How to extract local geometry coordinates from 3ds 4 files:
exported/saved at frame 0
using 3ds ftk (file translation kit, available ftp.autodesk.com)

X3ds = sx a1 Xl + sy a2 Yl + sz a3 Zl + Tx
Y3ds = sx b1 Xl + sy b2 Yl + sz b3 Zl + Ty
Z3ds = sx c1 Xl + sy c2 Yl + sz c3 Zl + Tz

where:
(X3ds,Y3ds,Z3ds) = coordinate from 3ds file
(sx,sy,sz) = frame 0 scale
(Tx,Ty,Tz) = frame 0 translation
a1/2/3,b1/2/3,c1/2/3 are the rotation matrix at frame 0.

The object is to solve for (Xl,Yl,Zl) , meshe's _local_ coordinate.

Notice we deal only with export at frame 0 (no interpolation),
if we were to aquire from arbitrary animation frame we would have to know how 3ds interpolates position, rotation, scale *splines*.
Notice how 3ds export ignores the pivot translation.
(a keyframer strictly speaking property, not really part of object definition).

Now down to business: (source code).

The solving is done in BQ (build quaternion mtx):

```
// °°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°
// build mtx to get local coords from frame 0 pos/rot/scale
// coordinates from file (X3ds,Y3ds,Z3ds) are not influenced by pivot !!!
// X3ds=sx rx1 x + sy rx2 y + sz rx3 z + px
// Y3ds=sx ry1 x + sy ry2 y + sz ry3 z + py
// Z3ds=sx rz1 x + sy rz2 y + sz rz3 z + pz
// X3ds,Y3ds,Z3ds = incoming coords from file
// px,py,pz = frame 0 pos
// rx,ry,rz = frame 0 rotate
// 3ds4 gives you [ a , [x,y,z] ]  where a=angle and (x,y,z] is the rotation axis.
// The equivalent Quaternion is :
// [ cos (a/2), x*sin(a/2), y*(sin(a/2), z*sin(a/2) ]
// sx,sy,sz = frame 0 scale
void bQ (INST_T *c)
{
 register MTX_T* m=&c->Itrihedral;
 VEC_T          p,s;
 ROT_T          r;

 p.x=p.y=p.z=0;// 1st key must be there
 r=c->source_obj->rkeys[0];
 s.x=s.y=s.z=1;

 QuaternionMatrix (&c->trihedral, &p, &r, &s);
 MTranspose (&c->trihedral, m);// rotation being orthogonal, inverse=transpose

 p.x=-c->source_obj->pkeys[0].x;
 p.y=-c->source_obj->pkeys[0].y;
 p.z=-c->source_obj->pkeys[0].z;
 TBR(m,&p);
```

```
 s=c->source_obj->skeys[0];
 if(fabs(s.x) < EPSILON && fabs(s.y) < EPSILON && fabs(s.z) < EPSILON)
        {
         sprintf(exitMsg,"Fatal: object scale=0,0,0 at frame 0 !\n");
         exit(-1);
        };

 m->m[0][0] /= s.x;
 m->m[0][1] /= s.x;
 m->m[0][2] /= s.x;
 m->m[0][3] /= s.x;

 m->m[1][0] /= s.y;
 m->m[1][1] /= s.y;
 m->m[1][2] /= s.y;
 m->m[1][3] /= s.y;

 m->m[2][0] /= s.z;
 m->m[2][1] /= s.z;
 m->m[2][2] /= s.z;
 m->m[2][3] /= s.z;

}
```

Since the child (of hierarchy) meshes coordinates are given in parent's space, we must solve concatenating with parent matrix, RC (reverse concatenate):

```
// reverse concatenation on 3ds hierarchy
// returns 3ds to local coordinate matrix
// °°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°
MTX_T RC (char *name, MTX_T m, database3ds* db)
{
 kfmesh3ds* k=0;
 MTX_T      tmpm;
 OBJ_T      O;
 INST_T     I;
 VEC_T      pos;
 VEC_T      sca;
 ROT_T      rot;

        GetObjectMotionByName3ds(db, name,  &k);
        ON_ERROR_RETURNR(tmpm);

        I.source_obj=&O;
        O.pkeys=&pos;
        O.rkeys=&rot;
        O.skeys=&sca;

        O.pkeys[0].x=k->pos[0].x;
        O.pkeys[0].y=k->pos[0].y;
        O.pkeys[0].z=k->pos[0].z;

        O.rkeys[0].qr.w=k->rot[0].angle;
        O.rkeys[0].qr.x=k->rot[0].x;
        O.rkeys[0].qr.y=k->rot[0].y;
        O.rkeys[0].qr.z=k->rot[0].z;
        Angle2Quat(&O.rkeys[0]);
```

```
        O.skeys[0].x=k->scale[0].x;
        O.skeys[0].y=k->scale[0].y;
        O.skeys[0].z=k->scale[0].z;

        bQ (&I);
        MMult ( &I.Itrihedral, &m, &tmpm);
        m=tmpm;

        if(k->parent[0])
          m=RC(k->parent, m , db);

        ReleaseObjectMotion3ds(&k);
        ON_ERROR_RETURNR(tmpm);

        return m;// copy on stack
}
```

