3D Game Programming 11
- ASE Animation

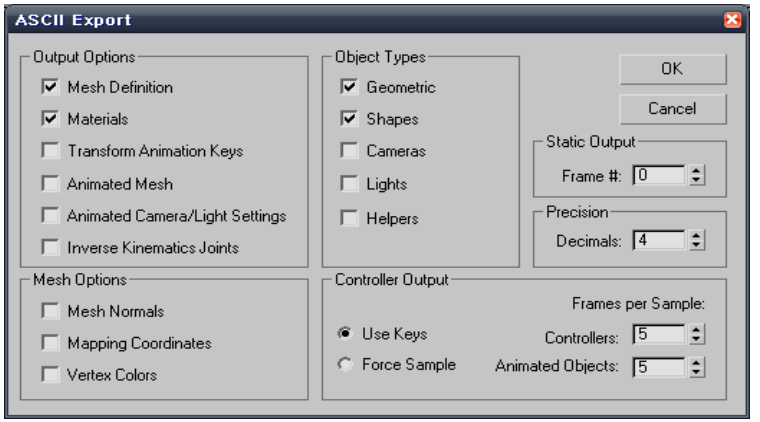
afewhee@gmail.com

- ASE 구조
- Geometry Parsing
  - Geometry
- Texture Parsing
  - Material
  - Texture
- Rigid Body Animation
  - Scene
  - Animation
- X-File Animation

- ASE Parsing
  - Parsing: 의미 분석
  - ASCII로 구성된 텍스트 내용에 렌더링에 관련된 의미를 부여

- ASE Exporting
  - 3D Max에서 작업한 Object를 ASE 형식으로 저장

  - 내용
    - Material
    - Geometry
    - Animation



- Program
  - ASE 파일에 대한 뷰어(Viewer) 작업
    → 이후 캐릭터 툴(Character Tool)로 발전

- Scene
  - File Name이름, Start, End Frame 등 Frame에 관련된 내용
  - Frame Speed: Frame/Second
  - Tick: Max에서 사용하는 시간 millisecond 보다 정밀
    Frame Speed= 30, Tick/Frame = 160 이면 1초 = 4800 Tick

- Material List
  - Total 재질 수
  - 텍스처 매핑에 대한 정보 ➔ Geometry는 이를 인덱스로 참고 ("MATERIAL_REF")
  - 텍스처 파일 이름이 한글이면 파일 이름이 제대로 출력이 안됨

- Geometry ➔See Geometry

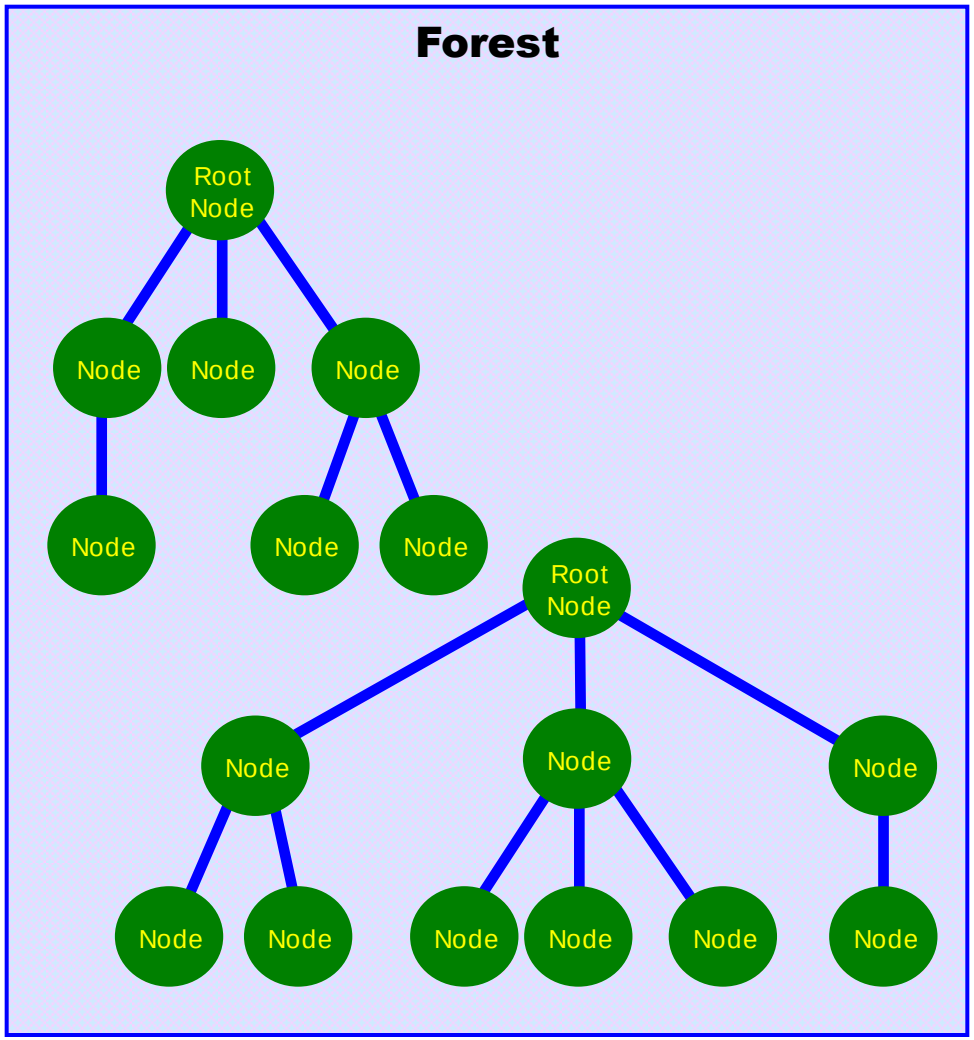
- Shape: Line으로 구성된 객체
- Camera: 카메라 움직임에 대한 연출을 기록
- Light Object 조명에 대한 연출: Omni (Point Light), Directional, Target (Spot Light)

