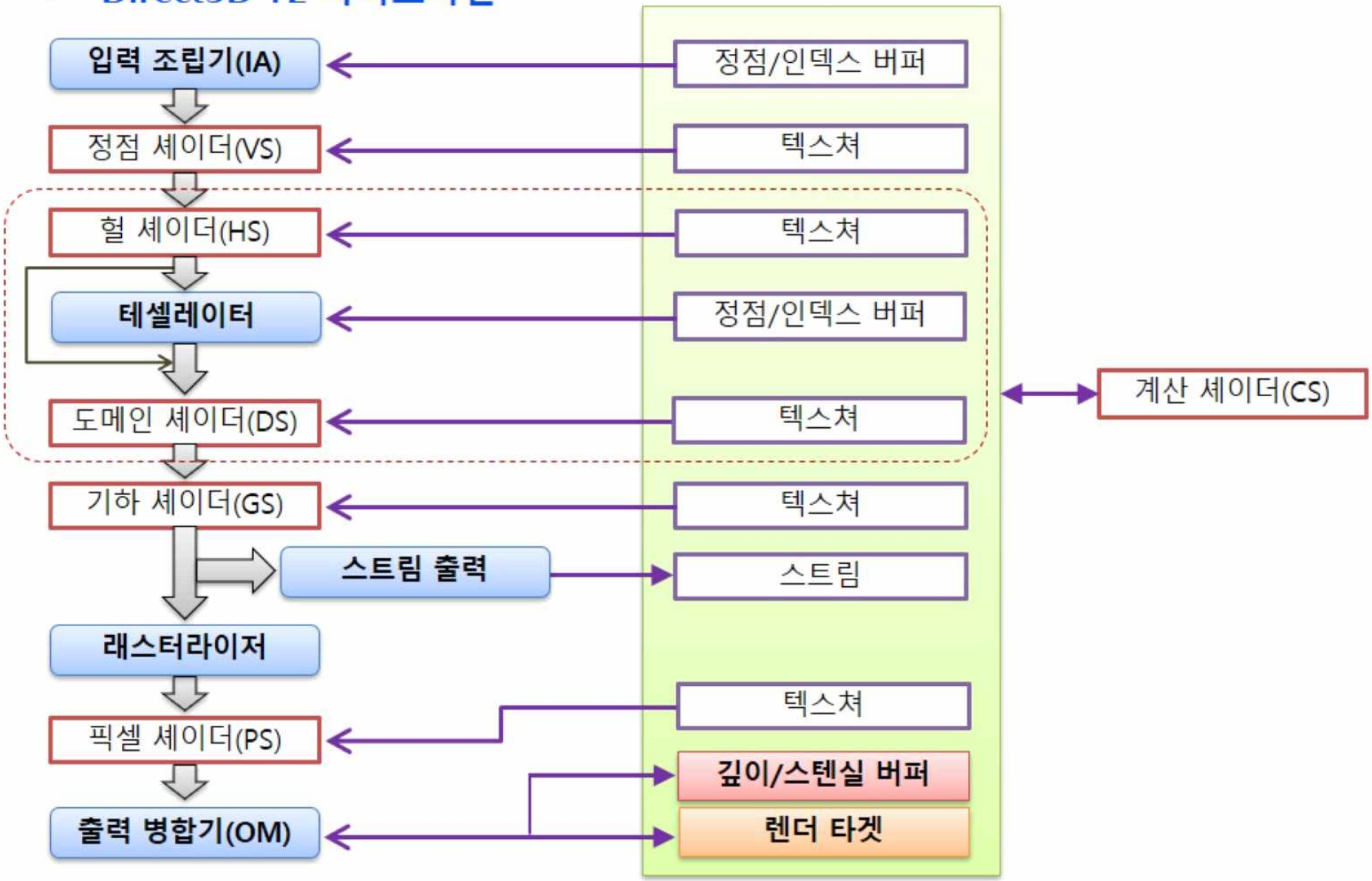
Game Programming with DirectX

Direct3D Graphics Pipeline (Shader Samples)

- **Bump Mapping**
- **Normal Mapping**
- **Displacement Mapping**
- Parallax Mapping

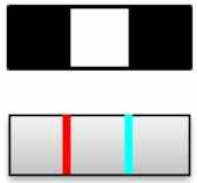
Direct3D 파이프라인

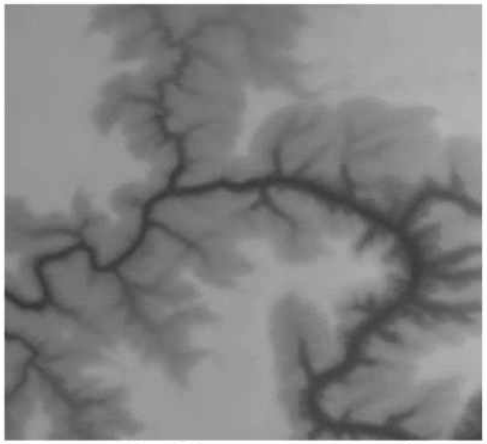
- Direct3D 12 파이프라인



법선 매핑(Normal Mapping)

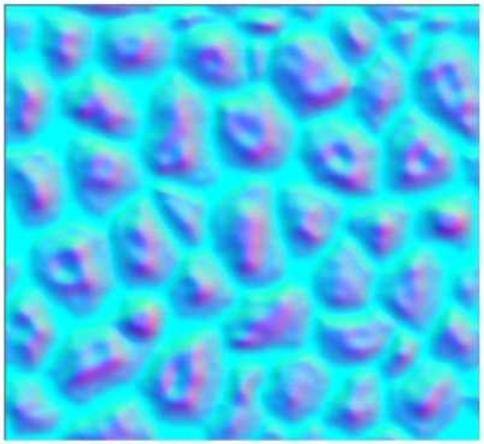
- 매핑(Mapping)
  높이 맵(Height Map): 높은 곳은 흰색, 낮은 곳은 검은색
  법선 맵(Normal Map): 법선 벡터를 색상(rgb)으로 표현
  변위 맵(Displacement Map), …



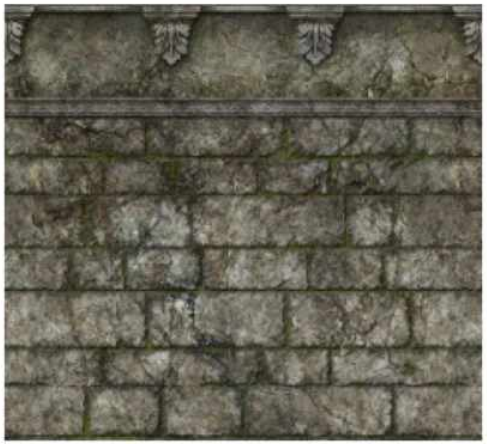


높이 맵(Height Map)

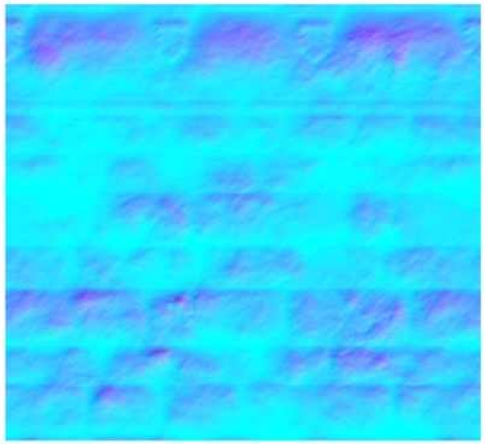
텍스쳐(Texture)



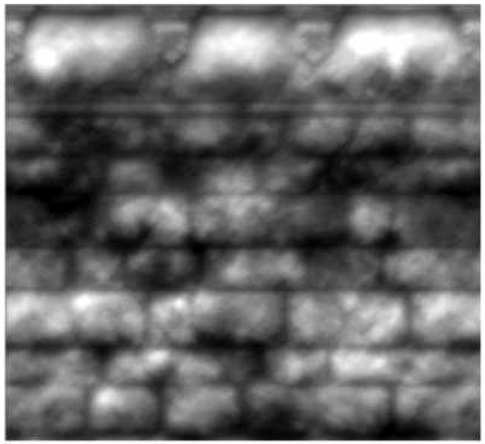
법선 맵(Normal Map)



텍스쳐(Texture)



법선 맵(Normal Map)

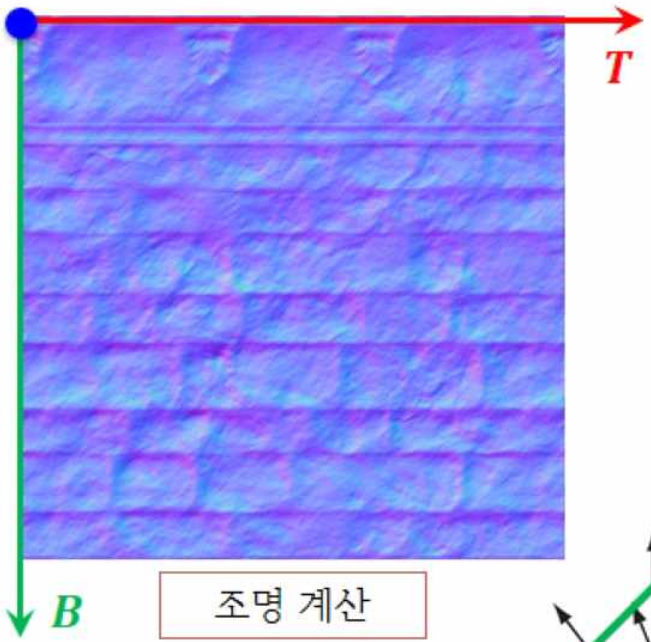


변위 맵(Displacement Map)

법선 매핑(Normal Mapping)

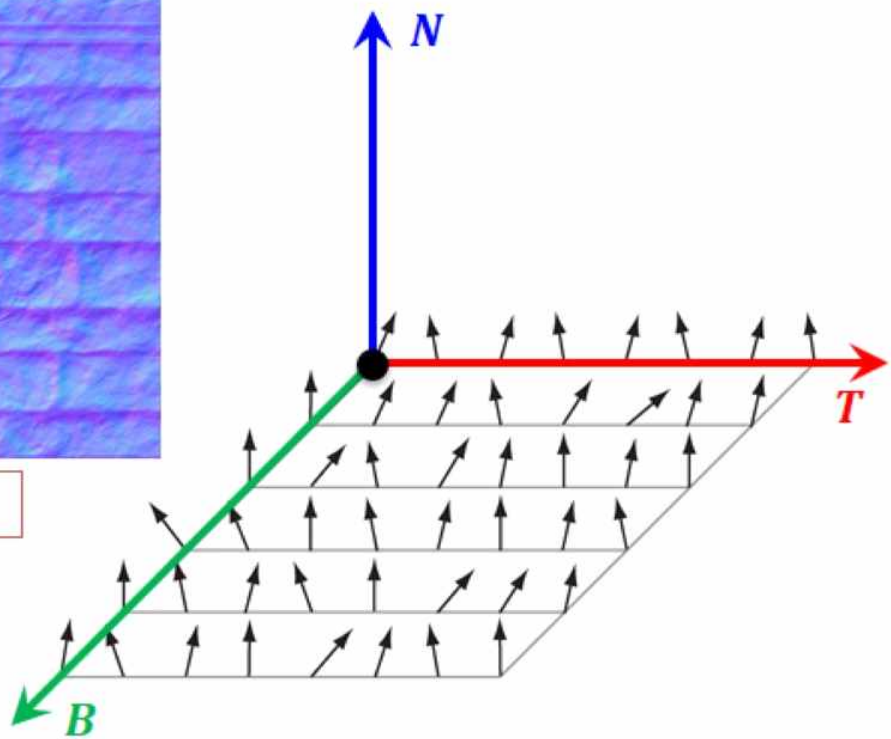
- **법선 매핑(Normal Mapping)**
  - 법선 맵(Normal Map)
    텍스쳐의 각 텍셀이 색상(rgb)이 아닌 법선 벡터(xyz)를 나타냄
    법선 벡터는 텍스쳐 좌표계(TBN)로 표현됨
    T: 접선 벡터(Tangent Vector), x-축
    B: 종법(접)선 벡터(Bi-Normal Vector, Bi-Tangent), y-축
    N: 법선 벡터(Normal Vector), z-축

텍스쳐 매핑



조명 계산

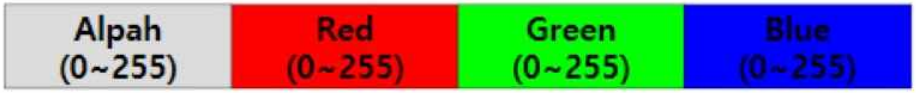
텍스쳐 좌표계(접선 좌표계)



