3D Game Programming 22
- Height Field: Picking

afewhee@gmail.com

- 지형 Picking (Height Field) 방법
  - 마우스의 위치를 3차원으로 환원
  - 카메라의 위치와 3차원으로 환원된 마우스의 위치를 충돌 직선으로 설정
  - 위의 직선의 y 값을 0으로 하고 이와 충돌 xz 평면상의 하는 Tile List들을 구함
  - 위에서 구한 Tile들의 삼각형을 직선과 삼각형 충돌 테스트 진행
    - 삼각형으로 평면을 구성하고 이 평면과 직선이 충돌하는 위치 구함
    - 이 위치가 삼각형의 내부에 있는지 검사
  - 직선과 충돌한 삼각형을 Sorting
  - 카메라에서 가장 가까운 삼각형 선택

- 필요한 기술
  - 마우스의 3차원 환원
  - 직선과 삼각형의 충돌
    - 3점으로 구성된 평면의 방정식 구하기
    - 점과 삼각형 충돌
    - D3DXIntersectTri() 함수로 삼각형과 직선의 충돌을 간단하게 구함
  - STL Vector와 Generic 알고리듬의 Sorting

- 3차원 좌표를 2차원으로 변경하는 방법
  - 뷰 행렬, 투영 행렬을 구한다.
  - 뷰포트 행렬을 만든다.
  - 변환 행렬 = 뷰 행렬 * 투영 행렬 * 뷰포트 행렬
  - 정점을 변환한다.
  - 변환된 정점의 x, y를 화면 좌표로 사용한다.

- 오브젝트의 이름 출력
  - 이름은 문자열이고 문자열은 화면에 의존하므로 오브젝트의 위치 3차원 값을 2차원 화면 좌표로 바꾸어야 함
  - 정점의 변환 원리를 이용
  - 3차원 뷰 볼륨 안의 정점은 변환을 거치면 화면 좌표계로 변경되므로 이것을 CPU를 통해서 계산

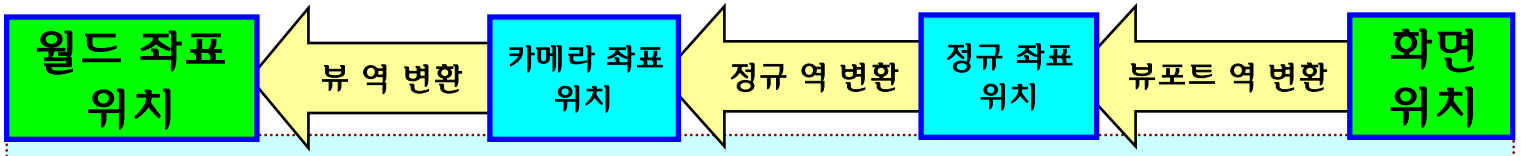
- 프로그램 방법

```
// 뷰포트 행렬을 구하는 것은 DirectX SDK에서 지원이 안되므로 직접 만듦
void D3DXMatrixViewport(D3DXMATRIX* pOut, const D3DVIEWPORT9* pV /*Viewport*/)
    {
    float   fW = 0;
    float   fH = 0;
    float   fD = 0;
    float   fY = 0;
    float   fX = 0;

    float   fM = FLOAT(pV->MinZ);

    fW = FLOAT(pV->Width)*.5f;
    fH = FLOAT(pV->Height)*.5f;
    fD = FLOAT(pV->MaxZ) - FLOAT(pV->MinZ);
    fX = FLOAT(pV->X) + fW;
    fY = FLOAT(pV->Y) + fH;

    *pOut = D3DXMATRIX(  fW,    0.f,  0, 0,
                         0.f,   -fH,  0, 0,
                         0.f,   0.f, fD, 0,
                         fX,    fY, fM, 1);
}
...
D3DXMATRIX  mtView;
D3DXMATRIX  mtProj;

pDev->GetTransform(D3DTS_VIEW, &mtView);
pDev->GetTransform(D3DTS_PROJECTION, &mtProj);

// 뷰포트 행렬 설정
D3DXMATRIX      mtVp;
D3DVIEWPORT9    vp;
pDev->GetViewport(&vp);
D3DXMatrixViewport(&mtVp, &vp);

// 변환 행렬 = 월드 행렬 * 뷰행렬 * 투영 행렬 * 뷰포트 행렬
D3DXMATRIX  mtTMpt  = mtView * mtProj * mtVp;
...
// 정점 변환
D3DXVec3TransformCoord(&vcOut, &vcIn, &mtTMpt);
RECT rc={ int(vcOut.x), int(vcOut.y), 0,0};
...
```

