3D Game Programming 21
- Height Field

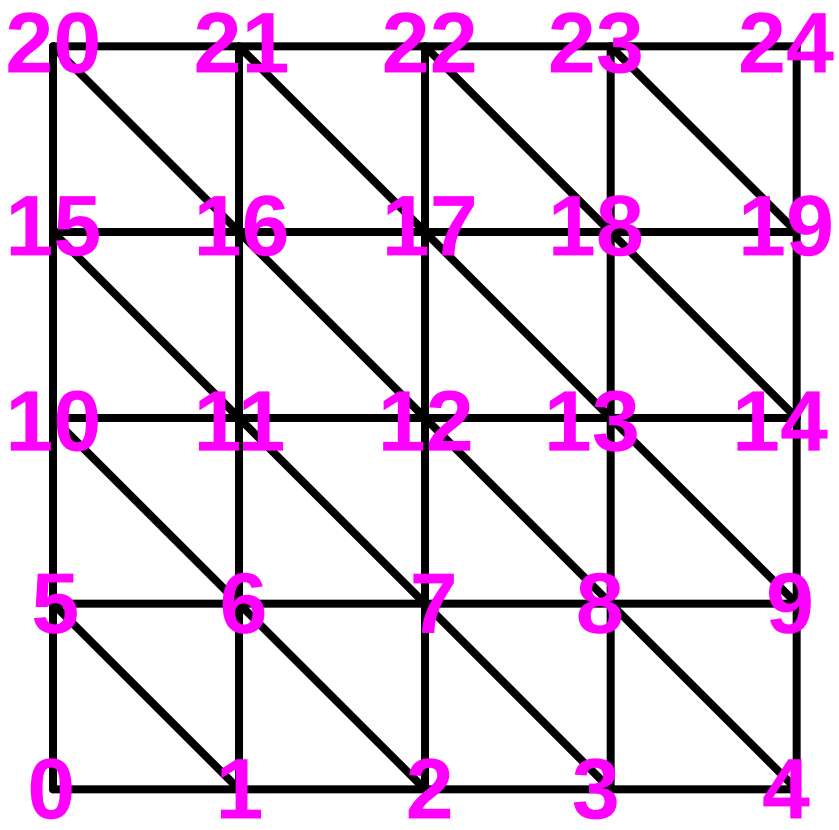
afewhee@gmail.com

- 높이 맵(Height Field) 개요
  - 실외 지형을 구성하는 방법으로 x축과 z축 방향에 대한 정점 사이의 거리를 일정한 간격으로 유지시켜 높이만 가지고도 지형을 구성할 수 있는 방법

- 장점
  - 격자가 일정한 크기를 가지고 있어서 지형을 구성하기가 쉽다.
  - 메모리를 절약할 수 있다.
  - 지형 위의 임의 지점의 높이 값을 간단한 계산으로 쉽게 찾을 수 있다.
  - 타일링, 스플레팅을 자유롭게 구사할 수 있다.
  - 작업에서나 구현 효율에서 단점 보다 장점이 많다.

