ASE Animation

3D Game은 정점을 통해서 화면의 장면을 연출합니다. 정점들은 사람이 하나하나 손으로 만들 수도 있지만 이런 방법은 분명히 한계가 있고, 대부분 3D 소프트웨어이라는 것을 통해서 그래픽을 담당하는 디자이너들을 통해서 만들어 집니다.
맥스(3DS Max)와 Mayas는 게임에서 사용되는 대표적인 소프트웨어로 이들 소프트웨어는 원래 3D 애니메이션영상을 위해서 만든 소프트웨어이지만 게임에서도 응용할 수 있어 지금까지 게임 개발에서 사용되고 있습니다.
3D 애니메이션과 3D 게임 둘 다 3D를 사용하지만 애니메이션은 3D를 화면의 Pixel이라는 공간에 집중하는 반면 게임은 Pixel 공간뿐만 아니라 시간까지 대상으로 삼고 있습니다. 따라서 이러한 소프트웨어로 만든 결과물을 게임에 필요한 데이터만 추출해서 사용합니다. 이와 같은 목적으로 만든 프로그램이 익스포터(Exporter)입니다.
익스포터는 독립적으로 실행되지 않고, 3D 소프트웨어에서 일부 기능을 위임 받아 실행되는데 이렇게 메인 프로그램 안에서 메인 프로그램의 내용을 가지고 독립적으로 실행하는 프로그램을 플러그인(Plug-in)이라 합니다.
프로그래머가 익스포터를 직접 만들기도 하지만 잘 만들어진 공개용 익스포터를 사용하거나 맥스 등에서 제공하는 내장된 플러그인을 사용하기도 합니다. 대표적으로 3DS, FBX, ASE 등이 있습니다.

이들 내장된 플러그인 중에서 ASE(ASCII Scene Export)는 static 3D 모델과 강체(Rigid body) 애니메이션 모델 데이터를 text로 저장해 주기 때문에 플러그인 제작에서 디버그 용으로도 자주 사용되고 특히, 3D 애니메이션을 처음 접하는 분들에게 무척 유용한 익스포터라 할 수 있습니다.
또한 맥스 SDK를 설치하면 ASE의 소스 코드가 포함되어 있어서 ASE 자체를 자신의 프로그램에 맞게 수정하거나 문자열 형식을 이진(binary) 파일 형태로 저장할 수 있는 옵션 등을 추가할 수도 있습니다.

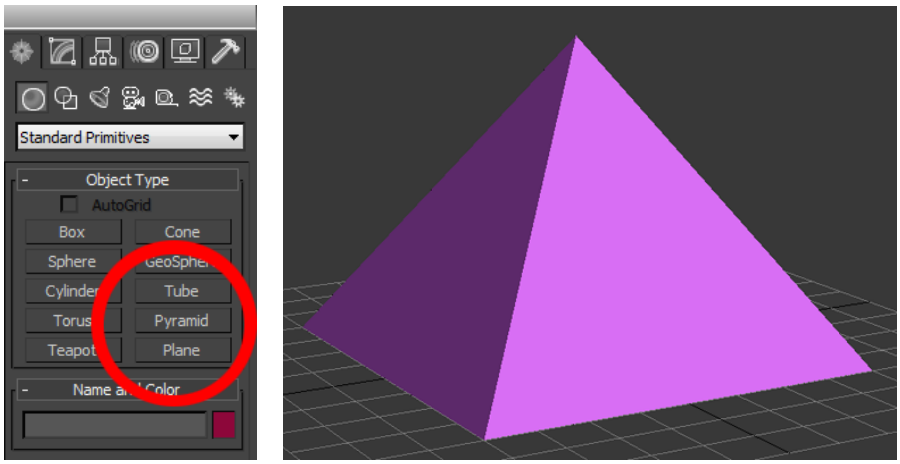
ASE는 독자적인 플러그인과 애니메이션 개발의 시작 단계에서 맥스 데이터의 구조를 이해하고 자료구조를 설정하는데 많은 도움을 주기 때문에 ASE는 일종의 애니메이션과 플러그인의 기초과정이라 볼 수 있습니다.

본 강좌는 ASE로 저장된 3D 모델 데이터를 해석하고 화면에 렌더링 해보는 것을 목표로 하겠습니다.

1 ASE Parsing

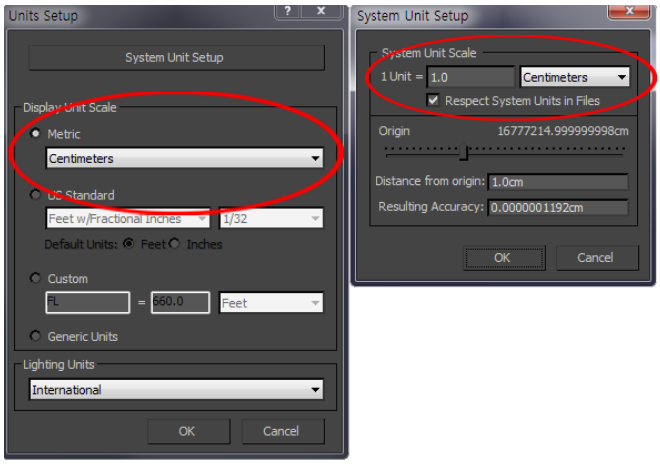
1.1 간단한 3D 모델

ASE 파일에 포함된 데이터를 렌더링 하기 위해서 간단한 Geometry를 만들어 파싱(Parsing)을 하기 위해서 맥스(3DS Max)의 'Create 창' 이나 '메뉴->Create->Standard Primitives'에서 "pyramid"를 선택합니다.



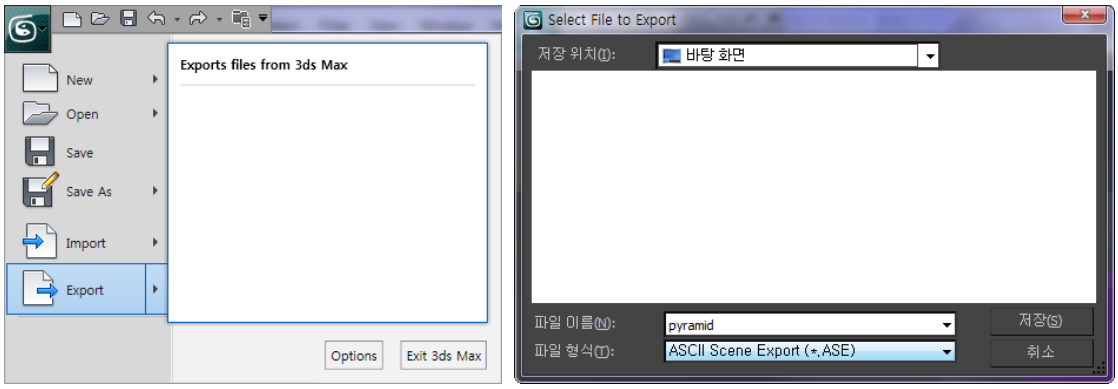
<3DS Max Geometry Tool, pyramid>

맥스 작업에 주의할 것은 System Unit와 Display Unit 가 다르면 작업할 때 수치와 익스포트(Export) 된 수치가 달라 질 수 있습니다. '메뉴->Customize->Units Setup'에서 이 둘을 일치시켜 놓습니다.

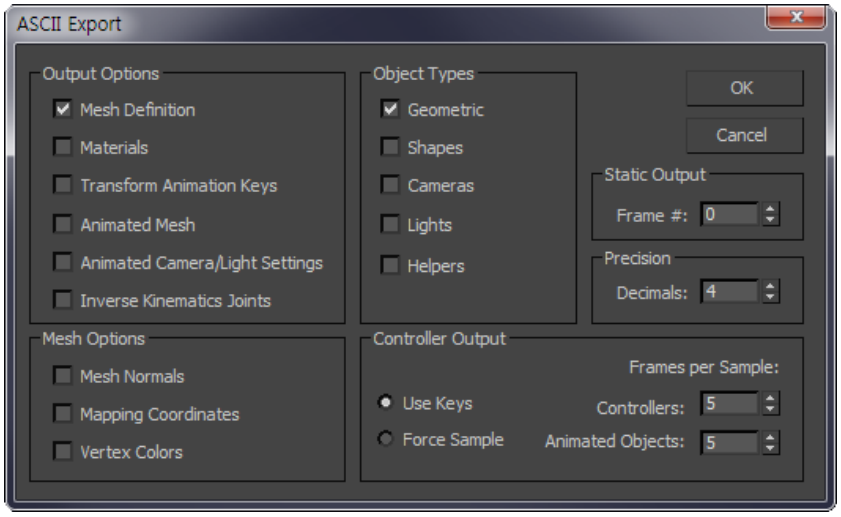


<3DS Max Units Setup>

맥스의 Modifier로 Pyramid의 정점 위치를 적당한 값으로 조정한 다음 이 파일을 ASE로 저장하기 위해서 '메뉴->Export'를 선택합니다.