- 게임에서는 주로 Scene, Material List, Geometry를 사용
  ➔ 이외의 내용은 따로 작업

- Geometry
  - Node로 구성
    - Node Name
    - Parent Node
  - World 행렬
    - Node의 크기(S), 회전(R), 이동(T)
  - Object를 구성하는 Mesh
    - vertex List
    - Face (Index) List
    - Normal vector List
    - Texture Coordinate List
    - Texture Face List
  - Animation
  - Material Index

- Node
  - Max의 Object는 계층적 자료구조인 Tree 자료 구조들이 결합된 Forest 형식으로 구성
  - 부모 노드 이름
  - 자신의 노드 이름

  - 링크의 순서대로 파일에 노드가 기록되어 있음
    ➔ 자료구조를 Tree로 만들 필요가 없음



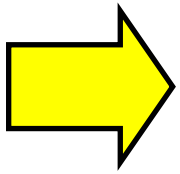
2. ASE Parsing

- 해석 방법
  - 자료구조
    - ASE의 구조와 동일하게 구성
  - List 구조의 자료 블록
    - '*' + '자료 구조 이름' + '{'
    - 자료의 끝: '}'
    - 필요에 따라 while 문을 사용해서 자료를 해석
  - 개별 자료
    - '*' + '자료 구조 이름' + '{'

Ex) ASE 자료 구조 설정

ASE File

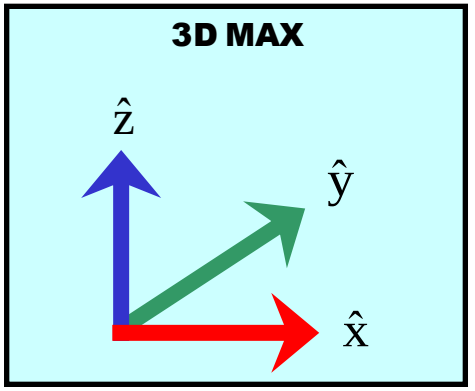
```
*GEOMOBJECT {
   *NODE_NAME "Pyramid01"
   *MESH {
      *MESH_NUMVERTEX 5
      *MESH_NUMFACES 7
      *MESH_VERTEX_LIST {
         *MESH_VERTEX   0  19.6581   -24.7863   66.1019
         *MESH_VERTEX   1  -28.2051  -68.6610   0.0050
         *MESH_VERTEX   2  67.5214   -68.6610   0.0050
         *MESH_VERTEX   3  67.5214   19.0883    0.0050
         *MESH_VERTEX   4  -28.2051  19.0883    0.0050
      }
      *MESH_FACE_LIST {
         *MESH_FACE   0:   A:   0 B:   1 C:   2 AB:
         *MESH_FACE   1:   A:   0 B:   2 C:   3 AB:
         *MESH_FACE   2:   A:   0 B:   3 C:   4 AB:
         *MESH_FACE   3:   A:   0 B:   4 C:   1 AB:
         *MESH_FACE   4:   A:   1 B:   5 C:   2 AB:
         *MESH_FACE   5:   A:   2 B:   5 C:   3 AB:
      }
   }
}
```