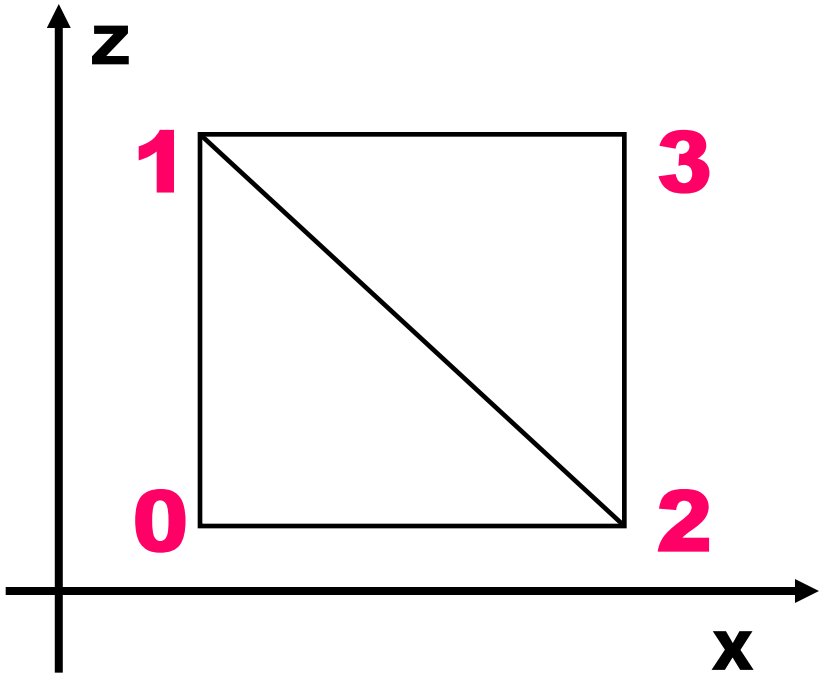
- 단점
  - LOD(Level of Detail)을 적용하기가 어렵다. ➔ 해상도가 낮은 멀리 떨어진 원경에 대해서 불 필요하게 많은 정점을 렌더링 할 수 있다. ➔ 블록(Block)으로 부분적인 LOD 적용

- Height Field 구성
  - 사선을 좌 상단에서 우 하단 방향의 빗금으로 구성하는 것이 유리



정점의 수 = (타일 + 1) * (타일 + 1)
삼각형(a,b,c) 인덱스 수 = 타일 * 타일 * 2

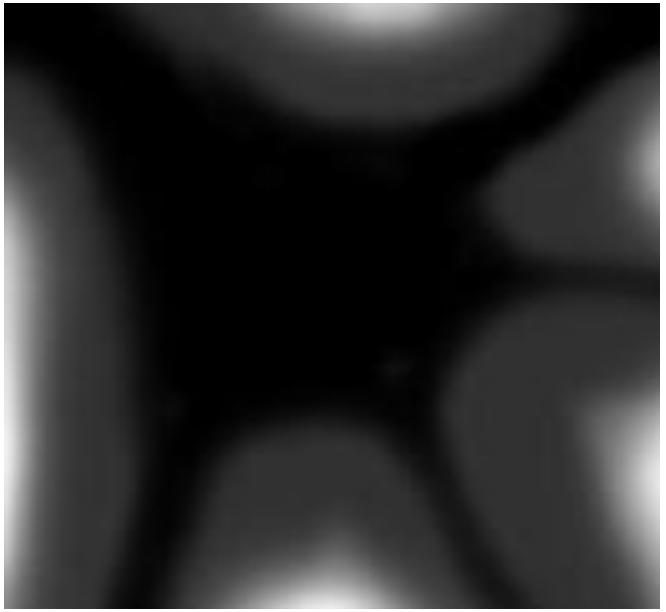
- Height Field 인덱스 구성




```
//인덱스
int _0 = (타일 수+1)*(z+0) +x;
int _1 = (타일 수+1)*(z+1) +x;
int _2 = (타일 수+1)*(z+0) +x +1;
int _3 = (타일 수+1)*(z+1) +x +1;
```

```
//삼각형 구성
m_pFce[n] = VtxIdx(_0, _1, _2); ++n;
m_pFce[n] = VtxIdx(_3, _2, _1); ++n;
```