'ASEII Export' 팝업 창에서 익스포트 할 여러 데이터 선택 중에서 그림처럼 'Output Options'의 'Mesh Definition'과 'Object Types'의 'Geometric' 만 체크한 후 Ok 버튼을 누릅니다.
이렇게 하면 'Select File to Export'에서 설정한 폴더에 ASE 파일이 저장됩니다.



asa00_pyramid.zip

피라미드 대신 주전자도 만들어 봅시다. 주전자 동일한 방법으로 맥스 파일을 만들고 익스포트를 합니다. 단, 이전의 피라미드와 다르게 'Materials'와 'Mesh Normals'를 같이 선택하고 나서 익스포트 해봅니다.

asa00_teapot.ASE

주전자(teapot) 파일을 열어 보면 Material과 정점의 Normal 벡터 정보를 포함하고 있어서 피라미드 예제보다 상당히 많은 내용을 담고 있습니다. 여기에 애니메이션과 관련된 데이터까지 익스포트 한다면 파일의 내용과 크기는 엄청 더 커집니다. 이렇게 내용이 많다고 너무 겁먹을 필요는 없습니다. 이것들은 단계적으로 하나씩 처리하면서 일정한 프로그램 형식을 적용하면 여러분들의 자

료구조에 눈 깜짝할 순간에 올라오게 되어있습니다.

D&C(Divide and Conquer) 개발 방법을 적용해서 거대한 파일의 내용은 작은 부분으로 나누어 해석하는 것이 필요합니다.
먼저 기본적인 텍스트 모드 파일 입/출력과 문자열에 포함된 필요 없는 문자 제거를 위한 함수를 준비합니다.

ASE 파일은 텍스트로 구성되어 있습니다. ASE 파일을 라인 단위로 해석하기 위기 위한 CLcAse 클래스는 Load() 함수에서 데이터를 파싱(Parsing) 합니다. 이 함수는 파일의 핸들을 fopen() 함수로 가져오고 close() 함수 반환하며 텍스트 형식으로 저장된 문자열을 읽어올 때 개행 문자(₩n)까지 라인 단위로 읽어오고 이를 해석하기 위해 fgets() 함수와 sscanf() 함수를 활용합니다.

```
INT CLcAse::Load()
...
        FILE*   fp;                     // ASE 파일 포인터
        FILE*   fpCnf;                  // 확인용 함수 포인터
        char sLine[MAX_PARSE_LINE];     // 읽기 버퍼

        fp   = fopen(m_sFile, "rt");
        fpCnf= fopen("Model/PyramidConfirm.txt", "wt");

        while(!feof(fp))
        {
                fgets(sLine, MAX_PARSE_LINE, fp);
                int iLen = strlen(sLine);
                if(iLen<3)
                        continue;

                LcStr_Trim(sLine);       // 공백, 탭, 개행 제거

                //확인용 파일에 공백을 제거한 문자열 쓰기
                fprintf(fpCnf, "%s₩n", sLine);
...
```

fgets() 함수에서 라인 단위로 읽은 문자열은 개행 문자까지 포함되어 있습니다. LcStr_Trim() 함수는 문자열의 왼쪽, 오른쪽 문자열의 공백, 탭, 그리고 개행 문자를 삭제한 후 반환하는 함수 입니다.

```
void LcStr_Trim(char* sBuf)
...
iLen = strlen(sBuf);
...
// 문자 ₩r ₩n 제거
for(i=iLen-1; i>=0; --i)
{
        char* p = sBuf + i;
        if( '₩n' == *p || '₩r' == *p)
                *(sBuf + i) = '₩0';
...
}

// 오른쪽 공백, 탭(₩t) 제거
...
for(i=iLen-1; i>=0; --i)
{
        char* p = sBuf + i;
        if( ' ' == *p || '₩t' == *p)
                continue;

        *(sBuf + i+1) = '₩0';
        break;
}
...
// 왼쪽 공백, 탭(₩t) 제거
for(i=0; i < iLen; ++i)
{
        char* p = sT + i;

        if( ' ' == *p || '₩t' == *p)
                continue;

        break;
}
...
```

다음 예제를 실행해 보면 ASE 파일이 저장된 폴더에 ASE에서 공백과 탭이 제거된 자료가 저장된 txt 파일을 볼 수 있습니다.

asa01_text_read.zip

1.2 구문 해석 방법

지금까지 텍스트 파일의 입출력, 문자열 시작과 끝에서 공백, 탭, 개행 문자들을 제거해 보았습니다. 이제부터 본격적으로 ASE를 해석할 차례입니다.
ASE 파일을 살펴 보면 각 라인의 내용이 {Keyword, Values}의 형식으로 되어 있음을 직관적으로 알 수 있습니다. 또한 Values는 없는 것도 있고, 하나만 있는 것도 있고 복수의 형태로 구성되어 있는 것도 있음을 볼 수 있고 이 모든 값들은 전적으로 Keyword에 따라 달라짐을 알 수 있습니다. 이것은 Keyword에 대해서 문장을 해석해야 함을 간접적으로 이야기 하고 있고 각 문장에 따라 코딩을 해야 함을 의미합니다.

문장의 끝에 '{' 기호가 있는 경우, 자료구조 블록의 시작을 알리고 블록의 끝에서는 '}' 로 지정하고 있음을 볼 수 있습니다. 이렇게 중 괄호로 표현된 부분은 자료구조를 선언하고 나서 while과 같은 loop 문으로 처리함을 의미하며 아마도 이 부분이 ASE파싱에서 가장 난이도가 있는 부분이기도 합니다. 이 부분은 자료구조에서 더 살펴보기로 하고 먼저 프로그램 구현을 위해서 다음과 같이 해석을 위한 키를 저장할 수 있는 구조체를 작성하고 ASE에 저장되어 있는 키들을 선언합니다. 키의 크기는 64byte 정도면 충분합니다.

```
typedef char AseKey[64];        // String Keword
...
AseKey  Keywords[] =            // Parsing Keyword
{
        "*GEOMOBJECT {",
        "*NODE_NAME"    ,,
        "*NODE_TM {"    ,
        "*MESH {"       ,
        "*MESH_NUMVERTEX",
...
```

이렇게 선언한 키들을 가지고 문장을 해석할 때 키에 해당 하는 데이터 해석을 작성하면 됩니다.

```
if(0 == _strnicmp(sLine, Keywords[0] , strlen(Keywords[0])) )
```

```
{
        // 데이터 해석
        …
}
```

키 비교에 대한 별도의 함수를 두는 것도 좋은 방법입니다.

```
BOOL CompareAseKey(char* val, char* key)
{
        return (0 == _strnicmp(val, key, strlen(key) ) ) ? 1: 0;
}
…
if(CompareAseKey(sLine, Keywords[1]) )
{
        // 데이터 해석
        …
}
```

Values에서 유의해 볼 것은 겹 따옴표(" ")로 구성되어 있는 값이 있다는 것도 잊지 않아야 합니다. 이렇게 큰 따옴표로 묶인 부분은 파일 이름이나 맥스에서 작업한 지오메트리(Geometry)의 오브젝트 이름과 같은 문자열 데이터로써 공백이 포함되는 경우도 있으므로 단순하게 sscanf()로 읽게 되면 낭패를 볼 수 있습니다. 따라서 큰 따옴표 처리를 위한 함수를 만들어야 합니다.

```
// 겹따옴표 안의 문자열 읽기
void LcStr_Quot(char* sDst, const char* sSrc)
…
        int iLen = strlen(sSrc);
        char* p = (char*)sSrc;

        while( 0 != *p)
        {
                if( '₩"' == *p && 0 == bStrt)
                        bStrt = 1;

                else if( '₩"' == *p && 1 == bStrt)
                {
                        *(sDst + nBgn) = 0;
```

```
                break;
        }

        if(nBgn>=0 && 1== bStrt)
                *(sDst + nBgn) = *p;

        if(1== bStrt)
                ++nBgn;

        ++p;
...
```

asa02_text_parse.zip

1.3 자료구조

어느 정도 코드를 갖추었으면 자료구조를 만들어야 합니다. ASE 파일의 자료구조는 크게 Scene, Material(재질), Geometry Object(GEOMOBJECT)로 구성되어 있습니다. Scene 자료구조는 애니메이션의 프레임과 시간이 저장되어 있습니다. Material 자료구조에는 조명에 필요한 재질, 정점 좌표에 적용이 되는 텍스처, 라이팅 등등의 정보들이 저장되어 있습니다.

가장 중요한 것은 GEOMOBJECT 자료구조 입니다. 이 자료구조에는 맥스에서 작업할 때 지정된 자신의 노드 이름과 링크가 설정이 되었을 때 부모 노드의 이름이 있습니다. 또한 장면을 구성하기 위한 월드행렬, 정점의 인덱스, 정점의 위치, 법선, 텍스처 좌표 등에 해당하는 메쉬 정보 그리고, 애니메이션에 관련된 프레임에 따라 위치, 이동, 크기변환에 대한 정보, 머티리얼(Material) 인덱스 등이 저장되어 있습니다.

