Real Time 3D Animation Using Progressive Transmission

by

Jonathan Greenberg

A Thesis

presented to the University of Manitoba

in partial fulfillment of

requirements of the degree of

Master of Science

In

The Department of Electrical and Computer Engineering

University of Manitoba

Winnipeg, Manitoba, Canada

Thesis Advisor: W. Kinsner, Ph.D., P.Eng.

April 2000

(viii + 110 + A-11 + B-2 + C-2 + D-278) = 411 pp.

National Library of Canada
Acquisitions and Bibliographic Services
395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale du Canada
Acquisitions et services bibliographiques
395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

0-612-51718-7

Canada

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

*****

COPYRIGHT PERMISSION PAGE

Real Time 3D Animation Using Progressive Transmission

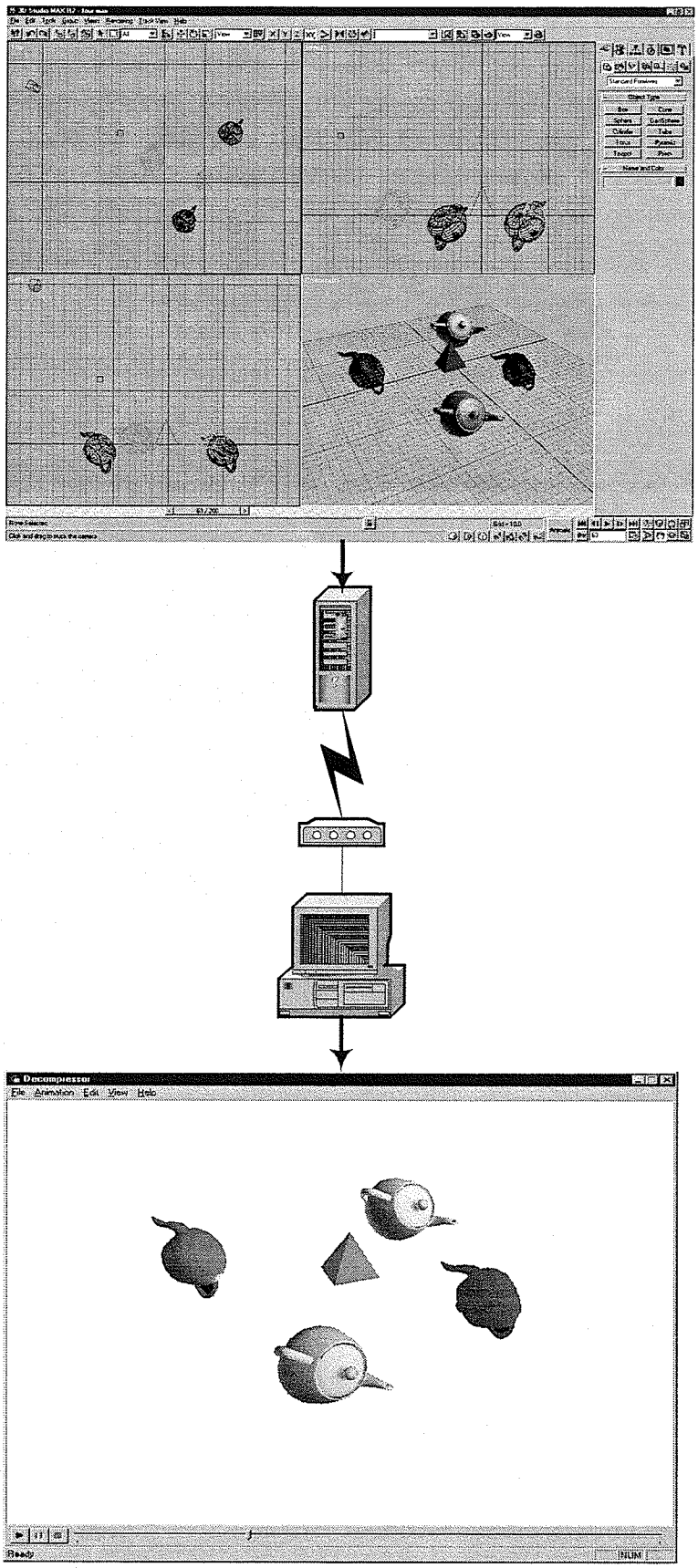
BY

Jonathan Greenberg

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of Manitoba in partial fulfillment of the requirements of the degree

of

Master of Science


Decompressor
File Animation Edit View Help
Ready
NUM

Abstract

Three-dimensional animated sequences are commonly used in demonstration, training and teaching applications to illustrate concepts to viewers. When accessed over the Internet, these sequences are generally compressed using lossy video compression techniques such as MPEG, Cinepak, H.261, or even more recent formats like Vivo, RealVideo and VDOnet. These current techniques are undesirable for the compression of three-dimensional (3D) animated sequences over narrow bandwidth channels. In order to keep bandwidth to a minimum, image quality, frame rate, and frame size are usually sacrificed, producing low quality compressed output that contains artifacts specific to the various compression techniques.

These animated sequences are generally rendered from source 3D scene descriptions, consisting of objects composed of polygons and/or higher order surfaces. These scene descriptions are typically very small relative to the rendered output they produce, and not resolution dependent. Thus, perhaps the current approach of compressing the final rendered animated sequence is not necessarily the best approach. Instead, it is proposed that a new lossless technique, consisting of sending a compressed and stream-optimized version of the 3D scene description rather than the rendered animation, be used for streaming 3D animation over the Internet. Our new approach enables the progressive transmission of the non-interactive scene description such that irrelevant information is entirely eliminated and the remaining information is delivered to the viewer in the order in which is required. Compression and decompression engines have been developed to demonstrate these principles. For some tested cases, the system is able to produce output less than half the size of the equivalent compressed VRML version, and more than 230 times smaller than the equivalent MPEG-1 animation of the same scene.

Acknowledgments

I would like to thank Dr. Witold Kinsner for his aid in this thesis project and formulating the project with Mecca Media Group through TR Labs in 1998. His insight into the various issues surrounding this work has proven invaluable, and I appreciate his additional help in acting as a liaison with both TR Labs and Mecca Media Group.

I must give thanks to both TR Labs and Mecca Media Group for their support during the development of this work. Mecca Media Group provided me with a number of sample scenes that proved helpful in the testing of this work.

I would also like to thank Patrick Oliver for his aid in reverse engineering the Bézier spline formulation used in 3D Studio Max. This aid allowed me to bypass some of the hurdles involved in attempting to work around some of the proprietary subsystems that exist in 3D Studio Max which Discreet was unwilling to explain.

Lastly I would like to thank my parents, without whose aid and support I would not have been able to complete this work. Their encouragement during this project is greatly appreciated.

Table of Contents

TABLE OF FIGURES

List of Abbreviations

3D	Three Dimensional
3DS	Autodesk 3D Studio
CFF	Critical Flicker Frequency
CSG	Constructive Solid Geometry
D3D	Direct3D
HTTP	HyperText Transmission Protocol
MPEG	Motion Picture Experts Group
MIP	Multum In Parvo
NTSC	National Television Systems Committee
NURBS	Non-Uniform Rational B-Splines
OGL	Open Graphics Language
PAL	Phase Alternating Lines
PSNR	Peak Signal to Noise Ratio
UDP	User Datagram Protocol
VRML	Virtual Reality Markup Language

CHAPTER I
INTRODUCTION

1.1 Motivation

The primary motivation for this project concept originated in observations of recent advancements of computer graphics in computer games. Games such as the Quake series, by id Software, and the Descent series by Parallax Software and Outrage Software have led the current wave of consumer directed computer and video games. These games do an excellent job of demonstrating the potential of real time rendering engines to approximate offline-rendering schemes like raytracing. The advent of consumer directed and inexpensive polygon acceleration hardware in the PC market has made these types of games possible as the transformation and rasterization load has been lifted from the host processor allowing more complex visual environments to be constructed in real time.

In 1998, Mecca Media Group requested that Telecommunications Research (TR) Labs assist in the compression of animation sequences used in their training material. In order to keep file sizes small to reduce transmission time, Mecca Media would hand tune the rendered animation sequences using image processing applications. This time consuming process would reduce color palettes and flatten background colors, allowing the sequences to be more easily compressed while maintaining extremely high image quality.

Shortly after the request for aid, in a conference call, Prof. Kinsner suggested the general approach used in this thesis to Mecca Media Group and TR Labs. We developed a proposal for the project with Mecca. It was noted that, at the time, in order to transmit synthesized video sequences created by animation software like Kinetix's 3D Studio Max, one had to employ

standard video compression techniques, such as MPEG or Cinepak. Nonetheless, if one were to know in advance that the content to be compressed would always be these rendered animation sequences, why not take advantage of this knowledge and build a new technique around it?

1.2 Problem Definition

This thesis attempts to address the problem of delivering 3D animation content over narrow bandwidth channels. Two primary techniques are currently commonly employed to transmit synthetic 3D content: rendering out two-dimensional animation frames which are compressed using video codecs, or instead compiling the scene description using a description language to be interpreted by the client (for example, VRML). Neither of these two techniques performs well over narrow channels, such as 28.8 Kbps modems, due to reasons that this thesis addresses.

This thesis attempts to improve on these current approaches by allowing for the progressive transmission of a scene description, enabling the viewing of sequences as they are actively streaming over a channel and by greatly reducing the amount of data required to specify this information. It is hypothesized that through the innovations presented in this thesis, the problem of transmission of 3D animation sequences over narrow band channels can be solved.

1.3 Organization of the Thesis

This thesis consists of several major sections, each with its own specific purpose to aid in explaining how the environment was constructed, the results it produced, and the conclusions drawn therefrom. The thesis structure is as follows:

Chapter 1 introduces this thesis by providing the motivation behind the thesis. In this section the problem we attempt to solve is defined. As well it presents the structure for the body of the thesis.

Chapter 2 compiles the background theory used to develop the concepts presented and described in this thesis. As a variety of different topics are sewn together in this thesis, a full understanding of the background is required to understand choices made in the design phase.

Chapter 3 describes the design process of this thesis. Building upon the information presented in the previous chapter, it lists the choices made and the optimizations exploitable through these decisions. By the close of the chapter, the full design of the two applications has been specified.

This specification is required for Chapter 4, as it describes how the design was implemented. This section describes the specifics involved in translating the design into working compression and decompression applications, and outlines the major pitfalls encountered en route.

The implemented applications are then used to apply the suite of experiments, a series of test case animated sequences, in Chapter 5. The sequences are described in detail, along with simple pictures of the source animation, and include statistics as to their complexity.

Results from these experiments are presented and analyzed in Chapter 6. The specifics are described of how experiments were conducted and comparative results were collected. Results from experiments are compared against several comparable existing systems.

From the results in Chapter 6, conclusions are offered in Chapter 7. Following these inferences, a series of recommendations and suggestions towards how this work might be expanded in the future are given.

CHAPTER II
BACKGROUND

A variety of different techniques and technologies were employed in the development of the concepts utilized in this thesis. This chapter describes the necessary background to facilitate the ideas presented in this thesis.

2.1 Three Dimensional Rendering

Animated sequences of real world objects are often constructed for demonstration purposes or product visualization. In order to model these objects effectively, likenesses of them are constructed using computer-based tools, in which the objects are reconstructed in artificial three-space, using building blocks known as primitives. These primitives are geometric constructs, usually consisting of triangles, but can also include geometric solids, and parametric surfaces. These scene descriptions are resolution independent, since the scene graph locates objects in a virtual world rather than describing the interactivity or values of individual screen pixels. The description contents are also generally view independent as well for this same reason.

In order to render the objects in the scene, a virtual camera is placed in the scene along with light sources, and a resultant frame is constructed using rendering techniques such as raytracing or radiosity. While these techniques can produce photorealistic output (outputted image quality indistinguishable from that produced through photography), they require a great deal of time to be computed, and are not practical for use as real time rendering techniques. Raytracing finds individual colour intensities for each screen pixel and thus, while producing more accurate output, raytracing is very computationally expensive. Radiosity calculates view

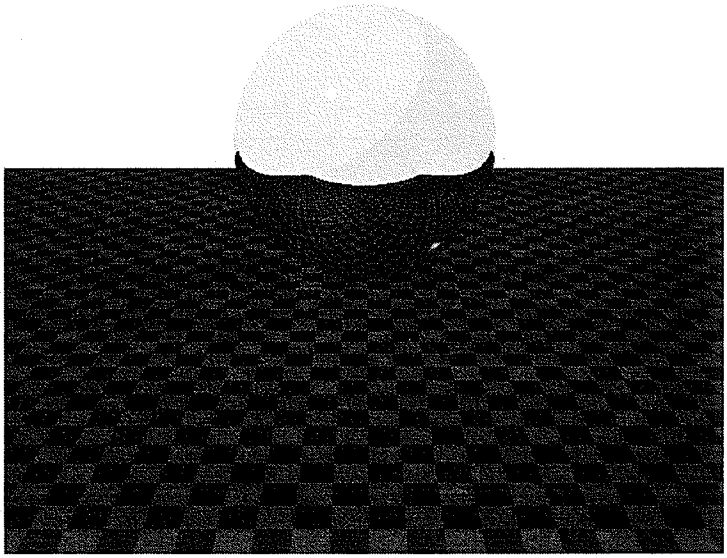


Fig. 2.1 A classic raytracing example - a reflective sphere floating over an checkered plane.

independent lighting of the objects in the scene that, while extremely effective at modeling the interaction of indirect lighting with surfaces, is even more computationally expensive than raytracing. Many of the current techniques that produce the most convincingly realistic images consist of hybridized models combining aspects of both raytracing and radiosity.

2.1.1 Raytracing

The primary concept of raytracing is attempting to model the delivery of light rays to the camera (or virtual eye) by projecting rays from the eye and determining how these rays interface with objects in the scene. This is accomplished by tracing rays from the eye point such that each ray passes through a point on the view plane corresponding to a pixel in the final rendered frame. These vectors are then tracked to determine which is the first object they penetrate. The color of the corresponding point in the view plane is then determined by the color of light reflected off the object at the point of intersection.

To determine the light color at the intersection point in raytracing, one first finds the interaction of the object itself with various surrounding light sources, often through the Phong lighting model [FDFH87, p. 778]. If a ray cast between the point of intersection and the light source penetrates another object, then the original intersection point is said to be in shadow. If not, depending on the reflectivity of the object, reflection rays are spawned and cast out along the reflection vector created between the surface normal vector, and the camera vector. If the object is semi-transparent, the line-of-sight vector is refracted through the object and a new secondary ray is spawned as well. Both of these new rays are then recursed to some set limit upon which they terminate. If the originally intersected object was neither highly reflective nor refractive, then the point in question will take on the color of its source object at that location.

Due to the recursive nature of the raytracing algorithm, raytracing is generally an extremely slow operation. A great deal of literature has been dedicated to optimizing and speeding up the raytracing process, however because of the exponential nature of raytracing, this has yet to be achieved on anything less than massive multiprocessing systems using special case scenes [Whit92, p. 10]. Since raytracing is such a slow operation, rendering is described as offline rendering, as for animated scenes groups of frames are queued together and rendered to some storage device in batches. These scenes cannot be viewed interactively and thus all frames must be fully rendered before the scene can be displayed.

2.1.2 Phong Lighting Model

In order to calculate the light intensities for vertices in a variety of rendering systems including but not limited to raytracing, the Phong lighting model is often employed. Phong's lighting model determines the light intensity on objects due to direct lighting. This lighting model,

developed by Phong Nui-Tuong [FDFH87, p. 729] is

$$I_\lambda = I_{a\lambda} k_a O_{d\lambda} + \sum_1^m f_{att} I_{mp\lambda} [k_d O_{d\lambda} (\overline{N} \circ \overline{L_m}) + k_s O_{s\lambda} (\overline{R_m} \circ \overline{V})^n] \quad (2.1)$$

where I_λ is the intensity of the light at the given wavelength λ at some point on an object, $I_{a\lambda}$ is the ambient light at wavelength λ, and k_a is the ambient-reflection coefficient of the object. The ambient portion of the equation attempts to provide a simplified model for indirect light transport between objects in a scene. The diffuse portion of the equation attempts to model basic reflectivity between objects in the scene. The $O_{d\lambda}$ value is the diffuse color of the object being illuminated, f_{att} is the attenuation rate, $I_{mp\lambda}$ is the diffuse light intensity at the given frequency for light number m, k_d is the diffuse-reflection coefficient of the object's material, $\overline{N}$ is the normal vector to the point, and $\overline{L_m}$ is the light vector to the current light source, m.

The specular portion of the equation provides a highly simplified model for determining the specular highlights (bright reflective highlights that appear on very shiny or polished surfaces) of an object. The k_s value is the specular-reflection coefficient of the material, $O_{s\lambda}$ is the specular color of the object being illuminated, $\overline{R_m}$ is the light vector reflected at the point for the light m, $\overline{V}$ is the vector between the point and the camera location and n is the material's specular-reflection exponent.

2.1.3 Radiosity

Radiosity is a radically different approach in that it is based around the energy absorbing properties of various materials. In radiosity, objects are divided into rectangular patches over which light has a uniform intensity [CoWa93]. Unlike raytracing in which only light sources emit light, in radiosity, all patches are considered light emitting. The radiosity of any patch is equal to

the sum of the emitted energy and reflected energy for that patch, where the reflected energy is the product of a reflection coefficient for the object's material and the energy incident on that specific patch from all other patches.

Radiosity tends to produce extremely realistic output, images that can often be mistaken for photographs of the scenes that have been modeled. Yet for its superior image quality, radiosity has a computational complexity a magnitude higher than raytracing. While radiosity is even less practical than raytracing for direct real time animation, it can be used to improve the appearance of real time 3D animation engines by precalculating the radiosity of static elements of the scene. This technique is currently employed quite frequently in video games and architectural walk-throughs.

2.1.4 Real Time Rendering

Instead of relying on raytracing or radiosity, which are very realistic, yet expensive techniques, most current real time rendering engines make use of Gouraud shading [Hill90] in combination with some lighting model. Gouraud shading makes use of the light intensities of polygon vertices to shade entire faces. Finding the illumination for every pixel of every primitive might be ideal but it is still too expensive, so instead the illumination values for each of the primitives' vertices are found. Light intensities of the primitives' internal points are then linearly interpolated. While this operates quite effectively for small polygon, large polygons shaded using Gouraud shading often exhibit inaccurate shading, as specular highlights will be lost on these larger surfaces, and the linear shading will not always reproduce non-linear effects (such as attenuation) that should be present.

Light intensities are generally determined using the Phong lighting model previously

described, or are precalculated using offline techniques such as per-polygon radiosity. While Phong lighting is often used because it allows for dynamic lighting of scenes, it does not account for light sources being obstructed. This can often result in visual errors such as objects that should be in shadow instead of appearing fully lit.

Unlike raytracing which operates by traversing scan line by scan line, pixel by pixel, through the entire frame to determine the entire image, real time rendering operates by rasterizing each primitive into the frame buffer in the order in which it is presented. This often requires that polygons be sorted based on distance from the viewpoint as otherwise, closer objects would end up obscured by those which should appear further away. This technique of first sorting polygons based on view depth is commonly called the Painter's Algorithm [Hill90] as it mimics how one oil painting on canvas obscures previous coats of paint when one applies a new color overtop.

Another method to determine polygon visibility is to track the screen depth of rendered polygons using a Z-buffer. When beginning a scene, the Z-buffer (a depth buffer with one entry for every screen pixel) is initialized to the furthest possible distance from the viewing plane. Whenever a new polygon is presented, each pixel of the polygon in question is tested against the Z-buffer. If the new pixel's depth value is closer to the eye point than the previous Z-buffer entry, the new pixel overwrites the existing one in the color buffer and the previous Z-buffer value is overwritten with the new one.

Though the appearance of lighting through Gouraud shading provides a reasonable simplification of raytracing, it should be noted that it is not nearly as accurate. Raytracing attempts to trace rays projecting from the eye into object space and by doing so naturally takes into account shadows, multiple levels of reflection, and refraction through surfaces. Real time techniques do not naturally take any of these three factors into account. Shadows, reflections, and refraction, to be accurately recreated generally require either preprocessing of scene data, or

special case techniques (for example, limiting shadows to being projected onto floors, or special case modeling information to be used in generation of silhouette boundaries. Special effects required to produce these missing features can greatly slow down real time rendering and are extremely limited in their ability to accurately recreate the raytraced images they attempt to replace.

One final factor important to consider is that frame rates, or the frequency at which frames are constructed and then rasterized, are generally not static in real time rendering systems. The time required to draw an individual frame is determined by two primary factors: the time to update and evaluate the geometry database and the time for all geometry to be processed through the rendering pipeline. Fluctuations in the time required by both of these processes must be taken into consideration.

Whereas in offline rendering systems, such as raytracing and radiosity, where rendering time has no direct effect on the rate at which frames are displayed to a viewer, rendering time directly determines the rate at which frames will be rendered in real time systems. Thus, scene complexity, scene overdraw, and geometry database processing must be kept to a reasonable level to allow the high frame rates preferred by the human visual system. Various studies have shown [Glas95, p. 18] that the *critical flicker frequency* (CFF) for most humans under ideal conditions is around 60 Hz. It should be noted that most common video formats run considerably under this critical threshold, with MPEG1 video running at 30 Hz when using NTSC source material, and 25 Hz when using PAL source material. For synthesized sources, frame rates typically range from 6 Hz targeted for low bandwidth Internet channels to the 30 Hz of NTSC video (NTSC video runs at 60 fields per second, where each frame requires 2 fields interlaced together, for a total of 30 frames per second). It should be noted that frame rates can be allowed to deviate from the CFF to some degree if they are not interactive. The greater the required user response time, the more

noticeable deviation from the CFF will become, and in some cases, such as military flight simulators, an even higher CFF may be required by some users.

2.1.5 The 3D Rendering Pipeline

While at first glance rendering a single triangle from 3D space into the 2D-screen space might appear to be a relatively trivial task it is actually quite involved. Before a primitive can be rendered, it must first travel through the 3D pipeline.

The primitive is first transformed from its local coordinate system into world coordinates by translating it by its world position, and rotating it to its world orientation. Once the primitive's new location has been determined, one checks whether or not it is inside the view volume. The view volume (also called the view frustum) is an imaginary flat pyramid defined in space by the camera position, the front and back clipping planes, and the field of view. If an object is entirely outside the view volume, it is trivially rejected from the pipeline.

If a primitive has not been rejected at this point, it is checked to confirm that its normal vector is not pointing away from the camera. If it is, then the primitive cannot be seen, and is rejected from the pipeline. This is referred to as back face culling of the primitives. If the primitive has not been rejected, it is then lit and rasterized.

The pipeline can be further optimized by trivially rejecting all primitives belonging to objects that are outside the view frustum. This is performed by calculating a bounding volume for the object and checking if the bounding volume intersects the view volume. The 3D rendering pipeline order is not strict, and can be modified depending on the optimizations chosen and advanced knowledge of the scene contents.

2.1.6 Representation of 3D Objects

A great deal of work has been performed over the lifetime of graphical research into determining optimal ways of describing artificial objects in artificial three-space. The oldest and most straight forward method is the description of 3D objects using mathematical formulas such as quadrics [FDFH87, p. 473]. This technique has long been used due to its utility in mathematics and compactness of the expressions. For example, to define a sphere, all that is required is the sphere's position and its radius. The combination of quadrics with other basic geometrical constructs such as pyramids and cubes led to the technique of *constructive solid geometry* (CSG). Constructive solid geometry takes combinations of these simple geometric solids, usually referred to as primitives, since they are the most basic building blocks of such systems, and combines them using set operations, as illustrated in Fig. 2.2.

In recent years CSG has fallen out of vogue and instead has been superseded by modeling surfaces with various types of parametric curves, including Bézier curves, B-splines, and *non-uniform rational B-splines* (NURBS). These curves have proven to be extremely powerful and easy to use in the modeling of curved objects. They also prove more generic than quadrics, as

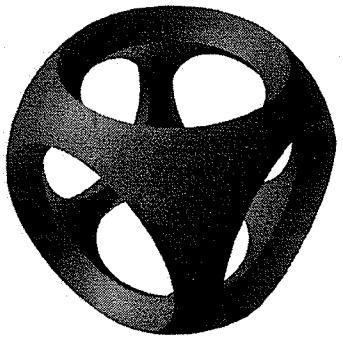


Fig. 2.2 An example of CSG-a sphere that has had six smaller spheres subtracted from it.

basic quadrics can be constructed as special cases of more general rational curves such as NURBS. In addition artists generally find them quite powerful, allowing them to model extremely complex objects more easily.

Though general modeling through quadrics is not as common anymore, one newer technique has proven very effective in modeling through the use of formulaic descriptions of objects. Implicit surface modeling, also commonly referred to as metaball modeling (see Fig. 2.3), allows the construction of complex surfaces through the placement of point energy sources with exponential decays. Several sources combined together form an energy field, and the surface described by this field is plotted at points along the field where the energy level crosses below some given threshold. Metaballs have proven quite useful in the modeling of organics as well as the modeling of flowing and pooling behaviour of liquids and other systems that can be approximated by field effects.

However, while all of these techniques allow for various advantages when modeling various objects and surfaces, the most universal and widely supported geometric primitive is the triangle. As the simplest possible three dimensional surface element, triangles require only three vertices to be defined in space. Being the simplest element, all the previously described

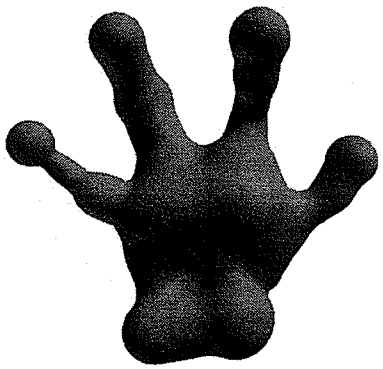


Fig. 2.3 An example of Metaball modeling-an alien hand constructed from 23 metaballs.

description techniques can be tessellated into meshes of triangles. This is especially important as evaluating the other more complex descriptions can be computationally expensive, often making them impractical for interactive rendering speeds. Thus, while tessellation of more complex surfaces may reduce rendering time, it does generally increase the required storage space for the descriptions of these surface types.

2.1.7 Texture Mapping

As geometrical objects become more and more complex, filling in detail explicitly using polygons can often be wasteful. The chief alternative approach to adding great detail while not increasing the complexity of a mesh is by painting detail over top of the geometry, a technique called texture mapping [FDFH87, p. 741]. While there is a variety of complex techniques possible, the most fundamental concept of texture mapping involves taking a bitmap and painting this bitmap directly onto the triangular mesh, greatly reducing the number of polygons required to define complex objects. The image painted onto the geometry is referred to as a texture map, and can vary in resolution. Vertices in a mesh are attributed coordinates in a U-V space, which define points mapped into the texture map. These coordinates are called texture coordinates, and by convention, the boundaries of a texture map are generally mapped between (0,0) for the lower left corner and (1,1) for the upper right corner.

As textures are being rendered onto 3D geometry, the original images end up appearing as skewed versions of the originals. These skewing operations, when combined with perspective correction can often produce distortions across the rendered texture due to linear interpolation. This distortion is generally more noticeable than other operations combining linear interpolation with perspective correction such as polygon shading. There are many of approaches that have

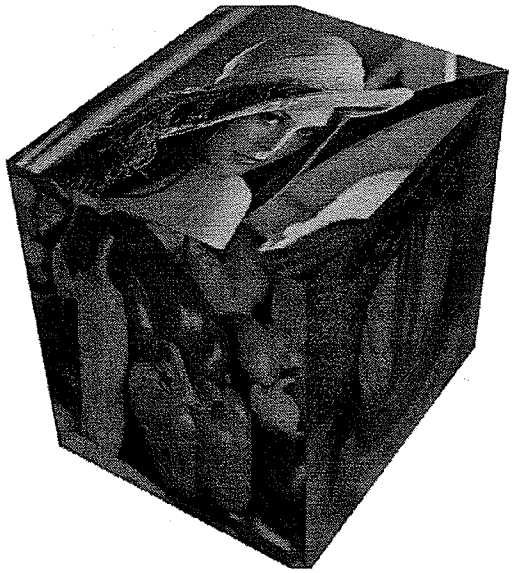


Fig. 2.4 Bitmaps are texture mapped onto the faces of a cube.

been taken to removing the distortion, but the most common approach, and the primary one supported by current real time 3D rendering engines is to use MIP (*multum in parvo* - many things in a small space) mapping. Instead of simply rendering the original texture map, the texture map is used to construct a series of MIP maps, lower resolution versions of the original texture, constructed by filtering the original image. By convention, these lower resolution maps usually have dimensions corresponding to half the previous map constructed, with the first map having a resolution equal to the original source texture. Thus, an original texture of 256 x 256 is reduced to the series of 128 x 128, 64 x 64, ... , 2 x 2, 1 x 1 images where each reduced image is a filtered version of the previous image.

When rendering, selection algorithms are applied on a per-pixel basis to determine which MIP map should be used. Selection algorithms are not described here, as they are generally abstracted from the user of a rendering engine. Once selected, the MIP map is then used for rendering the texture. Yet, this can still produce some visual anomalies due to regions where the selection algorithms break down and fail to select the ideal candidate MIP map. To produce

better results, a technique known as trilinear filtering can be used. Trilinear filtering attempts correct for these problems by allowing the selection algorithm to return an inexact solution as to which MIP map corresponds to the given pixel. This solution, which lies between two known MIP map levels, is used to interpolate between these two maps. The MIP maps are then also each bilinearly filtered, as the U-V lookup corresponding to the pixel often does not relate to an exact texel in the MIP maps, and these MIP maps are also of different resolutions.

2.2 Progressive Transmission

The concept of progressive transmission is a straightforward one. Rather than waiting for an entire message to be received in its entirety before its contents are evaluated, progressive transmission demands that the message be evaluated in pieces as it is being received over a channel. The use of progressive transmission has several natural advantages over the simpler technique of simply waiting till a message is received in full. Whereas the simpler method requires one to wait until a message has fully arrived before it can be processed, progressive transmission allows for better resource management on the client side by allowing processing to be multiplexed with data retrieval. This assumption is generally quite plausible as in general the time required for transmission of data is considerably greater than the time required to evaluate it.

Progressive transmission techniques also allow for client side control over the channel through the ability to stop the transmission once enough data has arrived. Further optimizations are possible, such as shaping the transmitted content to match dynamic channel characteristics, or progressive detail refinement [DaKi99].

Nonetheless, while progressive transmission of a data source offers several immediate benefits, taking full advantage of them often introduces some degree of additional overhead, as

new portions of data sent to a client must be marked based on their temporal and/or spatial importance to the client. This naturally leads us to a discussion of wavelets, prime candidates for progressive transmission techniques due to their locality in space and time.

2.3 Wavelet Transforms

Since the introduction of the concepts of using cosine and sine waves to construct more complicated waveforms was introduced by Joseph Fourier [OpWY83, ch. 4], the use of superposition of simple waveforms to approximate more complex ones has been a mainstay of signal processing. Wavelet theory attempts to replace more traditional Fourier analysis by using simple aperiodic signals as the basis functions rather than the more traditional periodic sine wave. The aperiodicity of the wavelet basis makes it a more practical construction tool for use with real world signals which tend also to be aperiodic. On the other hand, a sine wave basis has long proven impractical in describing rapid discontinuities, a factor easily demonstrated by the appearance of Gibbs phenomena in square wave signals [OpWY83, p 186]. As an individual component in the frequency domain will produce an infinite signal in the time domain, many frequency components are required to reconstruct local discontinuities.

Wavelets are localized in both time and frequency, and thus are said to be compactly supported [Glas95, 244]. By scaling and stretching the wavelet function, time domain signals can be reconstructed. The compactness of the wavelet basis allows it to describe discontinuities in the time domain signal using fewer coefficients than would be required by a sinusoidal basis. There is a wide variety of wavelet bases that have been constructed to date, but for this work we limit our study to the Haar basis.

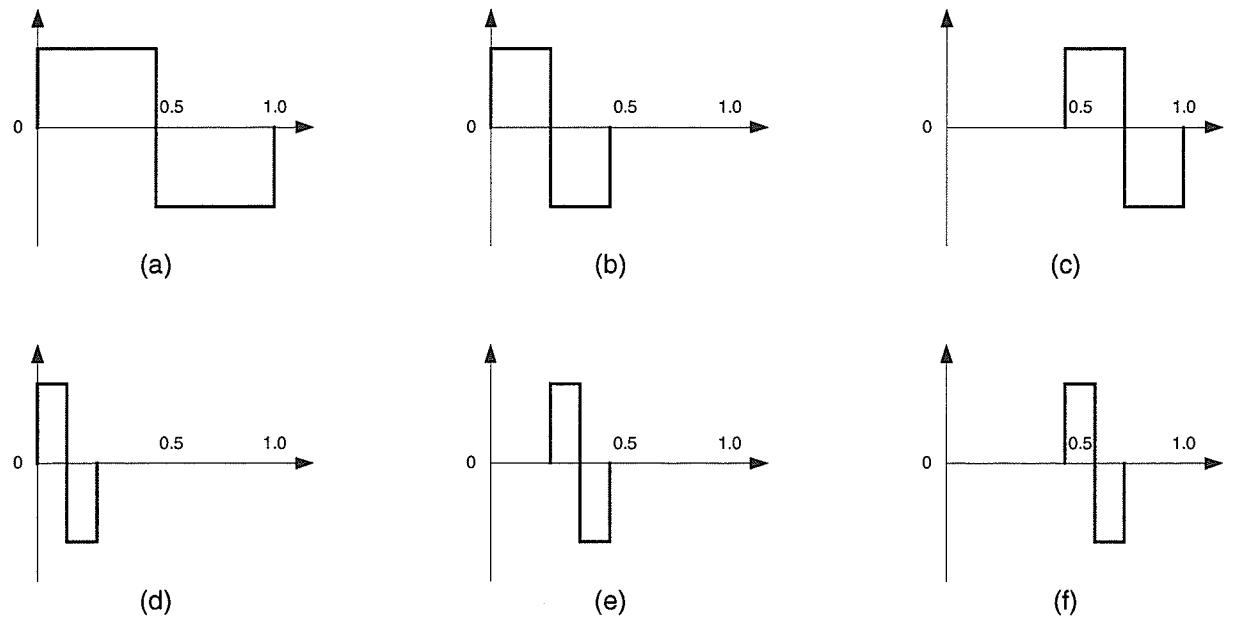


Fig. 2.5 The first six functions of the Haar wavelet. (a) $\psi_{0,0}$ (b) $\psi_{1,0}$ (c) $\psi_{1,1}$ (d) $\psi_{2,0}$ (e) $\psi_{2,1}$ (f) $\psi_{2,2}$

2.3.1 Haar Basis

The Haar wavelet as shown by Fig. 2.5 is defined as [Sayo96, p. 313]

$$\psi_{0,0}(x)=\begin{cases} 1, 0\le x<\frac{1}{2} \\ -1, \frac{1}{2}\le x<1 \end{cases} \tag{2.2}$$

where the 'mother function' used to generate all sub-band functions for the Haar basis is given by

$$\begin{aligned} \psi_{j,k}(x) &= \psi_{0,0}(x)\left(2^j x-k\right) \\ &= \begin{cases} 1, 2^{-j}k\le x<2^{-j}\left(k+\frac{1}{2}\right) \\ -1, 2^{-j}\left(k+\frac{1}{2}\right)\le x<2^{-j}(k+1) \end{cases} \end{aligned} \tag{2.3}$$

where j is the level of the wavelet and k is the location parameter. Higher values of j indicate narrower support bases, while different values of k determine the location of this order of wavelet in the signal. In general for all wavelets, and specifically for the Haar basis, for any given level j,

there are 2^k possible valid locations [Glas95, p. 263].

The Haar wavelet can also be thought of as an averaging function. Given some initial data set with a power of 2 size, a new set is constructed of equal length. The lower half of the set is replaced by the mean of each pair of the original set values, while the upper half of the set is replaced by the differences between each pair of the original set values. As the mean lies equidistant between both original points of a pair, all information from the original set can be easily and quickly reconstructed. Due to its extreme simplicity, the Haar wavelet is considered the simplest and quickest of all wavelet transforms. But its extreme simplicity makes it one of the poorest performing wavelets in terms of compact signal description. This is due to its having zero order matching properties, causing it only to be able to concisely match functions that are piecewise constant. For more efficient matchings of higher order curves, more complex wavelet bases such as the Daubechies basis must be used [Glas95, p. 278].

Wavelets, such as the Haar basis and especially the previously mentioned more complex wavelets, are often used in data compression applications due to their quality of having compact support. This factor often causes large amounts of correlation between the coefficients of the transformed signals due to large quantities of zero values. The natural redundancy of these large quantities of zero coefficients enables data compression techniques to reduce the storage requirements for the signal.

2.4 Data Compression

Data compression allows one to reduce the transmission time as well as the amount of storage necessary to specify some given information, with a consequence of increasing processing overhead. This section provides background on some of the basic concepts behind data

compression and how they apply to this thesis work. Compression for this thesis focuses on both the two approaches for data compression: lossless and lossy compression. Lossless compression requires exact reconstruction of the original source signal, while lossy compression allows for some degree of signal degradation during the compression process. Lossless compression deals with the removal of redundant information from the signal, while lossy compression additionally removes what is deemed unnecessary or irrelevant.

2.4.1 Information Theory

The idea of quantizing the amount of information a certain event represents was first presented by Claude Shannon with the concept of self-information [Sayo96, p. 13]. Shannon stated that given the probability $P(A)$ of some event A taking place, it can be stated that the self information of A is provided by

$$i(A)=\log_x \frac{1}{P(A)}=-\log_x P(A) \tag{2.4}$$

It is interesting to note that the self information of an event corresponds to an opposite level of probability of it occurring. For example, high probability events can be seen to have low self-information, while low probability events are deemed to contain a high degree of self information. This can be understood conceptually be considering how humans observe and rate the importance of events. When rare or seldom occurring events take place, they tend to be noted. Consider the simple example of the operation of an automobile. If one experiences a safe journey to a destination, it is rarely considering noteworthy. If one instead experiences an accident during the

trip, the trip is given great importance and is generally recounted to others.

Further, given a set of independent events A_i within some channel C, where S is the sample space from which these events are taken, then it is said that the average self-information of the channel is given by

$$H_{avg} = \sum P(A_i) i(A_i) = -\sum P(A_i) \log_x P(A_i) \tag{2.5}$$

where H_{avg} is said to be the Shannon zero order entropy, or level of disorder of the channel. Zero order entropy assumes independence of the various symbols in terms of position and correlation and only focuses on symbol frequency. Higher order entropies focus on the correlation and dependence of the various symbols on their immediate neighbors.

2.4.2 Dictionary Coding

As natural language text tends to contain a very large quantity of correlation between various words and their component parts, the fundamental concept of dictionary coding is that of textual substitution [Sayo96, p. 98]. When a data string in the stream is encountered it is compared against a dictionary. If the string is found in the dictionary, a code representing the string is transmitted rather than codes for the individual characters themselves. A simple example of this concept would be replacing the four characters '~~b~~the' (where ~~b~~ is a spacing character) with some token every time it occurs in this thesis text. By doing so, we could significantly reduce the number of characters required to specify the entire document. In data streams that are highly correlated, such as plain text, textual substitution of dictionary codes generally produces very good results.

Ideally, as statistics of a data stream tend to change as the stream changes, dictionaries must be allowed to adapt to local statistics. Most adaptive techniques owe their roots to LZ77, a fundamental dictionary technique based on two separate approaches developed by Jacob Ziv and Abraham Limpel in 1977 [Sayo96, p. 100]. To encode an incoming stream of data, a sliding dictionary window is employed. The sliding dictionary is of a fixed size and contains characters that have previously passed through the encoder, in the order they appeared. The portion of string that has yet to be encoded is placed in a look-ahead buffer [Kins00, ch. 16].

When data is encountered in the look-ahead buffer, a search is performed into the dictionary to determine if a portion of the string formed by the look-ahead buffer already appears in the dictionary. Should the string begin at the tail of the dictionary, the string is allowed to overrun into the look-ahead buffer. If the search string is found, an offset into the dictionary where the portion of the string was found as well as the portion's length are computed. These two values are combined with the character following the string portion in the look-ahead buffer to form a code triple. This third element is transmitted to account for the degenerate case in which the first character in the look-ahead buffer (and thus the first character in the search string) cannot be found in the dictionary. Once the triple has been transmitted, the dictionary window slides to cover the information the triple contained, and the process is repeated.

LZ77 has many natural advantages that make it an excellent candidate technique. Though encoding of the stream is highly dependent on the efficiency of the dictionary search, decoding a stream is extremely fast as it merely consists of a long series of table lookups. As well it requires a minimal amount of memory to operate, as the decoder merely requires enough memory to store the dictionary and look-ahead buffers in order to operate. Due to its many advantages, LZ77 has spawned a large number of variations including, but not limited to PKZip, LHarc, and Arj.

2.4.3 LZSS Compression

J. A. Storer and T. G. Szymanski introduced a variation on LZ77, often referred to as either LZSS or the Storer technique. The primary improvement introduced by Storer and Szymanski is the use of a cyclic dictionary window rather than merely sliding it over the data stream [Stor88, p. 64]. Additionally, the triple associated with LZ77 is no longer required due to the introduction of a header bit.

The encoder for LZSS works as follows: The dictionary is initialized to a set of zeros. The stream is loaded into the look-ahead buffer. A search is performed into the dictionary buffer to find the longest possible match string to the string formed by the look-ahead buffer. Unlike LZ77, where the search can extend into the look-ahead buffer itself, in LZSS if it is required to search past the end of the dictionary, the search wraps back to the dictionary's beginning.

If a string is found, we compute its offset and length. These two values are transmitted with a 1-bit header of value 0. Should it not be possible to find the string in the dictionary, the string is set equal to the first character in the look-ahead buffer. This character is transmitted using a 1-bit header of value 1. In either case, the string is removed from the look-ahead buffer and added to the dictionary, wrapping back to the beginning of the dictionary should its end be reached.

This technique has several immediate advantages over LZ77. The first and most obvious is that rather than wasting at least a single character for each of the offset and length portions of a triple in the degenerate case, LZSS merely wastes a single bit. Another advantage is that there is a smaller memory footprint required by the decoder as only the memory for the dictionary is necessary.

There are two methods proposed by Storer and Szymanski to improve the performance of this technique. It should be noted that normally length values for a offset/length pair would range

between 0 and 2^L-1, where L is the number of bits used to encode the length. The upper bound of length values that can be encoded using L bits can be improved by considering that certain length values would never be used by the encoder. For example, trivially, length values of both 0 and 1 can be eliminated by noting that a length of zero is meaningless, and a length of 1 translates into the non-pairing degenerate case for which length is always implied to be equal to 1. Thus the length value range can be shifted to fall between 2 and 2^L+1, allowing for improved performance of the compressor.

The second improvement attempts to improve the encoder's search performance by imposing a deletion heuristic on the dictionary. Should a search return a successful result, *S*, of length greater than 1, the deletion heuristic searches the dictionary for the shortest possible occurrence of a substring of *S*, *R*. Should *R* be found at a location other than *S*, it is deleted from the dictionary, as the string *S* is already in the dictionary making *R* redundant (relative to *S*). This allows the dictionary to contain a larger variety of candidate strings, potentially allowing for greater performance through matching a larger number of dictionary strings.

2.4.4 Image Compression

While LZ77 and LZSS are general purpose lossless compression algorithms, the more one knows about a given information source, the greater the number of specific optimizations that can be applied. Two dimensional images have long been a major focus of data compression, due to the large amount of data the otherwise uncompressed images require. Images are typically stored as two-dimensional arrays referred to as bitmaps, where each location in this array - sometimes referred to as a pixel - requires a static number of bits to describe its color or shading information. The general case colored bitmap requires 8 bits per color component, where the color

components are red, green and blue.

As images designed for human viewing generally contain large amounts of correlation and redundancy, a wide variety of techniques have been employed in the past to reduce image storage requirements. Though currently in use techniques are too numerous to list here, several proven techniques include GIF, JPEG, and Portable Network Graphics (PNG, designed to replace GIF). Image coding techniques, like all other compression techniques, fall under the two categories of lossy techniques - which tend to attempt to remove data otherwise ignored by the human visual system, and lossless techniques which retain all original image information.

Lossy image compression techniques often fall under the category of transform coders, which attempt to transform the original image from image space into some form of transform space, first possibly having gone through some form of color space conversion. Once the image has been transformed into some other space, low frequency coefficients are often truncated, and the remaining high frequency coefficients are retained. These coefficients are then compressed using some lossless technique, and are stored awaiting the application of an inverse process to restore an image visually incomparable to the original. For example, MPEG encoders attempt to compress keyframes (which are treated as images) by dividing the image into a series of 8x8 pixel blocks on which it then performs a discrete cosine transform. For more general case image compression, one might choose instead to use fast Fourier transforms, or one of the various wavelet bases.

Image compression remains an active area of research in which there is ongoing work investigating every aspect of the compression pipeline, from color spaces and transformation techniques to frequency selection and coding schemes. One of the techniques currently in vogue is using wavelets as the transformation technique, as wavelets divide the image into different sub-bands which can then be encoded separately.

2.4.5 Video Compression

Where image compression focuses on the correlation between neighbouring regions within an image, video compression attempts to focus on correlations between successive images. Though video compression will be only lightly touched on, several primary points should be made. As hinted above, video compressors operate by dividing a scene into two parts - keyframes and *tweening* (a corruption of the word inbetween) frames. Keyframes, as discussed later, are reference frames. In the case of video compression, these keyframes are used as references points while the tweening frames describe information about changes that take place between successive keyframes. Thus, an original video sequence running at 30 frames per second might be divided such that there is a reference keyframe every 5 true frames, and tweening frames are used to reconstruct the frames that would lie between these keyframes.

In general, video codecs attempt to reduce the number of required keyframes and improve the efficiency of the tweening. The most common method of doing this is to divide the keyframes into blocks of some set resolution, as described earlier. Tweening information is then used to describe the transformation of these blocks required to reconstruct the equivalent true frames such that the original is indistinguishable from the synthesized image. In general as time progresses and more of these delta frames are applied, error in the reconstructed frames accumulates. Thus keyframes are required periodically to reset the sequence and prevent further error.

It can be seen that, in general, tweening frames would require very little storage as compared to the keyframes which require the specification of an entire image. The size of a compressed video file grows linearly with the number and resolution of these keyframes. Therefore, much effort has been made in video compression research to reduce the storage

required for keyframes (generally by falling back on image compression techniques), as well as to reduce the errors that would be introduced by these delta frames.

2.4.6 Geometry Compression

Whereas video and image compression are often used to reduce the storage and bandwidth requirements for photographed sequences, geometry compression revolves around the reduction of storage space for models to be used in synthesizing images. A number of techniques have been developed for compressing three dimensional geometry. The majority of these are based around several basic methodologies. Currently in vogue are derivatives based upon the progressive mesh work of Hugues Hoppes [Hopp96]. Progressive meshes take some original mesh $\hat{M}$ and reduce it to some low detail mesh M^0 where n progressive levels can then be applied successively to M^0, producing $M^1, M^2, \ldots, M^n$, such that after then n levels $\hat{M} = M^n$.

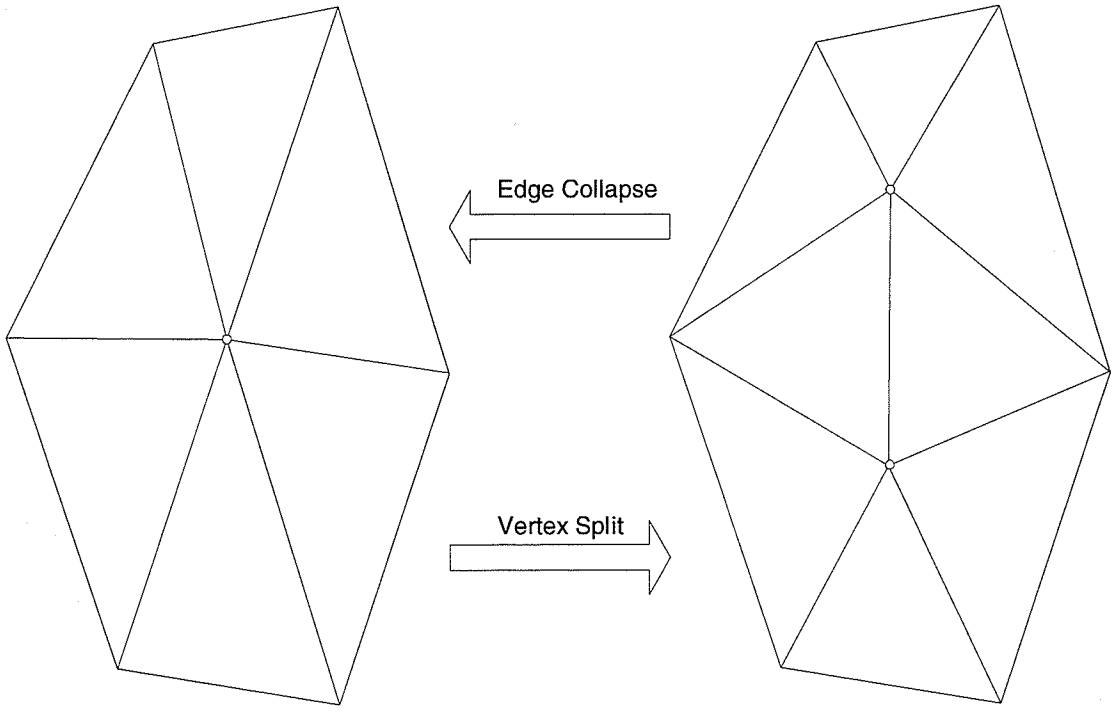


Fig. 2.6 The vertex split and edge collapse mesh operations.

This progressive refinement on the original is performed by constructing a hierarchy of edge collapsing operations which take two existing vertices connected by an edge and replace them by a single new vertex equidistant from the two original points. The edge collapse has a corresponding inverse operation, the vertex split (see Fig. 2.6). The vertex split takes an existing vertex and converts it into an edge, reconnecting the original polygons, and generating new triangles along marked previously existing edges.

Though progressive meshes have proven quite powerful, there is another more fundamental method of compressing geometry. Groups of triangles are packaged together in triangle strips or triangle fans. Triangle strips are structured such that each new triangle consists of the newest vertex specified plus the previous two. Triangle fans are structured such that each new triangle consists of the newest vertex specified plus the previous one and the original vertex in the fan. Thus n vertices define n-2 triangles in both cases. Both of these structures share the feature of each successive triangle sharing vertices with the previous one. By sharing vertices, one reduces the number of specific indices required to be specified for a given number of triangles. However, these structures have one basic limitation - the number of triangles that can be contained in a single strip or fan is generally quite small. As well, determining an ideal set of triangle strips or fans to construct from a triangle mesh can be an extremely time consuming process, as one is trying to find a successive series of longest paths across an object that together make most efficient use of its topology.

Yet, where speed of determining these structures is not of great concern, they have proven extremely useful, and some recent technologies have even attempted to exploit properties of both triangle strips and progressive meshes [Ross99]. Triangle strips and triangle fans remain one of the main techniques of performing simple lossless compression on groups of triangles. Because of their simplicity of use, they've remained a fundamental of the real time rendering community since

its inception. They also make extremely efficient use of memory and enable simple local register based caching of vertex data on a 3D accelerator. This has made them one of the preferred techniques for speeding up hardware rendering, as less bandwidth is required to transmit strips and fans, they require little effort to decode, and don't cause any misses in the graphics hardware's vertex cache.

2.5 The Hilbert Space Filling Curve

The Hilbert curve, first proposed by David Hilbert in 1890 [PeJS92, p. 96] and which is a subset of the more general Peano curves, defines a one dimensional curve that passes through every point in a two dimensional space. Each time the resolution of the space to be filled is zoomed in on by a factor of 2, the curve is iterated. The original space the curve filled is subdivided into four new quadrants. When iterated, each linear portion of the previous curve is replaced by an entire Hilbert curve. Thus, four original vertices used to construct the curve are replaced by 16 vertices, and then 64 vertices, and so on, ad infinitum. As the number of iterations grow, the curve begins to fill the given space entirely, and yet it should be noted the curve is both open, and never crosses itself [Kins99].

This self-similarity defines the Hilbert curve as a fractal, and interestingly, the Hilbert curve was the very first illustrated example of a fractal. This self similarity and winding behaviour that the curve takes on in a given space provides it with some useful properties. For example, the Hilbert curve has been demonstrated to be effective for color dithering techniques as it provides a straight forward distribution of points while not producing unwanted connective artifacts [PeJS92]. It has also been demonstrated that the Hilbert curve exhibits extremely powerful clustering behaviour [MJFS99]. This clustering behaviour can be used in image compression to

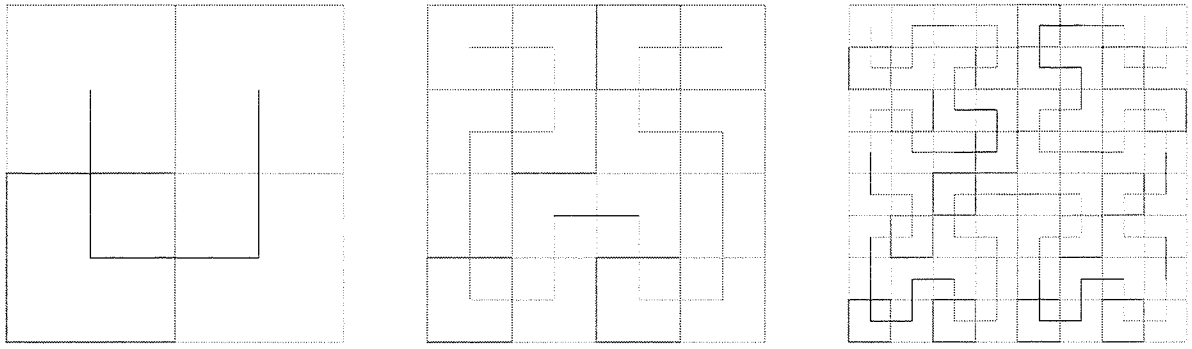


Fig. 2.7 Three iterations of the Hilbert curve

further reduce entropy by clustering together pixels with a higher degree of correlation than other linear mappings (such as a more straightforward linear traversal) demonstrate.

2.6 Quaternions

Quaternions were invented by Sir William Hamilton in 1843 as an alternate form of expressing complex numbers [WaWa92]. Rather than describing complex numbers using two components, one real and the other imaginary, Hamilton conceived of a system wherein imaginary numbers consist of a real scalar component and a three component imaginary vector,

$$q=(s,v)=s+v_x i+v_y j+v_z k \tag{2.6}$$

where s is the scalar part of the quaternion q, and v is a three-space vector, with v_x, v_y, and v_z as its respective components. This allowed him to construct complex volumes to further extend concepts beyond the limitations of a single complex plane.

Quaternions can be used to represent Euler angles, and unlike Euler angles, quaternions

do not suffer from the problem of gimbal lock. Quaternions also allow for the easy implementation of interpolation between object orientations. Quaternion space defines a hypersphere, and thus to interpolate between two quaternions, one finds the shortest arc between two points on the surface of the hypersphere. This process is called spherical interpolation or *slerping* [WaWa92].

To demonstrate the necessity for quaternions, it can be shown that more than three degrees of freedom are required if one wishes to avoid the problem of gimbal lock [Heck97, p. 15]. Common techniques of doing this include specifying an entire nine value 3x3 matrix, 16 value 4x4 matrix, or using two three-space vectors, but the minimal representation is to use only four degrees of freedom. Quaternions only require four values to be specified and thus provide an accurate and yet concise representation of orientation values.

2.7 3D Animation

There are two primary techniques used to animate 3D models in time: motion attributes and keyframes. For the motion attribute technique, during each timestep, objects get assigned a translation, rotation or scale factor. These attributes are either constants, or based on some set formula and allow for very simple animations. While most motion attribute systems are fairly simplistic, motion attributes can be used in the creation of procedural animation systems. Procedural animation, while very powerful, is often rather complicated to both implement and use, and thus is currently not very commonly utilized.

Keyframing, on the other hand, is a very common methodology that owes its origins to hand-drawn cel animation techniques developed by Walt Disney studios [WaWa92, p. 345]. In traditional keyframing, only certain 'key' frames, frames containing characteristic positions of the

models, are actually drawn by the core animation staff. The frames in between these keyframes are then hand-rendered by interpolating between the keyframes. This process is often referred to as *inbetweening* or *tweening*.

In 3D animation, the process is similar. Most commercial animation packages make use of individual frames in the same method as traditional cel animation. Thus, specific frames are used as keyframes, where object characteristics are specified, and the characteristics of the models are interpolated for the remaining frames.