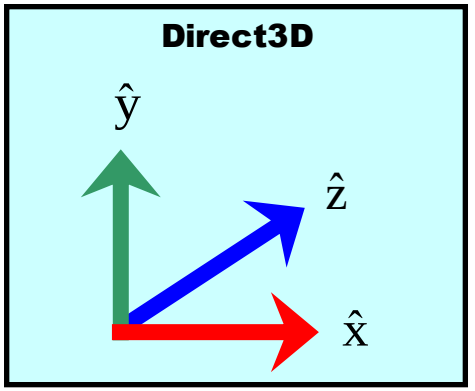


```
//ASE Structure

struct AseVtx
{
   FLOAT x, y, z;
   AseVtx() : x(0), y(0), z(0){}
};

struct AseFce
{
   WORD a, b, c;
   AseFce() : a(0), b(0), c(0){}
};

struct AseGeo
{
   char     sNodeName[64];
   int      iNumVtx;
   int      iNumFce;
   AseVtx*  pLstVtx;
   AseFce*  pLstFce;
};
```

- 뷰어 (Viewer) 기본 요소
  - 인풋, 카메라 클래스
  - 오브젝트의 크기를 눈으로 판별할 수 있는 Grid 필요

- 위치에 대한 자료 해석
  - 3D Max는 오른손 좌표계 사용
    ➔ Direct3D 왼손 좌표계로 전환(y, z 교환)






```
//ASE File
*MESH_VERTEX  0 19.6581  -24.7863  66.1019
```

```
//Program
Float3 (19.6581, 66.1019 , -24.7863)
```

```
INT        nIdx=0;
FLOAT      x=0.F, y=0.F, z=0.F;
sscanf(sLine, "%*s %d %f %f %f", &nIdx, &x, &y, &z);

m_pGeo[nGeoIdx].pLstVtx[nIdx].x = x;
m_pGeo[nGeoIdx].pLstVtx[nIdx].y = z;
m_pGeo[nGeoIdx].pLstVtx[nIdx].z = y;
```

- Face (Index)에 대한 자료 해석
  - 3D Max는 오른손 좌표계 사용
    ➔ Direct3D 왼손 좌표계로 전환(c, b 교환)

```
//ASE File
*MESH_FACE   6:   A:   3 B:   5 C:   4
```

```
//Program
WORD3 (3, 4, 5)
```

```
INT       nIdx=0;
INT       a=0, b=0, c=0;

sscanf(   sLine
          , "%*s %d: %s %d %s %d %s %d"
          , &nIdx
          , sTmp1, &a, sTmp1, &b, sTmp1, &c);

m_pGeo[nGeoIdx].pLstFce[nIdx].a = a;
m_pGeo[nGeoIdx].pLstFce[nIdx].b = c;
m_pGeo[nGeoIdx].pLstFce[nIdx].c = b;
```


- ASE Class에 대한 추상화

- Model Interface

```
struct ILcMdl
{
    virtual ~ILcMdl(){};
    virtual INT         Create(void* pDev, void* sFile)=0;
    virtual void        Destroy()=0;
    virtual  INT        FrameMove()=0;
    virtual void        Render()=0;
};
```

- ASE 객체 생성

```
INT LcAse_Create(char* sCmd
                , ILcMdl** pData    // Output data
                , void* pDev        // Device
                , void* sName   = NULL   // Model File Name
                , void* pOriginal= NULL   // Original ILcMdl Pointer for Clone
                , void* p4=NULL       // Not Use
                , void* p5=NULL       // Not Use
                );
```