call RC () with: keyframer mesh name "name", identity matrix m, database3ds db(the file being parsed).

This will return a matrix that converts from 3ds semi-absolute coordinates to local.

Now you are ready to feed these local coordinates to a matrix that will instantiate them to world: which is what you'll do to use them in your 3d keyframer engine.

Xabs = sx a1 (Xl-Px) + sy a2 (Yl-Py) + sz a3 (Zl-Pz) + Tx
Y3ds = sx b1 (Xl-Px) + sy b2 (Yl-Py) + sz b3 (Zl-Pz) + Ty
Z3ds = sx c1 (Xl-Px) + sy c2 (Yl-Py) + sz c3 (Zl-Pz) + Tz

Where:

(Xabs,Yabs,Zabs) = absolute coordinate
(Px,Py,Pz) = mesh pivot (constant)

This matrix needs concatenation for child instances.I have found that the pivot subtraction (for rotating about pivot), must not be concatenated, this is quite unfortunate so that we need to concatenate scale, rotation, translation, and then subtract (TBR function) only when we actually make the local to world matrix.

Do this to mtx m returned from FC():

```
 p.x=-c->source_obj->pkeys[0].x;
 p.y=-c->source_obj->pkeys[0].y;
 p.z=-c->source_obj->pkeys[0].z;
 TBR(m,&p);
```

This is forward concatenate FC: returns the local-->world matrix.

```
// °°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°
// fwd concatenation on 3ds hierarchy (test)
// compound instance mtx (trihedral) of k with parent, recurse and ret final instance mtx
// pass+return on stack
MTX_T FC (char *name, MTX_T m, database3ds* db)
{
 kfmesh3ds* k=0;
 MTX_T      tmpm, trihedral;
 VEC_T      pos;
 ROT_T      rot;
 VEC_T      sca;
```

```
	GetObjectMotionByName3ds(db, name,  &k);
	ON_ERROR_RETURNR(tmpm);

	pos.x=k->pos[0].x;
	pos.y=k->pos[0].y;
	pos.z=k->pos[0].z;

	rot.qr.w=k->rot[0].angle;
	rot.qr.x=k->rot[0].x;
	rot.qr.y=k->rot[0].y;
	rot.qr.z=k->rot[0].z;
	Angle2Quat(&rot);

	sca.x=k->scale[0].x;
	sca.y=k->scale[0].y;
	sca.z=k->scale[0].z;

	QuaternionMatrix (&trihedral, &pos, &rot, &sca);
	MMult (&m, &trihedral, &tmpm);// concatenate with parent
	m=tmpm;

	if(k->parent[0])
	 m=FC (k->parent, m , db);

	ReleaseObjectMotion3ds(&k);
	ON_ERROR_RETURNR(tmpm);

	return m;// copy on stack

}
```

