTH_STENCIL_DESC CSkyBoxShader::CreateDepthStencilState()
{
   D3D12_DEPTH_STENCIL_DESC d3dDepthStencilDesc;
   d3dDepthStencilDesc.DepthEnable = FALSE;
   d3dDepthStencilDesc.DepthWriteMask = D3D12_DEPTH_WRITE_MASK_ZERO;
   d3dDepthStencilDesc.DepthFunc = D3D12_COMPARISON_FUNC_NEVER;
   d3dDepthStencilDesc.StencilEnable = FALSE;
   d3dDepthStencilDesc.StencilReadMask = d3dDepthStencilDesc.StencilWriteMask = 0xff;
   d3dDepthStencilDesc.FrontFace.StencilFailOp = D3D12_STENCIL_OP_KEEP;
   d3dDepthStencilDesc.FrontFace.StencilDepthFailOp = D3D12_STENCIL_OP_INCR;
   d3dDepthStencilDesc.FrontFace.StencilPassOp = D3D12_STENCIL_OP_KEEP;
   d3dDepthStencilDesc.FrontFace.StencilFunc = D3D12_COMPARISON_FUNC_ALWAYS;
   d3dDepthStencilDesc.BackFace.StencilFailOp = D3D12_STENCIL_OP_KEEP;
   d3dDepthStencilDesc.BackFace.StencilDepthFailOp = D3D12_STENCIL_OP_DECR;
   d3dDepthStencilDesc.BackFace.StencilPassOp = D3D12_STENCIL_OP_KEEP;
   d3dDepthStencilDesc.BackFace.StencilFunc = D3D12_COMPARISON_FUNC_ALWAYS;
   return(d3dDepthStencilDesc);
}
```

```
class CSkyBoxShader : public CTexturedShader { ... }
```

```
SamplerState gClampSamplerState : register(s1);
```

```
float4 PSSkyBox(VS_TEXTURED_OUTPUT input) : SV_TARGET {
   return(gtxtTexture.Sample(gClampSamplerState, input.texCoord));
}
```

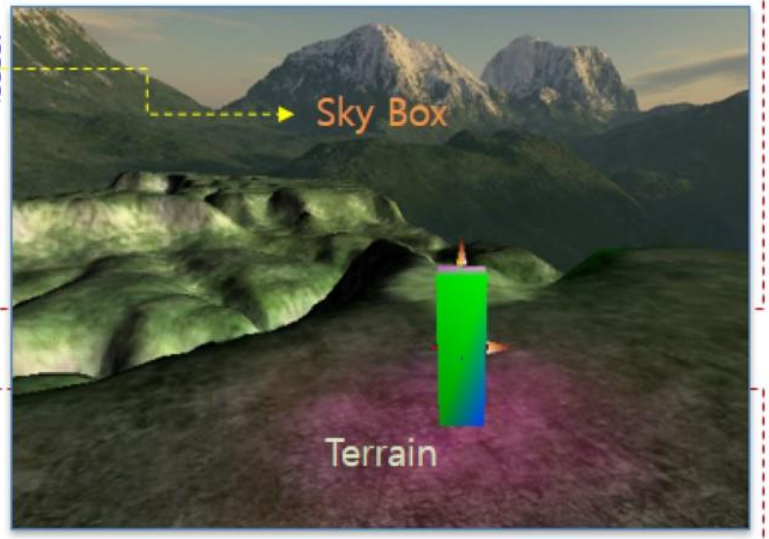
```
pd3dSamplerDescs[1].Filter = D3D12_FILTER_MIN_MAG_MIP_LINEAR;
pd3dSamplerDescs[1].AddressU = D3D12_TEXTURE_ADDRESS_MODE_CLAMP;
pd3dSamplerDescs[1].AddressV = D3D12_TEXTURE_ADDRESS_MODE_CLAMP;
pd3dSamplerDescs[1].AddressW = D3D12_TEXTURE_ADDRESS_MODE_CLAMP;
```

텍스쳐(Texture)

- **스카이 박스(Sky Box)**
  - 오버드로우(Overdraw)의 문제

```
void CScene::Render(...)
{
   ...
   m_pSkyBox->Render(...);
   m_pTerrain->Render(...);
   for (int i = 0; i < m_nShaders; i++) m_ppShaders[i]->Render(...);
}
```

스카이 박스는 가장 멀리 있는 객체들의 그림