Choosing the best method of interpolating the inbetweened frames is by no means trivial. Linear interpolation, while perhaps the easiest technique to implement, often proves inappropriate. Linear interpolation can often lead to very inappropriate motion, since although it does guarantee continuous motion, it does not guarantee continuous derivatives. This can cause very abrupt changes in motion. For example, an attempted interpolation of projectile motion with very few keyframes would contain visual errors as the projectile passed the peak of the arc.

Instead, it is recommended that spline interpolation be used in the place of linear interpolation for most cases. Spline interpolation using parametric curves produces naturally curved paths, which usually allow for a more realistic reconstruction of motion. Motion describing splines also require fewer keyframes than a series of linearly interpolated keyframes to define rather complicated motion.

2.8 Parametric Curves

In general, parametric representations of curves and surfaces are more useful than implicit forms. Where a parametric curve is defined as its parameter u is swept from 0 to 1, implicit curve definitions of similar curves require the possible solving of nonlinear equations. Traversing an

implicit curve to determine its shape is not straightforward; solving for a specific variable can produce multiple roots or points without any roots. In general multivalued implicit functions cannot be used to trace out their curves or surfaces.

Furthermore, parametric curve forms allow the natural progression through the curve structure - a parametric definition of a circle allows the construction of the circle by traversing its circumference in one direction. This facilitates the approximation of the parametric curve using line segments, and parametric surfaces using polygons. By varying the rate at which the parametric curve is sampled one can construct approximations of different resolutions.

Lastly, parametric definitions of curves allow for a large number of powerful variations useful in computer graphics. Only the two specifically used in this thesis are presented below, but a wide variety of curve and surface definitions are used consistently in computer graphics to define surface structures and motion paths.

2.8.1 Bézier Curves

One of the more dominant parametric curve representations is the Bézier curve, named after its originator. A set of control points $p_0, p_1, \cdots, p_n$ define a Bézier curve $Q(u)$ of degree n such that [WaWa92, p. 73]:

$$Q(u)=\sum_{i=0}^{n} p_i B_{i,n}(u) \tag{2.7}$$

where

$$B_{i,n}(u)={}^{n}C_i u^i (1-u)^{(n-i)} \tag{2.8}$$

is a Bernstein polynomial, and ${}^{n}C_i$ is a binomial coefficient

$$^{n}C_{i}=\frac{n!}{i!(n-i)!} \tag{2.9}$$

It should be noted that in computer graphics, Bézier curves used tend to be of degree $n=3$. The four Bernstein polynomial basis functions for a third degree Bézier curve are:

$$B_{0,3}=(1-u)^3 \tag{2.10}$$

$$B_{1,3}=3u(1-u)^2 \tag{2.11}$$

$$B_{2,3}=3u^2(1-u) \tag{2.12}$$

$$B_{3,3}=u^3 \tag{2.13}$$

Thus the full representation of a Bézier curve of the third degree (or Bézier cubic) is given by

$$Q(u)=p_0(1-u)^3+3\,p_1u(1-u)^2+3\,p_2u^2(1-u)+p_3u^3 \tag{2.14}$$

Sometimes it is preferable to specify Bézier curves using only two endpoints, as interior points are unavailable. Without proof it is stated that the derivatives at the endpoints of a Bézier curve of degree *n* are given by [WaWa92, p. 75]:

$$\frac{dQ(0)}{du}=n(p_1-p_0) \text{ and} \tag{2.15}$$

$$\frac{dQ(1)}{du}=n(p_n-p_{n-1}) \tag{2.16}$$

or for our cubic we have

$$Q'(0)=3(p_1-p_0) \tag{2.17}$$

$$Q'(1)=3(p_3-p_2) \tag{2.18}$$

Substituting Eqn. 2.17 and Eqn. 2.18 into Eqn. 2.14 provides:

$$Q(u)=(1-3u^2+2u^3)\,p_0+(u-2u^2+u^3)\,p_0'+(3u^2-2u^3)\,p_3+(u^3-u^2)\,p_3' \tag{2.19}$$

which is only in terms of a curve's endpoints and their derivatives.

2.8.2 TCB Curves

Local tension, continuity and bias (TCB) curves were first introduced in order to give a high degree of control over keyframe based animation using only a few very simple parameters. First presented by D. Kochanek and R. Bartels [WaWa92, p. 355], this variation on the simple Hermite form uses these three new parameters to calculate the tangent vectors between two endpoints of a curve - or rather, between the positions or orientations at two specific keyframes. TCB splines attempt to control the curve shape through understandable parameters rather than simply mathematically abstract numbers. This has made them desirable for artists attempting to construct keyframed animation sequences without large amounts of confusing mathematics. All three parameters, range between 0 and 1. Tension controls how sharply the curve bends, continuity controls the rate of change in curve speed and direction, and bias controls the direction of the curve as it passes through the given keyframe.

Given the current keyframe in a sequence k_i, the incoming tangent for the keyframe is given by

$$TS_i=\frac{(1-t)(1-c)(1+b)}{2}(P_i-P_{i-1})+\frac{(1-t)(1+c)(1-b)}{2}(P_{i+1}-P_i) \qquad (2.20)$$

and the outgoing tangent is given by

$$TD_i=\frac{(1-t)(1+c)(1+b)}{2}(P_i-P_{i-1})+\frac{(1-t)(1-c)(1-b)}{2}(P_{i+1}-P_i) \qquad (2.21)$$

where t is the tension term, c is the continuity term, b is the bias term, and P_i is the position in 3D space of the k_i-th keyframe. These values can be calculated for any pair of keyframes, since to determine a value between keyframes k_i and k_{i+1}, we need the position at the keyframe k_i

as the start point for the Hermite form, the position at the keyframe k_{i+1} as the end point, the outgoing tangent for the k_i keyframe TD_i, and the incoming tangent for the k_{i+1} keyframe TS_{i+1}. These values are plugged into the basic Hermite equation to extract position or orientation information for a given set of keyframes.

In order to adjust the speed at which the curves are interpolated to allow for nonuniform spacing of keyframes, an easing parameter is introduced to adjust the keytiming. The easing parameter of some local segment i of a curve with a global parameter U is given by:

$$u=\frac{U-t_i}{t_{i+1}-t_i} \tag{2.22}$$

The parametric continuity of the curve at $U=t_i$ implies that the value of dQ/dU (Q being the curve value) is the same regardless from which side the position P_i is approached. This is shown as

$$\frac{dQ}{dU}|_{u\to t_i}=\frac{dQ}{dU}|_{t_i\leftarrow u} \tag{2.23}$$

When expanded using the chain rule, dQ/dU is given by

$$\frac{dQ}{dU}=\frac{dQ}{du}\cdot\frac{du}{dU}=\frac{1}{t_{i+1}-t_i}\frac{dQ}{du} \tag{2.24}$$

where i is the segment of the curve over which U falls. Substituting Eqn. 2.23 into Eqn 2.24:

$$\frac{1}{t_i-t_{i-1}}\frac{dQ}{du}|_{u\to t_i}=\frac{1}{t_{i+1}-t_i}\frac{dQ}{du}|_{t_i\leftarrow u} \tag{2.25}$$

demonstrates that there is a discontinuity in the local derivatives at the key position. The discontinuity is given by

$$\frac{dQ/qU|_{t_i\leftarrow u}}{dQ/qU|_{u\to t_i}}=\frac{t_{i+1}-t_i}{t_i-t_{i-1}} \tag{2.26}$$

Thus, in order to deal with non-uniform distributions of keyframes, the start and end tangents of a

segment are multplied by [WaWa92, p. 355]

$$\frac{2(t_{i+1}-t_i)}{t_{i+1}-t_{i-1}} \quad \text{and} \tag{2.27}$$

$$\frac{2(t_{i+1}-t_i)}{t_{i+2}-t_i} \tag{2.28}$$

respectively. This scaling factor is often referred to as easing the curve.

2.9 Existing Systems for the Transmission of 3D Animation

While the approach taken for the transmission of 3D animation in the thesis is a novel one, this thesis is by no means the first attempt at the transmission of geometric content over the Internet. Most current techniques revolve around the transmission of content that is meant to be viewed interactively, with the viewer able to at least be able to control a virtual camera. None of the current 3D techniques described below appear to stream 3D content scheduled based upon visibility.

2.9.1 Virtual Reality Markup Language

Virtual Reality Markup Language (VRML) was originally conceived to be to 3D graphics what HTML is to text. VRML was meant to provide a simple description language for 3D content that would allow users to navigate virtual worlds in a similar fashion to how users browse the World Wide Web. As VRML was somewhat ahead of its time, and the consumer hardware required to display it was not widespread during VRML's early development, it has never really taken off to any large extent.

VRML consists of a straightforward textual description of 3D scenes, including triangle based geometry, keyframed motion, and the ability to hyperlink to other VRML files. In general VRML assumes a user controlled dynamic camera, and allows for user navigation through VRML scenes.

While VRML is a relatively simple scripting language, the recently updated VRML97 (VRML 2.0), specification now allows inline Java code to be included [CaBM97]. This allows for the possibility of VRML to describe some extremely complex and diverse animated behavior, including the ability to respond to user input, collision detection and other such dynamic systems. However, interactivity comes at a price, since the ability of users to navigate through scenes requires that content must account for free manipulation of the virtual camera. Scenes in VRML must take into consideration the ability of the viewer to walk through and view the virtual world from any possible location or point of view.

Yet, though VRML is quite powerful in its ability to describe scenes and allow interaction with them, it can be observed to be highly non-optimal for use over narrow bandwidth channels. Though VRML files are generally compressed using GNU *gzip* to reduce their storage requirements, the scene description is in raw text. Gzip is a compression technique based around LZ77 that uses Huffman encoding to encode the pointer-length pairs into a tree. While providing very efficient compression, this technique requires a great deal of overhead due to the manipulation of the Huffman tree structures.

The final primary drawback of VRML is the requirement for it to be entirely specified before it can be rendered. Objects that are described under VRML must be fully specified upon creation, preventing the possibility of progressive transmission of these objects. For a VRML file to be viewed by a user, it must first be fully downloaded, then decompressed and parsed before it can be viewed.

2.9.2 Macromedia Flash and Shockwave

Macromedia's Flash and Shockwave products are probably the most commercially successful examples of vector based animation systems. Used as multimedia presentation tools, Flash and Shockwave support the streaming of various types of multimedia over the Internet. Flash focuses specifically on streaming vector based animation at very low datarates, while Shockwave has a more general purpose approach. While both applications do an excellent job of streaming different types of 2D animation, neither focus heavily on nor optimize their performance for 3D content and thus are generally not appropriate solutions for animating 3D sequences.

2.9.3 Progressive Meshes

As described earlier, progressive mesh technology was first demonstrated as a technique to transmit polygonal meshes over the Internet. As progressive meshes allow the progressive filling in of detail, they provide a simple and effective technique for building up interior detail of meshes over time. To build the mesh up progressively, each 'block' transmitted consists of an entire detail level. The new detail level is added to the previous data via vertex splitting and thus the mesh progressively gets more and more detailed as time progresses and more data arrives.

The power of the technique has been demonstrated by its wide spread adoption as a continuous level of detail technique and its wide use as a basis others have attempted to refine through further research, including Hoppe's own ([Guez99], [ElVa98], [Ross99] for just several examples). Progressive mesh support was even included in early versions of Microsoft's DirectX 3D programming software development kit.

2.9.4 Microsoft Chrome

Microsoft's Chrome product has been shelved and remains an experiment, but its original purpose was to provide a framework to allow for easy integration of 3D content side-by-side with commonplace 2D content on the World Wide Web. At the time it was also to be used as a basis for a future 3D graphical user interface planned for some future version of the Windows operating system. Whether or not Chrome eventually gets reintroduced in a future Microsoft product remains to be seen.

2.9.5 WildTangent's Web Driver

WildTangent's Web Driver is an interesting plugin that sits as an environment in which can run various types of content developed using either JavaScript, Visual Basic or C++. Whereas the other systems mentioned here are primarily directed to just the display of 3D content, or in some cases, limited interaction with this content, Web Driver is powerful enough that it is focused on running much more complex content. Most of the examples provided by WildTangent demonstrate simple 3D games, and allow for fairly high degrees of interaction, supporting game

controllers and dynamics engines.

While Web Driver doesn't inherently support the progressive transmission of model data, it does support the progressive transmission of textures, and it allows for scenes to be built object by object over time. For example, it is possible to create a simple game in which the control code, artificial intelligence, and physics engines are all loaded first, followed by the models and their textures. A game can be made initially playable to the user, and more detail can be filled in later, making the game look more attractive once the essentials have arrived. Thus Web Driver can be used to produce extremely advanced interactive web content, but does not feature many of the basic progressive transmission technologies demonstrated by the other systems listed in this section.

2.9.6 Pulse Entertainment Pulse Player

Pulse Entertainment's Pulse Player is a recent introduction in providing 3D animation over the Internet. Once initial content has been received by the player, further information required for the animation, such as object motion or speech can be streamed into the player as it becomes necessary due to user selected actions. Pulse Player supports an extensive set of options, including a scripting language JoeScript, definable behaviors, and skeletal animation. Pulse Player does not appear to support progressive transmission of the models themselves, and instead seems to merely support the active streaming of behavior scripts and audio.

2.9.7 MPEG-4 Version 2

Revised in December 1999, MPEG-4 version 2 includes the capabilities to stream 3D geometry in addition to video and audio. As 3D information content is less predictable than video

and audio content, and may require interactivity from the server (in the case of interactive virtual worlds, for example), 3D content is transmitted through a side-stream separate from the usual audiovisual content.

Based upon existing work for VRML [MPEG4a-d], the MPEG-4 version 2 supported scene description language allows for a wide variety of capabilities. For example, it is possible to integrate video and audio streams into virtual worlds in the form of either video textures (for example, a virtual television in the virtual world) or positional audio. Also included as of version 2 of MPEG-4 is the ability to construct deformable human models derived from a standard parameter set, to enable the use of avatars in virtual environments. Support for modeling of the human body and its motion is the primary 3D enhancement to version 2, as great effort has been put into the efficient compression of body gestures using discrete cosine transforms.

At this time, much of the MPEG-4 standard is in development and therefore is still not yet public. It currently appears that the scene description format utilizes a superset of VRML, allowing for the same primitive types and constructs. As well, the scene description system is to support progressive scene and object construction, with an absolute limit of 1024 active objects at any time. The MPEG-4 3D subset also appears to support some form of progressive geometry, based on the progressive mesh work mentioned earlier. The 3D subsystems have been designed with interactivity as a primary focus so that MPEG-4 content could be integrated into set-top boxes or simple games. Therefore MPEG-4 has been optimized to support the up-streaming of data back to servers in addition to allowing event triggers to be directly specified in the MPEG-4 files.

2.10 Summary

Analysis of these techniques and understandings of their limitations led to the design of this thesis and the development of the techniques described within. The combination of understanding the limitations in current techniques as well as the realization of how these limitations could be exploited allowed for the design of our new methodology, as described in the section that follows.

The sections described in the background provide the necessary theoretical information to follow why current techniques are inadequate and do not properly exploit natural tendencies in 3D animated sequences. By understanding how previous 3D animation systems fell short, we could improve on their performance and produce a more effective solution to streaming non-interactive real time 3D animation over the Internet.

CHAPTER III
DESIGN

The design of this thesis took a top-down approach. Design began by considering how it might be able to generally improve on current video compression techniques of displaying 3D animation over the Internet.

3.1 Animation Coherence

One of the most important factors in compressing animation or video is attempting to properly exploit coherence between animation frames. It is generally assumed that the source material naturally contains much coherence, but this temporal redundancy is often difficult to utilize. Where video compression techniques require that the encoder try to artificially extrapolate coherence between groups of pixels, compressing 3D animation from its source scene descriptions provides us with explicit knowledge of the internal coherence. While much research has been put into trying to segment a scene into complicated components and then tracking their motion, scene descriptions directly state how objects are constructed and how they travel through their environment.

The scene descriptions to be dealt with in this thesis consist of machined parts, such as those in engines and turbines, shown in operation or cutaways. These machined parts are static objects whose shapes do not warp and bend during animation, but instead are displayed moving relative to one another according to their function. Thus, if a limitation is placed on the system stating that object topologies are not allowed to change, an even greater level of coherence can be assumed in scenes. Once an object has been fully described, it does not require future description

in the sequence. Instead, only its movements through space need to be provided, allowing for a level of coherence in the animated sequence not possible under pixel-based video codec.

3.2 Interactive vs. Non-interactive

Though it might not immediately be clear, the requirements of rendering interactive environments are quite different from those required to render non-interactive ones. In general, interactive environments need a great deal more information, as they must allow for user navigation of the surroundings. These needs can be restricted to some degree, yet so long as the user is allowed to take control of the virtual camera used in rendering the scene, every possible permutation of how the user might choose to travel through the world must be accounted for. Authors of the environment must consider how the world will appear from any of the various vantage points the viewer is allowed to choose from.

In contrast, in a non-interactive environment, the viewer has no control over the camera. While less immersive than an interactive world, non-interactive worlds generally require less information, as the world that is viewed by the user has specifically been restricted by the world author. Consider the simple example of a theatrical set, where the stage usually consists of only specific portions of a room. Or consider a scene in a movie in which only a portion of the environment, which is presumed by the viewer to extend beyond the screen boundaries, can be seen. In interactive environments, these unseen regions would have to be accounted for, while in a non-interactive environment, anything that the author deems out of view need not be taken into consideration. Because of this, non-interactive environments generally do not require as much information to be described.

Furthermore, in a non-interactive deterministic environment it is possible to predict every

object that will be visible, and to evaluate what portions of them will be visible to the camera. Foreknowledge of what in the scene will be visible, can allow the further removal of unnecessary information and scene geometry. Information as to when parts of the scene become visible allows for the progressive construction of the scene description, adding information into the scene as it becomes required.

3.3 Data Types and Ranges

In order to reduce the overall size of a scene description, we must apply some form of compression technique. Before a compression technique may be selected, efforts must be made to reduce the size of the overall animation sequences. If one can decrease the number of bits required to specify different values, or group like values together to exploit natural redundancy before compression is applied, one can improve the performance of various compression techniques, as the entropy of the source signal has been reduced. This is attempted by using some knowledge of the limitations of current realtime 3D animation.

The first attempt to reduce the entropy of the file consists of byte-plane partitioning a series of related floating-point values (or *floats*). For the purposes of this thesis, only single precision (32 bit IEEE standard) floats are used. These floats are structured such that they are made up of a sign bit, followed by an 8-bit exponent and then a 23-bit mantissa.

It has been observed that objects with axial symmetry about their local origin or objects with vertices clustered together will tend to have floating point values with the same exponents. By packing these exponents together, one should be able to reduce the first-order entropy of the vertex data. This can be accomplished by simply rotating the floating point structure by one bit, and placing the sign bit at the end of the structure and byte aligning the exponent. To increase the

likelihood of equal byte values, vectors are separated into their vector components (x, y and z) and lined up with the same components from other vertices. Thus, a series of three vertices a, b and c will be broken up into $a_x b_x c_x$, $a_y b_y c_y$ and $a_z b_z c_z$, with each float's component bytes separated and aligned in the same manner. This should allow further compartmentalization of the unvarying portions of a series of floating point vertices or vectors.

Another observation is that real time 3D engines only tend to be able to render on the order of several hundred thousand polygons per second due to limitations imposed by current consumer hardware. Because of this, artists creating models for use in such engines are usually very limited in the number of polygons they can use for individual models as compared to non-real time 3D engines. Such models rarely exceed 3000-4000 triangles, However many low detail models require less than 100 triangles. Much research has been conducted in recent years ([HDDM93], [Hopp96], [ElVa98], [Guéz99]) to reduce model complexity to fall within the restrictions imposed by real time rendering engines, while minimally affecting the appearance. Since an approximate upper limit to the number of possible triangles is known, the storage requirements of numeric indices and reference numbers can be reduced.

The following scheme is used for reducing the storage requirements of unsigned 16-bit integers (often referred to as an unsigned short). When reading an unsigned short value from a file, a first byte is read. If this byte has a value between 0 and 199 (inclusive), we simply use the value of this byte for our unsigned short value. Thus, values between 0 and 199 only require one byte of storage. If the value of the byte is between 200 and 255, then we apply the following formula

$$f=(a-199)*200+b \tag{3.1}$$

where f is the final unsigned integer value, a is the unsigned value of the first byte read, and b is the unsigned value of the second byte read. This allows for a maximum value of 11455 which is large enough to accommodate future increases in system capability, while optimizing for smaller array sizes.

While the above technique for packing floats applies to general 3-value floats, one can improve performance to specifically reduce the storage requirements of normal vectors. As normal vectors are all of unit length, it can be easily observed that they describe points on a unit circle. If one uses spherical coordinates to describe a unit normal rather than those of Euclidean geometry, we now require merely two values to describe normal vector, as the length component will always be fixed at a value of 1. If we encode the remaining two angular values with 12 bits each, we can reduce the original 12 bytes required to store a normal vector down to a mere 3 bytes with a minimal reduction in resolution.

3.4 General Software Issues

Early on it was decided that as the foundation of this thesis was the compression of 3D animated sequences, the thesis would require two separate applications - one to compress the sequences and the other to decompress and view them. While the specifics of the compressor were not particularly important at first, it was known that the decompressor would need to perform two basic functions simultaneously. It would first have to be able to render the sequences and second, decompress the incoming streams. The decompressor would have to be able to do both simultaneously as the basic design calls for the viewing of animation sequences as they arrive.

Due to the speed required in order to decompress a sequence and render it simultaneously,

the decompression scheme would have to keep processing to a minimum and rely upon the majority of optimization of the sequence taking place during the compression phase. Though consumer hardware continues to improve year after year, the decompressor would have to be able to run on at least a reasonably powerful consumer system, while not limited to only the true high end of the consumer market. It was decided, accordingly, that the decompressor's 3D engine would make use of one of the existing 3D application programming interfaces (APIs) and that the decompressor would require a 3D accelerator.

3.5 Constructing Objects

From the very beginning of this work, one of the decisions that was made was that objects should be allowed to be constructed over time, as new information about these objects becomes necessary to the viewer. How to practically approach this problem was another matter. One possible approach was to simply adopt progressive mesh technology. This was ruled out, as while progressive meshes do a very good job of refining the detail of a model by adding new vertices between existing ones during vertex split operations, they do not account for a more fundamental problem. If objects are being animated, and one wishes to only append new polygons into the scene as they are required, unless the viewer is zooming in on that object, new polygons will be required at the object boundaries. Consider for example the case of an object slowly rotating in place. As it spins, new portions of it come into view, and the new polygons representing these portions must be added to the scene. Progressive meshes couldn't properly account for the continuous addition of new polygon boundaries, especially at all levels of the detail hierarchy. Thus, a new approach would have to be considered.

Another issue that required attention was that, as our decompression would take place

online, while the scene was being rendered, the decompression process could not be allowed to become overly complex. If the calculation or memory manipulation overhead for the decompression process became excessive it would slow down the real time rendering process, possibly to the point of making it either extremely slow or useless. Compression could be allowed to be time expensive, as it would take place offline and would be time dependent.

Instead of trying to re-evaluate progressive meshes, a method was conceived where an object was merely defined as an array of triples of indices into a vertex array. As these new triangular faces would be required, new vertices would be added to the vertex array if necessary. Unfortunately this would also negate any possibility of being able to use either triangle strips or fans as the constant addition of new triangles to existing strips would incur overhead and thus undermine any benefits strips might otherwise naturally provide. This overhead would result from the need to maintain the strip and fan structure when internal portions of the structures might be missing from the list of required faces. As well, these structures would either suffer or break down entirely when encountering degenerate cases where very small numbers of disconnected triangles were added to the scene during a given time slice. Overhead caused by these degenerate cases, which were predicted to be quite common, would remove any possible advantages strips or fans might have initially provided.

Yet it should be noted that vertex arrays can be very efficient structures. It has been shown that smooth surfaces tend to have vertices with average valences of 6 (attached to six different edges) [ZSSw96]. As individual vertices are reused in an average of six triangles, the vertex array structure naturally accounts for a large amount of redundancy that would otherwise need to be stored as separate structures. Vertex arrays can be seen to be very concise and efficient structures for specifying triangle information.

While triangle strips could no longer be used by the decompressor's rendering engine,

OpenGL does however support the use of vertex arrays to optimize the transmission of vertex data to rendering hardware. Thus the construction of arrays of face and vertex data by the decompressor provides an effective methodology for accelerating rendering, while allowing for simpler data management by the decompressor.

The array structures had been decided on for representing triangle data, but the acquisition of this information had yet to be worked out. The problem was that to efficiently send animation data - in other words to send this data in the order in which it is required - is no simple task. This difficulty arose from the problematic nature of the visible surface determination problem. Determining the order in which a scene becomes visible, and which parts of that scene actually can be seen by a virtual viewer is not an easy task. The solution decided upon was to render the animation sequence in the correct order, and simply check what polygons appeared and the order in which they appeared. To accomplish this task, polygons would be tagged using colors, where the colors would represent number references. By scanning the pixels of the resulting rendered images for colors, these number references could be retrieved. The list of retrieved triangles would identify which polygons could be seen in the given frame, and would be appended to a list along with a reference to the frame number in which these polygons first appeared.

Once all of the given list of frames had been rendered, the large list of required triangles would be sorted by the frame numbers in which they appeared. This list would now contain the animation's polygonal data in the order in which it needed to be streamed. In order to construct the frame list to ensure the correct enumeration of all polygons, one would have to sample the frames at a rate double the expected playback frame rate. As well, different rendering engines do not guarantee exact per-pixel duplication of the rendered result, due to differences in how various engines select which pixels to render. These differences determine whether or not various types of triangle edges will themselves be rendered when the entire triangle is being scan converted and

filled in with a given color. Selection of the incorrect edges to hide could cause a small triangle that is mainly occluded to not appear when rendered under one engine but appear under another. To account for these anomalies, each frame is rendered with double the width and height of the target frame resolution, so that sub-pixels of the target frame are accounted for, and possible differences in edge selection by the rendering engines will not affect the final playback output.

3.6 Redefining Materials

3D Studio Max Release 2.0 allows two basic methods for defining the surface appearance of a geometric shape, the use of a single solid color (referred to as an object's wire color) or a material. Materials can vary greatly in complexity, and are meant to allow objects to take on the surface properties of real world objects, such as metals, plastics, and organics. Materials as defined in 3D Studio Max can be very complicated, as they allow for hierarchies of sub-materials interacting together to produce some final surface property. While this enhances 3D Studio Max's capabilities by allowing more accurate modeling of an artist's conception, it makes the potential complexity of object materials too high for this same system to be practical for real time reconstruction. As well, many of the procedural textures used to generate the sub-materials have high computational overheads possibly making them impractical for real time reconstruction as well.

Thus, the remaining solution for the transmission of object materials is to make use of the post-processed material generated by 3D Studio Max for its rendering. To generate the object material, the most straightforward manner would be to make use of the built-in rendering engine available to 3D Studio Max to render out the complex material hierarchy to a simple planar texture map. Two different methods of exporting the material properties would then be

supported. The first would provide low detail information, and would work most appropriately when dealing with scenes that require very little texture detail and contain small numbers of vertices. Instead of attempting to transmit the entire texture, one would instead sample the material values at each vertex, and transmit simple vertex colors in the place of an entire texture. For simple textures with little detail, or images of fairly continuous color gradients, this technique could often prove effective of at least conveying the general visual properties of an object while requiring very little information be transmitted. For more complex textures, this technique would break down, as detail and discontinuities in the texture would not be maintained, causing the appearance of visual flaws in the final playback rendering.

The higher detail, and more general purpose solution to be used would be to take the sampled texture and transmit it through the stream as a texture map. Texture maps compiled from material hierarchies would need to be sampled at high resolutions (256 x 256), yet often a high amount of detail in the texture would not be required. In addition, in many cases it would have been desirable to to allow cases in which texture detail is increased over time, for example if an object is being approached by the virtual camera. Therefore, when objects were to be scan converted, they would also be analyzed to determine a rough approximation to the target screen area occupied by the object. This area estimate, which could be determined by noting the two-dimensional bounding box occupied by visible pixels belonging to the particular object, would allow the determination of how much texture detail would be required for the playback render.

The need to be able to add texture detail as time passes naturally coincides with the preference of real time 3D rendering engines to use MIP maps to enhance visual quality rendered texture maps. As constructing these MIP maps by a real time engine would normally require extra computational overhead, one can alleviate this overhead from the rendering engine by providing these MIP maps to it directly.

When being transmitted to the decompressor, one would prefer to have the texture maps or rather, the various MIP map levels compressed to reduce storage space. This would be extremely important as a 256 x 256 pixel texture map at 24 bit color would require 196608 bytes to be specified. As it would be preferred to transmit the textures with as little data loss as possible, a compression technique that effectively reduced the storage requirements while leveraging the use of the MIP maps was required. One final requirement on a candidate algorithm was that decompression of the images must require a minimal amount of processing overhead, as the decompression engine would not be able to suffer the heavy processing overhead required by standard algorithms like JPEG. A heavy processing overhead would significantly impact the performance of the decompressor application's rendering engine, and cause noticeable drops in frame rate.

Due to the need to transmit multi-resolution versions of the texture maps so they could be constructed over time progressively in conjunction with the need to compress the texture data, wavelets were selected as the best candidate set of techniques. When wavelets are applied to image data, they naturally provide a series of multi-resolution reproductions of the original image. As wavelets are compactly supported, coefficients tend to compress effectively using simple run length encoders, or multipurpose lossless encoders like LZ77 and its derivatives. However, though they do not require a great deal of processing overhead while leveraging the multi-resolution features of MIP maps, wavelet decompression can exhibit heavy processing overhead for more complex wavelets of high degrees. Therefore, the design would call for only the simplest of wavelets, the Haar wavelet, to be used for the compression/decompression process at this time. The design specified that it would be possible to allow for future inclusion of support for other wavelet types though, as future increases in available processing power delivered by the availability of faster and faster consumer computers would compensate for the increased

overhead.

As it is necessary to encode large quantities of floating point coefficients, one would prefer to encode related coefficients together. Typically, a simple manner is to simply treat the entire two-dimensional array of coefficients as a simple one-dimensional array, and encode them. While this will allow a lossless compressor to take advantage of horizontally related coefficients, it ignores natural correlation between lines of the original coefficient array. If instead coefficients are grouped using a Hilbert space filling curve, a greater rate of correlation should be arrived at. This is due to the winding structure that the Hilbert curve takes - four coefficients are clustered together in a square block, and then four of these clusters are clustered again. This clustering structure is repeated until the final block is the resolution of the entire bitmap. This has an advantage of placing coefficients together that are spatially proximate in the coefficient array. Spatially proximate coefficients can usually be assumed to exhibit a greater degree of correlation than otherwise, and thus allow greater performance from the lossless compressor. Utilizing a Hilbert winding also naturally breaks a square two-dimensional array into quadrants, making it a helpful and efficient structure for building up MIP maps, as each new level of coefficients can simply assume all previous levels simply made up the new map's first quadrant. Coefficients would be taken for each color plane, with each plane quantized and encoded separately to be reconstructed by the decompression system later. Coefficients would be linearly quantized to 8 bits per coefficient. Though this will not provide entirely lossless compression of the texture maps, the relatively small amount of quantization error should not greatly impact texture reconstruction.

While the overall compressor system would determine an upper limit for required texture detail as a result of the amount of screen area required by the models using the texture, the compressor could further reduce the upper limit required. In some cases original source materials

may have less detail than the maximum MIP map required by the bounding box. For example, if the texture is a simple solid color, and the bounding box indicated that a 128x128 texture was required, bandwidth would be wasted if all MIP map levels up to 128x128 were transmitted, when only a single low level was actually required. To minimize the chances of this happening, a simple test is performed. The texture is rendered out to the standard resolution of 256x256. Based on the current maximum MIP map level (by default equal to the bounding box selected value or some user maximum, whichever is less) the next texture is reconstructed to the current maximum from the coefficients, and to one level less. Thus, if we have a current maximum of 128x128, we reconstruct the texture at 128x128 and at 64x64. We would then take the smaller of these two textures, and scale it up to the resolution of the higher level (doubling its resolution) using a bilinear filter. The peak signal-to-noise ratio (PSNR) of these two images would be taken. If the PSNR was above a given threshold, it would be known that the higher MIP map level is not necessary and the maximum could be reduced by one level. This test would then be repeated until it failed on the given texture.

3.7 Streaming Information

While the term streaming implies that the animation compressor would be directly transmitting data over the Internet to a waiting decompression engine, this was never meant to be the case. Instead, it was decided early on that animation sequences would be compressed into a simple output file, which could then be transmitted to some client machine. Though other protocols existed, the simple and straightforward method of allowing the information to be streamed via HTTP, the basic protocol used for transmitting information from World Wide Web browsers, would provide a simple general case of streaming technology. Though not necessarily

the best protocol suited towards real time operation, using HTTP guarantees support by all World Wide Web servers, requires no additional server-side modification, and provides natural advantages such as guaranteed packet delivery and packet reordering. As well, since many development environments have their own built in functions to accommodate streaming information over HTTP, using HTTP would presumably reduce development time and allow focus on the more complex issues of scene geometry.

Though basic methodologies for reduction of geometry had been determined, the specifics of how to stream this information needed to be worked out. The fundamental problem was that as stated previously, geometry had to be separated into chunks of information to be transmitted. As well, the information being sent would consist of more than straightforward triangle information, as there were texture maps to contend with, as well as defined parent child relationships, scene viewing parameters, and motion paths for objects. Due to the variety of information being transmitted and the need for information to be broken up to compensate for limitations of time, a static structure to the stream would not be possible.

Instead, a command language was required. By using a command language to specify scene information, this information could be output from the compressor in a freeform manner. This would allow for the interleaving of the various types of information in order to break the output using time stamps. The commands themselves would have to compact, in the form of operational codes (*opcodes*) to reduce storage overhead for the instructions themselves. Thus, each instruction would be assigned a simple short integer value to identify it. Following each instruction would be a parameter list specific to the instruction. For example, the specification of a parent-child relationship between two objects would always require the exact same amount of information, as one would need to simply specify the parent and the child. Meanwhile, the addition of a series of faces to an object, for example, would require a dynamic parameter list

whose size was dependent on the number of triangles being added to the object.

Though one could potentially have any sized lists of certain geometry specific data types, such as vertices and faces, being appended to an object, upper limits were placed on how large these lists could be. These artificial limits, set in the compressor, were put in place to allow more efficient operation by the decompressor as, by separating large lists of geometrical information into smaller portions, commands could be more easily parsed, spreading out the command processing overhead. As well, as the decompression buffer was to be small to improve cache efficiency, it was believed that keeping commands limited to smaller blocks could speed up and improve instruction processing.

While these commands would be necessary for describing an animation, not all information required by the animation would need to be directly stated. Instead, unless otherwise specified, some information could be inferred through the command structure. For example, though each object in a scene requires an identification code, and is referenced by object specific commands through these same codes, the codes for each object never have to be directly specified. Instead, when a new object is added to the scene its identification code is implied by the order in which it was added to the scene. At the beginning of each animation being processed, a marker keeping track of total object count is initialized. Whenever a new object is added to the scene, it is assumed to have an identification code equal to the previous total number of objects. Hence, the first object has an identification code of 0, the second has the identification code of 1, and so on.

Through similar assumptions, we assumed objects, unless otherwise specified, had default scales of (1.0,1.0,1.0), were located at the world space origin, and were oriented looking down the Z-axis. Similarly, objects were presumed not to move or change orientation unless otherwise specified. While these examples are quite straightforward, more complex examples of information inferences are possible, such as the dynamic calculation of vertex normals if they weren't directly

included in the data stream. These dynamic vertex normal values could be calculated by averaging the face normals, which themselves are constructed by normalizing the cross product of the edge vectors of any two sides of a triangle face, touching each particular vertex. While dynamically generating normals should produce effective and accurate results it could produce incorrect values for triangle faces making up the object borders due to the trimming of invisible bordering faces.

3.8 Compressing Irrelevant and Redundant Information

While great effort had been made to eradicate all information not specifically required from the data stream to reduce its size, stream length could still be reduced further through compression. As discussed in the background section, there is a variety of compression techniques that could be applied depending on the type of information that was being compressed. Compression used in this thesis was often two fold in nature - first the compression of individual portions of relevant information through foreknowledge of valid data ranges, followed by a pass over the entire constructed stream using a lossless compression technique.

For the first pass range based compression, the system would compress data by quantizing a specific range using a given number of bits. For example, when transmitting quaternions it was known that, as quaternions refer to specific positions along a hypersphere, all four of its parameter values would range between -1.0 and 1.0. Thus, instead of transmitting an entire floating point value for each quaternion parameter, four 16 bit values were transmitted, where each 16 bit unsigned word represented values ranging between -1 and 1.

Similarly, when transmitting vertex coordinates for geometry, though coordinate values could potentially be any valid floating point number, a valid range for a given object could still be

determined by finding its axis aligned bounding box. Given that the bounding box for an object would be transmitted upon the object being added to the stream (assuming that the use of 32 bit packed floats had not been specified), its vertices could be located in 3D space by providing an index of the range (0.0,1.0). This marker could then be encoded to different resolutions depending on how many bits were specified by the user. It was decided that in addition to the 32 bit packed floats, three vertex encoding resolutions would be provided - 8 bits per component, 10 bits per component, and 16 bits per component. The 8 and 16 bit values were selected as they fall on byte boundaries and therefore would be easier for a lossless compressor to correlate. The 10 bit value was selected because, as mentioned in the background, 10 bits should provide enough resolution to encode a 3D model without significant (i.e. visible) data loss.

This general technique of range-coding floating point values was used throughout the compressor. The rest of the specific cases used, which are numerous, are all outlined in Appendix A, which details all the various opcodes.

Once the entire stream would be constructed, it would need to be compressed, as a second pass. In order to compress it, a lossless compression technique that performed well but required little processing overhead would be required. This restriction greatly limited the range of techniques that could be used, as a technique with too high of an overhead would overwhelm the decompression engine and slow (or possibly stall) the rendering of the animation sequence. More complicated techniques like Huffman coding, which rely on large amounts of data manipulation, or Arithmetic coding, which is computationally complex, would prove unsatisfactory.

Instead it was initially decided to utilize Limpel Zev Welsh (LZW) compression, as the decompression process consists of table lookup operations in conjunction with dictionary building. LZW is extremely computationally efficient, its only algorithmic limitations being the need to refresh its dictionary to compensate for changing channel statistics and the memory

overhead required to store the dictionary. Nonetheless the LZW algorithm is patented, and thus must be licensed at cost in order to be used in an application. As paying for rights to the technique was out of the question, its efficiency was moot, and focus shifted from LZW towards other dictionary coding derivatives of LZ77.

After some evaluation, the Storer and Szymanski technique (LZSS) was eventually decided upon. Though its compression performance was not as good as LZW and other more complex techniques, LZSS had the advantage of being computationally simple if one limited the use of heuristics. For example, if the deletion heuristic is not used, the decompression portion of the algorithm is limited to table lookups into the sliding dictionary followed by overwriting old dictionary information. This turns LZSS into an extremely simple algorithm from the computational point of view, because all the overhead of searching into the dictionary to find corresponding best string matches takes place in the compressor, when the overhead can be afforded. If one instead uses the deletion heuristic, compression performance increases, but at significant cost to the decompressor as it must now perform matching string lookups into the dictionary. Unless extremely effective hashing functions are used, this can prove extremely expensive as either linear searches into the dictionary must be performed or large structures must be managed to keep track of dictionary ordering. For these reasons, LZSS was chosen, though design called for it to be implemented without the use of the deletion heuristic.

3.9 Summary

The design of the applications required for the demonstration of the concepts presented in this thesis was an involved process. It required the conception of a compressor and decompressor, and how they could be flexible enough to allow for the dynamic construction of

non-interactive sequences by the decompressor. We developed a method of determining how to arrange scene elements and how to determine the order in which they needed to be arranged using color selection.

With limitations of possible implementation and targeted hardware in mind, we decided to utilize wavelets, specifically the Haar basis, for the compression and transmission of objects' textures. By decomposing the original models' textures into wavelet coefficients, texture maps could be transmitted in a multiresolution manner more desirable for the use with MIP maps, as well as allowing these textures to be more easily compressed. The lossless compression required by the system had to require as little processing overhead as possible, and thus the LZSS algorithm was selected. Together all of these design considerations attempted to account for implementation requirements and the limited processing time that would be available to the decompressor.

CHAPTER IV
IMPLEMENTATION

Once the design had been finalized and completed, decisions had to be made regarding how one would go about implementing the design. Software platforms had to be selected, a technique for compression had to be chosen, and the compressor and decompressor applications had to be constructed. A variety of pitfalls forced several of the decisions that were made.

The software was written under Windows NT 4.0 using Microsoft Visual C++ 6.0. The development platform was a Pentium II running at 400 MHz with 64 MB of RAM. The engines were split such that the decompressor would operate as a stand alone application while the compressor would be accessed as a plugin for 3D Studio Max Release 2.0.

4.1 The Decompression Engine

The decompressor application was written using the Microsoft Foundation Class (MFC) framework. The MFC framework is a series of programming libraries that sit ontop of the fundamental Win32 API (the core of current Microsoft Windows operating systems). By encapsulating the original API in a structured, consistent and object oriented framework, MFC enables programmers to simplify and accelerate the development process. As with the Win32 environment, MFC is largely an event driven system, where events fired by the operating system trigger callback functions that retrieve and deal with these system events and messages.

While using MFC can slow down operation slightly, due to the processing overhead used to simplify the architecture, it allows the rapid and easy development of interfaces. This was extremely helpful for this project, as it allowed the quick construction of the application interface.

The MFC architecture also simplified the work needed to create a multithreaded application, as the MFC architecture is based around a separation of data, and how this data is visualized. In the MFC paradigm, data and its manipulation takes place in a Document, while the data visualization takes place in a View. A single document can possess multiple views, as one might choose a variety of different ways in which to visualize some group of data. Under the Document/View paradigm, the document and its views are separate entities, each running in their own threads. The document reports to the views when they require updating, and the view provides a means for user input into the document. As the MFC architecture is based around object orientation, all of the Decompressor source code was written to take maximum advantage of the object oriented programming model.

The Decompression required the ability to run multithreaded, as it had to allow for the decompression and compilation of incoming data, while simultaneously displaying this information to the user in the form of 3D animation. The MFC architecture naturally provided for these requirements by placing the rendering portion of the code in the View, while placing the streaming, decompression and database compilation code in the Document. The Document could then indicate to its View when rendering would be allowed, and what portions of the database could be rendered (to prevent mutual exclusion issues). The use of multithreading allowed the issue of properly balancing the rendering process and the decompression process to be offloaded onto the operating system.

Implementation of the Document attempted to abstract the entire file streaming process so that it would be invisible to the View, and so that the animation database could be constructed as time proceeded. Therefore, the Document contained a single property, an animation file. This animation file (a C++ class CAnimationFile) was based upon an MFC library class called CAsyncMonikerFile. The CAsyncMonikerFile class was provided by Microsoft with the

introduction of ActiveX, Microsoft's attempt at an Internet application model. The CAsyncMonikerFile class allows one to treat a file existing on an HTTP server as though it was simply a file on a local system. As this data was streamed using TCP/IP, data arrival was guaranteed (assuming no lost or broken connections) and all packets were naturally ordered. While TCP was not as efficient a solution as one could potentially achieve by designing a special case UDP solution, using TCP enabled us to largely ignore the issues associated with transfer protocols, and instead focus on the issues related to the data being streamed.

The CAsyncMonikerFile class had several additional advantages. If one used it to read a file, one merely had to provide the class with the universal resource locator code (URL), and the class would automatically fetch the file off the Internet. As the ActiveX subsystems of the Windows platform operate through libraries provided with Microsoft's Internet Explorer, CAsyncMonikerFile could take advantage of the browser's cache and retrieve the file from the local cache if it was located therein. Furthermore, the CAsyncMonikerFile class, when used in read mode, operated by trapping an event whenever data had arrived and was awaiting processing. This caused the class to behave as an effective data buffer, storing multiple blocks of

```
i.    An event is received stating that data is available for
      decoding
ii.   While there is data in the CAsyncMonikerFile buffer, do:
iii.  Take blocks of a maximum size of 512 bytes from the buffer.
iv.   If the file header has yet to be decoded:
         read and evaluate the raw bytes provided to gather the
         file's attributes.
v.    Take the remainder of the block and perform a lossless
      decompression on its contents, adding this new information to
      secondary buffer AnimBuffer.
vi.   Parse the contents of the AnimBuffer for command tokens.
vii.  If we have received enough data through the file stream such
      that the current connection speed should allow for continuous
      streaming, let the View know that it should attempt to render
      the database.
viii. If the CAsyncMonikerFile buffer is not empty, go to step iii
      and repeat.
```

Fig. 4.1 The behavior of the data retrieval thread

data, to be processed when deemed convenient.

This CAsyncMonikerFile based animation file class contained an animation database (class CAnimation). As data was received and parsed, as shown in Fig. 4.1, the animation database would be filled in with further data describing the animation sequence to be displayed. As one would not wish to begin animating until some set minimum of information had arrived and had been added to the database, the Document would place the View in a standby mode. This standby mode was indicated to the user through a simple message in the View stating that the system was prebuffering the scene.

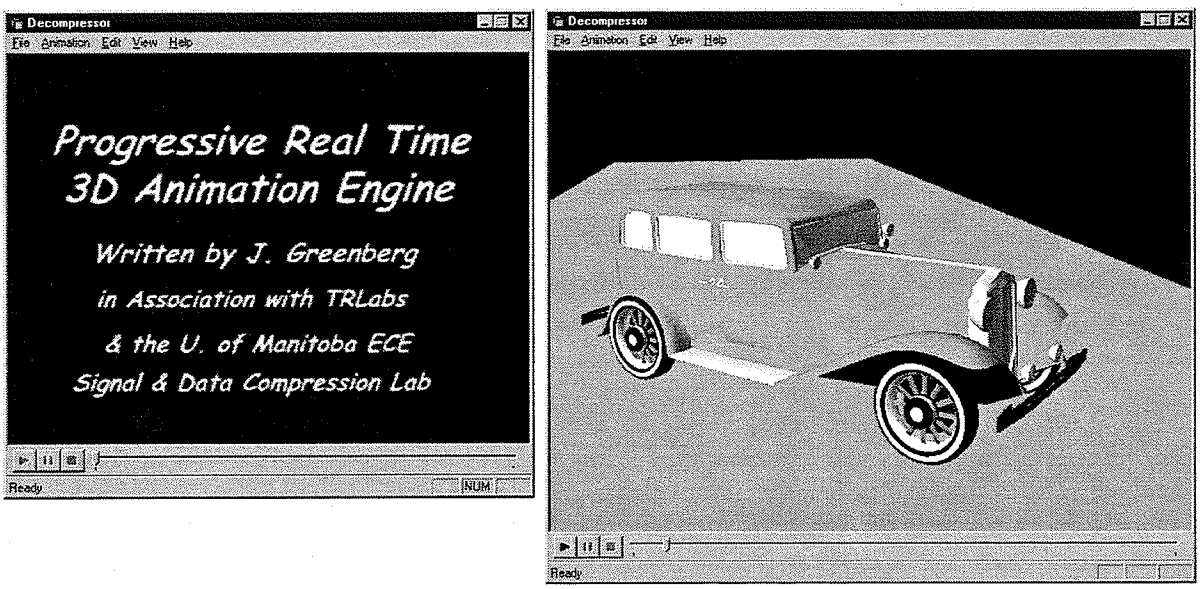
Determination of the minimum amount of required information of the scene was determined using a two-fold process. The user could indicate the speed of the connection to the Internet using the main application menu. Based on the selection, a preset handicap factor was chosen. There were 5 handicap factors corresponding to the user selected connection rate - 28.8 kbps, 56 kbps, ISDN (64 kbps), ADSL (128 kbps) and T1 (1.5 Mbps). Handicap factors were set equal to approximately 70% of the total potential speed of a given connection rate. For example, for a 28.8 kbps connection, the handicap factor would have been set to 2500. Thus, for a 30 second animation sequence being downloaded over this same connection, the file would have to be buffered except for the last 75,000 bytes. As the header contained the length of the compressed file, the Decompressor would calculate the minimum number of bytes required such that once that number of bytes had been retrieved, one could safely begin animating. This global 'safety' factor would indicate that from the given point forward, assuming an average distribution of animation data over the remaining animation frames, the decompressor would never need to pause the playback due to lack of information.

Should this estimate prove incorrect, or should problems be encountered during the streaming process due to network congestion, a fallback was provided. The animation stream

contained markers stating that it was safe to play the sequence until a given frame number. If this marker was reached during playback, and this marker was not equal to the final frame of the sequence, playback would pause until the next such marker had been retrieved and the 'safety frame' had been updated to some frame number beyond the current frame.

As objects could be constructed over time, the View engine had to be able to compensate for the dynamic state of its database. The simplest way to prevent mutual exclusion complications was to append incoming data to the existing database - in the case of mutual exclusion complications, one could always simply use the last known safe state of the database without worrying about touching data that was currently in flux. Each 3D object, defined as the class CVisualObject or a class inheriting its properties, was added to the database by first allocating the memory required for all of the model's components. The CVisualObject class contained markers for each of its properties defining how much of that particular data type had been appended to the model's database entry. For example, each model has a variable named `CurrentTotalVertices` that would contain the number of vertices that had already been added to the model. Every time a block of new vertices was appended to the model, the `CurrentTotalVertices` value would be updated to reflect the new vertex count. As these values would not be updated until after the entire block had been appended, one need not worry that the rendering routines would overrun the database, or attempt to operate on data values being actively modified. If the entire database entry for a model was completed, a flag `bCompleted` would be marked, indicating that the model could now be compiled to take advantage of extra capabilities available to the rendering engine.

The rendering engine found in the View adjusted its behavior for each model it attempted to render. It evaluated whether or not the model had been completely specified by the database. If a model had yet to be completely filled in, the rendering engine operated more carefully, rendering what had been made available in the database. If normal vectors had not been specified



(a) (b)

Fig. 4.2 Sample screens from the decompressor application. (a) awaiting a file to process (b) sequence playback paused

for the model, face normals were calculated on the fly based on the cross product of the face's edge vectors (producing a flat shading effect). However once an object had been fully described, if normal vectors had not been manually added to the model, normal vectors could be compiled quite accurately for the model, a normally calculation intensive operation. Vertex normal vectors were constructed by averaging the face normals for every face connected to each vertex.

The actual rendering performed by the View was performed using OpenGL version 1.1, as specified in the thesis design. Using OpenGL allowed for the rendering of the models, complete with the textures transmitted by the compression engine being mapped onto the proper polygons. As OpenGL includes an advanced lighting model very similar to the general Phong model described in the background, it was possible to reproduce lighting conditions in the original source sequences similar to the lighting produced in 3D Studio Max. Though lighting information was exported by the compressor, very little of the lighting portion of the project was completed due to lack of information regarding the way the lighting parameters of 3D Studio Max were

meant to be used. A shortage of information as to the internal workings of 3D Studio Max in regard to lighting as well as other factors made the implementation of the lighting portion of the decompressor no longer a priority. Nevertheless a framework for lighting support was implemented, and the decompressor would attempt to emulate the default two directional lights 3D Studio included in scenes that otherwise contained no lights.

When a user selected a file to decode, the decompressor connected to the given URL, and attempted to download the file. Assuming it succeeded in linking to the file, data began streaming over the channel. The first part of every file was always a simple header describing specific information required to display and synchronize the sequence playback (see Appendix B for further details). Along with information about timing was a 3 byte marker describing the target resolution of the sequence. Once this information had been received the decompressor would dynamically resize the application window so that the rendering window was equal in dimensions to the target resolution. The application prevented a user from selectively resizing the window, as the target resolution would determine the sampling resolution. If the sampling resolution was less than double the final playback resolution, the original scene would have been under sampled and this would result in missing polygons in the final scene.

Though it was easy to observe and visually confirm how rendering was performed in the decompressor, checking the behaviour of the decompression processes themselves was more difficult. To this end, diagnostic routines were added to the system to allow the tracking of the decompression and to allow one to observe the contents and order of the various opcodes in a given file. A logging window was available to the client that displayed a list of all previously processed commands for the current file, and listed them in the order in which they arrived. The diagnostic window allowed the quick determination of whether or not commands were being processed correctly and whether or not the LZSS algorithm was operating as desired.

Finally, in addition to all of the basic functionality of decompressing the animation sequence, an interface was added to allow users to control the sequence playback, consisting of play, pause, and stop buttons, as well as a slider bar that indicates timing. When an animation sequence began downloading, it signaled the View to begin animation playback. This signal disabled the play button, and enabled the pause and stop buttons. As playback occured, the slider bar was updated to indicate the current time index into the animation sequence. If the pause button was hit, animation playback ceased, but the timing location into the sequence was retained. Once the system was paused, the play button was enabled, and the pause and stop buttons were disabled. If the play button was hit, playback resumed at the current timing location, as shown by the slider bar. Should the stop button have been hit during playback, playback ceased, and the animation sequence was reset to the opening frame. Since playback was merely based on the current data in the animation database being built up, stopping or pausing the playback of the sequence did not affect the streaming of the sequence.

4.2 Compression Engine

The compression engine went through several iterations before finally arriving at its current state. The original idea for implementing the compression engine was to write it as a utility script utilizing the MaxScript scripting language included with 3D Studio Max Release 2.0. It was felt that MaxScript would provide us with the advantage of rapid development due to the simplicity of the language, while only sacrificing a minor amount of runtime speed due to the language being interpreted rather than compiled. Development began under MaxScript, but we slowly realized that the memory and processing overhead required by the compressor simply proved overwhelming to MaxScript. Attempting to compress any significantly complex scene

with the MaxScript engine would cause 3D Studio Max to freeze up, become non-responsive and cease to function correctly. It seemed that the color scanning used to determine polygon visibility required increasingly larger and larger amounts of memory, as colors would be constantly appended to an existing color list.

Thus, development had to be restarted utilizing the 3D Studio Max plug-in software developer's kit (SDK). One advantage of 3D Studio Max's design is that the entire application is build around a framework of plug-in functions. These plug-ins are created as dynamic link libraries (DLLs) which are loaded at run-time by 3D Studio Max, and allow for the extension of the application's functionality through third party additions. Almost every built in feature, from the graphical user interface to the rendering engines themselves, can be supplemented or replaced by user designed plug-ins.

This allows one to take advantage of the vast array of features and technologies previously developed for 3D Studio Max, as well as native support for its content. Using an existing application like 3D Studio Max for a base also prevented a great deal of "reinventing the wheel" when building the compression engine, as code for much of the functionality required for the for manipulation of data types and geometry already existed as library functions. Seeing as the compression engine needed to take an existing sequence created in 3D Studio Max and convert it into a streamable file, a file exporter plug-in was developed.

The exporter plugin, when actived by a user through the File menu in 3D Studio Max, would provide a dialog box through which users could select a series of options as to how the scene was to be compressed. Options accessible through the dialog included camera selection, material quality, target resolution, animation subset and playback speed, how many bits should be used to encode vertex components, enabling simple lighting, and the inclusion of normals (see Figure 4.3).

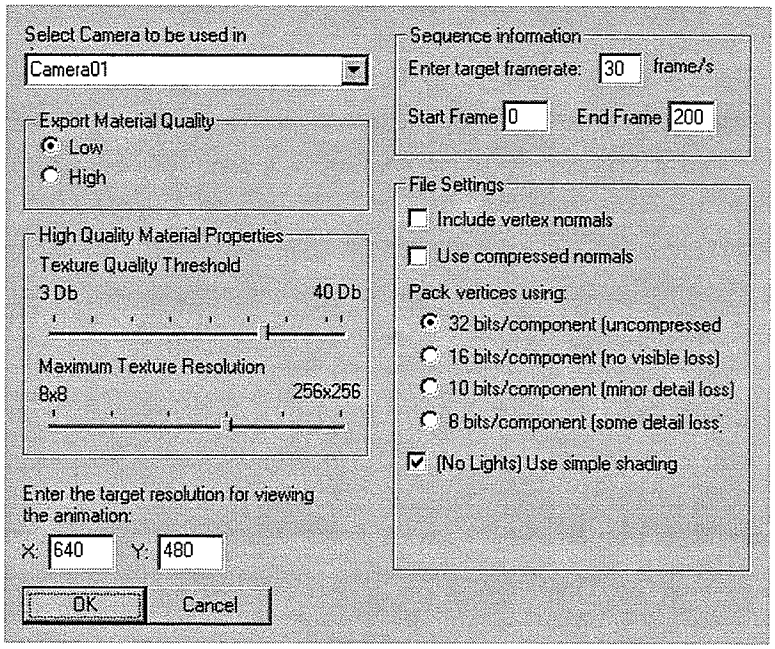


Fig. 4.3 Compression engine user options dialog box

Due to difficulties in managing changes in the active camera, only a single camera could be exported with a specific scene. Therefore, all scenes had to contain at least one camera for export, and the camera to be used had to be selected in the available pull-down menu. This camera's settings would be used during the visibility test to determine what ended up in the final output. It should be noted that the compressor did not support the use of controllers for camera attributes such as field of view or clipping planes. While field of view values set in 3D Studio Max were retained and transmitted by the compression engine, as it was unknown how 3D Studio Max utilizes the field of view value internally to determine the view frustum, frustum values are not recreated correctly by the decompressor. It is currently recommended that camera attributes and the scene's aperture width be left in their default settings to ensure correct rendering by the decompressor.