- RAW
  - 해더가 0이고 흑백의 값(0~255)으로 구성된 이미지 파일
  - 헤더가 없어 높이 맵의 예제로 자주 사용

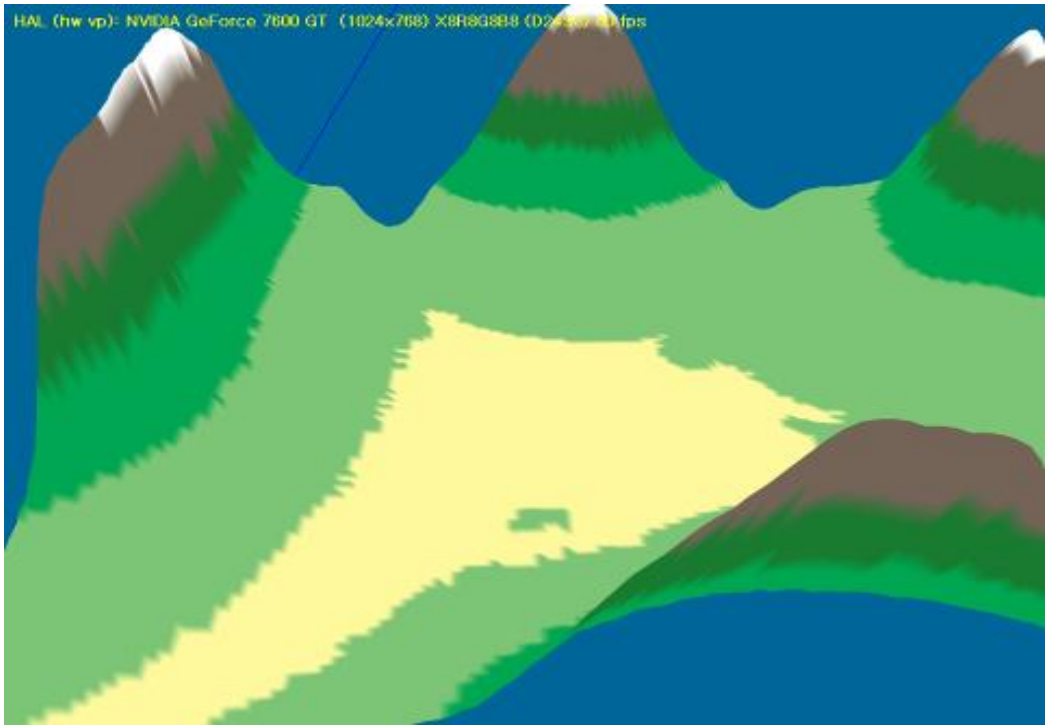


- RAW 읽기
  - 헤더가 없으므로 파일의 크기는 정점의 수와 비례

```
fp = fopen(sRaw, "rb");

fseek(fp, 0, SEEK_END);
long lSize = ftell(fp);              // 파일의 크기를 가져옴
fseek(fp, 0, SEEK_SET);
BYTE* pH = new BYTE[lSize];  // 파일 전체를 읽음
fread(pH, lSize, 1, fp); fclose(fp);

for(z=0; z<=m_TileN; ++z)
{
    for(x=0; x<=m_TileN; ++x)
    {
      // 이미지는 좌 상단에서 시작하므로 좌 하단에서 정점의 높이가 될 수 있도록 조정
      FLOAT h = pH[ (m_TileN - z) * (m_TileN+1) + x] * m_fHscl;
      n = z * (m_TileN+1) + x;
      m_pVtx[n].p.y = h;
    }
}
```

