Topic 4:

3D QuickTime Movies

3D ON THE WEB

3D on the Web these days is usually done by pre-rendering animating gif's or QuickTime movies, and VRML really hasn't taken off mainly because it's too difficult to create VRML data and no one can seem to get VRML viewers to install and remain stable. But if all you want to do is put a 3D model on a web site and apply some basic animation to it, then QuickTime and QuickDraw 3D is definitely the way to go. It's stable, everyone has it (especially with QuickTime 3.0 which comes with QuickDraw 3D by default), and creating the data is very simple. Unlike VRML, however, 3D models in a QuickTime movie are not interactive... yet. The QuickTime movie simply plays back a pre-recorded sequence. Nonetheless, this is still better than pre-rendering a 3D animation because there is no image quality loss when you enlarge the playback window, plus the files are much smaller and faster to download from a web site.

I'm not going to turn this chapter into a QuickTime tutorial, but I will cover the basic information you need to know in order to create QuickTime movies which contain animating 3D models. QuickTime movies let you create many different types of media tracks, and one of those track types is a 3D Track. Another type of track is the "tween" track. A tween track is essentially the animation track, and it let's you interpolate the orientation of a 3D model over a period of time.

Unfortunately, creating a 3D movie is a fairly complex process with a lot of steps. I'll try to explain what each step is doing, but if you don't understand everything, don't worry about it. As long as you can use the code, it isn’t completely critical that you understand why it works.

STEP 1: WRITING A 3DMF FILE

The 3D data contained in a QuickTime movie is actually a 3DMF file. Your movie can either reference an actual 3DMF file to use (an alias to the file) or you can embed the 3DMF file binary data in the QuickTime movie. For our purposes, we will want to embed the data in the movie so that it is all self-contained.

First we are going to write a function called `Make3DMovie`. This is the function to which you pass the QuickDraw 3D geometry you want in the QuickTime movie:

```
/***************** MAKE 3D MOVIE ******************/
//
// INPUT: myModel =   reference to 3D data we want to put
//                    in the QT movie.
//        myView  =   reference to a View object to use
//                    for various things.
//

void Make3DMovie(TQ3Object myModel, TQ3ViewObject myView)
{
FSSpec                spec;
short                 tempFolderVRefNum;
long                  tempFolderDirID;
TQ3ViewHintsObject    viewHintsObj;

  /**************************/
  /* INITIALIZE SOME STUFF */
  /**************************/

    /* CREATE A VIEW HINTS OBJECT BASED ON THIS VIEW */

viewHintsObj = Q3ViewHints_New(myView);


      /****************/
      /* SAVE TO 3DMF */
      /****************/

  /* WE’RE GOING TO PUT TEMPORARY 3DMF FILE INTO TEMP FOLDER */

  FindFolder(kOnSystemDisk,kTemporaryFolderType,kCreateFolder,
         &tempFolderVRefNum,&tempFolderDirID);
      /* CREATE FSSPEC FOR TEMP 3DMF FILE */

  FSMakeFSSpec(tempFolderVRefNum, tempFolderDirID, "\ptemp.3dmf",
            &spec);


        /* SAVE THE 3DMF FILE */

  SaveMy3DMFModel(myModel, viewHintsObj, myView, &spec);


      /*************************/
      /* BUILD QUICKTIME MOVIE */
      /*************************/

  RecordMyMovie(&spec);


      /***********/
      /* CLEANUP */
      /***********/

  Q3Object_Dispose(viewHintsObj);          // nuke the view hints object
  FSpDelete(&spec);                        // nuke the temp 3dmf file
}
```

The first thing this code does is create a View Hints object. This is critical since this object will tell QuickTime where to put the camera, what background color to use, etc. Without a view hints object, you will probably get a big, blank QuickTime movie.

Next, we go ahead and write the 3DMF file to a temporary file in the Temporary Files folder in the System folder. This 3DMF file is only used while creating the QuickTime movie, thus, we delete it at the end when we are done.