Alpah (0~255)	Red (0~255)	Green (0~255)	Blue (0~255)
Alpah (0.0f~1.0f)	Red (0.0f~1.0f)	Green (0.0f~1.0f)	Blue (0.0f~1.0f)

법선 매핑(Normal Mapping)

- **법선 매핑(Normal Mapping)**
  - 법선 맵(Normal Map)
    법선 벡터를 색상(rgb: 예를 들어 8-비트 색상)으로 어떻게 표현할 수 있는가?
    - 단위 벡터(Unit Vector) (a, b, c) ⇨ 8-비트 색상 (r, g, b)
      $-1 \le a \le 1, -1 \le b \le 1, -1 \le c \le 1$
      $r = (0.5 * a + 0.5) * 255$
      $g = (0.5 * b + 0.5) * 255$
      $b = (0.5 * c + 0.5) * 255$

    - 8-비트 색상 (r, g, b) ⇨ 단위 벡터(Unit Vector) (a, b, c)
      $a = (2.0 * r / 255) - 1.0$
      $b = (2.0 * g / 255) - 1.0$
      $c = (2.0 * b / 255) - 1.0$

    ① 텍스쳐 샘플링 (r, g, b) ⇨ $0 \le r \le 1, 0 \le g \le 1, 0 \le b \le 1$

    ```
    Texture2D gtxtNormalMap;
    float3 vNormal = gtxtNormalMap.Sample(gSampler, input.uv);
    ```

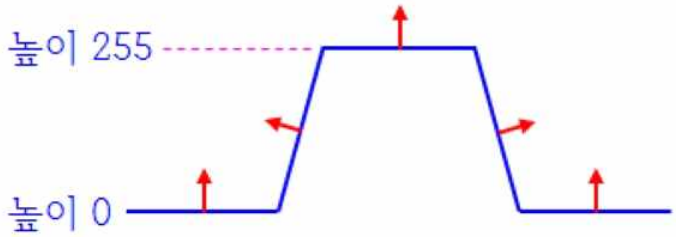
    ② [0, 1] ⇨ [-1, 1]
    $x \in [0, 1]$ ⇨ $2 * x - 1.0$

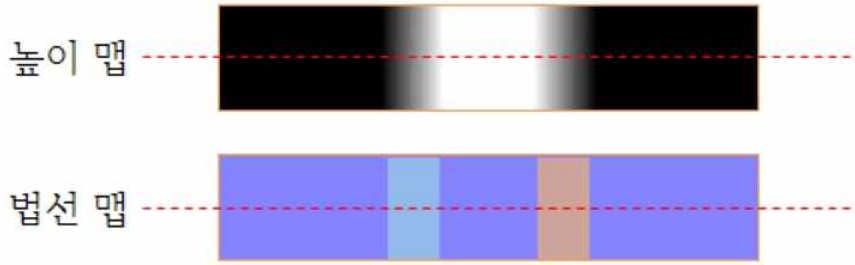
    ```
    Texture2D gtxtNormalMap;
    float3 vNormal = gtxtNormalMap.Sample(gSampler, input.uv);
    vNormal = vNormal * 2.0f – 1.0f;
    ```

    - 압축 텍스쳐 형식을 사용하는 경우 BC7(DXGI_FORMAT_BC7_UNORM)을 사용
      텍스쳐 파일을 BC7 형식으로 변환: "BC6HBC7EncoderDecoder11"

법선 매핑(Normal Mapping)

- **법선 매핑(Normal Mapping)**
  - 법선 맵(Normal Map) 생성하기
    높이 맵(Height Map): 표면의 높이 정보를 가진 텍스쳐





  - 높이 맵에서 법선 벡터 계산하기

I(i-1,j-1)	I(i,j-1)	I(i+1,j-1)
I(i-1,j)	I(i,j)	I(i+1,j)
I(i-1,j+1)	I(i,j+1)	I(i+1,j+1)

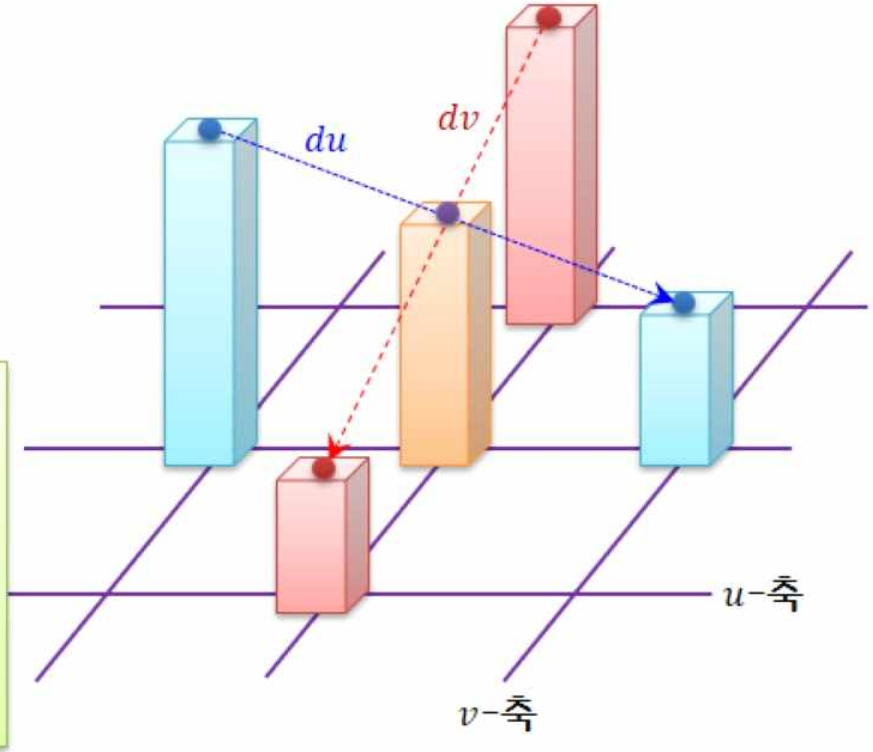
$$\frac{\partial I}{\partial u}=\frac{1}{2}\{(I(i+1,j)-I(i,j))+(I(i,j)-I(i-1,j))\}=\frac{1}{2}\{I(i+1,j)-I(i-1,j)\}$$

$$\frac{\partial I}{\partial v}=\frac{1}{2}\{(I(i,j+1)-I(i,j))+(I(i,j)-I(i,j-1))\}=\frac{1}{2}\{I(i,j+1)-I(i,j-1)\}$$

기울기 벡터: $du=\left(1,0,\frac{\partial I}{\partial u}\right), dv=\left(0,1,\frac{\partial I}{\partial v}\right)$

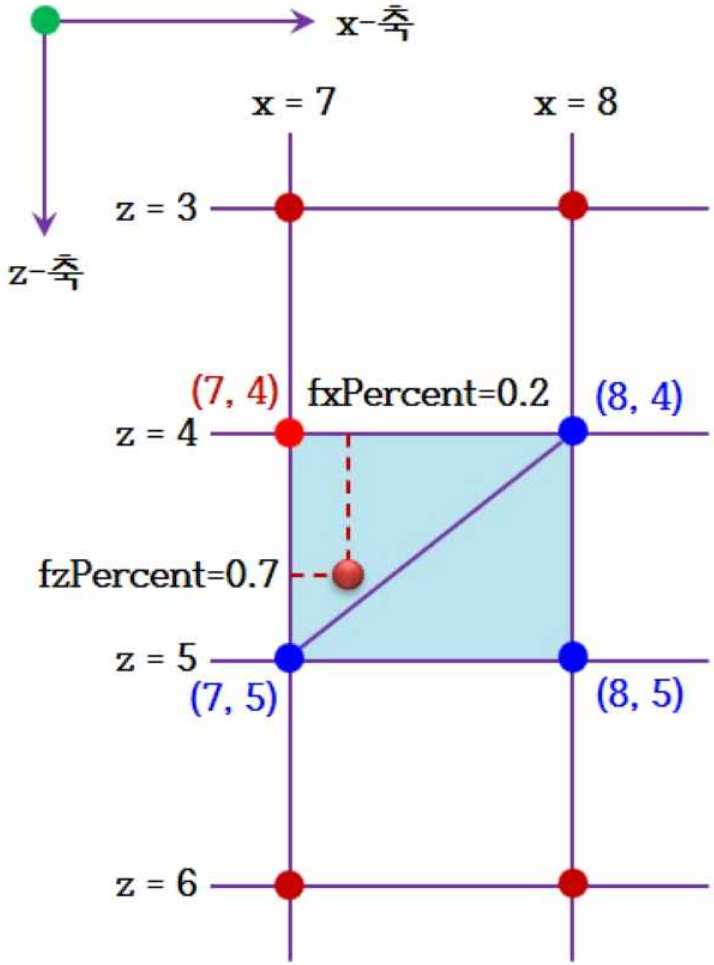
$$\boldsymbol{n}=\frac{du\times dv}{|du\times dv|}=\frac{\left(\frac{-1}{2}\{I(i+1,j)-I(i-1,j)\},\frac{1}{2}\{I(i,j+1)-I(i,j-1)\},1\right)}{\left|\left(\frac{-1}{2}\{I(i+1,j)-I(i-1,j)\},\frac{1}{2}\{I(i,j+1)-I(i,j-1)\},1\right)\right|}$$

$$\boldsymbol{n}=\frac{(-\{I(i+1,j)-I(i-1,j)\},\{I(i,j+1)-I(i,j-1)\},2)}{|(-\{I(i+1,j)-I(i-1,j)\},\{I(i,j+1)-I(i,j-1)\},2)|}$$

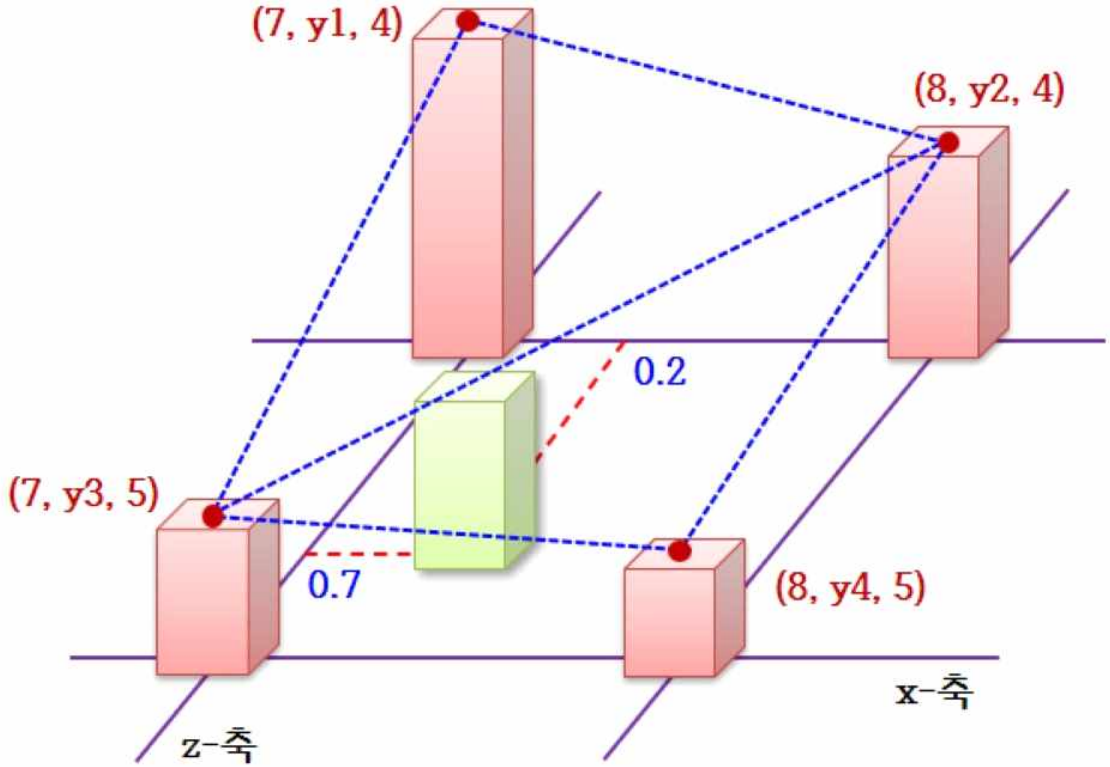


법선 매핑(Normal Mapping)

- **지형의 높이**
  - 높이 맵의 픽셀 값을 사용하여 지형의 높이 계산(보간)



지형 좌표 (x, z) = (72, 47), 확대 배율 = 10



법선 매핑(Normal Mapping)