ASE 파일에서 자료구조를 구성하는 가장 쉬운 방법은 ASE에 저장된 키와 값을 이용해서 프로그램 언어로 바꿀 수 있도록 먼저 의사 코드(psudo-code) ASE 파일을 복사해서 직접 작성해 보는 것입니다.

```
struct SCENE {
        int SCENE_FIRSTFRAME      0
        int SCENE_LASTFRAME     100
        int SCENE_FRAMESPEED     30
        int SCENE_TICKSPERFRAME 160
```

```
};

struct GEOMOBJECT {
        char NODE_NAME[64] "Pyramid001"

        struct MESH
        {
                int MESH_NUMVERTEX 6
                int MESH_NUMFACES 8

                VEC3 MESH_VERTEX[0]     0.0000   0.0000  60.0000
                VEC3 MESH_VERTEX[1]   -40.0000 -40.0000   0.0000
                VEC3 MESH_VERTEX[2]    40.0000 -40.0000   0.0000
...
                IDX3 MESH_FACE[0]    0   1   2
                IDX3 MESH_FACE[1]    0   2   3
                IDX3 MESH_FACE[2]    0   3   4
...
```

<자료구조 구성: asa03_struct.zip의 Model/asa00_pyramid_struct.ase>

이렇게 초벌로 작업한 내용을 c언어 등 프로그램 언어로 바꾸고 Ase 클래스를 완성합니다.

```
class CLcAse
...
        typedef char AseKey[64];        // ASE Keyword

        struct AseVtx                   // ASE Mesh의 x, y, z를 읽기위한 구조체
        {
                FLOAT x, y, z;
        };

        struct AseFce                   // ASE Face의 a, b, c를 읽기위한 구조체
        {
                WORD a, b, c;
        };
```

```
        struct AseGeo                   // ASE Geometry
        {
                char    sNodeName[64];  // Node 이름
                int     iNumVtx;        // Vertex의 수
                int     iNumFce;        // Index의 수
                AseVtx* pLstVtx;        // Vertex 리스트
                AseFce* pLstFce;        // Face 리스트
        };

protected:
        char    m_sFile[MAX_PATH];      // 모델 파일

        INT     m_iNGeo;                // Geometry 개수
        AseGeo* m_pGeo;                 // Geometry
...
```

Ase 클래스가 완성이 되면 Load() 함수에서 ASE 파일의 "{", "}" 안의 데이터는 동적으로 할당하고 loop문으로 해석하면 자료구조의 큰 골격은 완성이 됩니다.

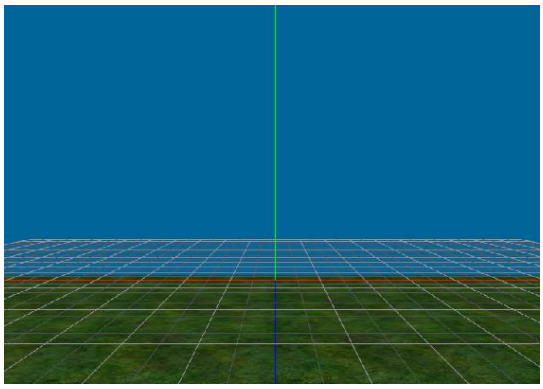
```
INT CLcAse::Load()
...
if(CompareAseKey(sLine, "*MESH_NUMVERTEX") )
{
        INT             iNVx;
        sscanf(sLine, "%*s %d", &iNVx);
        m_pGeo[nGeoIdx].iNumVtx = iNVx;
        m_pGeo[nGeoIdx].pLstVtx = new CLcAse::AseVtx[iNVx];
}

if(CompareAseKey(sLine, "*MESH_NUMFACES") )
...
        sscanf(sLine, "%*s %d", &iNIx);
        m_pGeo[nGeoIdx].iNumFce = iNIx;
        m_pGeo[nGeoIdx].pLstFce = new CLcAse::AseFce[iNIx];
...
if(CompareAseKey(sLine, "*MESH_VERTEX_LIST {") )
{
```

```
        while(!feof(fp))
        {
                fgets(sLine, MAX_PARSE_LINE, fp);
...
                if(CompareAseKey(sLine, "*MESH_VERTEX") )
...
                        sscanf(sLine, "%*s %d %f %f %f", &nIdx, &x, &y, &z);
                        m_pGeo[nGeoIdx].pLstVtx[nIdx].x = x;
                        m_pGeo[nGeoIdx].pLstVtx[nIdx].y = y;
                        m_pGeo[nGeoIdx].pLstVtx[nIdx].z = z;
...
if(CompareAseKey(sLine, "*MESH_FACE_LIST {") )
{
        while(!feof(fp))
        {
                fgets(sLine, MAX_PARSE_LINE, fp);
...
                if(CompareAseKey(sLine, "*MESH_FACE") )
...
                        sscanf(sLine, "%*s %d: %*s %d %*s %d %*s %d", &nIdx, &a, &b,  &c);
                        m_pGeo[nGeoIdx].pLstFce[nIdx].a = a;
                        m_pGeo[nGeoIdx].pLstFce[nIdx].b = b;
                        m_pGeo[nGeoIdx].pLstFce[nIdx].c = c;
...
```

asa03_struct.zip

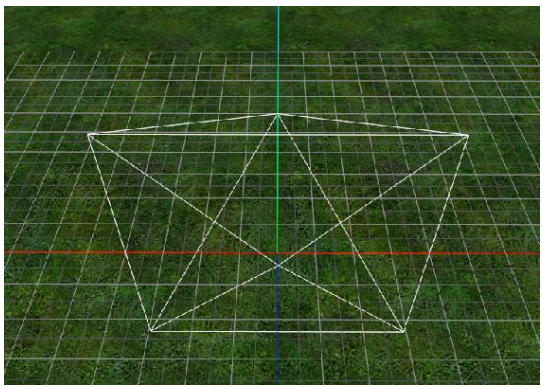
1.4 뷰어와 피라미드

ASE에서 데이터를 읽고 해석 하는 기본 작업은 거의 다 끝이 났습니다. 이제부터 화면에 장면을 연출하면서 코드를 보강 해야 합니다. 먼저 준비해야 할 것은 ASE 모듈을 테스트 할 수 있는 테스트 드라이버(test driver: 모듈 테스트 프로그램)가 필요합니다. 테스트 드라이버는 키보드, 카메라, 그리고 오브젝트의 크기를 눈으로 확인할 수 있는 그리드(Grid)가 보이는 구성합니다.



asa04_viewer_basic.zip

피라미드 ASE 파일에서 해석한 버텍스와 인덱스를 렌더링하면 맥스와 다르게 그림과 같이 피라미드가 누워있는 모습을 볼 수 있습니다. 이것은 맥스가 오른손 좌표계를 사용하면서 Up 방향의 축을 z축으로 사용하고 있기 때문입니다. 따라서 이 부분을 D3D에 맞게 변경해야 합니다.



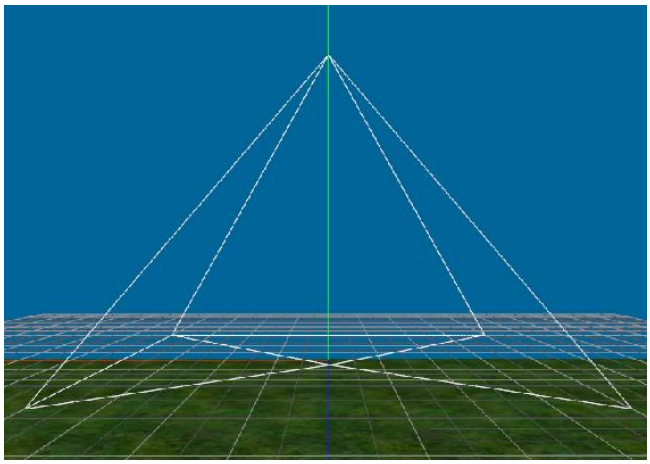
<D3D 환경으로 변경하지 않은 ASE>

ASE 해석에서 정점의 위치를 읽는 경우 맥스의 Y 축은 D3D의 Z축이 되고 맥스의 Y축은 D3D의 Z축이 됩니다. 따라서 다음과 같이 y와 z를 읽어야 합니다.

```
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;
```

맥스는 오른손 좌표계를 사용하고 DirectX는 왼손 좌표계를 사용합니다. 삼각형을 그리는 순서가 맥스는 시계 반대방향(CCW: Count Clock Wise)로 그리고 DirectX는 시계 방향(CW: Clock Wise)로 그립니다. 따라서 CW로 그릴 수 있도록 다음과 같이 b와 c를 교환해줍니다..

```
INT     nIdx=0, a=0, b=0, c=0;
sscanf(sLine, "%*s  %d:  %*s  %d  %*s  %d  %*s  %d", &nIdx, &a, &b, &c);
m_pGeo[nGeoIdx].pLstFce[nIdx].a = a;
m_pGeo[nGeoIdx].pLstFce[nIdx].b = c;
m_pGeo[nGeoIdx].pLstFce[nIdx].c = b;
```



<D3D환경으로 변경된 ASE 해석:asa05_viewer_pyramid.zip>

1.5 추상화

ASE를 해석하는 코드는 애니메이션과 오브젝트에 대한 처리가 정교해질수록 코드가 점점 커지게 되어 있습니다. 따라서 이런 부분을 라이브러리로 만들어 놓고 사용한다면 ASE 코드를 만드는 개발자나 이 코드를 게임에 적용하는 메인 프로그래머에게 적용에 대해서 번잡하지 않게 만들어 작업의 효율을 높일 수 있습니다.
라이브러리는 단순히 코드의 집합입니다. 라이브러리를 쉽게 사용하고 기능 추가, 버그 해결 등 유지보수를 원활히 하려면 단순 클래스 보다 이를 추상화하는 것이 좋습니다. 그래서 우리는 지금까지 사용한 클래스를 다음과 같이 모델 인터페이스를 최상위로 정의하고 CLcAse 클래스는 이를 상속해서 ASE를 처리하는 구조로 변경합니다. 또한 객체의 생성은 LcAse_Create() 함수와 같이 new 연산자가 아닌 함수가 담당하는 것으로 정의합니다.