```
void CScene::Render(...)
{
   ...
   m_pTerrain->Render(...);
   for (int i = 0; i < m_nShaders; i++) m_ppShaders[i]->Render(...);
   m_pSkyBox->Render(...);
}
```

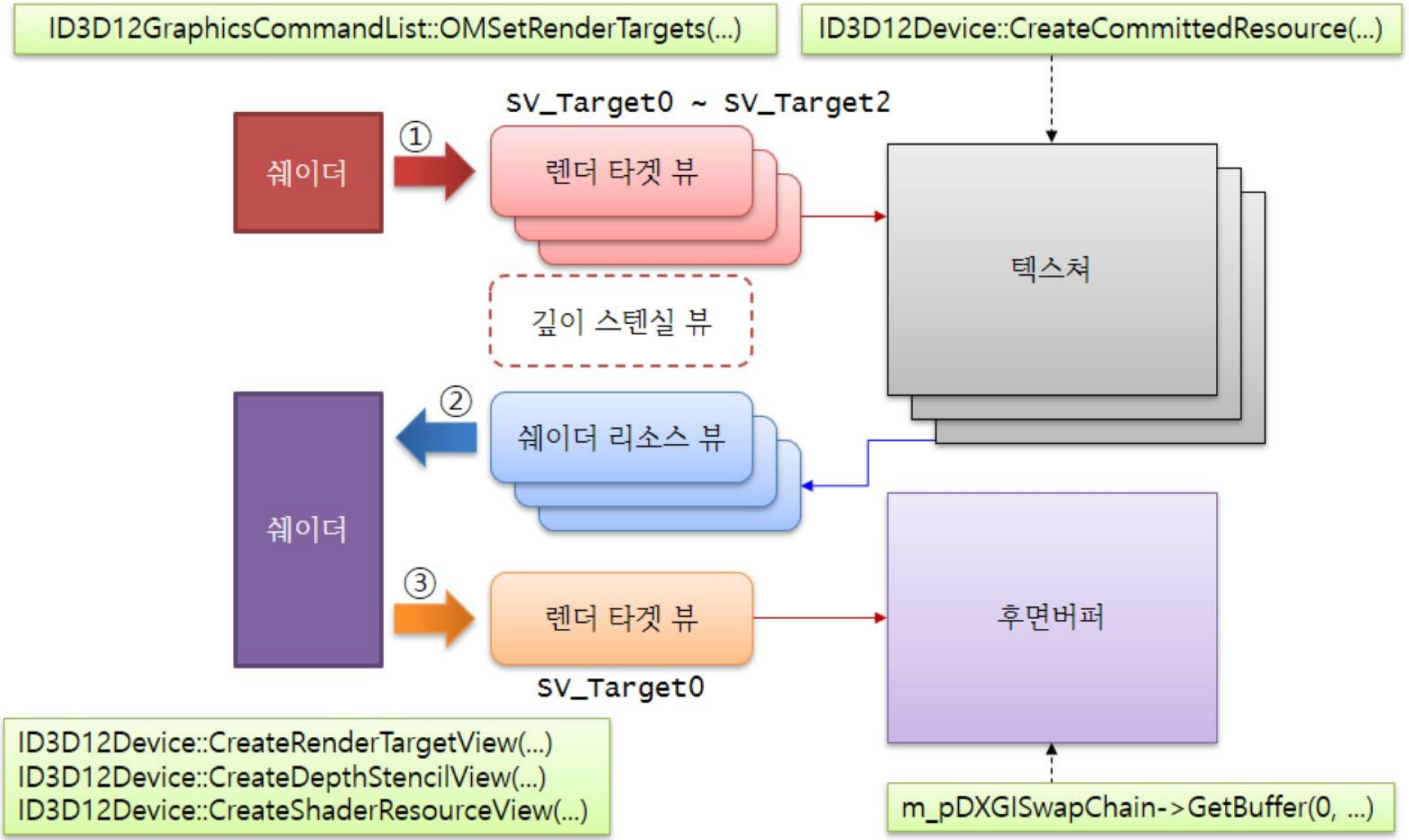
```
output.position = mul(float4(input.position, 1.0f), WVP).xyww;
```