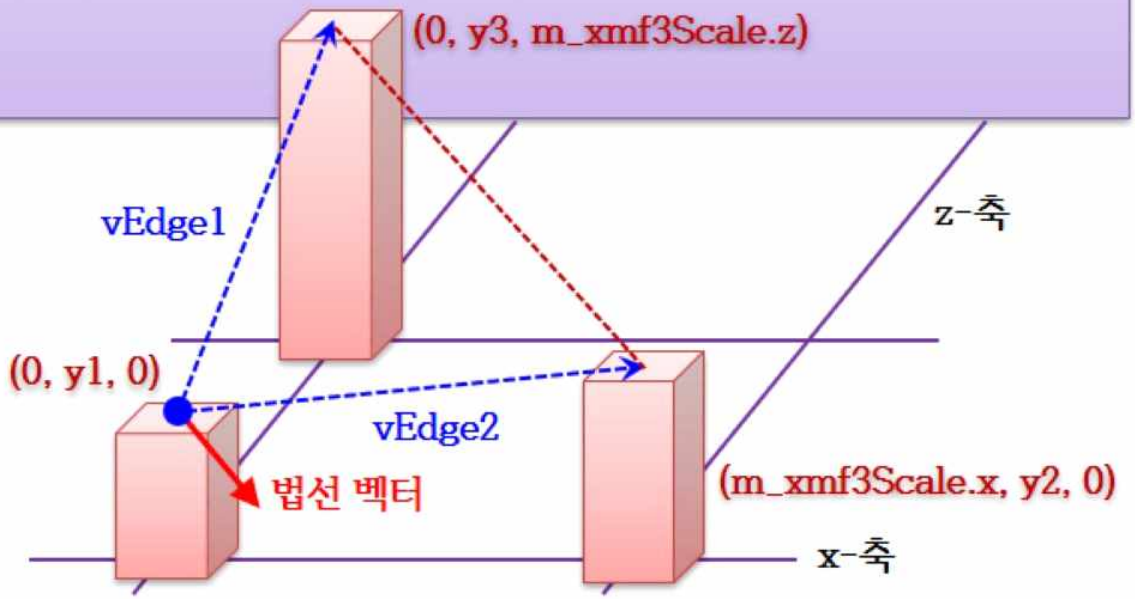
- 정점의 법선 벡터

```
XMFLOAT3 CHeightMapImage::GetHeightMapNormal(int x, int z)
{
   if ((x < 0.0f) || (z < 0.0f) || (x >= m_nWidth) || (z >= m_nLength)) return(XMFLOAT3(0.0f, 1.0f, 0.0f));

   int nHeightMapIndex = x + (z * m_nWidth);
   int xHeightMapAdd = (x < (m_nWidth - 1)) ? 1 : -1;
   int zHeightMapAdd = (z < (m_nLength - 1)) ? m_nWidth : -m_nWidth;
   float y1 = (float)m_pHeightMapPixels[nHeightMapIndex] * m_xmf3Scale.y;
   float y2 = (float)m_pHeightMapPixels[nHeightMapIndex + xHeightMapAdd] * m_xmf3Scale.y;
   float y3 = (float)m_pHeightMapPixels[nHeightMapIndex + zHeightMapAdd] * m_xmf3Scale.y;
   XMFLOAT3 xmf3Edge1 = XMFLOAT3(0.0f, y3 - y1, m_xmf3Scale.z);
   XMFLOAT3 xmf3Edge2 = XMFLOAT3(m_xmf3Scale.x, y2 - y1, 0.0f);
   XMFLOAT3 xmf3Normal = Vector3::CrossProduct(xmf3Edge1, xmf3Edge2, true);
   return(xmf3Normal);
}
```

v1 = (0, y1, 0)
v2 = (m_xmf3Scale.x, y2, 0)
v3 = (0, y3, m_xmf3Scale.z)

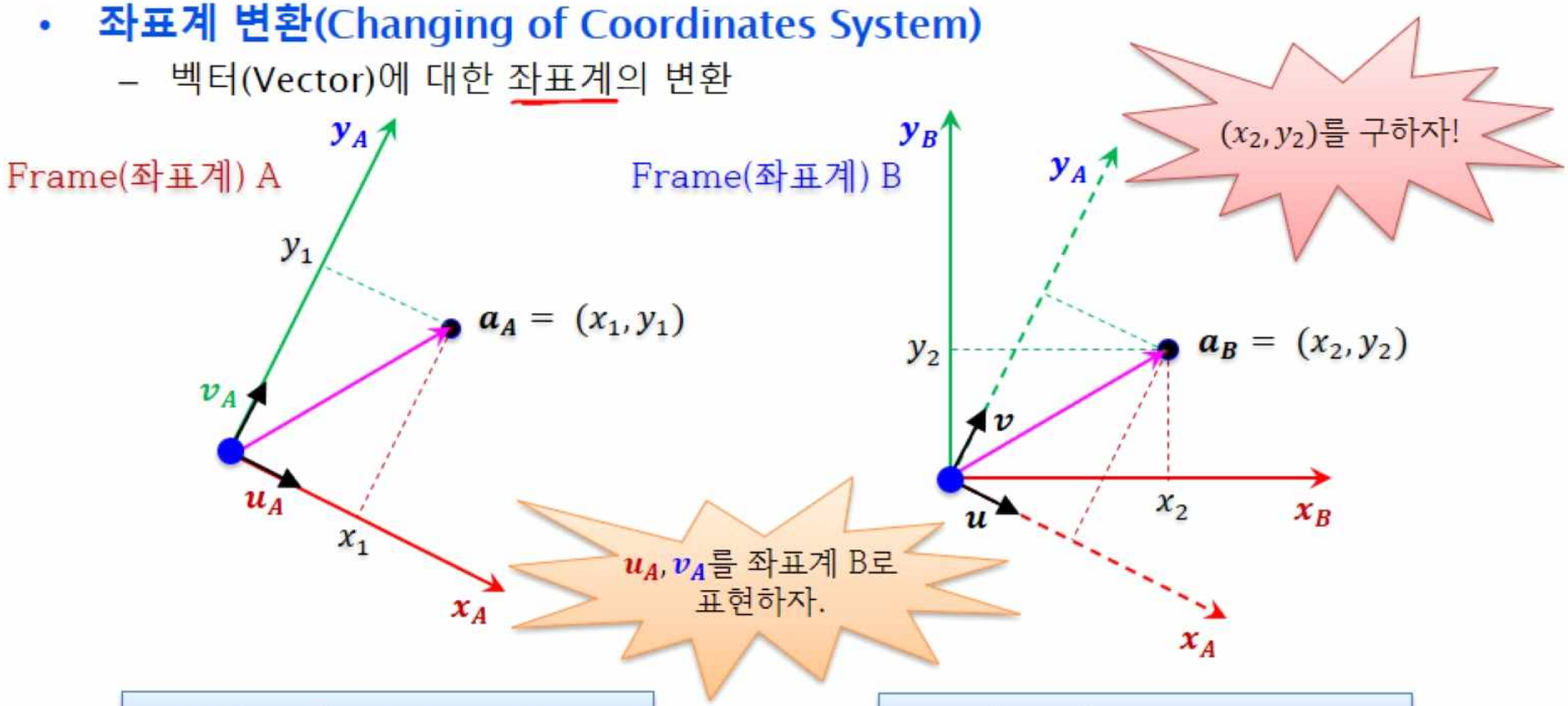
vEdge1 = v3 – v1
vEdge2 = v2 – v1

vEdge1과 vEdge2의 외적이 법선 벡터



법선 매핑(Normal Mapping)

- **좌표계 변환(Changing of Coordinates System)**
  - 벡터(Vector)에 대한 좌표계의 변환



$$a_A = (x_1, y_1) = x_1 u_A + y_1 v_A$$

$$a_B = (x_2, y_2) = x_2 u_B + y_2 v_B$$

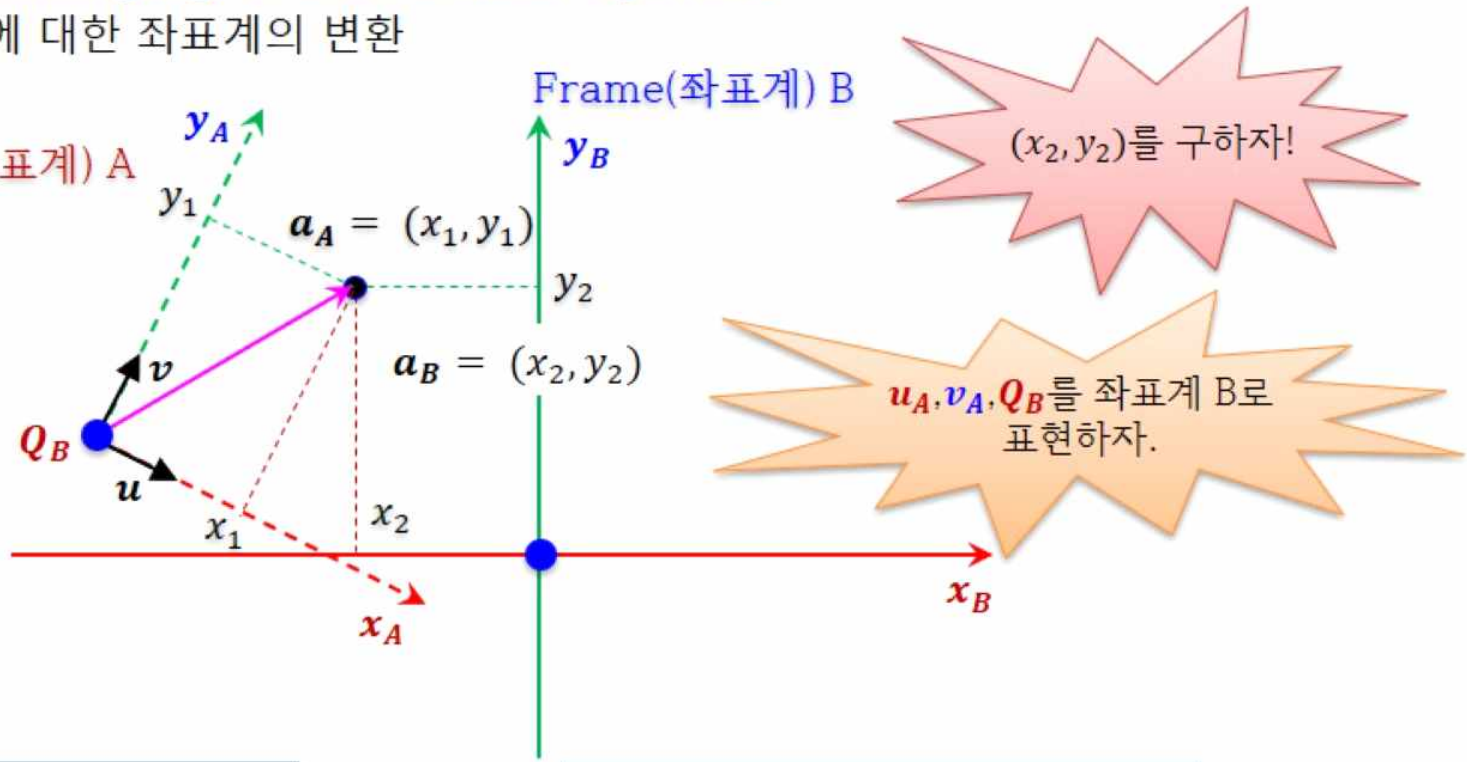
$$a_B = (x_2, y_2) = x_1 u + y_1 v$$

$$a_A = (x_1, y_1, z_1) = x_1 u_A + y_1 v_A + z_1 w_A$$

$$a_B = (x_2, y_2, z_2) = x_1 u + y_1 v + z_1 w$$

법선 매핑(Normal Mapping)

- **좌표계 변환(Changing of Coordinates System)**
  - 점(Point)에 대한 좌표계의 변환



$$a_A = (x_1, y_1) = x_1 u_A + y_1 v_A$$

$$a_B = (x_2, y_2) = x_2 u_B + y_2 v_B$$

$$a_B = (x_2, y_2) = x_1 u + y_1 v + Q_B$$

$$a_A = (x_1, y_1, z_1) = x_1 u_A + y_1 v_A + z_1 w_A$$

$$a_B = (x_2, y_2, z_2) = x_1 u + y_1 v + z_1 w + Q_B$$

법선 매핑(Normal Mapping)

- **좌표계 변환의 행렬 표현**
  - 좌표계 변환의 행렬 표현

벡터: $\boldsymbol{a_B} = (x_2, y_2, z_2) = x_1\boldsymbol{u} + y_1\boldsymbol{v} + z_1\boldsymbol{w}$

점: $\boldsymbol{a_B} = (x_2, y_2, z_2) = x_1\boldsymbol{u} + y_1\boldsymbol{v} + z_1\boldsymbol{w} + \boldsymbol{Q_B}$

동차좌표계: $\boldsymbol{a_B} = (x_2, y_2, z_2, w) = x_1\boldsymbol{u} + y_1\boldsymbol{v} + z_1\boldsymbol{w} + q\boldsymbol{Q_B}$