- ASE 클래스

```
class CLcAse : public ILcMdl
```


- While Loop로 개수 파악 ← Geometry 숫자는 기록이 안되어 있음

```
while(!feof(fp))
{
    ...
    if(0 == _strnicmp(sLine, "*GEOMOBJECT {", strlen("*GEOMOBJECT {") ))
      ++m_nGeo;
    ...
}
...
// 파일의 끝이므로 파일 포인터를 처음으로 이동
fseek(fp, 0, SEEK_SET);
```

- 지오메트리 생성

```
m_pGeo = new AseGeo[m_nGeo];
INT nGeoIdx = -1;
AseGeo*          pGeo       = NULL;
```

- 지오메트리 해석

```
while(!feof(fp))
{
    if(0 == _strnicmp(sLine, "*GEOMOBJECT {", strlen("*GEOMOBJECT {") ))
    {
      ++nGeoIdx;
      pGeo = &m_pGeo[nGeoIdx];
      ...
      pGeo->...
```


- Material
  - Geometry에 적용할 텍스처 파일 이름 기록
    Ex) *BITMAP "d:₩psd₩mong.jpg"
  - Geometry 끝에 사용 Material 인덱스 기록
    Ex) *MATERIAL_REF 3
- Texture 좌표에 대한 자료 해석
  - 3D Max는 S.T. 좌표계 사용
    ➔ Direct3D U.V. 좌표계로 전환(x,1.f-y)

```
//ASE File
*MESH_TVERT 3 0.4811 0.6722 0.1822
```

```
//Program
FLOAT U = 0.4811, V= 1.0 - 0.6722
```

```
INT     nIdx=0;
FLOAT  u=0.F, v=0.F, w=0.F;

sscanf(sLine, "%*s %d %f %f %f", &nIdx, &u, &v, &w);

pGeo->pLstTvtx[nIdx].u = u;
pGeo->pLstTvtx[nIdx].v = 1.0f-v;
pGeo->pLstTvtx[nIdx].w = w;
```


- Texture Face (T-Face)
  - 3D Max는 하나의 정점을 여러 텍스처 좌표가 공유해서 사용
  - Texture 좌표 수 >= 정점 좌표 수
  - 정점의 수를 텍스처 좌표 수 만큼 늘림
  - 텍스처 Face List와 정점 Face List 를 비교해서 새로 만들어진 정점의 위치 결정

Ex) T-Face Reading

```
INT    nIdx=0;
INT    a=0, b=0, c=0;

sscanf(sLine, "%*s %d %d %d %d", &nIdx, &a, &b, &c);

pGeo->pLstTfce[nIdx].a = a;
pGeo->pLstTfce[nIdx].b = c;
pGeo->pLstTfce[nIdx].c = b;
```


※ T-Face를 기준으로 한 정점 결정

- 새로운 정점의 수를 텍스처 좌표 수로 정한다.
- 새로운 정점에 텍스처 좌표를 먼저 설정한다.
- T-Face Index와 정점의 Face Index를 비교해서 Face Index에 해당하는 위치를 새로운 정점의 위치에 복사한다.
- 최종 Index List 는 T-Face로 한다.

- Face List와 T-Face List의 인덱스 비교

```
//ASE FILE Face
*MESH_FACE_LIST {
*MESH_FACE    0:  A:    0 B:   58 C:    1
*MESH_FACE    1:  A:   59 B:    1 C:   58
...
*MESH_FACE  479:  A:  169 B:  170 C:  120
*MESH_FACE  480:  A:  245 B:  120 C:  170
*MESH_FACE  481:  A:  170 B:  171 C:  245
*MESH_FACE  482:  A:  246 B:  245 C:  171
...
*MESH_FACE  487:  A:  122 B:  144 C:   18
}
```

```
// ASE FILE- T-Face
*MESH_TFACELIST {
*MESH_TFACE  0    108   79   255
*MESH_TFACE  1    256   255  179
...
*MESH_TFACE  479  427   428  520
*MESH_TFACE  480  521   520  428
*MESH_TFACE  481  428   429  521
*MESH_TFACE  482  522   521  429
...
*MESH_TFACE  487  396   395  398
}
```