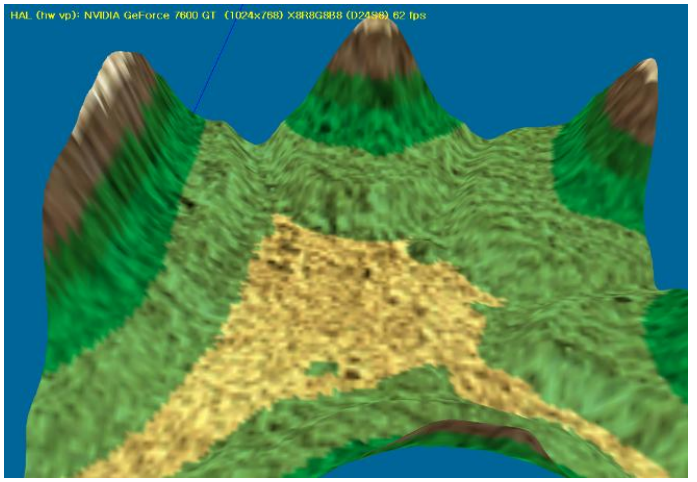
- 높이에 따른 Diffuse 적용

```
for(z=0; z<=m_TileN; ++z)
{
    for(x=0;x<=m_TileN; ++x)
    {
        FLOAT h;
        n = z * (m_TileN+1) + x;
        h = pH[n]*m_fHscl;
        m_pVtx[n].p.y = h;

        DWORD d;

        if( h < 1.f )
            d = D3DCOLOR_XRGB(255, 249, 157);
        else if( h < 45.0f )
            d = D3DCOLOR_XRGB(124, 197, 118);
        else if( h < 85.5f )
            d = D3DCOLOR_XRGB(  0, 166,  81);
        else if( h < 120.0f )
            d = D3DCOLOR_XRGB( 25, 123,  48);
        else if( h < 170.5f )
            d = D3DCOLOR_XRGB(115, 100,  87);
        else
            d = D3DCOLOR_XRGB(255, 255, 255);

        m_pVtx[n].d = d;
    }
}
```



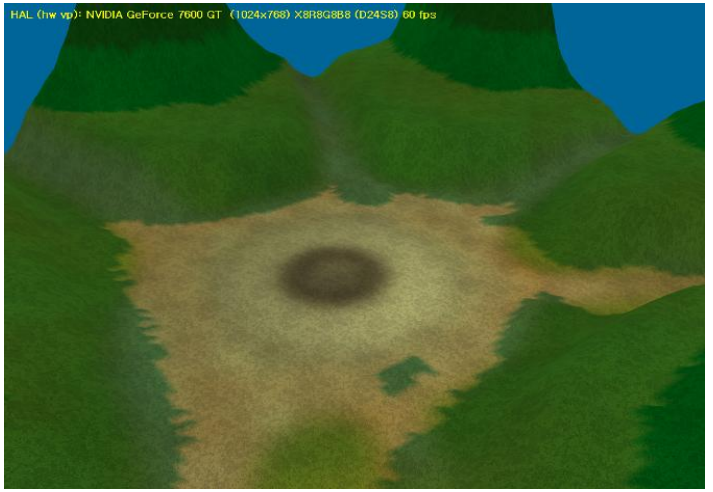
- Diffuse와 Detail Map
  - Image는 좌상단에서부터 읽게 되므로 uv에서 v만 조정
    m_pVtx[n].u = FLOAT(x)/m_TileN;
    m_pVtx[n].v = 1.f - FLOAT(z)/m_TileN;
  - Address Mode를 wrap, 또는 mirror로 조정

```
for(z=0; z<=m_TileN; ++z)
{
    for(x=0;x<=m_TileN; ++x)
    {
        n = z * (m_TileN+1) + x;

        m_pVtx[n].p = D3DXVECTOR3( FLOAT(x), 0.F, FLOAT(z));
        m_pVtx[n].p *= m_TileW;

        m_pVtx[n].u = FLOAT(x)/m_TileN;
        m_pVtx[n].v = 1.f - FLOAT(z)/m_TileN;

        m_pVtx[n].u *= m_fUV;
        m_pVtx[n].v *= m_fUV;
    }
}
```