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

```
        virtual INT     FrameMove()=0;
        virtual void    Render()=0;
};

INT LcAse_Create(char* sCmd
                , ILcMdl** pData        // 생성된 반환 객체
                , void* pDev            // Device
                , void* sName  = NULL   // 모델 파일 이름
...
class CLcAse : public ILcMdl
...
```

지금까지 테스트 드라이버 프로그램에서 객체 생성을 클래스로 한 것을 LcAse_Create() 함수로 생성하도록 합니다.

```
class CMain : public CD3DApplication
...
        ILcMdl*         m_pMdl;
...

HRESULT CMain::Init()
        ...
        if( FAILED( LcAse_Create(NULL, &m_pMdl, m_pd3dDevice, "Model/asa00_pyramid.ASE")))
                return -1;
...
```

객체는 CMain::Destroy() 함수에서 해제하고, 렌더링은 CMain::Render() 함수에서 처리합니다.

```
HRESULT CMain::Render()
        ...
        if( FAILED( m_pd3dDevice->BeginScene() ) )
                return -1;
        ...
        SAFE_RENDER(    m_pMdl          );
        ...
        m_pd3dDevice->EndScene();
```

```
HRESULT CMain::Destroy()
...
        SAFE_DELETE(    m_pMdl  );
...
```

asa06_object.zip

간단하게 최상위 클래스로 만들고 이를 구현하는 방법을 살펴보았습니다. 다시 강조 하지만 ASE 모델을 추상화하면 개발 및 유지 보수가 좋아집니다. ASE와 같이 복잡한 코드를 작업할 때 객체의 추상화는 반드시 필요하다고 할 수 있습니다.

1.6 Geometries

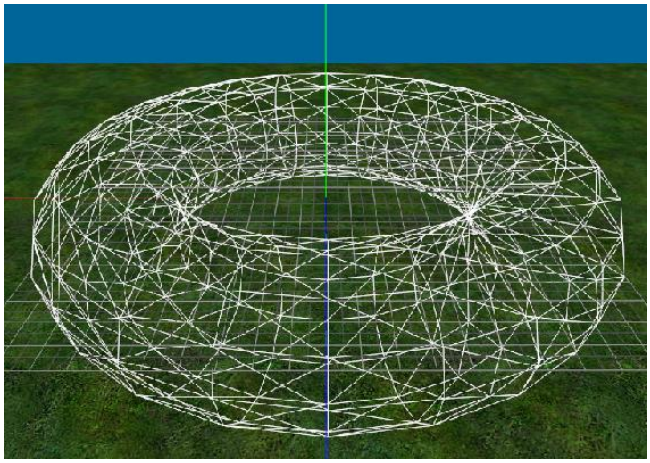
캐릭터, 건물 등 게임에서 사용하는 오브젝트들은 대부분 여러 지오메트리로 구성되어 있습니다. 특히 애니메이션 오브젝트들은 하나의 메쉬로 구성되어 있지는 않습니다. 이것은 오브젝트를 개별적으로 만들어서 하나로 합치기 때문입니다.

ASE는 Geometry의 개수를 기록하지 않습니다. 따라서 다음과 같이 while 루프를 이용해서 지오메트리의 개수를 찾아야 합니다. 지오메트리의 숫자를 파악하는 동안 파일 포인터가 옮겨졌기 때문에 파일 포인터를 반드시 처음으로 이동시켜야 합니다.

```
// 지오메트리를 찾아서 개수를 정한다.
while(!feof(fp))
{
        ...
        if(0 == _strnicmp(sLine, "*GEOMOBJECT {", strlen("*GEOMOBJECT {") ))
                ++m_nGeo;
        ...
}
...
// 파일 포인터를 처음으로 이동한다.
fseek(fp, 0, SEEK_SET);
```

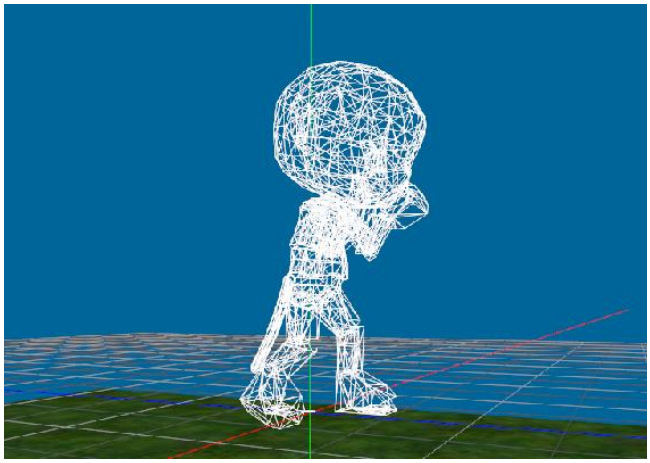
지오메트리의 숫자를 파악하고 하고 나서 지오메트리에 대한 메모리를 확보하고 지오메트리의 포인터를 얻어서 ASE 데이터를 해석합니다.

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

while(!feof(fp))
…
        if(0 == _strnicmp(sLine, "*GEOMOBJECT {", strlen("*GEOMOBJECT {") ))
        {
                ++nGeoIdx;
                //해석할 지오메트리 포인터
                pGeo = &m_pGeo[nGeoIdx];
…
                //해당 지오메트리에 대한 ASE 해석
                pGeo->…
```



<단일 지오메트리 - asa11_geo_single.zip>

피라미드, torus, 주전자 등을 테스트 했다면 게임에서 사용하는 모델을 올려봅니다.



<다중 지오메트리 - asa12_geo_multi.zip>

1.7 텍스처

애니메이션이 없는 오브젝트의 마지막 단계로 텍스처 적용이 남아 있습니다. ASE 파일 구조 안에서 텍스처에 대한 정보는 머티리얼(Material: 재질) 안에 저장되어 있습니다. 맥스의 머티리얼은 광원에 대한 정보와 정점에서 사용하는 텍스처에 대한 파일 정보가 저장되어 있습니다.
머티리얼은 여러 하위 머티리얼(서브 머티리얼:sub material)을 가질 수 있습니다. 이것은 지오메트리가 하나의 메쉬로 구성되어 있지만 각각의 정점은 다른 머티리얼을 참고 할 수 있도록 맥스가 구성되어 있기 때문입니다. 이에 대한 예는 각각 다른 머티리얼을 가진 두 이상의 오브젝트를 하나로 합칠 때 발생합니다. 따라서 프로그래머는 이에 대해서도 처리 해야 하지만 이 강좌에서는 이것을 다루기에 내용이 많으므로 모든 각각의 오브젝트는 단 하나의 머티리얼만 사용한다는 가정을 하고 서브 머티리얼은 략하겠습니다.

UV좌표는 화면의 왼쪽 상단이 (0,0), 오른쪽 하단이 (1,1) 좌표입니다. ST 좌표는 화면의 왼쪽 하단이 (0,0), 오른쪽 상단이(1,1) 인 수학에서 사용하는 2차원 좌표계로 구성되어 있는 좌표계입니다.
DirectX는 UV 좌표를 사용하지만 맥스 ST 좌표를 사용합니다. 따라서 맥스의 텍스처 좌표는 DirectX에 맞게 변경이 되야 하는데 Y에 대한 값을 다음과 같이 변경하면 됩니다.

```
Y = 1.0F - Y
```

맥스는 텍스처 좌표를 3차원 좌표 UVW를 사용합니다. D3D의 게임은 x, y만 필요하기 때문에 앞의 두 개만 해석해서 사용하면 됩니다.

```
INT     nIdx=0;
FLOAT   u=0. 0F, v=0. 0F, w=0. 0F;
sscanf(sLine, "%*s %d %f %f %f", &nIdx, &u, &v, &w);
pGeo->pLstTvtx[nIdx].u = u;
pGeo->pLstTvtx[nIdx].v = 1.0f - v;
```

맥스는 정점 버퍼와 텍스처에 대한 Tvertex 버퍼를 개별적으로 가지고 있습니다. 이것은 하나의 정점을 여러 텍스처 좌표가 공유하도록 해서 정점의 숫자를 줄일 수 있게 됩니다. 따라서 정점의 위치 좌표 개수는 텍스처의 좌표 보다 같거나 작습니다.
이러한 이유로 삼각형을 구성하는 인덱스 또한 따로 가지고 있습니다. 위치에 대한 인덱스(Face)와 텍스처 좌표에 대한 인덱스(T-Face)의 숫자는 거기에 저장된 인덱스는 반드시 일치하지는 않습니다. 따라서 T-Face에 대한 내용을 따로 읽어와야 합니다.
이렇게 다르다면 어떻게 렌더링에 대한 인덱스와 정점의 위치 그리고 텍스처 좌표를 구성해야 해야 할까요?