- 프로그램 방법

```
// UV먼저 설정
for(int j=0; j< pGeo->nTvtx; ++j)
{
    pVtxR[j].u = pGeo->pTvtx[j].u;
    pVtxR[j].v = pGeo->pTvtx[j].v;
}
```

```
// Vertex Setting
for(int n=0; n<pGeo->nFce; ++n)
{
    INT nT = 0;
    INT nV = 0;

    nT = pGeo->pTFce[n].a;      // T-face U V 인덱스를 가져온다.
    nV = pGeo->pFce[n].a;       // Vertex 버퍼에서 정점의 위치를 가져온다.
    pVtxR[nT].p = pGeo->pVtx[nV].p;

    nT = pGeo->pTFce[n].b;
    nV = pGeo->pFce[n].b;
    pVtxR[nT].p = pGeo->pVtx[nV].p;

    nT = pGeo->pTFce[n].c;
    nV = pGeo->pFce[n].c;
    pVtxR[nT].p = pGeo->pVtx[nV].p;
}
```


Animation 종류

Rigid Body (강체)

- 강체란 외부에서 물체에 힘을 가해도 크기가 변형되지 않는 물체로 Rigid Body Animation 은 이러한 점을 이용해서 Geometry를 구성하는 모든 정점 사이의 거리 비율이 변환(Transform) 후에도 변하지 않는 Animation
- 오브젝트에 하나의 행렬을 적용해서 변환을 수행
- 3D의 변환은 Rigid Body라 할 수 있음

Skinning

- 오브젝트에 변환을 적용하면 변환 전과 변환 후 정점 사이의 비율이 달라짐
- 구현방법 한 정점에 변환에 관련된 여러 행렬을 적용

Animation 구현 방법

Key Frame Animation

- 일정한 시간간격으로 정점의 모든 좌표를 기록
- 계산 량이 적어 속도에 이점이 있으나 메모리는 정점의 수* 프레임 간격에 비례
- Ex) MD2, MD3 Model

Bone Animation

- Key Frame처럼 모든 정점의 변환 결과를 기록하는 것이 아니라 애니메이션을 구성하는 Bone의 행렬을 시간에 따라 기록
- 골격체(Skeleton)를 이용한 애니메이션으로 하나의 Bone 행렬에는 하나의 Geometry가 대응
- Key Frame 보다 메모리를 적게 차지
- 계층적으로 Bond을 구성
- Bone이 많을 수록 Animation에 대한 연산 량이 증가
- Bone과 Bone사이의 연결부위에 Crack이 발생

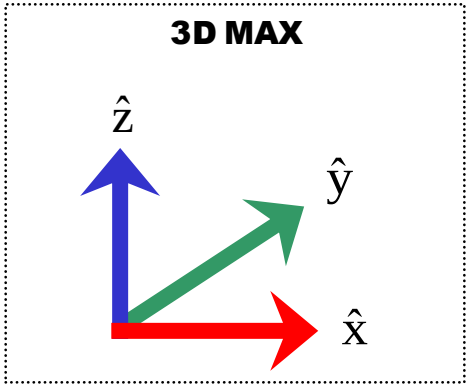
- 대부분의 게임에서 가장 많이 사용되는 애니메이션

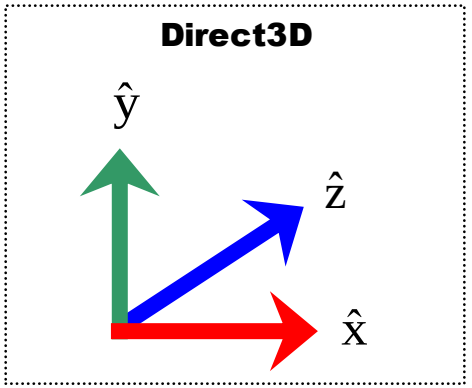
- Animation 구현 방법
  - Skinning Animation
    - Bone Animation의 단점을 극복하기 위해 하나의 정점에 여러 행렬을 적용한 애니메이션으로 행렬에 각각 그 영향력에 따라 다른 Weight(비중)을 주어 전체 변환 행렬을 만듦
    - 행렬이 하나면 Bone Animation과 동일
    - 이음 새 부분에서 Crack을 없앨 수 있음
    - 행렬 계산을 CPU를 이용한 Software 계산을 하지 않고 DirectX의 고정 파이프라인을 이용한다면 하나의 정점에 네 개까지 행렬을 연결해서 사용할 수 있음
    - 정점 쉐이더 (Vertex Shader)를 이용하면 간단하게 계산됨
    - 그래픽 카드의 성능에 많이 의존
  - Direct3D의 X-File은 Skinning Animation이 가능
    - Ex) DirectX SDK Tiny 예제