Due to the switch from Maxscript to the native 3D Studio Max SDK, we had to start over and considered reevaluating our approach to the problem of determining polygon visibility. Our

original approach called for using the built-in rendering engine of 3D Studio Max, as it was felt that it was already highly optimized, accurate and would simplify our task of rendering the polygons. As well, if the application were to be written in MaxScript, this would prove the most practical approach as access to the rendering engine's functionality is built in, and the design of a custom renderer to run under MaxScript would have proven quite slow and cumbersome.

Yet, the restrictions of MaxScript were no longer applicable, as the language had been abandoned in favor of C/C++. Therefore we considered turning towards using OpenGL as a rendering engine to accelerate the performance of the visibility determination. Implementation of the new renderer began, and we constructed a system that used OpenGL to render the polygons using colors to mark the polygons, as specified in the design. Unfortunately, it was discovered that OpenGL does not guarantee that a color specified for a polygon face will be the color rendered by the video hardware. On some video card drivers, colors would undergo gamma correction, or other forms of color shift which would make our color scanning operations useless, as they would not find the correct colors.

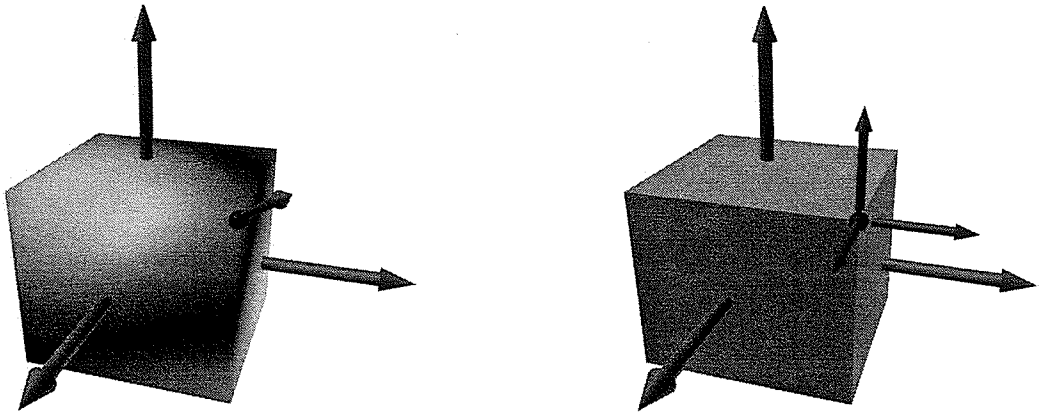
Consequently we abandoned the use of OpenGL for rendering the visibility determination information, and instead, returned to our original approach of using the built-in rendering engine included with 3D Studio Max. In order to render each frame required for visibility determination, for every frame, all objects in the scene were sampled. From these sampled objects, a new object was constructed duplicating all the polygonal data of the original objects except that all object faces were colored to designate which face number they were, as well as which original object they belonged to. Face colors were set through the 3D Studio Max vertex color material texture. To scan for colors, this new duplicate object would be constructed for a given frame, all original objects would be hidden and objects would be disabled, and then the frame would be rendered to a bitmap. This bitmap would then be scanned pixel by pixel for color values, as the color values

would be used as lookup values to determine which faces and objects in the scene actually appeared in the final bitmap.

It had originally been planned to develop the color scanning system using 24 bit color (the red, green, and blue components, 8 bits each, appended together) to establish the lookup value - the high 12 bits would refer to the object number while the lower 12 bits would refer to the face number. These values would be added with 1 so that the color scanner could immediately ignore any colors returned with value of 0. While at first glance, using 24 bits to define these ranges seemed more than adequate, as 2^{12} (4096) appears to be a generous limitation given the constraints of the real time rendering engine. However early testing revealed that this range could prove inadequate in cases where the original objects being used in the scene were extremely complex. While only small portions of these objects might appear in a scene, the object beyond the view range of the virtual camera could still exceed the limit of 4096 triangle faces. As all of the object's faces must be enumerated to correctly evaluate visibility, objects whose polygon count exceeded the limit would cause problems to the system. The solution eventually selected for this problem was to switch the 3D Studio Max rendering engine from rendering in 32 bit color, where 24 bits were used for color and the remaining 8 bits were used for an alpha channel, to rendering in 64 bit color. As the alpha channel proved useless to the color scanner, the system now used a 48 bit range, which provided us with 24 bits for both objects and faces, and solved the problem of inadequate ranges.

There was one remaining issue affected by the inadequate range problem for color scanning. One shortcoming in the initial design of the model exporter was that it never accounted for the use of smoothing groups in the definition of 3D objects. Smoothing groups are used in 3D Studio Max to define continuous surfaces that share vertices and surface properties within a larger model. These smoothing groups proved very important to the accurate representation of models,

as normal vectors for vertices are defined by the faces that share the vertex, but only those faces within the given smoothing group. Accordingly each vertex could have multiple normal vectors, one for each smoothing group that shares the vertex. A simple example of the need for smoothing groups would be the case of attempting to model a cube (see Fig. 4.4). To correctly model a cube, one requires three smoothing groups, one for each set of opposing sides. The vertex normals of each smoothing group will be perpendicular to the particular side of the cube that contains them. Thus, if one considers the corner vertices of the cube, each one will have three different normal vectors corresponding to the three different smoothing groups that intersect with it. Conversely, if we did not separate the cube into smoothing groups, each corner vertex's normal vector would point along a diagonal away from all of the cube sides. While not immediately apparent, the replacement of the three vertices with the single average value of the three (which the single vertex represents) will cause obvious visual errors in how lighting effects appear on the cube. Instead of appearing as several sides meeting at right angles to one another, the cube will appear smoothed and thus the various sides of the cube will not appear distinct from



(a) (b)

Fig. 4.4 A comparison between using and not using smoothing groups to construct models with separate distinct surfaces. Longer arrows show the face normals for each side of the cube, while the shorter arrows represent the resulting vertex normal(s) for the given vertex, represented by a blue dot. (a) incorrect lighting without smoothing groups (b) correct lighting with smoothing groups.

one another.

As smoothing groups were initially overlooked in the system design, they had to be added to the capabilities of the system after the fact. While considering several alternatives, it was decided that any solution chosen would have to require a minimal number of changes to the existing systems, induce a minimal information overhead to the system to specify the smoothing groups, and require a minimal processing overhead of the decompressor. The final selection made was to account for the smoothing groups by taking each original object and dividing it into several new objects corresponding to each smoothing group. As 3D Studio Max allows for up to 32 smoothing groups per model, the capabilities of the color scanner had to account for 32 times more objects. With the original 12 bit range for model enumeration, this was a problem, as the effective upper range was reduced by 5 bits to 2^7, or only 128 total models, a number much too low for practical use. The switch from the 32 bit rendering engine to the 64 bit rendering engine for use in color scanning, as described earlier, solved this problem.

The solution of splitting the original objects by smoothing groups proved very effective, as it required no additional support by the decompression engine. The splitting process consisted of taking the original model as described by 3D Studio Max, and replacing it with new objects, all belonging to a dummy parent object. This dummy parent object had no properties other than to take on the original position and pivot of the source object, and to duplicate its motion controllers. All the new objects that represented the source smoothing groups were child objects of the dummy node, and duplicated its materials and their relative polygonal information. Thus, all the smoothing groups operated as though they were components of the original object seamlessly. As a result no new streaming instructions were created to reflect the new support for smoothing groups, and no new functionality had to be added to the decompression engine to support smoothing groups.

At first glance it would seem that splitting an original object into up to 32 new objects would induce a great deal of overhead. Yet, in most cases the overhead induced by the object splitting was rather minimal, and in some cases, the new objects could require less storage than the original. The reason for the additional overhead being so small is that very little duplicate information would need to be reproduced between the various objects. The only information that would always require duplication would be vertices shared by several smoothing groups, as each new sub-object would require its own reference to the coordinates and possibly the colors of the shared vertices. Face information, material information, and texture coordinates would naturally be unique to each face regardless of the subdivision of the original object. Thus these factors would not encumber the stream with additional information. As one would have to code the smoothing group information in any case, and would have to do it in such a way as to minimize decompression processing overhead, the expense of reproducing a small number of the source object's vertices was not a great one.

Another significant improvement to the system realized through the inclusion of support for smoothing groups was that materials should be separate and distinct from the objects that utilize them. The original design called for each object to possibly have a texture map, and this texture map to simply be part of that object - a design oversight. The design did not account for more than one object in a scene having the same material. Though this flaw was not obvious at first, it became so after the addition of support for smoothing groups, as under the original design each portion of the subdivided source object would require its own texture map of the same original material. This would have introduced a great deal of unnecessary redundancy. Instead, it was decided that each material would be granted an identification tag. When objects were being enumerated, each unique material was assigned one of these tags. Enumerated objects would then reference a material tag, rather than the material itself. When a new object was being added

to the stream, that object's material tag would be checked to determine if the material had previously been added to the stream. If it had not, a material identification number would be assigned internally, the representative texture map would be constructed, and its lowest level mipmap would be transmitted into the stream immediately. When other higher level mipmaps were to be transmitted, they would be identified in the stream by the decompressor via the compression engine's internal reference number. As these reference numbers were created by assigning the previous total number of textures transmitted into the stream as a material identifier, reference numbers were implied to the decompression engine simply through the order in which the textures arrived. Accordingly, material identification codes were transmitted much in the same manner as object identification codes were.

While material textures would be sent incrementally, ideally one would want to transmit the fewest MIP map levels possible. Two different techniques were used attempt to reduce the texture detail exported into the stream, in addition to the two-dimensional screen aligned bounding box test. The simplest of the two was allowing the user to set the texture detail. The option dialog had two sliders - a texture quality slider and a maximum MIP map slider. The texture quality slider was used to let the user set the minimum allowable PSNR. When texture MIP map levels were transmitted, all coefficients were included. Thus, to allow the texture to compress more easily, low energy coefficients were zeroed out to make the textures more compressible. The energy threshold was determined by iteratively approximating the threshold against the texture coefficients and then comparing the resulting texture map with the original texture. If the PSNR between the reconstructed texture and the original was above a user-set minimum allowable PSNR the energy threshold would be incremented, whereas if it was below, the energy threshold would be decremented. Once the energy threshold had been determined, any coefficients in this texture whose absolute value was below the threshold would be converted to

zero. By the user lowering the allowable PSNR, more coefficients would be turned into zeros, making the texture more compressible at the cost of texture detail.

The second technique to reduce the texture detail transmitted was to apply the reconstructed maximum needed MIP map with its next lower level, as described in the design section of this thesis. When comparing the two reconstructed levels, PSNR was determined and was compared against the user-set PSNR threshold. If the comparative PSNR was above the threshold, the maximum MIP map resolution was reduced by a level and the procedure was repeated until a failure was encountered. Both techniques were automatically applied and both were affected by the user setting of the PSNR threshold.

In order to transmit textures, compound materials would need to be combined into a single final texture image. This was accomplished by using the internal 3D Studio Max production quality rendering engine and rendering the objects' material projected onto a fully lit square. This square was scaled to entirely fill the rendering space, such that the output from the render would be an exact 256x256 version of the compound texture. As the texture was mapped using basic two-dimensional mapping, some materials (for example, procedural or noise based textures used to simulate marbling and similar structures) would not show up correctly. For such problematic materials it was recommended that users use the built-in 3D Studio Max menu command allowing them to render the material to a file, and use the resulting bitmap as the new material.

In addition to transmitting information about how objects were meant to appear the compression engine also had to provide the trajectories and paths of these objects. Motion transmitted by the compression engine was transmitted using keyframing information provided internally by 3D Studio Max to allow for accurate reconstruction by the decompression engine during playback. Failure to accurately reproduce exact position and orientation information for

objects by the decompression engine was a much greater problem for this work than under other animation systems. This was due to the polygon sampling done to determine visibility. While in other animation systems (those evaluated in the background phase of the thesis) objects were always fully defined such that they could be viewed from any position or orientation, though possibly with little detail, models exported by this system could not. The information sent into the stream was entirely view dependent, and required that the object's geometry be exactly recreated in space. If orientation or position was incorrectly calculated, it would result in obvious visual errors, such as portions of objects not appearing, or some objects not appearing at all.

Motion paths exported from 3D Studio Max had to be exactly reconstructed using the keyframing parameters provided by 3D Studio. While this at first glance might appear simple, as 3D Studio seemed to provide keyframing information to interpolate values, the values provided in the case of TCB and Bézier curves could not simply be plugged into the standard equations to interpolate the correct curves. The values provided had to be processed by the either of the two engines before they could be used in the equations described in the background section. Bézier position values needed to be scaled to properly interpolate curves, as the tangent values provided by 3D Studio Max were not the desired final values and would result in nearly straight-line paths. The scaling factor required was to multiply the provided tangent values by the total number of ticks (ticks were a measure of the time required per frame by 3D Studio Max; by default there are 160 ticks per frame) by the number of frames between the keyframes. Bézier tangents were preprocessed in the compressor before being exported. The TCB curves did not include their tangent values, and instead required that they be calculated by the decompressor upon the keyframe having been received.

4.3 Summary

Implementation of this thesis required the development of two applications, a compressor and a decompressor. After deciding that constructing the compressor using MaxScript would prove impractical, the compressor was written as a 3D Studio Max Release 2.0 plug-in. The decompressor was written as a standard MFC application utilizing OpenGL to perform the 3D rendering and TCP/IP to stream remote sequences.

As is common with application development, some problems were overlooked in the design phase and had to be compensated for in the final implementation. Smoothing groups, which allow objects to be broken up into distinctly separate surfaces were not originally accounted for in the design phase and had to be added to the system. In addition, a flexible material referencing system had also been left out of the design and was created during implementation.

Due to the lack of documentation as to the inner workings of 3D Studio Max, several portions of the implementation, such as curve reproduction, were extremely difficult. The lack of access to this information made the development of the lighting portion of the decompressor impossible and thus it was never fully implemented in this thesis.

CHAPTER V
EXPERIMENTATION

5.1 Test Suite

One complication that arose in this thesis was that there was no preexisting test suite of standard animation sequences to test our system against. No preexisting benchmarks existed for the compression of 3D animation sequences. Thus, we had to construct a series of simple tests to demonstrate the system using 3D Studio Max. Objects used in these sequences were of static topology and varying complexity. Due to the great length of time required to compress sequences, only a limited number of tests were conducted.

While this solved the immediate problem of the lack of a preexisting test suite, a further issue existed. How were we to measure the performance of our animation system? As the new animation system is meant to possibly replace the need for traditional video compression techniques, the best solution might be to compare final file size against a standard video codec. Nonetheless, this does not account for natural differences and inconsistencies between the two techniques. Another approach might be to compare the size of the original 3D Studio Max file with the compiled streamable file generated by the compression engine. Again, while providing some insight into the relative performance of the compression system, equivalent systems are not being compared. For most complex animation sequences, the compressed streamable output should generally contain less data - but bitmaps accessed by Max materials are generally not stored with the geometry, and thus must be accounted for separately.

One final obvious solution remains: to compare output produced by our compression system to other transmittable geometry systems. Unfortunately, with the exception of VRML,

none of the commercial techniques for streaming animation provide free authoring tools. It is possible to generate progressive mesh files using free tools included with the DirectX SDK provided by Microsoft, but they do not account for animation, and merely focus on object geometry. Thus we are limited to comparing against VRML 2.0 files. To provide a slightly more fair comparison, file sizes are compared against a hybrid LZ77 compressed form (using GNU `GZip`) of the VRML file. While none of the comparative systems provide an exact relative measure of the capabilities of our system, or properly equate how performance is affected by the progressive nature of our system, they do provide a general idea of overall compression performance.

5.2 Test Cases

When applying our compression system, each animation was compressed using the LZSS technique. Measurements of file length were taken both before and after the LZSS technique had been applied to demonstrate the capabilities of our LZSS implementation. Shannon's entropy of the 0^{th} and 1^{st} order were also taken of the precompressed files to demonstrate potential compressibility and to again show the limitations of our use of LZSS.

Five basic test cases were constructed in 3D Studio Max, each demonstrating basic capabilities of the compression engine and how it handles different scenarios. Tests were designed to demonstrate different qualities that animated scenes might require. Included were scenes in which different portions of models got revealed over time, scenes in which new models were added to the visible scene over time, scenes in which models were connected through hierarchies and inherited each others' transformations, and scenes utilizing both simple and complex materials. The car and airplane models used were constructed by Viewpoint Labs.

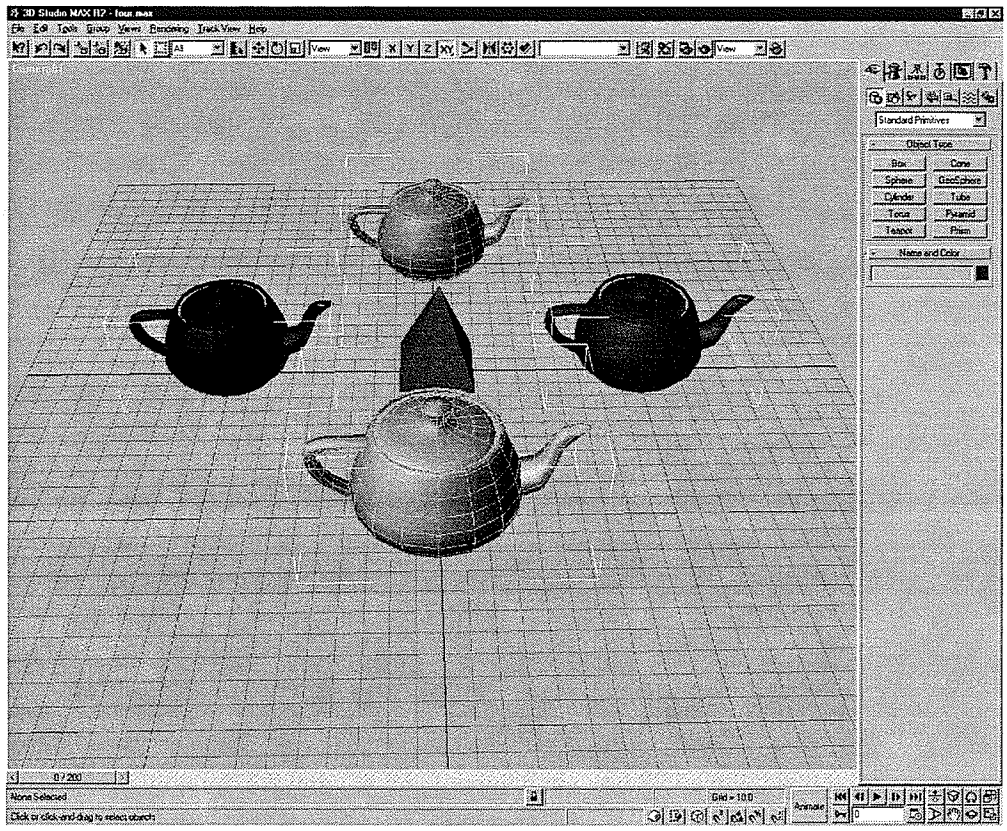


Fig. 5.1 Teapot scene in 3D Studio Max R2

5.2.1 Four Teapots

This sequence consists of four teapots flying and spinning around a static pyramid in a rough counter clockwise circle. At the same time, a camera and its target spin around the scene in a clockwise direction, giving the impression of greater speed. This scenario demonstrates objects using Bézier position tracking and TCB rotation tracking, as well as the effect of solid colors rather than materials. Each teapot contains 1024 faces and 530 vertices, while the pyramid contains 8 faces and 6 vertices. The sequence is 200 frames long, at 30 frames per second.

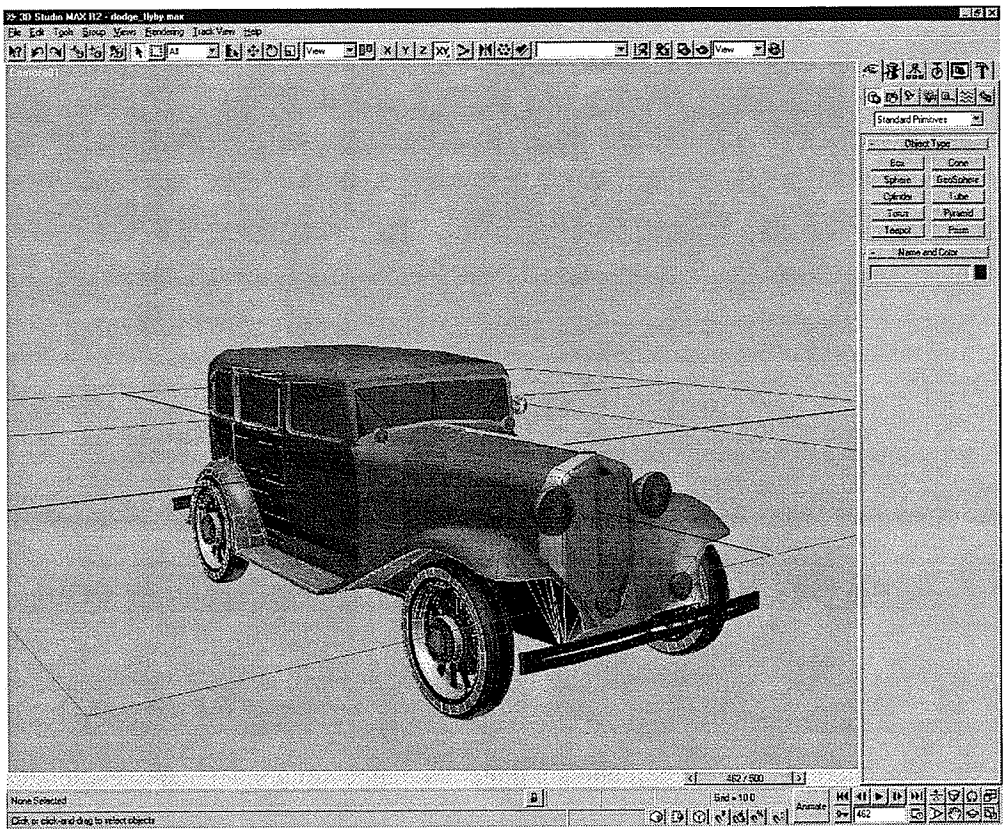


Fig. 5.2 Dodge scene in 3D Studio Max R2

5.2.2 Dodge Flyby

This sequence displays a slow fly-around of an automobile. The car remains still in the scene, while the camera flies around it, slowly showing off the entire car. This scenario provides a direct demonstration of the progressive addition of scene information. Additionally, the scene demonstrates the system's ability to handle non-bitmap based materials as all of the materials used in constructing the car consist of simple internal materials. The automobile originally consisted of 16642 faces and 10644 vertices. The sequence is 500 frames long, at 30 frames per second.

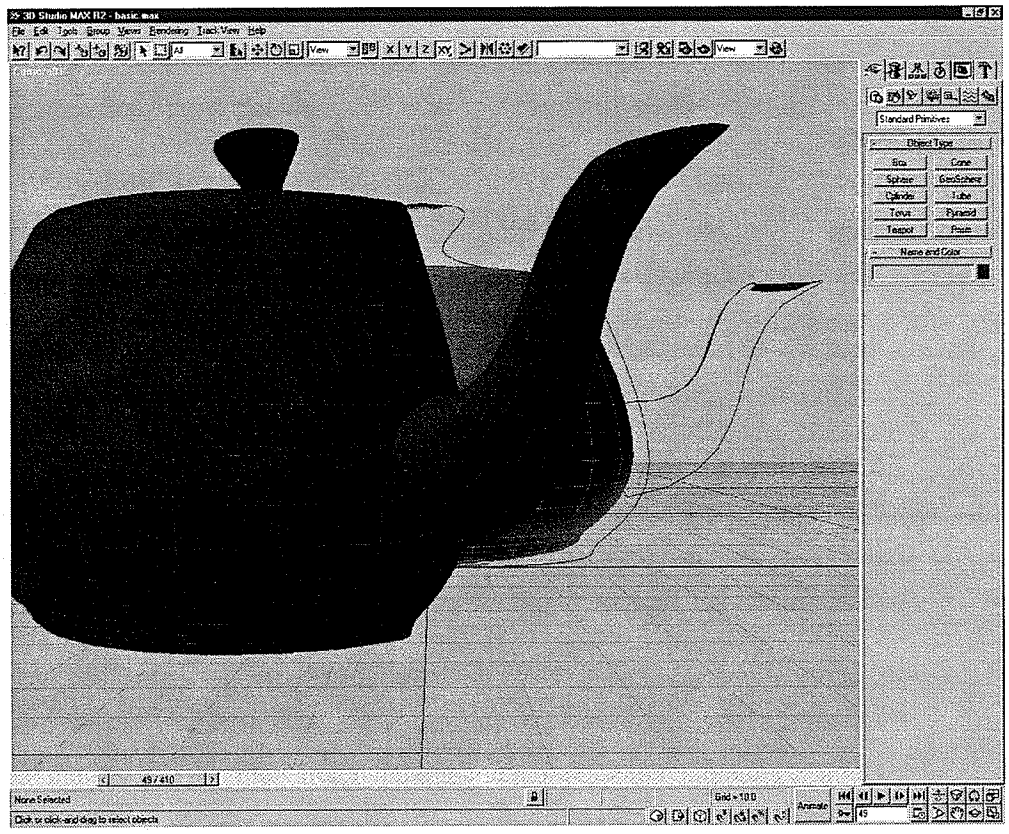


Fig. 5.3 Teapot cutaway scene in 3D Studio Max R2

5.2.3 Teapot Cutaway

This sequence displays a teapot, with smaller teapot bodies inside. The initial teapot is divided in half along its length, and opens along the seam to reveal another smaller teapot body inside. This body is split and opened, again revealing another teapot body inside. This is split and opened revealing a final teapot inside. The scene originally consisted of 7789 faces and 4350 vertices. The sequence is 410 frames long, running at 30 frames per second. The texture maps used by the two materials have the resolutions of 320x200 and 320x240, where one was originally JPEG compressed, while the other was GIF compressed.

This animation sequence demonstrates several features, including the use of texture maps, and object cutaways. Object cutaways feature objects being opened up to further new objects

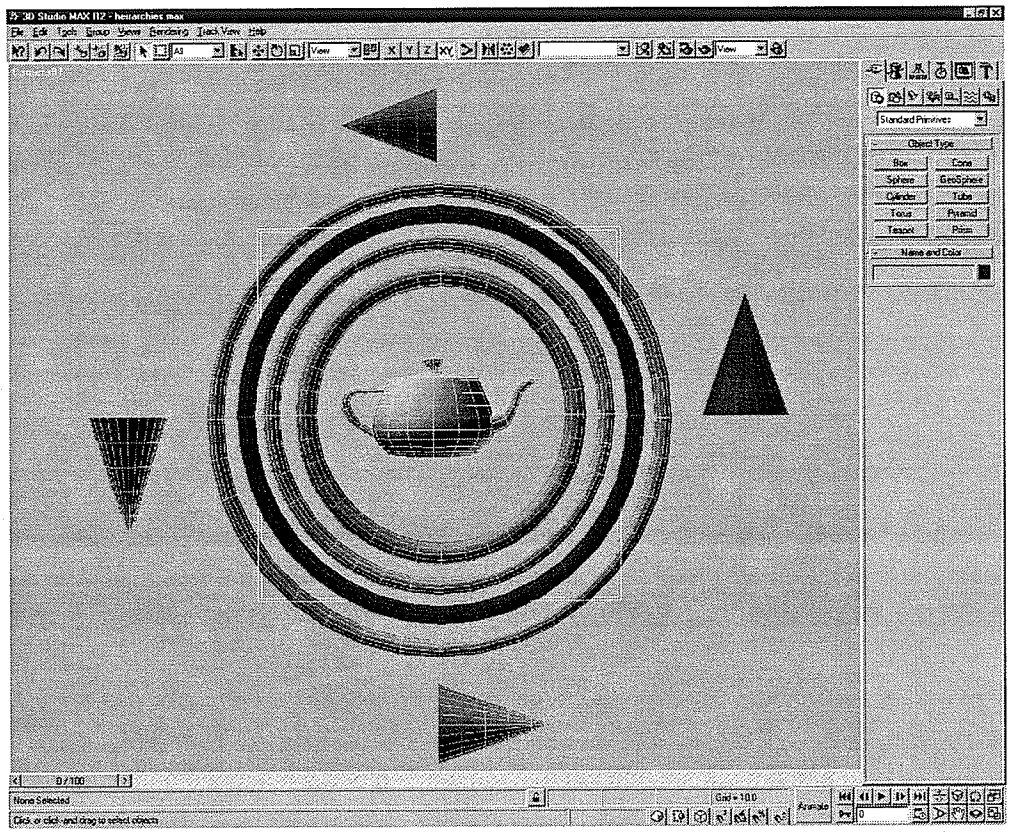


Fig. 5.4 Tori hierarchy scene in 3D Studio Max R2

inside, demonstrating the progressive addition of objects and information to the sequence. Cutaways are often helpful in demonstrating the operation of machinery and are thus commonly used to explain the operation of mechanical objects in teaching and training.

5.2.4 Tori Hierarchy

This sequence displays a series of tori all with a common centroid. In their center is a teapot, and around them is a group of cones. The tori are arranged in a hierarchy, where the outermost ring is the parent to its inner neighbor, which in turn is the parent of its inner ring. This third ring is the parent of innermost ring. Though each of the rings has its own rotation parameters, its absolute orientation is a function of the orientation of its parents. The teapot remains static and is provided as a point of reference. The cones share a common invisible parent

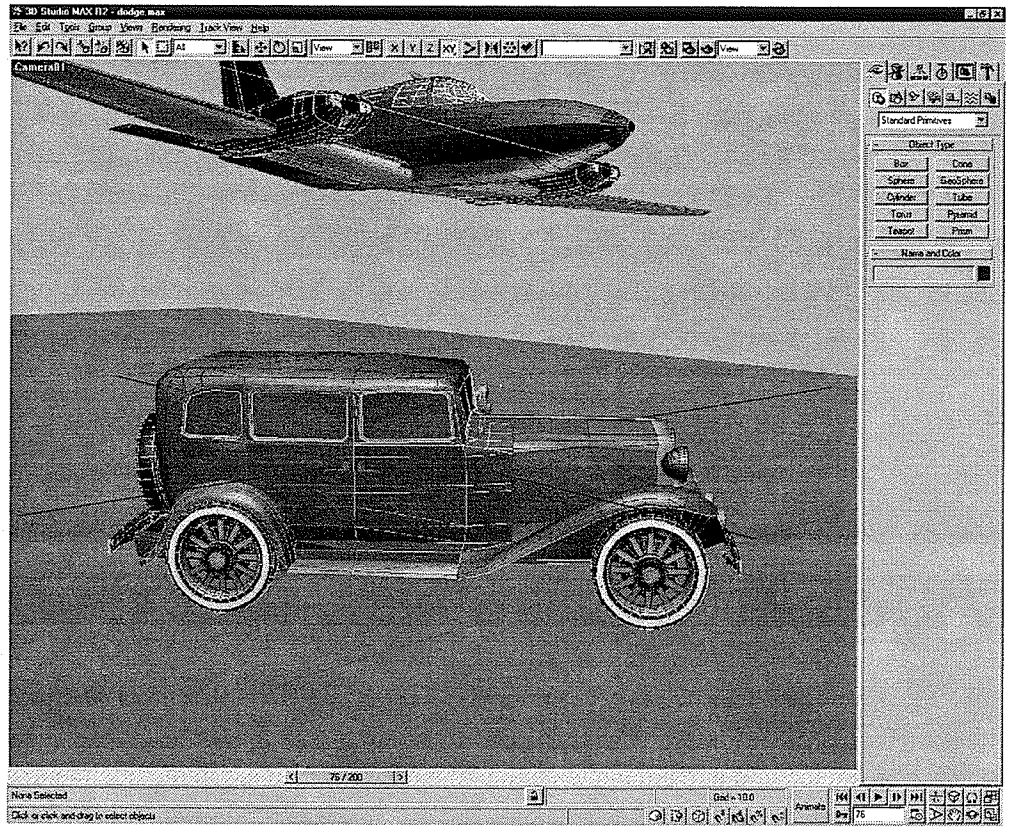


Fig. 5.5 Plane & Dodge scene in 3D Studio Max R2

node, and it is the parent rather that the cones themselves which rotate about the tori. The path taken by the cones is a result of the revolution of this invisible parent node.

The tori hierarchy demonstrates the inheritance of transformations from one object to another. The scene originally consisted of 5632 faces and 2842 vertices. The sequence is 100 frames long, at 30 frames per second.

5.2.5 Plane & Dodge

This sequence consists of the familiar automobile model and an airplane. The car is parked on a gray slab. The plane flies by the car and the camera tracks the plane as it passes the car. As the camera pans, the driver's side and underside of the Dodge never come into view. The

plane & automobile scene demonstrates the effectiveness of the removal of unseen polygons from a sequence. Materials on both the car and plane are simple colored materials. The scene originally consisted of 21631 faces and 14605 vertices. The sequence is 200 frames long, at 30 frames per second.

5.3 Summary

Due to the lack of an existing test suite, five simple scenes, displaying attributes common to training and teaching demonstrations were constructed. These scenes contain a wide range of qualities, including variations in material detail, and different degrees of geometric complexity. They also demonstrate several common examples of the types of scenarios often seen in training sequences, such as cutaways or flybys.

CHAPTER VI
RESULTS AND DISCUSSION

6.1 Experimental Suite

Results from the various experiments conducted are compiled by experiment, and compared against the results from exporting the same scenes to the standard 3DS (originated with 3D Studio) file format, to VRML which has been compressed using GNU `GZip`, and rendered by 3D Studio and compressed using MPEG-1. 3D Studio Max no longer directly uses the 3DS format instead supporting a more flexible 'MAX' format. Still, the older 3DS format has been around long enough to have become an industry standard, and now virtually every major modeling package supports the importing or exporting of 3DS format scene descriptions.

For reference purposes, our new format is labeled as the PRT format, where PRT stands for Progressive Real-time Transmission. Sequences are exported using the following settings - high quality textures, a target resolution of 640x480, no normal vectors, and default lighting. Textures were limited to a maximum resolution of 256x256, with a texture threshold setting of 30 Db. Entropy values provided are calculated based on the data sources before any form of additional compression has been applied. In the case of PRT files, entropy calculations were taken before the LZSS algorithm had been applied.

For comparison, VRML files were exported from 3D Studio Max using the built in exporter plug-in with the following settings: four digits of floating point precision, indented formatting, no normals are included, primitives are to be used, the polygon type was set to triangle, and the transformation controller sampling rate was set to 10 Hz. It can be seen that for comparative purposes, neither of the two transmittable 3D formats (VRML and PRT) were set to

include normal vectors.

MPEG-1 compressed video is compressed based on frames rendered by 3D Studio Max to a resolution of 320x240, at a frame rate of 29.97 frames per second, and at a bitrate of 1.7M bits per second. Also provided for comparison is MPEG-1 compressed video from frames rendered to 640x480 with a bitrate of 6.8M bits per second.

Finally, limited testing was performed over a 28.8 kbps modem to confirm that the new compression system does in fact work and allows the playback of sequences before the entire file has been transferred. Typical connection rates tested were in the range of 28800 to 24000.

6.2 Experimental Results

From the experimental results (Tables 6.1 through 6.5) it can be seen that in all cases, for the reasonably simple scenes being compressed, the polygonal formats clearly outperform the

Table 6.1 Comparative results on the teapot example.

Output	*File Length (bytes)*	*Vertex Error Rate (%)*	*File Length before Compression (bytes)*	*Compression Ratio (vs. 3DS)*	*Entropy 0^{th} Order (bits/Byte)*	*Entropy 1^{st} Order (bits/Byte)*
MPEG-1 (640x480)	5,795,841	NA	185,248,768	NA	1.078	0.804
MPEG-1 (320x240)	1,445,889	NA	46,317,568	NA	1.106	0.836
3D Studio (3DS)	91,420	0.00%	NA	1:1	5.353	4.797
VRML (WRL.GZ)	41,560	0.00%	166,450	2.2:1	3.641	3.135
PRT - 8 bits per vertex component	25,011	0.377703%	28,884	3.66:1	6.648	5.730
PRT - 10 bits per vertex component	27,219	0.098442%	30,525	3.36:1	6.683	5.927
PRT - 16 bits per vertex component	28,544	0.001459%	35,277	3.20:1	6.907	5.769
PRT - 32 bits per vertex component	42,511	0.00%	47,823	2.15:1	6.998	6.036

Table 6.2 Comparative results on the dodge-flyby example.

Output	*File Length (bytes)*	*Vertex Error Rate (%)*	*File Length before Compression (bytes)*	*Compression Ratio (vs. 3DS)*	*Entropy 0^{th} Order (bits/Byte)*	*Entropy 1^{st} Order (bits/Byte)*
MPEG-1 (640x480)	14,401,537	NA	461,735,936	NA	3.240	2.188
MPEG-1 (320x240)	3,608,577	NA	115,444,736	NA	3.186	2.143
3D Studio (3DS)	368,661	0.00%	NA	1:1	5.953	5.610
VRML (WRL.GZ)	149,728	0.00%	667,326	2.46:1	3.601	3.108
PRT - 8 bits per vertex component	126,943	0.208319%	140,661	2.90:1	7.330	6.543
PRT - 10 bits per vertex component	137,444	0.051496%	148,961	2.68:1	7.407	6.738
PRT - 16 bits per vertex component	160,169	0.000821%	172,764	2.30:1	7.482	6.869
PRT - 32 bits per vertex component	213,748	0.00%	235,218	1.72:1	7.531	6.779

video compressor. Video compressors, such as MPEG-1, perform best when large portions of a scene remain static for long periods of time. Unfortunately, in flyby or cutaway scenarios, large quantities of the visible scene tend to change shape as new portions of objects are revealed and brought into view. While typically MPEG compressors do quite well, achieving a compression rate of around 32:1 of the original rendered video frames, they still under perform all provided examples of scene description compression. Though each of the provided examples is short in running length, it should be noted that none of the scene description based techniques are dependent on running time to determine final file length. Instead, video compression codecs are subject to the scene description running time, as they attempt to compress individual keyframes as well as inter-keyframe information. Increasing the running length requires the increasing of either the number of keyframes, the amount of inter-keyframe information, or both. Thus it can be seen that the size of video-compressed sequences is a direct function of their resolution (bitrate) and

Table 6.3 Comparative results on the teapot cutaway example.
* Additional storage required for texture maps utilized by scene materials

Output	*File Length (bytes)*	*Vertex Error Rate (%)*	*File Length before Compression (bytes)*	*Compression Ratio (vs. 3DS)*	*Entropy 0^{th} Order (bits/Byte)*	*Entropy 1^{st} Order (bits/Byte)*
MPEG-1 (640x480)	11,919,361	NA	378,789,888	NA	3.466	2.789
MPEG-1 (320x240)	2,959,361	NA	94,706,688	NA	3.468	2.799
3D Studio (3DS)	210,465 + 131,531*	0.00%	NA	1:1	5.688	4.984
VRML (WRL.GZ)	84,681 + 131,531*	0.00%	370,434 + 131,531*	1.58:1	3.513	3.015
PRT - 8 bits per vertex component	100,707	0.212224%	104,051	3.40:1	7.035	6.231
PRT - 10 bits per vertex component	104,003	0.053415%	106,889	3.29:1	7.089	6.335
PRT - 16 bits per vertex component	109,469	0.000782%	115,151	3.12:1	7.167	6.320
PRT - 32 bits per vertex component	128,709	0.00%	136,991	2.66:1	7.195	6.326

length.

Though it can be seen from the various tables that at 10 bits per vertex component - the ideal bit coding rate for vertices - we arrive at smaller files than those of compressed VRML, our scheme does not appear to always perform a great deal better. Of the five experiments performed, the file size using our scheme is on average 66% of the size of the equivalent compressed VRML file.

While the PRT files are smaller, the lack of being dramatically smaller in the average case can be explained through one chief factor, the LZSS compressor we employ does not do a very good job compressing the signal when compared with more processor intensive techniques based around LZ77 such as Zip, GZip, Arj, or even the LZSS technique using the deletion heuristic. These other techniques provide much deeper search heuristics to identify redundancy, and make

Table 6.4 Comparative results on the tori hierarchy example.

Output	*File Length (bytes)*	*Vertex Error Rate (%)*	*File Length before Compression (bytes)*	*Compression Ratio (vs. 3DS)*	*Entropy 0th Order (bits/Byte)*	*Entropy 1st Order (bits/Byte)*
MPEG-1 (640x480)	2,914,305	NA	93,086,208	NA	1.856	1.439
MPEG-1 (320x240)	731,137	NA	23,275,008	NA	1.857	1.418
3D Studio (3DS)	117,091	0.00%	NA	1:1	5.520	4.903
VRML (WRL.GZ)	37,370	0.00%	180,286	3.13:1	3.373	2.878
PRT - 8 bits per vertex component	25,932	0.314385%	28,305	4.52:1	6.856	5.875
PRT - 10 bits per vertex component	27,843	0.082481%	30,140	4.21:1	6.999	6.059
PRT - 16 bits per vertex component	30,463	0.001162%	35,478	3.84:1	7.009	5.858
PRT - 32 bits per vertex component	44,075	0.00%	49,320	2.66:1	7.175	6.143

use of more complex structures to facilitate the further compression of the position/length pairs. The great complexity of the data stream is primarily due to the data packing techniques used to reduce the storage requirements of various data types. As the entropy values demonstrate, a very large amount of natural redundancy in the information has been removed in order to shrink the bandwidth requirements so only a limited amount of additional compression could be expected.

Yet, while entropy values are high, they still provide some room in which to further compress the data. This is demonstrated by the 0^{th} and 1^{st} order entropies of the PRT files before compression has been applied. Consider for example the tori hierarchy experiment - the average zero order entropy value is 7.01, and the average first order entropy value is 5.98. This implies that a compression system should be able to achieve at least around a 12% reduction in overall file size (only accounting for the zero order redundancy). Yet, for the entire set of vertex bit coding rates, the LZSS compressor only achieves a mere 10.2% average compression rate.

Table 6.5 Comparative results on the Plane & Dodge example.

Output	*File Length (bytes)*	*Vertex Error Rate (%)*	*File Length before Compression (bytes)*	*Compression Ratio (vs. 3DS)*	*Entropy 0th Order (bits/Byte)*	*Entropy 1st Order (bits/Byte)*
MPEG-1 (640x480)	5,761,025	NA	185,248,768	NA	3.053	1.878
MPEG-1 (320x240)	1,449,985	NA	46,317,568	NA	3.053	1.906
3D Studio (3DS)	620,206	0.00%	NA	1:1	6.419	6.154
VRML (WRL.GZ)	206,924	0.00%	948,548	3.0:1	3.528	3.036
PRT - 8 bits per vertex component	101,264	0.193178%	118,327	6.12:1	7.236	6.361
PRT - 10 bits per vertex component	110,658	0.047624%	124,345	5.60:1	7.319	6.564
PRT - 16 bits per vertex component	131,333	0.000759%	145,395	4.72:1	7.388	6.737
PRT - 32 bits per vertex component	174,087	0.00%	198,225	3.56:1	7.476	6.616

Another simple test to determine the performance of a basic compression algorithm such as LZSS is to examine the Shannon's entropy of the file after compression has been performed. If the compressor has done a good job, the final entropy should be 8 bits/byte - that is to say the file's statistics should resemble those of white noise. Instead, the average 0^{th} order Shannon's entropy of all the PRT files constructed was 7.89 and the average 1^{st} order Shannon's entropy was 7.37. Though both numbers are high, they still suggest that there is some room for improvement. In contrast, the GZip compressed VRML files have an average 0^{th} order entropy of 7.98 and average 1^{st} order entropy of 7.47 - numbers that imply that a higher amount of correlation has been removed from the VRML files. The greater performance of GZip over our LZSS implementation can be demonstrated by compressing the PRT files (before LZSS has been applied) using GZip. On average the Gzip compressed files are 85.7% smaller than the equivalent LZSS compressed files. Together with the earlier example of the tori hierarchy, it is implied not

only that the performance of the LZSS compressor is underwhelming, but that there exists room to improve by using more effective techniques.

Still, for our purposes, greater rates of compression must be sacrificed to keep the decompression overhead at a minimum. As VRML is entirely downloaded before it is decompressed and parsed, it can allow for the greater decompression overhead required by GZip as it does not impact performance and instead merely causes a minor delay before the start of playback. In our scheme, decompression of the stream must be interleaved with the actual processing and rendering of the scene itself. As the system was required to operate on today's hardware and not merely under theoretical future constraints, the overhead allowed to the decompressor was quite limited. When more processing time becomes available in the future, through the common adoption of better consumer hardware, the compression scheme can simply be replaced with a better performing technique.

While the compression performance of the system appears to have room for future improvement, there are cases where our scheme clearly outperforms compressed VRML. If one considers the case of the cutaway teapots, the compressed VRML file (with the included texture maps) is clearly larger than our compressed sequence for every case. If one includes the materials, the best case PRT file (at 10 bits per vertex component) is merely one third of the combined VRML sequence. In all cases the 10 bits per vertex component case always outperforms the equivalent compressed VRML sequences, and in all cases except the Dodge flyby the 16 bits per vertex component sequences also had smaller final file sizes than their VRML counterparts. Compressed VRML was clearly beaten in these cases, in spite of the relatively poor performance of LZSS when compared to GZip.

In the case of the scene of the airplane flying by the stationary automobile, our scheme greatly outperforms any of the other schemes. This is largely due to the large number of polygons

culled from the scene. The final scene contains 10161 triangles versus the original 21632 triangles, and 9461 vertices versus the original 14605 vertices, a reduction in polygon count of over 50% and a reduction in vertex count by 35%. Combined, these factors allow our final scene description to be a mere 53.5% of the compressed VRML equivalent. This scene provides an excellent example of how effective view dependent culling can be at removing information from non-interactive scenes. By removing this extraneous information, the scene not only requires less storage space but also requires less processing overhead for the decompression engine to render, and thus can be rendered at a faster rate for smoother animation.

It should be acknowledged that view dependent culling can prove ineffective on objects that are entirely revealed in a scene, such as the various tori in the tori hierarchy scene, and the teapots in the teapot scene. As well, minor expansion can occasionally take place due to smoothing groups, as previously explained, as some vertices must be reproduced when objects are divided up. However, as the other scenes demonstrate, objects that are partially obscured through a scene greatly benefit from those unseen faces being eliminated.

Though our new system outperforms VRML in terms of output file size, it has a minor

Table 6.6 Performance results from playback of PRT files on a Pentium II - 400 Mhz under Windows NT 4.0 with Service Pack 6 installed. Frame rates for each scene were generated by averaging four playings of each of the four sub-files.

Scene Name	*Playback Average Frames per Second*	*Total Vertices in Final Scene*	*Percentage of Original Vertices*	*Total Faces in Final Scene*	*Percentage of Original Faces*
Teapot	134.36	2,131	99.77%	3,958	96.37%
Dodge	30.17	10,701	100.01%	13,603	81.17%
Teapot Cutaway	96.76	3,700	85.06%	5,948	76.36%
Tori Hierarchy	90.10	2,391	84.13%	4,370	77.59%
Plane & Dodge	34.42	9,461	65.78%	10,161	46.97%

visualization artifact. In scenes exported with 8 or 10 bits per vertex component there can be minor rendering anomalies along object seams. It should be noted that these anomalies take place despite the average error per vertex being extremely low - our worst case example had an average error rate below 0.4%. Though this vertex accuracy problem is not exhibited in individual models, the lack of floating point precision in the 8 and 10 bits per vertex component scenes can cause minor differences in vertex values between different models. These differences arise due to vertex values being scaled to fit into a specific bit range, where the the front and tail of the range are determined using the bounding box of the exported polygons of that object. As the bounding boxes will be different for each individual object within a greater model, vertices in these varying sub-objects will each have a different amount of accuracy to the original sub-object outline. While these slight differences do not impact the appearance of a sub-object itself, they can cause minor disturbances in the appearance of the seams between these sub-objects. These disturbances

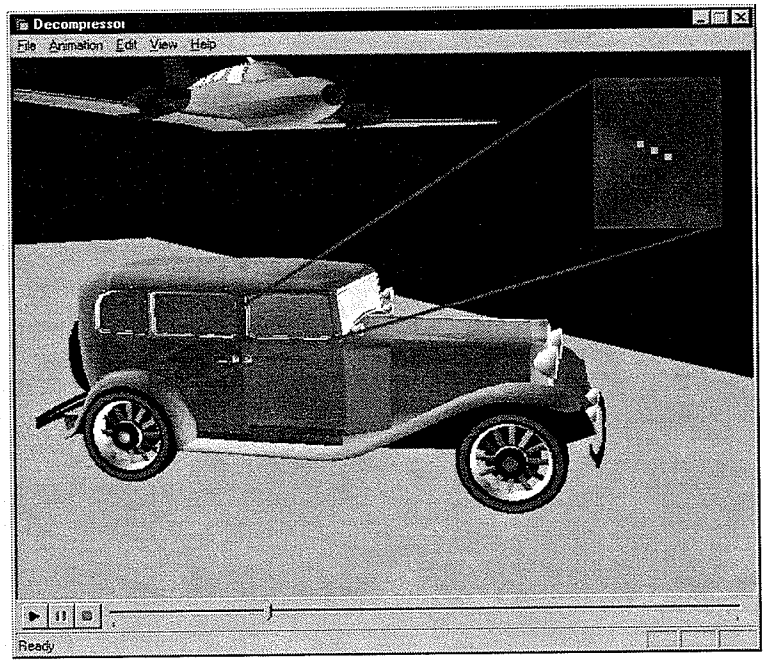


Fig. 6.1 A rendering of the 10 bits per vertex component version of the airplane & automobile scene demonstrating artifacts from incorrect alignment of sub-objects in a model.

along the seams generally appear as individual pixels missing from the seam line (see Fig. 6.1).

There is a further artifact our system produces when downloading sequences. In scenes that do not include normal vectors, normal vectors need to be interpolated. However, this calculation can not be correctly performed until objects are fully specified. This calculation is also performed in the same manner in VRML rendering systems. But, in our system we attempt to render scenes before they are fully specified, and thus when normals are not included with the scene. In such cases flat shading, the technique of using a constant face normal rather than 3 vertex normals to determine local per-face shading, rather than Gouraud shading must be performed on the scene until the calculations can be performed. This causes a temporary reduction in render quality until the object is fully specified in the data stream. This problem can be largely alleviated by dividing larger models into a series of sub-objects. As stated before, this may cause some minor artifacting in 10 and 8 bits-per-vertex sequences.

Even if we ignore these artifacts, the rendering quality of the real time engines employed by our system as well as those of VRML renderers is generally not as good as the rendered images produced by the offline rendering engine used by 3D Studio Max. Whereas lighting is performed per-vertex in the real time rendering engines, it is performed per-pixel in the 3D Studio Max renderer. This has its greatest impact when lighting large surfaces that are not heavily tessellated. In the per-vertex lighting systems, large surfaces with very little tessellation will exhibit incorrect or missing specular highlights. This problem could be alleviated to a degree by tessellating surfaces received by the decompressor to provide for finer lighting detail. Unfortunately the problem of shadows not appearing under the real time rendering engines is not as easily dealt with - there is no current practical solution to the problem of arbitrarily oriented objects casting shadows against one another in real time. Thus determination of shadowing would need to be done as a preprocessing step during the compression phase of the system, and

would thus in a best case scenario, add extra information into the compressed output.

Moreover, the offline rendered output makes use of full frame anti-aliasing. Anti-aliasing is a technique employed in off-line rendering engines to improve visual quality by super sampling each frame pixel. The simplest method to anti-alias a scene requires the sampling of sub-pixel values at each of the four corners of the square frame pixel. These values are averaged to produce the final destination pixel stored in the frame. As it requires quadrupling the rendering fill-rate (the number of pixels that need to be drawn), full frame anti-aliasing is only now coming into vogue in consumer real time rendering hardware with the introduction of the 3dfx VSA-100 chip. This new chip allows the automatic enabling of antialiasing for its rendering, allowing for much more visually pleasing rendered output. It should also be noted that the addition of anti-aliasing and its smoothing properties should generally remove the seam-line artifacts described earlier.

Yet rendered frame quality must be sacrificed in order to keep processing loads to a minimum, while achieving small final file sizes. Some of these higher order effects are simply not currently practical given the state of today's 3D hardware. For example, it would be possible to predetermine, during the compression phase, the shadowing performed by the scene lights and to determine which objects are in shadow and how they shadow each other. These shadows could be stored as texture maps and transmitted along with object descriptions. Unfortunately, this would require additional storage (a shadow texture per object) and would only work for static objects that never moved. General purpose shadowing is simply not practical as it requires too great a processing overhead. Thus, as of yet, the determination of scene shadowing and likewise, scene reflection is not currently practical on consumer hardware. As stated earlier, render quality of real time systems must be somewhat sacrificed, though in general these sacrifices are often minimal. As our scenes are for training purposes, the lack of accurate (to offline rendering

engines') lighting behaviour is not a critical flaw.

For the compression portion, the greatest problem with our new system is the time required to perform the visibility testing. With the current implementation, our system requires approximately 8-12 seconds per frame to rebuild an object database of medium complexity (~5000-6000 triangles), perform a simple render, and scan the bitmap for polygon colors. Thus a typical 5 second sequence requires the testing of 300 frames (we double the frame rate from 30 to test against the CFF), or approximately 50 minutes of processing on a Pentium II-400 with 64MB of RAM. Database construction is a memory intensive process and thus can generally run faster on systems with more memory to work with.

As the object setup typically takes the greatest amount of time for more complex scenes under our compression system, the system is not as heavily affected by either rendering or scanning time. Database construction time could be reduced if the database were only constructed once for the initial frame and the objects simply transformed afterwards. In the meantime, the compression of a scene can take a great deal of time, as compression time becomes relative to animation length, much as it is with more traditional video compression techniques. But unlike traditional video compression techniques, our final output file size will not grow significantly should an animation sequence be stretched out in time.

Instead, if equated with exporting to VRML and then performing lossless compression, our system performs poorly when comparisons are based on time of construction. The entire process of exporting to VRML and then applying GZip typically takes mere seconds regardless of scene complexity.

Nevertheless our compression system has a natural advantage over VRML, since our technique can begin animating before the entire scene has been received by the client machine. If very low bandwidth channels, such as 28.8 kbps modems are used, this advantage is not as

obvious, as only very small portions of the stream won't need to be cached. For example, for the 10 bit version of the Dodge experiment which is 16.7 seconds long, the first 95694 bytes of the 137444 total must be locally cached and processed before animation may begin. This may not seem like a great deal of savings, but it means that the sequence could begin playback under our system when less than two thirds of the equivalent GZip compressed VRML file had been received.

6.3 Summary

Performance of our system is better than that of compressed VRML from the point of view of required download time as well as final storage requirements. By reducing the amount of information found in the scene, greater levels of compression are achieved. However, the employed methods of reducing the amount of data required for transmission come at a cost - there are some minor visualization artifacts present in current renderings.

It is believed that these visualization artifacts should be eliminated by future improvements in rendering hardware. As well, the efficiency of the basic LZSS algorithm as a lossless compression technique is questionable, and this allows room for future improvement in the system.

Chapter VII
Conclusions and Recommendations

7.1 Conclusions

Following the analysis of the results obtained from the experiments it is clear that the newly introduced methodology of compressing 3D animation sequences produces smaller resulting files than those of compressed VRML. In addition, these scenes can begin animating earlier when downloaded over the Internet to client systems as the new system successfully breaks the scene up temporally. The amount of the scene that must be buffered before animation begins is scaled against the user-specified bandwidth, allowing the progressive transmission of sequences on even very low bandwidth channels such as 28.8 kbps modems. By combining optimal scene culling by the compressor in conjunction with the progressive construction at the decompressor, it is possible to transmit only the absolute minimum amount of information from the original scene database necessary in order to specify the sequence. This buildup of scene data over time allows the informational content of the scene to be delivered in order and thus enable the rendering of objects that have been received while new objects continue to arrive in the background.