DirectX SDK 함수: D3DXVec3Project()

- 2차원 마우스의 3차원 환원
  - 변환을 이용한 방법
    - 마우스를 뷰포트 역 변환, 투영 역 변환, 뷰잉 역 변환을 거쳐 3차원 좌표로 만든다. == 마우스의 위치를 근접 평면(Near Plan) 안의 위치로 바꾼다.
    - 뷰잉 변환 행렬 * 투영 변환 행렬 * 뷰포트 변환 행렬의 곱을 구하고 이의 역 행렬을 이용하면 뷰 포트, 투영, 뷰잉 변환 행렬의 역 행렬을 다 구하지 않아도 된다.
    - 마우스의 3차원 좌표 = (Mouse X, Mouse Y, 0.f) * (뷰 행렬 * 투영 행렬 * 뷰포트 행렬)$^{-1}$
  - 직접 계산 방법
    - 마우스의 위치를 투영 변환에 맞게 [-1,1]로 정규화한다.
    - 이 값을 뷰 행렬의 역 행렬과 연산을 한다.
    - 카메라의 위치를 더한다.

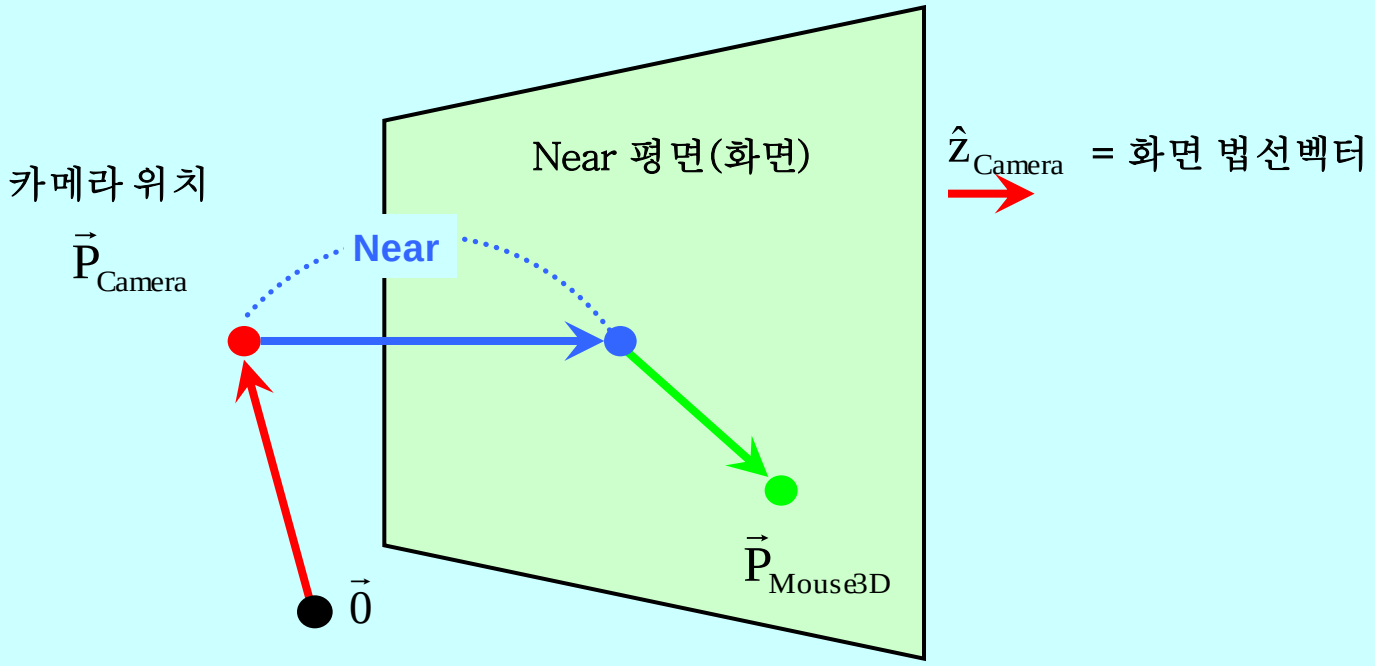
● 변환을 이용한 마우스의 3차원 좌표 프로그램




```
D3DXMATRIX          mtViw;
D3DXMATRIX          mtPrj;
D3DXMATRIX          mtVpt;// 뷰포트 행렬

D3DVIEWPORT9    vp;
GDEVICE->GetViewport(&vp);
D3DXMatrixViewport(&mtVpt, &vp);

GDEVICE->GetTransform(D3DTS_VIEW, &mtViw);
GDEVICE->GetTransform(D3DTS_PROJECTION, &mtPrj);

// 변환 행렬 = 월드 행렬 * 뷰행렬 * 투영 행렬 * 뷰포트 행렬
D3DXMATRIX  mtTMpt  = mtViw * mtPrj * mtVpt;
D3DXMATRIX  mtTMptI;

// 변환 행렬의 역 행렬을 구한다.
D3DXMatrixInverse(&mtTMptI, NULL, &mtTMpt);

// 스크린의 좌표를 설정한다.
//(z=0인 것은 Near 평면의 점들은 투영 변환을 거치면 깊이가 0인 점을 이용)
D3DXVECTOR3         vcScn(MouseX, MouseY, 0.f);

// 앞서 구한 역 행렬을 이용해서 3차원 좌표로 만든다.
D3DXVec3TransformCoord(&vcScn, &vcScn, &mtTMptI);
```