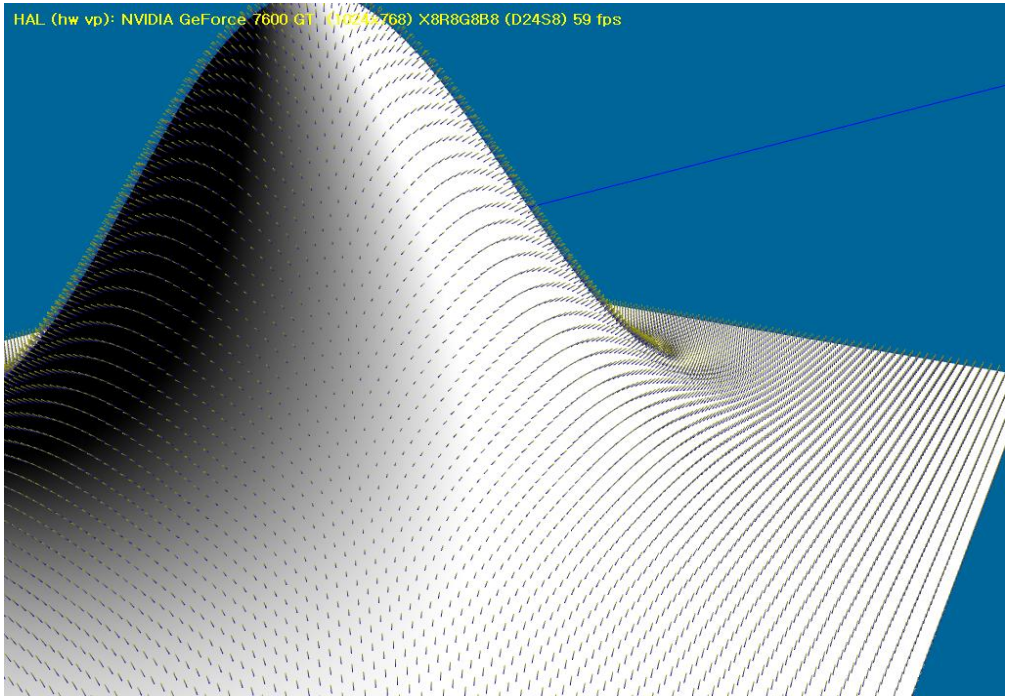


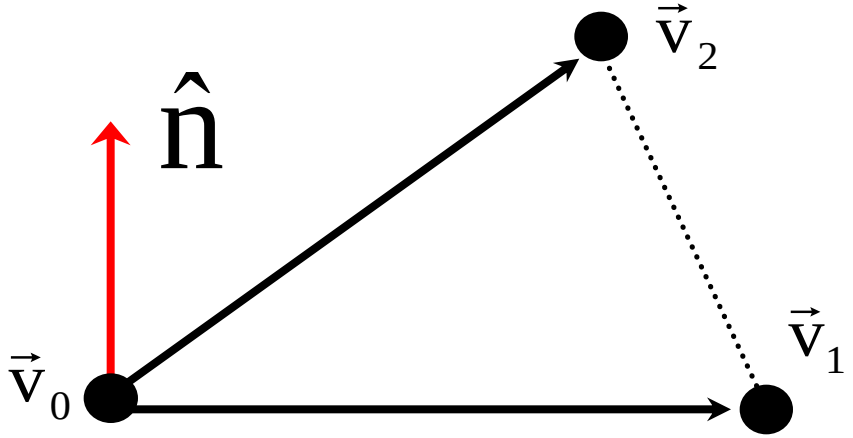
- Diffuse + Diffuse맵 + Detail Map Multi Texturing
  - 멀티 텍스처링에서 곱셈 연산을 계속 실행하면
    어두워 지므로 MODULATE2X나 MODULATE4X 사용

```
pDev->SetSamplerState(0, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR);
pDev->SetSamplerState(0, D3DSAMP_MINFILTER, D3DTEXF_LINEAR);
pDev->SetSamplerState(0, D3DSAMP_MIPFILTER, D3DTEXF_LINEAR);

pDev->SetSamplerState(1, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR);
pDev->SetSamplerState(1, D3DSAMP_MINFILTER, D3DTEXF_LINEAR);
pDev->SetSamplerState(1, D3DSAMP_MIPFILTER, D3DTEXF_LINEAR);

pDev->SetTextureStageState(0, D3DTSS_COLORARG1, D3DTA_TEXTURE);
pDev->SetTextureStageState(0, D3DTSS_COLORARG2, D3DTA_DIFFUSE);
pDev->SetTextureStageState(0, D3DTSS_COLOROP, D3DTOP_MODULATE);

pDev->SetTextureStageState(0, D3DTSS_ALPHAARG1, D3DTA_TEXTURE);
pDev->SetTextureStageState(0, D3DTSS_ALPHAARG2, D3DTA_DIFFUSE);
pDev->SetTextureStageState(0, D3DTSS_ALPHAOP, D3DTOP_MODULATE);

pDev->SetTextureStageState(1, D3DTSS_COLORARG1, D3DTA_CURRENT);
pDev->SetTextureStageState(1, D3DTSS_COLORARG2, D3DTA_TEXTURE);
pDev->SetTextureStageState(1, D3DTSS_COLOROP, D3DTOP_MODULATE2X);

pDev->SetTextureStageState(1, D3DTSS_ALPHAARG1, D3DTA_CURRENT);
pDev->SetTextureStageState(1, D3DTSS_ALPHAARG2, D3DTA_TEXTURE);
pDev->SetTextureStageState(1, D3DTSS_ALPHAOP, D3DTOP_MODULATE);

pDev->SetTexture(0, m_pTxDif);
pDev->SetTexture(1, m_pTxDet);

pDev->SetFVF(VtxDUV2::FVF);

pDev->DrawIndexedPrimitiveUP(...);

pDev->SetTexture(0, NULL);
pDev->SetTexture(1, NULL);
```