- Scene 해석
  - SCENE_FILENAME : Export한 파일 이름
  - SCENE_FIRSTFRAME : 시작 프레임 인덱스
  - SCENE_LASTFRAME : 마지막 프레임 인덱스
  - SCENE_FRAMESPEED : 초당 프레임 수
    - = Frame / Second
    - 30 Frame ➔ 33.333ms
  - SCENE_TICKSPERFRAME
    - 프레임당 Tick: Max는 millisecond 보다 더 정밀한 Tick 단위 사용
    - Tick은 이후에 millisecond로 환원해서 사용
    - 160 ➔ 위에서 30 Frame 이면 1초를 160* 30 = 4800 Tick

- Node_TM 해석
  - Node Transform Matrix (변환 행렬)
  - Geometry(Node)가 가지는 하나의 월드 행렬
  - 행렬 또는 크기, 위치, 회전을 따로 해석

  - Node_TM 해석 방법
    - Max는 오른손 좌표계를 사용
      ➔ Direct3D는 왼손 좌표계. 변환 필요
    - TM_ROW1 부분과 TM_ROW2 부분 교환
    - Y와 Z를 교환





*TM_ROW0	-0.1225	-0.3713	0.9204
*TM_ROW1	-0.4945	-0.7812	-0.3810
*TM_ROW2	0.8605	-0.5018	-0.0879
*TM_ROW3	-0.0443	-0.2532	10.2929

```
D3DXMATRIX TM_NODE(
-0.1225,   0.9204,    -0.3713 ,   0,
 0.8605,  -0.0879,    -0.50189,   0,
-0.4945,  -0.3810,    -0.78120,   0,
-0.0443,  10.2929,    -0.25329,   1
);
```

- Node TM
  - 3D Max가 정한 Node의 월드 변환 행렬
  - Node TM World = Node TM Local * Parent TM World

- Node TM Local
  - Parser가 구현해야 하는 지역 좌표계에서의 변환(크기, 회전, 이동)행렬
  - ASE의 Node TM은 전부월드 좌표계로 구성되어 있음 ➔ 모델 좌표계로 전환 필요

  - Node TM Local = Node TM World * (Parent TM World)$^{-1}$

```
D3DXMATRIX mtPrn = pGeoPrn->TmInf.mtW;

D3DXMATRIX mtPrnI; // 부모 월드 행렬의 역행렬

D3DXMatrixInverse(&mtPrnI, NULL, &mtPrn);

// TM Local = TM World * (Parent World)-1
pGeo->TmInf.mtL = pGeo->TmInf.mtW * mtPrnI;
```


- Animation은 크기, 회전, 이동을 따로 해석
  - Geometry가 이동만 있는 경우 이동만 기록. 회전, 크기 도 마찬가지

- *TM_ANIMATION {
  - *NODE_NAME: 해당 애니메이션 Bone 이름
  - 이동
    - "*CONTROL_POS_TRACK {": 이동에 대한 애니메이션 Track
    - "*CONTROL_POS_SAMPLE": Geometry의 X,Y,Z 위치 변환 값
  - 크기
    - "*CONTROL_SCALE_TRACK {": 크기 변환에 대한 애니메이션 Track
    - *CONTROL_SCALE_SAMPLE: X, Y, Z에 대한 크기 변환 값
  - 회전
    - *CONTROL_ROT_TRACK {: 회전에 대한 애니메이션 Track
    - *CONTROL_ROT_SAMPLE: 회전 변환 값 (Quaternion: x,y,z,w)
    - 회전 변환은 절대적인 기준을 이용하면 Euler Angle 등을 이용해야 하는데 필연적으로 Gimbal lock 문제 발생
    - 3D Max는 회전에 대해서 만큼 이전 값에 대한 상대적인 값으로 기록하며 Quaternion을 사용 ➔ 사원수를 누적해서 사용