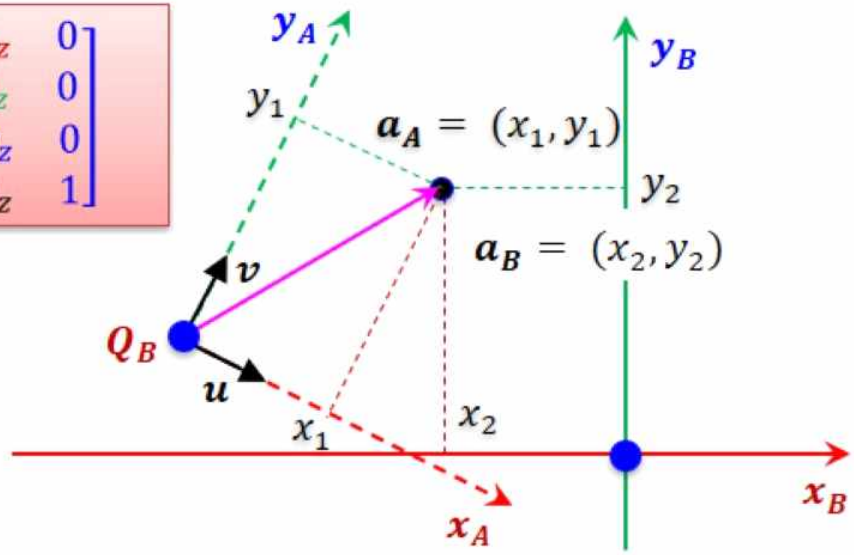
$$
\begin{aligned}
x_2 &= x_1u_x + y_1v_x + z_1w_x + qQ_x \\
y_2 &= x_1u_y + y_1v_y + z_1w_y + qQ_y \\
z_2 &= x_1u_z + y_1v_z + z_1w_z + qQ_z \\
q &= q
\end{aligned}
$$

점: $q = 1$
벡터: $q = 0$

$$
\boldsymbol{u} = (u_x, u_y, u_z, 0)
$$
$$
\boldsymbol{v} = (v_x, v_y, v_z, 0)
$$
$$
\boldsymbol{w} = (w_x, w_y, w_z, 0)
$$
$$
\boldsymbol{Q_B} = (Q_x, Q_y, Q_z, 1)
$$

$$
[x_2 \quad y_2 \quad z_2 \quad q] = [x_1 \quad y_1 \quad z_1 \quad q]
\begin{bmatrix}
u_x & u_y & u_z & 0 \\
v_x & v_y & v_z & 0 \\
w_x & w_y & w_z & 0 \\
Q_x & Q_y & Q_z & 1
\end{bmatrix}
$$

$\boldsymbol{u} = (u_x, u_y, u_z)$: 좌표계 A의 x-축을 좌표계 B로 표현
$\boldsymbol{v} = (v_x, v_y, v_z)$: 좌표계 A의 y-축을 좌표계 B로 표현
$\boldsymbol{w} = (w_x, w_y, w_z)$: 좌표계 A의 z-축을 좌표계 B로 표현



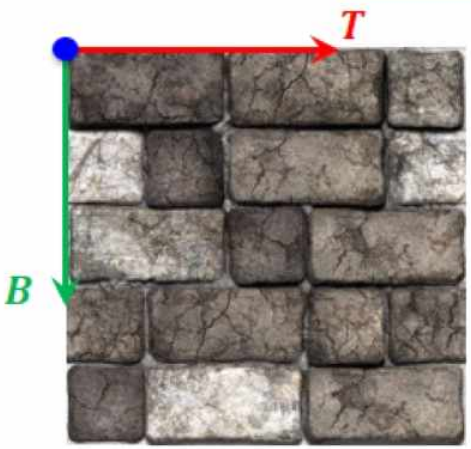
법선 매핑(Normal Mapping)

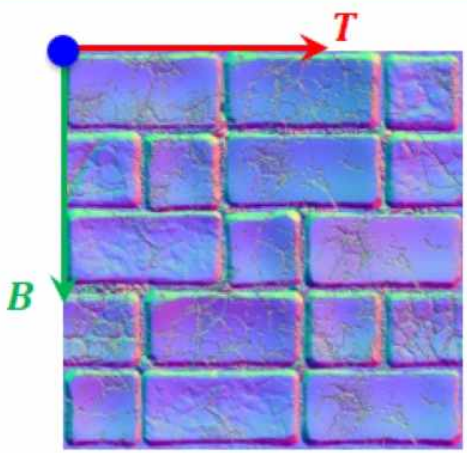
- **좌표계**
  - 월드 좌표계: 조명 계산을 위한 벡터
  - 모델 좌표계(로컬 좌표계)
  - 정점 접선 좌표계(Vertex Tangent Coordinates)
    정점 좌표계, 접선 좌표계
    각 정점마다 별도의 좌표계를 가정
    $x-$축: 정점의 접선 벡터(Tangent Vector), 텍스쳐의 $u-$축
    $y-$축: 정점의 종접선 벡터(Bi-Tangent Vector), 텍스쳐의 $v-$축
    $z-$축: 정점의 법선 벡터(Normal Vector)

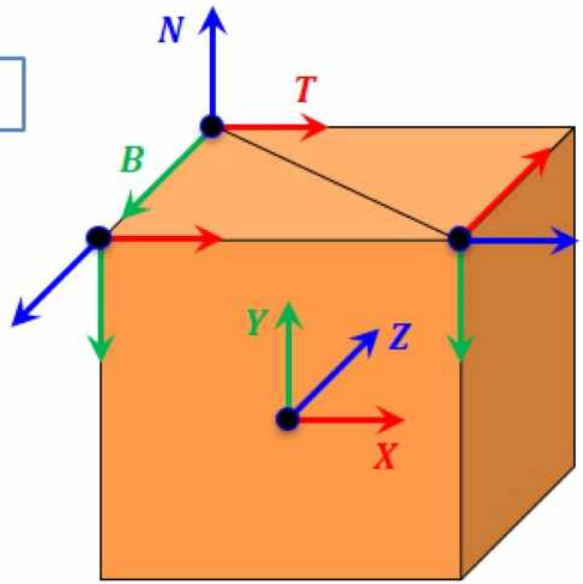
$$\boldsymbol{M}_{model} = \begin{bmatrix} T_x & T_y & T_z \\ B_x & B_y & B_z \\ N_x & N_y & N_z \end{bmatrix}$$

각 정점에 대한 TBN 직교 벡터

$\boldsymbol{M}_{model}$: 접선 좌표계를 모델 좌표계로 변환

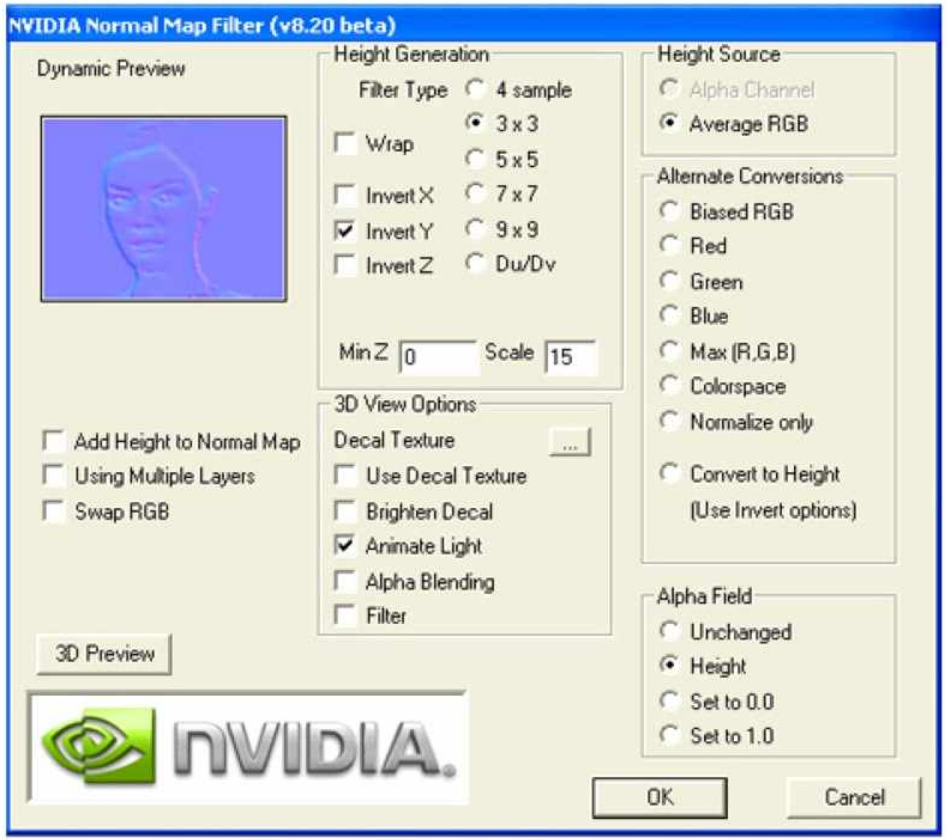






법선 매핑(Normal Mapping)

- **법선 매핑(Normal Mapping)**
  - 법선 맵(Normal Map) 생성하기
    - NVIDIA Texture Tools for Adobe Photoshop
      https://developer.nvidia.com/nvidia-texture-tools-adobe-photoshop
      "NVIDIA Normal Map Filter"



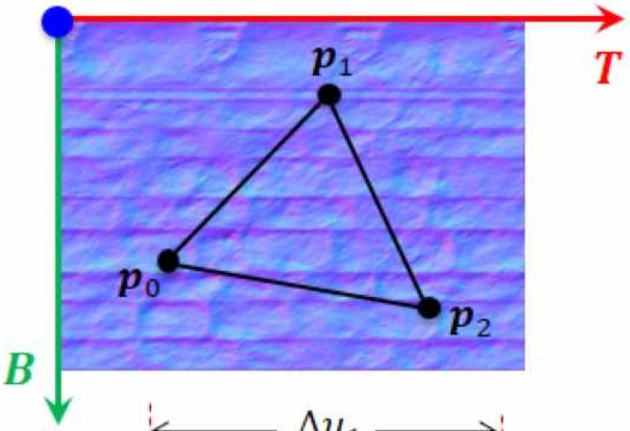
    - GIMP(GNU Image Manipulation Program)
      http://www.gimp.org/

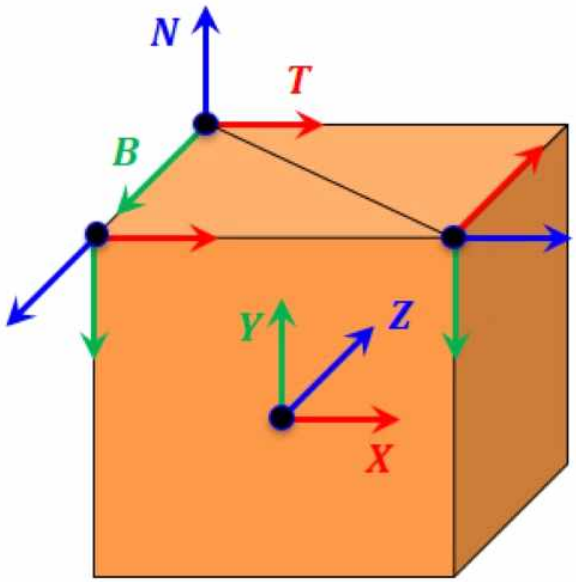
CrazyBump

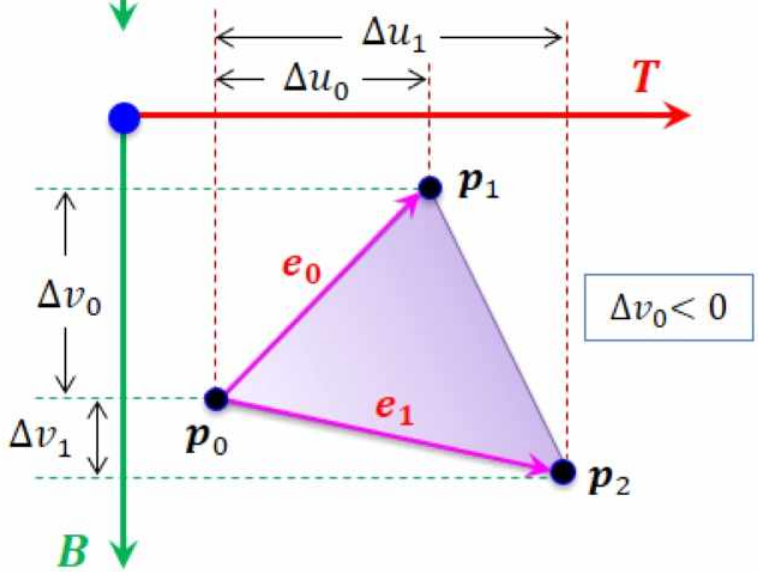
ZBrush

법선 매핑(Normal Mapping)