- 정점의 법선 벡터 만들기
  - 정점에 라이팅을 적용하려면 법선 벡터가 필요





$$\vec{N} = (\vec{v}_1 - \vec{v}_0) \times (\vec{v}_2 - \vec{v}_0)$$

$$\hat{n} = \frac{\vec{N}}{|\vec{N}|}$$

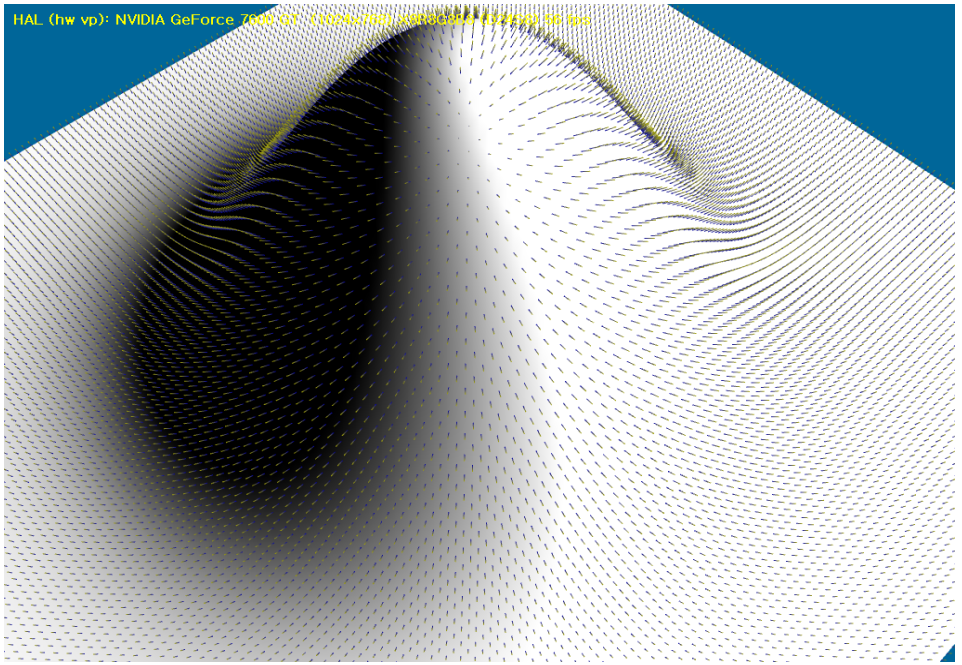
```
D3DXVECTOR3  N;

D3DXVECTOR3  A = v1 - v0;
D3DXVECTOR3  B = v2 - v0;

D3DXVec3Cross(&N,  &A, &B);
D3DXVec3Normalize(&N,  &N);
```