In case you are not familiar with the process of saving a 3DMF file, here is the code for SaveMy3DMFModel:

```
/*************** SAVE MY 3DMF MODEL **************/
//
// INPUT: theModel      =  ref to model to save to file
//        viewHintsObj  = ref to view hints object to put in file
//        myView        = ref to View object
//        myFSSpec      =  fsspec of file to save to.
//

void Save3DMFModel(TQ3Object theModel, TQ3ViewHintsObject viewHintsObj,
                 TQ3ViewObject myView, FSSPec myFSSpec)
```

```
{
TQ3FileObject    fileObj;
TQ3Object        theModel;
TQ3Status        myStatus;
TQ3ViewStatus    myViewStatus;
OSErr            iErr;
TQ3StorageObject myStorageObj;

     /*****************************/
     /* CREATE THE NEW 3DMF FILE */
     /*****************************/

  FSpDelete(&myFSSpec);                   // delete any old one
  iErr = FSpCreate(&myFSSpec, 'ttxt', '3DMF', smSystemScript);
  if (iErr)
     DoError("\pFSpCreate failed!");


       /************************/
       /* CREATE A FILE OBJECT */
       /************************/

  /* CREATE NEW STORAGE OBJECT WHICH IS THE 3DMF FILE */

  myStorageObj = Q3FSSpecStorage_New(myFSSpec);
  if (myStorageObj == nil)
     DoError("\pQ3FSSpecStorage failed!");


       /* CREATE NEW FILE OBJECT */

  fileObj = Q3File_New();
  if (fileObj == nil)
     DoError("\pQ3File_New failed!");


       /* SET THE STORAGE FOR THE FILE OBJECT */

  if (Q3File_SetStorage(fileObj, myStorageObj) == kQ3Failure)
     DoError("\pQ3File_SetStorage failed!");

  Q3Object_Dispose(myStorageObj);


  /*******************************/
  /* OPEN 3DMF FILE FOR WRITING */
  /*******************************/

  myStatus = Q3File_OpenWrite(fileObj, kQ3FileModeNormal);
  if (myStatus != kQ3Success)
     DoError("\pQ3File_OpenWrite failed!");


     /***********************/
     /* WRITE THE 3DMF FILE */
     /***********************/

       /* START WRITING PROCESS */
  myStatus = Q3View_StartWriting(myView,fileObj);
  if ( myStatus != kQ3Success )
     DoError("\pQ3View_StartWriting failed!");

         /* SUBMIT LOOP */
     do
     {
       Q3Object_Submit(viewHintsObj, myView);
       Q3Geometry_Submit(theModel, myView);
       myViewStatus = Q3View_EndWriting(myView);
     } while (myViewStatus == kQ3ViewStatusRetraverse);


     /***********/
     /* CLEANUP */
     /***********/

  Q3File_Close(fileObj);
  Q3Object_Dispose(fileObj);
}
```