```
struct OUTPUT
{
   float4 position : SV_POSITION;
   ...
}
```

```
d3dDepthStencilDesc.DepthEnable = TRUE; //Depth = 1.0f
d3dDepthStencilDesc.DepthWriteMask = D3D12_DEPTH_WRITE_MASK_ZERO;
//d3dDepthStencilDesc.DepthWriteMask = D3D12_DEPTH_WRITE_MASK_ALL;
d3dDepthStencilDesc.DepthFunc = D3D12_COMPARISON_FUNC_LESS_EQUAL;
```

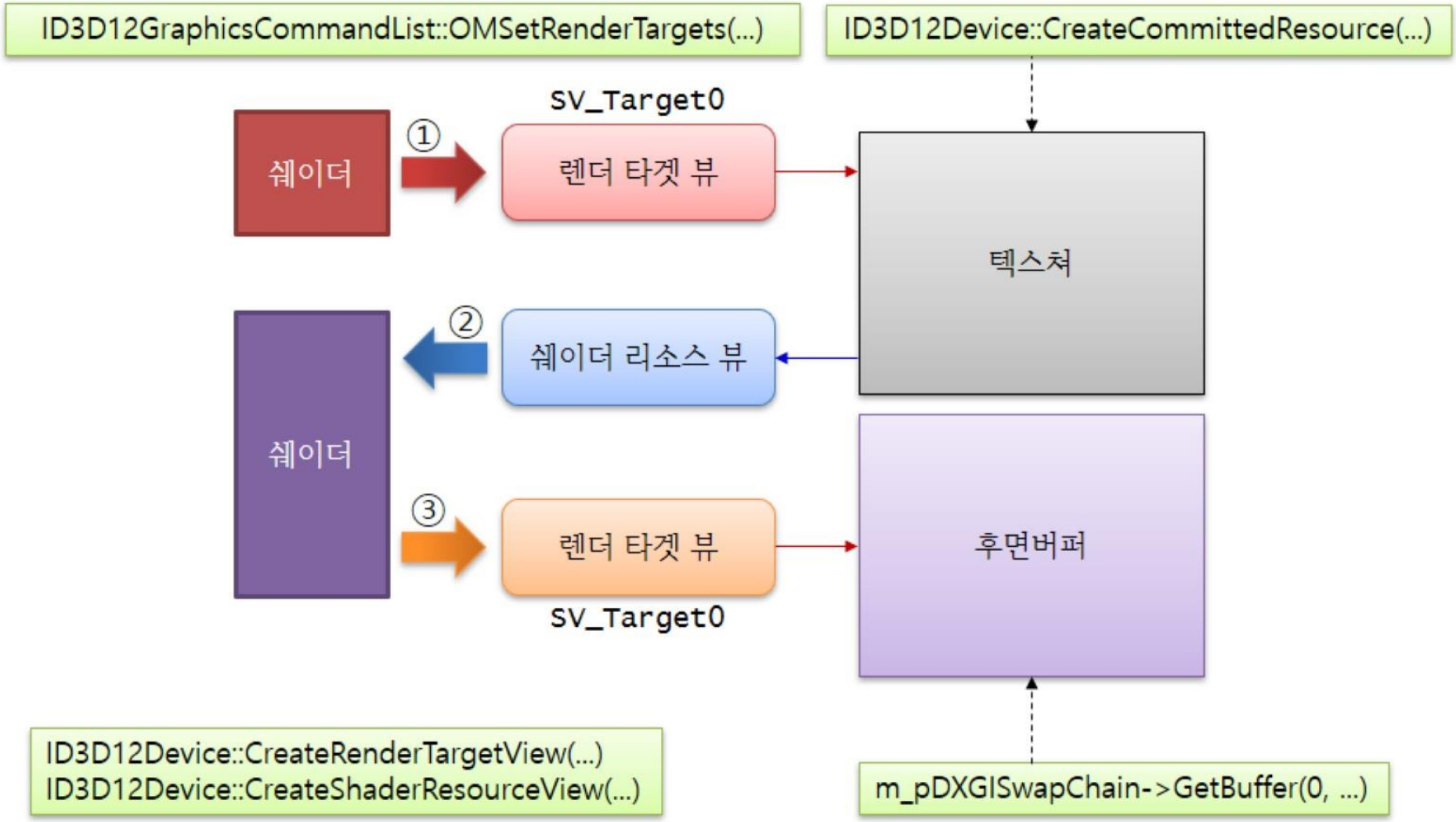
텍스쳐(Texture)

- **리소스 뷰(Resource Views)**
  - 하나의 리소스를 여러가지 뷰를 통하여 접근할 수 있음



텍스쳐(Texture)

- **후처리(PostProcessing)**
  - 렌더링한 결과 이미지에 대한 색상의 변화
  - 픽셀 쉐이더에서 현재 렌더링하는 픽셀의 인접 픽셀의 색상을 알 수 있는가?

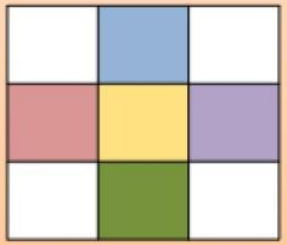


텍스쳐(Texture)

- **후처리(PostProcessing)**
  - 흑백처리, 블러링(Blurring), 영상 필터(Image Filter), ...
    ① 텍스쳐로 렌더링
    ② 텍스쳐를 리소스로 연결
    ③ 사각형을 렌더링할 때 후처리