Now some typedefs and miscellaneous functions to make all above.

```
// °°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°
#define EPSILON 0.0001
char exitMsg[128];



typedef struct
{
	float   x,y,z;
} VEC_T;

typedef struct
{
	float   h,p,r;// heading,pitch,roll
} EULER_T;

typedef struct
{
	float   w,x,y,z;// a quaternion (real angle, imaginary dir cosines of axis of rotation)
} QUAT_T;

typedef union
{	// rotation may be euler angles or quaternion so we use a union
	EULER_T  er;
	QUAT_T   qr;
} ROT_T;
```

```
typedef struct
{       // 3 rows 4 cols
        float   m[3][4];
}  MTX_T;


typedef struct _OBJ_T              // object geometry definition
{
        int     npkeys;            // n of pos keys
        VEC_T*  pkeys;             // pos keys[]
        int*    tpkeys;            // time of pos key
        int     nrkeys;            // n of rot keys
        ROT_T*  rkeys;             // rot keys[]
        int*    trkeys;            // time of rot key
        int     nskeys;            // n of scale keys
        VEC_T*  skeys;             // scale keys[]
        int*    tskeys;            // time of scale key
        VEC_T   pivot;             // rotation pivot
} OBJ_T;

//  This structure describes a particular instance of object.
typedef struct _INST_T {
        struct _INST_T* parent;          // instance hierarchy (replicating the geometry hierarchy, so we
can rt the children)
        OBJ_T         * source_obj;     // what it looks like, this can be a root/child mesh
        MTX_T           trihedral;      // transforms local[x, y, z] to world
        MTX_T           Itrihedral;     // transforms world to local[x, y, z]
        } INST_T;


static MTX_T ident = {
        1.0, 0.0, 0.0, 0.0,
        0.0, 1.0, 0.0, 0.0,
        0.0, 0.0, 1.0, 0.0
        };

// °°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°
void IdentMatrix (MTX_T *Mtx)
{
        *Mtx = ident;
}

// pre translation (before rotation)
// °°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°
void TBR(MTX_T* m,VEC_T* p)
{
        m->m[0][3] += (m->m[0][0]*p->x + m->m[0][1]*p->y + m->m[0][2]*p->z);
        m->m[1][3] += (m->m[1][0]*p->x + m->m[1][1]*p->y + m->m[1][2]*p->z);
        m->m[2][3] += (m->m[2][0]*p->x + m->m[2][1]*p->y + m->m[2][2]*p->z);
}

// post translation (after rotation)
// °°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°
void TAR(MTX_T* m,VEC_T* p)
{
        m->m[0][3] += p->x;
        m->m[1][3] += p->y;
        m->m[2][3] += p->z;
}
```

```
//  works for inverse of orthogonal mtx (conserves distances)
void MTranspose (MTX_T *m,MTX_T *r)
{
 register int i, j;

        for (i=0; i< 3; ++i)
                for (j=0; j< 3; ++j)
                        r->m[i][j] = m->m[j][i];


 r->m[0][3] =-m->m[0][0]*m->m[0][3]-m->m[1][0]*m->m[1][3]-m->m[2][0]*m->m[2][3];
 r->m[1][3] =-m->m[0][1]*m->m[0][3]-m->m[1][1]*m->m[1][3]-m->m[2][1]*m->m[2][3];
 r->m[2][3] =-m->m[0][2]*m->m[0][3]-m->m[1][2]*m->m[1][3]-m->m[2][2]*m->m[2][3];

}

// °°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°
// this is actually R=M2*M1 (mtx mult not commutative)
// 4th row=0001
void MMult (MTX_T *Mt1,MTX_T *Mt2,MTX_T *R)
{
 register int i;

 for (i=0; i< 3; i++)
 {
 R->m[i][0] =
 (Mt2->m[i][0]*Mt1->m[0][0])+(Mt2->m[i][1]*Mt1->m[1][0])+(Mt2->m[i][2]*Mt1->m[2][0]);
 R->m[i][1] =
 (Mt2->m[i][0]*Mt1->m[0][1])+(Mt2->m[i][1]*Mt1->m[1][1])+(Mt2->m[i][2]*Mt1->m[2][1]);
 R->m[i][2] =
 (Mt2->m[i][0]*Mt1->m[0][2])+(Mt2->m[i][1]*Mt1->m[1][2])+(Mt2->m[i][2]*Mt1->m[2][2]);
 R->m[i][3] =
 (Mt2->m[i][0]*Mt1->m[0][3])+(Mt2->m[i][1]*Mt1->m[1][3])+(Mt2->m[i][2]*Mt1->m[2][3])+Mt2-
>m[i][3];
 }

}

// °°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°
// add/remove scale from rotation part, m CAN == sm
void MScale(MTX_T *sm, VEC_T* s, MTX_T *m)
{

    m->m[0][0]=sm->m[0][0]*s->x;
    m->m[0][1]=sm->m[0][1]*s->y;
    m->m[0][2]=sm->m[0][2]*s->z;

    m->m[1][0]=sm->m[1][0]*s->x;
    m->m[1][1]=sm->m[1][1]*s->y;
    m->m[1][2]=sm->m[1][2]*s->z;

    m->m[2][0]=sm->m[2][0]*s->x;
    m->m[2][1]=sm->m[2][1]*s->y;
    m->m[2][2]=sm->m[2][2]*s->z;

}

#define QMAGP(q) (q->qr.x * q->qr.x + q->qr.y * q->qr.y + q->qr.z * q->qr.z + q->qr.w * q->qr.w)
```

```
// °°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°
void QuatNormalize(ROT_T *q)
{
 float norm;

       norm = QMAGP(q);

       if (norm > EPSILON)
       {
        norm = (float)(1.0 / sqrt(norm));

        q->qr.x *= norm;
        q->qr.y *= norm;
        q->qr.z *= norm;
        q->qr.w *= norm;
       }

}

// °°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°
void Angle2Quat(ROT_T *q)
{
 float s,m;

        // normalise rotn axis vector
        m=sqrt(q->qr.x*q->qr.x + q->qr.y*q->qr.y + q->qr.z*q->qr.z);
        if(m > EPSILON)
        {
         q->qr.x/=m;
         q->qr.y/=m;
         q->qr.z/=m;
        }

        q->qr.w*= 0.5;

        s=sin(q->qr.w);
        q->qr.w=cos(q->qr.w);

        q->qr.x*=s;
        q->qr.y*=s;
        q->qr.z*=s;


        QuatNormalize(q);
}


// °°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°
// 3ds4 gives you [ a , [x,y,z] ]  where a=angle and (x,y,z] is the rotation axis.
// The equivalent Quaternion is :
// [ cos (a/2), x*sin(a/2), y*(sin(a/2), z*sin(a/2) ]
// from Watt&Watt pag. 362
void QuaternionMatrix (MTX_T* m, VEC_T *p, ROT_T *r, VEC_T *s)
{
 float  W, X, Y, Z;

        W=r->qr.w; X=r->qr.x; Y=r->qr.y; Z=r->qr.z;

        m->m[0][0] = (1.0 - (2*(Y*Y + Z*Z)) );
```

```
	m->m[0][1] = 2*(X*Y  + W*Z);
	m->m[0][2] = 2*(X*Z  - W*Y);

	m->m[1][0] = 2*(X*Y  - W*Z);
	m->m[1][1] = (1.0 - (2*(X*X + Z*Z)) );
	m->m[1][2] = 2*(Y*Z  + W*X);

	m->m[2][0] = 2*(X*Z  + W*Y);
	m->m[2][1] = 2*(Y*Z  - W*X);
	m->m[2][2] = (1.0 - (2*(X*X + Y*Y)) );

/* this way of concatenating scale is equivalent to mult w scale mtx:
[sx][0] [0]
[0] [sy][0]
[0] [0] [sz]
*/
	MScale(m, s, m);

	m->m[0][3] = p->x;
	m->m[1][3] = p->y;
	m->m[2][3] = p->z;

}
```