STEP 2: CREATING THE MOVIE FILE

After saving the 3DMF file to the Temporary Items folder, we call `RecordMyMovie` to actually create a QuickTime movie from the 3D data:

```
/******************** RECORD MY MOVIE ***************************/
//
// INPUT: fsSpec = 3DMF file to include in movie.
//

void RecordMyMovie(FSSpec *fsSpec)
{
Movie       theMovie,movie2;
short       resRefNum ;
short       resId = 0;
OSErr       iErr;
StandardFileReply  reply;
FSSpec      *specPtr,tempSpec;
Handle      tempH;
short       tempFolderVRefNum;
long        tempFolderDirID;

          /*******************************/
          /* LET USER SET FILE SAVE INFO */
          /*******************************/

  StandardPutFile("\pSave Movie As...","\pMy3DMovie.mov",&reply);
  if (!reply.sfGood)
    return;
  specPtr = &reply.sfFile;
```

```
     /*********************************/
     /* CREATE NEW SCRATCH MOVIE FILE */
     /*********************************/

   /* ONCE AGAIN, WE’LL SAVE THIS TO THE TEMP FOLDER */

FindFolder(kOnSystemDisk,kTemporaryFolderType,kCreateFolder,
            &tempFolderVRefNum,&tempFolderDirID);

FSMakeFSSpec(tempFolderVRefNum, tempFolderDirID, "\ptemp.mov",
            &tempSpec);


        /* CREATE THE SCRATCH MOVIE */

iErr = CreateMovieFile (&tempSpec,
                        'TVOD',
                        smCurrentScript,
                        createMovieFileDeleteCurFile,
                        &resRefNum,
                        &theMovie);
if (iErr)
   DoError("\pCreateMovieFile failed!");


        /*********************/
        /* SET MOVIE TO LOOP */
        /*********************/

tempH = NewHandleClear(sizeof(long));
AddUserData(GetMovieUserData(theMovie), tempH, 'LOOP');
DisposeHandle(tempH);


     /********************************/
     /* ADD THE 3D DATA TO THE MOVIE */
     /********************************/

CreateMyMovie3DTrack(theMovie, fsSpec);


     /******************************************/
     /* NOW TURN SCRATCH MOVIE INTO REAL MOVIE */
     /******************************************/

movie2 = FlattenMovieData(theMovie, flattenAddMovieToDataFork,
                          specPtr, 'TVOD', smCurrentScript,
                          createMovieFileDeleteCurFile);
if (movie2 == nil)
   DoError("\pFlattenMovieData failed!");


     /**************************/
     /* CLOSE THE SCRATCH FILE */
     /**************************/
```

```
  iErr = CloseMovieFile(resRefNum);
  if (iErr)
    DoFatalAlert("\pRecordNewMovie: CloseMovieFile failed!");


      /***********/
      /* CLEANUP */
      /***********/

  DisposeMovie(movie2);
  DisposeMovie(theMovie);
  DeleteMovieFile(&tempSpec);
}
```

This function outlines the basic process of creating the QuickTime movie. We need to build our movie, so we create a scratch file in the Temporary Items folder to work with. The function `CreateMyMovie3DTrack` will add the 3D data to the movie and is listed below. Once our scratch movie is created, we call `FlattenMovieData` which essentially merges the QuickTime data with the 3DMF data and allows us to create a new movie which has the 3D data embedded in it.

Once we've flattened the movie, the final movie file has been created. All we do from here is close a few files, dispose of the movie data, and delete the scratch movie file.