```
SamplerState gssSamplerState : register(s0);
Texture2D gtxtTexture : register(t0);
```

```
float4 PSTextureToScreen(VS_TEXTURED_OUTPUT input) : SV_Target
{
   float4 cColor = float4(0.0f, 0.0f, 0.0f, 1.0f);
   uint nWidth, nHeight, nMipLevels;
   gtxtTexture.GetDimensions(0, nWidth, nHeight, nMipLevels);
   //cColor = gtxtTexture.Load(int3(input.uv.x * nWidth, input.uv.y * nHeight, 0)); //uvm
   int2 loc2 = int2(input.uv.x * nWidth, input.uv.y * nHeight); //픽셀 좌표
   //int2 loc2 = input.position.xy; //픽셀 좌표
   if ((loc2.x <= 0) || (loc2.x >= nWidth-1) || (loc2.y <= 0) || (loc2.y >= nHeight - 1))
      cColor = gtxtTexture[loc2];
   else
   {
      float4 crl = gtxtTexture[int2(loc2.x+1, loc2.y)] - gtxtTexture[int2(loc2.x-1, loc2.y)];
      float4 ctb = gtxtTexture[int2(loc2.x, loc2.y+1)] - gtxtTexture[int2(loc2.x, loc2.y-1)];
      cColor = (gtxtTexture[loc2] + crl + ctb) * 0.2f;
   }
   return(cColor);
}
```



```
cColor = gtxtTexture[loc2];
cColor += (gtxtTexture[int2(loc2.x+1, loc2.y)] + gtxtTexture[int2(loc2.x-1, loc2.y)]);
cColor += (gtxtTexture[int2(loc2.x, loc2.y+1)] + gtxtTexture[int2(loc2.x, loc2.y-1)]);
return(cColor * 0.2f);
```

텍스쳐(Texture)

- **버퍼 리소스(Buffer Resource)**

```
ID3D12Buffer *m_pd3dTriCenterBuffer;
ID3D12ShaderResourceView *m_pd3dsrvTriCenter;
```

```
D3D12_BUFFER_DESC d3dBufferDesc;
::ZeroMemory(&d3dBufferDesc, sizeof(D3D12_BUFFER_DESC));
d3dBufferDesc.Usage = D3D12_USAGE_DEFAULT;
d3dBufferDesc.ByteWidth = 2 * m_nFaces * sizeof(xmf3ECTOR4);
d3dBufferDesc.BindFlags = D3D12_BIND_SHADER_RESOURCE;
d3dBufferDesc.CPUAccessFlags = 0;
xmf3ECTOR4 *pd3dvData = new xmf3ECTOR4[2 * m_nFaces];
...
D3D12_SUBRESOURCE_DATA d3dBufferData;
::ZeroMemory(&d3dBufferData, sizeof(D3D12_SUBRESOURCE_DATA));
d3dBufferData.pSysMem = pd3dvData;
pd3dDevice->CreateBuffer(&d3dBufferDesc, &d3dBufferData, &m_pd3dTriCenterBuffer);

D3D12_SHADER_RESOURCE_VIEW_DESC d3dsrvDesc;
::ZeroMemory(&d3dsrvDesc, sizeof(D3D12_SHADER_RESOURCE_VIEW_DESC));
d3dsrvDesc.Format = DXGI_FORMAT_R32G32B32A32_FLOAT;
d3dsrvDesc.ViewDimension = D3D12_SRV_DIMENSION_BUFFER;
d3dsrvDesc.Buffer.FirstElement = 0;
d3dsrvDesc.Buffer.NumElements = 2 * m_nFaces;
pd3dDevice->CreateShaderResourceView(m_pd3dTriCenterBuffer, &d3dsrvDesc, &m_pd3dsrvTriCenter);
pd3dDeviceContext->VSSetShaderResources(3, 1, &m_pd3dsrvTriCenter);
```

```
int m_nFaces;
```

```
typedef struct D3D12_BUFFER_SRV {
    UINT FirstElement;
    UINT NumElements;
} D3D12_BUFFER_SRV;
```

```
Buffer<float4> gTriCenterBuffer : register(t3);

VS_PIPE_IN output;

output.pos = gTriCenterBuffer.Load(nFace * 2);
output.dir = gTriCenterBuffer.Load(nFace * 2 + 1);
```

```
struct VS_PIPE_IN {
    float3 pos : POSITION;
    float3 dir : DIRECTION;
};
```