Don't forget #include "3dsftk.h" in your 3ds parser file.

Todo:

1)the locmatrix[] matrix in 3ds 4 file: is it the same as obtained from RC() ? (at frame 0)
2)ftk is unsupported by autodesk, can they confirm this information ? why didn't they just save _local_ coordinates ? Why no such explanation ?

Some _answers_ from Autodesk : (step back in awe...)

I tracked down the original author of the FTK, he does not work for Autodesk anymore, and asked him your questions:

<< 1)the locmatrix[] matrix in 3ds 4 file: is it the same as obtained from RC() ? (at frame 0) >>

Almost, but not quite. The locmatrix also contains hints about orientation of an object that the keyframe data would not. In particular, the locmatrix will have rotations in it that were affected by the viewport the object was first created in. It's the only way 3DS can tell what "front, back, top, bottom, etc." of an object is. That's why the locmatrix will have negative signs in the rotation lines of the matrix. If you want to know what the objects local coordinate system is, just invert the locmatrix and multiply the vertices out. That's what 3DS does before it applies the rotations and positions of the very first key.

A little history here. In release one of 3DS, the locmatrix and the first key frame DIDN'T MATCH at all unless you asked the keyframer to update. Now that updating of position is automatic. Some of the legacy still remains (like in duck.3ds) so its best to use the inverse of the locmatrix to get from post transform to pre transform.

<< 2)ftk is unsupported by Autodesk, can they confirm this information ? why didn't they just save _local_ coordinates ? Why no such explanation ? >>

In the prehistory of 3DS it was a 16 bit application written for 640k dos.

Prior to that, it was a product on an Atari ST. At that time it didn't have a sophisticated key frame interpolation system AND floating point was an expensive operation. The mesh was represented in its post transform state to speed up editing operations like move vertex.

3DS Max saves "local" cooridinates, although really it's a coordinate system multiply removed from the actual scene representation.

3)I need a job as 3d programmer, have any work, anywhere ?

Guglielmo Fanini
gufanini@tin.it
Milan, Italy
14 Aug 98