STEP 3: THE 3D & TWEEN TRACKS

So, the easy part is over. Let the fun begin! Creating the 3DMF file and the movie file was easy, but adding the 3D and animation data to the movie is a very messy process. It all makes sense, but it's cumbersome so take it slow.

```
#define   kMovieWidth    300       // dimensions of the movie to make
#define   kMovieHeight   300

#define   kMovieTimeScale  100     // # samples per second in movie

#define   kStartScale    1.0       // scale from start to end value
#define   kEndScale      2.0

#define   kStartRotY     0.0       // rot on y-axis from start to end
#define   kEndRotY       (kQ3Pi * 2)

#define   kStartX     0.0          // move from start to end coords
```

```
#define   kStartY     0.0
#define   kStartZ     -300.0
#define   kEndX       0.0
#define   kEndY       0.0
#define   kEndZ       300.0


#define   kDataCompressorType zlibDataCompressorSubType

enum                              // enums to help me out
{
  TRANS_ID_SCALE = 1,
  TRANS_ID_ROTY,
  TRANS_ID_MOVE
};

/************** CREATE MY MOVIE 3D TRACK ***************/
//
// Creates the movie video track and adds our animation to it.
//

void CreateMyMovie3DTrack(Movie theMovie, FSSpec *the3DMFFile)
{
Track       theTrack,tweenTrack;
Media       theMedia,tweenMedia;
OSErr       iErr;
ThreeDeeDescriptionHandle sampleDescription = nil;
long        eof;
short       fref;
TQ3Vector3D tweenTranslate;
long        referenceIndex1, referenceIndex2, referenceIndex3;
QTAtomContainer inputMap;
QTAtomContainer tweenSample;
TQ3Vector3D tweenScale;
TQ3RotateTransformData tweenRotate;
Handle      the3DMFData,compressed3DMFData = nil,dataToUse;

       /*********************************/
       /* READ IN THE 3DMF BINARY DATA */
       /*********************************/

  FSpOpenDF(the3DMFFile, fsRdPerm, &fref);      // open 3DMF file
  GetEOF(fref, &eof);                           // get size of file
  the3DMFData = NewHandleClear(eof);            // get mem for it
  HLock(the3DMFData);
  FSRead(fref, &eof, *the3DMFData);             // read the data
  FSClose(fref);                                // close the file


     /***********************************************/
     /* MAKE A SAMPLE DESCRIPTION FOR THE 3D MEDIA */
     /***********************************************/

  sampleDescription = (ThreeDeeDescriptionHandle)NewHandleClear(
                      sizeof(ThreeDeeDescription));
  (**sampleDescription).descSize = sizeof(ThreeDeeDescription);

          /**********************/
```

```
        /* CREATE THE 3D TRACK */
        /***********************/

  theTrack = NewMovieTrack (theMovie,
              FixRatio(kMovieWidth,1),
              FixRatio(kMovieHeight,1),
              kNoVolume);

  if (theTrack == nil)
    DoError("\pNewMovieTrack failed!");


        /*************************/
        /* CREATE THE TRACK MEDIA */
        /*************************/

  theMedia = NewTrackMedia (theTrack,
                        ThreeDeeMediaType,
                        kMovieTimeScale,
                        0,nil);
  if (theMedia == nil)
    DoError("\pNewTrackMedia failed!");


          /***********************/
          /* COMPRESS THE 3D DATA */
          /***********************/

  iErr = CompressMyHandle(the3DMFData, &compressed3DMFData);
  if (iErr)
    dataToUse = the3DMFData;
  else
  {
    (**sampleDescription).decompressorType = kDataCompressorType;
    dataToUse = compressed3DMFData;
  }

          /*****************************/
          /* ADD THE 3DMF DATA AS MEDIA */
          /*****************************/

  iErr = BeginMediaEdits(theMedia);
  if (iErr)
    DoError("\pBeginMediaEdits failed!");

  iErr = AddMediaSample(theMedia,
                         dataToUse,
                         0,
                         GetHandleSize(dataToUse),
                         kMovieTimeScale*5,
                         (SampleDescriptionHandle)sampleDescription,
                         1,            // one sample
                         0,            // self-contained samples
                         nil);
  if (iErr)
    DoError("\pAddMediaSample failed!");

  EndMediaEdits(theMedia);
```

```
            /*********************************/
            /* ADD THE 3D MEDIA TO THE TRACK */
            /*********************************/

InsertMediaIntoTrack(theTrack, 0, 0, GetMediaDuration(theMedia),
                      kFix1);
DisposeHandle((Handle)sampleDescription);



            /******************/
            /* DO TWEEN TRACK */
            /******************/

            /* MAKE TWEEN TRACK */

tweenTrack = NewMovieTrack(theMovie, 0, 0, 0);


            /* NEW TWEEN MEDIA */

tweenMedia = NewTrackMedia(tweenTrack, TweenMediaType,
                           gMovieTimeScale, nil, 0);