DirectX SDK 함수: D3DXVec3Unproject()

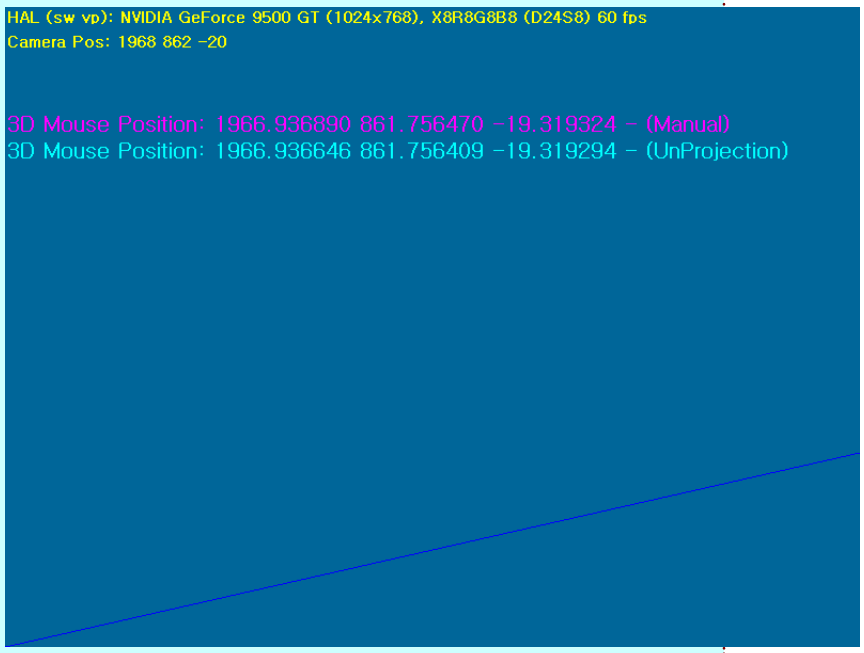
2차원 마우스의 3차원 환원



$$\vec{P}_{Mouse3D} = \vec{P}_{Camera} + Near * \hat{z}_{Camera} + \vec{P}_{UnTMmouse}$$