```
//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
```

<ASE의 Face와 T-face의 비교>

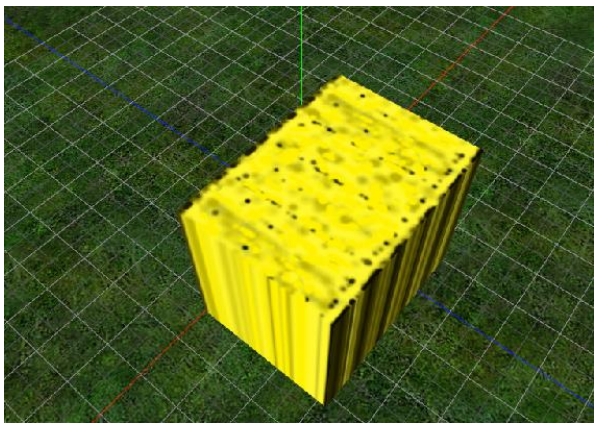
ASE의 Face와 T-face의 인덱스 내용은 다르지만 전체 개수는 일치합니다. 또한 리스트에서 참조하고 있는 정점의 번호와 텍스처 좌표 번호는 항상 일치 합니다. 예를 들어 그림을 보면 Face 리스트 480 사용하고 있는 정점의 인덱스 245는 481, 482에도 사용하고 있고 이러한 패턴은 T-face에서도 그대로 480,481, 482에서 사용하고 있음을 볼 수 있습니다.

이러한 패턴을 기억하고 프로그램을 작성해 봅시다. 먼저 T-face 해석으로 T-Face도 정점의 인덱스와 마찬가지로 렌더링의 CW를 위해 b와 c를 교환 합니다.

```
INT     nIdx=0, 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;
```

맥스는 여러 텍스처 좌표가 하나의 정점 위치를 공유할 수 있기 때문에 정점 위치를 가지고 렌더링의 정점을 정하는 것이 아니라 텍스처 좌표를 가지고 정점을 구성해야 합니다.
즉, 최종 정점의 숫자는 텍스처 좌표 숫자와 일치 시키고, 삼각형의 인덱스 구성도 T-Face로 정해서 렌더링 합니다. 이에 따라 T-vertex를 중심을 위치를 vertex에서 복사해서 사용해야 합니다.

```
//1. 렌더링 버퍼(pVtxR )에 UV를 먼저 설정
for(int j=0; j< pGeo->nTvtx; ++j)
{
        pVtxR[j].u = pGeo->pTvtx[j].u;
        pVtxR[j].v = pGeo->pTvtx[j].v;
}

//2. 렌더링 버퍼(pVtxR )에 위치(x,y,z)를 결정
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;
}
```



<텍스처:asa13_texture.zip>

2 Rigid body Animation

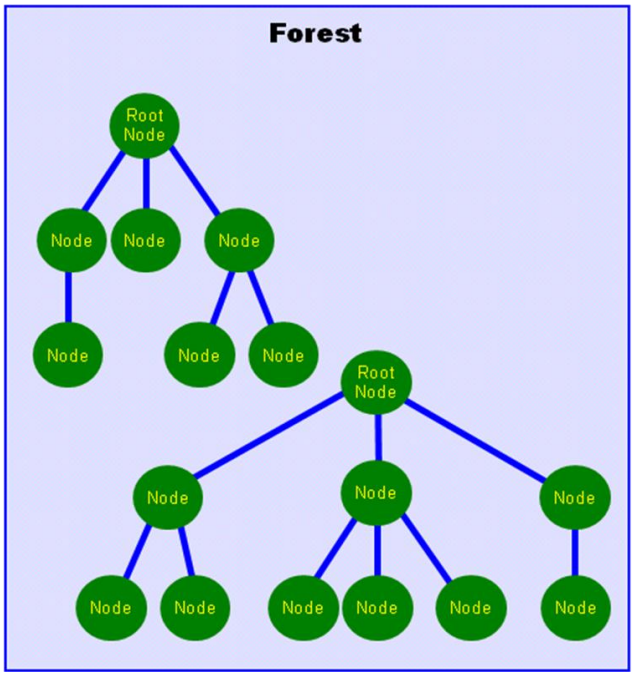
애니메이션은 여러 분류가 있는데 그 중에서 지오메트리를 구성하고 있는 정점들 사이의 간격이 변하지 않는 강체(Rigid body) 애니메이션과 피부처럼 접히거나 늘어나는 스키닝(Skinning) 애니메이션으로 구분할 수 있습니다.
강체 애니메이션은 지오메트리의 모든 정점들에 대해서 같은 값을 가지는 행렬이 적용 됩니다. 스키닝은 지오메트리를 같이 구성하지만 행렬의 값은 개별적으로 설정할 수 있습니다. 과거에는 스키닝을 지원하는 그래픽 카드가 워낙 비싸서 게임에서는 강체 애니메이션을 주로 사용했습니다. 지금은 가격이 많이 내려 캐릭터에 대한 애니메이션은 대부분 스키닝을 이용합니다.
강체 애니메이션은 관절에서 폴리곤이 갈라지는 부분만 뺀다면 스키닝보다 구현과 속도 면에서 우수 합니다. 따라서 기계 팔과 움직임은 강체 애니메이션을 적용한다면 오히려 속도 향상의 이득이 있습니다.
ASE는 강체 애니메이션을 지원합니다. 그리고 강체 애니메이션을 구현하고 맥스 데이터의 행렬 연산을 이해한다면 스키닝을 배우는데 좀 더 쉽게 배울 수 있습니다.

이 장에서는 ASE에 포함된 애니메이션 데이터를 해석하고 이를 구현해 보도록 하겠습니다.

2.1 변환 행렬(TM: Transform Matrix)

애니메이션이 없는 ASE 구조는 해석하기가 쉽지만 애니메이션이 있는 경우 지오메트리에 포함된 각각의 행렬과 이들의 연관 관계의 이해가 중요합니다.
맥스의 3D 장면은 그림과 같이 나무 자료구조(Tree)의 집합인 숲(Forest) 자료구조로 구성되어 있습니다. 오브젝트의 각각의 노드를 우리는 지오메트리로 알고 있고, ASE 익스포터는 루트 노드부터 깊이 우선 순회(Depth First Search)로 각각의 지오메트리의 자료를 파일로 저장합니다.

또한 지오메트리에 포함된 행렬은 지역행렬이 아닌 월드행렬입니다. 따라서 애니메이션을 구현할 때 이들 월드행렬을 지역행렬로 바꾸어서 루트부터 행렬을 갱신하고 자식 노드로 이를 반복하는 구조로 애니메이션을 구현해야 됩니다.



<나무 자료구조(Tree) 구조로 구성된 숲 자료구조(Forest)>

애니메이션은 노드(지오메트리)의 월드행렬을 구하는 일입니다. 지역행렬을 갱신하고 부모 행렬과 곱하면 자신의 월드행렬이 되므로 오브젝트를 구성하는 모든 객체들은 각자 자기의 모델 좌표계에 대한 행렬, 즉 자신의 지역행렬 값을 가지고 있어야 합니다.

ASE파일의 *NODE_TM{} 안에 오브젝트의 월드행렬에 대한 정보가 있습니다. D3D 환경에 맞추기 위해서 정점 해석에서 z와 y을 바꾸었듯이 이들 행렬도 y축과 z축에 대한 교환이 필요합니다. 따라서 행렬을 읽고 2열, 3열의 교환 뿐만 2행, 3행의 교환도 필요합니다.

예를 들어 다음과 같은 값으로 NODE_TM이 구성되어 있는 경우를 해석해 봅시다.

```
*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
```

이것은 DirectX의 D3DXMATRIX 행렬로 변환을 할 때 다음과 같이 TM_ROW1 부분과 TM_ROW2 부분을 교환합니다.

```
-0.1225          -0.3713           0.9204
 0.8605          -0.5018          -0.0879
-0.4945          -0.7812          -0.3810
-0.0443          -0.2532          10.2929
```

다음으로 2 번째와 3 번째 열을 교환합니다.

```
-0.1225          0.9204          -0.3713
 0.8605          -0.087          -0.50189
-0.4945          -0.381          -0.78120
-0.0443          10.292          -0.25329
```

마지막으로 4열의 값을 0, 0, 0, 1 의 값으로 채우면 이 지오메트리의 월드행렬을 얻게 됩니다.

```
-0.1225    0.9204  -0.3713      0
 0.8605   -0.087   -0.50189     0
-0.4945   -0.381   -0.78120     0
-0.0443   10.292   -0.25329     1
```

```
while(…)
…
        if(CompareAseKey(sLine, Keywords[TM_ROW1]) )
…
                pGeo->tmInf.mtW[2][0]  = x;
                pGeo->tmInf.mtW[2][1]  = z;
                pGeo->tmInf.mtW[2][2]  = y;
                pGeo->tmInf.mtW[2][3]  = w;
…
        if(CompareAseKey(sLine, Keywords[TM_ROW2]) )
…
                pGeo->tmInf.mtW[1][0]  = x;
                pGeo->tmInf.mtW[1][1]  = z;
                pGeo->tmInf.mtW[1][2]  = y;
                pGeo->tmInf.mtW[1][3]  = w;
```

asa21_tm.zip

앞서 이야기 했듯이 이 행렬은 지오메트리(노드)의 월드행렬입니다. 따라서 이를 지역행렬로 바꾸어야 하는데 이 행렬은 익스포터가 다음과 같은 수식으로 만들고 출력한 행렬 값입니다.

지오메트리 행렬(자신의 월드행렬) = (자신의 지역행렬) * (부모의 월드행렬)

따라서 자신의 지역행렬은 앞의 수식의 양변에 부모 행렬의 역행렬을 곱하면 지오메트리 자신의 지역행렬을 구할 수 있습니다.

자신의 지역행렬 = (자신의 월드행렬) * (부모 월드행렬의 역행렬)

$\equiv$ Local = World * $(\text{Parent World})^{-1}$

이것을 프로그램으로 구현하면 다음과 같습니다.