          /* MAKE EMPTY TWEEN SAMPLE */

iErr = QTNewAtomContainer(&tweenSample);
if (iErr)
   DoError("\pQTNewAtomContainer failed!");


       /*************************/
       /* ADD TWEEN ENTRY: SCALE */
       /*************************/

            /* SET ENDING SCALE VALUE */

tweenScale.x =
tweenScale.y =
tweenScale.z = kEndScale;
AddMyTweenEntryToSample(tweenSample, TRANS_ID_SCALE,
                      kTweenType3dScale, &tweenScale,
                      sizeof(tweenScale));

            /* SET INITIAL SCALE VALUE */

tweenScale.x =
tweenScale.y =
tweenScale.z = kStartScale;
SetMyTweenEntryInitialConditions(tweenSample, TRANS_ID_SCALE,
                                  &tweenScale, sizeof(tweenScale));

          /*****************************/
          /* ADD TWEEN ENTRY: ROTATE-Y */
          /*****************************/
```

```
        /* SET ENDING ROTATE VALUE */

tweenRotate.axis = kQ3AxisY;
tweenRotate.radians = kEndRotY;
addTweenEntryToSample(tweenSample, TRANS_ID_ROTY, kTweenType3dRotate,
                      &tweenRotate, sizeof(tweenRotate));

        /* SET STARTING ROTATE VALUE */

tweenRotate.axis = kQ3AxisY;
tweenRotate.radians = kStartRotY;
setTweenEntryInitialConditions(tweenSample, TRANS_ID_ROTY,
                             &tweenRotate, sizeof(tweenRotate));


     /*******************************/
     /* ADD TWEEN ENTRY: TRANSLATE */
     /*******************************/


          /* SET ENDING COORDINATE */

tweenTranslate.x = kEndX;
tweenTranslate.y = kEndY;
tweenTranslate.z = kEndZ;
addTweenEntryToSample(tweenSample, TRANS_ID_MOVE,
                      kTweenType3dTranslate, &tweenTranslate,
                      sizeof(tweenTranslate));


        /* SET STARTING COORDINATE */

tweenTranslate.x = kStartX;
tweenTranslate.y = kStartY;
tweenTranslate.z = kStartZ;
setTweenEntryInitialConditions(tweenSample, TRANS_ID_MOVE,
                           &tweenTranslate, sizeof(tweenTranslate));


     /**********************************/
     /* MAKE TWEEN SAMPLE DESCRIPTION */
     /**********************************/

sampleDescription = (ThreeDeeDescriptionHandle)NewHandleClear(
                    sizeof(SampleDescription));
(**sampleDescription).descSize = sizeof(SampleDescription);


     /****************************************/
     /* ADD TWEEN SAMPLE TO THE TWEEN MEDIA */
     /****************************************/

BeginMediaEdits(tweenMedia);

iErr = AddMediaSample(tweenMedia, tweenSample, 0,
                    GetHandleSize(tweenSample), kMovieTimeScale*5,
                  (SampleDescriptionHandle)sampleDescription, 1,
                   0, nil);
if (iErr)
   DoError("\pAddMediaSample failed!");

EndMediaEdits(tweenMedia);


         /***********************************/
         /* ADD TWEEN MEDIA INTO THE TRACK */
         /***********************************/

InsertMediaIntoTrack(tweenTrack, 0, 0, GetMediaDuration(tweenMedia),
                     kFix1);

QTDisposeAtomContainer(tweenSample);    // throw away a few things
DisposeHandle((Handle)sampleDescription);


       /************************************************/
       /* CREATE REFERENCES BETWEEN 3D AND TWEEN TRACKS */
       /************************************************/


AddTrackReference(theTrack, tweenTrack, kTrackModifierReference,
                  &referenceIndex1);
AddTrackReference(theTrack, tweenTrack, kTrackModifierReference,
                  &referenceIndex2);
AddTrackReference(theTrack, tweenTrack, kTrackModifierReference,
                  &referenceIndex3);


   /**************************************/
   /* CREATE INPUT MAP FOR THE 3D TRACK */
   /**************************************/

QTNewAtomContainer(&inputMap);


     /* ADD ENTRIES TO INPUT MAP */
     //
     // NOTE:  This order seems to determine how transforms
     // are applied.  Do transform *then* rotate to get
     //  a rot->trans concatenation.  In other words,
     // do in reverse order that you want transforms
     // applied.
     //

AddMyTweenEntryToInputMapEntry(inputMap, referenceIndex1,
                     TRANS_ID_MOVE, kTrackModifierType3d4x4Matrix,
                     nil);
AddMyTweenEntryToInputMapEntry (inputMap, referenceIndex2,
                     TRANS_ID_ROTY, kTrackModifierType3d4x4Matrix,
                     nil);
AddMyTweenEntryToInputMapEntry (inputMap, referenceIndex3,
                     TRANS_ID_SCALE, kTrackModifierType3d4x4Matrix,
                     nil);
```

```
      /* ATTACH INPUT MAP TO 3D MEDIA HANDLER */

  SetMediaInputMap(theMedia, inputMap);


            /***********/
            /* CLEANUP */
            /***********/

  QTDisposeAtomContainer(inputMap);
  DisposeHandle(the3DMFData);
  if (compressed3DMFData != nil)
    DisposeHandle(compressed3DMFData);
}
```