- **법선 매핑(Normal Mapping)**
  - 좌표계 변환
    법선 맵(Normal Map)의 법선 벡터는 텍스쳐 좌표계로 표현됨
    조명 계산을 위한 법선 벡터는 월드 좌표계(모델 좌표계)를 사용
  - 하나의 삼각형에 대한 접선 좌표계







$\Delta v_0 < 0$

$e_0 = p_1 - p_0$
$e_1 = p_2 - p_0$

$p_0,\ p_1,\ p_2$: 정점(모델 좌표계)
$t_0,\ t_1,\ t_2$: 텍스쳐 좌표

$t_0 = (u_0, v_0), t_1 = (u_1, v_1), t_2 = (u_2, v_2)$
$(\Delta u_0, \Delta v_0) = (u_1 - u_0, v_1 - v_0)$
$(\Delta u_1, \Delta v_1) = (u_2 - u_0, v_2 - v_0)$

$e_0 = \Delta u_0 T + \Delta v_0 B$
$e_1 = \Delta u_1 T + \Delta v_1 B$

법선 매핑(Normal Mapping)

- **법선 매핑(Normal Mapping)**
  - 텍스쳐 좌표계(접선 좌표계)

$$\boldsymbol{e_0} = \Delta u_0 \boldsymbol{T} + \Delta v_0 \boldsymbol{B}$$
$$\boldsymbol{e_1} = \Delta u_1 \boldsymbol{T} + \Delta v_1 \boldsymbol{B}$$

$$\begin{pmatrix} \boldsymbol{e_0} \\ \boldsymbol{e_1} \end{pmatrix} = \begin{pmatrix} x_0 & y_0 & z_0 \\ x_1 & y_1 & z_1 \end{pmatrix} = \begin{bmatrix} \Delta u_0 & \Delta v_0 \\ \Delta u_1 & \Delta v_1 \end{bmatrix} \begin{pmatrix} T_x & T_y & T_z \\ B_x & B_y & B_z \end{pmatrix}$$

$$\begin{pmatrix} \boldsymbol{e_0} \\ \boldsymbol{e_1} \end{pmatrix} = \begin{bmatrix} \Delta u_0 & \Delta v_0 \\ \Delta u_1 & \Delta v_1 \end{bmatrix} \begin{pmatrix} \boldsymbol{T} \\ \boldsymbol{B} \end{pmatrix}$$

$$\begin{pmatrix} \boldsymbol{T} \\ \boldsymbol{B} \end{pmatrix} = \begin{bmatrix} \Delta u_0 & \Delta v_0 \\ \Delta u_1 & \Delta v_1 \end{bmatrix}^{-1} \begin{pmatrix} \boldsymbol{e_0} \\ \boldsymbol{e_1} \end{pmatrix}$$

$$\begin{pmatrix} T_x & T_y & T_z \\ B_x & B_y & B_z \end{pmatrix} = \begin{bmatrix} \Delta u_0 & \Delta v_0 \\ \Delta u_1 & \Delta v_1 \end{bmatrix}^{-1} \begin{pmatrix} x_0 & y_0 & z_0 \\ x_1 & y_1 & z_1 \end{pmatrix}$$

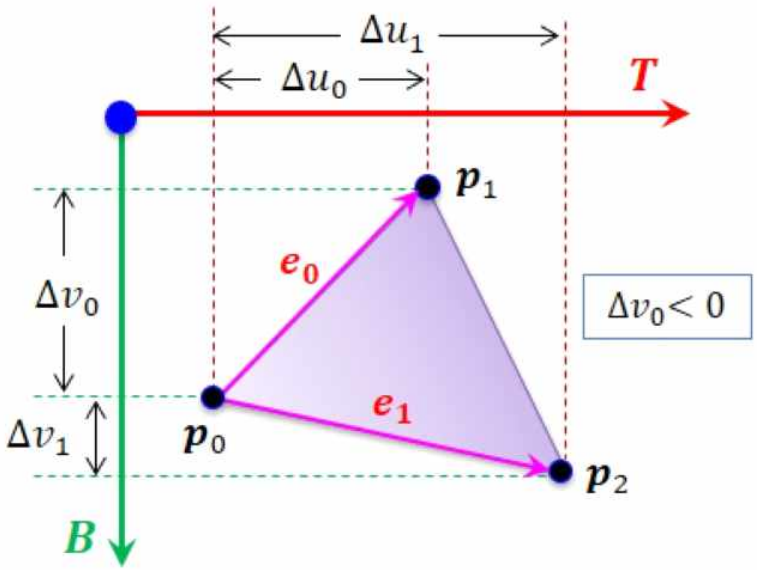
$\boldsymbol{e_0} = \boldsymbol{p_1} - \boldsymbol{p_0} = (x_0, y_0, z_0)$
$\boldsymbol{e_1} = \boldsymbol{p_2} - \boldsymbol{p_0} = (x_1, y_1, z_1)$

$\boldsymbol{t_0} = (u_0, v_0), \boldsymbol{t_1} = (u_1, v_1), \boldsymbol{t_2} = (u_2, v_2)$
$(\Delta u_0, \Delta v_0) = (u_1 - u_0, v_1 - v_0)$
$(\Delta u_1, \Delta v_1) = (u_2 - u_0, v_2 - v_0)$

$\boldsymbol{T}$는 텍스쳐 좌표계의 $\boldsymbol{u}$축을 모델 좌표계로 표현한 것임
$\boldsymbol{B}$는 텍스쳐 좌표계의 $\boldsymbol{v}$축을 모델 좌표계로 표현한 것임

$\boldsymbol{e_0}, \boldsymbol{e_1}$: 정점(모델 좌표계)에서 구할 수 있음

$(\Delta u_0, \Delta v_0), (\Delta u_1, \Delta v_1)$: 텍스쳐 좌표에서 구할 수 있음



$\boldsymbol{p_0}, \boldsymbol{p_1}, \boldsymbol{p_2}$: 정점(모델 좌표계)
$\boldsymbol{t_0}, \boldsymbol{t_1}, \boldsymbol{t_2}$: 텍스쳐 좌표

삼각형의 접선 좌표계

$\boldsymbol{TBN}$을 직교하도록 만들고 정규화

$$\boldsymbol{N} = \boldsymbol{T} \times \boldsymbol{B}$$
$$\boldsymbol{T'} = \|\boldsymbol{T} - (\boldsymbol{N} \cdot \boldsymbol{T})\boldsymbol{N}\|$$
$$\boldsymbol{B'} = \left\| \boldsymbol{B} - (\boldsymbol{N} \cdot \boldsymbol{B})\boldsymbol{N} - \frac{(\boldsymbol{T'} \cdot \boldsymbol{B})\boldsymbol{T'}}{\boldsymbol{T'} \cdot \boldsymbol{T'}} \right\|$$

법선 매핑(Normal Mapping)

- **접선 좌표계(Tangent Space)**
  - 텍스쳐 좌표계(접선 좌표계)

$\boldsymbol{p_0} = (x_0, y_0, z_0),\ \boldsymbol{p_1} = (x_1, y_1, z_1),\ \boldsymbol{p_2} = (x_2, y_2, z_2)$

$t_0 = (u_0, v_0), t_1 = (u_1, v_1), t_2 = (u_2, v_2)$
$(\Delta u_0, \Delta v_0) = (u_1 - u_0, v_1 - v_0)$
$(\Delta u_1, \Delta v_1) = (u_2 - u_0, v_2 - v_0)$

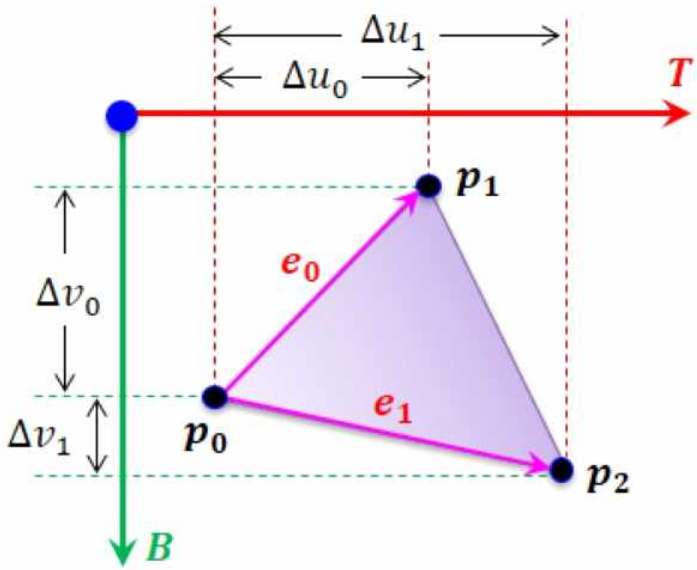
$$x_1 = x_0 + (u_1 - u_0)T_x + (v_1 - v_0)B_x = x_0 + \Delta u_0 T_x + \Delta v_0 B_x$$
$$x_2 = x_0 + (u_2 - u_0)T_x + (v_2 - v_0)B_x = x_0 + \Delta u_1 T_x + \Delta v_1 B_x$$
$$x_1 - x_0 = \Delta u_0 T_x + \Delta v_0 B_x$$
$$x_2 - x_0 = \Delta u_1 T_x + \Delta v_1 B_x$$
$$\begin{pmatrix} x_1 - x_0 \\ x_2 - x_0 \end{pmatrix} = \begin{bmatrix} \Delta u_0 & \Delta v_0 \\ \Delta u_1 & \Delta v_1 \end{bmatrix} \begin{pmatrix} T_x \\ B_x \end{pmatrix}$$
$$\begin{pmatrix} T_x \\ B_x \end{pmatrix} = \begin{bmatrix} \Delta u_0 & \Delta v_0 \\ \Delta u_1 & \Delta v_1 \end{bmatrix}^{-1} \begin{pmatrix} x_1 - x_0 \\ x_2 - x_0 \end{pmatrix}$$
$$= \frac{1}{|\Delta u_0 \Delta v_1 - \Delta u_1 \Delta v_0|} \begin{bmatrix} \Delta v_1 & -\Delta v_0 \\ -\Delta u_1 & \Delta u_0 \end{bmatrix} \begin{pmatrix} x_1 - x_0 \\ x_2 - x_0 \end{pmatrix}$$

$$\begin{pmatrix} T_x & T_y & T_z \\ B_x & B_y & B_z \end{pmatrix} = \begin{bmatrix} \Delta u_0 & \Delta v_0 \\ \Delta u_1 & \Delta v_1 \end{bmatrix}^{-1} \begin{pmatrix} x_1 - x_0 & y_1 - y_0 & z_1 - z_0 \\ x_2 - x_0 & y_2 - y_0 & z_2 - z_0 \end{pmatrix}$$

$\boldsymbol{N} = \boldsymbol{T} \times \boldsymbol{B}$

$$\boldsymbol{M} = \begin{bmatrix} T_x & T_y & T_z \\ B_x & B_y & B_z \\ N_x & N_y & N_z \end{bmatrix}$$

접선 좌표계를 모델 좌표계로 변환하는 행렬



$\boldsymbol{p_0}$, $\boldsymbol{p_1}$, $\boldsymbol{p_2}$: 정점(모델 좌표계)
$\boldsymbol{t_0}$, $\boldsymbol{t_1}$, $\boldsymbol{t_2}$: 텍스쳐 좌표

$\boldsymbol{e_0} = \boldsymbol{p_1} - \boldsymbol{p_0} = (x_1 - x_0, y_1 - y_0, z_1 - z_0)$
$\boldsymbol{e_1} = \boldsymbol{p_2} - \boldsymbol{p_0} = (x_2 - x_0, y_2 - y_0, z_2 - z_0)$

삼각형의 접선 좌표계

법선 매핑(Normal Mapping)

- **법선 매핑(Normal Mapping)**
  - 정점 접선 좌표계(Vertex Tangent Coordinates)
    삼각형에 대한 접선 좌표계
    법선 벡터: 정점의 법선 벡터(정점을 포함하는 삼각형의 법선 벡터의 평균)
    모델에서 정점 v의 접선 벡터 T는 v를 포함하는 삼각형의 접선 벡터들의 평균
    모델에서 정점 v의 종법선 벡터 B는 v를 포함하는 삼각형의 종접선 벡터들의 평균
    $\boldsymbol{TBN}$ 벡터들이 직교하도록 만들고 정규화(Orthonormal)

$$\boldsymbol{M} = \begin{bmatrix} T_x & T_y & T_z \\ B_x & B_y & B_z \\ N_x & N_y & N_z \end{bmatrix} \qquad \boldsymbol{M}^{-1} = \begin{bmatrix} T_x & B_x & N_x \\ T_y & B_y & N_y \\ T_z & B_z & N_z \end{bmatrix}$$