- TM_ANIMATION 위치, 크기 변환 해석: y, z를 교환

- TM_ANIMATION Quaternion 해석
  - Max는 사원수의 축(x,y,z), 그리고 각도(w: radian)를 기록하고 있다.
  - 회전에서 x, y, z, w 순으로 값을 읽는다.
  - 마지막 w 값은 각도(radian) 이므로 이것에서 sin, cos을 구한다.
  - 최종 사원수에 다음과 같이 대입한다.
    - 사원수.x = sin(w/2.f) * x;
    - 사원수.y = sin(w/2.f) * y;
    - 사원수.z = sin(w/2.f) * z;
    - 사원수.w = cos(w/2.f);

```
*CONTROL_ROT_SAMPLE   960   0.0749   -0.0561   -0.9956   0.0723
```

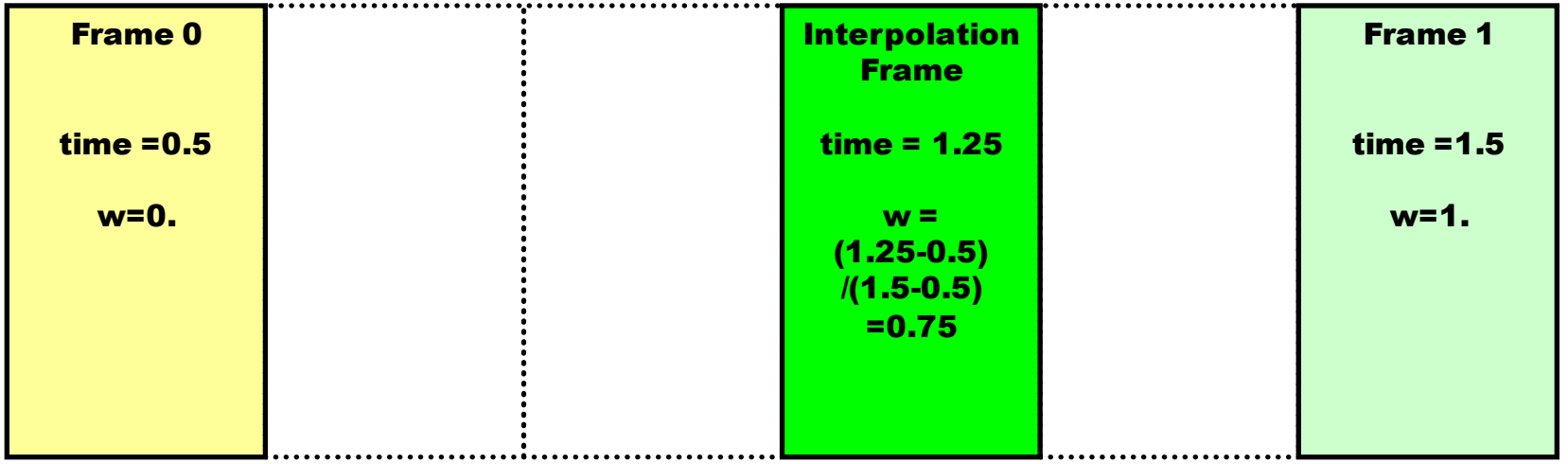
```
D3DXQUATERNION          q1;
INT                     nTrck;
FLOAT    x=0.F, y=0.F, z=0.F, w=0.F;

sscanf(sLine, "%*s %d %f %f %f %f", &nTrck, &x, &y, &z, &w);

q1.x = sinf(w/2.0f) * x;
q1.z = sinf(w/2.0f) * y;
q1.y = sinf(w/2.0f) * z;
q1.w = cosf(w/2.0f);
```


- TM_ANIMATION Quaternion 누적
  - 절대적인 기준을 사용해야 하므로 사원수를 누적

```
//STL을 이용하는 경우에 사원수 누적

INT iSize = pGeo->vRot.size();
if(0==iSize)
{
        AseTrack        trck(nTrck, q1.x, q1.y, q1.z, q1.w);
        pGeo->vRot.push_back(trck);
}
else
{
        D3DXQUATERNION          q2;
        D3DXQUATERNION          q3;
        AseTrack* ptrck = &pGeo->vRot[iSize-1];
        q2.x = ptrck->x;
        q2.y = ptrck->y;
        q2.z = ptrck->z;
        q2.w = ptrck->w;
        D3DXQuaternionMultiply(&q3, &q2, &q1);
        AseTrack        trck(nTrck, q3.x, q3.y, q3.z, q3.w);
        pGeo->vRot.push_back( trck );
}
```