```
D3DXMATRIX mtPrn = pGeoPrn->TmInf.mtW;          // 부모의 월드행렬
D3DXMATRIX mtPrnI;                              // 부모 월드행렬의 역행렬
D3DXMATRIX mtL;                                 // 자신의 지역행렬
D3DXMatrixInverse(&mtPrnI, NULL, &mtPrn);       // 부모 월드행렬의 역행렬

// 자신의 지역행렬 = (자신의 월드행렬) * (부모 월드행렬의 역행렬)
pGeo->TmInf.mtL = pGeo->TmInf.mtW * mtPrnI;
```

ASE 파일의 지오메트리는 자료구조의 숲(Forest) 구조로 구성되어 있고 ASE 익스포터는 각각의 지오메트리를 깊이 우선 순회로 접근해서 데이터를 출력한다고 했습니다. 따라서 파일에서 순차적으로 읽은 지오메트리의 순서는 곧 숲을 구성하는 나무(Tree) 자료구조와 일치하므로 for 루프를 통해서 순서대로 지역행렬을 만들면 됩니다.

루트와 같이 만약 부모가 없다면 부모의 월드행렬의 역행렬은 항등행렬로 설정해서 지역행렬을 구하면 됩니다.

```
void CLcAse::ParseReBuild()
...
// TmLocal = TmWorld * TmWorld(parent)^-1
for(i=0; i<m_nGeo; ++i)
{
```

```
        AseGeo* pGeoCur = &m_pGeo[i];
        AseGeo* pGeoPrn = NULL;

        // 부모 지오메트리 찾기
        for(j=0; j<m_nGeo; ++j)
        {
                AseGeo* pGeoT = &m_pGeo[j];

                if(0==_stricmp(pGeoCur->sNodePrn, pGeoT->sNodeCur))
                {
                        pGeoPrn         = pGeoT;
                        pGeoCur->pGeoPrn= pGeoT;
                        break;
...
        // 부모 지오메트리 발견
        if(pGeoPrn)
        {
                D3DXMATRIX mtPrn = pGeoPrn->TmInf.mtW;
                D3DXMATRIX mtPrnI;
                D3DXMATRIX mtL;
                D3DXMatrixInverse(&mtPrnI, NULL, &mtPrn);

                pGeoCur->TmInf.mtL = pGeoCur->TmInf.mtW * mtPrnI;
        }

        // 부모 노드가 없는 루트의 경우 자신의 월드를 지역 행렬로 설정
        else
                pGeoCur->TmInf.mtL = pGeoCur->TmInf.mtW;
...
```

마지막으로 지오메트리의 정점 위치는 전체 장면의 월드 좌표계에서 바라본 위치 입니다.

(지오메트리 정점 위치) = (지역 좌표계 위치) * (월드 행렬)

따라서 이들은 지역 좌표계 위치로 바꾸어 합니다. 간단히 지오메트리 정점의 위치에 월드 행렬의 역행렬로 변환 하면 됩니다.(앞에서 언급 했듯이 월드 행렬은 ASE의 "*NODE_TM"에 저장된 값입니다.)

(지역 좌표계 위치) = (지오메트리 정점 위치) * (월드 행렬의 역행렬)

```
// 지역 좌표계로 위치 변환
// v(local position) = v(Geometry world position)*TmWorld^-1
for(i=0; i<m_nGeo; ++i)
{
	AseGeo* pGeoCur = &m_pGeo[i];

	// 월드 행렬의 역행렬을 구함
	D3DXMATRIX mtWI;
	D3DXMatrixInverse(&mtWI, NULL, &pGeoCur->TmInf.mtW);

	// 정점 위치를 지역 좌표계 위치로 변경
	for(j=0; j< pGeoCur->iNumVtx; ++j)
	{
		D3DXVECTOR3 vcI = *((D3DXVECTOR3*)&(pGeoCur->pLstVtx[j]));
		D3DXVECTOR3 vcO;

		// 정점 위치를 지역 좌표계 위치로 변경
		D3DXVec3TransformCoord(&vcO, &vcI, &mtWI);

		// 저장
		*((D3DXVECTOR3*)&(pGeoCur->pLstVtx[j])) = vcO;
...
```

asa22_tm_local.zip

2.2 SCENE 해석

지역행렬을 구했다면 애니메이션을 할 수 있습니다. 애니메이션 스피드, 전체 애니메이션 수 등은 ASE의 "SCENE" 부분에 있습니다. ASE의 SCENE은 간단하므로 쉽게 해석할 수 있습니다.

```
// ASE SCENE를 저장하기 위한 구조체
struct AseScene
{
```

```
        INT nFrameFirst;        // 시작 프레임 인덱스
        INT nFrameLast;         // 마지막 프레임 인덱스
        INT nFrameSpeed;        // 프레임 스피드
        INT nFrameTick;         // 프레임당 시간 간격
};
...
INT CLcAse::ParseScene(FILE* fp)
...
        if(CompareAseKey(sLine, Keywords[FRAME_FIRST]) )
                sscanf(sLine, "%*s %d", &m_AseScene.nFrameFirst);

        if(CompareAseKey(sLine, Keywords[FRAME_LAST]) )
                sscanf(sLine, "%*s %d", &m_AseScene.nFrameLast);

        if(CompareAseKey(sLine, Keywords[FRAME_SPEED]) )
                sscanf(sLine, "%*s %d", &m_AseScene.nFrameSpeed);

        if(CompareAseKey(sLine, Keywords[FRAME_TICK]) )
                sscanf(sLine, "%*s %d", &m_AseScene.nFrameTick);
```

asa23_scene.zip

2.3 Animation Track

오브젝트가 Biped애니메이션이 있을 경우, 익스포트 할 때 Transform Animation Key를 선택하면 ASE는 지오메트리의 "*CONTROL_POS_TRACK", "*CONTROL_ROT_TRACK", "*CONTROL_SCALE_TRACK" 키워드 안에 애니메이션 데이터를 저장합니다.
애니메이션 데이터는 이동과 크기 변환에 대해서 {키워드, 시간, x, y, z} 형식으로 저장합니다. 회전은 {키워드, 시간, x, y, z, w} 형식으로 저장하며 이 값은 사원수로 변경해서 사용합니다.

주의 해야 할 것은 ASE 파일에 저장된 맥스의 시간은 Tick을 사용하고 이것은 우리가 사용하는 1/1000초가 아닙니다. 시간을 게임 시간으로 변경해야 하는데 애니메이션 시간을 SCENE 안에 SCENE_TICKSPERFRAME 저장된 값으로 나누면 해당 프레임을 얻고 이 프레임에 다시 SCENE의 SCENE_FRAMESPEED 을 곱하면 절대 시간을 얻게 됩니다.

좀 더 구체적인 방법은 이후 시간에 대한 애니메이션에서 더 논의 하고 여기서는 데이터 해석에

집중하겠습니다.

먼저 위치, 회전, 크기 변환을 저장하기 위한 구조체를 구성합니다.

```
struct AseTrack
{
	FLOAT	x, y, z, w;	// Animation 정보 저장
	INT	nF;		// Frame Index
};
```

구조체에서 x, y, z, w 를 Frame Index보다 앞에 순서대로 놓으면 이 값을 저장한 변수의 주소를 위치 또는 사원수로 static casting이 가능하므로 불필요한 변수의 복사가 줄어듭니다.

앞에서 지오메트리 숫자는 기록이 없어서 키워드를 가지고 숫자를 세었습니다. 이 방법은 그리 훌륭한 방법이 못 되는데 그것은 STL이라는 좋은 자료구조가 C++에서 제공하기 때문입니다. 애니메이션 트랙 데이터는 STL의 vector 컨테이너를 사용해서 저장해 봅시다. 애니메이션 데이터를 저장할 수 있는 구조체를 선언합니다.

```
struct AseTrackAni
{
	char	sNodeCur[64];			// Current Node Name
	std::vector<AseTrack >	vRot;		// Rotation
	std::vector<AseTrack >	vTrs;		// Translation
	std::vector<AseTrack >	vScl;		// Scaling
};
```

위치에 대한 애니메이션 저장은 프레임 인덱스를 구하고 앞에서 선언한 구조체의 인스턴스를 만들어 애니메이션 컨테이너에 저장합니다.