접선 좌표계 → 모델 좌표계

모델 좌표계 → 접선 좌표계

$$\boldsymbol{N_W} = \boldsymbol{nMW} = \boldsymbol{n(MW)}$$
$$\boldsymbol{T_W} = \boldsymbol{tMW} = \boldsymbol{t(MW)}$$
$$\boldsymbol{B_W} = \boldsymbol{N_W} \times \boldsymbol{T_W}$$

$$\boldsymbol{M_W} = \begin{bmatrix} T_{w.x} & T_{w.y} & T_{w.z} \\ B_{w.x} & B_{w.y} & B_{w.z} \\ N_{w.x} & N_{w.y} & N_{w.z} \end{bmatrix}$$

```
struct CVertex
{
    XMFLOAT3 position;
    XMFLOAT3 normal;
    XMFLOAT3 tangent;
    XMFLOAT2 uv;
};
```

모델 좌표계

각 정점에 대한 $\boldsymbol{TBN}$ 직교 벡터

① 법선 맵을 생성(그래픽 도구 또는 다른 방법으로)
② 각 삼각형에 대하여 접선 벡터 $\boldsymbol{T}$를 계산, 각 정점에 대한 접선 벡터(모델 좌표계)를 계산하여 정점에 설정 (3D 모델링 도구에서 처리할 수 있음)
③ 법선 맵의 텍스쳐 리소스를 생성하여 정점 셰이더에 연결
④ 정점 셰이더에서 각 정점의 접선 벡터와 법선 벡터를 월드 좌표계로 변환
⑤ 픽셀 셰이더에서 각 픽셀의 보간된 접선 벡터와 법선 벡터를 사용하여 $\boldsymbol{TBN}$ 변환 행렬($\boldsymbol{M_W}$)을 생성
⑥ 법선 맵 텍스쳐에서 샘플링한 법선 벡터를 $\boldsymbol{TBN}$ 변환 행렬을 사용하여 월드 좌표계로 변환
⑦ 픽셀 셰이더에서 조명 계산

법선 매핑(Normal Mapping)

- 법선 매핑(Normal Mapping)

```
VS_OUTPUT VS(VS_INPUT input)
{
    VS_OUTPUT output;
    output.position = mul(float4(input.position, 1.0f), gmtxWorld);
    output.positionW = output.position.xyz;
    output.tangentW = mul(input.tangent, (float3x3)gmtxWorld);
    output.normalW = mul(input.normal, (float3x3)gmtxWorld);
    output.position = mul(mul(output.position, gmtxView), gmtxProjection);
    output.uv = input.uv;
    return(output);
}

float4 PS(VS_OUTPUT input) : SV_Target
{
    float3 N = normalize(input.normalW);
    float3 T = normalize(input.tangentW - dot(input.tangentW, N) * N);
    float3 B = cross(N, T);
    float3x3 TBN = float3x3(T, B, N);

    float3 normal = gtxtNormal.Sample(gssNormal, input.uv).rgb;
    float3 normal = 2.0f * normal - 1.0f; //[0, 1] → [-1, 1]
    float3 normalW = mul(normal, TBN);

    float4 cIllumination = Lighting(input.positionW, normalW);
    return(cIllumination);
}
```

모델 좌표계의 기저 벡터

```
struct VS_INPUT {
    float3 position : POSITION;
    float3 normal : NORMAL;
    float3 tangent  : TANGENT;
    float2 uv : TEXCOORD;
};
```

각 정점의 TBN 벡터(월드 좌표계)

```
Texture2D gtxtNormal : register(t2);
SamplerState gssNormal : register(s2);
```

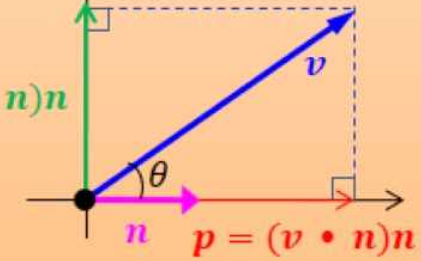
월드 좌표계의 기저 벡터

```
struct VS_OUTPUT {
    float4 position : SV_POSITION;
    float3 positionW : POSITION;
    float3 normalW : NORMAL;
    float3 tangentW : TANGENT;
    float2 uv : TEXCOORD;
};
```

$$TBN = \begin{bmatrix} T_{w.x} & T_{w.y} & T_{w.z} \\ B_{w.x} & B_{w.y} & B_{w.z} \\ N_{w.x} & N_{w.y} & N_{w.z} \end{bmatrix}$$

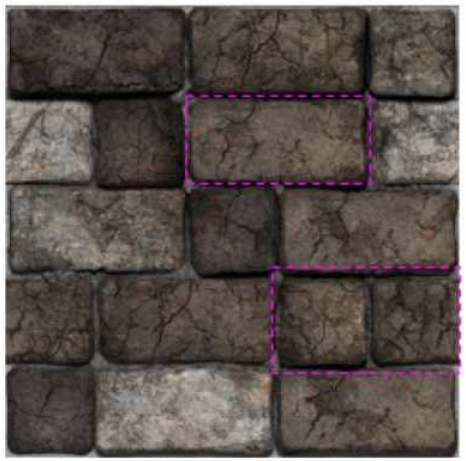
Gram-Schmidt

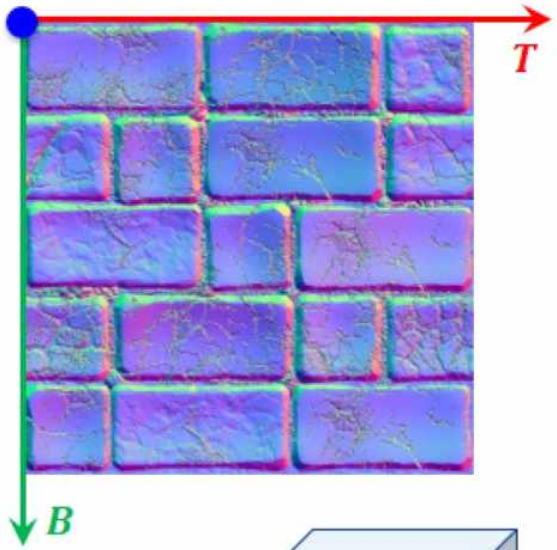
$$q = v - (v \bullet n)n$$

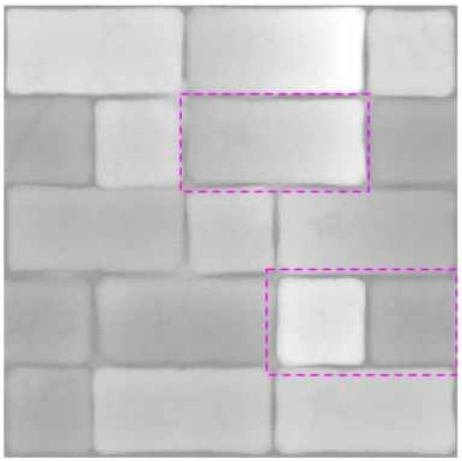


변위 매핑(Displacement Mapping)

- **변위 매핑(Displacement Mapping)**
  - 법선 맵(Normal Map)
    조명 효과를 계산하기 위한 법선 벡터 정보를 저장
  - 변위 맵(Displacement Map)
    메쉬의 변이 정보를 나타내는 높이 맵(하나의 색상: [0, 1], 스칼라 맵)
    일반적으로 법선 맵의 알파 채널을 사용
    기하 셰이더 또는 테셀레이션 단계(Domain Shader)에서 사용
    변위 맵을 생성할 수 있는 도구가 필요(예: www.crazybump.com)

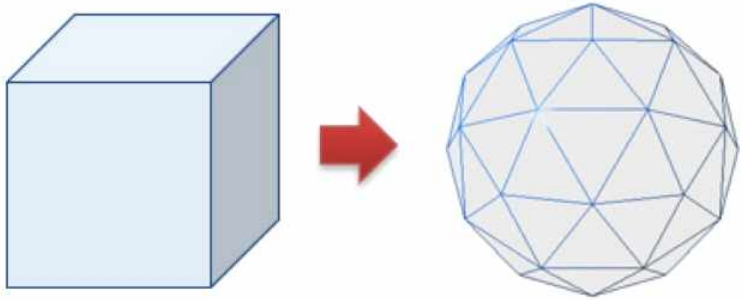






$$\boldsymbol{p} = \boldsymbol{p} + \{ s * (h - 1) \}\boldsymbol{n}$$
$$h \in [0,1] \Rightarrow \{ s * (h - 1) \} \in [-s, 0]$$

$$\boldsymbol{p} = \boldsymbol{p} + \{ s * (1 - h) \}\boldsymbol{n}$$
$$h \in [0,1] \Rightarrow \{ s * (1 - h) \} \in [0, s]$$



법선 매핑(Normal Mapping)

- 변위 매핑(Displacement Mapping)

```
cbuffer cbDisplacement : register(b3) {
   float gfDisplacementScale;
};
```

```
VS_OUTPUT VS(VS_INPUT input)
{
   VS_OUTPUT output;
   float offset = gtxtDisplacement.SampleLevel(gssDisplacement, input.uv, 0).a;
   input.position += input.normal * gfDisplacementScale * (offset – 1.0f);
   output.position = mul(float4(input.position, 1.0f), gmtxWorld);
   output.positionW = output.position.xyz;
   output.tangentW = mul(input.tangent, (float3x3)gmtxWorld);
   output.normalW = mul(input.normal, (float3x3)gmtxWorld);
   output.position = mul(mul(output.position, gmtxView), gmtxProjection);
   output.uv = input.uv;
   return(output);
}
```

```
struct VS_INPUT {
   float3 position : POSITION;
   float3 normal : NORMAL;
   float3 tangent  : TANGENT;
   float2 uv : TEXCOORD;
};
```

```
Texture2D gtxtDisplacement : register(t3);
SamplerState gssDisplacement : register(s3);
```

```
struct VS_OUTPUT {
    float4 position : SV_POSITION;
    float3 positionW : POSITION;
    float3 normalW : NORMAL;
    float3 tangentW : TANGENT;
    float2 uv : TEXCOORD;
};
```

```
float4 PS(VS_OUTPUT input) : SV_Target
{
   float3 N = normalize(input.normalW);
   float3 T = normalize(input.tangentW - dot(input.tangentW, N) * N);
   float3 B = cross(N, T);
   float3x3 TBN = float3x3(T, B, N);

   float3 normal = gtxtDisplacement.Sample(gssDisplacement, input.uv).rgb;
   normal = 2.0f * normal - 1.0f; //[0, 1] → [-1, 1]
   float3 normalW = mul(normal, TBN);

   return(Lighting(input.positionW, normalW));
}
```

변위 매핑(Displacement Mapping)

- 변위 매핑(Displacement Mapping)

```
VS_OUTPUT VS(VS_INPUT input)
{
    VS_OUTPUT output;
    output.positionW = mul(input.position, gmtxWorld);
    output.tangentW = mul(input.tangent, (float3x3)gmtxWorld);
    output.normalW = mul(input.normal, (float3x3)gmtxWorld);
    output.uv = input.uv;

    float fDistToCamera = distance(output.positionW, gvCameraPosition);
    float fTessFactor = saturate((fDistToCamera - gfMinDistance) / (gfMaxDistance - gfMinDistance));
    output.fTessFactor = gfMinTessFactor + fTessFactor * (gfMaxTessFactor - gfMinTessFactor);

    return(output);
}
```

```
struct VS_INPUT {
    float3 position : POSITION;
    float3 normal : NORMAL;
    float3 tangent : TANGENT;
    float2 uv : TEXCOORD;
};
```

```
struct VS_OUTPUT
{
    float3 positionW : POSITION;
    float3 normalW : NORMAL;
    float3 tangentW : TANGENT;
    float2 uv : TEXCOORD;
    float fTessFactor : TESSFACTOR;
};
```

```
cbuffer cbTessellation : register(b2)
{
    float gfMinDistance;
    float gfMaxDistance;
    float gfMinTessFactor;
    float gfMaxTessFactor;
};
```

```
pd3dCommandList->IASetPrimitiveTopology(D3D_PRIMITIVE_TOPOLOGY_3_CONTROL_POINT_PATCHLIST);
```

변위 매핑(Displacement Mapping)