Okay, the above function is a monster, I know. The general flow of events goes like this:

1. Load the 3DMF file's binary data into memory.
2. Create the 3D track & 3D media
3. Compress the 3D data.
4. Assign the 3D data to the media.
5. Assign the media to the track.
6. Create the Tween track & media.
7. Create a tween "sample".
8. Add scale, rotate, and translate information to the sample.
9. Add the sample to the tween media.
10. Add the tween media to the tween track.
11. Create references between the 3D and Tween tracks.
12. Cleanup

When you look at it broken down like this, it starts to make a little more sense. You can get more information on creating tween tracks by downloading the “Tween Media Handler” documentation off of the QuickTime web site at www.quicktime.apple.com.

The code above added a scale, y-axis rotate, and a translate tween to the movie, but we are not limited to this. We could have done just a rotate, or just a translate tween. We could have also added x and z-axis rotates as well. The header file Movies.h contains enums for all of the possible tween types. There are quite a few of them and if you really get into it, you can make your 3D QuickTime movies do some really cool stuff!

ADDING TWEEN ENTRIES

There are still three functions to write which are needed by the CreateMyMovie3DTrack function above. These functions are responsible for actually adding tween data to samples.

```
/************** ADD MY TWEEN ENTRY TO SAMPLE ****************/
//
// Adds a tween to the sample.
//

OSErr AddMyTweenEntryToSample(QTAtomContainer tweenSample, long tweenID,
                              long tweenType, void *tweenData,
                              long tweenDataSize)
{
OSErr err;
QTAtom tweenAtom;

      /* CREATE ENTRY FOR THIS TWEEN IN THE SAMPLE */

  err = QTInsertChild(tweenSample, kParentAtomIsContainer, kTweenEntry,
                      tweenID, 0, 0, nil, &tweenAtom);
  if (err)
    return(err);


      /* DEFINE THE TYPE OF IT */

  err = QTInsertChild(tweenSample, tweenAtom, kTweenType, 1, 0,
                      sizeof(tweenType), &tweenType, nil);
  if (err)
    return(err);


      /* DEFINE THE DATA FOR IT */
  err = QTInsertChild(tweenSample, tweenAtom, kTweenData, 1, 0,
                      tweenDataSize, tweenData, nil);

  return(err);
}
```

```
/********* SET TWEEN ENTRY INITIAL CONDITIONS ***********/
//
// Sets the 1st tween value in the sample.
//

OSErr SetMyTweenEntryInitialConditions(QTAtomContainer tweenSample,
                                    long tweenID, void *initialData,
                                    long initialDataSize)
{
QTAtom tweenAtom;

            /* LOOK-UP THE TWEEN ENTRY */

  tweenAtom = QTFindChildByID(tweenSample, kParentAtomIsContainer,
                              kTweenEntry, tweenID, nil);
  if (!tweenAtom)
    return(paramErr);


            /* ADD THE INITIAL DATA OFFSET */

  return(QTInsertChild(tweenSample, tweenAtom, 'icnd', 1, 0,
                      initialDataSize, initialData, nil));
}
```

```
/*********** ADD TWEEN ENTRY TO INPUT MAP ENTRY ************/

OSErr AddMyTweenEntryToInputMapEntry(QTAtomContainer inputMap,
                                    long referenceIndex,
                                    long tweenID, long tweenType,
                                    char *name)
{
OSErr err;
QTAtom inputAtom;

          /* ADD AN INPUT ENTRY TO THE INPUT MAP */

  err = QTInsertChild(inputMap, kParentAtomIsContainer,
                      kTrackModifierInput, referenceIndex, 0, 0,
                      nil, &inputAtom);
  if (err)
    return(err);


          /* SET THE TYPE OF THE MODIFIER INPUT */
```

```
         //
         // note: for 3D, this is almost always
         // kTrackModifierType3d4x4Matrix
         //

  err = QTInsertChild(inputMap, inputAtom, kTrackModifierType, 1, 0,
                      sizeof(tweenType), &tweenType, nil);
  if (err)
    return(err);

         /* SET THE SUB INPUT ID (ID OF THE TWEEN ENTRY) */

  err = QTInsertChild(inputMap, inputAtom, kInputMapSubInputID, 1, 0,
                      sizeof(tweenID), &tweenID, nil);
  if (err)
    return(err);


           /* DEFINE THE NAME */

  if (name)
  {
    long nameLen = 1;
    Ptr p = name;

    while (*p++)
      nameLen++;

    err = QTInsertChild(inputMap, inputAtom, kTrackModifierInputName,
                        1, 0,  nameLen, name, nil);
    if (err)
      return(err);
  }
  return(noErr);
}
```