```
if(CompareAseKey(sLine, Keywords[POS_SAMPLE]) )
...
	sscanf(sLine, "%*s %d %f %f %f", &nFrm, &x, &y, &z);

	// 프레인 인덱스 = sample tick/(SCENE Tick)
	nFrm /= m_AseScene.nFrameTick;

	// 애니메이션 벡터 컨테이너에 저장
```

```
        AseTrack trck(nFrm, x, y, z, 0);
        pTrckAni->vTrs.push_back(trck);
...
```

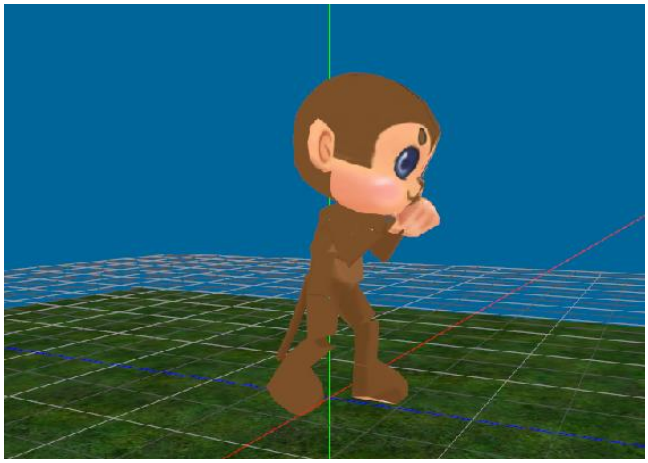
크기 변환 애니메이션도 위치 변환과 유사하게 처리합니다.

```
if(CompareAseKey(sLine, Keywords[SCALE_TRACK]) )
...
        sscanf(sLine, "%*s %d %f %f %f", &nFrm, &x, &y, &z);
        nFrm /= m_AseScene.nFrameTick;
        AseTrack trck(nFrm, x, y, z, 0);
        pTrckAni->vScl.push_back(trck);
```

회전 변환에 대한 애니메이션 해석은 파일에서 읽어온 x, y, z, w 값에서 사원수를 생성하고 저장합니다. ASE에 저장된 회전은 이전 프레임과의 차이만 저장합니다. 따라서 이것을 절대적인 행렬값으로 바꾸어야 하고 계속 누적을 해야 하는데 사원수를 사용하면 단순한 곱셈 만으로 누적을 편리하게 구현할 수 있기 때문입니다.

```
if(CompareAseKey(sLine, Keywords[ROT_SAMPLE]) )
...
        sscanf(sLine, "%*s %d %f %f %f %f", &nFrm, &x, &y, &z, &w);
        nFrm /= m_AseScene.nFrameTick;

        // x, y, z, w 에서 사원수를 구한다
        x = sinf(w/2.0f) * x;
        y = sinf(w/2.0f) * y;
        z = sinf(w/2.0f) * z;
        w = cosf(w/2.0f);

        INT iSize = pTrckAni->vRot.size();

        // 사원수를 처음하는 경우
        if(0==iSize)
        {
                AseTrack trck(nFrm, x, y, z, w);
                pTrckAni->vRot.push_back(trck);
```

```
        }

        // 앞의 사원수를 얻고 누적을 위해서 곱셈한 후, 이 사원수 값을 저장
        else
        {
                D3DXQUATERNION  q1(x,y,z,w);
                D3DXQUATERNION* q2;
                D3DXQUATERNION  q3;

                // 이전 프레임의 사원수 값
                q2 = (D3DXQUATERNION*)&pTrckAni->vRot[iSize-1];

                // 사원수 누적(곱셈)
                D3DXQuaternionMultiply(&q3, q2, &q1);

                // 사원수 저장
                AseTrack trck(nFrm, q3.x, q3.y, q3.z, q3.w);
                pTrckAni->vRot.push_back( trck );
...
```

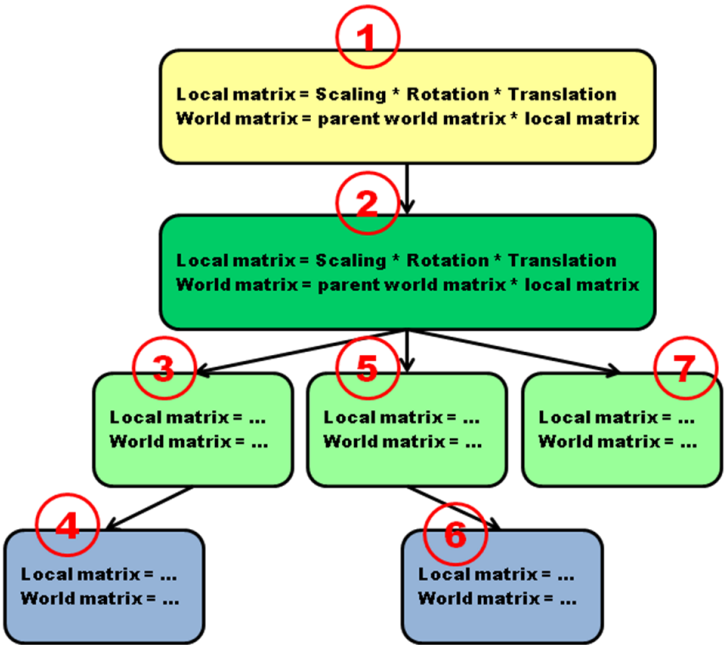


<ASE Track: asa24_track.zip>

2.4 Hierarchic Animation

애니메이션 해석을 마지막으로 ASE 해석은 끝났습니다. 남은 것은 기술적인 것으로 계층 애니메이션, 애니메이션 보간, 텍스트 대신 이진 파일 사용, 객체들의 인스턴스 관리, ASE 익스포터 수정 등이 남아 있습니다.

계층 애니메이션은 다음 그림과 같이 시간에 따라 루트부터 DFS(Depth First Search: 깊이 우선 순회) 방법으로 하위 노드들을 접근하면서 지역 행렬을 갱신하고, 지역행렬에서 월드 행렬을 갱신하는 애니메이션입니다.



< DFS 방식의 노드 순회와 행렬 연산 순서>

지역 행렬을 구성하기 위해서 지오메트리 인덱스, 프레임 인덱스를 인수로 받고 행렬을 반환하는 멤버 함수를 추가합니다. 이 멤버 함수는 크기 변환, 회전 변환, 이동 변환 순서대로 지역행렬을 계산 합니다. 만약 애니메이션이 없으면 지역 행렬 값을 사용합니다.
애니메이션이 있으면 해당 프레임의 인덱스로 애니메이션 트랙의 인덱스를 찾아서 행렬을 만드는데 회전을 저장하고 있는 트랙은 사원수를 사용하므로 D3DXMatrixRotationQuaternion() 함수 등을 사용해서 사원수에서 행렬을 얻습니다.

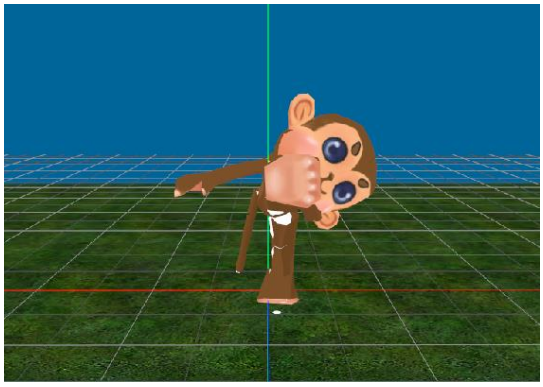
```
INT CLcAse::GetAniTrack(void* mtOut, INT nGeo, INT nFrame)
...
D3DXMATRIX      mtA(1,0,0,0,   0,1,0,0,   0,0,1,0,   0,0,0,1);
AseGeo*         pGeo    = &m_pGeo[nGeo];
INT             iSizeR = pGeo->vRot.size();
INT             iSizeP = pGeo->vTrs.size();
D3DXQUATERNION* q= NULL;
D3DXVECTOR3* p= NULL;
```

```
// 회전
...
for(i=0; i<iSizeR-1; ++i)
{
	if(pGeo->vRot[i].nF <=nFrame && nFrame <pGeo->vRot[i+1].nF)
	{
		q = (D3DXQUATERNION*)&pGeo->vRot[i];
		break;
...
if(q)
	D3DXMatrixRotationQuaternion(&mtA, q);

...
// 이동
for(i=0; i<iSizeP-1; ++i)
{
	if(pGeo->vTrs[i].nF <=nFrame && nFrame <pGeo->vTrs[i+1].nF)
	{
		p = (D3DXVECTOR3*)&pGeo->vTrs[i];
		break;
...
if(p)
{
	mtA._41 = p->x; mtA._42 = p->y; mtA._43 = p->z;
}
...
// 최종
*((D3DXMATRIX*) mtOut) = mtA;
```

앞서 이야기 했듯이 ASE의 파일 내용은 DFS 방식을 따라 저장되어 있습니다. 따라서 우리는 간단히 순차적으로 지역행렬, 월드행렬을 갱신하면 애니메이션에 대한 행렬 갱신은 마무리가 됩니다.