As this work demonstrates, transmitting 3D animated sequences designed for use with teaching and training applications, and thus containing objects of static topology, are better served by utilizing an object oriented compression technique rather than the more traditional video compressors. In addition to naturally smaller file sizes, transmitting geometry allows for increased viewing resolutions and higher frame rates without additional storage costs. Yet the image quality of such sequences is generally not as high at these same resolutions as that of offline rendered sequences, as they have much greater time in which to perform more detailed lighting

and post-processing effects.

Differing levels of content quality can be set in our system for the data stream by reducing the resolution of vertex data exported, with a cost of larger file sizes for increased quality. However reducing the precision of transmitted vertices too far can result in some unwanted artifacts appearing along inter-object seams. As these artifacts appear when using the apparently ideal coordinate quantization rate of 10 bits per component, we can conclude that this apparently ideal rate only applies to smooth surfaces, and does not include multiple objects connected along seam lines. Regardless, it is believed that a large number of these artifacts will naturally vanish through the introduction of full-screen anti-aliasing support, which will be available in upcoming consumer 3D hardware accelerators such as the 3dfx VSA-100, ATI Rage-6 and NVidia NV-20, all of which are due within the next 6 months.

In addition to advancements in 3D hardware, continuing improvements in microprocessor technology allow for the replacement of some of the techniques used in this system. As demonstrated experimentally, the LZSS compressor utilized in this thesis performs its fundamental purpose, but does so rather poorly when compared to other more computationally intensive techniques. Nonetheless, the system still outperforms all tested examples of 3D animation compression. While there remains room for improvement of some of the techniques presented in this thesis as discussed below, this work has been demonstrated experimentally to outperform existing systems in the compression and decompression of 3D animated sequences containing objects with static topologies.

7.2 Recommendations

There are several recommendations on how the work in this thesis could be furthered and improved in future work. The first class of recommendations seek to take advantage of the growth in processor capability since the beginning of this thesis. Consumer targetted processor speeds are currently more than twice as powerful as they were when we began this work (as of this writing 1 GHz AMD Athlon processors are now available), and thus there is now room for the introduction of more complex algorithms. The first and most obvious expansion that could be introduced is to consider more powerful lossless compression techniques over the current simple LZSS implementation in use. Even an adaptation of the unused deletion heuristic should improve performance.

The next recommendation in this class is to consider using more complex wavelet bases for the compression of the texture maps. A variety of wavelet functions of intermediate complexity have been heavily examined in the research community, such as the Daubechies and Cioflet families. While even the lower order Daubechies and Cioflet bases are a great deal more computationally complex than the simple Haar basis (the simplest of all wavelets), Haar is generally a poor performer compared to other wavelet bases [ViBL95] used for image compression.

The next class of recommendations is to expand on this thesis work by improving performance. For example, it would be worthwhile to investigate at what exact bit rate the pixel artifacts begin to appear. If it is due to a specific error threshold, it should be more effective to simply have the compression engine choose the lowest bit rate that will remain above the given threshold. In any case, it would be better to simply automatically select the best candidate bit rate rather than allow the user to choose the bit rate, as is currently possible.

Another more complex idea worth pursuing is to examine if it is possible to efficiently combine progressive meshes with our existing system by compensating for the growing object boundaries. This could allow for even better compression ratios as selective quality could be enhanced or reduced along varying portions of objects. Thus objects far from the virtual camera could be sent less detailed information without seriously affecting what the viewer sees. Progressive meshes could be further enhanced by combining them together with subdivision surfaces. Subdivision surfaces are a recent technology that allows the refinement of polygonal meshes to synthesize additional detail [ZSSw96]. It might be practical to dedicate some of the soon to be available additional processing capabilities towards treating the meshes as subdivision surfaces. This could allow for smoother, more realistic looking, and finer detailed models without requiring additional information added to the data stream. It could even allow for reduction of the data stream in cases where it is known that the subdivision would introduce the required detail. Thus, the integration of both progressive mesh and subdivision surface technology should be able both to reduce greatly the needed bandwidth as well as improve rendered image quality.

One final area worth pursuing would be the integration of orthogonal illumination maps [TCRS00] to fake the appearance of greater polygonal detail in models with low triangle counts. These textures are bump-mapped (bump mapping involves perturbing surface normals to provide the illusion of greater surface-light interaction) onto existing objects to produce detail that would normally be impossible without the addition of many orders of magnitude greater geometry detail.

Orthogonal illumination maps allow per-pixel lighting rather than the normal per-vertex lighting allowed in real time engines. This is accomplished through a six-pass rendering scheme which renders precalculated axis aligned normal maps (texture maps whose values represent normals rather than colors) that have been adjusted against the given light sources onto the target objects. Such normal maps are stored using palettized textures, which could provide for even

greater compression rates. These special texture maps could be integrated into the existing texture transport system, and could allow for the addition of very high detail at minimal cost, in terms of both bandwidth and decompression overhead. The only major flaw in this scheme is that it requires a great deal of fill rate from the 3D accelerator. Yet, just as the upcoming generation of 3D hardware accelerators should solve the issue of anti-aliasing, it will provide the fill rate and support necessary for hardware bump-mapping. As with subdivision surfaces and progressive meshes, there remains a lot of room in which to experiment with additions and improvements to the existing system.

7.3 Contributions

This thesis contributes to science and technology by:

- introducing a novel approach to geometry compression;
- demonstrating the power and legitimacy of accurate geometry culling in non-interactive scenarios enabling 3D content to be efficiently transmitted to viewers;
- demonstrating a novel approach to the progressive transmission of triangular meshes;
- demonstrating the practicality of utilizing wavelets to reconstruct MIP map levels for progressive transmission of texture maps; and
- providing Mecca Media Group with a superior compression scheme and a practical alternative to using video compression to transmit animation sequences for use in teaching and training.

REFERENCES

[Auto95] Autodesk Technical Support, "How to calculate KXP spline derivatives", 1995, http://www.autodesk.com/support/techdocs/fax700/fax746.htm

[CoWa93] Andrew S. Glassner, *Radiosity and Realistic Image Synthesis*, Cambridge, Academic Press Inc., 1993, 381 pp.

[CaBM97] Rikk Carey, Gavin Bell, Chris Marrin, *Virtual Reality Modeling Language (VRML97)*, VRML Consortium Inc., 1997, 236 pp.

[DaKi99] Richard Danserau, Witold Kinsner, "Rényi generalized entropy analysis of images from a progressive wavelet transmission," Proceedings of the Canadian Conference on Electrical and Computer Engineering (CCECE99 Edmonton), May 1999.

[ElVa98] Jihad El-Sana, Amitabh Varshney, "Topology simplification for polygonal virtual environments," IEEE Transactions on Visualization and Computer Graphics, Vol. 4, No. 2, 1998, pp. 133-144.

[Ever99] Cass Everitt, "Orthogonal illumination maps," Prepared for OpenGL.org, August 1999, http://www.opengl.org/News/Special/oim/Orth.html

[Ever00] Cass Everitt, *High Quality Hardware Accelerated Per-Pixel Illumination*, draft, Master's thesis, Department of Computer Science, Mississippi State University, 2000, 25 pp.

[FDFH87] James D. Foley, Andries van Dam, Steven K. Feiner, John F. Hughes, *Computer Graphics – Principles and Practice*, Reading, Addison-Wesley Publishing Company, 1987, 1174 pp.

[Fosn96] Ron Fosner, *OpenGL Programming for Windows 95 and Windows NT,* Reading, Addison-Wesley Publishing Company, 1996, 259 pp.

[Glas95] Andrew S. Glassner, *Principles of Digital Image Synthesis,* San Francisco, Morgan Kaufmann Publishers Inc., 1995, 1206 pp.

[GHJS98] Tran Gieng, Bernd Hamann, Kenneth Joy, Gregory Schussman, and Isaac Trotts, "Constructing hierarchies for triangular meshes", IEEE Transactions on Visualization and Computer Graphics, Vol. 4, No. 2, 1998, pp. 145-160.

[Guéz99] André Guéziec, "Locally toleranced surface simplification", IEEE Transactions on Visualization and Computer Graphics, Vol. 5, No. 2, 1999, pp. 168-189.

[Heck97] Chris Hecker, "Physics part 4: The third dimension", *Game Developer Magazine*,

June 1997, pp. 15-26.

[Hill90] F. S. Hill, Jr., *Computer Graphics*, Englewood Cliffs, Prentice-Hall Inc., 1990, 754 pp.

[HDDM93] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, Werner Steutzle, "Mesh optimization", SIGGRAPH 1993 Proceedings, 1993, pp. 19-26.

[Hopp96] Hugues Hoppe, "Progressive meshes", SIGGRAPH 1996 Proceedings, 1996, pp. 99-108.

[Kins91] Witold Kinsner, "Review of data compression methods including Shannon-Fao, Huffman, arithmetic, Storer, Lempel-Ziv-Welch, fractal, neural networks, and wavelet algorithms," *Technical Report*, DEL91-1, Department of Electrical & Computer Engineering, University of Manitoba, Winnipeg, 1991, 157 pp.

[Kins94] Witold Kinsner, "Fractal dimensions: Morphological, entropy, spectra, and variance classes," *Technical Report*, DEL94-4, Department of Electrical & Computer Engineering, University of Manitoba, Winnipeg, 1994, 146 pp.

[Kins99] Witold Kinsner, Fractal & Chaos Engineering. *Lecture Notes*; Department of Electrical & Computer Engineering, University of Manitoba, Winnipeg, 1999.

[Kins00] Witold Kinsner, Signal & Data Compression. *Lecture Notes*; Department of Electrical & Computer Engineering, University of Manitoba, Winnipeg, 2000, 617 pp.

[MJFS99] Bongki Moon, H.V. Jagadish, Christos Faloutsos, Joel Saltz, "Analysis of the clustering properties of the Hilbert space-filling curve", To appear in IEEE Transactions on Knowledge and Data Engineering, submitted August 1999, 25 pp.

[MPEG4a] Requirements group, "MPEG-4 applications", International Organisation for Standardisation - Coding of Moving Pictures and Audio Committee, March 1999, 60 pp.

[MPEG4b] Requirements group, "MPEG-4 requirements, version 13", International Organisation for Standardisation - Coding of Moving Pictures and Audio Committee, December 1999, 30 pp.

[MPEG4c] Systems group, "Multi-users technology", International Organisation for Standardisation - Coding of Moving Pictures and Audio Committee, December 1999, 5 pp.

[MPEG4d] "Call for proposals for an MPEG-4 texual format", International Organisation for Standardisation - Coding of Moving Pictures and Audio Committee, December 1999, 14 pp.

[OpWY83] Alan Oppenheim, Alan Willsky, Ian Young, *Signals and Systems*, New Jersey, Prentice-Hall Inc., 1983, 796 pp.

[PeJS92] Heinz-Otto Peitgen, Hartmut Jurgens, and Dietmar Saupe, *Chaos and Fractals - New Frontiers of Science*, New York, Springer-Verlag, 1992, 984 pp.

[Ross99] Jarek Rossignac, "Edgebreaker: Connectivity compression for triangle meshes", IEEE Transactions on Visualization and Computer Graphics, Vol. 5, No. 1, 1999, pp. 47-61.

[Sayo96] Khalid Sayood, *Introduction to Data Compression*, San Francisco, Morgan Kaufmann Publishers Inc., 1996, 475 pp.

[Stor88] James A. Storer, *Data Compression: Methods and Thoery,* Rockville, Computer Science Press Inc., 1988, 406 pp.

[TCRS00] M. Tarini, P. Cignoni, C. Rocchini, and R. Scopigno, "Real time, accurate, multi-featured rendering of bump mapped surfaces", Eurographics, Vol. 19, No. 3, 2000, 12 pp.

[ViBL95] John D. Villasenor, Benjamin Belzer, and Judy Liao, "Wavelet filter evaluation for image compression," IEEE Transactions on Image Processing, Vol. 2, August 1995, pp 1053-1060.

[WaWa92] Alan Watt, Mark Watt, *Advanced Animation and Rendering Techniques – Theory and Practice*, New York, ACM Press, 1992, 455 pp.

[Whit92] Scott Whitman, *Multiprocessor Methods for Computer Graphics Rendering*, Jones and Bartlett Publishers, 1992, 218 pp.

[WNDa97] Mason Woo, Jackie Neider, Tom Davis. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.1*, 2nd edition, Addison-Wesley Publishing Company, 1997, 650 pp.

[ZSSw96] Denis Zorin, Peter Schröder, Wim Sweldens, "Interpolating subdivision for meshes of arbitrary topology", SIGGRAPH 1996 Proceedings, 1996, 20 pp.

APPENDIX A
OPCODE LISTING

Each OpCode is listed along with a brief explanation, the opcode's numeric equivalent and the instruction's list of arguments. To simplify the description of the various opcodes argument lists, arguments are listed as "t*ype_name*", where *type* provides a reference to the data type used to store the value, and *name* gives a name reference for the specific ordered value. Types are italicized to ease reading.

Name: OPCODE_COMPLETETOFRAME **Value**: 1
Arguments: *Byte*_F0 *Byte*_F1 *Byte*_F2
Comments: The 3 bytes represent the successive bytes in a 24 bit unsigned frame number. This opcode updates the `ReadyTillFrame` marker to the specified frame number.

Name: OPCODE_ADDVERTS **Value**: 2
Arguments: *Short*_Which *Short*_Total *DataBlock*_VertexValues
Comments: This opcode appends a list of vertices to an object. The first short value specifies which object reference number the vertices should be appended to. This object must already exist in the animation database. The second short value provides the total number of vertices that are listed in the data block that follows.
The size of the data block varies due to the size of each vertex component and the number of vertices in the list. Vertex data size is specified in the header for the file. Total size of the data block should be equal to the number of bit per vertex component * 3 * Total number of vertices / 8, rounded up to the nearest integer value.

Name: OPCODE_ADDNORMS **Value**: 3
Arguments: *Short*_Which *Short*_Total *DataBlock*_NormValues
Comments: This opcode appends a list of normal vectors to an object. The first short value specifies which object reference number the normals should be appended to. This object must already exist in the animation database. The second short value provides the total number of normals that are listed in the data block that follows.
The size of the data block depends on the setting of `bCompressedNormals` (specified in the header). If compressed normals are being used, the datablock size is equal to the total number of normals * 3. If uncompressed normals are being used, the datablock size is equal to the total number of normals * 6. Uncompressed normals range between -1.0 and 1.0, and are scaled 16 bit values.

Name: OPCODE_ADDFACES **Value**: 4
Arguments: *Short*_Which *Short*_Total *DataBlock*_FaceValues
Comments: This opcode appends a list of face references to an object. The first short value specifies which object reference number the faces should be appended to. This object must already exist in the animation database. The second short value provides the total number of faces that are listed in the data block that follows.
The data block is a series of shorts, where the total number of shorts is equal to the total number of faces in the list * 3.

Name: OPCODE_ADDTEXCOORDS **Value**: 5
Arguments: *Short*_Which *Short*_Total *DataBlock*_TexCoordValues
Comments: This opcode appends a list of texture coordinates references to an object. The first short value specifies which object reference number the coordinates should be appended to. This object must already exist in the animation database. The second short value provides the total number of coordinates that are listed in the data block that follows.
Each vertex coordinate is packed into a 24 bit word, with the U value occupying the top 12 bits, and the V value occupying the bottom 12 bits. These bit values represent numbers ranging between -8.0 and +8.0. Thus the size of the data block is equal to the total number in the list * 3.

Name: OPCODE_ADDVERTCOLORS **Value**: 6
Arguments: *Short*_Which *Short*_Total *DataBlock*_ColorValues
Comments: This opcode appends a list of vertex colors to an object. The first short value specifies which object reference number the coordinates should be appended to. This object must already exist in the animation database. The second short value provides the total number of colors that are listed in the data block that follows. Colors are represented as 16 bit color values with the original red and blue values divided by 8 and the green value divided by 4. The size of the data block is equal to the number of colors in the list * 2.

Name: OPCODE_ASSIGNOBJECT_SOLIDCOLOR **Value**: 12
Arguments: *Short*_Which *Word*_ColorValue
Comments: This opcode assigns a solid color to an entire model, instead of the model owning a texture or possessing per-vertex coloring. The first short value specifies which object reference number the coordinates should be appended to. This object must already exist in the animation database. Colors are represented as 16 bit color values with the original red and blue values divided by 8 and the green value divided by 4.

Name: OPCODE_ADDOBJECT **Value**: 24
Arguments: *Short*_TotalVertices *Short*_TotalFaces <*Float*_MinCoord> <*Float*_MaxCoord>
Comments: This opcode creates a new object in the animation database, and assigns this new object a reference number equal to the number of previously existing models in the database. The object is assigned total vertex and face counts based on the arguments specified. If the bits-per-vertex value specified in the file header was

not 32, there follows two 3D coordinate values that define a bounding box for the model. All new vertices added to the list, again assuming the bits-per-vertex value was not 32, are scaled against the bounding box values.

Name: OPCODE_SETPARENTCHILD **Value**: 28
Arguments: *Short*_Child *Short*_Parent
Comments: This opcode defines a parent-child relationship between two existing models in the database. This is accomplished by assigning the parent to the child node, as each node can have a single parent, but multiple children. Both the child and parent models must already exist in the database.

Name: OPCODE_KEYFRAME_TCBPOS **Value**: 32
Arguments: *Short*_Which *Byte*_Time0 *Byte*_Time1 *Byte*_Time2 *Byte*_Time3 *Float*_PosX *Float*_PosY *Float*_PosZ *Word*_Tension *Word*_Continuity *Word*_Bias *Word*_EaseIn *Word*_EaseOut
Comments: This opcode defines a TCB position keyframe for an object - in conjunction with several other TCB keyframes an object can be meant to travel through space. The object must already exist in the database. Time values are stored as total number of ticks into the animation the keyframe takes place at. Keyframes times are signed, as they are allowed to exist before the 0-frame.

Name: OPCODE_KEYFRAME_LINPOS **Value**: 33
Arguments: *Short*_Which *Byte*_Time0 *Byte*_Time1 *Byte*_Time2 *Byte*_Time3 *Float*_PosX *Float*_PosY *Float*_PosZ
Comments: This opcode defines a linear position interpolation keyframe for an object - in conjunction with another linear position interpolation keyframe an object can be meant to travel through space. The object must already exist in the database. Time values are stored as total number of ticks into the animation the keyframe takes place at. Keyframes times are signed, as they are allowed to exist before the 0-frame.

Name: OPCODE_KEYFRAME_BEZPOS **Value**: 34
Arguments: *Short*_Which *Byte*_Time0 *Byte*_Time1 *Byte*_Time2 *Byte*_Time3 *Float*_PosX *Float*_PosY *Float*_PosZ *Float*_InTanX *Float*_InTanY *Float*_InTanZ *Float*_OutTanX *Float*_OutTanY *Float*_OutTanZ
Comments: This opcode defines a Bezier position interpolation keyframe for an object - in conjunction with another Bezier position interpolating keyframe an object can be meant to travel through space. The object must already exist in the database. Time values are stored as total number of ticks into the animation the keyframe takes place at. Keyframes times are signed, as they are allowed to exist before the 0-frame.

Name: OPCODE_KEYFRAME_TCBROT **Value**: 35
Arguments: *Short*_Which *Byte*_Time0 *Byte*_Time1 *Byte*_Time2 *Byte*_Time3 *Float*_Angle *Word*_AxisX *Word*_AxisY *Word*_AxisZ *Word*_Tension *Word*_Continuity *Word*_Bias *Word*_EaseIn *Word*_EaseOut

Comments: This opcode defines a TCB rotation keyframe for an object - in conjunction with several other TCB keyframes an object can be meant to reorient itself over time. The rotation performed by a TCB rotation is a relative rotation. The object must already exist in the database. Time values are stored as total number of ticks into the animation the keyframe takes place at. Keyframes times are signed, as they are allowed to exist before the 0-frame.

Name: OPCODE_KEYFRAME_LINROT **Value**: 36

Arguments: *Short*_Which *Byte*_Time0 *Byte*_Time1 *Byte*_Time2 *Byte*_Time3 *Word*_QuatX *Word*_QuatY *Word*_QuatZ *Word*_QuatW

Comments: This opcode defines a linear rotation keyframe for an object - in conjunction with another linear keyframes an object can be meant to reorient itself over time. The rotation performed by a linear rotation is an absolute rotation. The object must already exist in the database. Time values are stored as total number of ticks into the animation the keyframe takes place at. Keyframes times are signed, as they are allowed to exist before the 0-frame.

Name: OPCODE_KEYFRAME_TCBSCL **Value**: 38

Arguments: *Short*_Which *Byte*_Time0 *Byte*_Time1 *Byte*_Time2 *Byte*_Time3 *Float*_ScaleX *Float*_ScaleY *Float*_ScaleZ *Word*_Tension *Word*_Continuity *Word*_Bias *Word*_EaseIn *Word*_EaseOut

Comments: This opcode defines a TCB scale keyframe for an object - in conjunction with several other TCB keyframes an object can be meant to change scale over time. The object must already exist in the database. Time values are stored as total number of ticks into the animation the keyframe takes place at. Keyframes times are signed, as they are allowed to exist before the 0-frame.

Name: OPCODE_KEYFRAME_LINSCL **Value**: 39

Arguments: *Short*_Which *Byte*_Time0 *Byte*_Time1 *Byte*_Time2 *Byte*_Time3 *Float*_ScaleX *Float*_ScaleY *Float*_ScaleZ

Comments: This opcode defines a linear scale interpolation keyframe for an object - in conjunction with another linear scale interpolation keyframe an object can be meant to change scale over time. The object must already exist in the database. Time values are stored as total number of ticks into the animation the keyframe takes place at. Keyframes times are signed, as they are allowed to exist before the 0-frame.

Name: OPCODE_KEYFRAME_BEZSCL **Value**: 40

Arguments: *Short*_Which *Byte*_Time0 *Byte*_Time1 *Byte*_Time2 *Byte*_Time3 *Float*_ScaleX *Float*_ScaleY *Float*_ScaleZ *Float*_InTanX *Float*_InTanY *Float*_InTanZ *Float*_OutTanX *Float*_OutTanY *Float*_OutTanZ

Comments: This opcode defines a Bezier scale interpolation keyframe for an object - in conjunction with another Bezier scale interpolating keyframe an object can be meant to change scale over time. The object must already exist in the database. Time values are stored as total number of ticks into the animation the keyframe takes place at. Keyframes times are signed, as they are allowed to exist before the 0-frame.

Name: OPCODE_CAMERA_KEYFRAME_TCBPOS **Value**: 51

Arguments: *Byte*_Time0 *Byte*_Time1 *Byte*_Time2 *Byte*_Time3 *Float*_PosX *Float*_PosY *Float*_PosZ *Word*_Tension *Word*_Continuity *Word*_Bias *Word*_EaseIn *Word*_EaseOut

Comments: This opcode defines a TCB position keyframe for the camera - in conjunction with several other TCB keyframes the camera can be meant to travel through space. Time values are stored as total number of ticks into the animation the keyframe takes place at. Keyframes times are signed, as they are allowed to exist before the 0-frame.

Name: OPCODE_CAMERA_KEYFRAME_LINPOS **Value**: 52

Arguments: *Byte*_Time0 *Byte*_Time1 *Byte*_Time2 *Byte*_Time3 *Float*_PosX *Float*_PosY *Float*_PosZ

Comments: This opcode defines a linear position interpolation keyframe for the camera - in conjunction with another linear position interpolation keyframe the camera can be meant to travel through space. Time values are stored as total number of ticks into the animation the keyframe takes place at. Keyframes times are signed, as they are allowed to exist before the 0-frame.

Name: OPCODE_CAMERA_KEYFRAME_BEZPOS **Value**: 53

Arguments: *Byte*_Time0 *Byte*_Time1 *Byte*_Time2 *Byte*_Time3 *Float*_PosX *Float*_PosY *Float*_PosZ *Float*_InTanX *Float*_InTanY *Float*_InTanZ *Float*_OutTanX *Float*_OutTanY *Float*_OutTanZ

Comments: This opcode defines a Bezier position interpolation keyframe for the camera - in conjunction with another Bezier position interpolating keyframe the camera can be meant to travel through space. Time values are stored as total number of ticks into the animation the keyframe takes place at. Keyframes times are signed, as they are allowed to exist before the 0-frame.

Name: OPCODE_CAMERA_KEYFRAME_TCBROT **Value**: 54

Arguments: *Byte*_Time0 *Byte*_Time1 *Byte*_Time2 *Byte*_Time3 *Float*_Angle *Word*_AxisX *Word*_AxisY *Word*_AxisZ *Word*_Tension *Word*_Continuity *Word*_Bias *Word*_EaseIn *Word*_EaseOut

Comments: This opcode defines a TCB rotation keyframe for the camera - in conjunction with several other TCB keyframes the camera can reorient itself over time. The rotation performed by a TCB rotation is a relative rotation. Time values are stored as total number of ticks into the animation the keyframe takes place at. Keyframes times are signed, as they are allowed to exist before the 0-frame.

Name: OPCODE_CAMERA_KEYFRAME_LINROT **Value**: 55

Arguments: *Byte*_Time0 *Byte*_Time1 *Byte*_Time2 *Byte*_Time3 *Word*_QuatX *Word*_QuatY *Word*_QuatZ *Word*_QuatW

Comments: This opcode defines a linear rotation keyframe for the camera - in conjunction with another linear keyframes the camera can reorient itself over time. The rotation performed by a linear rotation is an absolute rotation. Time values are stored as

total number of ticks into the animation the keyframe takes place at. Keyframes times are signed, as they are allowed to exist before the 0-frame.

Name: OPCODE_CAMERA_KEYFRAME_TCBRLL **Value**: 57
Arguments: *Byte*_Time0 *Byte*_Time1 *Byte*_Time2 *Byte*_Time3 *Float*_RollValue
Comments: This opcode defines a TCB roll keyframe for the camera around the camera's z-axis. Time values are stored as total number of ticks into the animation the keyframe takes place at. Keyframes times are signed, as they are allowed to exist before the 0-frame.

Name: OPCODE_CAMERA_KEYFRAME_LINRLL **Value**: 58
Arguments: *Byte*_Time0 *Byte*_Time1 *Byte*_Time2 *Byte*_Time3 *Float*_RollValue
Comments: This opcode defines a linear roll keyframe for the camera around the camera's z-axis. Time values are stored as total number of ticks into the animation the keyframe takes place at. Keyframes times are signed, as they are allowed to exist before the 0-frame.

Name: OPCODE_CAMERA_KEYFRAME_BEZRLL **Value**: 59
Arguments: *Byte*_Time0 *Byte*_Time1 *Byte*_Time2 *Byte*_Time3 *Float*_RollValue
Comments: This opcode defines a Bezier roll keyframe for the camera around the camera's z-axis. Time values are stored as total number of ticks into the animation the keyframe takes place at. Keyframes times are signed, as they are allowed to exist before the 0-frame.

Name: OPCODE_ADDCAMERA **Value**: 60
Arguments: *Word*_Field_of_View *Float*_NearPlane *Float*_FarPlane
Comments: Assigns a basic camera to the sign, providing a FOV value scaled up to between (0,3.141592), as well as a near and far plane. These values together are used to construct the view frustum.

Name: OPCODE_ADDCAMERA_WITHLOCATION **Value**: 61
Arguments: *Word*_Field_of_View *Float*_NearPlane *Float*_FarPlane *Float*_PosX *Float*_PosY *Float*_PosZ
Comments: Assigns a basic camera to the sign, providing a FOV value scaled up to between (0,3.141592), as well as a near and far plane. These values together are used to construct the view frustum. The camera absolute location is provided in x,y,z coordinates.

Name: OPCODE_SETCAMERA_ORIENTATION **Value**: 62
Arguments: *Word*_QuatX *Word*_QuatY *Word*_QuatZ *Word*_QuatW
Comments: Provides the existing camera with a fixed orientation. The four 16 bit values values define a quaternion by scaling the values to the range (-1.0,1.0).

Name: OPCODE_SETCAMERA_OFFSET **Value**: 63
Arguments: *Float*_Val0 *Float*_Val1 *Float*_Val2 *Float*_Val3 *Float*_Val4
*Float*_Val5 *Float*_Val6 *Float*_Val7 *Float*_Val8 *Float*_Val9

*Float*_Val10 *Float*_Val11 *Float*_Val12

Comments: Defines a 4x3 (for rows, 3 columns) matrix which is converted into a 4x4 matrix internally to provide an offset for the camera.

Name: OPCODE_SETOFFSET **Value:** 66

Arguments: *Short*_Which *Float*_Val0 *Float*_Val1 *Float*_Val2 *Float*_Val3 *Float*_Val4 *Float*_Val5 *Float*_Val6 *Float*_Val7 *Float*_Val8 *Float*_Val9 *Float*_Val10 *Float*_Val11 *Float*_Val12

Comments: Defines a 4x3 (for rows, 3 columns) matrix which is converted into a 4x4 matrix internally to provide an offset for specified object indicated by 'which'.

Name: OPCODE_SETPOSITION **Value:** 67

Arguments: *Short*_Which *Float*_PosX *Float*_PosY *Float*_PosZ

Comments: The absolute location is provided in x,y,z coordinates for the object specified by 'which' .

Name: OPCODE_SETORIENTATION **Value:** 68

Arguments: *Short*_Which *Word*_QuatX *Word*_QuatY *Word*_QuatZ *Word*_QuatW

Comments: Provides an existing object specified by 'which' with a fixed absolute orientation. The four 16 bit values values define a quaternion by scaling the values to the range (-1.0,1.0).

Name: OPCODE_SETPOSITION **Value:** 67

Arguments: *Short*_Which *Float*_PosX *Float*_PosY *Float*_PosZ

Comments: The absolute location is provided in x,y,z coordinates for the object specified by 'which' .

Name: OPCODE_SETSCALE **Value:** 69

Arguments: *Short*_Which *Float*_ScaleX *Float*_ScaleY *Float*_ScaleZ

Comments: The absolute scaling factor is provided as a float triple for the object specified by 'which' .

Name: OPCODE_ADDLIGHT_OMNI **Value:** 70

Arguments: *Word*_Color

Comments: Appends a new omni-directional point source light to the scene, with the given 16b encoded color. Colors are represented as 16 bit color values with the original red and blue values divided by 8 and the green value divided by 4.

Name: OPCODE_ADDLIGHT_SPOT **Value:** 71

Arguments: *Word*_Color

Comments: Appends a new spotlight to the scene, with the given 16b encoded color. Colors are represented as 16 bit color values with the original red and blue values divided by 8 and the green value divided by 4.

Name: OPCODE_ADDLIGHT_DIRECTIONAL **Value:** 72

Arguments: *Word*_Color

Comments: Appends a new directional to the scene, with the given 16b encoded color. Colors are represented as 16 bit color values with the original red and blue values divided by 8 and the green value divided by 4.

Name: OPCODE_SETLIGHT_ORIENTATION **Value**: 73
Arguments: *Short*_Which *Word*_QuatX *Word*_QuatY *Word*_QuatZ *Word*_QuatW
Comments: Provides an existing light specified by 'which' with a fixed absolute orientation. The four 16 bit values values define a quaternion by scaling the values to the range (-1.0,1.0).

Name: OPCODE_SETLIGHT_POSITION **Value**: 74
Arguments: *Short*_Which *Float*_PosX *Float*_PosY *Float*_PosZ
Comments: The absolute location is provided in x,y,z coordinates for the light specified by 'which' .

Name: OPCODE_SETOFFSET **Value**: 75
Arguments: *Short*_Which *Float*_Val0 *Float*_Val1 *Float*_Val2 *Float*_Val3 *Float*_Val4 *Float*_Val5 *Float*_Val6 *Float*_Val7 *Float*_Val8 *Float*_Val9 *Float*_Val10 *Float*_Val11 *Float*_Val12
Comments: Defines a 4x3 (for rows, 3 columns) matrix which is converted into a 4x4 matrix internally to provide an offset for specified light indicated by 'which'.

Name: OPCODE_LIGHT_KEYFRAME_TCBPOS **Value**: 76
Arguments: *Short*_Which *Byte*_Time0 *Byte*_Time1 *Byte*_Time2 *Byte*_Time3 *Float*_PosX *Float*_PosY *Float*_PosZ *Word*_Tension *Word*_Continuity *Word*_Bias *Word*_EaseIn *Word*_EaseOut
Comments: This opcode defines a TCB position keyframe for a light - in conjunction with several other TCB keyframes a light can be meant to travel through space. The light must already exist in the database. Time values are stored as total number of ticks into the animation the keyframe takes place at. Keyframes times are signed, as they are allowed to exist before the 0-frame.

Name: OPCODE_LIGHT_KEYFRAME_LINPOS **Value**: 77
Arguments: *Short*_Which *Byte*_Time0 *Byte*_Time1 *Byte*_Time2 *Byte*_Time3 *Float*_PosX *Float*_PosY *Float*_PosZ
Comments: This opcode defines a linear position interpolation keyframe for a light - in conjunction with another linear position interpolation keyframe a light can be meant to travel through space. The light must already exist in the database. Time values are stored as total number of ticks into the animation the keyframe takes place at. Keyframes times are signed, as they are allowed to exist before the 0-frame.

Name: OPCODE_LIGHT_KEYFRAME_BEZPOS **Value**: 78
Arguments: *Short*_Which *Byte*_Time0 *Byte*_Time1 *Byte*_Time2 *Byte*_Time3 *Float*_PosX *Float*_PosY *Float*_PosZ *Float*_InTanX *Float*_InTanY *Float*_InTanZ

*Float*_OutTanX *Float*_OutTanY *Float*_OutTanZ

Comments: This opcode defines a Bezier position interpolation keyframe for a light - in conjunction with another Bezier position interpolating keyframe a light can be meant to travel through space. The light must already exist in the database. Time values are stored as total number of ticks into the animation the keyframe takes place at. Keyframes times are signed, as they are allowed to exist before the 0-frame.

Name: OPCODE_LIGHT_KEYFRAME_TCBROT **Value**: 79

Arguments: *Short*_Which *Byte*_Time0 *Byte*_Time1 *Byte*_Time2 *Byte*_Time3 *Float*_Angle *Word*_AxisX *Word*_AxisY *Word*_AxisZ *Word*_Tension *Word*_Continuity *Word*_Bias *Word*_EaseIn *Word*_EaseOut

Comments: This opcode defines a TCB rotation keyframe for a light - in conjunction with several other TCB keyframes a light can be meant to reorient itself over time. The rotation performed by a TCB rotation is a relative rotation. The light must already exist in the database. Time values are stored as total number of ticks into the animation the keyframe takes place at. Keyframes times are signed, as they are allowed to exist before the 0-frame.

Name: OPCODE_LIGHT_KEYFRAME_LINROT **Value**: 81

Arguments: *Short*_Which *Byte*_Time0 *Byte*_Time1 *Byte*_Time2 *Byte*_Time3 *Word*_QuatX *Word*_QuatY *Word*_QuatZ *Word*_QuatW

Comments: This opcode defines a linear rotation keyframe for a light - in conjunction with another linear keyframes a light can be meant to reorient itself over time. The rotation performed by a linear rotation is an absolute rotation. The light must already exist in the database. Time values are stored as total number of ticks into the animation the keyframe takes place at. Keyframes times are signed, as they are allowed to exist before the 0-frame.

Name: OPCODE_LIGHT_KEYFRAME_TCBRLL **Value**: 82

Arguments: *Short*_Which *Byte*_Time0 *Byte*_Time1 *Byte*_Time2 *Byte*_Time3 *Float*_RollValue

Comments: This opcode defines a TCB roll keyframe for a light around the light's z-axis. Time values are stored as total number of ticks into the animation the keyframe takes place at. Keyframes times are signed, as they are allowed to exist before the 0-frame.

Name: OPCODE_LIGHT_KEYFRAME_LINRLL **Value**: 83

Arguments: *Short*_Which *Byte*_Time0 *Byte*_Time1 *Byte*_Time2 *Byte*_Time3 *Float*_RollValue

Comments: This opcode defines a linear roll keyframe for a light around the light's z-axis. Time values are stored as total number of ticks into the animation the keyframe takes place at. Keyframes times are signed, as they are allowed to exist before the 0-frame.

Name: OPCODE_LIGHT_KEYFRAME_BEZRLL **Value**: 84

Arguments: *Short*_Which *Byte*_Time0 *Byte*_Time1 *Byte*_Time2 *Byte*_Time3 *Float*_RollValue

Comments: This opcode defines a Bezier roll keyframe for the light around the light's z-axis.

Time values are stored as total number of ticks into the animation the keyframe takes place at. Keyframes times are signed, as they are allowed to exist before the 0-frame.

Name: OPCODE_ADDTARGET_TO_CAMERA **Value**: 87
Arguments: *Short*_Which
Comments: This opcode assigns a target to the camera. This will cause the camera to always reorient itself point at the specified object regardless of their locations.

Name: OPCODE_ADDTARGET_TO_LIGHT **Value**: 88
Arguments: *Short*_source *Short*_target
Comments: This opcode assigns a target to a light. This will cause the light to always reorient itself point at the specified object regardless of their locations.

Name: OPCODE_ADDTARGET_TO_OBJECT **Value**: 89
Arguments: *Short*_source *Short*_target
Comments: This opcode assigns a target to an object. This will cause the object to always reorient itself point at the specified target regardless of their locations.

Name: OPCODE_ADDMIPMAP **Value**: 100
Arguments: *Short*_Which
Comments: This opcode indicates that the stream is changing from the presentation of opcodes to a group of wavelet coefficients corresponding to an entire MIP map level. The specification of a MIP map causes the application to enter MIP map processing mode whereupon all data is assumed to belong to the current portion of the wavelet coefficient space. It should also be noted that the initial MIP map level is presumed to be entirely an approximation, and thus is not quantized but merely provided directly in its original color values.

Name: OPCODE_ASSIGN_MAPTOOBJ **Value**: 110
Arguments: *Short*_object *Short*_texture
Comments: This opcode assigns a specific texture to a specific object. Both the texture and object must previously exist in the database.

Name: OPCODE_ADDTEXMAPTYPE_HAAR **Value**: 160
Arguments: *Float*_Min *Float*_Max *Byte*_Shininess *Word*_SpecularColor
Comments: This opcode appends a new texture map object to the database. Specifically, this texture is encoded using the Haar wavelet transform, whose coefficients are quantized between the provided min and max values. To aid in lighting calculations, a shininess value and a corresponding specular reflection color are also specified.

Name: OPCODE_ENDOFFILEMARKER **Value**: 198
Arguments: None

Comments: This opcode marks the end of the file. Any information following this opcode should be ignored.

Name: OPCODE_SETBACKGROUNDCOLOR **Value**: 240
Arguments: *Word*_ColorValue
Comments: This opcode assigns a global background color to the scene, other than the usual default value of black. Colors are represented as 16 bit color values with the original red and blue values divided by 8 and the green value divided by 4.

APPENDIX B
HEADER STRUCTURE

This section describes the header structure for files constructed by the compressor and read by the decompressor. The header structure is dynamic. Depending on switches set in the first four bytes, the length of the header can grow to include more specific information. While the rest of the file is always compressed using the LZSS algorithm, the header is never compressed and transmitted as raw data, as its statistics will not match the rest of the file.

The first four bytes are initially read and decoded according to following table:

Bit	*Information*	*Affect Header Length*
1..0	2 bit code representing how many bits per vertex component are used by this file. 00 = 32b, 01 = 16b, 10 = 10b, 11 = 8b	
2	Use simple shading?	
3	Is tick rate equal to 160	Yes
4	Is source frame rate set to 30 frames per second	Yes
5	Use high quality materials?	
6	Are there vertex normals in this file, or should the decompressor calculate them?	
7	Are included normals compressed (only applicable if bit 6 is on)	
8 ..15	Reserved for future use	
16..23	Major version	
24..31	Minor version	

When the first four bytes are being decoded, if bit 4 is off, then immediately a short value is pulled from the header and used as the tick rate. If bit 5 is off, another short value is also immediately pulled from the file and used as the source frame rate. Depending on the settings of bits 4 and 5, header length can fluctuate by up to four bytes.

Major and minor version numbers are treated as though they were numbers on the left and right side of a decimal point, forming a simple fixed point number. For example, a major value of

16 and a minor value of 32 would be equivalent to 16.125. This value is compared against a constant shared by the compressor and decompressor to confirm that versions match, as well as to confirm that the file attempting to be parsed is of the proper file type.

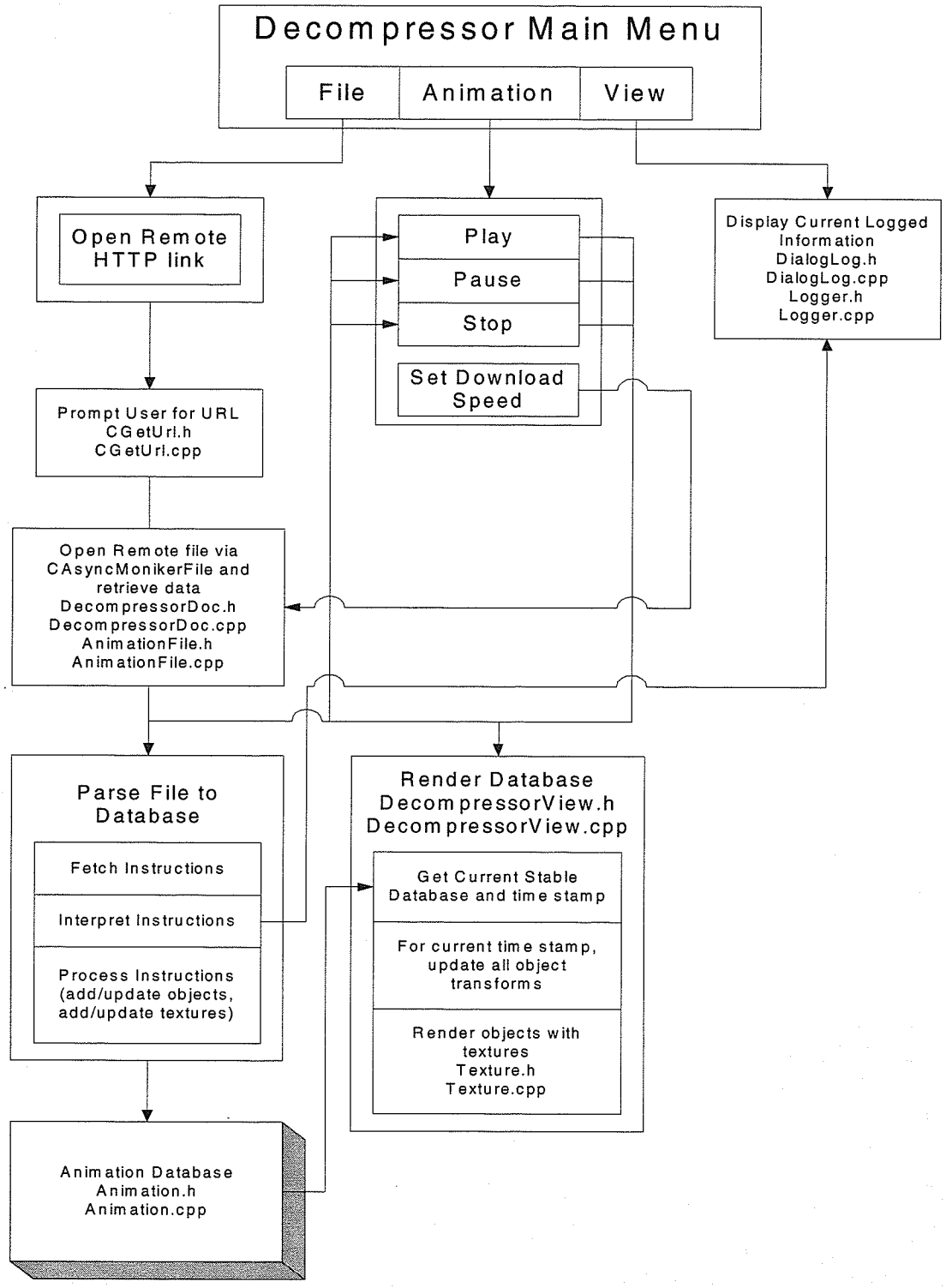
Once the initial boolean values are read in, 3 bytes are read. These three bytes represent a 24 bit unsigned integer value setting the final frame number. Total time required for the animation sequence can be determined by multiplying the final frame number by the source frame per second number.

The next three bytes in the file contain the target resolution. Resolution is retrieved from these bytes by taking the high 12 bits as the rendering height, and the low 12 bits as the rendering width. Once retrieved, the decompressor will adjust its resolution to match the target values, and center itself in the screen.

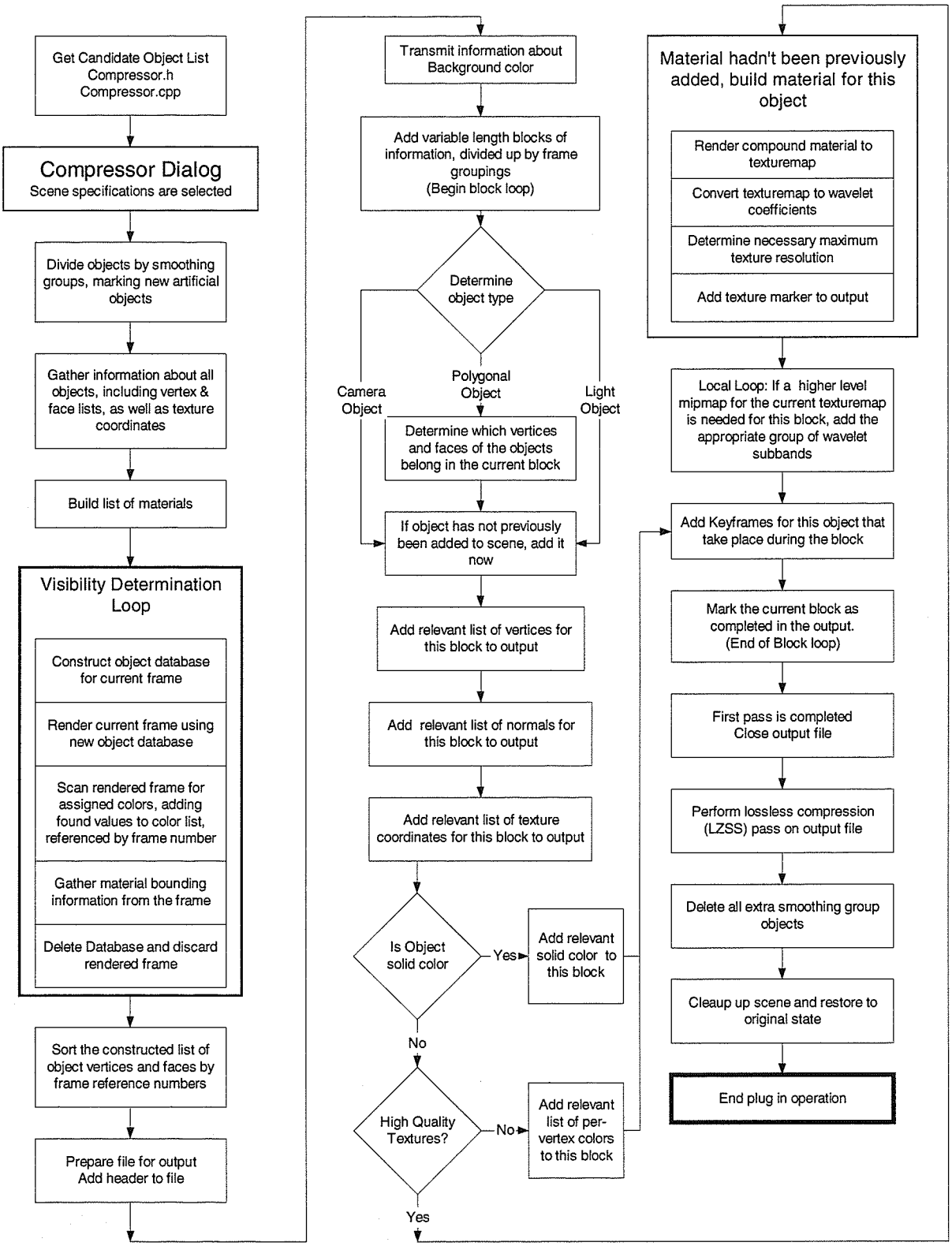
The final four bytes of the header are the length of the compressed file. As this information is not provided by the channel, and is not naturally available to the decompressor, this value is used to determine a threshold of retrieved data that must be overcome before playback will begin.

