Tutorial for converting Ms3d-animations to DirectX

(written by Chris)

First I would like to apologize for possible language faults since English is not my mother tongue.

This tutorial shows how to programm the most critical parts for a ms3d-viewer in DirectX and is for everyone who does not want to spend forty hours for debugging, how I have done it. One reason why programming this could be tricky is the fact that ms3d-animations use a right-handed coordinate system and DirectX a left-handed one. So the problem is finding the correct order of matrix- and quaternion-multiplications and changing the right signs of rotations, positions and normals.
The following code-snippets shows in a minimalistic way how to solve this problems:

```
// the first four steps can be done when reading the data or
// afterwards how I have done it here

// 1. the z-axis-values of the vertex vectors have to be inverted

for(int i = 0; i < iNumVertices; i++)
{
        m_pVertices[i].vPos.z  *= -1.0f;
        m_pVertices[i].vNorm.z *= -1.0f;
}


// 2. the order of the triangle vertices should be converted from
//    ABC to ACB

for(int i = 0; i < iNumIndices / 3; i++)
{
        WORD wIndexB = m_pIndices[i * 3 + 1];

        m_pIndices[i * 3 + 1] = m_pIndices[i * 3 + 2];
        m_pIndices[i * 3 + 2] = wIndexB;
}


// 3. for creating the relative bonespace matrices the rotation-
//     x- and y-values and the position-z-value have to be inverted

for(int i = 0; i < iNumBones; i++)
{
        m_Bones[i].vRot.x *= -1.0f;
        m_Bones[i].vRot.y *= -1.0f;
        m_Bones[i].vPos.z *= -1.0f;
        //     rotation in XYZ-order !
        Matrix4 m = Matrix4RotationXYZ(m_Bones[i].vRot);
        m._41 = m_Bones[i].vPos.x;
        m._42 = m_Bones[i].vPos.y;
        m._43 = m_Bones[i].vPos.z;
        m_Bones[i].mBoneRel = m;
// 4. the same inverting for the keyframe rotations and positions

        for(int j = 0; j < m_Bones[i].iNumRotKeyFrames; j++)
        {
                m_Bones[i].RotKeyframes[j].vRot.x *= -1.0f;
                m_Bones[i].RotKeyframes[j].vRot.y *= -1.0f;
        }

        for(int j = 0; j < m_Bones[i].iNumPosKeyFrames; j++)
        {
                m_Bones[i].PosKeyframes[j].vPos.z *= -1.0f;
        }

        //    transformation matrices
        m_Bones[i].mAnimAbs = Matrix4Identity();
        m_Bones[i].mBoneAbs  = Matrix4Identity();
}


// 5. bone-setup

// a. calculating the absolute bonespace matrices

if(mpParentBoneAbs)  {  mBoneAbs = m_Bones[i].mBoneRel  *  *mpParentBoneAbs;  }

else {  mBoneAbs = m_Bones[i].mBoneRel;  }

m_Bones[i].mBoneAbs = mBoneAbs;


// b. transforming the vertices and normals with the inverse absolute bonespace matrices;
//     this can be done when the animation is initialized !
for(int j = 0; j <  iNumBones; j++) {
        for(int k = 0; k < m_Bones[j].iNumVertexIndices; k++) {
                m_pBoneSpaceVertices[m_Bones[j].wVertexIndices[k]].vPos =
                   Vec3TransformCoordsInverse(m_pVertices[m_Bones[j].wVertexIndices[k]].vPos,
m_Bones[j].mBoneAbs);
                m_pBoneSpaceVertices[m_Bones[j].wVertexIndices[k]].vNormal =
                   Vec3TransformNormalInverse(m_pVertices[m_Bones[j].wVertexIndices[k]].vNormal,
                   m_Bones[j].mBoneAbs); } }

// 6. animation

// 6.1. rotation interpolation

//       a Quaternion has to be created from the last keyframe rotation (in XYZ-order !)
qL = QuaternionRotationXYZ(m_Bones[dwIndex].RotKeyFrames[i].vRot.x,
m_Bones[dwIndex].RotKeyFrames[i].vRot.y,
m_Bones[dwIndex].RotKeyFrames[i].vRot.z);
```

```
//      a second Quaternion has to be created from the next keyframe rotation (in XYZ-order !)
qN = QuaternionRotationXYZ(m_Bones[dwIndex].RotKeyFrames[i + 1].vRot.x,
                           m_Bones[dwIndex].RotKeyFrames[i + 1].vRot.y,
                           m_Bones[dwIndex].RotKeyFrames[i + 1].vRot.z);

//      spheric linear interpolation
D3DXQuaternionSlerp(&qI, &qL, &qN, fFactor);

//      (if the keyframes are not in the same interval, only one of them is used without interpolation)


// 6.2. position interpolation

//      I take here "ease-in-ease-out"-interpolation (  2 * powf(s, 3.0f) * (v1 - v2) + 3 * powf(s, 2.0f) * (v2 - v1) + v1;  );
//      ( hermite-interpolation is described in msViewer2 )
vPos = Vec3InterpolateCoordsEaseInEaseOut(m_Bones[dwIndex].PosKeyFrames[i].vPos,
                                          m_Bones[dwIndex].PosKeyFrames[i + 1].vPos,  fFactor);

//      (if the keyframes are not in the same interval, only one of them is used without interpolation)



// 6.3. calculating the absolute animation matrices

//      creating the animation matrix from the interpolated quaternion and position
D3DXMATRIX mA;
D3DXMatrixRotationQuaternion(&mA, &qI);
mA._41 = vPos.x;
mA._42 = vPos.y;
mA._43 = vPos.z;

if(mpParentAnimAbs)  {  mP = *mpParentAnimAbs;  }
else  {   mP = Matrix4Identity();   }
m_Bones[i].mAnimAbs = mA * m_Bones[i].mBoneRel * mP;


// 6.4. transforming the vertices and normals with the absolute animation matrices
for(int j = 0; j < iNumBones; j++) {
        for(int k = 0; k < m_Bones[j].iNumVertexIndices; k++) {
                m_pVertices[m_Bones[j].wVertexIndices[k]].vPos =
Vec3TransformCoords(m_pBoneSpaceVertices[m_Bones[j].wVertexIndices[k]].vPos,
m_Bones[j].mAnimAbs);
                m_pVertices[m_Bones[j].wVertexIndices[k]].vNorm =
Vec3TransformNormal(m_pBoneSpaceVertices[m_Bones[j].wVertexIndices[k]].vNorm,
                  m_Bones[j].mAnimAbs); } }
```

I hope this tutorial was helpful, Chris