The complexity of this code may seem a bit overwhelming at first, but once you start working with it you will begin to understand how it works. Soon you'll be able to modify it to do all sorts of complex animations by adding multiple tweens in the tween track. You can make a model fly in, turn around and fly away. Or you can make a model spin like mad while rapidly scaling in and out.

COMPRESSING THE 3D DATA

In the function `CreateMyMovie3DTrack` we called another function named CompressMyHandle which took our 3DMF binary data and compressed it. This feature is only available in QuickTime 3.0 or newer, therefore, do not compress 3D data for QuickTime movies which need to playback correctly with older versions of QuickTime.

QuickTime 3.0 has several compressors and decompressors built into it, including data compressors for compressing any arbitrary block of data. Technically we're using the Component Manager to do this, but the functionality is by way of QuickTime 3.0.

Here is the code which does the compression:

```
/*************** COMPRESS MY HANDLE ****************/
//
// INPUT: srcHandle = handle to source data to compress
// OUTPUT: compressedSrcData = handle to compressed data
//

OSErr CompressMyHandle(Handle srcHandle, Handle *compressedSrcData)
{
OSErr            err = noErr;
ComponentInstance  dataCompressor = nil;
unsigned long    srcSize = GetHandleSize(srcHandle);
unsigned long    compressSize;
unsigned long    whoCares;


       /* INIT HANDLE IN CASE WE CAN'T DO COMPRESSION */

  *compressedSrcData = nil;


          /* OPEN THE COMPRESSOR COMPONENT */

  err = OpenADefaultComponent(DataCompressorComponentType,
                          kDataCompressorType, &dataCompressor);
  if (err)
    goto bail;


       /*************************************************/
       /* GET SIZE OF COMPRESSION BUFFER & ALLOCATE ONE */
       /*************************************************/

  err = DataCodecGetCompressBufferSize(dataCompressor, srcSize,
                                        &compressSize);
  if (err)
    goto bail;

  *compressedSrcData = NewHandle(compressSize + sizeof(long));
  if (err = MemError())
    goto bail;

  HLockHi(srcHandle);
  HLockHi(*compressedSrcData);
```

```
            /**********************/
            /* DO THE COMPRESSION */
            /**********************/

  err = DataCodecCompress(dataCompressor, *srcHandle, srcSize,
                        **compressedSrcData + sizeof(long),
                        compressSize - sizeof(long), &compressSize,
                        &whoCares);
  if (err)
    goto bail;


          /* PUT SIZE INTO 1ST LONG OF DATA */

  ***(long ***)compressedSrcData = EndianS32_NtoB(srcSize);


            /***********/
            /* CLEANUP */
            /***********/

  HUnlock(*compressedSrcData);
  HUnlock(srcHandle);

        /* RESIZE HANDLE TO FIT EXACTLY RIGHT */

  SetHandleSize(*compressedSrcData, compressSize + sizeof(long));

bail:
  if (dataCompressor)
    CloseComponent(dataCompressor);

  return err;
}
```

The compressor type values are found in `QuickTimeComponents.h` and you should generally use `zlibDataCompressorSubType` with 3D data.

SUMMARY

Creating a QuickTime movie with a 3D track is a fairly complex thing to do (relative to doing other things with QuickDraw 3D), but once you get used to it, you'll find that you can create some really cool QuickTime movies.

It's safe to say that QuickDraw 3D with QuickTime is the easiest way to get 3D content onto the web.