APPENDIX C
STRUCTURE CHARTS

C.1 Decompressor Structure Chart



C.2 Compressor Structure Chart



APPENDIX D
SOURCE CODE

D.1 Compressor Source Code

D.1.1 Opcodes.h

```
/*************************************************************************
 *<
      FILE:             Opcodes.h

      DESCRIPTION:      Compiler defines to aid readability

      CREATED BY:       Jonathan Greenberg

      HISTORY:          Thesis work

 *>   Copyright (c) 2000, All Rights Reserved.
 *************************************************************************/

#define OPCODE_COMPLETETOFRAME                  1
#define OPCODE_ADDVERTS                         2
#define OPCODE_ADDNORMS                         3
#define OPCODE_ADDFACES                         4
#define OPCODE_ADDTEXCOORDS                     5
#define OPCODE_ADDVERTCOLORS                    6
#define OPCODE_ASSIGNOBJECT_SOLIDCOLOR          12
#define OPCODE_ADDOBJECT                        24
#define OPCODE_SETPARENTCHILD                   28
#define OPCODE_KEYFRAME_TCBPOS                  32
#define OPCODE_KEYFRAME_LINPOS                  33
#define OPCODE_KEYFRAME_BEZPOS                  34
#define OPCODE_KEYFRAME_TCBROT                  35
#define OPCODE_KEYFRAME_LINROT                  36
#define OPCODE_KEYFRAME_TCBSCL                  38
#define OPCODE_KEYFRAME_LINSCL                  39
#define OPCODE_KEYFRAME_BEZSCL                  40
#define OPCODE_KEYFRAME_TCBRLL                  41
#define OPCODE_KEYFRAME_LINRLL                  42
#define OPCODE_KEYFRAME_BEZRLL                  43
#define OPCODE_CAMERA_KEYFRAME_TCBPOS           51
#define OPCODE_CAMERA_KEYFRAME_LINPOS           52
#define OPCODE_CAMERA_KEYFRAME_BEZPOS           53
#define OPCODE_CAMERA_KEYFRAME_TCBROT           54
#define OPCODE_CAMERA_KEYFRAME_LINROT           55
#define OPCODE_CAMERA_KEYFRAME_TCBRLL           57
#define OPCODE_CAMERA_KEYFRAME_LINRLL           58
#define OPCODE_CAMERA_KEYFRAME_BEZRLL           59
#define OPCODE_ADDCAMERA                        60
#define OPCODE_ADDCAMERA_WITHLOCATION           61
#define OPCODE_SETCAMERA_ORIENTATION            62
```

```
#define OPCODE_SETCAMERA_OFFSET                    63
#define OPCODE_SETOFFSET                           66
#define OPCODE_SETPOSITION                         67
#define OPCODE_SETROTATION                         68
#define OPCODE_SETSCALE                            69
#define OPCODE_ADDLIGHT_OMNI                       70
#define OPCODE_ADDLIGHT_SPOT                  71
#define OPCODE_ADDLIGHT_DIRECTIONAL           72
#define OPCODE_SETLIGHT_ORIENTATION           73
#define OPCODE_SETLIGHT_POSITION              74
#define OPCODE_SETLIGHT_OFFSET                     75
#define OPCODE_LIGHT_KEYFRAME_TCBPOS          76
#define OPCODE_LIGHT_KEYFRAME_LINPOS          77
#define OPCODE_LIGHT_KEYFRAME_BEZPOS          78
#define OPCODE_LIGHT_KEYFRAME_TCBROT          79
#define OPCODE_LIGHT_KEYFRAME_LINROT          80
#define OPCODE_LIGHT_KEYFRAME_BEZROT          81
#define OPCODE_LIGHT_KEYFRAME_TCBRLL          82
#define OPCODE_LIGHT_KEYFRAME_LINRLL          83
#define OPCODE_LIGHT_KEYFRAME_BEZRLL          84
#define OPCODE_ADDTARGET_TO_CAMERA            87
#define OPCODE_ADDTARGET_TO_LIGHT             88
#define OPCODE_ADDTARGET_TO_OBJECT            89
#define OPCODE_TARGET_KEYFRAME_TCBPOS         90
#define OPCODE_TARGET_KEYFRAME_LINPOS         91
#define OPCODE_TARGET_KEYFRAME_BEZPOS         92
#define OPCODE_ADDMIPMAP                           100
#define OPCODE_ASSIGN_MAPTOOBJ                     110
#define OPCODE_ENDOFFILEMARKER                     198

#define OPCODE_ENH_LIGHT_KEYFRAME_TCBROT  210
#define OPCODE_ENH_CAMERA_KEYFRAME_TCBROT 211
#define OPCODE_ENH_KEYFRAME_TCBROT            212

#define OPCODE_SETBACKGROUNDCOLOR             240

#define OPCODE_ADDTEXMAPTYPE_HAAR             160
      // Note this is to allow for the future addition of
      // more basis function types.  This can become useful
      // as more CPU time is made available with future hardware

#define MAJORVERSION                               2
#define MINORVERSION                               6

#define BPC_32    0
#define BPC_16    1
#define BPC_10    2
#define BPC_8     3
#define BPC_12  4

#define ITEMSPERBLOCK         30
//#define VIEWSCREENREDUCT    0.707f
#define VIEWSCREENREDUCT      0.767f
// Types of keyframe controllers
#define     CNT_TCBPOS  0
```

```
#define     CNT_TCBROT  1
#define     CNT_LINPOS  2
#define     CNT_LINROT  3
#define     CNT_BEZPOS  4
#define     CNT_BEZROT  5
#define     CNT_TCBRLL  6
#define     CNT_LINRLL  7
#define     CNT_BEZRLL  8
#define     CNT_TCBSCL  9
#define     CNT_LINSCL  10
#define     CNT_BEZSCL  11

#define FLOAT_PI              3.14159265358979323846264338327950f
#define FLOAT_HALFPI          1.57079632679489661923132169163980f
#define FLOAT_RAD_TO_DEG      57.29577951308232087679815481410500f
#define EPSILON 0.0001

#define LIGHT_AMBIENT         0
#define LIGHT_DIRECTIONAL     1
#define LIGHT_POINT                 2
#define LIGHT_SPOTLIGHT       3

#define MAXTEXTUREPOWER       8
#define MAXTEXTURERES         256
#define MINTEXTUREPOWER       3
#define MINTEXTURERES         8

#define TEXTYPE_NONE          0
#define TEXTYPE_HAAR          1

#define TABLESIZE             18041
#define MINBITSIZE                  9
#define MAXBITSIZE                  14

#define COMPRESSION_THRESHOLD 1.333

#define LZSSBITPOS                        14
#define LZSSBITLEN                        5
#define LZSSMAXLEN                        ((1 << LZSSBITLEN)+2)
#define LZSSWINDOW_LENGTH           (1 << LZSSBITPOS)

#define SPEED_288 0
#define SPEED_56k 1
#define SPEED_ISDN      2
#define SPEED_ADSL      3
#define SPEED_T1  4

// uncomment to disable datalogging - logging causes an initial
// burp in rendering speed
//#define DO_NOT_LOG  1
```

D.1.2 Vertcol.h

```
#include "Max.h"

#define VCOL_CLASS_ID 0x0934851
static Class_ID vcolClassID(VCOL_CLASS_ID,0);

//--------------------------------------------------------------------------
//  VCol
//--------------------------------------------------------------------------
// This file includes data to rebuild the missing Vertex Color texture map
// type that SHOULD be included in the 3DSMax SDK, but somehow they forgot.
// As we need this type to do the object database reconstruction, it has been
// included here.

#define VCOL_VERSION 1

class VCol;

class VColDlg: public ParamDlg {
      public:
            HWND hwmedit;       // window handle of the materials editor
dialog
            IMtlParams *ip;
            VCol *theTex;      // current VCol being edited.
            HWND hPanel; // Rollup pane
            TimeValue curTime;
            int isActive;
            BOOL valid;

            //-----------------------------
            VColDlg(HWND hwMtlEdit, IMtlParams *imp, VCol *m);
            ~VColDlg();
            BOOL PanelProc(HWND hDlg, UINT message, WPARAM wParam, LPARAM
lParam );

            void LoadDialog(BOOL draw);  // stuff params into dialog
            void ReloadDialog();
            void UpdateMtlDisplay() { ip->MtlChanged(); }
            void ActivateDlg(BOOL onOff);
            void Invalidate() { valid = FALSE;  InvalidateRect(hPanel,NULL,0);
}
            // methods inherited from ParamDlg:
            Class_ID ClassID() {return vcolClassID;  }
            void SetThing(ReferenceTarget *m);
            ReferenceTarget* GetThing() { return (ReferenceTarget *)theTex; }
            void DeleteThis() { delete this;  }
            void SetTime(TimeValue t);
      };

//--------------------------------------------------------------
// VCol: A Composite texture map
//--------------------------------------------------------------
class VCol: public Texmap {
      friend class VColDlg;
      VColDlg *paramDlg;
```

```
	Interval ivalid;
	BOOL useUVW;
	public:
		VCol();
		ParamDlg* CreateParamDlg(HWND hwMtlEdit, IMtlParams *imp);
		ULONG	Requirements(int	subMtlNum)	{	return
useUVW?MTLREQ_UV:MTLREQ_UV2; }
		void Update(TimeValue t, Interval& valid);
		void Reset();
		Interval Validity(TimeValue t) { Interval v; Update(t,v); return
ivalid; }
		void NotifyChanged();

		// Evaluate the color of map for the context.
		AColor EvalColor(ShadeContext& sc);
		float EvalMono(ShadeContext& sc);
		AColor EvalFunction(ShadeContext& sc, float u, float v, float du,
float dv);

		// For Bump mapping, need a perturbation to apply to a normal.
		// Leave it up to the Texmap to determine how to do this.
		Point3 EvalNormalPerturb(ShadeContext& sc);

		Class_ID ClassID() {	return vcolClassID; }
		SClass_ID SuperClassID() { return TEXMAP_CLASS_ID; }
		void GetClassName(TSTR& s) { s= _T("Vertex Color"); }
		void DeleteThis() { delete this; }

		int NumSubs() { return 0; }

		// From ref
		int NumRefs() { return 0; }

		RefTargetHandle Clone(RemapDir &remap = NoRemap());
		RefResult NotifyRefChanged( Interval changeInt,
			RefTargetHandle hTarget,
			PartID& partID, RefMessage message );

		// IO
		IOResult Save(ISave *isave);
		IOResult Load(ILoad *iload);
	};

class VColClassDesc:public ClassDesc {
	public:
	int			IsPublic() { return 1; }
	void *			Create(BOOL loading) {  return new VCol; }
	const TCHAR *	ClassName() { return _T("Vertex Color"); }
	SClass_ID		SuperClassID() { return TEXMAP_CLASS_ID; }
	Class_ID		ClassID() { return vcolClassID; }
	const TCHAR*	Category() { return TEXMAP_CAT_COLMOD;  }
	};

static VColClassDesc vcolCD;
```

D.1.3 Vertcol.cpp

```
#include "vertcol.h"
#include "resource.h"

HINSTANCE hInstance2;

void VCol::Reset() {
      useUVW = FALSE;
      ivalid.SetEmpty();
      }

void VCol::NotifyChanged() {
      NotifyDependents(FOREVER, PART_ALL, REFMSG_CHANGE);
      }

VCol::VCol() {
      paramDlg = NULL;
      Reset();
      }

AColor VCol::EvalColor(ShadeContext& sc) {
      if (gbufID) sc.SetGBufferID(gbufID);
      Point3 p = sc.UVW(1);
      return AColor(p.x,p.y,p.z,1.0f);
      }

float VCol::EvalMono(ShadeContext& sc) {
      return Intens(EvalColor(sc));
      }

Point3 VCol::EvalNormalPerturb(ShadeContext& sc) {
      return Point3(0,0,0);
      }

RefTargetHandle VCol::Clone(RemapDir &remap) {
      VCol *mnew = new VCol();
      *((MtlBase*)mnew) = *((MtlBase*)this);  // copy superclass stuff
      return (RefTargetHandle)mnew;
      }

ParamDlg* VCol::CreateParamDlg(HWND hwMtlEdit, IMtlParams *imp) {
      VColDlg *dm = new VColDlg(hwMtlEdit, imp, this);
      dm->LoadDialog(TRUE);
      paramDlg = dm;
      return dm;
      }


void VCol::Update(TimeValue t, Interval& valid) {
      if (!ivalid.InInterval(t)) {
            ivalid.SetInfinite();
            }
      valid &= ivalid;
      }
```

```
RefResult VCol::NotifyRefChanged(Interval changeInt, RefTargetHandle hTarget,
   PartID& partID, RefMessage message ) {
      switch (message) {
            case REFMSG_CHANGE:
                  ivalid.SetEmpty();
                  if (paramDlg)
                              paramDlg->Invalidate();
                  break;

            case REFMSG_GET_PARAM_DIM: {
                  GetParamDim *gpd = (GetParamDim*)partID;
                  return REF_STOP;
                  }

            case REFMSG_GET_PARAM_NAME: {
                  GetParamName *gpn = (GetParamName*)partID;
                  return REF_STOP;
                  }
            }
      return(REF_SUCCEED);
      }


#define MTL_HDR_CHUNK 0x4000
#define USE_UVW_CHUNK 0x5000

IOResult VCol::Save(ISave *isave) {
      IOResult res;
      // Save common stuff
      isave->BeginChunk(MTL_HDR_CHUNK);
      res = MtlBase::Save(isave);
      if (res!=IO_OK) return res;
      isave->EndChunk();
      if (useUVW) {
            isave->BeginChunk(USE_UVW_CHUNK);
            isave->EndChunk();
            }
      return IO_OK;
      }


IOResult VCol::Load(ILoad *iload) {
      IOResult res;
      while (IO_OK==(res=iload->OpenChunk())) {
            switch(iload->CurChunkID())  {
                  case MTL_HDR_CHUNK:
                        res = MtlBase::Load(iload);
                        break;
                  }
            iload->CloseChunk();
            if (res!=IO_OK)
                  return res;
            }
      return IO_OK;
      }

static BOOL CALLBACK   PanelDlgProc(HWND hwndDlg, UINT msg, WPARAM wParam,
```

```
LPARAM lParam) {
      VColDlg *theDlg;
      if (msg==WM_INITDIALOG) {
            theDlg = (VColDlg*)lParam;
            theDlg->hPanel = hwndDlg;
            SetWindowLong(hwndDlg, GWL_USERDATA,lParam);
            }
      else {
          if ( (theDlg = (VColDlg *)GetWindowLong(hwndDlg, GWL_USERDATA) ) ==
NULL )
                return FALSE;
            }
      theDlg->isActive = 1;
      int   res = theDlg->PanelProc(hwndDlg,msg,wParam,lParam);
      theDlg->isActive = 0;
      return res;
      }

VColDlg::VColDlg(HWND hwMtlEdit, IMtlParams *imp, VCol *m) {
      hwmedit = hwMtlEdit;
      ip = imp;
      hPanel = NULL;
      theTex = m;
      isActive = 0;
      valid = FALSE;
      hPanel = ip->AddRollupPage(
            hInstance2,
            MAKEINTRESOURCE(IDD_VCOL),
            PanelDlgProc,
            _T("Vertex Color Parameters"),
            (LPARAM)this );
      curTime = imp->GetTime();
      }

void VColDlg::ReloadDialog() {
      Interval valid;
      theTex->Update(curTime, valid);
      LoadDialog(FALSE);
      }

void VColDlg::SetTime(TimeValue t) {
      Interval valid;
      if (t!=curTime) {
            curTime = t;
            theTex->Update(curTime, valid);
            LoadDialog(FALSE);
            InvalidateRect(hPanel,NULL,0);
            }
      }

VColDlg::~VColDlg() {
      theTex->paramDlg = NULL;
      SetWindowLong(hPanel, GWL_USERDATA, NULL);
      }


BOOL VColDlg::PanelProc(HWND hwndDlg, UINT msg, WPARAM wParam, LPARAM lParam )
{
```

```
    int id = LOWORD(wParam);
    int code = HIWORD(wParam);
  switch (msg)    {
        case WM_INITDIALOG:
              {
              ShowWindow(GetDlgItem(hwndDlg,IDC_VC_VC),SW_HIDE);
              ShowWindow(GetDlgItem(hwndDlg,IDC_VC_UVW),SW_HIDE);
              return TRUE;
              }
              break;
        case WM_COMMAND:

              break;
        case WM_PAINT:
              if (!valid) {
                    valid = TRUE;
                    ReloadDialog();
                    }
              break;
        case WM_CLOSE:    break;
        case WM_DESTROY:        break;
        case CC_SPINNER_CHANGE:
              break;
        case WM_CUSTEDIT_ENTER:
        case CC_SPINNER_BUTTONUP:
              theTex->NotifyChanged();
            UpdateMtlDisplay();
              break;

    }
    return FALSE;
    }


void VColDlg::LoadDialog(BOOL draw) {
    if (theTex) {
          Interval valid;
          theTex->Update(curTime,valid);
          }
    }

void VColDlg::SetThing(ReferenceTarget *m) {
    assert (m->ClassID()==vcolClassID);
    assert (m->SuperClassID()==TEXMAP_CLASS_ID);
    if (theTex) theTex->paramDlg = NULL;
    theTex = (VCol *)m;
    if (theTex) theTex->paramDlg = this;
    LoadDialog(TRUE);
    }

void VColDlg::ActivateDlg(BOOL onOff) {
    }
```

D.1.4 PixelPeano.h

```
// PixelPeano.h: interface for the CPixelPeano class.
//
// This file defines a structure for use with traversing Hilbert
// curves of some given order.
//////////////////////////////////////////////////////////////////////

#if
!defined(AFX_PIXELPEANO_H__1E990F53_8429_11D3_9002_00AA00B9C442__INCLUDED_)
#define AFX_PIXELPEANO_H__1E990F53_8429_11D3_9002_00AA00B9C442__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

enum {
      PIXEL_UP,
      PIXEL_LEFT,
      PIXEL_RIGHT,
      PIXEL_DOWN
};


class CPixelPeano
{
public:
      Reset();
      Switch();
      CPixelPeano(int scale, BOOL bMiddle);
      BOOL Process();
      inline Push(unsigned int iCommand,unsigned int iLevel)
      {
            CommandStack[iStackPointer] = iCommand;
            LevelStack[iStackPointer++] = iLevel;
      }
      inline BOOL Empty()
      {
            if (iStackPointer == 0) return TRUE;
            return FALSE;
      }
      inline BOOL Pop(unsigned int &iCommand,unsigned int &iLevel)
      {
            if (Empty()) return FALSE;
            iCommand = CommandStack[--iStackPointer];
            iLevel = LevelStack[iStackPointer];
            return TRUE;
      }
      CPixelPeano();
      virtual ~CPixelPeano();

      unsigned int coordx,coordy;

private:
      int iStackPointer;
      unsigned char CommandStack[512];
```

```
        unsigned char LevelStack[512];
        int iSecondStackPointer;
        unsigned char SecondCommandStack[512];
        unsigned char SecondLevelStack[512];
        BOOL bUseSecond;
        BOOL _bMiddle;
        int _scale;
};

#endif                                                                    //
!defined(AFX_PIXELPEANO_H__1E990F53_8429_11D3_9002_00AA00B9C442__INCLUDED_)
```

D.1.5 PixelPeano.cpp

```
// PixelPeano.cpp: implementation of the CPixelPeano class.
//
//////////////////////////////////////////////////////////////////////

#include "Max.h"
#include "PixelPeano.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CPixelPeano::CPixelPeano()
{
      coordx = coordy = 0;
      iStackPointer = 0;
      bUseSecond = FALSE;
}

CPixelPeano::Reset()
{
      bUseSecond = FALSE;
      if (_bMiddle)
      {
            iStackPointer = 0;
            Push(PIXEL_RIGHT,_scale-1);
            Push(PIXEL_UP,0);
            Push(PIXEL_UP,_scale-1);
            Push(PIXEL_RIGHT,0);
            Push(PIXEL_UP,_scale-1);
            coordx = 0; coordy = 1;
            for (int i=0;i<_scale-1;i++)
                  coordy*=2;
      }
      else
      {
            coordx = coordy = 0;
            iStackPointer = 0;
            Push(PIXEL_UP,_scale);
      }
}

CPixelPeano::CPixelPeano(int scale, BOOL bMiddle = TRUE)
{
      bUseSecond = FALSE;
      if (bMiddle)
      {
            iStackPointer = 0;
            Push(PIXEL_RIGHT,scale-1);
            Push(PIXEL_UP,0);
```

```
                Push(PIXEL_UP,scale-1);
                Push(PIXEL_RIGHT,0);
                Push(PIXEL_UP,scale-1);
                coordx = 0; coordy = 1;
                for (int i=0;i<scale-1;i++)
                      coordy*=2;
        }
        else
        {
                coordx = coordy = 0;
                iStackPointer = 0;
                Push(PIXEL_UP,scale);
        }
        _scale = scale;
        _bMiddle = bMiddle;
}

CPixelPeano::~CPixelPeano()
{

}

void swap(int &a, int &b)
{
        int t = a;
        a = b;
        b = t;
}

CPixelPeano::Switch()
{
        int count;
        if (bUseSecond)
        {
                // restoring backup
                count = iSecondStackPointer;
                iStackPointer = iSecondStackPointer;
                for (int i=0;i<count;i++)
                {
                        CommandStack[i] = SecondCommandStack[i];
                        LevelStack[i] = SecondLevelStack[i];
                }
        }
        else
        {
                // backup current value
                count = iStackPointer;
                iSecondStackPointer = iStackPointer;
                for (int i=0;i<count;i++)
                {
                        SecondCommandStack[i] = CommandStack[i];
                        SecondLevelStack[i] = LevelStack[i];
                }
        }
        bUseSecond = !bUseSecond;
}

BOOL CPixelPeano::Process()
{
```

```
unsigned int level, command;
if (!Pop(command,level)) return FALSE;
if (level == 0)
{
      switch(command)
      {
      case PIXEL_LEFT:
            coordx--; break;
      case PIXEL_RIGHT:
            coordx++; break;
      case PIXEL_UP:
            coordy--; break;
      case PIXEL_DOWN:
            coordy++; break;
      }
      return FALSE;
}
else if (level == 1)
{
      switch(command)
      {
            case PIXEL_LEFT:
            {
                  Push(PIXEL_LEFT,0);
                  Push(PIXEL_DOWN,0);
                  Push(PIXEL_RIGHT,0);
            } break;
            case PIXEL_RIGHT:
            {
                  Push(PIXEL_RIGHT,0);
                  Push(PIXEL_UP,0);
                  Push(PIXEL_LEFT,0);
            } break;
            case PIXEL_UP:
            {
                  Push(PIXEL_UP,0);
                  Push(PIXEL_RIGHT,0);
                  Push(PIXEL_DOWN,0);
            } break;
            case PIXEL_DOWN:
            {
                  Push(PIXEL_DOWN,0);
                  Push(PIXEL_LEFT,0);
                  Push(PIXEL_UP,0);
            } break;
      }
}
else switch(command)
{
      case PIXEL_LEFT:
      {
            Push(PIXEL_DOWN,level-1);
            Push(PIXEL_LEFT,0);
            Push(PIXEL_LEFT,level-1);
            Push(PIXEL_DOWN,0);
            Push(PIXEL_LEFT,level-1);
            Push(PIXEL_RIGHT,0);
```

```
                Push(PIXEL_UP,level-1);
            } break;
            case PIXEL_RIGHT:
            {
                Push(PIXEL_UP,level-1);
                Push(PIXEL_RIGHT,0);
                Push(PIXEL_RIGHT,level-1);
                Push(PIXEL_UP,0);
                Push(PIXEL_RIGHT,level-1);
                Push(PIXEL_LEFT,0);
                Push(PIXEL_DOWN,level-1);
            } break;
            case PIXEL_UP:
            {
                Push(PIXEL_RIGHT,level-1);
                Push(PIXEL_UP,0);
                Push(PIXEL_UP,level-1);
                Push(PIXEL_RIGHT,0);
                Push(PIXEL_UP,level-1);
                Push(PIXEL_DOWN,0);
                Push(PIXEL_LEFT,level-1);
            } break;
            case PIXEL_DOWN:
            {
                Push(PIXEL_LEFT,level-1);
                Push(PIXEL_DOWN,0);
                Push(PIXEL_DOWN,level-1);
                Push(PIXEL_LEFT,0);
                Push(PIXEL_DOWN,level-1);
                Push(PIXEL_UP,0);
                Push(PIXEL_RIGHT,level-1);
            } break;
        }
        return TRUE;
}
```

D.1.6 Resource.h

```
// Microsoft Developer Studio generated include file.
// Used by Compressor.rc
//
#define IDS_LIBDESCRIPTION              1
#define IDS_CATEGORY                    2
#define IDS_CLASS_NAME                  3
#define IDS_PARAMS                      4
#define IDS_PROGRESS_MSG0               5
#define IDS_PROGRESS_MSG1               6
#define IDS_PROGRESS_MSG2               7
#define IDS_PROGRESS_MSG3               8
#define IDS_PROGRESS_MSG4               9
#define IDS_PROGRESS_MSG5               10
#define IDD_PANEL                       101
#define IDD_COMPRESSDIALOG              102
#define IDD_OPENGLWINDOW                103
#define IDC_CLOSEBUTTON                 1000
#define IDC_DOSTUFF                     1000
#define IDC_XRES                        1003
#define IDC_YRES                        1004
#define IDC_RADIOLOWQ                   1006
#define IDC_RADIOHIGHQ                  1007
#define IDC_COMBOCAMERAS                1008
#define IDC_FRAMERATE                   1009
#define IDC_STARTFRAME                  1013
#define IDC_ENDFRAME                    1014
#define IDC_VERTNORM                    1015
#define IDC_COMPNORM                    1016
#define IDC_RADIO16B                    1018
#define IDC_RADIO10B                    1019
#define IDC_RADIO8B                     1021
#define IDC_RADIO32B                    1022
#define IDC_DIRLIGHT                    1023
#define IDC_SLIDER_THRESHOLD            1024
#define IDC_SLIDER_TEXRES               1026
#define IDD_VCOL                        1061
#define IDC_VC_VC                       1254
#define IDC_VC_UVW                      1255
#define IDC_COLOR                       1456
#define IDC_EDIT                        1490
#define IDC_SPIN                        1496

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        105
#define _APS_NEXT_COMMAND_VALUE         40001
#define _APS_NEXT_CONTROL_VALUE         1026
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

D.1.7 Compressor.h

```
/**********************************************************************
 *<
      FILE:             Compressor.h

      DESCRIPTION:      Template Utility

      CREATED BY:       Jonathan Greenberg

      HISTORY:          Thesis work

 *>   Copyright (c) 2000, All Rights Reserved.
 **********************************************************************/

#ifndef __COMPRESSOR__H
#define __COMPRESSOR__H

#include "Max.h"
#include "resource.h"
#include "istdplug.h"
#include "Opcodes.h"

#include "iparamm.h"

#define COMPRESSOR_CLASS_ID   Class_ID(0x52024799, 0x7235b712)
#define CURRENT_DESCRIPTOR descVer1
#define PBLOCK_LENGTH   1
#define CURRENT_VERSION 1
#define DBGMODE 1
#define MAXOBJECTS 8192
#define DEFAULT_XRES 640
#define DEFAULT_YRES 480

TCHAR *GetString(int id);

extern ClassDesc* GetCompressorDesc();

extern HINSTANCE hInstance;

// externally accessible globals
int iSampleYres;
int iSampleXres;
int iFrameRate;
int iEndFrame;
int iStartFrame;
BOOL bMaterialQuality;
BOOL bVertexNormals;
BOOL bCompressNormals;
BOOL bUseShading;
int iBitsPerVertex;

//typedef FCNINVWAVELET (float [], float [], unsigned int);
typedef void (* FCNINVWAVELET )(float [], float [], unsigned int);

typedef struct TriFace {
```

```
        int Vert1;
        int Vert2;
        int Vert3;
} _TriFace;

typedef struct Vert {
        float x;
        float y;
        float z;
} _Vert;

typedef struct FrameList {
        FrameList *next;
        int frameNum;
public:
        int AddedRefNumber;
} _FrameList;

class GeoNode {
public:
        INode*              curNode;
        BOOL        IsHelper;
        BOOL        IsTarget;
        int                 ParentIdx;
        BOOL        IsChild;
        TriFace*    TrueFaceList;
        int*        TexIndexList;
        int*        AddedFaceList;
        int*        FaceAddedOnFrame;
        int*        AddedVertList;
        int*        VertAddedOnFrame;
        int                 nTrueFaceCount;
        int                 nTrueVertexCount;
        int                 nAddedFaceCount;
        int                 nAddedVertCount;
public:
        BOOL bKillMe;
        BOOL IsSuperObject();
        BOOL bMaterialAssigned;
        InitBoundingBox();
        int iMinX,iMinY,iMaxX,iMaxY;
        int LastRllKeySent;
        BOOL IsAssignedTarget;
        int TargetIdx;
        INode* TargetNode;
        int LastPosKeySent;
        int LastRotKeySent;
        int LastSclKeySent;
        Point3 pMaxCoord;
        Point3 pMinCoord;
        int RefNumber;
        int nSentFace;
        int nSentVert;
        int MaterialID;
        GeoNode();
        ~GeoNode();
```

```
};

class MatNode {
public:
      int RefNumber;
      MatNode();
      ~MatNode();
      Mtl* curNode;
      unsigned char* pTextureMap;
      float fTexmapMinVal;
      float fTexmapMaxVal;
      float* WaveletCoeff;
      int iMipNeededOnFrame[16];
      BOOL bMipSent[16];
      int iLastTexturePowerSent;
      int iMaxNecessaryTexPower;
};

typedef struct ListTriangle {
      float V0[3];
      float V1[3];
      float V2[3];
      unsigned char color[4];
      float Zdist;
} _ListTriangle;

class Compressor : public SceneExport {
      public:
            float fTotalError;
            int   iTotalVerts;
            pbOutToFile(unsigned int code,int bitlength, BOOL bFinish =
FALSE);
            BOOL LZSS_OutputFile2(int headersize, const TCHAR *filename);

            int iSampleRate;
            void LZSS_OutputCode(FILE *fo, unsigned int code,int bitlength,
                  int &OutputBitCount,unsigned int &OutputBitBuffer);
            void  LZSS_Scan(unsigned char * Dictionary,unsigned char *
LookAhead,
                  unsigned int &position,unsigned int &length);
            BOOL LZSS_OutputFile(int headersize, const TCHAR * filename);
            unsigned int iUnCompressedCount;
            unsigned int iCompressedCount;
            BOOL IsMarkedForDeletion(GeoNode *node);
            MarkNodeForDeletion(GeoNode *node);
            int iSuperObjCount;
            INode* SuperObjects[MAXOBJECTS];
            BOOL BuildSubObjectList();
            BuildMaterialList();
            BOOL _BForce2Side;
            Point3  RetrieveTexelColor(float  U,  float  V,  unsigned  char
*Texture);
            int iDesiredTexturePower;
            float fTextureQualityThreshold;
            DetermineNecessaryTexRes(MatNode* node);
            SortMipMapAppearanceList(int frame);
```

```
            MarkObjBoundingBox(int x, int y, int ObjNum);
            unsigned char WaveletQuantize(float min, float max, float coeff);
            GetCoeffRange(float *coeffs,float &min,float &max);
            DeltaEncode(float *coeff,int size,int dim);
            void HaarInv(float a[], float source[], unsigned int size);
            void DetermineThreshold(float *coeffs, unsigned char* map, float
targetPSNR);
            void HaarTR(float a[], float source[], unsigned int size);
            Daub4TR(float a[], float source[], unsigned int size);
            RetrieveOriginalMap(MatNode* node,unsigned char* map);
            BuildWaveletCoeffs(MatNode *node);
            AddTextureCoeffs(MatNode *node);
            aaOutToFile(AngAxis aa);
            Matrix3 GetLocalMatrix(INode *node,TimeValue t);
            MarkTargets();
            void CreateNewObject(GeoNode* node);
            TransmitTargetKeysThisSecond(GeoNode *node, int frame, int
&LastTKeyNum);
            BOOL bPerspView;
            INode* OldCamera;
            Matrix3 VptTM;
            CleanUpRendering(int framenumber, int SampleRate, int Sample);
            ViewExp* ViewPort;
            BuildIndexList(INode** PtrList,int *IndexList,int count);
            cOutToFile(unsigned char cval);
            CloseOutputFile();
            int LinVertSearch(GeoNode *node, int VertNum);
            quatOutToFile(Quat q);
            MarkCompleteToFrame(int frame);
            unsigned long BytesSoFar;
            FrameList * GetFramePtrFromTime(int t);
            BOOL bObjectsAddedThisFrame;
            FrameList * frHead;
            TransmitKeysThisSecond(GeoNode *node, int frame);
            wOutToFile(int ival);
            FILE* fileOut;
            int InitFileForOutput(const TCHAR * filename);
            void fOutToFile(float f);
            iOutToFile(int i);
            int nLastRefNum;
            int nLastMatRefNum;
            void TransmitPolyBlock(GeoNode *node,int startFace, int startVert,
int FinalFrameThisBlock);
            void BuildDataBlock(int uptoframe);
            void SortVertListbyFrame(GeoNode *node,int lo,int hi);
            void SortFaceListbyFrame(GeoNode *node,int lo,int hi);
            void SortByFrame(GeoNode *node);
            void InsertVertex(GeoNode *node, int Vert, int Frame, int loc);
            int VertSearch(GeoNode *node, int VertNum);
            void AddVertices(GeoNode *node,int Face,int Frame);
            void InsertFace(GeoNode *node,int Face,int Frame, int loc);
            int FaceSearch(GeoNode *node, int FaceNum);
            void AddFaceToList(int ObjNum, int FaceNum, int FrameNum);
            BOOL  _BUseEnvMap;
            Point3      _p3OldAmbient;
            float _fOldLightLevel;
```

```
            Point3      _p3OldTint;
            Point3  _p3OldBackgrd;
            void  PrepareForRendering(int  framenumber,  int  SampleRate,  int
Sample);
            void RenderCurrentFrame(int frame, Bitmap *bm, int SampleRate, int
Sample);

            INode*  ConstructObjectDatabase(int  frame,  int  SampleRate,  int
Sample);
            void ScanRenderForFaces(Bitmap* bm, int frame);
            BOOL BuildVertexFaceList();
            int GetGeometryNodeIndex(INode *node);
            int iCurrentCount;
            BOOL nodeEnum(INode *node);
             DummyCount(INode* node);
            int iTotalNodeCount;
             SortObjectList(ExpInterface *ei, Interface *gi, Compressor *exp);
             Compressor();
             ~Compressor();
            int iGeomCount;
            int iMatCount;
            int iCamCount;
            int iLightCount;
            int nSelectedCamera;
            GeoNode GeometryList[MAXOBJECTS];
            MatNode MaterialList[MAXOBJECTS];
            INode* CameraPtrList[MAXOBJECTS];
            int   CameraIndexList[MAXOBJECTS];
            INode* LightPtrList[MAXOBJECTS];
            int   LightIndexList[MAXOBJECTS];
            int LightPtrIndexList[MAXOBJECTS];

            friend  BOOL  CALLBACK  CompressorOptionsDlgProc(HWND  hDlg,  UINT
message, WPARAM wParam, LPARAM lParam);
            int  CompressScene(const  TCHAR  *filename,  ExpInterface  *ei,
Interface *gi, Compressor *exp);
            int ExtCount() {return 1;}
            const TCHAR *Ext(int i) {if (i==0) return _T("PRT"); else return
_T("");}
            const TCHAR *LongDesc()                    {return  _T("Progressive
Real Time 3D Animation file");}
            const TCHAR *ShortDesc()                   {return _T("PRT file");}
            const TCHAR *AuthorName()                  {return     _T("Jonathan
Greenberg"); }
            const TCHAR *CopyrightMessage()            {return    _T("Copyright
(c) 1999"); }
            const TCHAR *OtherMessage1()         {return    _T("Developed    for
TRLabs and MeccaMedia"); }
            const TCHAR *OtherMessage2()         {return _T(""); }
            unsigned int Version()                     {return 11; }
            void ShowAbout(HWND hWnd);
            int DoExport(const TCHAR *name,ExpInterface *ei,Interface *i, BOOL
suppressPrompts=FALSE);
};

class CompressorClassDesc:public ClassDesc {
      public:
```

```
      int               IsPublic() {return 1;}
      void *                  Create(BOOL loading = FALSE) {return new
Compressor();}
      const TCHAR *     ClassName() {return GetString(IDS_CLASS_NAME);}
      SClass_ID         SuperClassID() {return SCENE_EXPORT_CLASS_ID;}
      Class_ID          ClassID() {return COMPRESSOR_CLASS_ID;}
      const TCHAR*      Category() {return GetString(IDS_CATEGORY);}
      void              ResetClassParams (BOOL fileReset);
};

// Handy file class

class WorkFile {
private:
      FILE *stream;
public:
                              WorkFile(const TCHAR *filename,const TCHAR
*mode) { stream = NULL; Open(filename, mode); };
                              ~WorkFile() { Close(); };
      FILE *                  Stream() { return stream; };
      int                     Close()     {     int     result=0;     if(stream)
result=fclose(stream); stream = NULL; return result; }
      void                Open(const TCHAR *filename,const TCHAR *mode) {
Close(); stream = _tfopen(filename,mode); }
};


#endif // __COMPRESSOR__H
```

D.1.8 Compressor.cpp

```
/**********************************************************************
 *<
      FILE:             Compressor.cpp

      DESCRIPTION:      Appwizard generated plugin

      CREATED BY:       Jonathan Greenberg

      HISTORY:          Thesis work

 *>   Copyright (c) 2000, All Rights Reserved.
 **********************************************************************/

#include "vertcol.h"
#include "Compressor.h"
#include "Opcodes.h"
#include "modstack.h"
#include "mnmath.h"
#include "stdmat.h"
#include "bmmlib.h"
#include "math.h"
#include "decomp.h"
#include "spline3d.h"
#include "PixelPeano.h"

ClassDesc* GetCompressorDesc();
Interface *intPtr;

#define MIN(a,b)    ((a) < (b)) ? (a) : (b)
#define BIGCONSTVALUE 0x7fffffff
#define WM_PAGEFLIP           WM_USER+1

HINSTANCE hInstance;
int controlsInit = FALSE;
static BOOL showPrompts;
HDC         hDC;
HWND        hWnd;

ClassDesc* GetVColDesc() { return &vcolCD;  }

VCol  *NewDefaultVColTex()  {  return  (VCol*)CreateInstance(TEXMAP_CLASS_ID,
Class_ID(VCOL_CLASS_ID,0)); }

// This function is called by Windows when the DLL is loaded.  This
// function may also be called many times during time critical operations
// like rendering.  Therefore developers need to be careful what they
// do inside this function.  In the code below, note how after the DLL is
// loaded the first time only a few statements are executed.

BOOL WINAPI DllMain(HINSTANCE hinstDLL,ULONG fdwReason,LPVOID lpvReserved)
{
      hInstance = hinstDLL;                          // Hang on to this DLL's
instance handle.
```

```
        if (!controlsInit) {
                controlsInit = TRUE;
                InitCustomControls(hInstance);      //  Initialize  MAX's  custom
controls
                InitCommonControls();               // Initialize Win95 controls
        }

        return (TRUE);
}

// This function returns a string that describes the DLL and where the user
// could purchase the DLL if they don't have it.
__declspec( dllexport ) const TCHAR* LibDescription()
{
        return GetString(IDS_LIBDESCRIPTION);
}

// This function returns the number of plug-in classes this DLL
//TODO: Must change this number when adding a new class
__declspec( dllexport ) int LibNumberClasses()
{
        return 1;
}

// This function returns the number of plug-in classes this DLL
__declspec( dllexport ) ClassDesc* LibClassDesc(int i)
{
        switch(i) {
                case 0: return GetCompressorDesc();
                default: return 0;
        }
}

// This function returns a pre-defined constant indicating the version of
// the system under which it was compiled.  It is used to allow the system
// to catch obsolete DLLs.
__declspec( dllexport ) ULONG LibVersion()
{
        return VERSION_3DSMAX;
}

TCHAR *GetString(int id)
{
        static TCHAR buf[256];

        if (hInstance)
                return LoadString(hInstance, id, buf, sizeof(buf)) ? buf : NULL;
        return NULL;
}



//////////////////////////////////////////////////////////////

static CompressorClassDesc CompressorDesc;
ClassDesc* GetCompressorDesc() {return &CompressorDesc;}
```

```
//TODO: Should implement this method to reset the plugin params when Max is
reset
void CompressorClassDesc::ResetClassParams (BOOL fileReset)
{

}

GeoNode::GeoNode()
{
      curNode = NULL;
      IsChild = FALSE;
      IsHelper = FALSE;
      IsTarget = FALSE;
      ParentIdx = -1;
      TrueFaceList = NULL;
      AddedFaceList = NULL;
      AddedVertList = NULL;
      FaceAddedOnFrame = NULL;
      VertAddedOnFrame = NULL;
      TexIndexList = NULL;
      nTrueFaceCount = 0;
      nTrueVertexCount = 0;
      nAddedFaceCount = 0;
      nAddedVertCount = 0;
      nSentVert = -1;
      nSentFace = -1;
      RefNumber = -1;
      pMaxCoord.x = pMaxCoord.y = pMaxCoord.z = -1e38f;
      pMinCoord.x = pMinCoord.y = pMinCoord.z =  1e38f;
      LastPosKeySent = 0;
      LastRotKeySent = 0;
      LastRllKeySent = 0;
      LastSclKeySent = -1;
      IsAssignedTarget = FALSE;
      MaterialID = -1;

      bMaterialAssigned = FALSE;
      bKillMe = FALSE;
}

GeoNode::~GeoNode()
{
      if (nTrueVertexCount > 0)
      {
            if (VertAddedOnFrame != NULL)
                  delete[] VertAddedOnFrame;
            if (AddedVertList != NULL)
                  delete[] AddedVertList;
      }
      if (nTrueFaceCount > 0)
      {
            delete[] TrueFaceList;
            if (AddedFaceList != NULL)
                  delete[] AddedFaceList;
            if (FaceAddedOnFrame != NULL)
                  delete[] FaceAddedOnFrame;
      }
```

```
        TrueFaceList = NULL;
        AddedFaceList = NULL;
        AddedVertList = NULL;
        FaceAddedOnFrame = NULL;
        VertAddedOnFrame = NULL;

        IsChild = FALSE;
        IsHelper = FALSE;
        ParentIdx = -1;
        nTrueFaceCount = 0;
        nTrueVertexCount = 0;
        nAddedFaceCount = 0;
        nAddedVertCount = 0;

//      if (bKillMe && curNode)
//            curNode->Delete(0,0);

        curNode = NULL;
}

GeoNode::InitBoundingBox()
{
        iMaxX = iMaxY = -65535;
        iMinX = iMinY =  65535;
}

BOOL GeoNode::IsSuperObject()
{
        Object *obj = curNode->EvalWorldState(0).obj;
        TriObject *Tri;
        if (IsHelper)
              Tri = NULL;
        else                    Tri                     =                    (TriObject*)
obj->ConvertToType(0,Class_ID(TRIOBJ_CLASS_ID,0));
        if (Tri && Tri->mesh.numFaces > 0)
        {
              int i=1;
              DWORD smooth = Tri->mesh.faces[0].smGroup;
              while(i<Tri->mesh.numFaces)
                    if ((Tri->mesh.faces[i++].smGroup & smooth) == 0)
                          return TRUE;
        }
//      StdMat* nodeMtl= (StdMat*) curNode->GetMtl();
//      if (!(!nodeMtl || nodeMtl->GetWire()))
//            return TRUE;
        return FALSE;
}

MatNode::MatNode()
{
      RefNumber = -1;
      curNode = NULL;
      iLastTexturePowerSent = -1;
      WaveletCoeff = NULL;
      for (int i=0;i<16;i++)
      {
            iMipNeededOnFrame[i] = 2147483647;
            bMipSent[i] = FALSE;
```

```
        }
                    // max value for an unsigned 32b int
        iMaxNecessaryTexPower = MAXTEXTUREPOWER;
        pTextureMap = NULL;
}

MatNode::~MatNode()
{
        if (WaveletCoeff)
                delete[] WaveletCoeff;
        if (pTextureMap)
                delete[] pTextureMap;
}

////////////////////////////////////////////////////////////////////////
/// Utility functions

float round(float f)
// rounds a positive float to the nearest integer
{
        if (f - 0.5f < floorf(f))
                return (floorf(f));
        else return (ceilf(f));
}

unsigned int ReScaleFloat(float val, float min, float max, int bitdepth)
// rescales a floating point value into the specified integer range and
// returns the scaled value
{
        float dif1 = val - min;
        float dif2 = max - min;
        float perc = dif1 / dif2;
        float scale;
        switch (bitdepth)
        {
                case BPC_16: scale = 65535.0; break;
                case BPC_12: scale = 4095.0; break;
                case BPC_10: scale = 1023.0; break;
                case BPC_8: scale = 255.0; break;
        }
        return ( (unsigned int) round(perc * scale) );
}

float ScaleUp(unsigned int val, float min, float max, int bitdepth)
{
        float scale;
        float fval = (float) val;
        switch (bitdepth)
        {
                case BPC_16: scale = 65535.0f; break;
                case BPC_12: scale = 4095.0f; break;
                case BPC_10: scale = 1023.0f; break;
                case BPC_8:  scale = 255.0f; break;
        }
        float perc = fval / scale;
        return ( min + perc * (max - min)  );
```

```
}

unsigned short Convert_16bColor(COLORREF col)
// converts a COLORREF value to a 16 bit color value
{
      unsigned char r,g,b;
      r = GetRValue(col);
      g = GetGValue(col);
      b = GetBValue(col);
      r = r >> 3;
      g = g >> 2;
      b = b >> 3;
      unsigned short outp = (r << 11) + (g << 5) + b;
      return (outp);
}

COLORREF ConvertRGBtoColorref(Point3 col)
// converts a 3-part float color value to a 24b packed dword value
{
      unsigned int color = 0;
      unsigned char * ar = (unsigned char*)&color;
      ar[0] = (unsigned char) (col.x * 255.0f);
      ar[1] = (unsigned char) (col.y * 255.0f);
      ar[2] = (unsigned char) (col.z * 255.0f);
      return color;
}

Point3 Transf(const Matrix3& M, const Point3& V)
// transforms vector V by the matrix M
{
      Point3 r;
      r.x = M.GetRow(0)[0]*V.x + M.GetRow(1)[0]*V.y + M.GetRow(2)[0]*V.z +
M.GetRow(3)[0];
      r.y = M.GetRow(0)[1]*V.x + M.GetRow(1)[1]*V.y + M.GetRow(2)[1]*V.z +
M.GetRow(3)[1];
      r.z = M.GetRow(0)[2]*V.x + M.GetRow(1)[2]*V.y + M.GetRow(2)[2]*V.z +
M.GetRow(3)[2];
      return (r);
}

BOOL CheckIdentity(const Matrix3& M)
// returns true if M is an identity Matrix.. allows for a little more leaway
// than the built in routine
{
      float sum;
      sum= M.GetRow(0)[1] + M.GetRow(0)[2] + M.GetRow(1)[0] + M.GetRow(1)[2] +
            M.GetRow(2)[0] + M.GetRow(2)[1] + M.GetRow(3)[0] + M.GetRow(3)[1]
+
            M.GetRow(3)[2];
      if (fabsf(sum) < 0.001f && (fabsf(M.GetRow(0)[0] - 1.0f) < 0.001f) &&
            (fabsf(M.GetRow(1)[1] - 1.0f) < 0.001f) &&
            (fabsf(M.GetRow(2)[2] - 1.0f) < 0.001f))
            return TRUE;
      else return FALSE;
}

FileDump(char *filename, unsigned char *Data, int size, int depth)
```

```
// a diagnostic routine. dumps a bitmap to a file
{
      FILE *fout;
      if ((fout = fopen(filename, "wb")) == NULL)
      {
            fprintf (stderr, "Cannot open output file.\n");
            return 0;
      }
      unsigned char *D = Data;
      for (int y=0;y<size;y++)
            for (int x=0;x<size;x++)
                  for (int c=0;c<depth;c++)
                        fputc(*(D++),fout);
      fflush(fout);
      fclose(fout);
      return 0;
}

FileFloatDump(char *filename, float *Data, int size, int depth)
// a diagnostic routine: dumps an image of floats to a file
{
      FILE *fout;
      if ((fout = fopen(filename, "wb")) == NULL)
      {
            fprintf (stderr, "Cannot open output file.\n");
            return 0;
      }
      float *D = Data;
      for (int y=0;y<size;y++)
            for (int x=0;x<size;x++)
                  for (int c=0;c<depth;c++)
                        fputc((unsigned char)*(D++),fout);
      fflush(fout);
      fclose(fout);
      return 0;
}

TriObject* GetTriObjectFromNode(INode * node, int & deleteIt)
// generates a triangle object given a 3DS node. deleteIT specifies
// that cleanup is needed
{
      deleteIt = FALSE;
      Object *obj = node->EvalWorldState(0).obj;
      if (obj->CanConvertToType(Class_ID(TRIOBJ_CLASS_ID, 0)))
      {
            TriObject *tri = (TriObject *) obj->ConvertToType(0,
                  Class_ID(TRIOBJ_CLASS_ID, 0));
            // Note that the TriObject should only be deleted

// if the pointer to it is not equal to the object
            // pointer that called ConvertToType()
            if (obj != tri) deleteIt = TRUE;
            return tri;
      }
      else
      {
            return NULL;
```

```
	}
}

//////////////////////////////////////////////////////////////////////////////
////////
///
///		Compressor Class (this is where the magic happens)

Compressor::Compressor()
{
	for (int i=0;i<MAXOBJECTS;i++)
	{
		CameraPtrList[i] = NULL;
	}
	iLightCount = 0;
	iCamCount = 0;
	iGeomCount = 0;
	iMatCount = 0;
	nLastRefNum = -1;
	nLastMatRefNum = -1;
	BytesSoFar = 0;
	iSampleRate = 2;

	fTotalError = 0.0f;
	iTotalVerts = 0;
}

Compressor::~Compressor()
{
}

void Compressor::ShowAbout(HWND hWnd)
{
}

int   Compressor::DoExport(const TCHAR *name,ExpInterface *ei,Interface *gi,
BOOL suppressPrompts)
{
	// Set a global prompt display switch
	showPrompts = suppressPrompts ? FALSE : TRUE;

	int status = 0;

	SortObjectList(ei,gi,this);

	status=CompressScene(name, ei, gi, this);

	if(status == 0)
		return 1;		// Dialog cancelled
	if(status > 0)
		DebugPrint("Progressive Tranmission Compressor status OK!\n");
	else if(status < 0)
		DebugPrint("Error somewhere in PRT!\n");
	return(status);
}

static BOOL CALLBACK
CompressorOptionsDlgProc(HWND  hDlg,  UINT  message,  WPARAM  wParam,  LPARAM
```

```
lParam) {
// used to generate and retrieve the values for the main-menu dialog
      static Compressor *exp;
      Interval range = intPtr->GetAnimRange();
      HWND hCombo, hSlider;
//    HWND hEdit;
      int i;

      switch(message) {
            case WM_INITDIALOG:
                  exp = (Compressor *)lParam;
                  CheckDlgButton( hDlg, IDC_RADIOLOWQ, TRUE);
                  CheckDlgButton( hDlg, IDC_RADIO32B, TRUE);
                  SetDlgItemInt(hDlg, IDC_XRES, DEFAULT_XRES, FALSE);
                  SetDlgItemInt(hDlg, IDC_YRES, DEFAULT_YRES, FALSE);
                  SetDlgItemInt(hDlg, IDC_FRAMERATE, GetFrameRate(), FALSE);
                  SetDlgItemInt(hDlg,                                IDC_STARTFRAME,
range.Start()/GetTicksPerFrame(), FALSE);
                  SetDlgItemInt(hDlg,                                  IDC_ENDFRAME,
range.End()/GetTicksPerFrame(), FALSE);

                  CheckDlgButton( hDlg, IDC_DIRLIGHT, TRUE);
/*                if (!bSceneContainsNoLights)
                        SetDlgItemInt(hDlg, IDC_DIRLIGHT, BN_DISABLE, TRUE);
                  */
                  // connect camera information to combo box here.
                  hCombo = GetDlgItem(hDlg, IDC_COMBOCAMERAS);
                  SendMessage(hCombo, CB_RESETCONTENT, 0, 0);

                  for (i=0;i<exp->iCamCount;i++)
                  {
                        SendMessage(hCombo,    CB_ADDSTRING,    0,    (LPARAM)
exp->CameraPtrList[i]->GetName());
                  }
                  SendMessage(hCombo, CB_SETCURSEL, (WPARAM) 0, 0);

                  hSlider = GetDlgItem(hDlg, IDC_SLIDER_THRESHOLD);
                  SendMessage(hSlider, TBM_SETTICFREQ, (WPARAM) 50, 0);
                  SendMessage(hSlider,  TBM_SETRANGE,  (WPARAM)TRUE,  (LPARAM)
MAKELONG(30,400));
                  SendMessage(hSlider,     TBM_SETPOS,     (WPARAM)(BOOL)TRUE,
(LPARAM)(LONG)300);

                  hSlider = GetDlgItem(hDlg, IDC_SLIDER_TEXRES);
                  SendMessage(hSlider,  TBM_SETRANGE,  (WPARAM)TRUE,  (LPARAM)
MAKELONG(MINTEXTUREPOWER,MAXTEXTUREPOWER));
                  SendMessage(hSlider,     TBM_SETPOS,     (WPARAM)(BOOL)TRUE,
(LPARAM)(LONG)MINTEXTUREPOWER+3);

                  if (exp->iCamCount == 0)
                  {
                        MessageBox(NULL,"The scene must contain a camera in
order to be exported.","Error!",MB_OK);
                        return TRUE;
                        // this REALLY should be replaced by a default from
the front facing viewport
                  }
                  ::CenterWindow(hDlg,GetParent(hDlg));
```

```
                ::SetFocus(hDlg); // For some reason this was necessary.
DS-3/4/96
                return FALSE;
            case WM_DESTROY:
                return FALSE;
            case WM_COMMAND:
                switch(LOWORD(wParam)) {
                    case IDOK:
                        {
                        // Unload values into local statics

                        //  THIS  IS  EQUIVILENT  TO  A  DOMODAL()==IDOK
being returned
                        // so we grab the dialog crap HERE

                            if              (IsDlgButtonChecked(hDlg,
IDC_RADIOLOWQ))
                                bMaterialQuality = FALSE;
                            else bMaterialQuality = TRUE;

                            if              (IsDlgButtonChecked(hDlg,
IDC_VERTNORM))
                                bVertexNormals = TRUE;
                            else bVertexNormals = FALSE;

                            if              (IsDlgButtonChecked(hDlg,
IDC_COMPNORM))
                                bCompressNormals = TRUE;
                            else bCompressNormals = FALSE;

                            if              (IsDlgButtonChecked(hDlg,
IDC_DIRLIGHT))
                                bUseShading = TRUE;
                            else bUseShading = FALSE;

                            if (IsDlgButtonChecked(hDlg, IDC_RADIO8B))
                                iBitsPerVertex = BPC_8;
                            else     if     (IsDlgButtonChecked(hDlg,
IDC_RADIO10B))
                                iBitsPerVertex = BPC_10;
                            else     if     (IsDlgButtonChecked(hDlg,
IDC_RADIO16B))
                                iBitsPerVertex = BPC_16;
                            else iBitsPerVertex = BPC_32;

                            BOOL bSuccess;
                            iSampleXres    =    GetDlgItemInt(    hDlg,
IDC_XRES, &bSuccess, FALSE) * 2;
                            if (!bSuccess) return FALSE;
                            iSampleYres    =    GetDlgItemInt(    hDlg,
IDC_YRES, &bSuccess, FALSE) * 2;
                            if (!bSuccess) return FALSE;
                            iFrameRate     =    GetDlgItemInt(    hDlg,
IDC_FRAMERATE, &bSuccess, FALSE);
                            if (!bSuccess) return FALSE;
                            iStartFrame    =    GetDlgItemInt(    hDlg,
IDC_STARTFRAME, &bSuccess, FALSE);
```

```
                                    if (!bSuccess) return FALSE;
                                    iEndFrame       =   GetDlgItemInt(   hDlg,
IDC_ENDFRAME, &bSuccess, FALSE);
                                    if (!bSuccess) return FALSE;
                                    ::EndDialog(hDlg, 1);


                                    hCombo          =          GetDlgItem(hDlg,
IDC_COMBOCAMERAS);
                                    exp->nSelectedCamera = SendMessage(hCombo,
CB_GETCURSEL, 0, 0);

                                    hSlider         =          GetDlgItem(hDlg,
IDC_SLIDER_THRESHOLD);
                                    exp->fTextureQualityThreshold            =
SendMessage(hSlider, TBM_GETPOS, 0, 0) / 10.0f;
                                    hSlider         =          GetDlgItem(hDlg,
IDC_SLIDER_TEXRES);
                                    int     count     =     SendMessage(hSlider,
TBM_GETPOS, 0, 0);
                                    exp->iDesiredTexturePower = count;


                                    hSlider         =          GetDlgItem(hDlg,
IDC_SLIDER_TEXRES);
                                }
                                return TRUE;
                            case IDCANCEL:
                                ::EndDialog(hDlg, 0);
                                return TRUE;
                        }
                    }
            return FALSE;
}

// Dummy function for progress bar
DWORD WINAPI fn(LPVOID arg)
{
      return(0);
}

int  Compressor::CompressScene(const  TCHAR  *  filename,  ExpInterface  *  ei,
Interface * gi, Compressor * exp)
{
// This is the true, practical main function where all the magic really
happens.
// After the dialog is brought up, the system begins the exporting process,
as
// described below.

      BOOL  bFailed = FALSE;  // did any setup tests fail?