- 변위 매핑(Displacement Mapping)

```
HSC_OUTPUT ConstantHS(InputPatch<VS_OUTPUT, 3> patch, uint nPatchID : SV_PrimitiveID)
{
   HSC_OUTPUT output;
   output.fTessEdges[0] = 0.5f * (patch[1].fTessFactor + patch[2].fTessFactor);
   output.fTessEdges[1] = 0.5f * (patch[2].fTessFactor + patch[0].fTessFactor);
   output.fTessEdges[2] = 0.5f * (patch[0].fTessFactor + patch[1].fTessFactor);
   output.fTessInsides[0] = output.fTessEdges[0];
   return(output);
}
```

```
struct HSC_OUTPUT {
   float fTessEdges[3] : SV_TessFactor;
   float fTessInsides[1] : SV_InsideTessFactor;
};
```

```
[domain("tri")]
[partitioning("fractional_odd")]
[outputtopology("triangle_cw")]
[outputcontrolpoints(3)]
[patchconstantfunc("ConstantHS")]
[maxtessfactor(64.0f)]
HS_OUTPUT HS(InputPatch<VS_OUTPUT, 3> input, uint i: SV_OutputControlPointID, uint nPID: SV_PrimitiveID)
{
   HS_OUTPUT output;
   output.positionW = input[i].positionW;
   output.normalW = input[i].normalW;
   output.tangentW = input[i].tangentW;
   output.uv = input[i].uv;
   return(output);
}
```

```
struct HS_OUTPUT {
   float3 positionW : POSITION;
   float3 normalW : NORMAL;
   float3 tangentW : TANGENT;
   float2 uv : TEXCOORD;
};
```

```
struct VS_OUTPUT {
   float3 positionW : POSITION;
   float3 normalW : NORMAL;
   float3 tangentW : TANGENT;
   float2 uv : TEXCOORD;
   float fTessFactor : TESSFACTOR;
};
```

변위 매핑(Displacement Mapping)

- 변위 매핑(Displacement Mapping)

```
[domain("tri")]
DS_OUTPUT DS(HSC_OUTPUT input, float3 uv : SV_DomainLocation, OutputPatch<HS_OUTPUT, 3> tri)
{
    DS_OUTPUT output;

    output.positionW = uv.x * tri[0].positionW + uv.y * tri[1].positionW + uv.z * tri[2].positionW;
    output.normalW = uv.x * tri[0].normalW + uv.y * tri[1].normalW + uv.z * tri[2].normalW;
    output.tangentW = uv.x * tri[0].tangentW + uv.y * tri[1].tangentW + uv.z * tri[2].tangentW;
    output.uv = uv.x * tri[0].uv + uv.x * tri[1].uv + uv.x * tri[2].uv;
    output.normalW = normalize(output.normalW);

    float fDistToCamera = distance(output.positionW, gvCameraPosition);
    float fMipLevel = clamp(((fDistToCamera - gfMipLevelInterval) / gfMipLevelInterval), 0.0f, gfMaxMipLevel);
    float fHeight = gtxtNormal.SampleLevel(gssNormal, output.uv, fMipLevel).a;

    out.positionW += (gfHeightScale * (fHeight - 1.0f)) * output.normalW;

    output.position = mul(mul(float4(output.positionW, 1.0f), gmtxView), gmtxProjection);

    return(output);
}
```

```
Texture2D gtxtNormal : register(t2);
```

$$\boldsymbol{p} = \boldsymbol{p} + \{s * (h-1)\}\boldsymbol{n}$$
$$h \in [0,1] \Rightarrow \{s * (h-1)\} \in [-s, 0]$$

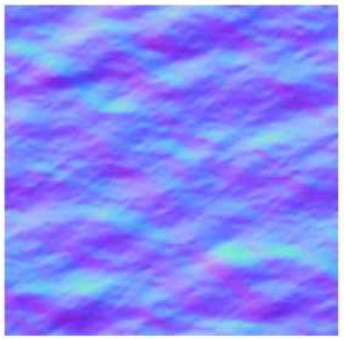
```
cbuffer cbDisplacement : register(b3) {
    float gfMipLevelInterval;
    float gfMaxMipLevel;
    float gfHeightScale;
};
```

```
struct HS_OUTPUT {
    float3 position : SV_POSITION;
    float3 positionW : POSITION;
    float3 normalW : NORMAL;
    float3 tangentW : TANGENT;
    float2 uv : TEXCOORD;
};
```

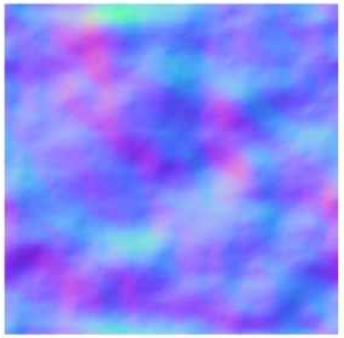
변위 매핑(Displacement Mapping)

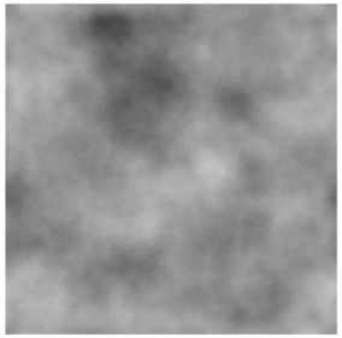
- 파도(Wave)

여러 개의 높이 맵을 스크롤
  텍스쳐 애니메이션(변환)
  다른 속력, 다른 방향
높이 맵들의 샘플링 결과를 결합(덧셈)
  정점의 높이(y-축)
법선 맵들의 샘플링 결과를 결합(덧셈)
  정점의 법선 벡터



주파수가 큰(주기가 작은) 법선 맵과 높이 값이 작은 높이 맵





주파수가 작은(주기가 큼) 법선 맵과 높이 값이 큰 높이 맵

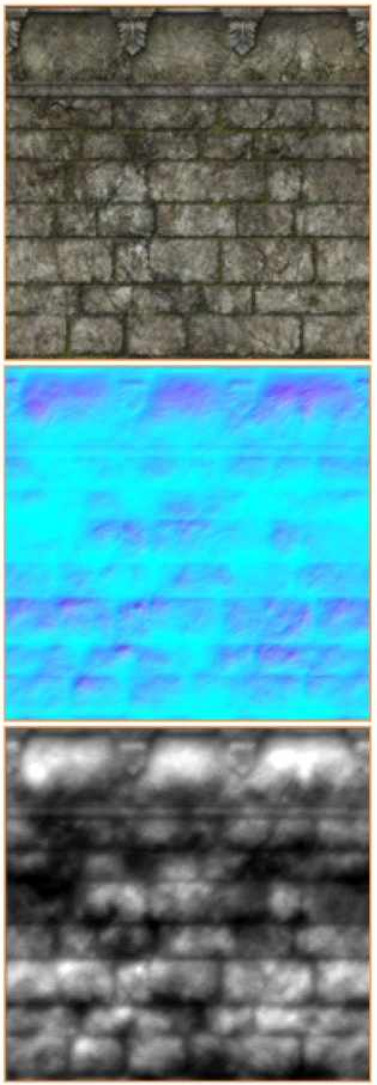
시차 매핑(Parallax Occlusion Mapping)

- **시차 매핑(Parallax Occlusion Mapping)**
  - 시차(Parallax)
    시점 변화에 따른 객체의 변이(Displacement)
    시점에 따라 표면이 다르게 보임

법선 맵을 사용: 표면을 정면으로 볼 때와 측면에서 볼 때가 같음
시차를 고려하면 표면을 정면으로 볼 때와 측면에서 볼 때가 다름

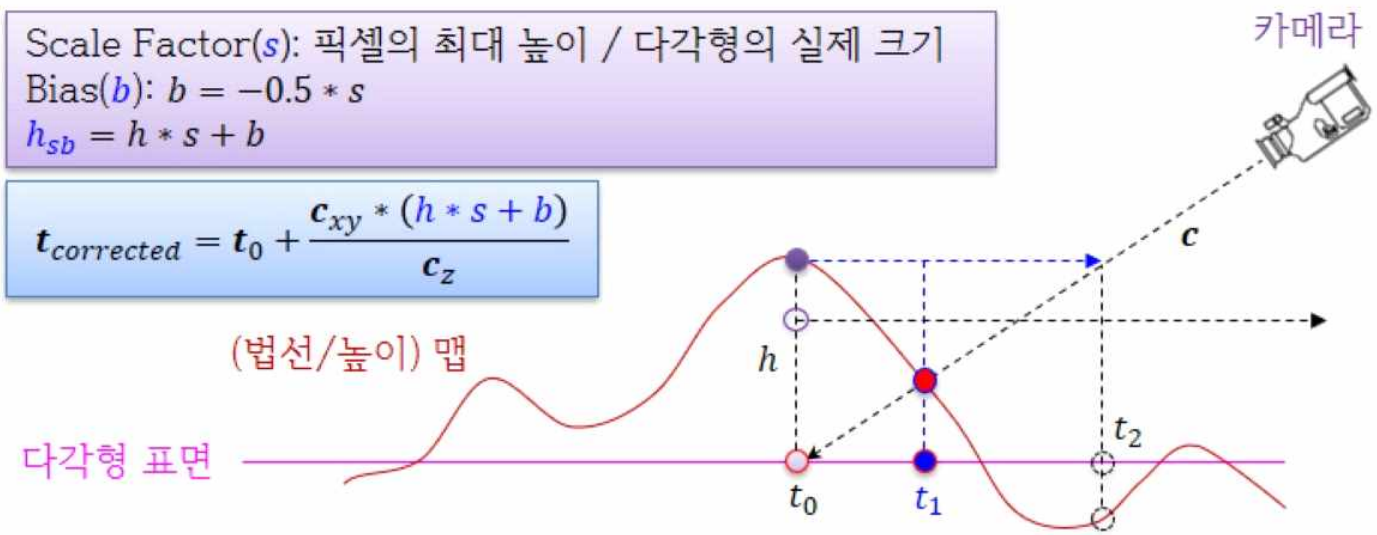
시차 매핑(Parallax Mapping)

- **시차 매핑(Parallax Mapping)**
  - 텍스쳐(색상, 법선, 높이) 맵은 실제 표면을 평면에 투영한 것임
    카메라의 위치에 따라 텍스쳐 좌표를 교정할 필요가 있음
  - 각 픽셀의 위치에서 텍스쳐 좌표를 교정
    - 텍스쳐 좌표
    - 표면의 높이
    - 픽셀에서 카메라까지의 벡터(접선 좌표계)
  - 텍스쳐 오프셋 벡터를 계산
    모든 계산은 접선 좌표계에서 수행
    시선 벡터와 조명의 방향 벡터를 접선 좌표계로 표현
    다각형 표면에 평행하게 따라가면서 시선 벡터와의 교점을 찾음



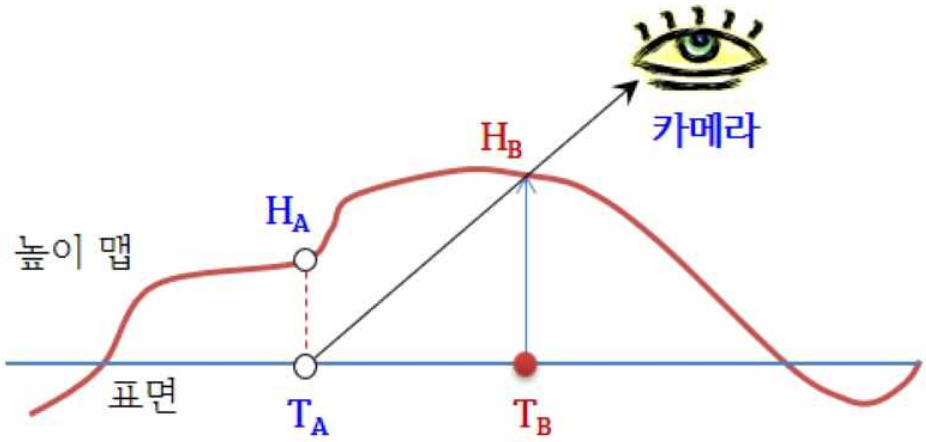
Scale Factor(s): 픽셀의 최대 높이 / 다각형의 실제 크기
Bias(b): $b = -0.5 * s$
$h_{sb} = h * s + b$

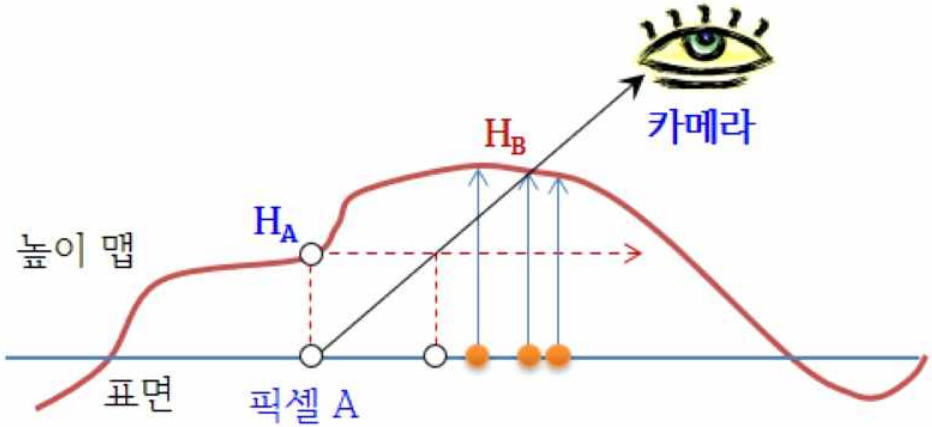
$$t_{corrected} = t_0 + \frac{c_{xy} * (h * s + b)}{c_z}$$