- 세점을 이용한 법선 벡터 프로그램

```
void CalculateNormal(D3DXVECTOR3* pOut
                                    ,  D3DXVECTOR3* v0
                                    ,  D3DXVECTOR3* v1
                                    ,  D3DXVECTOR3* v2)
{
        D3DXVECTOR3 n;
        D3DXVECTOR3 A = *v2 - *v0;
        D3DXVECTOR3 B = *v1 - *v0;
        D3DXVec3Cross(&n, &A, &B);
        D3DXVec3Normalize(&n, &n);
    *pOut = n;
}
```

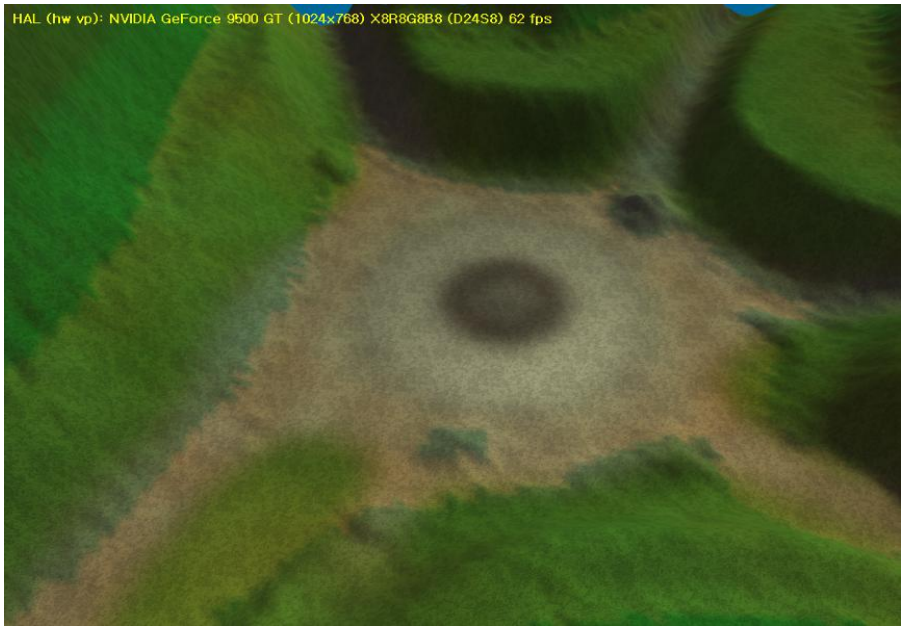
- D3DXPlaneFromPoints() 함수
  - 이 함수는 3점을 이용해 평면의 방정식을 구하는데 이 때 법선 벡터 x,y,z는 평면의 a,b,c에 대응

```
D3DXPLANE t;
D3DXPlaneFromPoints(&t
                    , &v0, &v1, &v2);

// 법선 벡터
D3DXVECTOR3 n(t.a, t.b, t.c);
```