      // disable keyboard accelerators to prevent strange behaviours in Max
from happening

      DisableAccelerators();
      if (iCamCount == 0)
      {
```

```
            MessageBox(NULL,"The scene must contain a camera in order to be
exported.","Error!",MB_OK);
            return 0;
      }

      if(showPrompts)
      {
            // Put up the options dialog to find out how they want the file
written!
            int          result          =          DialogBoxParam(hInstance,
MAKEINTRESOURCE(IDD_COMPRESSDIALOG),
                  gi->GetMAXHWnd(), CompressorOptionsDlgProc, (LPARAM)exp);
            if(result <= 0)
                  return 0;
      }
      else
      { // if they're running this via a batch process of some kind, force
these default options.
            iSampleXres = DEFAULT_XRES;
            iSampleYres = DEFAULT_YRES;
            bMaterialQuality = FALSE;
      }

      // okay, let's check to make sure we can actually open the file
      if ((fileOut = fopen("____temp", "wb")) == NULL)
//    if ((fileOut = fopen(filename, "wb")) == NULL)
      {
            bFailed = TRUE;
            // messagebox saying unable to write to file
      }
      else fclose(fileOut);

      // now that we have all the basic information we need about all the
      // objects themselves

      if (!bFailed && BuildSubObjectList() && BuildVertexFaceList())
      {
      // build a face/vertex list for each polygonal object, using a
      // dynamically created array, to conserve memory.
            intPtr->ProgressStart(GetString(IDS_PROGRESS_MSG1),    TRUE,    fn,
NULL);

            // hide all the objects in the list
            for (int i=0;i<iGeomCount;i++)
                  GeometryList[i].curNode->Hide(TRUE);
            for (i=0;i<iSuperObjCount;i++)
                  SuperObjects[i]->Hide(TRUE);

DebugPrint("Constructing Materials.\n");
      // Generate the list of all materials, and which objects are
      // using which materials
            BuildMaterialList();

            iSampleRate = (int)(ceilf(30.0f / (float)iFrameRate)) * 2;

      // #now proceed through the frame list (loop)
            for(int frame = iStartFrame;frame < iEndFrame+1;frame++)
```

```
            {
                  intPtr->ProgressUpdate((int)((float)frame/(float)iEndFrame*1
00.0f));
      // in the current frame:
                  for (int Sample=0;Sample<iSampleRate;Sample++)
                  {
#ifdef _DEBUG
                  char buffer[200];
                  sprintf(buffer,"[Frame #%d, Sample #%d]\n",frame,Sample);
                  DebugPrint(buffer);
#endif

            // transmit all active lights not in the active light list
            // If any lights previously active turn off in this frame,
transmit the "off" message
            // Draw all current objects in the database, using the schema:
            // (object number << 24 + face number) + 1.
DebugPrint("Constructing Object.\n");
                        INode*                    falseobj                  =
ConstructObjectDatabase(frame,iSampleRate,Sample);

            // Backup old frame and environment settings
                        PrepareForRendering(frame,iSampleRate,Sample);

            // Render frame using 3DSMax's renderer
                        Bitmap* bmap = 0;
                        BitmapInfo bi("0image.png");
                        bi.SetType(BMM_TRUE_64);
//                bi.SetType(BMM_TRUE_32);
                        bi.SetWidth(iSampleXres);
                        bi.SetHeight(iSampleYres);
                        bmap = TheManager->Create(&bi);
                        bmap->OpenOutput(&bi);
                        bmap->SetFilter(BMM_FILTER_NONE);
                        bmap->SetDither(BMM_DITHER_NONE);

//DebugPrint("Rendering current frame.\n");
                        RenderCurrentFrame(frame,bmap,iSampleRate,Sample);
                        BMM_Color_64 black64= {0,0,0,0};
                        Bitmap* readmap = 0;
                        readmap = TheManager->Create(&bi);
                        readmap->CopyImage(bmap,COPY_IMAGE_RESIZE_LO_QUALITY,b
lack64);

                        bmap->Close(&bi);
                        if (falseobj) falseobj->DeleteThis();
                        if (bmap) bmap->DeleteThis();

            // Read rendered frame back into a Bitmap (RGB 16/16/16)
            // Scan Bitmap per-pixel.
                  // Compare color to global color list (Binary search into
list.
                  // If not found, add to this frame color list, and add to
global color list
DebugPrint("Scanning for objects.\n");
                        ScanRenderForFaces(readmap,frame);
                        if (readmap) readmap->DeleteThis();
```

```
                    // restore old frame-related environment settings
                        CleanUpRendering(frame,iSampleRate,Sample);

DebugPrint("Frame Completed.\n");
                    }
              }
              intPtr->ProgressEnd();

              intPtr->ProgressStart(GetString(IDS_PROGRESS_MSG2),    TRUE,    fn,
NULL);
              for (int objnumber=0;objnumber<iGeomCount;objnumber++)
              {
                    // now we resort each object's added face/vertex list based
on
                    //  when they were added to the scene (since to speed
things, we
                    // originally sorted on reference number).
                    intPtr->ProgressUpdate((int)((float)objnumber/(float)iGeomCo
unt*100.0f));
                    SortByFrame(&(GeometryList[objnumber]));
              }
              intPtr->ProgressEnd();

              intPtr->ProgressStart(GetString(IDS_PROGRESS_MSG3),    TRUE,    fn,
NULL);

              // create output file and add filetype parameters
              int HeaderLength = InitFileForOutput(filename);

              // exporting background info
              Point3 BgrndColor = intPtr->GetBackGround(0,FOREVER);

              if (fabsf(BgrndColor.x) > 0.001f || fabsf(BgrndColor.y) > 0.001f
||
                    fabsf(BgrndColor.z) > 0.001f)
              {
                    iOutToFile(OPCODE_SETBACKGROUNDCOLOR);
                    wOutToFile(Convert_16bColor(ConvertRGBtoColorref(BgrndColor)
));
              }

              if (iStartFrame == iEndFrame)
                    iEndFrame++;
              for
(frame=iStartFrame+iFrameRate;frame<iEndFrame+iFrameRate;frame+=iFrameRate)
              {
                    if (frame > iEndFrame) frame = iEndFrame;
                    BuildDataBlock(frame);
              }
              // Transmit color list as face/vertex/obj info.
              // If first time adding an object:
                    // Transmit object placers (3D offsets)
                    // Transmit object material information
              // Add list of objects in this frame to active object list
              // Add keyframe information for any currently active objects that
require it
              intPtr->ProgressEnd();
```

```
            CloseOutputFile();

            // okay, since we're basically done, let's losslessly compress
the
            // whole file.  As it turns out, we're using the LZSS technique.
            if (!LZSS_OutputFile(HeaderLength, filename)) return FALSE;

            // Everything is finished, so restore the scene to normal
            // unhide all objects in the list
            for (i=0;i<iGeomCount;i++)
                  GeometryList[i].curNode->Hide(FALSE);
            for (i=0;i<iSuperObjCount;i++)
                  SuperObjects[i]->Hide(FALSE);


            for (i=0;i<iGeomCount;i++)
                  if (IsMarkedForDeletion(&GeometryList[i]))
                  {
                        GeometryList[i].curNode->Detach(0,1);
                        GeometryList[i].curNode->SetMtl(NULL);
                        GeometryList[i].curNode->SetTMController(NULL);
                        GeometryList[i].curNode->DeleteThis();
                  }
      }
      else
      {
            for (int i=0;i<iGeomCount;i++)
                  if (IsMarkedForDeletion(&GeometryList[i]))
                  {
                        GeometryList[i].curNode->Detach(0,1);
                        GeometryList[i].curNode->SetMtl(NULL);
                        GeometryList[i].curNode->SetTMController(NULL);
                        GeometryList[i].curNode->DeleteThis();
                  }
      }

#ifdef _DEBUG
      char buffer[200];
      if (iTotalVerts == 0) iTotalVerts=1;
      sprintf(buffer,"Average Error Per Vertex = %f percent\n",fTotalError / (
float)iTotalVerts * 100.0f);
      MessageBox(NULL,buffer,"Final Statistics",MB_OK);
#endif

      // Renable keyboard accelerators in Max
      EnableAccelerators();
      return 1;
}

Compressor::SortObjectList(ExpInterface * ei, Interface * gi, Compressor *
exp)
{
      // builds a list of all the current objects, and sorts them into their
      // various categories.

      intPtr = gi;
      INode* Head = intPtr->GetRootNode();
      iTotalNodeCount = 0;
```

```
	int iNumChildren = Head->NumberOfChildren();

	// dummy count first, just to get node count
	for (int idx=0; idx<iNumChildren; idx++)
		DummyCount(Head->GetChildNode(idx));

	intPtr->ProgressStart(GetString(IDS_PROGRESS_MSG0), TRUE, fn, NULL);

	iCurrentCount = 0;
	for (idx=0; idx<iNumChildren; idx++)
	{
		if (intPtr->GetCancel())
			break;
		nodeEnum(Head->GetChildNode(idx));
	}

	// We're done sorting. Finish the progress bar.
	intPtr->ProgressEnd();
//	BuildIndexList(CameraPtrList,CameraIndexList,iCamCount);
//	BuildIndexList(LightPtrList,LightIndexList,iLightCount);
	MarkTargets();
	return 0;
}

Compressor::DummyCount(INode* node)
// a recursive function used in the enumeration of the number of nodes
// in the scene
{
	iTotalNodeCount++;
	for (int c = 0; c < node->NumberOfChildren(); c++)
		DummyCount(node->GetChildNode(c));
	return 0;
}

BOOL Compressor::nodeEnum(INode * node)
// Enumerates all nodes in the scene, and identifies what they are
// for later use
{
	iCurrentCount++;
	intPtr->ProgressUpdate((int)((float)iCurrentCount/iTotalNodeCount*100.0f
));
	ObjectState os = node->EvalWorldState(0);

	if (node->IsGroupHead())
	{
		DebugPrint("Group Begins:\n");
	}

	// The obj member of ObjectState is the actual object we will export.
	if (os.obj)
	{
		// We look at the super class ID to determine the type of the
object.
		switch(os.obj->SuperClassID())
		{
		case GEOMOBJECT_CLASS_ID:
			{
				GeometryList[iGeomCount].curNode = node;
```

```
                        GeometryList[iGeomCount].IsHelper = FALSE;
                        GeometryList[iGeomCount].IsChild = node->IsTarget();
                        if (node->GetParentNode()->IsRootNode() )
                        {
                              GeometryList[iGeomCount].IsChild = FALSE;
                              GeometryList[iGeomCount].ParentIdx = -1;
                        }
                        else
                        {
                              GeometryList[iGeomCount].ParentIdx              =
GetGeometryNodeIndex(node->GetParentNode());
                              GeometryList[iGeomCount].IsChild = TRUE;
#ifdef DBGMODE
                              DebugPrint("Parent of node = ");
                              DebugPrint(node->GetParentNode()->GetName());
                              DebugPrint("; ");
#endif
                        }
                        iGeomCount++;
#ifdef DBGMODE
                        DebugPrint("GEOMETRY: ");
                        DebugPrint(node->GetName());
                        DebugPrint("\n");
#endif
                  }
                  break;
            case CAMERA_CLASS_ID:
                  CameraPtrList[iCamCount++] = node;
#ifdef DBGMODE
                  DebugPrint("CAMERA: ");
                  DebugPrint(node->GetName());
                  DebugPrint("\n");
#endif
                  CameraIndexList[iCamCount-1] = iGeomCount;
                  GeometryList[iGeomCount].curNode = node;
                  GeometryList[iGeomCount].IsHelper = FALSE;
                  if (node->GetParentNode()->IsRootNode() )
                  {
                        GeometryList[iGeomCount].IsChild = FALSE;
                        GeometryList[iGeomCount].ParentIdx = -1;
                  }
                  else
                  {
                        GeometryList[iGeomCount].ParentIdx                =
GetGeometryNodeIndex(node->GetParentNode());
                        GeometryList[iGeomCount].IsChild = TRUE;
#ifdef DBGMODE
                        DebugPrint("Parent of node = ");
                        DebugPrint(node->GetParentNode()->GetName());
                        DebugPrint("; ");
#endif
                  }
                  iGeomCount++;
                  break;
            case LIGHT_CLASS_ID:
                  LightPtrList[iLightCount++] = node;
#ifdef DBGMODE
                  DebugPrint("LIGHT: ");
```

```
                    DebugPrint(node->GetName());
                    DebugPrint("\n");
#endif
                    LightIndexList[iLightCount-1] = iGeomCount;
                    GeometryList[iGeomCount].curNode = node;
                    GeometryList[iGeomCount].IsHelper = FALSE;
                    if (node->GetParentNode()->IsRootNode() )
                    {
                          GeometryList[iGeomCount].IsChild = FALSE;
                          GeometryList[iGeomCount].ParentIdx = -1;
                    }
                    else
                    {
                          GeometryList[iGeomCount].ParentIdx                    =
GetGeometryNodeIndex(node->GetParentNode());
                          GeometryList[iGeomCount].IsChild = TRUE;
#ifdef DBGMODE
                          DebugPrint("Parent of node = ");
                          DebugPrint(node->GetParentNode()->GetName());
                          DebugPrint("; ");
#endif
                    }
                    iGeomCount++;
                    break;
             case SHAPE_CLASS_ID:
#ifdef DBGMODE
                    DebugPrint("SHAPER: ");
                    DebugPrint(node->GetName());
                    DebugPrint("\n");
#endif
                    break;
             case HELPER_CLASS_ID:
                    GeometryList[iGeomCount].curNode = node;
                    GeometryList[iGeomCount].IsHelper = TRUE;
                    if (node->GetParentNode()->IsRootNode())
                    {
                          GeometryList[iGeomCount].IsChild = FALSE;
                          GeometryList[iGeomCount].ParentIdx = -1;
                    }
                    else
                    {
                          GeometryList[iGeomCount].ParentIdx                    =
GetGeometryNodeIndex(node->GetParentNode());
                          GeometryList[iGeomCount].IsChild = TRUE;
#ifdef DBGMODE
                          DebugPrint("Parent of node = ");
                          DebugPrint(node->GetParentNode()->GetName());
                          DebugPrint("; ");
#endif
                    }
                    iGeomCount++;
#ifdef DBGMODE
                    DebugPrint("HELPER: ");
                    DebugPrint(node->GetName());
                    DebugPrint("\n");
#endif
                    break;
             }
```

```
	}

	for (int c = 0; c < node->NumberOfChildren(); c++)
		if (!nodeEnum(node->GetChildNode(c)))
			return FALSE;

	if (node->IsGroupHead())
	{
		DebugPrint("Group Ends:\n");
	}

	return TRUE;
}

int Compressor::GetGeometryNodeIndex(INode * node)
// finds the index into the geometry array for the given node pointer.
// if its not found, a -1 is returned.
{
	for (int i=0;i<iGeomCount;i++)
		if (node == GeometryList[i].curNode)
			break;
	if (i == iGeomCount)
		return -1;
	else return i;
}

Compressor::BuildIndexList(INode ** PtrList, int *IndexList, int count)
// finds the location of each pointer reference in PtrList and places
// its reference number in IndexList.
{
	for (int i=0;i<count;i++)
	{
		for(int j=0;GeometryList[j].curNode != PtrList[i] &&
j<iGeomCount;j++);
		if (GeometryList[j].curNode != PtrList[i])
		{
			DebugPrint("Error encountered while trying to build index
list.\n");
		}
		else
			IndexList[i] = j;
	}
}

Compressor::MarkTargets()
// assigns all the target references to the various nodes
{
	for (int i=0;i<iGeomCount;i++)
	{
		if (GeometryList[i].curNode->GetTarget())
		{
			GeometryList[i].TargetNode =
GeometryList[i].curNode->GetTarget();
			for (int j=0;j<iGeomCount;j++)
				if (GeometryList[j].curNode ==
GeometryList[i].TargetNode)
				{
```

```
                                    GeometryList[i].TargetIdx = j;
                                    GeometryList[j].IsTarget = TRUE;
                                    break;
                              }
                        assert(j != iGeomCount);
                  }
                  else
                  {
                        GeometryList[i].TargetNode = NULL;
                        GeometryList[i].TargetIdx = -1;
                  }

            }
}

Compressor::BuildMaterialList()
// Generates the list of materials and assigns them to our constructed
// object database
{
      for (int i=0;i<iGeomCount;i++)
      {
            Mtl* mnode = GeometryList[i].curNode->GetMtl();
            if (mnode)
            {
                  BOOL bFound = FALSE;
                  for (int j=0;j<iMatCount;j++)
                        if (mnode == MaterialList[j].curNode)
                        {
                              bFound = TRUE;
                              break;
                        }
                  if (!bFound)
                  {
                        GeometryList[i].MaterialID = iMatCount;
                        MaterialList[iMatCount++].curNode = mnode;

                  }
                  else GeometryList[i].MaterialID = j;
            }
      }
}

int FindSmthVert(int *VertList,int VertCount,int VertNum)
// performs a linear search into a vertex list for a specific
// vertex number, and returns it's index into the array
{
      int loc = -1;
      for (int i=0;i<VertCount;i++)
            if (VertList[i] == VertNum)
            {
                  loc = i;
                  break;
            }
      return (loc);
}

BOOL Compressor::BuildSubObjectList()
// Takes each object and slices it into new objects based on smoothing groups
```

```
{
	int newCount = iGeomCount;
	iSuperObjCount = 0;
	int *Vertices[32];
	int *Faces[32];
	int *FaceID[32];
	for (int x=0;x<32;x++)
	{
		Vertices[x] = new int[MAXOBJECTS];
		Faces[x] = new int[MAXOBJECTS*3];
		FaceID[x] = new int[MAXOBJECTS];
	}

	int FaceCount[32];
	int VertCount[32];

	for (int loop=0;loop<newCount;loop++)
	{
		if (!GeometryList[loop].IsHelper &&
GeometryList[loop].IsSuperObject())
		{
			Object *obj =
GeometryList[loop].curNode->EvalWorldState(0).obj;
			TriObject *Tri;
			Tri = (TriObject*)
obj->ConvertToType(0,Class_ID(TRIOBJ_CLASS_ID,0));
			if (Tri && Tri->mesh.numFaces > 0)
			{
				SuperObjects[iSuperObjCount++] =
GeometryList[loop].curNode;
				// A mesh can have up to a total of 32 smoothing
groups.
				// As each group is marked using a bitmask in each
face,
				// we take the original object and split it into up
to 32
				// new objects
				// if any of these new objects don't have any
vertices
				// we discard them.  If they do, we append them to
the
				// end of the geometry list, and make them children
				// of a new dummy object.  We assign the original
				// material to all of these children
				for (int i=0;i<32;i++)
				{
					FaceCount[i] = 0;
					VertCount[i] = 0;
				}
				for (i=0;i<Tri->mesh.numFaces;i++)
				{
					DWORD smooth = Tri->mesh.faces[i].smGroup;
					DWORD mask = 1;
					for (int j=0;j<32 && ((smooth & (mask << j)) ==
0);j++);
					if (j==32) j=0;
					int loc1 =
```

```
FindSmthVert(Vertices[j],VertCount[j],Tri->mesh.faces[i].v[0]);
                        int                         loc2                        =
FindSmthVert(Vertices[j],VertCount[j],Tri->mesh.faces[i].v[1]);
                        int                         loc3                        =
FindSmthVert(Vertices[j],VertCount[j],Tri->mesh.faces[i].v[2]);
                        if (loc1 == -1)
                        {
                              Vertices[j][VertCount[j]]                         =
Tri->mesh.faces[i].v[0];
                              loc1 = VertCount[j]++;
                        }
                        if (loc2 == -1)
                        {
                              Vertices[j][VertCount[j]]                         =
Tri->mesh.faces[i].v[1];
                              loc2 = VertCount[j]++;
                        }
                        if (loc3 == -1)
                        {
                              Vertices[j][VertCount[j]]                         =
Tri->mesh.faces[i].v[2];
                              loc3 = VertCount[j]++;
                        }
                        FaceID[j][FaceCount[j]] = i;
                        Faces[j][FaceCount[j]*3+0] = loc1;
                        Faces[j][FaceCount[j]*3+1] = loc2;
                        Faces[j][FaceCount[j]*3+2] = loc3;
                        FaceCount[j]++;
                  }

                  // first, let's create a dummy node containing
nothing, but placed at the
                  // coordinates of the original object, with the
original orientation

                  TriObject* dummyobj = CreateNewTriObject();
                  dummyobj->mesh.setNumVerts(0);
                  dummyobj->mesh.setNumFaces(0);
                  INode*  meshnode = 0;
                  meshnode = intPtr->CreateObjectNode(dummyobj);
                  TCHAR dummyname[255];
                  _tcscpy(dummyname,GeometryList[loop].curNode->GetName(
));
                  _tcscat(dummyname,_T(" - Dummy Parent"));
                  meshnode->SetName(dummyname);
                  INode* root = intPtr->GetRootNode();
                  if (root)
                        root->AttachChild(meshnode,0);

                  GeometryList[loop].curNode = meshnode;
                  Control* contr = NULL;
                  contr     =     (Control*)SuperObjects[iSuperObjCount-
1]->GetTMController()->Clone();
                  GeometryList[loop].curNode->SetTMController(contr);
                  if (!contr)
                  {
                        Matrix3                     tm                        =
GetLocalMatrix(SuperObjects[iSuperObjCount-1],0);
                        GeometryList[loop].curNode->SetNodeTM(0,tm);
```

```
                        }
                        // reproduce any original pivot point it had
                        Point3          offsetPos                       =
SuperObjects[iSuperObjCount-1]->GetObjOffsetPos();
                        Quat        offsetRot   =  SuperObjects[iSuperObjCount-
1]->GetObjOffsetRot();
                        ScaleValue  offsetScl   =  SuperObjects[iSuperObjCount-
1]->GetObjOffsetScale();

                        GeometryList[loop].curNode->SetGroupHead(TRUE);
                        GeometryList[loop].bKillMe = TRUE;
                        MarkNodeForDeletion(&GeometryList[loop]);

                        // so now that we have a list of all the necessary
faces and vertices, we can
                        // build up our new objects
                        for (i=0;i<32;i++)
                              if (FaceCount[i] > 0)
                              {
                                    TriObject* newobj = CreateNewTriObject();
                                    // copy verts
                                    newobj->mesh.setNumVerts(VertCount[i]);
                                    for (int j=0;j<VertCount[i];j++)

                                          newobj->mesh.setVert(j,Tri->mesh.ver
ts[Vertices[i][j]]);

                                    // copy faces
                                    newobj->mesh.setNumFaces(FaceCount[i]);
                                    for (j=0;j<FaceCount[i];j++)
                                          newobj->mesh.faces[j].setVerts(Faces
[i][j*3],

                                                Faces[i][j*3+1],Faces[i][j*3+2
]);

                                    Matrix3 MI(1);
                                    if   (Tri->mesh.getNumTVerts()   ==   0   &&
SuperObjects[iSuperObjCount-1]->GetMtl())
                                          Tri->mesh.ApplyUVWMap(MAP_PLANAR,1.0
f,1.0f,1.0f,0,0,0,0,0,MI);
                                    if (Tri->mesh.getNumTVerts() > 0)
                                    {
                                          // copy texture verts
                                          newobj->mesh.setNumTVerts(VertCount[
i]);

                                          newobj->mesh.setNumTVFaces(FaceCount
[i]);

                                          for      (j=0;j<FaceCount[i]      &&
Tri->mesh.numTVerts > 0;j++)

                                          {
                                                Point3 uv;
                                                TVFace          tf          =
Tri->mesh.tvFace[FaceID[i][j]];

                                                newobj->mesh.setTVert(Faces[i]
[j*3  ],Tri->mesh.tVerts[tf.t[0]]);

                                                newobj->mesh.setTVert(Faces[i]
[j*3+1],Tri->mesh.tVerts[tf.t[1]]);

                                                newobj->mesh.setTVert(Faces[i]
```

```
[j*3+2],Tri->mesh.tVerts[tf.t[2]]);
                                        newobj->mesh.tvFace[j].t[0]  =
Faces[i][j*3  ];
                                        newobj->mesh.tvFace[j].t[1]  =
Faces[i][j*3+1];
                                        newobj->mesh.tvFace[j].t[2]  =
Faces[i][j*3+2];
                                    }
                                }
                                else
                                {
                                    newobj->mesh.setNumTVerts(0);
                                    newobj->mesh.setNumTVFaces(0);
                                }
                                newobj->mesh.buildRenderNormals();
                                for (j=0;j<newobj->mesh.numFaces;j++)
                                    newobj->mesh.FlipNormal(j);

                                // now that we've duplicated all the
object's properties into
                                // the sub object.. we now set some
internal settings so we
                                // know its not a real object, but so the
subobject operates
                                // like an ordinary object
                                INode*          newnode = 0;

                                newnode                                 =
intPtr->CreateObjectNode(newobj);
                                newnode->SetMtl(SuperObjects[iSuperObjCoun
t-1]->GetMtl());
                                newnode->SetWireColor(SuperObjects[iSuperO
bjCount-1]->GetWireColor());

                                TCHAR dummyname[255];
                                _tcscpy(dummyname,SuperObjects[iSuperObjCo
unt-1]->GetName());

                                TCHAR namenum[255];
                                _stprintf(namenum," part #%d",i);
                                _tcscat(dummyname,namenum);
                                newnode->SetName(dummyname);
                                newnode->SetGroupMember(TRUE);
                                GeometryList[iGeomCount].curNode        =
newnode;

                                GeometryList[iGeomCount].bKillMe = TRUE;
                                GeometryList[iGeomCount].IsChild = TRUE;
                                GeometryList[iGeomCount].ParentIdx = loop;

                                GeometryList[iGeomCount].curNode->SetObjOf
fsetPos(offsetPos);
                                GeometryList[iGeomCount].curNode->SetObjOf
fsetRot(offsetRot);
                                GeometryList[iGeomCount].curNode->SetObjOf
fsetScale(offsetScl);

                                MarkNodeForDeletion(&GeometryList[iGeomCou
nt]);

                                SuperObjects[iSuperObjCount-
```

```
1]->AttachChild(newnode,0);
                                        iGeomCount++;
                                    }
                              }
                        }
                  }
                  for (x=0;x<32;x++)
                  {
                        delete[] Vertices[x];
                        delete[] Faces[x];
                        delete[] FaceID[x];
                  }

                  return TRUE;
}

Compressor::MarkNodeForDeletion(GeoNode *node)
{
      TCHAR dummyname[255];
      _tcscpy(dummyname,node->curNode->GetName());
      _tcscat(dummyname,_T("**DELETEME!!**"));
      node->curNode->SetName(dummyname);
}

BOOL Compressor::BuildVertexFaceList()
// Takes all the valid objects in the scene and reproduces their geometric
// attributes in our local database.
{
      int maxfacecount = 1, maxobjcount = 1;;
//for (int zz=0;zz<(24-FACEBITCOUNT);zz++)
      for (int zz=0;zz<24;zz++)
            maxfacecount*=2;
//    for (zz=0;zz<FACEBITCOUNT;zz++)
      for (zz=0;zz<24;zz++)
            maxobjcount*=2;
      char buffer[200];
//    if (maxfacecount > 56*200) maxfacecount = 56*200-1;
//    if (maxobjcount > 56*200) maxobjcount = 56*200-1;

      if (iGeomCount> maxobjcount)
      {
            sprintf(buffer,"Your scene contains more objects than are allowed
in the scene.\nUse the System Information tool under the File menu for more
information.",maxobjcount);
            MessageBox(0, buffer, "Error", MB_OK);
            return FALSE;
      }

      // Init all the data structures to hold the geometry for the various
      // meshes as they are added to.
      for (int i=0;i<iGeomCount;i++)
            if (!GeometryList[i].IsHelper)
            {
                  Object                              *obj                              =
GeometryList[i].curNode->EvalWorldState(0).obj;
                  TriObject *Tri;
                  if (GeometryList[i].IsHelper)
                        Tri = NULL;
```

```
                else                Tri                 =                (TriObject*)
obj->ConvertToType(0,Class_ID(TRIOBJ_CLASS_ID,0));
                if (Tri)
                {
                      GeometryList[i].nTrueFaceCount = Tri->mesh.numFaces;
                      GeometryList[i].nTrueVertexCount = Tri->mesh.numVerts;
                      GeometryList[i].TrueFaceList            =            new
TriFace[GeometryList[i].nTrueFaceCount];
                      GeometryList[i].AddedFaceList           =            new
int[GeometryList[i].nTrueFaceCount];
                      GeometryList[i].FaceAddedOnFrame          =          new
int[GeometryList[i].nTrueFaceCount];
                      GeometryList[i].AddedVertList           =            new
int[Tri->mesh.numVerts];
                      GeometryList[i].VertAddedOnFrame          =          new
int[Tri->mesh.numVerts];
                      GeometryList[i].TexIndexList            =            new
int[Tri->mesh.numVerts];

                      if ( (GeometryList[i].nTrueFaceCount > 0 &&
                            GeometryList[i].nTrueVertexCount > 0 ) &&
                           (GeometryList[i].nTrueFaceCount                ==
NULL ||
                           GeometryList[i].nTrueVertexCount     == NULL ||
                           GeometryList[i].TrueFaceList         == NULL ||
                           GeometryList[i].AddedFaceList        == NULL ||
                           GeometryList[i].FaceAddedOnFrame     == NULL ||
                           GeometryList[i].AddedVertList        == NULL ||
                           GeometryList[i].VertAddedOnFrame     == NULL) )
                      {
                           MessageBox(0,
                                 "Unable to allocate memory required to
compress this scene.\nClose some applications, or increase the virtual memory
available on your system.",
                                 "Error", MB_OK);
                           exit(1);
                      }
                      for (int j=0;j<GeometryList[i].nTrueFaceCount;j++)
                      {
                           GeometryList[i].TrueFaceList[j].Vert1          =
Tri->mesh.faces[j].v[0];
                           GeometryList[i].TrueFaceList[j].Vert2          =
Tri->mesh.faces[j].v[1];
                           GeometryList[i].TrueFaceList[j].Vert3          =
Tri->mesh.faces[j].v[2];

                           GeometryList[i].AddedFaceList[j]               =
BIGCONSTVALUE;

                           GeometryList[i].FaceAddedOnFrame[j]            =
BIGCONSTVALUE;
                      }
                      for (j=0;j<Tri->mesh.numVerts;j++)
                      {
                           GeometryList[i].AddedVertList[j]               =
BIGCONSTVALUE;
                           GeometryList[i].VertAddedOnFrame[j]            =
BIGCONSTVALUE;
```

```
                        }
                        if (GeometryList[i].nTrueFaceCount > maxfacecount)
                        {
                              sprintf(buffer,"One or more of the objects in
this scene contains more than %d faces.\nUse the System Information tool under
the File menu for more information.",maxfacecount);
                              MessageBox(0, buffer, "Error", MB_OK);
                              return FALSE;
                        }
                        if (Tri->mesh.numTVerts == 0)
                        {
                              StdMat*            nodeMtl=            (StdMat*)
GeometryList[i].curNode->GetMtl();
                              if (!(!nodeMtl || nodeMtl->GetWire()))
                              {
                                    sprintf(buffer,"Object \'%s\' is missing
and requires mapping coordinates to be exported.",
                                          GeometryList[i].curNode->GetName());
                                    MessageBox(0,buffer,"Error", MB_OK);
                                    return FALSE;

                              }
                        }

                        if           (GeometryList[i].TexIndexList           &&
GeometryList[i].curNode->GetMtl())
                              for (int j=0;j<Tri->mesh.numFaces;j++)
                              {
                                    GeometryList[i].TexIndexList[GeometryList[
i].TrueFaceList[j].Vert1] =
                                          Tri->mesh.tvFace[j].t[0];
                                    GeometryList[i].TexIndexList[GeometryList[
i].TrueFaceList[j].Vert2] =
                                          Tri->mesh.tvFace[j].t[1];
                                    GeometryList[i].TexIndexList[GeometryList[
i].TrueFaceList[j].Vert3] =
                                          Tri->mesh.tvFace[j].t[2];
                              }
                        if (Tri != obj)
                              Tri->DeleteMe();
                  }
            }
      return TRUE;
}

INode*   Compressor::ConstructObjectDatabase(int   frame,int   SampleRate,int
Sample)
// takes the object database we've constructed and generates a duplicate
object
// set, except each face of each object is colored so that each face color
// identifies which object originally owned it.
{
      long time = frame * GetTicksPerFrame() / SampleRate*(1+Sample);

      TriObject* obj = CreateNewTriObject();


      int currvertsum = 0;
```

```
      int currfacesum = 0;
      MNMesh builder;
      builder.SetFlag(MN_MESH_CVERTS, TRUE);

      float divis = 1.0f/65535.0f;
//    float divis = 1.0f/255.0f;

      for (int i=0;i<iGeomCount;i++)
            if (!GeometryList[i].IsHelper)
            {
                  Matrix3                        ObjTM                        =
GeometryList[i].curNode->GetObjTMAfterWSM(time);

                  Object*                        tempObj                      =
GeometryList[i].curNode->EvalWorldState(time).obj;
                  if (tempObj->CanConvertToType(Class_ID(TRIOBJ_CLASS_ID,0)))
                  {
                        TriObject*        tempTri        =        (TriObject*)
tempObj->ConvertToType(time,
                              Class_ID(TRIOBJ_CLASS_ID, 0));
                        for (int j=0;j<GeometryList[i].nTrueFaceCount;j++)
                        {
                              __int64   color   =   (i+1)   +   ((__int64)j   *
(__int64)16777216);
//                            __int64 color = (i+1) + (j << FACEBITCOUNT);
                              unsigned   char   *chcolor   =   (unsigned   char
*)&color;
                              Point3 vert;
                              vert                                             =
Transf(ObjTM,tempTri->mesh.verts[tempTri->mesh.faces[j].v[0]]);
                              builder.NewVert(vert);
                              vert                                             =
Transf(ObjTM,tempTri->mesh.verts[tempTri->mesh.faces[j].v[1]]);
                              builder.NewVert(vert);
                              vert                                             =
Transf(ObjTM,tempTri->mesh.verts[tempTri->mesh.faces[j].v[2]]);
                              builder.NewVert(vert);
                              VertColor c;
                              c.x   =   (float)(chcolor[0]+(chcolor[1]<<8))   *
divis;
                              c.y   =   (float)(chcolor[2]+(chcolor[3]<<8))   *
divis;
                              c.z   =   (float)(chcolor[4]+(chcolor[5]<<8))   *
divis;
//                            c.x = (float)chcolor[0] * divis;
//                            c.y = (float)chcolor[1] * divis;
//                            c.z = (float)chcolor[2] * divis;
                              builder.NewCVert(c);
                              builder.NewCVert(c);
                              builder.NewCVert(c);
                              int                        vv[3]                 =
{currvertsum,currvertsum+1,currvertsum+2};
                              builder.NewTri(vv,NULL,vv);

                              currvertsum+=3;
                              currfacesum++;
                        }
                        if (tempObj != tempTri) tempTri->DeleteMe();
```

```
                }
            }
        builder.OutToTri(obj->mesh);
        INode*  meshnode = 0;
        meshnode = intPtr->CreateObjectNode(obj);
        meshnode->SetName(_T("Temp Renderable Thing"));
        meshnode->SetWireColor(RGB(255,0,255));

        Color black(0.0,0.0,0.0);
        Color white(1.0,1.0,1.0);

        StdMat* VCmat = NewDefaultStdMat();
        Color blue(0.0,0.0,1.0);

        VCmat->SetWire(FALSE);
        VCmat->SetAmbient(black,0);
        VCmat->SetDiffuse(white,0);
        VCmat->SetSpecular(black,0);
        VCmat->SetFilter(black,0);
        VCmat->SetShininess(0.0,0);
        VCmat->SetShading(1);
        VCmat->SetShinStr(0.0,0);
        VCmat->EnableMap(ID_DI,TRUE);
        VCol *VertColor = NewDefaultVColTex();
        VCmat->SetSubTexmap(ID_DI, VertColor);
        meshnode->SetMtl(VCmat);
        meshnode->SetMotBlur(0);
        meshnode->SetCastShadows(FALSE);
        meshnode->SetShadeCVerts(0);
        meshnode->BackCull(FALSE);
        return (meshnode);
}

void  Compressor::PrepareForRendering(int  framenumber,int  SampleRate,  int
Sample)
{
        long time = framenumber * GetTicksPerFrame() / SampleRate * (1+Sample);
        _p3OldTint = intPtr->GetLightTint(time,FOREVER);
        intPtr->SetLightTint(time,Color(0.0,0.0,0.0));
        _fOldLightLevel = intPtr->GetLightLevel(time,FOREVER);
        intPtr->SetLightLevel(time,0.0);
        _p3OldAmbient = intPtr->GetAmbient(time,FOREVER);
        intPtr->SetAmbient(time,Color(1.0,1.0,1.0));
        _BUseEnvMap = intPtr->GetUseEnvironmentMap();
        intPtr->SetUseEnvironmentMap(FALSE);
        _p3OldBackgrd = intPtr->GetBackGround(time,FOREVER);
        intPtr->SetBackGround(time,Color(0.0,0.0,0.0));
        _BForce2Side = intPtr->GetRendForce2Side();
        intPtr->SetRendForce2Side(TRUE);

        // get active viewport
        ViewPort = intPtr->GetActiveViewport();
        ViewPort->GetAffineTM(VptTM);
        OldCamera = ViewPort->GetViewCamera();
        bPerspView = ViewPort->IsPerspView();
        ViewPort->SetViewCamera(CameraPtrList[nSelectedCamera]);
}

void Compressor::RenderCurrentFrame(int frame, Bitmap *bm, int SampleRate, int
```

```
Sample)
{
	int val;
	IScanRenderer* rend = (IScanRenderer*) intPtr->GetDraftRenderer();
	rend->GetForceWire();
	rend->SetAntialias(FALSE);
	rend->SetAutoReflect(FALSE);
	rend->SetPixelSize(1.0f);
	rend->SetAutoReflLevels(1);
	rend->SetFilter(FALSE);
	rend->SetShadows(FALSE);
	intPtr->AssignCurRenderer(rend);
	val = intPtr->OpenCurRenderer(CameraPtrList[nSelectedCamera],NULL);
	RendParams rp;
	rp.fieldRender = FALSE;
	rp.frameDur = 1;
	rp.superBlack = TRUE;
	rp.rendHidden = FALSE;
	rp.force2Side = TRUE;
	rp.mtlEditAA = FALSE;
	rp.useEnvironAlpha = FALSE;
	rp.dontAntialiasBG = TRUE;
	rp.atmos = NULL;
	rp.rendType = RENDTYPE_NORMAL;
	FrameRendParams frp;
	Color White(1.0,1.0,1.0);
	Color Black(0.0,0.0,0.0);
	frp.ambient = White;
	frp.background = Black;
	frp.globalLightLevel = White;
	intPtr->SetRendShowVFB(TRUE);
	val = intPtr->CurRendererRenderFrame(frame * GetTicksPerFrame() /
		SampleRate * (Sample+1),bm);
	intPtr->CloseCurRenderer();
}

void Compressor::ScanRenderForFaces(Bitmap* bm,int frame)
{
	unsigned char *bitmapwalker = NULL;
	int ObjNum, FaceNum, i;
	int BMtype;
	unsigned long* bPtr = (unsigned long*)
bm->Storage()->GetStoragePtr(&BMtype);

	bitmapwalker = (unsigned char*) bPtr;
	int y;

DebugPrint("Scanning...\n");


/*	FILE *fout;
	if ((fout = fopen("true.bmp", "wb")) == NULL)
	{
		fprintf (stderr, "Cannot open output file.\n");
//		exit(3);
	}
	unsigned char Header[54] =
	{
```

```
            0x42, 0x4d, 0x36, 0x84, 0x03, 0x00, 0x00, 0x00,
            0x00, 0x00, 0x36, 0x00, 0x00, 0x00, 0x28, 0x00,
            0x00, 0x00,
            (unsigned char)(iSampleXres & 0x000000FF),
            (unsigned char)((iSampleXres & 0x0000FF00) >> 8),
            0x00, 0x00,
            (unsigned char)(iSampleYres & 0x000000FF),
            (unsigned char)((iSampleYres & 0x0000FF00) >> 8),
            0x00, 0x00, 0x01, 0x00, 0x18, 0x00, 0x00, 0x00,
            0x00, 0x00, 0x00, 0x84, 0x03, 0x00, 0x12, 0x0b,
            0x00, 0x00, 0x12, 0x0b, 0x00, 0x00, 0x00, 0x00,
            0x00, 0x00, 0x00, 0x00, 0x00, 0x00
      };

      for (i=0;i<54;i++)
            fputc(Header[i],fout);
      for (y=iSampleYres-1;y>=0;y--)
            for (int x=0;x<iSampleXres;x++)
            {
                  fputc(bitmapwalker[(x+y*iSampleXres)*6+0],fout);
                  fputc(bitmapwalker[(x+y*iSampleXres)*6+2],fout);
                  fputc(bitmapwalker[(x+y*iSampleXres)*6+4],fout);
            }
      fflush(fout);
      fclose(fout);
*/

      for (i=0;i<iGeomCount;i++)
            GeometryList[i].InitBoundingBox();

      __int64 bitmask = 0x0000000000ffffff;
      for (y=0;y<iSampleYres;y++)
      {
            for (int x=0;x<iSampleXres;x++)
            {
                  __int64 color = 0;
                  unsigned char* chcolor = (unsigned char*)&color;
                  *(chcolor) = *(bitmapwalker);
                  *(chcolor+1) = *(bitmapwalker+1);
                  *(chcolor+2) = *(bitmapwalker+2);
                  *(chcolor+3) = *(bitmapwalker+3);
                  *(chcolor+4) = *(bitmapwalker+4);
                  *(chcolor+5) = *(bitmapwalker+5);
                  if (color != 0)
                  {
                        color--;
                        ObjNum  = (int)(color & bitmask);
                        FaceNum = (int)(color / 0x0000000000ffffff);
            assert(ObjNum < iGeomCount);
                        AddFaceToList(ObjNum,FaceNum,frame);
                        MarkObjBoundingBox(x,y,ObjNum);
                  }
                  bitmapwalker+=6;
            }
      }

      // now using the bounding box of the object, we guesstimate which sized
```

```
texture we would
      // need to cover the object.  Once we figure this out, we mark it in
the appropriate array.
      SortMipMapAppearanceList(frame);
}

void Compressor::AddFaceToList(int ObjNum, int FaceNum, int FrameNum)
{
      // first, we begin by retrieving the object node
      GeoNode *cNode = &(GeometryList[ObjNum]);

      // then, we binary search thru the existing faces to see
      // if this one is in the database
      int location = FaceSearch(cNode,FaceNum);
      // if its in the database, quit
      if (cNode->nAddedFaceCount > 0 && cNode->AddedFaceList[location] ==
FaceNum) return;
      // if its not, insert sort the new entry into the database of that
specific
      // node, and time stamp it with the given frame number.
      InsertFace(cNode,FaceNum,FrameNum,location);
      AddVertices(cNode,FaceNum,FrameNum);
}

int Compressor::FaceSearch(GeoNode *node, int FaceNum)
// binary search for a face's index
{
      int top = 0, bottom = node->nAddedFaceCount-1, mid;
      while (top < bottom)
      {
            mid = (top + bottom) / 2;
            if (node->AddedFaceList[mid] == FaceNum)
                  return mid;
            else if (node->AddedFaceList[mid] > FaceNum)
                  bottom = mid-1;
            else top = mid+1;
      }
      if (node->AddedFaceList && node->AddedFaceList[top] < FaceNum)
            return (top+1);
      else return (top);
}

void Compressor::InsertFace(GeoNode *node, int Face, int Frame, int loc)
// inserts a face into the list
{
      for (int i=node->nAddedFaceCount;i>loc;i--)
      {
            node->AddedFaceList[i] = node->AddedFaceList[i-1];
            node->FaceAddedOnFrame[i] = node->FaceAddedOnFrame[i-1];
      }
      node->AddedFaceList[loc] = Face;
      node->FaceAddedOnFrame[loc] = Frame;
      node->nAddedFaceCount++;
}

int Compressor::VertSearch(GeoNode *node, int VertNum)
{
```

```
// binary search for a vertex's index
      int top = 0, bottom = node->nAddedVertCount-1, mid;
      while (top < bottom)
      {
            mid = (top + bottom) / 2;
            if (node->AddedVertList[mid] == VertNum)
                  return mid;
            else if (node->AddedVertList[mid] > VertNum)
                  bottom = mid-1;
            else top = mid+1;
      }
      if (node->AddedVertList[top] < VertNum)
            return (top+1);
      else return (top);
}

int Compressor::LinVertSearch(GeoNode *node, int VertNum)
{ // assumes unsorted list
      int i=0;
      while (i<node->nAddedVertCount)
            if (node->AddedVertList[i++] == VertNum)
                  return (i-1);
      return (i);
}

void Compressor::InsertVertex(GeoNode *node, int Vert, int Frame, int loc)
// inserts a vertex into the list
{
      for (int i=node->nAddedVertCount;i>loc;i--)
      {
            node->AddedVertList[i] = node->AddedVertList[i-1];
            node->VertAddedOnFrame[i] = node->VertAddedOnFrame[i-1];
      }
      node->AddedVertList[loc] = Vert;
      node->VertAddedOnFrame[loc] = Frame;
      node->nAddedVertCount++;
}

void Compressor::AddVertices(GeoNode *node, int Face, int Frame)
// adds a face into the list
{
      int v1,v2,v3;
      v1 = node->TrueFaceList[Face].Vert1;
      v2 = node->TrueFaceList[Face].Vert2;
      v3 = node->TrueFaceList[Face].Vert3;
      // Search AddedVerticesList. if v1 is there, abort
      int location = VertSearch(node,v1);
      if  (node->nAddedVertCount==0  ||  node->AddedVertList[location]  !=  v1)

            // add the vertex
            InsertVertex(node, v1, Frame, location);
      location = VertSearch(node,v2);
      if (node->AddedVertList[location] != v2)
            // add the vertex
            InsertVertex(node, v2, Frame, location);
      location = VertSearch(node,v3);
      if (node->AddedVertList[location] != v3)
```

```
            // add the vertex
            InsertVertex(node, v3, Frame, location);
}

void Compressor::SortByFrame(GeoNode *node)
{
      if (node->nAddedFaceCount > 0)
            SortFaceListbyFrame(node,0,node->nAddedFaceCount-1);
      if (node->nAddedVertCount > 0)
            SortVertListbyFrame(node,0,node->nAddedVertCount-1);
}

void Compressor::SortFaceListbyFrame(GeoNode *node, int lo, int hi)
{
      // performs a Quicksort on the faces based on what frame
      // the face was added on
      int oldhi = hi, oldlo = lo, mid;
      if (oldhi > oldlo)
      {
            mid = node->FaceAddedOnFrame[(oldlo+oldhi) >> 1];
            while( lo <= hi)
            {
                  while ( (lo < oldhi) && (node->FaceAddedOnFrame[lo] < mid) )
lo++;
                  while ( (hi > oldlo) && (node->FaceAddedOnFrame[hi] > mid) )
hi--;
                  if (lo <= hi)
                  {
                        int tempa, tempb;
                        tempa = node->FaceAddedOnFrame[lo];
                        tempb = node->AddedFaceList[lo];
                        node->FaceAddedOnFrame[lo]                          =
node->FaceAddedOnFrame[hi];
                        node->AddedFaceList[lo] = node->AddedFaceList[hi];
                        node->FaceAddedOnFrame[hi] = tempa;
                        node->AddedFaceList[hi] = tempb;
                        lo++; hi--;
                  }
            }
            if (oldlo < hi) SortFaceListbyFrame(node, oldlo, hi);
            if (lo < oldhi) SortFaceListbyFrame(node, lo, oldhi);
      }
}

void Compressor::SortVertListbyFrame(GeoNode *node, int lo, int hi)
{
      // performs a Quicksort on the faces based on what frame
      // the vertex was added on
      int oldhi = hi, oldlo = lo, mid;
      if (oldhi > oldlo)
      {
            mid = node->VertAddedOnFrame[(oldlo+oldhi) >> 1];
            while( lo <= hi)
            {
                  while ( (lo < oldhi) && (node->VertAddedOnFrame[lo] < mid) )
lo++;
                  while ( (hi > oldlo) && (node->VertAddedOnFrame[hi] > mid) )
```

```
hi--;
                    if (lo <= hi)
                    {
                          int tempa, tempb;
                          tempa = node->VertAddedOnFrame[lo];
                          tempb = node->AddedVertList[lo];
                          node->VertAddedOnFrame[lo]                                    =
node->VertAddedOnFrame[hi];
                          node->AddedVertList[lo] = node->AddedVertList[hi];
                          node->VertAddedOnFrame[hi] = tempa;
                          node->AddedVertList[hi] = tempb;
                          lo++; hi--;
                    }
              }
              if (oldlo < hi) SortVertListbyFrame(node, oldlo, hi);
              if (lo < oldhi) SortVertListbyFrame(node, lo, oldhi);
        }
}

Compressor::MarkObjBoundingBox(int x, int y, int ObjNum)
// is the current x/y coord outside the given bounding box? if so,
// update the box.
{
        if (x > GeometryList[ObjNum].iMaxX)
              GeometryList[ObjNum].iMaxX = x;
        if (x < GeometryList[ObjNum].iMinX)
              GeometryList[ObjNum].iMinX = x;
        if (y > GeometryList[ObjNum].iMaxY)
              GeometryList[ObjNum].iMaxY = y;
        if (y < GeometryList[ObjNum].iMinY)
              GeometryList[ObjNum].iMinY = y;
}

Compressor::SortMipMapAppearanceList(int frame)
{
        // Go through all the objects, and determine which mipmap scale
        // their bounding box corresponds to
        for (int obj=0;obj<iGeomCount;obj++)
        {
              int sizex = (GeometryList[obj].iMaxX - GeometryList[obj].iMinX) /
2;
              int sizey = (GeometryList[obj].iMaxY - GeometryList[obj].iMinY) /
2;
              int nearest = (int)sqrt(sizex*sizey);
              int val = MINTEXTURERES;
              for (int i=0;i<MAXTEXTUREPOWER-MINTEXTUREPOWER;i++)
              {
                    val*=2;
                    if (nearest > val && GeometryList[obj].MaterialID != -1 &&
                          MaterialList[GeometryList[obj].MaterialID].iMipNeededO
nFrame[i] > frame)
                          MaterialList[GeometryList[obj].MaterialID].iMipNeededO
nFrame[i] = frame;
              }
        }
}

int Compressor::InitFileForOutput(const TCHAR * filename)
```

```
// builds the file header, and returns its length, since the header
// isn't actually going to be compressed.
{
//    if ((fileOut = fopen(filename, "wb")) == NULL)
      if ((fileOut = fopen("____temp", "wb")) == NULL)
      {
            // error has occured
      }

      // We now send a simple 4 byte header which includes some settings,
      // and a version number for debug purposes and extendability.

      // Now let's create a simple mask to be sent as the second byte.  This
specifies
      // various settings available in the file type.
      unsigned char boolbyte;
      boolbyte = (bCompressNormals << 7) + (bVertexNormals << 6) +
            (bMaterialQuality << 5) + iBitsPerVertex;
      if (iFrameRate != 30)
            boolbyte |= 0x10;
      if (GetTicksPerFrame() != 160)
            boolbyte |= 0x08;

      if (iLightCount == 0 && bUseShading)
            boolbyte |= 0x04;

      cOutToFile(boolbyte);
      cOutToFile(0);                  //   this   byte   reserved   for   future
expandability
      cOutToFile((unsigned char)MAJORVERSION);
      cOutToFile((unsigned char)MINORVERSION);
                                                  // so a Version number of 10
refers to an
                                                  // internal version of 1.0

      if (iFrameRate != 30)
            iOutToFile(iFrameRate);
      if (GetTicksPerFrame() != 160)
            iOutToFile(GetTicksPerFrame() );
      // If the frame rate of the source is NOT 30 frames per second, mark
      // this, and transmit the framerate to aid in interpretation of
      // frame markers.

      // now output 3 bytes which state how long the animation is
      unsigned int diff = iEndFrame - iStartFrame;
      unsigned int uiTime = (unsigned int) diff;
      unsigned char * bTime = (unsigned char*) &uiTime;
      cOutToFile(bTime[0]);
      cOutToFile(bTime[1]);
      cOutToFile(bTime[2]);

      // devise the target resolution, and shove it into 24b
      if (iSampleXres > 8192) iSampleXres = 8192;
      if (iSampleYres > 8192) iSampleYres = 8192;
      unsigned int res = (iSampleXres/2) + ((iSampleYres/2) << 12);
      unsigned char * bRes = (unsigned char*) &res;
      cOutToFile(bRes[0]);
```

```
	cOutToFile(bRes[1]);
	cOutToFile(bRes[2]);

	// reserve 4 bytes to eventually store filelength
	cOutToFile(0);
	cOutToFile(0);
	cOutToFile(0);
	cOutToFile(0);
	return (BytesSoFar);
}

Compressor::CloseOutputFile()
{
	iOutToFile(OPCODE_ENDOFFILEMARKER);
	fclose(fileOut);
}

void Compressor::BuildDataBlock(int uptoframe)
// Builds the data block up until and including the given frame.  All object
// sections and motion information required is transmitted
{
	for (int currobj = 0;currobj < iGeomCount;currobj++)
	{
		GeoNode *node = &(GeometryList[currobj]);
		if (node->nAddedFaceCount > 0)
		{
			// first, if there were any polygons that needed to be
			// added for this current second of animation, build them
			// into the file.
			int lastFaceSent = node->nSentFace;
			int lastVertSent = node->nSentVert;
			for (int count = lastFaceSent+1;count<node->nAddedFaceCount
&&
				node->FaceAddedOnFrame[count]<=uptoframe;count++,node-
>nSentFace++);
			for (count = lastVertSent+1;count<node->nAddedVertCount &&
				node->VertAddedOnFrame[count]<=uptoframe;count++,node-
>nSentVert++);
			TransmitPolyBlock(node,lastFaceSent+1,lastVertSent+1,uptofra
me);
			// now, if any motion paths begin during this second, add
them
			// into the file
			TransmitKeysThisSecond(node,uptoframe);
			continue;
		}
		else
		{
			// move lights around here
			GeoNode* Light;
			static int LastTarKeySent[256],LastLightKeySent[256],start =
TRUE;
			if (start)
			{
				start = FALSE;
				for (int loop=0;loop<256;loop++)
				{
					LastTarKeySent[loop] = 0;
```

```
                        LastLightKeySent[loop] = 0;
                    }
                }

                for (int count = 0;count < iLightCount;count++)
                {
                    Light = &(GeometryList[LightIndexList[count]]);
                    if (Light == node)
                    {
                        if (Light->RefNumber < 1 && ((GenLight*)
Light->curNode)->GetUseLight())
                        {
                            // this light hasn't been added to the
scene, so we must add it
                            // but first, we must figure out what
kind of light we're dealing
                            // with here.
                            switch(((GenLight*)
Light->curNode)->Type())
                            {
                                case OMNI_LIGHT:
                                // "Omnidirectional light" which we
treat as a
                                // simple Ambient light source
                                {
                                    iOutToFile(OPCODE_ADDLIGHT_OMN
I);
                                } break;
                                case TSPOT_LIGHT:
                                // "Targetted Spotlight"
                                case FSPOT_LIGHT:
                                // Free Spotlight...
                                {
                                    iOutToFile(OPCODE_ADDLIGHT_SPO
T);
                                } break;
                                case DIR_LIGHT:
                                // Basic Directional Light
                                case TDIR_LIGHT:
                                // Directional light using a Target
                                {
                                    iOutToFile(OPCODE_ADDLIGHT_DIR
ECTIONAL);
                                } break;
                            }
                            Point3 lightColor = ((GenLight*)
Light->curNode)->GetRGBColor(0);
                            wOutToFile(Convert_16bColor(ConvertRGBtoCo
lorref(lightColor)));

                            Control *PosCon, *RotCon;
                            PosCon =
Light->curNode->GetTMController()->GetPositionController();
                            RotCon =
Light->curNode->GetTMController()->GetRotationController();
                            Control *TarPosCon = NULL;
                            if (Light->curNode->GetTarget() )
```

```
                                        TarPosCon                       =
Light->curNode->GetTarget()->
                                              GetTMController()->GetPosition
Controller();
                                    IKeyControl *poskeys = NULL, *rotkeys =
NULL, *tarposkeys = NULL;
                                    int rcount = 0;
                                    int pcount = 0;
                                    int tcount = 0;

                                    Light->RefNumber = count+1;
                                    if      (PosCon      &&      ((GenLight*)
Light->curNode)->Type()!= TSPOT_LIGHT &&
                                          ((GenLight*)
Light->curNode)->Type()!= FSPOT_LIGHT)
                                    {
                                          poskeys                       =
GetKeyControlInterface(PosCon);
                                          pcount = poskeys->GetNumKeys();
                                    }
                                    if      (RotCon      &&      ((GenLight*)
Light->curNode)->Type()!= OMNI_LIGHT)
                                    {
                                          rotkeys                       =
GetKeyControlInterface(RotCon);
                                          rcount = rotkeys->GetNumKeys();
                                    }
                                    if (TarPosCon)
                                    {
                                          tarposkeys                    =
GetKeyControlInterface(TarPosCon);
                                          tcount = tarposkeys->GetNumKeys();
                                    }
                                    Matrix3               Mat           =
GetLocalMatrix(Light->curNode,0);
                                    if (pcount == 0)
                                    {
                                          iOutToFile(OPCODE_SETLIGHT_POSITION)
;
                                          iOutToFile(Light->RefNumber);
                                          Point3 Position = Mat.GetRow(3);
                                          fOutToFile(Position.x);
                                          fOutToFile(Position.y);
                                          fOutToFile(Position.z);
                                    }
                                    if      (rcount      ==      0      &&
(!Light->curNode->GetTarget() ||
                                          Light->curNode->GetTarget()       &&
tcount == 0))
                                    //  if  its  a  free  camera  with  no
orientation keyframes
                                    // or a target camera whose
                                    {
                                          iOutToFile(OPCODE_SETLIGHT_ORIENTATI
ON);
                                          iOutToFile(Light->RefNumber);
                                          Quat Rotation(Mat);
                                          quatOutToFile(Rotation);
                                    }
```

```
                                    // now determine the offset for the light
                                    Point3            offsetPos                =
Light->curNode->GetObjOffsetPos();
                                    Quat        offsetRot                      =
Light->curNode->GetObjOffsetRot();
                                    ScaleValue  offsetScl                      =
Light->curNode->GetObjOffsetScale();
                                    Matrix3 OffsetTM(1),Scl(1),Tran(1);
                                    ApplyScaling(Scl, offsetScl);
                                    Matrix3 Rot;
                                    offsetRot.MakeMatrix(Rot);
                                    Tran.Translate(offsetPos);
                                    OffsetTM = Scl * Rot * Tran;


                                    if (!CheckIdentity(OffsetTM))
                                    {
                                          iOutToFile(OPCODE_SETLIGHT_OFFSET);
                                          iOutToFile(Light->RefNumber);
                                          for (int l1=0;l1<4;l1++)


                                                for (int l2=0;l2<3;l2++)
                                                      fOutToFile(OffsetTM.GetR
ow(l1)[l2]);
                                    }
                              }
                              if (((GenLight*) Light->curNode)->GetUseLight())
                              {
                                    TransmitKeysThisSecond(Light,uptoframe);
/*                                  if (Light->curNode->GetTarget())
                                          TransmitTargetKeysThisSecond(Light,u
ptoframe,
                                                LastTarKeySent[Light->RefNumbe
r]);*/
                              }
                              break;
                        }
                  }


                  // move cameras around here
                  GeoNode*                        Camera                       =
&(GeometryList[CameraIndexList[nSelectedCamera]]);
                  if (Camera == node)
                  {
                        static int LastTargetKeySent = 0,
                                    LastCameraKeySent = 0;
                        if (Camera->RefNumber != 1)
                        {
                              Control *PosCon, *RotCon;
                              PosCon                                           =
Camera->curNode->GetTMController()->GetPositionController();
                              RotCon                                           =
Camera->curNode->GetTMController()->GetRotationController();
                              IKeyControl *poskeys = NULL;
                              int rcount = 0;
                              int pcount = 0;
```

```
                        IKeyControl *rotkeys = NULL;
                        Camera->RefNumber = 1;
                        if (PosCon)
                        {
                              poskeys = GetKeyControlInterface(PosCon);
                              pcount = poskeys->GetNumKeys();
                        }
                        if (RotCon)
                        {
                              rotkeys = GetKeyControlInterface(RotCon);
                              rcount = rotkeys->GetNumKeys();
                        }
                        // grab controller info for the target
                        Control *TarPosCon = NULL;
                        if (Camera->curNode->GetTarget() )
                              TarPosCon = Camera->curNode->GetTarget()->
                                    GetTMController()->GetPositionContro
ller();
                        IKeyControl *tarposkeys = NULL;
                        int tcount = 0;
                        if (TarPosCon)
                        {
                              tarposkeys                                    =
GetKeyControlInterface(TarPosCon);
                              tcount = tarposkeys->GetNumKeys();
                        }

                        // When the camera has been added, we give it a
reference
                        // number of '1' to indicate this.  So, if the
reference id
                        // is not '1', we know we have to tell the
scene to add the camera
                        if (pcount == 0)
                              iOutToFile(OPCODE_ADDCAMERA_WITHLOCATION);
                        else iOutToFile(OPCODE_ADDCAMERA);
                        CameraObject*     obj     =     (CameraObject*)
Camera->curNode->GetObjectRef();
                        float CameraFov=obj->GetFOV(0);
                        float
nearPlane=obj->GetEnvRange(0,ENV_NEAR_RANGE);
                        float                                           farPlane
=obj->GetEnvRange(0,ENV_FAR_RANGE);

                        // transmit FOV
                        wOutToFile(ReScaleFloat(CameraFov,0.0f,FLOAT_PI,
BPC_16));
                        // transmit near plane
                        fOutToFile(nearPlane);
                        // transmit far plane
                        fOutToFile(farPlane);
                        Matrix3 Mat = GetLocalMatrix(Camera->curNode,0);
                        if (pcount == 0)
                        {
                              Point3 Position = Mat.GetRow(3);
                              fOutToFile(Position.x);
                              fOutToFile(Position.y);
                              fOutToFile(Position.z);
```

```
                              }
                              if        (rcount        ==        0        &&
(!Camera->curNode->GetTarget() ||
                                    (Camera->curNode->GetTarget() && tcount ==
0)))
                                    //  if  its  a  free  camera  with  no
orientation keyframes
                                    // or a target camera whose
                              {
                                    iOutToFile(OPCODE_SETCAMERA_ORIENTATION);
                                    Quat Rotation(Mat);
                                    quatOutToFile(Rotation);
                              }

                              // now determine the offset for the camera
                              Point3            offsetPos                    =
Camera->curNode->GetObjOffsetPos();
                              Quat        offsetRot                          =
Camera->curNode->GetObjOffsetRot();
                              ScaleValue  offsetScl                          =
Camera->curNode->GetObjOffsetScale();
                              Matrix3 OffsetTM(1);
                              OffsetTM.PreTranslate(offsetPos);
                              PreRotateMatrix(OffsetTM, offsetRot);
                              ApplyScaling(OffsetTM, offsetScl);
                              if (!CheckIdentity(OffsetTM))
                              {
                                    iOutToFile(OPCODE_SETCAMERA_OFFSET);
                                    for (int l1=0;l1<4;l1++)
                                          for (int l2=0;l2<3;l2++)
                                                fOutToFile(OffsetTM.GetRow(l1)
[l2]);
                              }
                        }
                        TransmitKeysThisSecond(Camera,uptoframe);
//                      if (Camera->curNode->GetTarget())

//                            TransmitTargetKeysThisSecond(Camera,uptoframe,La
stTargetKeySent,LastCameraKeySent);
                  }
                  // well, its not a light or a camera, but it might be an
                  // object that we need because its a parent to one of
                  // our chosen displayed objects.
                  else if (node->RefNumber > -1)
                        TransmitKeysThisSecond(node,uptoframe);

            }
      }
      MarkCompleteToFrame(uptoframe);
}

void  Compressor::TransmitPolyBlock(GeoNode  *node,  int  startFace,  int
startVert, int FinalFrameThisBlock)
// transmits all polygon-type data for the given frame set
{
      bObjectsAddedThisFrame = FALSE;

      Control *PosCon, *RotCon;
```

```
	PosCon = node->curNode->GetTMController()->GetPositionController();
	RotCon = node->curNode->GetTMController()->GetRotationController();
	IKeyControl *poskeys = NULL;
	IKeyControl *rotkeys = NULL;
	int rcount = 0;
	int pcount = 0;
	if (PosCon)
	{
		poskeys = GetKeyControlInterface(PosCon);
		if (poskeys)
			pcount = poskeys->GetNumKeys();
	}
	if (RotCon)
	{
		rotkeys = GetKeyControlInterface(RotCon);
		if (rotkeys)
			rcount = rotkeys->GetNumKeys();
	}

	Object* tempObj;

//	if (rcount == 0 && pcount == 0)
		tempObj = node->curNode->EvalWorldState(0).obj;
//	else tempObj = node->curNode->GetObjectRef()->Eval(0).obj;
	tempObj = node->curNode->GetObjectRef()->Eval(0).obj;

	TriObject* tempTri = (TriObject*) tempObj->ConvertToType(0,
			Class_ID(TRIOBJ_CLASS_ID, 0));
	BOOL bObjectSentThisFrame = FALSE;

	if (tempTri)
	{
		if (!tempTri->mesh.normalsBuilt)
		tempTri->mesh.buildNormals();
	}


	if (node->RefNumber < 0)
	{
		// if the object hasn't been referenced in the output stream
before,
		// create a new reference number for it, and feed its init info
into
		// the stream
		CreateNewObject(node);
		bObjectsAddedThisFrame = TRUE;
		bObjectSentThisFrame = TRUE;
	}
	// now transmit the face and vert info in blocks

	// first, let's transmit the list of vertices for this second
	// in blocks of 30 or fewer.
	for (int curr = startVert;curr<node->nSentVert+1;curr+=ITEMSPERBLOCK)
	{
		int VertsThisBlock;
		// send the opcode for adding vertices
		iOutToFile(OPCODE_ADDVERTS);
		iOutToFile(node->RefNumber);
```

```
            if (node->nSentVert+1 - curr < ITEMSPERBLOCK)
                  VertsThisBlock = node->nSentVert - curr + 1;
            else
            {
                  VertsThisBlock = ITEMSPERBLOCK;
            }
            iOutToFile(VertsThisBlock);
            switch (iBitsPerVertex)
            {
            case BPC_32:
                  {
                        // *** Note that for 32b, actual full coordinates are
                        // sent, not rescaled coordinates.