시차 매핑(Parallax Mapping)

- **시차 매핑(Parallax Mapping)**
  - 법선(범프) 맵은 픽셀 위치에서 실제 표면을 근사한 것
    실제 표면의 높이가 변하는 상황에서 시차가 발생
  - 시차는 보는 방향에 따라 발생함





실제 표면에서 카메라가 보는 높이는 H_A가 아니라 H_B
높이는 높이 맵을 사용하므로 텍스쳐 좌표가 필요
$T_{corrected}$를 직접 구하는 것은 복잡한 계산이 필요

$$T_{corrected} = T_A + \frac{\boldsymbol{c}_{xy} * \left((H_A * s) + b\right)}{\boldsymbol{c}_z}$$

근사 방법:

$$T_{corrected} = T_A + \boldsymbol{c}_{xy} * \left((H_A * s) + b\right)$$

T_A : 텍스쳐 좌표
$\boldsymbol{c}_{xy}$: 접선 좌표계(표면)에서 카메라 시선 벡터
$\boldsymbol{c}_z$: 접선 좌표계에서 카메라 시선 벡터의 z-좌표
H_A : 텍스쳐 좌표 T_A의 높이 값
s : 스케일(Scale)
b : 바이어스(Bias)

시차 매핑(Parallax Mapping)

- 시차 매핑(Parallax Mapping)

```
float4 PS(VS_OUTPUT input) : SV_Target
{
   float3 vCameraPosition = gvCameraPosition.xyz;
   float3 vToCamera = normalize(vCameraPosition – input.positionW);

   float fHeight = gtxtNormalMap.Sample(gHeightMapSamplerState, input.uv).a;
   //float fHeight = gtxtHeightMap.Sample(gHeightMapSamplerState, input.uv).r;
   float fHeight = fHeight * gfParallaxScale + gfParallaxBias;
   float2 vTexCorrected  = input.uv  + fHeight * vToCamera.xy;

   float4 cColor = gtxtColorMap.Sample(gColorMapSamplerState, vTexCorrected);

   float3 normal = gtxtNormalMap.Sample(gNormalMapSamplerState, vTexCorrected).rgb;
   normal = 2.0f * normal - 1.0f;

   float3 N = normalize(input.normalW);
   float3 T = normalize(input.tangentW - dot(input.tangentW, N) * N);
   float3 B = cross(N, T);
   float3x3 TBN = float3x3(T, B, N);
   float3 normalW = mul(normal, TBN);

   float4 cIllumination = Lighting(input.positionW, normalW);

   return(cColor * cIllumination);
}
```

$$T_{corrected} = T_A + \boldsymbol{c}_{xy} * \left((H_A * s) + b\right)$$

```
struct VS_INPUT {
   float3 position : POSITION;
   float3 normal : NORMAL;
   float3 tangent : TANGENT;
   float2 uv : TEXCOORD;
};
```

```
struct VS_OUTPUT {
    float4 position : SV_POSITION;
    float3 positionW : POSITION;
    float3 normalW : NORMAL;
    float3 tangentW : TANGENT;
    float2 uv : TEXCOORD;
};
```

```
cbuffer cbParallax : register(cb1) {
   float gfParallaxScale; //0.04f
   float gfParallaxBias; //0.02f
};
```

HLSL(High Level Shading Language for DirectX)

- 내장함수(Intrinsic Functions)

mul(x, y)

vector * vector = scalar
  x의 차원 = y의 차원
vector * matrix = vector
  vector의 차원 = matrix의 행 수
matrix * vector = vector
  vector의 차원 = matrix의 열 수

```
float3 vLightW = float3(...);
float3 vNormalW = float3(...);
float3 vTangentW = float3(...);
float3 vBinormalW = float3(...);
float3x3 mtxTransform = float3x3(vTangentW, vBinormalW, vNormalW);

float3 vLightTS = mul(mtxTransform, vLightW);

float3 vLightTS = mul(vLightW, mtxTransform);
```

$$\boldsymbol{M} = \begin{bmatrix} T_x & T_y & T_z \\ B_x & B_y & B_z \\ N_x & N_y & N_z \end{bmatrix} \qquad \boldsymbol{M^{-1}} = \begin{bmatrix} T_x & B_x & N_x \\ T_y & B_y & N_y \\ T_z & B_z & N_z \end{bmatrix} \qquad \boldsymbol{v} = (x, y, z)$$

$$\boldsymbol{Mv} = \begin{bmatrix} T_x & T_y & T_z \\ B_x & B_y & B_z \\ N_x & N_y & N_z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} T_x x + T_y y + T_z z \\ B_x x + B_y y + B_z z \\ N_x x + N_y y + N_z z \end{bmatrix} \qquad \boldsymbol{vM} = \begin{bmatrix} x & y & z \end{bmatrix} \begin{bmatrix} T_x & T_y & T_z \\ B_x & B_y & B_z \\ N_x & N_y & N_z \end{bmatrix} = \begin{bmatrix} T_x x + B_x y + N_x z \\ T_y x + B_y y + N_y z \\ T_z x + B_z y + N_z z \end{bmatrix}$$

$$\boldsymbol{M^{-1}v} = \begin{bmatrix} T_x & B_x & N_x \\ T_y & B_y & N_y \\ T_z & B_z & N_z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} T_x x + B_x y + N_x z \\ T_y x + B_y y + N_y z \\ T_z x + B_z y + N_z z \end{bmatrix} \qquad \boldsymbol{vM^{-1}} = \begin{bmatrix} x & y & z \end{bmatrix} \begin{bmatrix} T_x & B_x & N_x \\ T_y & B_y & N_y \\ T_z & B_z & N_z \end{bmatrix} = \begin{bmatrix} T_x x + T_y y + T_z z \\ B_x x + B_y y + B_z z \\ N_x x + N_y y + N_z z \end{bmatrix}$$

시차 매핑(Parallax Mapping)

- 시차 매핑(Parallax Mapping)

```
VS_OUTPUT VS(VS_INPUT input)
{
   VS_OUTPUT output = (VS_OUTPUT)0;

   float4 positionW = mul(float4(input.position, 1.0f), gmtxWorld);
   float3 normalW = mul(float4(input.normal, 1.0f), gmtxWorld).xyz;

   output.position = mul(mul(positionW, gmtxView), gmtxProjection);
   output.uv = input.uv;

   float3x3 mtxTangentToWorld;
   mtxTangentToWorld[0] = mul(input.tangent, gmtxWorld);
   mtxTangentToWorld[2] = mul(input.normal, gmtxWorld);
   mtxTangentToWorld[1] = mul(cross(input.tangent, input.normal), gmtxWorld));

   //float3x3 mtxWorldToTangent = transpose(mtxTangentToWorld); //역행렬
   //output.toLight = normalize(mul(gvLightPosition – positionW.xyz, mtxWorldToTangent));
   //output.toCamera = normalize(mul(gvCameraPosition – positionW.xyz, mtxWorldToTangent));
   //output.normal = normalize(mul(normalW, mtxWorldToTangent));

   output.toLight = normalize(mul(mtxTangentToWorld, gvLightPosition – positionW.xyz)); //v * W^-1
   output.toCamera = normalize(mul(mtxTangentToWorld, gvCameraPosition – positionW.xyz));
   output.normal = normalize(mul(mtxTangentToWorld, normalW));

   return(output);
}
```

```
struct VS_INPUT {
   float3 position : POSITION;
   float3 normal : NORMAL;
   float3 tangent  : TANGENT;
   float2 uv : TEXCOORD;
};
```

```
struct VS_OUTPUT {
   float4 position : SV_POSITION;
   float2 uv : TEXCOORD0;
   float3 normal : NORMAL;
   float3 toLight : TEXCOORD1;
   float3 toCamera : TEXCOORD2;
};
```

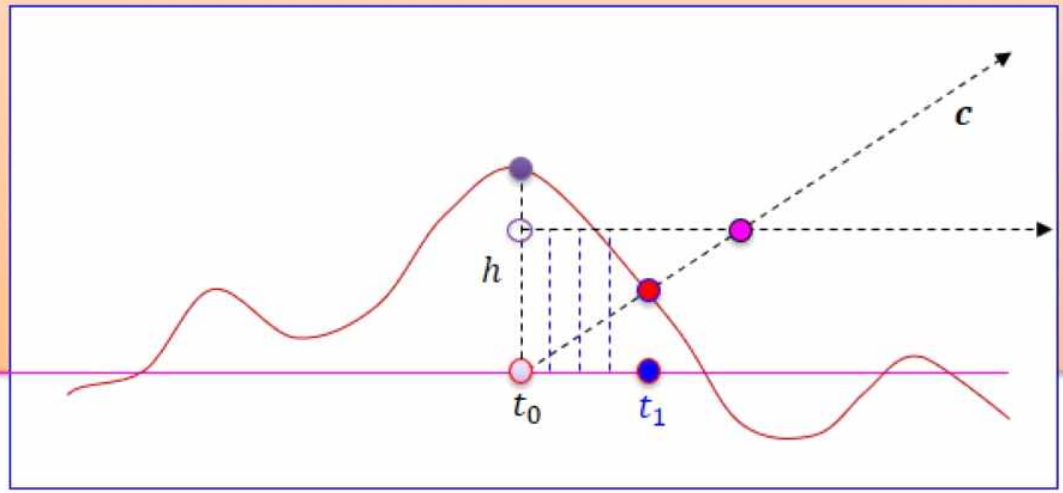
시차 매핑(Parallax Mapping)

- 시차 매핑(Parallax Mapping)

```
float4 PS(VS_OUTPUT input) : SV_Target
{
    float fParallaxLimit = -length(input.toCamera.xy) / input.toCamera.z;
    fParallaxLimit *= gfScale; //높이 맵 스케일

    float2 vOffsetDir = normalize(input.toCamera.xy);
    float2 vMaxOffset = vOffsetDir * fParallaxLimit;

    int nSamples = (int)lerp(gnMaxSamples, gnMinSamples, dot(input.toCamera, input.normal));

    float fStepSize = 1.0f / (float)nSamples;

    float2 dx = ddx(input.uv);
    float2 dy = ddy(input.uv);

    float fCurrRayHeight = 1.0f;
    float2 vCurrOffset = float2(0.0f, 0.0f);
    float2 vLastOffset = float2(0.0f, 0.0f);

    float fLastSampleHeight = 1.0f;
    float fCurrSampleHeight = 1.0f;

    int nCurrSample = 0;
```

```
static int gnMaxSamples= 20;
static int gnMinSamples= 4;
static float gfScale = 0.1f;
```

$$t_{corrected} = t_0 + \frac{c_{xy} * (h * s + b)}{c_z}$$



시차 매핑(Parallax Mapping)

- 시차 매핑(Parallax Mapping)

```
while (nCurrSample < nSamples) {
   fCurrSampleHeight = gtxtNormalMap.SampleGrad(gssSampler, input.uv + vCurrOffset, dx, dy).a;
   if (fCurrSampleHeight > fCurrRayHeight) {
      float fDelta1 = fCurrSampleHeight - fCurrRayHeight;
      float fDelta2 = (fCurrRayHeight + fStepSize) - fLastSampleHeight;
      float fRatio = fDelta1 / (fDelta1 + fDelta2);
      vCurrOffset = (fRatio) * vLastOffset + (1.0f - fRatio) * vCurrOffset;
      nCurrSample = nSamples + 1;
   }
   else {
      nCurrSample++;
      fCurrRayHeight -= fStepSize;
      vLastOffset = vCurrOffset;
      vCurrOffset += fStepSize * vMaxOffset;
      fLastSampleHeight = fCurrSampleHeight;
   }
}
float2 vFinalCoords = input.uv + vCurrOffset;
float4 vFinalNormal = gtxNormalMap.Sample(gssSampler, vFinalCoords) * 2.0f - 1.0f;
float4 vFinalColor = gtxtColorMap.Sample(gssSampler, vFinalCoords);
float3 vAmbient = vFinalColor.rgb * 0.1f;
float3 vDiffuse = vFinalColor.rgb * max(0.0f, dot(input.toLight, vFinalNormal.xyz)) * 0.5f;
vFinalColor.rgb = vAmbient + vDiffuse;

return(vFinalColor);
}
```

```
Texture2D gtxtColorMap : register(t2);
Texture2D gtxtNormalMap : register(t3);
```

```
DXGI_FORMAT Object.SampleGrad(
   sampler_state S,
   float Location,
   float DDX,
   float DDY
   [, int Offset]
);
```