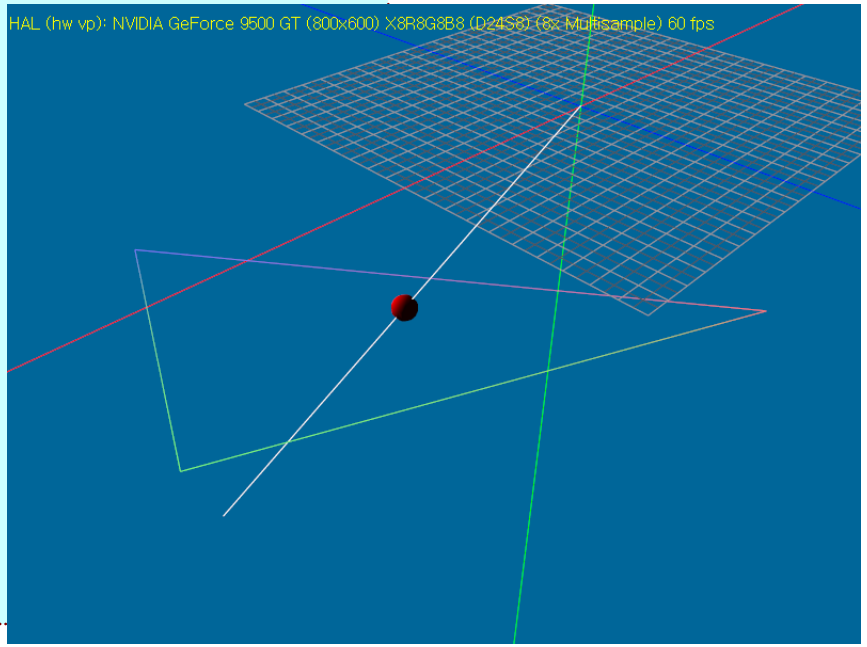
● 직접 계산한 마우스의 3차원 좌표 프로그램

```
D3DXMATRIX      mtViw;
D3DXMATRIX      mtPrj;
D3DVIEWPORT9    vp;

GDEVICE->GetViewport(&vp);

// 뷰 행렬과 투영 행렬을 구한다.
GDEVICE->GetTransform(D3DTS_VIEW, &mtViw);
GDEVICE->GetTransform(D3DTS_PROJECTION, &mtPrj);

// 뷰 행렬의 역 행렬을 구한다.
D3DXMatrixInverse(&mtViwI, NULL, &mtViw);
FLOAT fScnW = vp.Width;          // 화면의 너비
FLOAT fScnH = vp.Height;         // 화면의 높이
FLOAT w = mtPrj._11;
FLOAT h = mtPrj._22;

// 스크린 좌표의 값을 [-1,1]로 정규화
vcScn.x =  ( 2.f * p2.x / fScnW - 1 ) / w;
vcScn.y = -( 2.f * p2.y / fScnH - 1 ) / h;
vcScn.z =  1.f;              // Near Value

// 뷰 행렬의 역 행렬 값들과 연산 (UnViewing Transform)
vcPickRayDir.x  = D3DXVec3Dot(&vcScn, &VEC3(m_mtViwI._11, m_mtViwI._21, m_mtViwI._31));
vcPickRayDir.y  = D3DXVec3Dot(&vcScn, &VEC3(m_mtViwI._12, m_mtViwI._22, m_mtViwI._32));
vcPickRayDir.z  = D3DXVec3Dot(&vcScn, &VEC3(m_mtViwI._13, m_mtViwI._23, m_mtViwI._33));

// 카메라의 위치를 더한다.
vcScn = vcPickRayDir + m_vcEye;
```