- 보간 (Interpolation)
  - 컴퓨터 마다 실행 속도가 다르므로 프레임과 프레임 사이를 보정해서 장면을 연출
  - 선형 보간 (Linear Interpolation)
    - 가장 간단한 보간 방법

- 선형 보간 방법
  - 선형 보간을 위해서 두 프레임 사이의 w 값을 계산



이동에 대한 Interpolation 구현

- w = (현재 프레임 – 현재 프레임보다 작거나 같은 프레임) /
  (현재 프레임보다 작거나 같은 프레임 - 다음 프레임)

Ex)

```
FLOAT w = (nFrame- pGeo->vTrs[nIdx].nF)/
          (pGeo->vTrs[nIdx+1].nF- pGeo->vTrs[nIdx].nF);

D3DXVECTOR3 p, p1, p2;
p1.x = pGeo->vTrs[nIdx].x;
p1.y = pGeo->vTrs[nIdx].y;
p1.z = pGeo->vTrs[nIdx].z;

p2.x = pGeo->vTrs[nIdx+1].x;
p2.y = pGeo->vTrs[nIdx+1].y;
p2.z = pGeo->vTrs[nIdx+1].z;
```

```
p = (1-w)*p1  + w* p2; //or p1 + w * (p2-p1);
```

회전에 대한 Interpolation 구현

Ex)

```
FLOAT w = (nFrame - pGeo->vRot[nIdx].nF)/
          (pGeo->vRot[nIdx+1].nF- pGeo->vRot[nIdx].nF);

D3DXQUATERNION q, q1, q2;
q1.x = pGeo->vRot[nIdx].x;
q1.y = pGeo->vRot[nIdx].y;
q1.z = pGeo->vRot[nIdx].z;
q1.w = pGeo->vRot[nIdx].w;

q2.x = pGeo->vRot[nIdx+1].x;
q2.y = pGeo->vRot[nIdx+1].y;
q2.z = pGeo->vRot[nIdx+1].z;
q2.w = pGeo->vRot[nIdx+1].w;

// 사원수 보간
q = (1-w) * q1  + w * q2; //or q = q1  + w * (q2-q1);

// 함수 사용
D3DXQuaternionSlerp(&q, &q1, &q2, w);

// 사원수 보간 후 이를 회전 행렬로 설정
D3DXMatrixRotationQuaternion(&mtA, &q);
```


- Tree 형식으로 구성된 애니메이션 구현

  - 크기 행렬, 회전 행렬, 이동 행렬을 보간을 통해서 구한다.
  - 지역 행렬(Local TM)을 구한다.

Local TM =

Scaling TM * Rotation TM * Translation TM

  - 부모 행렬을 곱해서 Geometry의 월드 행렬을 만든다.

World TM = Local TM * Parent World TM

- ASE Parser의 주요 내용
  - 위치를 x, z, y로 해석한다.
  - 위치에 대한 삼각형 인데스는 a, c, b 이다.
  - 텍스처 좌표는 u, 1.0f – v 이다.
  - 텍스처 좌표 인덱스는 a, c, b 이다.
  - 위치는 텍스처 좌표보다 같거나 작다.
  - 텍스처 좌표 인덱스와 위치 인덱스 수는 같다.
  - 텍스처가 있다면 정점 인덱스 대신 텍스처 인덱스를 사용한다.
  - 정점을 구성하려면 k 번째 텍스처 좌표 인덱스에서 텍스처 버퍼 번호를 얻어 uv 좌표를 얻고, k 번째 정점 위치 인덱스에서 위치 버퍼 번호를 얻어 정점 위치를 얻는다.
  - TM은 2번째와 3번째 행을 교환한 다음, 2번째 열과 3번째 열 또한 교환한다.
  - ASE는 월드 행렬로 구성되어 있어서 자신의 지역 행렬 = (자신의 월드 행렬) * (부모의 월드 행렬의 역 행렬)로 구한다.
  - 지역 좌표계 정점 위치 = (ASE에서 구한 위치) * (지오메트리 월드 행렬 역 행렬)
  - 애니메이션의 회전 값은 사원수로 저장되어 있으며 누적 값이 아닌 이전 프레임과 상대적인 값이다. ➔ 사원수 누적을 한다.
  - 회전은 사원수 보간을 이용한다.