```
INT CLcAse::FrameMove()
...
INT i=0;
// 시간을 갱신하고 현재의 프레임 계산
m_dCur = timeGetTime();
```

```
if(m_dCur>m_dBgn+m_nFrmS)
{
...
        m_nFrmC =m_nFrmF;
}

for(i=0; i<m_nGeo; ++i)
{
        AseGeo* pGeo = &m_pGeo[i];
        D3DXMATRIX mtPrn(1,0,0,0,   0,1,0,0,   0,0,1,0,   0,0,0,1);

        // 자신의 지역행렬 계산
        this->GetAniTrack(&pGeo->mtL, i, m_nFrmC);

        if(pGeo->pGeoPrn)
                mtPrn   = pGeo->pGeoPrn->mtW;

        // 자신의 월드행렬 = 자신의 지역행렬 * 부모 월드행렬
        pGeo->mtW = pGeo->mtL * mtPrn;
...
```

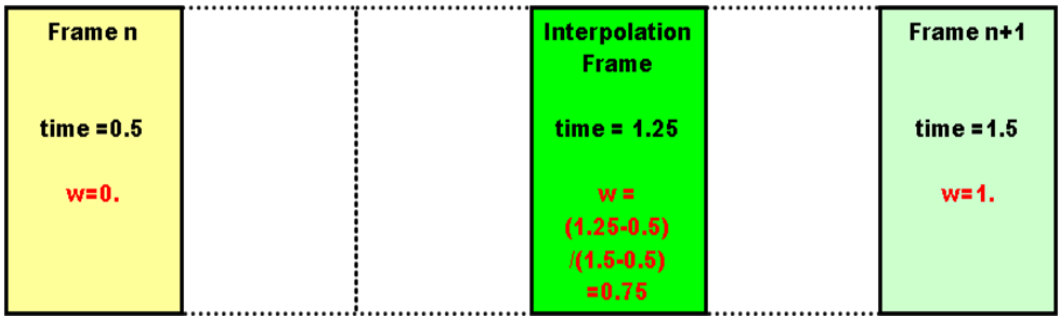


<계층 애니메이션: asa25_hierarchic.zip>

2.5 Interpolation Animation

애니메이션 프레임 간격이 작으면 프로그램 구현이 쉬우나 상대적으로 필요한 메모리가 커집니다. 애니메이션의 종류가 많아질 수록 기하 급수적으로 필요한 메모리가 늘어나게 되는데 눈으로 봤을 때 거의 차이가 없어 보이고 연산이 늘어나도 렌더링에 거의 영향을 주지 않는다면 메모리를 아끼

는 것이 좋을 수도 있습니다.
애니메이션 데이터의 보간(Interpolation)은 렌더링 스피드를 유지한 채 좀 더 다양한 애니메이션을 구현하기 위해서 프레임 수를 줄이고 대신 이들 중간을 계산하는 작업입니다.
예를 들어서 그림과 같이 ASE에 저장된 n 프레임(0.5초)과 n+1 프레임(1.5초) 사이에 현재 애니메이션 타임(1.25 초) 대한 행렬들을 구한다고 생각해 봅시다. 만약 n과 n+1의 시간 간격이 작으면 그냥 n 또는 n+1 프레임에 저장된 행렬을 사용하면 되지만 지금의 예처럼 어느 정도 시간이 있다면 두 프레임(n, n+1)을 사용해서 1.25초에 대한 행렬들을 계산해야 합니다.



<프레임과 비중 값(Weight)>

가장 유연한 방법은 선형 보간입니다. 물론 베지어, 에르미트 보간을 사용해도 되는데 이들은 3차 방정식을 사용하고 있습니다. 그런데 선형 보간은 1차 방정식이라 계산 방법이 무척 쉽고, 렌더링 속도에도 거의 영향이 없습니다.

알려진 두 개의 값 V0, V1 사이에 비중 값(Weight)이 주어진 선형 보간 수식은 두 점이 주어진 직선의 방정식과 동일합니다.

```
V' = (1 - w) * V0 + w * V1        또는
   = V0 + w * (V1 - V0)           <- 직선의 방정식
```

앞의 그림에서 선형 보간에서 비중 값(w)은 시간으로 정해진다고 할 때 두 프레임의 시간과 현재의 시간을 가지고 계산이 됩니다.

```
w = (현재 프레임 시간 - 현재 프레임보다 작거나 같은 프레임 시간) /
    (현재 프레임보다 작거나 같은 프레임 시간 - 다음 프레임 시간)
  = (1.25 - 0.5)/(1.5 - 0.5) = 0.75
```

그림과 다르게 시간 대신 프레임 인덱스를 사용하더라도 w값은 동일한 방식으로 계산이 됩니다.

```
w = (현재 프레임 - 현재 프레임보다 작거나 같은 프레임) /
```

(현재 프레임보다 작거나 같은 프레임 - 다음 프레임)

앞서 우리가 해석한 ASE는 시간 대신 프레임 인덱스를 사용하고 있는데 위치에 대한 w 값, 보간한 위치, 그리고 애니메이션 행렬을 간단히 구현할 수 있습니다.

```
FLOAT w = FLOAT(nFrame- pGeo->vTrs[i].nF)/(pGeo->vTrs[i+1].nF- pGeo->vTrs[i].nF);
p = p1  + w * (p2-p1);
mtA._41 = p.x;  mtA._42 = p.y;  mtA._43 = p.z;
```

회전에서도 비슷하게 적용할 수 있습니다. 먼저 w 계산입니다.

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

사원수에 대한 보간은 sin Θ를 포함한 수식을 사용합니다.

$$\hat{q} = \frac{1}{\sin\theta}\left[\sin\theta(1-t)\ \hat{q}_i + \sin(\theta t)\ \hat{q}_f\right]$$

이 수식은 D3DXQuaternionSlerp() 등으로 구현되어 있습니다.

```
D3DXQuaternionSlerp(&q, &q1, &q2, w);
```

만약 두 프레임 사이의 각도가 작고 덧셈('+') 연산자, 곱셈('*') 연산자가 정의된 DirectX의 D3DXQUATERNION 객체를 사용한다면 선형 보간처럼 사용할 수 있습니다.

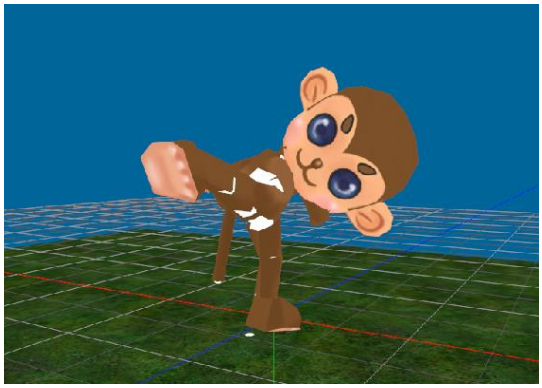
```
q = q1  + w * (q2-q1);
```

보간된 사원수에서 D3DXMatrixRotationQuaternion() 등과 같은 함수를 사용해서 행렬을 얻는다면 회전에 대한 보간은 완료가 됩니다.

```
D3DXMatrixRotationQuaternion(&mtA, &q);
```

```
// 선형 보간을 사용하는 애니메이션 트랙
INT CLcAse::GetAniTrack(void* mtOut, INT nGeo, INT nFrame)
...
// Rotation
```

```
D3DXQUATERNION q;
q1 = (D3DXQUATERNION*)&pGeo->vRot[i];
q2 = (D3DXQUATERNION*)&pGeo->vRot[i+1];
FLOAT   w = FLOAT(nFrame - pGeo->vRot[i].nF)/(pGeo->vRot[i+1].nF- pGeo->vRot[i].nF);

D3DXQuaternionSlerp(&q, q1, q2, w);
D3DXMatrixRotationQuaternion(&mtA, &q);

//
D3DXVECTOR3 p;
p1 = (D3DXVECTOR3*)&pGeo->vTrs[i  ];
p2 = (D3DXVECTOR3*)&pGeo->vTrs[i+1];
FLOAT   w = FLOAT(nFrame- pGeo->vTrs[i].nF)/(pGeo->vTrs[i+1].nF- pGeo->vTrs[i].nF);

p = *p1  + w * (*p2 - *p1);
mtA._41 = p.x;  mtA._42 = p.y;  mtA._43 = p.z;
...
```



asa26_interpolation.zip

ASE 파일의 애니메이션에 대한 중요 부부의 설명은 모두 끝이 났습니다. 좀 더 확실한 학습은 스스로가 직접 하나하나 구현해 보는 것이 가장 좋습니다. 중요한 내용을 몇 가지 정리한다면 다음과 같습니다.

1) 위치를 x, z, y로 해석한다.
2) 위치에 대한 삼각형 인덱스는 a, c, b 이다.
3) 텍스처 좌표는 u, 1.0f - v 이다.
4) 텍스처 좌표 인덱스는 a, c, b 이다.
5) 위치는 텍스처 좌표보다 같거나 작다.
6) 텍스처 좌표 인덱스와 위치 인덱스 수는 같다.

7) 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.
8) TM은 2번째와 3번째 행을 교환한 다음, 2번째 열과 3번째 열 또한 교환한다.
9) ASE는 월드행렬로 구성되어 있어서 자신의 지역행렬 = (자신의 월드행렬) * (부모의 월드행렬의 역행렬)로 구한다.
10) 지역 좌표계 정점 위치 = (ASE에서 구한 위치) * (지오메트리 월드행렬 역행렬) 이다.
11) 애니메이션의 회전 값은 사원수로 저장되어 있으며 누적 값이 아닌 이전 프레임과 상대적인 값이다.
12) 회전은 사원수 보간을 이용한다.