                        // first we build a giant block of bytes and create a
                        // virtual float array that points at the same memory
                        // space.  Note that we rotate the floats by one bit,
                        // so that the sign bit becomes part of the mantisa
                        // rather than part of the exponent (which are more
                        // likely to be similar).
                        unsigned char ByteBlock[ITEMSPERBLOCK * 3 * 4];
                        float * SameArray = (float*) ByteBlock;
                        for (int i=curr, j=0;i<curr+VertsThisBlock;i++,j++)
                        {
                              unsigned int RotateMe;
                              SameArray[j*3                              ]          =
tempTri->mesh.verts[node->AddedVertList[i]].x;
                              memcpy(&RotateMe,&SameArray[j*3  ],4);
                              RotateMe = _rotr(RotateMe,1);
                              memcpy(&SameArray[j*3  ],&RotateMe,4);

                              SameArray[j*3+1]                                    =
tempTri->mesh.verts[node->AddedVertList[i]].y;
                              memcpy(&RotateMe,&SameArray[j*3+1],4);
                              RotateMe = _rotr(RotateMe,1);
                              memcpy(&SameArray[j*3+1],&RotateMe,4);

                              SameArray[j*3+2]                                    =
tempTri->mesh.verts[node->AddedVertList[i]].z;
                              memcpy(&RotateMe,&SameArray[j*3+2],4);
                              RotateMe = _rotr(RotateMe,1);
                              memcpy(&SameArray[j*3+2],&RotateMe,4);
                        }
                        // now that the array is full, let's start packing
the bytes
                        // together so that the like ordered bytes are
together

                        unsigned char SecondByteBlock[ITEMSPERBLOCK * 3 * 4];
                        for (j=0;j<12;j++)
                              for (i = 0;i<VertsThisBlock;i++)

                                    SecondByteBlock[j*VertsThisBlock+i]       =
ByteBlock[12*i+j];
                        for (j=0;j<VertsThisBlock*3*4;j++)
                              cOutToFile(SecondByteBlock[j]);
                  }
```

```
                break;
            case BPC_16:
                {     // all X's are sent together, followed by all Y's,
and Z's
                for (int i=curr;i<curr+VertsThisBlock;i++)
                  wOutToFile(ReScaleFloat(
                        tempTri->mesh.verts[node->AddedVertList[i]
].x,
                        node->pMinCoord.x,node->pMaxCoord.x,BPC_16
));
                for (i=curr;i<curr+VertsThisBlock;i++)
                  wOutToFile(ReScaleFloat(
                        tempTri->mesh.verts[node->AddedVertList[i]
].y,
                        node->pMinCoord.y,node->pMaxCoord.y,BPC_16
));
                for (i=curr;i<curr+VertsThisBlock;i++)
                  wOutToFile(ReScaleFloat(
                        tempTri->mesh.verts[node->AddedVertList[i]
].z,
                        node->pMinCoord.z,node->pMaxCoord.z,BPC_16
));
                }
                break;
            case BPC_10:
                {
                for (int i=curr;i<curr+VertsThisBlock;i++)
                  pbOutToFile(ReScaleFloat(
                        tempTri->mesh.verts[node->AddedVertList[i]
].x,
                        node->pMinCoord.x,node->pMaxCoord.x,BPC_10
),
                        10,FALSE);
                for (i=curr;i<curr+VertsThisBlock;i++)
                  pbOutToFile(ReScaleFloat(
                        tempTri->mesh.verts[node->AddedVertList[i]
].y,
                        node->pMinCoord.y,node->pMaxCoord.y,BPC_10
),
                        10,FALSE);
                for (i=curr;i<curr+VertsThisBlock;i++)
                  pbOutToFile(ReScaleFloat(
                        tempTri->mesh.verts[node->AddedVertList[i]
].z,
                        node->pMinCoord.z,node->pMaxCoord.z,BPC_10
),
                        10,i==curr+VertsThisBlock-1);
                }
                break;
            case BPC_8:
                {   // all X's are sent together, followed by all Y's, and
Z's
                for (int i=curr;i<curr+VertsThisBlock;i++)
                  cOutToFile(ReScaleFloat(
                        tempTri->mesh.verts[node->AddedVertList[i]
].x,
                        node->pMinCoord.x,node->pMaxCoord.x,BPC_8)
);
```

```
                        for (i=curr;i<curr+VertsThisBlock;i++)
                              cOutToFile(ReScaleFloat(
                                    tempTri->mesh.verts[node->AddedVertList[i]
].y,
                                    node->pMinCoord.y,node->pMaxCoord.y,BPC_8)
);
                        for (i=curr;i<curr+VertsThisBlock;i++)
                              cOutToFile(ReScaleFloat(
                                    tempTri->mesh.verts[node->AddedVertList[i]
].z,
                                    node->pMinCoord.z,node->pMaxCoord.z,BPC_8)
);
                  } break;
            }

            // Calculate the Error factor in the vertices
            if (iBitsPerVertex != BPC_32)
                  for (int i=curr;i<curr+VertsThisBlock;i++)
                  {
                        float OrigX, OrigY, OrigZ, NewX, NewY, NewZ;
                        OrigX = tempTri->mesh.verts[node->AddedVertList[i]].x;
                        OrigY = tempTri->mesh.verts[node->AddedVertList[i]].y;
                        OrigZ = tempTri->mesh.verts[node->AddedVertList[i]].z;
                        NewX                                                =
ScaleUp(ReScaleFloat(OrigX,node->pMinCoord.x,node->pMaxCoord.x,
                              iBitsPerVertex),node->pMinCoord.x,node->pMaxCoor
d.x,iBitsPerVertex);
                        NewY                                                =
ScaleUp(ReScaleFloat(OrigY,node->pMinCoord.y,node->pMaxCoord.y,
                              iBitsPerVertex),node->pMinCoord.y,node->pMaxCoor
d.y,iBitsPerVertex);
                        NewZ                                                =
ScaleUp(ReScaleFloat(OrigZ,node->pMinCoord.z,node->pMaxCoord.z,
                              iBitsPerVertex),node->pMinCoord.z,node->pMaxCoor
d.z,iBitsPerVertex);

                        float Dist = sqrtf( (OrigX-NewX)*(OrigX-NewX) +
                              (OrigY-NewY)*(OrigY-NewY) + (OrigZ-NewZ)*(OrigZ-
NewZ) );
                        float  Base  =  sqrtf(  OrigX*OrigX  +  OrigY*OrigY  +
OrigZ*OrigZ );
                        if (fabsf(Base) > 0.00001)
                              fTotalError += Dist/Base;
                        iTotalVerts++;
                  }
            else iTotalVerts+=VertsThisBlock;

            // Deal with Normals
            if (bVertexNormals)
            {
                  iOutToFile(OPCODE_ADDNORMS);
                  iOutToFile(node->RefNumber);
                  iOutToFile(VertsThisBlock);
                  if (bCompressNormals)
                  {
                        for (int i=curr;i<curr+VertsThisBlock;i++)
                        {
                              Point3                    Norm                =
```

```
Normalize(tempTri->mesh.getNormal(node->AddedVertList[i]));

                                float theta, psi;
                                if (fabs(Norm.x) < 0.00005)
                                      theta = FLOAT_HALFPI;
                                else theta = atanf(Norm.y/Norm.x);
                                if (fabs(Norm.z) < 0.00005)
                                      psi = FLOAT_HALFPI;
                                else psi = atanf(sqrtf(Norm.x * Norm.x + Norm.y
* Norm.y) / Norm.z);
                                unsigned int compound = 0;
                                unsigned int value;
                                unsigned char * fval = (unsigned char*) &value;
                                compound          =          ReScaleFloat(theta,-
FLOAT_HALFPI,FLOAT_HALFPI,BPC_12);
                                value = (compound << 12) | ReScaleFloat(psi,-
FLOAT_HALFPI,
                                      FLOAT_HALFPI,BPC_12);
                                cOutToFile(fval[0]);
                                cOutToFile(fval[1]);
                                cOutToFile(fval[2]);
                          }
                    }
                    else
                    {
                          for (int i=curr;i<curr+VertsThisBlock;i++)
                            wOutToFile(ReScaleFloat(
                                  Normalize(tempTri->mesh.getNormal(node->Ad
dedVertList[i])).x,
                                  -1.0f,1.0f,BPC_16));
                          for (i=curr;i<curr+VertsThisBlock;i++)
                            wOutToFile(ReScaleFloat(
                                  Normalize(tempTri->mesh.getNormal(node->Ad
dedVertList[i])).y,
                                  -1.0f,1.0f,BPC_16));
                          for (i=curr;i<curr+VertsThisBlock;i++)
                            wOutToFile(ReScaleFloat(
                                  Normalize(tempTri->mesh.getNormal(node->Ad
dedVertList[i])).z,
                                  -1.0f,1.0f,BPC_16));
                    }
              }
        }

        // now let's transmit the faces, in blocks of 30 or fewer
        for (curr = startFace;curr<node->nSentFace+1;curr+=ITEMSPERBLOCK)
        {
              int FacesThisBlock;
              // send the opcode for adding vertices
              iOutToFile(OPCODE_ADDFACES);
              iOutToFile(node->RefNumber);
              if (node->nSentFace+1 - curr < ITEMSPERBLOCK)
                    FacesThisBlock = node->nSentFace - curr + 1;
              else FacesThisBlock = ITEMSPERBLOCK;
              iOutToFile(FacesThisBlock);
              for (int i=curr;i<curr+FacesThisBlock;i++)
              {
                    int location;
```

```
                    // Note that a linear search, though slow and inefficient,
is
                    // required here due to our having already sorted the Verts
                    // based on frame added.  Thus the verts are no longer in
order
                    // and we therefore cannot do a binary search.  Yet, it was
                    // more critical to have the speedup in the initial search,
                    // so thus we use it there rather than here.

                    location                                                =
LinVertSearch(node,node->TrueFaceList[node->AddedFaceList[i]].Vert1);
                    if (node->AddedVertList[location] !=
                          node->TrueFaceList[node->AddedFaceList[i]].Vert1)
                          DebugPrint("Error in data stream... Vertices didn't
match!\n");
                    iOutToFile(location);
                    location                                                =
LinVertSearch(node,node->TrueFaceList[node->AddedFaceList[i]].Vert2);
                    if (node->AddedVertList[location] !=
                          node->TrueFaceList[node->AddedFaceList[i]].Vert2)
                          DebugPrint("Error in data stream... Vertices didn't
match!\n");
                    iOutToFile(location);
                    location                                                =
LinVertSearch(node,node->TrueFaceList[node->AddedFaceList[i]].Vert3);
                    if (node->AddedVertList[location] !=
                          node->TrueFaceList[node->AddedFaceList[i]].Vert3)
                          DebugPrint("Error in data stream... Vertices didn't
match!\n");
                    iOutToFile(location);
              }
        }

        // now transmit any needed texture information
        if (bMaterialQuality && node->MaterialID != -1)
        {
              // high quality materials - use texture exporter
              // begin by transmitting U/V texture coordinates for
              // each sent vertex by using 12 bits per part (24b for
              // the U/V set per vertex.

              // now let's transmit the U/V coordinates, in blocks of 30 or
fewer
              for (curr = startVert;curr<node->nSentVert+1;curr+=ITEMSPERBLOCK)
              {
                    int VertsThisBlock;
                    // send the opcode for adding vertices
                    iOutToFile(OPCODE_ADDTEXCOORDS);
                    iOutToFile(node->RefNumber);
                    if (node->nSentVert+1 - curr < ITEMSPERBLOCK)
                          VertsThisBlock = node->nSentVert - curr + 1;
                    else VertsThisBlock = ITEMSPERBLOCK;
                    iOutToFile(VertsThisBlock);
                    for (int i=curr;i<curr+VertsThisBlock;i++)
                    {
                          float                                texU          =
tempTri->mesh.tVerts[node->TexIndexList[node->AddedVertList[i]]].x;
```

```
                        float                           texV                    =
-tempTri->mesh.tVerts[node->TexIndexList[node->AddedVertList[i]]].y;
                        unsigned int texcoord =
                              ReScaleFloat(texU,-8.0f,8.0f,BPC_12) << 12;
                        texcoord |= ReScaleFloat(texV,-8.0f,8.0f,BPC_12);
                        unsigned char *tc = (unsigned char*)&texcoord;
                        cOutToFile(tc[0]);
                        cOutToFile(tc[1]);
                        cOutToFile(tc[2]);
                  }
            }

            if (MaterialList[node->MaterialID].iLastTexturePowerSent == -1)
            // texture hasn't been sent at all yet, so we need to build the
            // base texture first
            {
                  BuildWaveletCoeffs(&(MaterialList[node->MaterialID]));
                  MaterialList[node->MaterialID].iLastTexturePowerSent      =
MINTEXTUREPOWER;
            // if highres textures are being used, we must sample to
determine
            // what texture resolutions are actually required
                  MaterialList[node->MaterialID].iMaxNecessaryTexPower      =
iDesiredTexturePower;
                  DetermineNecessaryTexRes(&(MaterialList[node->MaterialID]));

                  // let's finally delta encode the whole group, following a
hilbert
                  // curve pattern to traverse the texture.  Note that the
8x8
                  // approximation is left alone and not delta encoded to
provide
                  // for quick and simple reconstruction on the other end.
/*                for                                                 (int
i=MAXTEXTUREPOWER,j=MAXTEXTURERES;i>MINTEXTUREPOWER;i--,j/=2)
                        DeltaEncode(MaterialList[node->MaterialID].WaveletCoef
f,i,j);
                  FileFloatDump("_de",MaterialList[node->MaterialID].WaveletCo
eff,MAXTEXTURERES,3);*/

                  GetCoeffRange(MaterialList[node->MaterialID].WaveletCoeff,
                        MaterialList[node->MaterialID].fTexmapMinVal,
                        MaterialList[node->MaterialID].fTexmapMaxVal);
                  // transmit the shininess characteristics
                  float                           shine0                  =
((StdMat*)MaterialList[node->MaterialID].curNode)->GetShininess(0);
                  float                           shine1                  =
((StdMat*)MaterialList[node->MaterialID].curNode)->GetShinStr(0);
                  float shine = shine0*shine1;
                  Color                            spec                   =
((StdMat*)MaterialList[node->MaterialID].curNode)->GetSpecular(0);

                  iOutToFile(OPCODE_ADDTEXMAPTYPE_HAAR);
                  fOutToFile(MaterialList[node->MaterialID].fTexmapMinVal);
                  fOutToFile(MaterialList[node->MaterialID].fTexmapMaxVal);
                  cOutToFile(ReScaleFloat(shine,0.0f,1.0f,BPC_8));
                  wOutToFile(Convert_16bColor(ConvertRGBtoColorref(spec)));
```

```
                if (MaterialList[node->MaterialID].RefNumber == -1)
                        MaterialList[node->MaterialID].RefNumber =
++nLastMatRefNum;
            }
            while (MaterialList[node->MaterialID].iMipNeededOnFrame[
                        MaterialList[node->MaterialID].iLastTexturePowerSent-
MINTEXTUREPOWER] <=
                        FinalFrameThisBlock &&
                    MaterialList[node->MaterialID].iLastTexturePowerSent <=
                    MaterialList[node->MaterialID].iMaxNecessaryTexPower &&
                    !MaterialList[node->MaterialID].bMipSent[
                        MaterialList[node->MaterialID].iLastTexturePowerSent-
MINTEXTUREPOWER])
                    AddTextureCoeffs(&(MaterialList[node->MaterialID]));
            if (!node->bMaterialAssigned)
            {
                    iOutToFile(OPCODE_ASSIGN_MAPTOOBJ);
                    iOutToFile(node->RefNumber);
                    iOutToFile(MaterialList[node->MaterialID].RefNumber);
                    node->bMaterialAssigned = TRUE;
            }
        }
        else
        {
            // low quality materials - use color exporter
            StdMat* nodeMtl= (StdMat*) node->curNode->GetMtl();
            if (!nodeMtl || nodeMtl->GetWire())
            {
                    if (bObjectSentThisFrame)
                    {
                        // if the object is using a wirecolor, just assign
the whole
                        // object that wirecolor.
                        COLORREF objColor = node->curNode->GetWireColor();
                        iOutToFile(OPCODE_ASSIGNOBJECT_SOLIDCOLOR);
                        iOutToFile(node->RefNumber);
                        wOutToFile(Convert_16bColor(objColor));
                    }
            }
            else
            {
                    // Otherwise, we need to sample colors at each vertex
                    // of the object, and export these.

                    // The technique we shall use will be to create a
supertexture
                    // for the object, and sample texel values from the texture
corresponding
                    // to the provided texture coordinates

                    if (!MaterialList[node->MaterialID].pTextureMap)
                    {
                        MaterialList[node->MaterialID].pTextureMap = new
unsigned char[MAXTEXTURERES*MAXTEXTURERES*3];
                        RetrieveOriginalMap(&(MaterialList[node->MaterialID]),
                            MaterialList[node->MaterialID].pTextureMap);
                    }
                    for (curr =
```

```
startVert;curr<node->nSentVert+1;curr+=ITEMSPERBLOCK)
                    {
                          int VertsThisBlock;
                          // send the opcode for adding vertices
                          iOutToFile(OPCODE_ADDVERTCOLORS);
                          iOutToFile(node->RefNumber);
                          if (node->nSentVert - curr < ITEMSPERBLOCK)
                                VertsThisBlock = node->nSentVert - curr + 1;
                          else VertsThisBlock = ITEMSPERBLOCK;
                          iOutToFile(VertsThisBlock);
                          for (int i=curr;i<curr+VertsThisBlock;i++)
                          {
                                float                    texU                    =
tempTri->mesh.tVerts[node->TexIndexList[node->AddedVertList[i]]].x;
                                float                    texV                    =
tempTri->mesh.tVerts[node->TexIndexList[node->AddedVertList[i]]].y;
                                Point3 vertcolor = RetrieveTexelColor(texU,texV,
                                      MaterialList[node->MaterialID].pTextureMap
) / 255.0f;
                                wOutToFile(Convert_16bColor(ConvertRGBtoColorref
(vertcolor)));
                          }
                    }

              }
        }
}

Compressor::aaOutToFile(AngAxis aa)
// writes out an angle/axis value
{
//    wOutToFile(ReScaleFloat(aa.angle,-FLOAT_HALFPI,FLOAT_HALFPI,BPC_16));
      fOutToFile(aa.angle);
      wOutToFile(ReScaleFloat(aa.axis.x,-1.0f,1.0f,BPC_16));
      wOutToFile(ReScaleFloat(aa.axis.y,-1.0f,1.0f,BPC_16));
      wOutToFile(ReScaleFloat(aa.axis.z,-1.0f,1.0f,BPC_16));
}

Compressor::cOutToFile(unsigned char cval)
// writes out a single character
{
      fputc(cval,fileOut);
      BytesSoFar++;
}

Compressor::iOutToFile(int ival)
// writes out a custom short - uses one byte for values below 200, two
// for values of 200 or greater
{
      assert(ival<11400);
      if (ival<200)
            cOutToFile((unsigned char)ival);
      else
      {
            int if1 = ival / 200;
            int if2 = ival % 200;
            cOutToFile((unsigned char)if1+199);
            cOutToFile((unsigned char)if2);
```

```
        }
}

void Compressor::fOutToFile(float fval)
// writes a raw float out to to the file
{
        unsigned char * fakearr = (unsigned char *) &fval;
        cOutToFile(fakearr[0]);
        cOutToFile(fakearr[1]);
        cOutToFile(fakearr[2]);
        cOutToFile(fakearr[3]);
}


Compressor::wOutToFile(int ival)
// writes out a 16 bit word
{
        unsigned short wval = (unsigned short) ival;
        unsigned char * bval = (unsigned char*) &(wval);
        cOutToFile(bval[0]);
        cOutToFile(bval[1]);
}

Compressor::quatOutToFile(Quat q)
// writes out a 4 float quaternion, range compressing the values
// so that only 8 bytes are required
{
        // instead of sending the quaternion as a full 4 floats,
        // we first normalize the Quat, reducing all 4 components
        // to the -1 to 1 range.  We then scale these values to
        // fit within a given integer range.
        q.Normalize();
        wOutToFile(ReScaleFloat(q.x,-1.0f,1.0f,BPC_16));
        wOutToFile(ReScaleFloat(q.y,-1.0f,1.0f,BPC_16));
        wOutToFile(ReScaleFloat(q.z,-1.0f,1.0f,BPC_16));
        wOutToFile(ReScaleFloat(q.w,-1.0f,1.0f,BPC_16));
}

Compressor::pbOutToFile(unsigned int code,int bitlength, BOOL bFinish)
// Outputs bit codes one byte at a time.
{
        static int OutputBitCount = 0;
        static unsigned int OutputBitBuffer = 0;

        OutputBitBuffer |= (unsigned long) code << (32-bitlength-
OutputBitCount);
        OutputBitCount += bitlength;
        while(OutputBitCount >= 8)
        {
                cOutToFile(OutputBitBuffer >> 24);
                OutputBitBuffer <<= 8;
                OutputBitCount -= 8;
                iCompressedCount++;
        }
        if (bFinish && OutputBitCount > 0)
        {
                cOutToFile(OutputBitBuffer >> 24);
                OutputBitBuffer <<= 8;
```

```
            OutputBitCount -= 8;
            iCompressedCount++;
            OutputBitCount = 0;
            OutputBitBuffer = 0;
      }
}

FrameList * Compressor::GetFramePtrFromTime(int t)
{
      for (FrameList* curr = frHead;curr && curr->frameNum != t;curr =
curr->next);
      return curr;
}

Compressor::TransmitKeysThisSecond(GeoNode *node, int frame)
// The name pretty much speaks for what this does
{
      Control *PosCon;
      PosCon = node->curNode->GetTMController()->GetPositionController();
      IKeyControl *poskeys = NULL;
      Control *RotCon;
      RotCon = node->curNode->GetTMController()->GetRotationController();
      IKeyControl *rotkeys = NULL;
      Control *SclCon;
      SclCon = node->curNode->GetTMController()->GetScaleController();
      IKeyControl *sclkeys = NULL;
      Control *RllCon;
      RllCon = node->curNode->GetTMController()->GetRollController();
      IKeyControl *rllkeys = NULL;
      int pcount = 0, rcount = 0, scount = 0, lcount = 0;

      if (PosCon)
            poskeys = GetKeyControlInterface(PosCon);
      if (RotCon)
            rotkeys = GetKeyControlInterface(RotCon);
      if (SclCon)
            sclkeys = GetKeyControlInterface(SclCon);
      if (RllCon)
            rllkeys = GetKeyControlInterface(RllCon);

      // TCB stands for Tension Continuity and Bias

      ITCBPoint3Key     tcbPosKey;
      IBezPoint3Key     bezPosKey;
      ILinPoint3Key     linPosKey;
      ITCBRotKey        tcbRotKey;
      IBezQuatKey       bezRotKey;
      ILinRotKey        linRotKey;
      ITCBScaleKey      tcbScaleKey;
      IBezScaleKey      bezScaleKey;
      ILinScaleKey      linScaleKey;
      ITCBFloatKey      tcbFloatKey;
      IBezFloatKey      bezFloatKey;
      ILinFloatKey      linFloatKey;
//    IKey  *newkey;

      Point3      pvalue;
```

```
      // now, what we do is we go thru all the keys, and let's say we have n
keys
      // that fall within the given time.  We add these N keys to the object,
along
      // with the N+1'th key as well, so we have a final point to interpolate
to.
      // so basically the scheme is to always send the keys that reach to the
end
      // of the current time sequence PLUS the next key as well.  As keys are
sent,
      // mark them as such.

      // note, it may be possible to use a built in hidden array called
      // 'keys' as part of the controller, to access them.  Consider this.
      // its easier than jumping thru them.

      BOOL bIsLight = FALSE;
      int  iLightNum = -1;
      for (int count=0;count<iLightCount;count++)
            if        ((GeometryList[LightIndexList[count]]).curNode        ==
node->curNode)
            {
                  bIsLight = TRUE;
                  iLightNum = count;
                  break;
            }

      // process the position controll for the object
      if (poskeys)
      {
            int numposkeys = poskeys->GetNumKeys();

            int LastKeyToSend;
            for (int j = node->LastPosKeySent; j < numposkeys &&
                  PosCon->GetKeyTime(j)                                    <
(frame+GetFrameRate())*GetTicksPerFrame(); j++);
            if (j < numposkeys) j++;
            LastKeyToSend = j;

            BOOL        bPosKnown        =        PosCon->ClassID()        ==
Class_ID(TCBINTERP_POSITION_CLASS_ID, 0);
            bPosKnown            |=            PosCon->ClassID()            ==
Class_ID(LININTERP_POSITION_CLASS_ID, 0);
            bPosKnown            |=            PosCon->ClassID()            ==
Class_ID(HYBRIDINTERP_POSITION_CLASS_ID, 0);
            BOOL          b2          =          PosCon->ClassID()          ==
Class_ID(POSITIONNOISE_CONTROL_CLASS_ID,0);
            BOOL          b3          =          PosCon->ClassID()          ==
Class_ID(EXPR_POS_CONTROL_CLASS_ID,0);
            BOOL b4 = PosCon->ClassID() == Class_ID(PATH_CONTROL_CLASS_ID,0);

            if (bPosKnown)
            {
                  for (int i = node->LastPosKeySent; i < LastKeyToSend; i++)
                  {
                        int  fTime  =  PosCon->GetKeyTime(i)  -  (iStartFrame  -
intPtr->GetAnimRange().Start()) *
```

```
                        GetTicksPerFrame();
                    // if we're scrolling thru keyframes that take place
before the start of
                    // the used animation sequence
                    if   ((fTime   <   0   &&   i   <   numposkeys-1   &&
PosCon->GetKeyTime(i+1) -
                        (iStartFrame - intPtr->GetAnimRange().Start()) *
GetTicksPerFrame() < 0) ||
                    // if the keyframes extend on past our segment of
concern, ignor them as well
                        (fTime > 0 && i > 0 &&
                        (PosCon->GetKeyTime(i-1)/GetTicksPerFrame() -
                            intPtr->GetAnimRange().Start()          >
iEndFrame) &&
                        (PosCon->GetKeyTime(i)/GetTicksPerFrame() -
                            intPtr->GetAnimRange().Start()          >
iEndFrame)))
                        continue;
                    unsigned char * bTime = (unsigned char*) &fTime;
                    if              (PosCon->ClassID()              ==
Class_ID(TCBINTERP_POSITION_CLASS_ID, 0))
                    {
                        if (bIsLight)
                        {
                            iOutToFile(OPCODE_LIGHT_KEYFRAME_TCBPOS);
                            iOutToFile(node->RefNumber);
                        }
                        else        if        (node->curNode        ==
CameraPtrList[nSelectedCamera])
                            iOutToFile(OPCODE_CAMERA_KEYFRAME_TCBPOS);
                        else
                        {
                            iOutToFile(OPCODE_KEYFRAME_TCBPOS);
                            iOutToFile(node->RefNumber);
                        }
                        poskeys->GetKey(i, &tcbPosKey);
                        // transmit a 32b time identifier
                        cOutToFile(bTime[0]);
                        cOutToFile(bTime[1]);
                        cOutToFile(bTime[2]);
                        cOutToFile(bTime[3]);
                        // transmit the position
                        fOutToFile(tcbPosKey.val.x);
                        fOutToFile(tcbPosKey.val.y);
                        fOutToFile(tcbPosKey.val.z);
                        // now transmit the tension, continuity, and
bias settings but
                        // first rescale them from the -1 to 1 range so
they fit in
                        // 16b each
                        wOutToFile(ReScaleFloat(tcbPosKey.tens,-
1.0f,1.0f,BPC_16));
                        wOutToFile(ReScaleFloat(tcbPosKey.cont,-
1.0f,1.0f,BPC_16));
                        wOutToFile(ReScaleFloat(tcbPosKey.bias,-
1.0f,1.0f,BPC_16));
                        wOutToFile(ReScaleFloat(tcbPosKey.easeIn,-
1.0f,1.0f,BPC_16));
```

```
                        wOutToFile(ReScaleFloat(tcbPosKey.easeOut,-
1.0f,1.0f,BPC_16));
                }
                else if (PosCon->ClassID() ==
Class_ID(LININTERP_POSITION_CLASS_ID, 0))
                {
                        poskeys->GetKey(i, &linPosKey);
        //              &linPosKey = newkey;
                        if (bIsLight)
                        {
                                iOutToFile(OPCODE_LIGHT_KEYFRAME_LINPOS);
                                iOutToFile(node->RefNumber);
                        }
                        else if (node->curNode ==
CameraPtrList[nSelectedCamera])
                                iOutToFile(OPCODE_CAMERA_KEYFRAME_LINPOS);
                        else
                        {
                                iOutToFile(OPCODE_KEYFRAME_LINPOS);
                                iOutToFile(node->RefNumber);
                        }
                        // transmit a 32b time identifier
                        cOutToFile(bTime[0]);
                        cOutToFile(bTime[1]);
                        cOutToFile(bTime[2]);
                        cOutToFile(bTime[3]);
                        // transmit the position
                        fOutToFile(linPosKey.val.x);
                        fOutToFile(linPosKey.val.y);
                        fOutToFile(linPosKey.val.z);
                }
                else if (PosCon->ClassID() ==
Class_ID(HYBRIDINTERP_POSITION_CLASS_ID, 0))
                {
                        poskeys->GetKey(i, &bezPosKey);
                        int cTime,pTime,nTime;
                        cTime = PosCon->GetKeyTime(i);

                        if (i==0)
                                pTime = cTime;

                        else pTime = PosCon->GetKeyTime(i-1);

                        if (i==poskeys->GetNumKeys()-1)
                                nTime = cTime;

                        else nTime = PosCon->GetKeyTime(i+1);

                        float pScaleFactor = -float(cTime - pTime);
                        float nScaleFactor = float(nTime - cTime);

                        if (bIsLight)
                        {
                                iOutToFile(OPCODE_LIGHT_KEYFRAME_BEZPOS);
                                iOutToFile(node->RefNumber);
                        }
                        else if (node->curNode ==
CameraPtrList[nSelectedCamera])
```

```
                                    iOutToFile(OPCODE_CAMERA_KEYFRAME_BEZPOS);
                              else
                              {
                                    iOutToFile(OPCODE_KEYFRAME_BEZPOS);
                                    iOutToFile(node->RefNumber);
                              }
                              // transmit a 32b time identifier
                              cOutToFile(bTime[0]);
                              cOutToFile(bTime[1]);
                              cOutToFile(bTime[2]);
                              cOutToFile(bTime[3]);
                              // transmit the position
                              fOutToFile(bezPosKey.val.x);
                              fOutToFile(bezPosKey.val.y);
                              fOutToFile(bezPosKey.val.z);
                              // transmit the tangent vectors
                              fOutToFile(bezPosKey.intan.x * pScaleFactor );
                              fOutToFile(bezPosKey.intan.y * pScaleFactor );
                              fOutToFile(bezPosKey.intan.z * pScaleFactor );
                              fOutToFile(bezPosKey.outtan.x * nScaleFactor );
                              fOutToFile(bezPosKey.outtan.y * nScaleFactor );
                              fOutToFile(bezPosKey.outtan.z * nScaleFactor );
                        }
                  }
                  node->LastPosKeySent = LastKeyToSend;
            }
            else
            {
                  // for non-standard position controllers, we instead sample
the position
                  // at 15 times in the last second

                  int currstart = (frame-1) * GetFrameRate();
                  int currend = frame * GetFrameRate();
                  int inc = GetFrameRate() / 15;
                  Matrix3 previousTM ;
                  if (node->LastPosKeySent > 0)
                        previousTM                                          =
node->curNode->GetNodeTM(node->LastPosKeySent);
                  for (int time=currstart;time<currend;time+=inc)
                  {
                        Matrix3 tm = node->curNode->GetNodeTM(time);
                        Point3 pos = tm.GetTrans();
                        Matrix3 nextTM = node->curNode->GetNodeTM(time+inc);
                        unsigned char * bTime = (unsigned char*) &time;

                        if      (!(previousTM.GetTrans()      ==      pos      &&
nextTM.GetTrans() == pos))
                        {
                              iOutToFile(OPCODE_KEYFRAME_LINPOS);
                              iOutToFile(node->RefNumber);
                              // transmit a 32b time identifier
                              cOutToFile(bTime[0]);
                              cOutToFile(bTime[1]);
                              cOutToFile(bTime[2]);
                              cOutToFile(bTime[3]);
                              // transmit the position
                              fOutToFile(pos.x);
```

```
                        fOutToFile(pos.y);
                        fOutToFile(pos.z);
                    }
                }
                node->LastPosKeySent = currend;
            }
        }

        // process the rotation controll for the object
        if (rotkeys)
        {
            int numrotkeys = rotkeys->GetNumKeys();

            int LastKeyToSend;
            for (int j = node->LastRotKeySent; j < numrotkeys &&
                RotCon->GetKeyTime(j)                                    <
(frame+GetFrameRate())*GetTicksPerFrame(); j++);
            if (j < numrotkeys) j++;
            LastKeyToSend = j;

            BOOL        bRotKnown        =        RotCon->ClassID()        ==
Class_ID(TCBINTERP_ROTATION_CLASS_ID, 0) ||
                RotCon->ClassID()  ==  Class_ID(LININTERP_ROTATION_CLASS_ID,
0) ||
                RotCon->ClassID()                                        ==
Class_ID(HYBRIDINTERP_ROTATION_CLASS_ID, 0);

            if (bRotKnown)
            {
                for (int i = node->LastRotKeySent; i < LastKeyToSend; i++)
                {
                    int  fTime  =  RotCon->GetKeyTime(i)  -  (iStartFrame  -
intPtr->GetAnimRange().Start()) *
                            GetTicksPerFrame();
                    unsigned char * bTime = (unsigned char*) &fTime;

                    if                (RotCon->ClassID()                ==
Class_ID(TCBINTERP_ROTATION_CLASS_ID, 0))
                    {
                        rotkeys->GetKey(i, &tcbRotKey);

                        if (bIsLight)
                        {
                            iOutToFile(OPCODE_LIGHT_KEYFRAME_TCBROT);
                            iOutToFile(node->RefNumber);
                        }
                        else        if        (node->curNode        ==
CameraPtrList[nSelectedCamera])
                            iOutToFile(OPCODE_CAMERA_KEYFRAME_TCBROT);
                        else
                        {
                            iOutToFile(OPCODE_KEYFRAME_TCBROT);
                            iOutToFile(node->RefNumber);
                        }
                        // transmit a 32b time identifier
                        cOutToFile(bTime[0]);
                        cOutToFile(bTime[1]);
                        cOutToFile(bTime[2]);
```

```
                        cOutToFile(bTime[3]);
                        // transmit the rotation
                        aaOutToFile(tcbRotKey.val);
                        wOutToFile(ReScaleFloat(tcbRotKey.tens,-
1.0f,1.0f,BPC_16));
                        wOutToFile(ReScaleFloat(tcbRotKey.cont,-
1.0f,1.0f,BPC_16));
                        wOutToFile(ReScaleFloat(tcbRotKey.bias,-
1.0f,1.0f,BPC_16));
                        wOutToFile(ReScaleFloat(tcbRotKey.easeIn,-
1.0f,1.0f,BPC_16));
                        wOutToFile(ReScaleFloat(tcbRotKey.easeOut,-
1.0f,1.0f,BPC_16));
                  }
                  else
                  {
                  // if we're scrolling thru keyframes that take place
before the start of
                  // the used animation sequence
                        if (fTime < 0 && i < numrotkeys-1 &&
RotCon->GetKeyTime(i+1) -
                              (iStartFrame                          -
intPtr->GetAnimRange().Start()) * GetTicksPerFrame() < 0)
                              continue;

                        rotkeys->GetKey(i, &linRotKey);
                        if (bIsLight)
                        {
                              iOutToFile(OPCODE_LIGHT_KEYFRAME_LINROT);
                              iOutToFile(node->RefNumber);
                        }
                        else        if        (node->curNode        ==
CameraPtrList[nSelectedCamera])
                              iOutToFile(OPCODE_CAMERA_KEYFRAME_LINROT);
                        else
                        {
                              iOutToFile(OPCODE_KEYFRAME_LINROT);
                              iOutToFile(node->RefNumber);
                        }
                        // transmit a 32b time identifier
                        cOutToFile(bTime[0]);
                        cOutToFile(bTime[1]);
                        cOutToFile(bTime[2]);
                        cOutToFile(bTime[3]);
                        // transmit the rotation
                        Matrix3 Transform(1);
                        for (int count=0;count<i+1;count++)
                        {
                              rotkeys->GetKey(count, &linRotKey);
                              Quat q(linRotKey.val);
                              Quat qi(Inverse(q));
                              Matrix3 rot;
                              qi.MakeMatrix(rot);
                              Matrix3 Result = rot * Transform;
                              Quat qr(Result);
                              Transform = Result;
                        }
                        Quat qout(Transform);
```

```
                        quatOutToFile(qout);
                    }
                }
                node->LastRotKeySent = LastKeyToSend;
            }
        }

        // process the roll controller for the object, if it has one
        if (rllkeys)
        {
            int numrllkeys = rllkeys->GetNumKeys();

            int LastKeyToSend;
            for (int j = node->LastRllKeySent; j < numrllkeys &&
                RllCon->GetKeyTime(j)                                    <
(frame+GetFrameRate())*GetTicksPerFrame(); j++);
            if (j < numrllkeys) j++;
            LastKeyToSend = j;

            for (int i = node->LastRllKeySent; i < LastKeyToSend; i++)
            {
//              int fTime = RllCon->GetKeyTime(i);
                int  fTime  =  RllCon->GetKeyTime(i)  -  (iStartFrame  -
intPtr->GetAnimRange().Start()) *
                    GetTicksPerFrame();
                // if we're scrolling thru keyframes that take place before
the start of
                // the used animation sequence
                if (fTime < 0 && i < numrllkeys-1 && RllCon->GetKeyTime(i+1)
-
                    (iStartFrame  -  intPtr->GetAnimRange().Start())  *
GetTicksPerFrame() < 0)
                    continue;
                unsigned char * bTime = (unsigned char*) &fTime;

                if (RllCon->ClassID() == Class_ID(TCBINTERP_FLOAT_CLASS_ID,
0))
                {
                    rllkeys->GetKey(i, &tcbFloatKey);
                    if (bIsLight)
                    {
                        iOutToFile(OPCODE_LIGHT_KEYFRAME_TCBRLL);
                        iOutToFile(node->RefNumber);
                    }
                    else          if          (node->curNode          ==
CameraPtrList[nSelectedCamera])
                        iOutToFile(OPCODE_CAMERA_KEYFRAME_TCBRLL);
                    else
                    {
                        iOutToFile(OPCODE_KEYFRAME_TCBRLL);
                        iOutToFile(node->RefNumber);
                    }
                    // transmit a 32b time identifier
                    cOutToFile(bTime[0]);
                    cOutToFile(bTime[1]);
                    cOutToFile(bTime[2]);
                    cOutToFile(bTime[3]);
                    // transmit the roll
```

```
                        fOutToFile(tcbFloatKey.val);
                    }
                    else            if            (RllCon->ClassID()            ==
Class_ID(LININTERP_FLOAT_CLASS_ID, 0))
                    {
                        rllkeys->GetKey(i, &linFloatKey);
                        if (bIsLight)
                        {
                            iOutToFile(OPCODE_LIGHT_KEYFRAME_LINRLL);
                            iOutToFile(node->RefNumber);
                        }
                        else            if            (node->curNode            ==
CameraPtrList[nSelectedCamera])
                            iOutToFile(OPCODE_CAMERA_KEYFRAME_LINRLL);
                        else
                        {
                            iOutToFile(OPCODE_KEYFRAME_LINRLL);
                            iOutToFile(node->RefNumber);
                        }
                        // transmit a 32b time identifier
                        cOutToFile(bTime[0]);
                        cOutToFile(bTime[1]);
                        cOutToFile(bTime[2]);
                        cOutToFile(bTime[3]);
                        // transmit the roll
                        fOutToFile(linFloatKey.val);
                    }
                    else            if            (RllCon->ClassID()            ==
Class_ID(HYBRIDINTERP_FLOAT_CLASS_ID, 0))
                    {
                        rllkeys->GetKey(i, &bezFloatKey);
                        if (bIsLight)
                        {
                            iOutToFile(OPCODE_LIGHT_KEYFRAME_BEZRLL);
                            iOutToFile(node->RefNumber);
                        }
                        else            if            (node->curNode            ==
CameraPtrList[nSelectedCamera])
                            iOutToFile(OPCODE_CAMERA_KEYFRAME_BEZRLL);
                        else
                        {
                            iOutToFile(OPCODE_KEYFRAME_BEZRLL);
                            iOutToFile(node->RefNumber);
                        }
                        // transmit a 32b time identifier
                        cOutToFile(bTime[0]);
                        cOutToFile(bTime[1]);
                        cOutToFile(bTime[2]);
                        cOutToFile(bTime[3]);
                        // transmit the roll
                        fOutToFile(bezFloatKey.val);
                    }
                }
                node->LastRllKeySent = LastKeyToSend;
            }

            // if the object has a target, we assign it now, making sure
            // to add a scene reference for the object, if one didn't previously
```

```
	// exist.  Of course, the target isn't necessary if it
	// never actually moves
	Control *TarPosCon = NULL;
	if (node->curNode->GetTarget() )
		TarPosCon = node->curNode->GetTarget()->
			GetTMController()->GetPositionController();
	IKeyControl *tarposkeys = NULL;
	int tcount = 0;
	if (TarPosCon)
	{
		tarposkeys = GetKeyControlInterface(TarPosCon);
		tcount = tarposkeys->GetNumKeys();
	}
	INode* ParentNode  = NULL;
	if (node->curNode->GetTarget())
		ParentNode = node->curNode->GetTarget()->GetParentNode();
	while (tcount == 0 && ParentNode)
	{
		if (ParentNode->GetTMController() &&
ParentNode->GetTMController()->
			GetPositionController())
		{
			tarposkeys = GetKeyControlInterface(ParentNode->
				GetTMController()->GetPositionController());
			tcount = tarposkeys->GetNumKeys();
		}
		ParentNode = ParentNode->GetParentNode();
	}


	if (node->curNode->GetTarget() && tcount > 0)
	{
		if (!node->IsAssignedTarget)
		{
			node->IsAssignedTarget = TRUE;
			if (GeometryList[node->TargetIdx].RefNumber == -1)
				CreateNewObject(&(GeometryList[node->TargetIdx]));
			if (bIsLight)
			{
				iOutToFile(OPCODE_ADDTARGET_TO_LIGHT);
				iOutToFile(node->RefNumber);
			}
			else if (node->curNode == CameraPtrList[nSelectedCamera])
				iOutToFile(OPCODE_ADDTARGET_TO_CAMERA);
			else
			{
				iOutToFile(OPCODE_ADDTARGET_TO_OBJECT);
				iOutToFile(node->RefNumber);
			}
			iOutToFile(GeometryList[node->TargetIdx].RefNumber);
		}
		// since the object might not actually be visible (it could just
be a
		// point in space) we must make sure to process its keys
preemptively.
		// if the object would normally get processed anyway, it gets its
keys
		// processed early now.
```

```
            TransmitKeysThisSecond(&(GeometryList[node->TargetIdx]),frame);
      }
}

Compressor::MarkCompleteToFrame(int frame)
{
      // mark that everything to this given frame
      // is known to be fine.
      iOutToFile(OPCODE_COMPLETETOFRAME);
      unsigned int uiTime = (unsigned int) frame - (iStartFrame -
intPtr->GetAnimRange().Start());
      unsigned char * bTime = (unsigned char*) &uiTime;
      int a = intPtr->GetAnimRange().Start();
      cOutToFile(bTime[0]);
      cOutToFile(bTime[1]);
      cOutToFile(bTime[2]);
}

Compressor::CleanUpRendering(int framenumber, int SampleRate, int Sample)
{
      long time = framenumber * GetTicksPerFrame() / SampleRate * (1+Sample);
      intPtr->SetLightTint(time,_p3OldTint);
      intPtr->SetLightLevel(time,_fOldLightLevel);
      intPtr->SetAmbient(time,_p3OldAmbient);
      intPtr->SetUseEnvironmentMap(_BUseEnvMap);
      intPtr->SetBackGround(time,_p3OldBackgrd);
      intPtr->SetRendForce2Side(_BForce2Side);

      if (OldCamera)
            ViewPort->SetViewCamera(OldCamera);
      else
      {
            ViewPort->SetAffineTM(VptTM);
            ViewPort->SetViewUser(bPerspView);
      }
      intPtr->ReleaseViewport(ViewPort);
}

Compressor::TransmitTargetKeysThisSecond(GeoNode *node, int frame, int
&LastPKeyNum)
{
      Control *TarPosCon, *TarRolCon;
      TarPosCon                                                    =
node->curNode->GetTarget()->GetTMController()->GetPositionController();
      TarRolCon                                                    =
node->curNode->GetTarget()->GetTMController()->GetRollController();
      IKeyControl *tarposkeys = NULL;
      IKeyControl *tarrolkeys = NULL;
      int trcount = 0, tpcount = 0;

      if (TarPosCon)
            tarposkeys = GetKeyControlInterface(TarPosCon);
      if (TarRolCon)
            tarrolkeys = GetKeyControlInterface(TarRolCon);

      ITCBPoint3Key     tcbPosKey;
      IBezPoint3Key     bezPosKey;
      ILinPoint3Key     linPosKey;
```

```
	ITCBFloatKey	tcbFloatKey;
	IBezFloatKey	bezFloatKey;
	ILinFloatKey	linFloatKey;

	Point3	pvalue;

	BOOL bIsLight = FALSE;
	int  iLightNum = -1;
	for (int count=0;count<iLightCount;count++)
		if	((GeometryList[LightIndexList[count]]).curNode	==
node->curNode)
		{
			bIsLight = TRUE;
			iLightNum = count;
			break;
		}

	BOOL bIsCamera = FALSE;
	if (&(GeometryList[CameraIndexList[nSelectedCamera]]) == node)
		bIsCamera = TRUE;

	// now, what we do is we go thru all the keys, and let's say we have n
keys
	// that fall within the given time.  We add these N keys to the object,
along
	// with the N+1'th key as well, so we have a final point to interpolate
to.
	// so basically the scheme is to always send the keys that reach to the
end
	// of the current time sequence PLUS the next key as well.  As keys are
sent,
	// mark them as such.