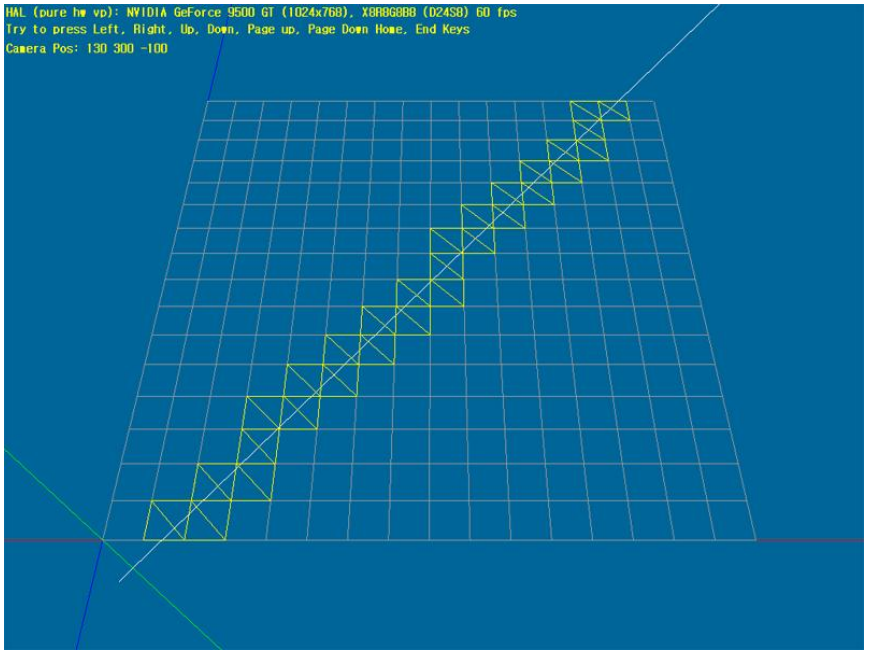


- 직선과 삼각형 충돌
  - D3DXIntersectTri() 함수 이용
    - 충돌 판정, 충돌 위치, 직선의 시작점에서 충돌 점까지의 거리를 구할 수 있음.

```
INT hr;
D3DXVECTOR3 LineBegin     = m_pLine[0].p;                       // 직선의 시작점
D3DXVECTOR3 LineDirection = m_pLine[1].p - m_pLine[0].p; // 직선의 방향
FLOAT       u, v, d;

hr = D3DXIntersectTri(
                &V0
                 , &V1
                 , &V2
                 , &LineBegin
                 , &LineDirection
                 , &u
                 , &v
                 , &d    // Line Begin에서 충돌 지점까지의 거리
                );

// 충돌 위치 구하기
if(TRUE == hr)
    m_vcPick = V0 + u * (V1 - V0) + v * (V2 - V0);
```



- 충돌 테스트할 Tile List 구하기
  - 카메라의 카메라의 위치를 직선의 시작 위치, 카메라의 시선 방향 (z축)으로 하는 직선을 만들 되 y 값은 0으로 한다.
  - 이 직선과 충돌하는 xz평면상의 타일들을 구한다.
  - 각 타일에 해당하는 2개의 삼각형을 벡터에 넣는다.




```
// Picking Tile Structure
struct LcRect
{
    VtxD      pVtx[4];    // Tile Position
};

// Picking structure
struct LcPck
{
    VtxD    pVtx[3];   // Picking Triangle
    FLOAT   fR;        // Picking Distance from Camera
    VEC3    vcP;       // Picking Position
};
```

```
INT Picking(...)
{
   ...
   // Setup Line
   pLine[0] = vcCamPos;
   ...
   // Gether Tile List
   for(int i=0; i<iNx; ++i)
      ...
      if( z>=0 && z<(iNx-1) && i<(iNx-1))
      {
         ...
         if( ...)
            vRect.push_back(rc);
      ...
   //There are Tile Lists
   if(!vRect.empty())
   {
      ...
      for(int i=0; i<iSize; ++i)
      {
         V0 = vRect[i].pVtx[0].p;
         V1 = vRect[i].pVtx[1].p;
         V2 = vRect[i].pVtx[2].p;

         // Triangle and Line Collision Test
         if( D3DXIntersectTri( &V0, &V1, &V2, &vcCamPos, &vcRayDir, &U, &V, &D))
         {
            // Pick Position
            Pck.vcP = V0 + U * (V1-V0) + V * (V2-V0);
            ...
            vPck.push_back( Pck );
         }

         V0 = vRect[i].pVtx[3].p;
         V1 = vRect[i].pVtx[2].p;
         V2 = vRect[i].pVtx[1].p;
         if( D3DXIntersectTri( &V0, &V1, &V2, &vcCamPos, &vcRayDir, &U, &V, &D))
         {
            Pck.vcP = V0 + U * (V1-V0) + V * (V2-V0);
            ...
         }
      }
      // Sorting
      if(!vPck.empty())
      {
         sort(vPck.begin(), vPck.end(), TsrtL<LcPck >());
         ...
         return 1;
```