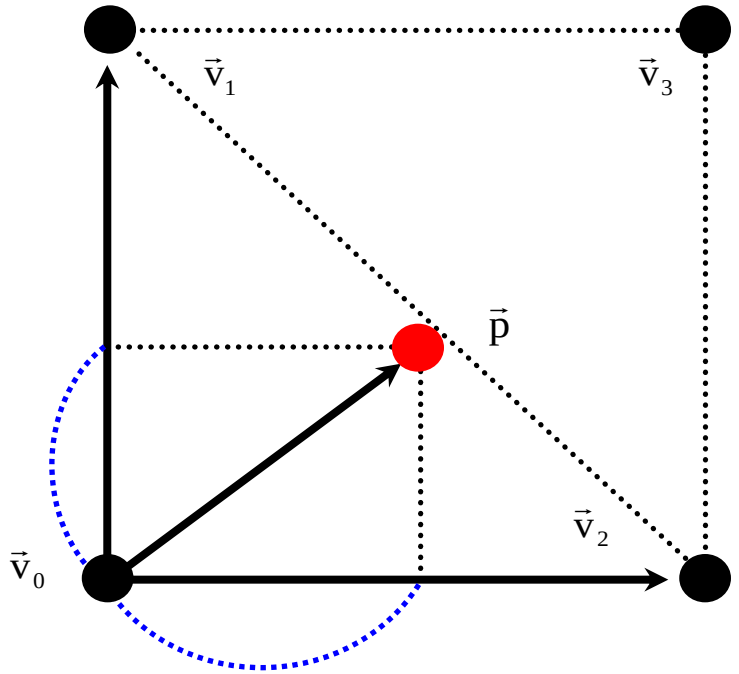


- 정점의 법선 벡터
  - 하나의 정점과 인접한 삼각형 법선의 평균값으로 정점의 법선 벡터 설정

```
for(z=1; z<m_TileN; ++z)
{
    for(x=1; x<m_TileN; ++x)
    {
        n = z * nVtxT + x;

        ...

        Normal = D3DXVECTOR3(0,0,0);

        for(k=0; k<6; ++k)
        {
            ...
            CalculateNormal(&Nor, &v0, &v1, &v2);
            Normal +=Nor;
        }

        Normal /=6.f;
        D3DXVec3Normalize(&Normal,&Normal);

        m_pVtx[n].n = Normal;
    }
}
```



- 높이 맵 위의 높이 구하기



$$\Delta z = \frac{(\vec{p}.z - \vec{v}_0.z)}{\text{Tile Width}}$$

$$\Delta x = \frac{(\vec{p}.x - \vec{v}_0.x)}{\text{Tile Width}}$$

$$\therefore \ \vec{p} = (\vec{v}_2 - \vec{v}_0) * \Delta x + (\vec{v}_1 - \vec{v}_0) * \Delta z$$

높이 구하기 프로그램

```
INT CMcField::GetHeight(D3DXVECTOR3* pOut, const D3DXVECTOR3* pIn)
{
    D3DXVECTOR3 vIn = *pIn;
    int nX = int( pIn->x/ m_TileW );
    int nZ = int( pIn->z/ m_TileW );
    // 실패
    if(nX<0|| nX>=m_TileN || nZ<0|| nZ>=m_TileN)
        return -1;
    //  1------3
    //  .\     |
    //  .  \   |
    //  .   \  |
    //  0------2
    int _0 = nX +0 + (m_TileN+1)*(nZ+0);
    int _1 = nX +0 + (m_TileN+1)*(nZ+1);
    int _2 = nX +1 + (m_TileN+1)*(nZ+0);
    int _3 = nX +1 + (m_TileN+1)*(nZ+1);
    FLOAT   dX = vIn.x - nX * m_TileW;
    FLOAT   dZ = vIn.z - nZ * m_TileW;
    D3DXVECTOR3 vcX;
    D3DXVECTOR3 vcZ;
    D3DXVECTOR3 vcOut;
    if( (dX+dZ) <=m_TileW)  // 아래 쪽 삼각형
    {
        vcX = m_pVtx[_2].p - m_pVtx[_0].p;
        vcZ = m_pVtx[_1].p - m_pVtx[_0].p;
        vcOut = vcX * dX/m_TileW + vcZ * dZ/m_TileW;
        vcOut +=m_pVtx[_0].p;
    }
    else                    // 위 쪽 삼각형
    {
        dX = m_pVtx[_3].p.x - vIn.x;
        dZ = m_pVtx[_3].p.z - vIn.z;
        vcX = m_pVtx[_1].p - m_pVtx[_3].p;
        vcZ = m_pVtx[_2].p - m_pVtx[_3].p;
        vcOut = vcX * dX/m_TileW + vcZ * dZ/m_TileW;
        vcOut +=m_pVtx[_3].p;
    }
    *pOut = vcOut;
    return 0;
}
```