	// note, it may be possible to use a built in hidden array called
	// 'keys' as part of the controller, to access them.  Consider this.
	// its easier than jumping thru them.

	if (tarposkeys)
	{
		int numtarposkeys = 0;
		if (tarposkeys)
			numtarposkeys = tarposkeys->GetNumKeys();

		int LastKeyToSend;
		for (int j = LastPKeyNum; j < numtarposkeys &&
			TarPosCon->GetKeyTime(j)	<
(frame+GetFrameRate())*GetTicksPerFrame(); j++);
		if (j < numtarposkeys) j++;
		LastKeyToSend = j;

		for (int i = LastPKeyNum; i < LastKeyToSend; i++)
		{
			int ftime = TarPosCon->GetKeyTime(i);
			unsigned int uiTime = (unsigned int) ftime;
			unsigned char * bTime = (unsigned char*) &uiTime;
			if	(TarPosCon->ClassID()	==
Class_ID(TCBINTERP_POSITION_CLASS_ID, 0))
```

```
                {
                        if (bIsLight)
                        {
                                iOutToFile(OPCODE_TARGET_KEYFRAME_TCBPOS);
                                iOutToFile(node->RefNumber+1);
                        }
                        else if (bIsCamera)
                        {
                                iOutToFile(OPCODE_TARGET_KEYFRAME_TCBPOS);
                                iOutToFile(0);
                        }
                        else
                        {
                                // its an object, not one of those
                        }
                        tarposkeys->GetKey(i, &tcbPosKey);
                        // transmit a 32b time identifier
                        cOutToFile(bTime[0]);
                        cOutToFile(bTime[1]);
                        cOutToFile(bTime[2]);
                        cOutToFile(bTime[3]);
                        // transmit the position
                        fOutToFile(tcbPosKey.val.x);
                        fOutToFile(tcbPosKey.val.y);
                        fOutToFile(tcbPosKey.val.z);
                        // now transmit the tension, continuity, and bias
settings but
                        // first rescale them from the -1 to 1 range so they
fit in
                        // 16b each
                        wOutToFile(ReScaleFloat(tcbPosKey.tens,-
1.0f,1.0f,BPC_16));
                        wOutToFile(ReScaleFloat(tcbPosKey.cont,-
1.0f,1.0f,BPC_16));
                        wOutToFile(ReScaleFloat(tcbPosKey.bias,-
1.0f,1.0f,BPC_16));
                        wOutToFile(ReScaleFloat(tcbPosKey.easeIn,-
1.0f,1.0f,BPC_16));
                        wOutToFile(ReScaleFloat(tcbPosKey.easeOut,-
1.0f,1.0f,BPC_16));
                }
                else if (TarPosCon->ClassID() ==
Class_ID(LININTERP_POSITION_CLASS_ID, 0))
                {
                        tarposkeys->GetKey(i, &linPosKey);
//                      &linPosKey = newkey;
                        if (bIsLight)
                        {
                                iOutToFile(OPCODE_TARGET_KEYFRAME_LINPOS);
                                iOutToFile(node->RefNumber+1);
                        }
                        else if (bIsCamera)
                        {
                                iOutToFile(OPCODE_TARGET_KEYFRAME_LINPOS);
                                iOutToFile(0);
                        }
                        else
                        {
```

```
                        // its an object, not one of those
                    }
                    // transmit a 32b time identifier
                    cOutToFile(bTime[0]);
                    cOutToFile(bTime[1]);
                    cOutToFile(bTime[2]);
                    cOutToFile(bTime[3]);
                    // transmit the position
                    fOutToFile(linPosKey.val.x);
                    fOutToFile(linPosKey.val.y);
                    fOutToFile(linPosKey.val.z);
                }
                else if (TarPosCon->ClassID() ==
Class_ID(HYBRIDINTERP_POSITION_CLASS_ID, 0))
                {
                    tarposkeys->GetKey(i, &bezPosKey);
                        int cTime,pTime,nTime;
                        cTime = TarPosCon->GetKeyTime(i);

                        if (i==0)
                            pTime = cTime;

                        else pTime = TarPosCon->GetKeyTime(i-1);

                        if (i==tarposkeys->GetNumKeys()-1)
                            nTime = cTime;

                        else nTime = TarPosCon->GetKeyTime(i+1);

                        float pScaleFactor = -float(cTime - pTime);
                        float nScaleFactor = float(nTime - cTime);
//                  &bezPosKey = newkey;
                    if (bIsLight)
                    {
                        iOutToFile(OPCODE_TARGET_KEYFRAME_BEZPOS);
                        iOutToFile(node->RefNumber+1);
                    }
                    else if (bIsCamera)
                    {
                        iOutToFile(OPCODE_TARGET_KEYFRAME_BEZPOS);
                        iOutToFile(0);
                    }
                    else
                    {
                        // its an object, not one of those
                    }
                    // transmit a 32b time identifier
                    cOutToFile(bTime[0]);
                    cOutToFile(bTime[1]);
                    cOutToFile(bTime[2]);
                    cOutToFile(bTime[3]);
                    // transmit the position
                    fOutToFile(bezPosKey.val.x);
                    fOutToFile(bezPosKey.val.y);
                    fOutToFile(bezPosKey.val.z);
                    // transmit the tangent vectors
                    fOutToFile(bezPosKey.intan.x * pScaleFactor );
                    fOutToFile(bezPosKey.intan.y * pScaleFactor );
```

```
                        fOutToFile(bezPosKey.intan.z * pScaleFactor );
                        fOutToFile(bezPosKey.outtan.x * nScaleFactor );
                        fOutToFile(bezPosKey.outtan.y * nScaleFactor );
                        fOutToFile(bezPosKey.outtan.z * nScaleFactor );
                    }
                }
                LastPKeyNum = LastKeyToSend;
            }
        }

void Compressor::CreateNewObject(GeoNode *node)
{
        Object* tempObj;
        tempObj = node->curNode->GetObjectRef()->Eval(0).obj;
        TriObject* tempTri = (TriObject*) tempObj->ConvertToType(0,
                    Class_ID(TRIOBJ_CLASS_ID, 0));
        BOOL bObjectSentThisFrame = FALSE;
        Control *PosCon, *RotCon, *SclCon;
        PosCon = node->curNode->GetTMController()->GetPositionController();
        RotCon = node->curNode->GetTMController()->GetRotationController();
        SclCon = node->curNode->GetTMController()->GetScaleController();
        IKeyControl *poskeys = NULL;
        IKeyControl *rotkeys = NULL;
        IKeyControl *sclkeys = NULL;
        int rcount = 0;
        int pcount = 0;
        int scount = 0;
        if (PosCon)
        {
                poskeys = GetKeyControlInterface(PosCon);
                if (poskeys)
                        pcount = poskeys->GetNumKeys();
        }
        if (RotCon)
        {
                rotkeys = GetKeyControlInterface(RotCon);
                if (rotkeys)
                        rcount = rotkeys->GetNumKeys();
        }
        if (SclCon)
        {
                sclkeys = GetKeyControlInterface(SclCon);
                if (sclkeys)
                        scount = sclkeys->GetNumKeys();
        }

        node->RefNumber = ++nLastRefNum;
        iOutToFile(OPCODE_ADDOBJECT);
        iOutToFile(node->nAddedVertCount);
        iOutToFile(node->nAddedFaceCount);
        if (iBitsPerVertex != BPC_32)
        {
                // we must now find a bounding box for all the vertices in
                // this node.  Note the bounding box is specified using the
                // added coordinates rather than the true object's bounding box
                // to enable maximal quantizing detail based on what is actually
                // exported in the scene.  While the bounding box can end up
                // anywhere, the origin is ALWAYS the pivot point.
```

```
        for (int i=0;i<node->nAddedVertCount;i++)
        {
                float tempx = tempTri->mesh.verts[node->AddedVertList[i]].x;
                float tempy = tempTri->mesh.verts[node->AddedVertList[i]].y;
                float tempz = tempTri->mesh.verts[node->AddedVertList[i]].z;
                if (tempx < node->pMinCoord.x) node->pMinCoord.x = tempx;
                if (tempx > node->pMaxCoord.x) node->pMaxCoord.x = tempx;
                if (tempy < node->pMinCoord.y) node->pMinCoord.y = tempy;
                if (tempy > node->pMaxCoord.y) node->pMaxCoord.y = tempy;
                if (tempz < node->pMinCoord.z) node->pMinCoord.z = tempz;
                if (tempz > node->pMaxCoord.z) node->pMaxCoord.z = tempz;
        }
        // transmit coordinates for the bounding box
        fOutToFile(node->pMinCoord.x);
        fOutToFile(node->pMinCoord.y);
        fOutToFile(node->pMinCoord.z);
        fOutToFile(node->pMaxCoord.x);
        fOutToFile(node->pMaxCoord.y);
        fOutToFile(node->pMaxCoord.z);
}

// if the object has no rotation keyframes, yet the object's rotation
// is not its default position, we force an orientation here
Matrix3 tm = GetLocalMatrix(node->curNode,0);
if (rcount == 0)
{
        Quat Rot(tm);
        if (fabs(Rot.x) > 0.005f || fabs(Rot.y) > 0.005f ||
                fabs(Rot.z) > 0.005f || fabs(Rot.w - 1.0f) > 0.005f)
        {
                // process parents
                // ignor parents for now...
                iOutToFile(OPCODE_SETROTATION);
                iOutToFile(node->RefNumber);
                Quat iRot(Inverse(Rot));
                quatOutToFile(iRot);
        }
}

// OR
// if the object has no position keyframe, yet the object's position
// is not its default position, we force a position here
if (pcount == 0)
{
        // ignor parents for now...
        Point3 Pos = tm.GetTrans();
        if (fabs(Pos.x) > 0.005f || fabs(Pos.y) > 0.005f ||
                fabs(Pos.z) > 0.005f)
        {
                iOutToFile(OPCODE_SETPOSITION);
                iOutToFile(node->RefNumber);
                fOutToFile(Pos.x);
                fOutToFile(Pos.y);
                fOutToFile(Pos.z);
        }
}
if (scount == 0)
```

```
	{
		// ignor parents for now...
		AffineParts aff;
		decomp_affine(tm,&aff);
		// only transmit the scale if its not (1.0,1.0,1.0)
		if (fabs(aff.k.x - 1.0f) > 0.005f || fabs(aff.k.y - 1.0f) > 0.005f
||
			fabs(aff.k.z - 1.0f) > 0.005f)
		{
			iOutToFile(OPCODE_SETSCALE);
			iOutToFile(node->RefNumber);
			fOutToFile(aff.k.x);
			fOutToFile(aff.k.y);
			fOutToFile(aff.k.z);
		}
	}

	// now determine the offset for the object
	Point3		offsetPos = node->curNode->GetObjOffsetPos();
	Quat		offsetRot = node->curNode->GetObjOffsetRot();
	ScaleValue	offsetScl = node->curNode->GetObjOffsetScale();
	Matrix3 OffsetTM(1),Scl(1),Tran(1);
	ApplyScaling(Scl, offsetScl);
	Matrix3 Rot;
	offsetRot.MakeMatrix(Rot);
	Tran.Translate(offsetPos);
	OffsetTM = Scl * Rot * Tran;

	if (!CheckIdentity(OffsetTM))
	{
		iOutToFile(OPCODE_SETOFFSET);
		iOutToFile(node->RefNumber);
		for (int l1=0;l1<4;l1++)
			for (int l2=0;l2<3;l2++)
				fOutToFile(OffsetTM.GetRow(l1)[l2]);
	}

	// This code allows for the recurrsive specification of parent/child
	// heirarchies in file structures.
	if (node->IsChild)
	{
		if (GeometryList[node->ParentIdx].RefNumber < 0)
			CreateNewObject(&(GeometryList[node->ParentIdx]));
		iOutToFile(OPCODE_SETPARENTCHILD);
		iOutToFile(node->RefNumber);
		iOutToFile(GeometryList[node->ParentIdx].RefNumber);
	}
}


Matrix3 Compressor::GetLocalMatrix(INode *node, TimeValue t)
{
	Matrix3 ntm, ptm, rtm(1), piv(1), tm;

	// Get Parent and Node TMs
	ntm = node->GetNodeTM(t);
	ptm = node->GetParentTM(t);
```

```
      // Compute the relative TM
      ntm = ntm * Inverse(ptm);

      return (ntm);
}

float Bilinear(float U,float V,unsigned char A,unsigned char B,unsigned char
C,unsigned char D)
{
       float r = (1.0f - U)*(1.0f - V)*(float)A +
             U*(1.0f - V)*(float)B + (1.0f - U)*V*(float)C +
             U*V*(float)D;
       return (r);
}

Point3  Compressor::RetrieveTexelColor(float  U,  float  V,  unsigned  char
*Texture)
{
      // given floating point X/Y coordinates into a texture, retrieves
      // the corresponding 24b color

      if (U<0.0f) U=fabsf(U);
      if (V<0.0f) V=fabsf(V);
      if (U>1.0f) U=1.0f;
      if (V>1.0f) V=1.0f;

      float xindex = U*(float)(MAXTEXTURERES-1),
                  yindex = V*(float)(MAXTEXTURERES-1);
      int x0 = (int)floor(xindex);
      int x1 = (int)ceil(xindex);
      int y0 = (int)floor(yindex);
      int y1 = (int)ceil(yindex);

      float fU = xindex - x0;
      float fV = yindex - y0;

      Point3 color;
      color.x                                                                =
Bilinear(fU,fV,Texture[3*(x0+y0*MAXTEXTURERES)],Texture[3*(x1+y0*MAXTEXTURERES
)],
            Texture[3*(x0+y1*MAXTEXTURERES)],Texture[3*(x1+y1*MAXTEXTURERES)])
;
      color.y                                                                =
Bilinear(fU,fV,Texture[3*(x0+y0*MAXTEXTURERES)+1],Texture[3*(x1+y0*MAXTEXTURER
ES)+1],
            Texture[3*(x0+y1*MAXTEXTURERES)+1],Texture[3*(x1+y1*MAXTEXTURERES)
+1]);
      color.z                                                                =
Bilinear(fU,fV,Texture[3*(x0+y0*MAXTEXTURERES)+2],Texture[3*(x1+y0*MAXTEXTURER
ES)+2],
            Texture[3*(x0+y1*MAXTEXTURERES)+2],Texture[3*(x1+y1*MAXTEXTURERES)
+2]);
      return color;
}

Compressor::BuildWaveletCoeffs(MatNode *node)
{
```

```
      unsigned char originalmap[MAXTEXTURERES*MAXTEXTURERES*3];
      RetrieveOriginalMap(node,originalmap);

      node->WaveletCoeff = new float[MAXTEXTURERES*MAXTEXTURERES*3];
      // first, let's copy the original map into the coefficient space
      // making sure to seperate out the 3 color spaces
      int sqr = MAXTEXTURERES*MAXTEXTURERES;
      for (int z=0;z<3;z++)
            for (int i=0;i<sqr;i++)
                  node->WaveletCoeff[z*sqr+i] = (float) originalmap[i*3+z];

//    FileDump("_or",originalmap,MAXTEXTURERES,3);
//    FileFloatDump("_fl",node->WaveletCoeff,MAXTEXTURERES,3);

      // now let's perform the wavelet transform on the original image
      // and then on the consecutive "approximation"s.
      float DataArray[MAXTEXTURERES];
      for (int scale=MAXTEXTURERES;scale>MINTEXTURERES;scale/=2)
            for                                                          (int
color=0;color<MAXTEXTURERES*MAXTEXTURERES*3;color+=MAXTEXTURERES*MAXTEXTURERES
)
            {
                  for (int i=0;i<scale;i++)
                  {
                        for (int j=0;j<scale;j++)
                              DataArray[j]                                =
node->WaveletCoeff[color+i*MAXTEXTURERES+j];
                        HaarTR(node->WaveletCoeff+color+i*MAXTEXTURERES,DataAr
ray,scale);
                  }
                  for (i=0;i<scale;i++)
                  {
                        for (int j=0;j<scale;j++)
                              DataArray[j]                                =
node->WaveletCoeff[color+i+j*MAXTEXTURERES];
                        float TempArray[MAXTEXTURERES];
                        HaarTR(TempArray,DataArray,scale);
                        for (j=0;j<scale;j++)
                              node->WaveletCoeff[color+i+j*MAXTEXTURERES]    =
TempArray[j];
                  }
            }

      // now threshold the coefficients to drop out low detail information
      DetermineThreshold(node->WaveletCoeff,originalmap,30.0f);
//    FileFloatDump("_tr",node->WaveletCoeff,MAXTEXTURERES,3);
}

Compressor::RetrieveOriginalMap(MatNode *node, unsigned char *map)
{
      // So, what we do is first, find out what the original texturemap was
      // for this object.  Right now, the easiest thing to do is just
      // support Diffuse Maps.  So for the immediate future, that's what I'm
      // supporting.
      // Eventually this function *should* take all the compound textures
      // that this object has, and render/blend them together into one
      // compounded texture.
```

```
	// In any case, construct a texture map, and return it to the 'map'
	// structure as a 24b color bitmap that *always* has the dimensions
	// 256x256.

	// Note that this technique _only_ works on textures that can be mapped
	// using UVW1 coordinates.  Textures that must rely on XYZ coordinates
	// will appear as they would using UVW1 coordinates - in other words,
	// they probably won't appear quite as desired.  If such textures are
	// necessary it is recommended that the RenderMap function in the
3DSMax
	// material editor be used to create a bitmap which would then be
mapped
	// to the object.

	Mtl* nodeMat = node->curNode;

	TriObject* obj = CreateNewTriObject();
	int currvertsum = 0;
	int currfacesum = 0;
	MNMesh builder;
	builder.SetFlag(MN_MESH_CVERTS, TRUE);
	Point3 vert;
	vert.x = -1.0f;
	vert.y = -1.0f;
	vert.z = -5.0f;
	builder.NewVert(vert);
	vert.x =  1.0f;
	vert.y = -1.0f;
	builder.NewVert(vert);
	vert.x = -1.0f;
	vert.y =  1.0f;
	builder.NewVert(vert);
	vert.x =  1.0f;
	vert.y =  1.0f;
	builder.NewVert(vert);
	UVVert uva(0.0f,0.0f,0.0f);
	UVVert uvb(0.0f,1.0f,0.0f);
	UVVert uvc(1.0f,0.0f,0.0f);
	UVVert uvd(1.0f,1.0f,0.0f);
	builder.NewTVert(uva,0,0);
	builder.NewTVert(uvb,0,0);
	builder.NewTVert(uvc,0,0);
	builder.NewTVert(uvd,0,0);
	VertColor c;
	c.x = 1.0f;
	c.y = 1.0f;
	c.z = 1.0f;
	builder.NewCVert(c);
	builder.NewCVert(c);
	builder.NewCVert(c);
	builder.NewCVert(c);
	int vv0[3] = {2,0,1};
	int vv1[3] = {1,3,2};
	builder.NewTri(vv0,NULL,vv0);
	builder.NewTri(vv1,NULL,vv1);
	builder.OutToTri(obj->mesh);
	Matrix3 I(1);
	obj->ApplyUVWMap(MAP_PLANAR,1,1,1,0,0,0,0,I,0);
```

```
        INode*  meshnode = 0;
        meshnode = intPtr->CreateObjectNode(obj);
        meshnode->SetName(_T("Texture Rendering thing"));
        meshnode->SetWireColor(RGB(255,255,255));

        Color black(0.0,0.0,0.0);
        Color white(1.0,1.0,1.0);
        Color grey (0.5,0.5,0.5);
        Color red (1.0,0.0,0.0);
        Color blue (0.0,0.0,1.0);

        meshnode->SetMtl((StdMat*)node->curNode);
        meshnode->SetMotBlur(0);
        meshnode->SetRcvShadows(FALSE);
        meshnode->SetCastShadows(FALSE);
        meshnode->SetRenderable(TRUE);
//      meshnode->SetCastShadows(FALSE);
        meshnode->SetCVertMode(FALSE);
        meshnode->SetShadeCVerts(TRUE);

        ViewParams vp;
        vp.projType = PROJ_PARALLEL;
//      vp.zoom = 1.0f/128.0f;
        vp.zoom = 1.0f/200.0f;
        vp.hither = 1.0f;
        vp.yon = 1500.0f;
        Matrix3 M(1);
        Point3 pos(0.0f,0.0f,-1.0f);
        M.SetRow(2,pos);
        vp.affineTM = M;

        RendParams rp;
//      rp.inMtlEdit = TRUE;
        FrameRendParams frp;
        frp.background = blue;
        frp.ambient = grey;
        frp.globalLightLevel = white;

        MNMesh dummybuilder;

        INode* tempRoot = 0;
        TriObject* dummyobj = CreateNewTriObject();
        dummybuilder.OutToTri(dummyobj->mesh);
        tempRoot = intPtr->CreateObjectNode(dummyobj);
        tempRoot->AttachChild(meshnode,1);

        GenLight *objLight = (GenLight*)intPtr->CreateInstance(
              LIGHT_CLASS_ID,Class_ID(DIR_LIGHT_CLASS_ID,0));

        objLight->NewLight(DIR_LIGHT);
        objLight->SetUseLight(TRUE);
        objLight->Enable(TRUE);
        Point3 LCol(1.0f,1.0f,1.0f);
        objLight->SetRGBColor(0,LCol);
        objLight->SetIntensity(0,1.0f);

        pos.z = 1.0f;
        M.SetRow(2,pos);
        INode* tempLight = intPtr->CreateObjectNode(objLight);
```

```
	tempLight->SetNodeTM(0,M);
	tempRoot->AttachChild(tempLight,1);

	Bitmap* bmap = 0;
	BitmapInfo bi;
	bi.SetType(BMM_TRUE_32);
	bi.SetWidth(MAXTEXTURERES);
	bi.SetHeight(MAXTEXTURERES);
	bmap = TheManager->Create(&bi);
	bmap->OpenOutput(&bi);
	bmap->SetFilter(BMM_FILTER_NONE);
	bmap->SetDither(BMM_DITHER_NONE);

	IScanRenderer* rend = (IScanRenderer*) intPtr->GetDraftRenderer();
	int success = rend->Open(tempRoot,0,&vp,rp,intPtr->GetMAXHWnd());
	success = rend->Render(0,bmap,frp,intPtr->GetMAXHWnd());
	rend->Close(intPtr->GetMAXHWnd());

	meshnode->Delete(0,0);
	tempLight->Delete(0,0);
	tempRoot->Delete(0,0);

	BMM_Color_64 black64= {0,0,0,0};
	Bitmap* readmap = 0;
	readmap = TheManager->Create(&bi);
	readmap->CopyImage(bmap,COPY_IMAGE_RESIZE_LO_QUALITY,black64);

	bmap->Close(&bi);
	if (bmap) bmap->DeleteThis();

	int BMtype = BMM_TRUE_24;
	unsigned char* bPtr = (unsigned char*)
readmap->Storage()->GetStoragePtr(&BMtype);

	unsigned char* bitmapwalker = (unsigned char*) bPtr;
	for (int j=0;j<MAXTEXTURERES;j++)
		for (int i=0;i<MAXTEXTURERES;i++)
		{
			map[(i+j*MAXTEXTURERES)*3+0] = *(bitmapwalker++);
			map[(i+j*MAXTEXTURERES)*3+1] = *(bitmapwalker++);
			map[(i+j*MAXTEXTURERES)*3+2] = *(bitmapwalker++);
		}

//	FileDump("Diagnostic", map, MAXTEXTURERES, 3);
}

Compressor::Daub4TR(float a[], float source[], unsigned int size)
{
	const float C0 =  0.4829629131445341f;
	const float C1 =  0.8365163037378079f;
	const float C2 =  0.2241438680420134f;
	const float C3 = -0.1294095225512604f;
	if (size<4) return 0;
	const unsigned int halfsize = size >> 1;
	for (unsigned int i=0,j=0;j<=size-4;j+=2,i++)
	{
		a[i               ] = C0*source[j] + C1*source[j+1] +
			C2*source[j+2] + C3*source[j+3];
```

```
            a[i+halfsize] = C3*source[j] - C2*source[j+1] +
                  C1*source[j+2] - C0*source[j+3];
      }
      a[i                ] = C0*source[size-2] + C1*source[size-1] +
            C2*source[0] + C3*source[1];
      a[i+halfsize] = C3*source[size-2] - C2*source[size-1] +
            C1*source[0] - C0*source[1];
      return 0;
}

void Compressor::HaarTR(float a[], float source[], unsigned int size)
{
      const unsigned int halfsize = size >> 1;

      for (unsigned int i=0,j=0;j<size;j+=2,i++)
      {
            a[i                ] = 0.5f*(source[j] + source[j+1]);
            a[i+halfsize] = source[j] - source[j+1];
      }
}

void Compressor::HaarInv(float a[], float source[], unsigned int size)
{
      const unsigned int halfsize = size >> 1;

      for (unsigned int i=0,j=0;i<halfsize;i++)
      {
            a[j++] = source[i] + source[i+halfsize]*0.5f;
            a[j++] = source[i] - source[i+halfsize]*0.5f;
      }
}

void Compressor::DetermineThreshold(float *coeffs, unsigned char* map, float
targetPSNR)
{
      // okay, here we take this texture, first back it up.
      // Once we've backed it up, we begin an iterative process
      // of hard thresholding the coefficients.

      float duplicate[MAXTEXTURERES*MAXTEXTURERES*3];
      float PSNR = 53.0f;
      BOOL bChanges = TRUE;

      // so long as the PSNR
      float oldPSNR = 0.0f;
      float threshold = PSNR - targetPSNR;
      float step = 1.0f;
      while (fabsf(PSNR - targetPSNR) > 1.5f && fabsf(PSNR-oldPSNR) > 0.05 &&
step > 0.001)
      {
            if (fabsf(PSNR-oldPSNR) < 0.05)
                  step/=10.0f;

            if (PSNR > targetPSNR)
                  threshold+=step;
            else
                  threshold-=step;
```

```
            oldPSNR = PSNR;

            for (int i=0;i<MAXTEXTURERES*MAXTEXTURERES*3;i++)
                  duplicate[i] = coeffs[i];
//FileDump("_original", map, MAXTEXTURERES, 3);
//FileFloatDump("_prethreshold", duplicate, MAXTEXTURERES, 3);

            // first, perform a thresholding
            for                                                    (int
color=0;color<MAXTEXTURERES*MAXTEXTURERES*3;color+=MAXTEXTURERES*MAXTEXTURERES
)
              for (int i=0;i<MAXTEXTURERES;i++)
                  for (int j=0;j<MAXTEXTURERES;j++)
                     if  (fabsf(duplicate[color+MAXTEXTURERES*i+j])<=threshold
&&
                           !(i<MINTEXTURERES && j<MINTEXTURERES))
                        duplicate[color+MAXTEXTURERES*i+j]=0.0f;

//FileFloatDump("_thresholded", duplicate, MAXTEXTURERES, 3);

            // second, perform the inverse wavelet transform
            float DataArray[MAXTEXTURERES];
            for (int scale=MINTEXTURERES*2;scale<=MAXTEXTURERES;scale*=2)
            {
                  for                                              (int
color=0;color<MAXTEXTURERES*MAXTEXTURERES*3;color+=MAXTEXTURERES*MAXTEXTURERES
)
                  {
                        for (int i=0;i<scale;i++)
                        {
                              for (int j=0;j<scale;j++)
                                    DataArray[j]                          =
duplicate[color+i*MAXTEXTURERES+j];
                              float TempArray[MAXTEXTURERES];
                              HaarInv(TempArray,DataArray,scale);
                              for (j=0;j<scale;j++)
                                    duplicate[color+i*MAXTEXTURERES+j]    =
TempArray[j];
                        }
                        for (i=0;i<scale;i++)
                        {
                              for (int j=0;j<scale;j++)
                                    DataArray[j]                          =
duplicate[color+i+j*MAXTEXTURERES];
                              float TempArray[MAXTEXTURERES];
                              HaarInv(TempArray,DataArray,scale);
                              for (j=0;j<scale;j++)
                                    duplicate[color+i+j*MAXTEXTURERES]    =
TempArray[j];
                        }
                  }
//FileFloatDump("_reconstructed", duplicate, MAXTEXTURERES, 3);
            }

            // third, let's calculate the new PSNR for the thresholded
            // version of the coefficients.
            float mse = 0.0f, origpeak = -1e38f;
```

```
            int sqr = MAXTEXTURERES*MAXTEXTURERES;
            for (color=0;color<3;color++)
              for (int i=0;i<MAXTEXTURERES;i++)
                  for (int j=0;j<MAXTEXTURERES;j++)
            {
                  mse+=       ((float)map[color+3*(i+j*MAXTEXTURERES)]       -
duplicate[color*sqr+i+j*MAXTEXTURERES]) *
                              ((float)map[color+3*(i+j*MAXTEXTURERES)]    -
duplicate[color*sqr+i+j*MAXTEXTURERES]);
                  if ( (float)map[color+3*(i+j*MAXTEXTURERES)] > origpeak)
                        origpeak = (float)map[color+3*(i+j*MAXTEXTURERES)];
            }
            mse/=255.0;
            PSNR = 10.0f*log10f(origpeak*origpeak/mse);
      }

      // now that we've hit our target, we replace our original texmap
      // coefficients with our thresholded version
      for                                                                (int
color=0;color<MAXTEXTURERES*MAXTEXTURERES*3;color+=MAXTEXTURERES*MAXTEXTURERES
)
        for (int i=0;i<MAXTEXTURERES;i++)
            for (int j=0;j<MAXTEXTURERES;j++)
              if (fabsf(duplicate[color+MAXTEXTURERES*i+j])<=threshold &&
                    !(i<MINTEXTURERES && j<MINTEXTURERES))
                  coeffs[color+MAXTEXTURERES*i+j]=0.0f;
}


Compressor::DeltaEncode(float *coeff, int size, int dim)
{
      // following the peano path, this function encodes
      // the differences between consecutive coefficient values
      unsigned int sqr = dim*dim;
      CPixelPeano peano(size,TRUE);
      for                                                                (int
z=0;z<MAXTEXTURERES*MAXTEXTURERES*3;z+=MAXTEXTURERES*MAXTEXTURERES)
      {
            float       oldval       =       coeff[z+(peano.coordx       +
MAXTEXTURERES*peano.coordy)];
            float newval;
            for (unsigned int i=1;i<sqr && !peano.Empty();i++)
            {
                  while(peano.Process());
                  newval          =          coeff[z+(peano.coordx          +
MAXTEXTURERES*peano.coordy)];
                  coeff[z+(peano.coordx    +    MAXTEXTURERES*peano.coordy)]-=
oldval;
                  oldval = newval;
            }
            peano.Reset();
      }
}

Compressor::DetermineNecessaryTexRes(MatNode* node)
{
      // this routine determines what maximum necessary texture resolution
```

```
      // we actually require.  After all, if they claim they want a 256x256
      // texture but we know that the loss of detail between 128x128 is below
      // the given texture quality threshold, then we don't really need the
      // 256x256 texture anyway. So we apply this to all of our texture
      // resolutions
      unsigned char orig[MAXTEXTURERES*MAXTEXTURERES*3];

      unsigned char recon[MAXTEXTURERES*MAXTEXTURERES*3];
      for (int i=0;i<MAXTEXTURERES*MAXTEXTURERES*3;i++) recon[i]=0;
      float PSNR[MAXTEXTUREPOWER];
      for (i=MINTEXTUREPOWER+1;i<MAXTEXTUREPOWER;i++) PSNR[i]=0.0f;
      PSNR[MINTEXTUREPOWER] = 1e10f;

      // first let's copy in the minimal wavelet approxiamation into the
"orig" array
      for                                                              (int
color=0;color<MAXTEXTURERES*MAXTEXTURERES*3;color+=MAXTEXTURERES*MAXTEXTURERES
)
            for (int i=0;i<MINTEXTURERES;i++)
                  for (int j=0;j<MINTEXTURERES;j++)
                        recon[color+j*MAXTEXTURERES+i] = (unsigned char)
                              node->WaveletCoeff[color+j*MAXTEXTURERES+i];

      // now we do a progressive comparison of consecutive
      for (int scale=MINTEXTUREPOWER+1;scale<MAXTEXTUREPOWER;scale++)
      {
            // first, copy the previous work into our reconstruction
scratchpad
            for (unsigned int i=0;i<MAXTEXTURERES*MAXTEXTURERES*3;i++)
                  orig[i] = recon[i];
            unsigned int size = 1;
            for (int j=0;j<scale;j++) size*=2;
            // now reconstruct
            for                                                        (int
color=0;color<MAXTEXTURERES*MAXTEXTURERES*3;color+=MAXTEXTURERES*MAXTEXTURERES
)
            {
                  float DataArray[MAXTEXTURERES];
                  for (i=0;i<size;i++)
                  {
                        for (unsigned int j=0;j<size;j++)
                              DataArray[j] = recon[color+i+j*MAXTEXTURERES];
                        float TempArray[MAXTEXTURERES];
                        HaarInv(TempArray,DataArray,size);
                        for (j=0;j<size;j++)
                              recon[color+i+j*MAXTEXTURERES] = (unsigned char)
TempArray[j];
                  }
                  for (i=0;i<size;i++)
                  {
                        for (unsigned int j=0;j<size;j++)
                              DataArray[j] = recon[color+i*MAXTEXTURERES+j];
                        float TempArray[MAXTEXTURERES];
                        HaarInv(TempArray,DataArray,size);
                        for (j=0;j<size;j++)
                              recon[color+i*MAXTEXTURERES+j]     =     (unsigned
```

```
char)TempArray[j];
                  }
            }
            // so now we have 'orig', the previous version, and 'recon', the
new version.
            // We need to take 'orig', and filter it to twice the size into a
new array
            // called twice.  A simple standard Bilinear filter is used.
            // (see Numerical Recipes in C page 123)
            unsigned char twice[MAXTEXTURERES*MAXTEXTURERES*3];

            for
(color=0;color<MAXTEXTURERES*MAXTEXTURERES*3;color+=MAXTEXTURERES*MAXTEXTURERE
S)
                  for (unsigned int i=0;i<size;i+=2)
                        for (unsigned int j=0;j<size;j+=2)
                  {
                        twice[i+j*MAXTEXTURERES+color]                        =
orig[i/2+j/2*MAXTEXTURERES+color];
                        twice[i+1+j*MAXTEXTURERES+color]                      =
(orig[i/2+j/2*MAXTEXTURERES+color] +
                              orig[(i+1)/2+j/2*MAXTEXTURERES+color])/2;
                        twice[i+(j+1)*MAXTEXTURERES+color]                    =
(orig[i/2+j/2*MAXTEXTURERES+color] +
                              orig[i/2+(j+1)/2*MAXTEXTURERES+color])/2;
                        twice[i+1+(j+1)*MAXTEXTURERES+color] =
                              (orig[i/2+j/2*MAXTEXTURERES+color] +
                              orig[i/2+(j+1)/2*MAXTEXTURERES+color] +
                              orig[(i+1)/2+j/2*MAXTEXTURERES+color] +
                              orig[(i+1)/2+(j+1)/2*MAXTEXTURERES+color])/4;
                  }

//FileDump("__orig", orig, MAXTEXTURERES, 3);
//FileDump("__recon", recon, MAXTEXTURERES, 3);
//FileDump("__twice", twice, MAXTEXTURERES, 3);

            // now calculate the PSNR
            float mse = 0.0f, origpeak = -1e38f;
            for
(color=0;color<MAXTEXTURERES*MAXTEXTURERES*3;color+=MAXTEXTURERES*MAXTEXTURERE
S)
                  for (unsigned int i=0;i<size;i++)
                        for (unsigned int j=0;j<size;j++)
            {
                  mse+=(float)(        (recon[color+i+j*MAXTEXTURERES]        -
twice[color+i+j*MAXTEXTURERES]) *
                        (recon[color+i+j*MAXTEXTURERES]                       -
twice[color+i+j*MAXTEXTURERES]));
                  if ( (float)recon[color+i+j*MAXTEXTURERES] > origpeak)
                        origpeak = (float)recon[color+i+j*MAXTEXTURERES];
            }

            if (mse == 0.0f)
                  PSNR[scale] = 1e10f;
            else
            {
                  mse/=254.0;
                  PSNR[scale] = 10.0f*log10f(origpeak*origpeak/mse);
```

```
            }
      }

      for              (i=MAXTEXTUREPOWER-1;i>MINTEXTUREPOWER                &&
(PSNR[i]>fTextureQualityThreshold);i--)
            node->iMaxNecessaryTexPower--;
}

Compressor::GetCoeffRange(float *coeffs, float &min, float &max)
// extract the minimum and maximum values from the coeff array
{
      float _min = 1e38f;
      float _max = -1e38f;
      float val;
      for (unsigned int j=0;j<MAXTEXTURERES*MAXTEXTURERES*3;j+=MAXTEXTURERES)
            for (unsigned int i=0;i<MAXTEXTURERES;i++)
            if (j>=MINTEXTURERES || i>=MINTEXTURERES)
            {
                  val = coeffs[j+i];
                  if (val>_max) _max = val;
                  if (val<_min) _min = val;
            }
      max = _max;
      min = _min;
}

Compressor::AddTextureCoeffs(MatNode *node)
{
      // adds a specific group of coefficients from the texture map's
      // wavelet coefficients to the outgoing datastream.
      // If this is the first group, we send the top left hand block.
      // Otherwise we transmit the L-shaped block composing the other
      // three.
      iOutToFile(OPCODE_ADDMIPMAP);
      if (node->RefNumber == -1)
            node->RefNumber = ++nLastMatRefNum;
      iOutToFile(node->RefNumber);
      unsigned int val = 1;
      for (int j=0;j<node->iLastTexturePowerSent;j++) val*=2;
      unsigned int sqr = val*val;
      CPixelPeano *peano;
      for              (unsigned              int              color              =
0;color<MAXTEXTURERES*MAXTEXTURERES*3;color+=MAXTEXTURERES*MAXTEXTURERES)
      {
            unsigned int pixcount = sqr;
            if (node->iLastTexturePowerSent == MINTEXTUREPOWER)
            {
                  peano = new CPixelPeano(node->iLastTexturePowerSent,FALSE);
                  cOutToFile(WaveletQuantize(0.0f,254.0f,
                        node->WaveletCoeff[color     +     peano->coordx     +
MAXTEXTURERES*peano->coordy]));
                  for (unsigned int i=1;i<pixcount && !peano->Empty();i++)
                  {
                        while(peano->Process());
      // Note that a very simplistic form of run length encoding is
      // used to allow for further compaction of the data stream.
      // since highly thresholded textures contain a large number of
```

```
      // zeros, instead of simply relying on the global compression
      // system to account for this, we directly take advantage of
      // this fact.  Whereas normally we would encode 8 bits per
      // wavelet coefficient, we instead encode where values are mapped
      // between 0x00 and 0xFE, with 0xFF being reserved.  0xFF
      // is a special symbol used to recognize that we wish to perform
      // a simple run length encoding of a string of 0.0f values,
      // with the next character sent representing the number of
      // zeros (up to 255 of course).
      // ***This is currently disabled

/*                    if
(WaveletQuantize(node->fTexmapMinVal,node->fTexmapMaxVal,
                              node->WaveletCoeff[color  +  peano->coordx
+ MAXTEXTURERES*peano->coordy]) ==
                        WaveletQuantize(node->fTexmapMinVal,node->fTexma
pMaxVal,0.0f))
                    {
                        // swap to a temporary peano stack so we can
scan for zeros
                        peano->Switch();
                        BOOL bZero = TRUE;
                        for (int count=0;count<256 && i+count<pixcount
&& bZero;count++)
                        {
                              while(peano->Process());
                              bZero                                    =
(WaveletQuantize(node->fTexmapMinVal,node->fTexmapMaxVal,
                                    node->WaveletCoeff[color          +
peano->coordx + MAXTEXTURERES*peano->coordy]) ==
                                    WaveletQuantize(node->fTexmapMinVal,
node->fTexmapMaxVal,0.0f));
                        }
                        // switch back to the real stack
                        peano->Switch();
                        if (count == 1 || count == 2)
                              cOutToFile(WaveletQuantize(node->fTexmapMi
nVal,node->fTexmapMaxVal,
                                    node->WaveletCoeff[color          +
peano->coordx + MAXTEXTURERES*peano->coordy]));
                        else
                        {
                              for (int loop=0;loop<count-1;loop++)
                                    while(peano->Process());
                              // send a special 0xFF code to indicate a
string of zeros
                              cOutToFile(255);
                              cOutToFile((unsigned char)count);
                        }
                    }
                    else */cOutToFile(WaveletQuantize(0.0f,254.0f,
                        node->WaveletCoeff[color   +   peano->coordx   +
MAXTEXTURERES*peano->coordy]));
                }
                }
                else
```

```
            {
                  pixcount-=(sqr>>2);
                  peano = new CPixelPeano(node->iLastTexturePowerSent,TRUE);
                  cOutToFile(WaveletQuantize(node->fTexmapMinVal,node->fTexmap
MaxVal,
                        node->WaveletCoeff[color     +     peano->coordx     +
MAXTEXTURERES*peano->coordy]));
                  for (unsigned int i=1;i<pixcount && !peano->Empty();i++)
                  {
                        while(peano->Process());
/*                      if
(WaveletQuantize(node->fTexmapMinVal,node->fTexmapMaxVal,
                                    node->WaveletCoeff[color  +  peano->coordx
+ MAXTEXTURERES*peano->coordy]) ==
                              WaveletQuantize(node->fTexmapMinVal,node->fTexma
pMaxVal,0.0f))
                        {
                              // swap to a temporary peano stack so we can
scan for zeros
                              peano->Switch();
                              BOOL bZero = TRUE;
                              for (int count=0;count<256 && i+count<pixcount
&& bZero;count++)
                              {
                                    while(peano->Process());
                                    bZero                                   =
(WaveletQuantize(node->fTexmapMinVal,node->fTexmapMaxVal,
                                          node->WaveletCoeff[color          +
peano->coordx + MAXTEXTURERES*peano->coordy]) ==
                                          WaveletQuantize(node->fTexmapMinVal,
node->fTexmapMaxVal,0.0f));
                              }
                              // switch back to the real stack
                              peano->Switch();
                              if (count == 1 || count == 2)
                                    cOutToFile(WaveletQuantize(node->fTexmapMi
nVal,node->fTexmapMaxVal,
                                          node->WaveletCoeff[color          +
peano->coordx + MAXTEXTURERES*peano->coordy]));
                              else
                              {
                                    for (int loop=0;loop<count-1;loop++)
                                          while(peano->Process());
                                    // send a special 0xFF code to indicate a
string of zeros
                                    cOutToFile(255);
                                    cOutToFile((unsigned char)count);
                              }
                        }
                        else
*/cOutToFile(WaveletQuantize(node->fTexmapMinVal,node->fTexmapMaxVal,
                              node->WaveletCoeff[color   +   peano->coordx   +
MAXTEXTURERES*peano->coordy]));
                  }
            }
```

```
            delete peano;
      }
      node->bMipSent[node->iLastTexturePowerSent-MINTEXTUREPOWER] = TRUE;
      node->iLastTexturePowerSent++;
}

unsigned char Compressor::WaveletQuantize(float min, float max, float coeff)
// map 'coeff' from the mix->max range into the range of 0 through 254
{
      float delta = 254.0f / (max - min);
      if (coeff > max) return ((unsigned char) max);
      if (coeff < min) return ((unsigned char) min);
      return ((unsigned char)round((coeff-min)*delta));
}

BOOL Compressor::IsMarkedForDeletion(GeoNode *node)
{
      TCHAR nodename[255];
      _tcscpy(nodename,node->curNode->GetName());
      TCHAR* pDest = _tcsstr(nodename,_T("**DELETEME!!**"));
      if (pDest)
            return TRUE;
      return FALSE;
}

void Compressor::LZSS_OutputCode(FILE *fo, unsigned int code,int bitlength,
                                                      int &OutputBitCount,unsigned
int &OutputBitBuffer)
{
//    static int OutputBitCount = 0;
//    static unsigned int OutputBitBuffer = 0;

      OutputBitBuffer |= (unsigned long) code << (32-bitlength-
OutputBitCount);
      OutputBitCount += bitlength;
      while(OutputBitCount >= 8)
      {
            fputc(OutputBitBuffer >> 24,fo);
            OutputBitBuffer <<= 8;
            OutputBitCount -= 8;
            iCompressedCount++;
      }
}

void Compressor::LZSS_Scan(unsigned char * Dictionary,unsigned char *
LookAhead,
            unsigned int &position,unsigned int &length)
{
      // A simple straight forward linear search into the dictionary.
      // Note that the special case of maxlength-1 for a position is
      // considered, by not allowing a maximum length string (this
      // code corresponds to our end-of-file code
      unsigned int maxlength = 0;
      unsigned int bestpos = 0;
      unsigned int bOverSized = 0;
      if (LZSSBITLEN+LZSSBITPOS > 16)
            bOverSized = 1;
```

```
      for (int i=0;i<LZSSWINDOW_LENGTH;i++)
            if (Dictionary[i] == LookAhead[0])
            {
                  unsigned int j=1;
                  while (Dictionary[(i+j) % LZSSWINDOW_LENGTH] == LookAhead[j]
&&
                        j<(LZSSMAXLEN) &&
                        !( i == (LZSSWINDOW_LENGTH-1) &&
                        j == LZSSMAXLEN-1) )
                    j++;
                  if (j>maxlength)
                  {
                        maxlength = j;
                        bestpos = i;
                  }
            }
      position = bestpos;
      length = maxlength;
}

BOOL Compressor::LZSS_OutputFile(int headersize, const TCHAR *filename)
// This performs LZSS compression using a cylic dictionary
// of size 16k, using 14 bit position and 5 bit length
{
      FILE* finput;
      FILE* foutput;
      // first, we open the file we just created
      if ((finput = fopen("____temp", "rb")) == NULL)
      {
            // messagebox saying unable to write to file
            MessageBox(NULL,"Unable to compress the file.  Check for available
drivespace on destination drive.",
                  "Error!",MB_OK);
            return FALSE;
      }
      if ((foutput = fopen(filename, "wb")) == NULL)
      {
            // messagebox saying unable to write to file
            MessageBox(NULL,"Unable to compress the file.  Check for available
drivespace on destination drive.",
                  "Error!",MB_OK);
            return FALSE;
      }

      // used to keep track of our progress.. first grab the length of the
input file
      long lunCompressedCount = 0;
      long lOriginalLength = 1;
      fseek(finput,0,SEEK_END);
      lOriginalLength = ftell(finput);
      rewind(finput);
      intPtr->ProgressStart(GetString(IDS_PROGRESS_MSG5), TRUE, fn, NULL);

      // dummy read the header.. the header isn't supposed to be compressed
      for (unsigned int i=0;i<(unsigned int)headersize;i++)
fputc(fgetc(finput),foutput);

      unsigned char * Dictionary = new unsigned char[LZSSWINDOW_LENGTH];
```

```
	unsigned char * LookAhead = new unsigned char[LZSSWINDOW_LENGTH];
	int OutputBitCount = 0;
	unsigned int OutputBitBuffer = 0;
	for (i=0;i<LZSSWINDOW_LENGTH;i++)
	{
		Dictionary[i] = 0;
		LookAhead[i] = 0;
	}
	int pDict = 0;

	i=0;
	int LookAheadLength = 0;
	BOOL bDone = FALSE;
	BOOL bOverSized = FALSE;
	int character;

	while	((LookAheadLength<LZSSWINDOW_LENGTH)	&&	((character	=
fgetc(finput)) != EOF))
		LookAhead[LookAheadLength++] = (unsigned char)character;

	if (LZSSBITLEN+LZSSBITPOS > 16)
		bOverSized = 1;

	while (LookAheadLength > 0)
	// repeat until we've cleaned out the LookAhead buffer
	{
		unsigned int position, length;
		LZSS_Scan(Dictionary,LookAhead,position,length);
		if (length == 0 || length == 1 || length == 2)
		{
			LZSS_OutputCode(foutput,1,1,OutputBitCount,OutputBitBuffer);
			LZSS_OutputCode(foutput,LookAhead[0],8,OutputBitCount,Output
BitBuffer);
			length = 1;
			lunCompressedCount++;
		}
		else
		{
			LZSS_OutputCode(foutput,0,1,OutputBitCount,OutputBitBuffer);
			LZSS_OutputCode(foutput,position	|	((length	-	2	-
bOverSized) << LZSSBITPOS)
				,LZSSBITPOS+LZSSBITLEN,OutputBitCount,OutputBitBuffer)
;
			lunCompressedCount+=length;
		}

		// copy lookahead code into dictionary, and do it cyclically.
this will
		// overwrite old entries once the dictionary has been filled
		for (i=0;i<length;i++)
		{
			Dictionary[pDict] = LookAhead[i];
			pDict = ((pDict + 1) % LZSSWINDOW_LENGTH);
		}
		// delete the used code from the lookahead
		for	(unsigned	int	j=0;j<LookAheadLength-length	&&
j<LZSSWINDOW_LENGTH-1;j++)
			LookAhead[j] = LookAhead[j+length];
```

```
            LookAheadLength-=length;

            if (character == EOF)
                  bDone = TRUE;
            // fill out the LookAhead buffer with any remaining characters
            while (LookAheadLength<LZSSWINDOW_LENGTH && !bDone &&
                  (character = fgetc(finput))  != EOF)
                  LookAhead[LookAheadLength++] = (unsigned char)character;

            // update the progress indicator
            intPtr->ProgressUpdate((int)((float)lunCompressedCount/(float)lOri
ginalLength*100.0f));
      }

      LZSS_OutputCode(foutput,0,1,OutputBitCount,OutputBitBuffer);
      LZSS_OutputCode(foutput,(1          <<          (LZSSBITPOS+LZSSBITLEN))-
1,LZSSBITPOS+LZSSBITLEN,
            OutputBitCount,OutputBitBuffer);
      LZSS_OutputCode(foutput,1,1,OutputBitCount,OutputBitBuffer);
      LZSS_OutputCode(foutput,0,8,OutputBitCount,OutputBitBuffer);
      LZSS_OutputCode(foutput,1,1,OutputBitCount,OutputBitBuffer);
      LZSS_OutputCode(foutput,0,8,OutputBitCount,OutputBitBuffer);
      delete[] Dictionary;
      delete[] LookAhead;
      fclose(foutput);
      fclose(finput);

      long lFinalLength;
      // delete our temporary file
      FILE *ftemp = fopen("____temp", "r");
      fseek(ftemp,0,SEEK_END);
      lFinalLength = ftell(ftemp);
      fclose(ftemp);
//    _unlink("____temp");
      TCHAR fnew[80], fext[6] = ".org";
      sprintf(fnew,"%s%s",filename,fext);
      MoveFile("____temp",fnew);
#ifdef _DEBUG
      char buffer[200];
      sprintf(buffer,"Total     file     length     before     compression     =
%d\n",lFinalLength);
      MessageBox(NULL,buffer,"Final Statistics",MB_OK);
#endif
      intPtr->ProgressEnd();

      foutput = fopen(filename, "r+b");
      fseek(foutput,0,SEEK_END);
      lFinalLength = ftell(foutput);
      rewind(foutput);
      for (i=0;i<(unsigned int)headersize-4;i++) fgetc(foutput);
      fpos_t pos;
      fgetpos(foutput,&pos);
      rewind(foutput);
      fsetpos(foutput,&pos);
      unsigned int ival = lFinalLength;
      unsigned char* ucVal = (unsigned char*) &ival;
      fputc(ucVal[0],foutput);
      fputc(ucVal[1],foutput);
```

```
        fputc(ucVal[2],foutput);
        fputc(ucVal[3],foutput);

        return TRUE;
}
```

D.2 Decompressor Source Code

D.2.1 StdAfx.h

```
// stdafx.h : include file for standard system include files,
//  or project specific include files that are used frequently, but
//      are changed infrequently
//

#if !defined(AFX_STDAFX_H__A4F36373_C3EB_11D1_87F8_00AA00B9C442__INCLUDED_)
#define AFX_STDAFX_H__A4F36373_C3EB_11D1_87F8_00AA00B9C442__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#define VC_EXTRALEAN            // Exclude rarely-used stuff from Windows
headers

#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>                 // MFC support for Windows Common
Controls

#include <afxole.h>

#endif // _AFX_NO_AFXCMN_SUPPORT


//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

#endif                                                                  //
!defined(AFX_STDAFX_H__A4F36373_C3EB_11D1_87F8_00AA00B9C442__INCLUDED_)
```

D.2.2 StdAfx.cpp

```
// stdafx.cpp : source file that includes just the standard includes
//    Decompressor.pch will be the pre-compiled header
//    stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"
```

D.2.3 Resource.h

```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by Decompressor.rc
//
#define IDD_ABOUTBOX                    100
#define IDR_MAINFRAME                   128
#define IDR_DECOMPTYPE                  129
#define IDD_GETURL                      130
#define IDR_RAWTITLE                    135
#define IDB_PLAYBUTTONS                 137
#define IDD_LOGWINDOW                   138
#define IDD_DIALOGTEST                  139
#define IDR_RAWBUFFER                   141
#define IDD_SPEED                       142
#define IDW_PLAYBUTTONS                 500
#define IDW_STYLES                      501
#define IDW_LOGGER                      502
#define IDC_EDIT                        1000
#define IDC_FPS                         1002
#define IDC_FSIZE                       1003
#define IDC_POLYS                       1004
#define IDC_VERTS                       1005
#define IDC_EDITTEXT                    1006
#define IDC_EDIT1                       1007
#define IDC_RADIO1                      1008
#define IDC_RADIO2                      1009
#define IDC_RADIO3                      1010
#define IDC_RADIO4                      1011
#define IDC_RADIO5                      1012
#define ID_FILE_OPENLOCATION            32771
#define ID_ANIMATION_PLAY               32772
#define ID_ANIMATION_PAUSE              32773
#define ID_ANIMATION_STOP               32774
#define ID_VIEW_DIAGNOSTICS             32775
#define ID_ANIMATION_DOWNLOADSPEED      32776

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_3D_CONTROLS                     1
#define _APS_NEXT_RESOURCE_VALUE        143
#define _APS_NEXT_COMMAND_VALUE         32777
#define _APS_NEXT_CONTROL_VALUE         1013
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

D.2.4 Quaternion.h

```
// Quaternion and Matrix math management system.
// Here lies the fun math

void QuaternionSlerp(float relativeTime,float spin,float *quat0,float *quat1,
                                float &xf, float &yf, float &zf, float &sf);

void QuaternionToMatrix(float s, float x, float y, float z,
                                      float *Matrix);

void QuaternionMultiply(float *q1, float *q2, float *qo);

void MatrixInverse(float *Original,float *Inverse);

void MatrixToQuaternion(float *Matrix,
                                      float &s, float &x, float &y, float &z);

void MatrixMultiply(float *Matrix1, float *Matrix2, float *MatrixOut);

void VecMatrixMultiply(float *Vector, float *Matrix, float *VectorOut);

void MatrixCopy(float *Src, float *Dest);

void TransposeMatrix(float *Src);

void VectorCrossProduct(float *Vec0, float *Vec1, float *VecOut);

void VectorNormalize(float *Vector);

void ConvertAAtoQuat(float angle, float *axis, float *quat);

float Ease( float t, float a, float b);

void qexp( float *q, float *dest );

void qmul( float* q1, float* q2, float* dest );

void qlog( float* q, float *dest );

// Returns dot product of normilized quternions q1*q2
float qdotunit( float* q1, float* q2 );

void qnegate( float *q );

void qlndif( float* p, float* q, float *dest );

void qinv( float *q, float *dest );
```

D.2.5 Quaternion.cpp

```
#include "stdafx.h"
#include "Quaternion.h"
#include "../Compressor/Opcodes.h"
#include <math.h>

void FloatSwap(float &f1, float &f2)
{
	float temp = f1;
	f1 = f2;
	f2 = temp;
}

float static inline fsqrt(float a)
{
/*	_asm
	{
		fld   dword ptr [a]
		fsqrt
	}
*/	float fval;
	_asm
	{
		fld         dword ptr [a]
		fsqrt
		fstp  dword ptr [fval]
	}
	return fval;
}

void QuaternionSlerp(float relativeTime,float spin,float *quat0,float *quat1,
                                float &xf, float &yf, float &zf, float &sf)
{
	float sclp,sclq, omega, sinom;
	xf = quat0[0];
	yf = quat0[1];
	zf = quat0[2];
	sf = quat0[3];
	float cosom = quat0[0] * quat1[0] + quat0[1] * quat1[1] +
	                quat0[2] * quat1[2] + quat0[3] * quat1[3];

/*	float alpha = relativeTime,k1,k2,angle;

  if (1.0f - fabsf (cosom) < EPSILON) {
    k1 = 1.0f - alpha;
    k2 = alpha;
  } else {
    angle = acosf (cosom);
    sinom = sinf (angle);
    float angleSpin = angle + spin*FLOAT_PI;
    k1 = sinf (angle - alpha*angleSpin) / sinom;
    k2 = sinf (alpha*angleSpin) / sinom;
  }

  xf = k1*quat0[0] + k2*quat1[0];
  yf = k1*quat0[1] + k2*quat1[1];
```

```
  zf = k1*quat0[2] + k2*quat1[2];
  sf = k1*quat0[3] + k2*quat1[3];*/

//
      if ( (1.0f + cosom) > EPSILON)
      {
            if ( (1.0f - cosom) > EPSILON)
            {
                  omega = acosf(cosom);
                  sinom = sinf(omega);
                  float angleSpin = omega + spin*FLOAT_PI;
                  sclp = sinf( omega - relativeTime*angleSpin) / sinom;
                  sclq = sinf( relativeTime * angleSpin) / sinom;
            }
            else
            {
                  sclp = 1.0f - relativeTime;
                  sclq = relativeTime;
            }
            xf = sclp*quat0[0] + sclq*quat1[0];
            yf = sclp*quat0[1] + sclq*quat1[1];
            zf = sclp*quat0[2] + sclq*quat1[2];
            sf = sclp*quat0[3] + sclq*quat1[3];
      }
      else
      {
            xf = -quat0[1]; yf = quat0[0];
            zf = -quat0[3]; sf = quat0[2];
            sclp = sinf( (1.0f - relativeTime)*FLOAT_HALFPI );
            sclq = sinf( relativeTime * FLOAT_HALFPI );
            xf = sclp*quat0[0] + sclq * xf;
            yf = sclp*quat0[1] + sclq * yf;
            zf = sclp*quat0[2] + sclq * zf;
      }
}

void QuaternionToMatrix(float s, float x, float y, float z,
                                    float *Matrix)
{
      float xsq,ysq,zsq,xy,xz,xw,yz,yw,zw;
      xsq = 2.0f * x * x;      ysq = 2.0f * y * y;      zsq = 2.0f * z * z;
      xy = 2.0f * x * y;       xz = 2.0f * x * z;       xw = 2.0f * x * s;
      yz = 2.0f * y * z;       yw = 2.0f * y * s;       zw = 2.0f * z * s;

      Matrix[0] = 1.0f - ysq - zsq;
      Matrix[4] = xy - zw; // was 1
      Matrix[8] = xz + yw;// was 2
      Matrix[1] = xy + zw;
      Matrix[5] = 1.0f - xsq - zsq;
      Matrix[9] = yz - xw;
      Matrix[2] = xz - yw;
      Matrix[6] = yz + xw;
      Matrix[10]= 1.0f - xsq - ysq;

      // bottom line, and righthand line
      Matrix[3]  =  Matrix[7]  =  Matrix[11]  =  Matrix[12]  =  Matrix[13]  =
Matrix[14] = 0.0f;
      Matrix[15] = 1.0f;
```

```
}

void QuaternionMultiply(float *q1, float *q2, float *qo)
{
      // multiplies two quaternions and returns the value in Qo
      qo[0] = q1[3]*q2[0] + q1[0]*q2[3] + q1[1]*q2[2] - q1[2]*q2[1];
      qo[1] = q1[3]*q2[1] - q1[0]*q2[2] + q1[1]*q2[3] + q1[2]*q2[0];
      qo[2] = q1[3]*q2[2] + q1[0]*q2[1] - q1[1]*q2[0] + q1[2]*q2[3];
      qo[3] = q1[3]*q2[3] - q1[0]*q2[0] - q1[1]*q2[1] - q1[2]*q2[2];
}

void MatrixToQuaternion(float *Matrix,
                                    float &s, float &x, float &y, float &z)
{
      float trace; // diagonal of matrix
      trace = Matrix[0] + Matrix[5] + Matrix[10];
//    if (trace > 0.0f)
      {
            trace = fsqrt(trace + 1.0f);
            float invtrace = 0.5f / trace;
            s = 0.5f * trace;
            x = (Matrix[9] - Matrix[7])*invtrace;
            y = (Matrix[2] - Matrix[8])*invtrace;
            z = (Matrix[4] - Matrix[1])*invtrace;
      }
/*    else
      {
            int i,j,k;
            i = 0;
            if (Matrix[5] > Matrix[0]) i = 1;
            if (Matrix[10] > Matrix[4*i+i]) i = 2;
            j = (i+1) % 3; k = (j+1) % 3;
            trace = fsqrt( (Matrix[4*i+i] - (Matrix[j*4+j] + Matrix[k*4+k]))
+ 1.0f);

      }*/
}

void MatrixMultiply(float *Matrix1, float *Matrix2, float *MatrixOut)
// Matrices are assumed to be 4x4 structures
{
    int i,j;
      float *M1 = Matrix1, *M2, *MO = MatrixOut;
      for(i = 0; i < 4; i++)
      {
            M2 = Matrix2;
        for(j = 0; j < 4; j++)
            {
/*                      MatrixOut[i][j] = Matrix1[i][0] * Matrix2[0][j] +
                                    Matrix1[i][1] * Matrix2[1][j] +
                                    Matrix1[i][2] * Matrix2[2][j] +
                                    Matrix1[i][3] * Matrix2[3][j];*/
                  *(MO++) = *(M1)   * *(M2)   + *(M1+1) * *(M2+4) +
                                  *(M1+2) * *(M2+8) + *(M1+3) * *(M2+12);
                  M2++;
            }
            M1+=4;
```

```
        }
}

void VecMatrixMultiply(float *Vector, float *Matrix, float *VectorOut)
// multiplies a 4 part vector by a matrix [4x4]
{
        float *V = Vector, *M = Matrix, *VO = VectorOut;
        for(int i = 0; i < 4; i++)
                *(VO++) = *(M+i) * *(V) + *(M+i+4) * *(V+1) +
                                *(M+i+8) * *(V+2) + *(M+i+12) * *(V+3);
}

void MatrixCopy(float *Src, float *Dest)
{
        float *M1 = Src, *M2 = Dest;
        for (int i=0;i<16;i+=4)
        {
                *(M2  ) = *(M1  );
                *(M2+1) = *(M1+1);
                *(M2+2) = *(M1+2);
                *(M2+3) = *(M1+3);
                M1+=4; M2+=4;
        }
}

void MatrixInverse(float *Original,float *Inverse)
// Calculates the inverse for the given matrix using Guass Jordan elimination
// specialized for a 4x4 matrix
{
        float dupe[16];
        for (int i=0;i<16;i++)
                dupe[i] = Original[i];

        int ipiv[4] = {0,0,0,0};
        int indxc[4],indxr[4], irow, icol;

        float big, fpivinv;

        for (i=0;i<4;i++)
        {
                big = 0.0f;
                for (int j=0;j<4;j++)
                        if (ipiv[j] != 1)
                                for (int k=0;k<4;k++)
                                {
                                        if (ipiv[k] == 0)
                                        {
                                                if (fabsf(dupe[j*4+k]) >= big)
                                                {
                                                        big = fabsf(dupe[j*4+k]);
                                                        irow = j;
                                                        icol = k;
                                                }
                                        }
                                        else if (ipiv[k] > 1) return;
                                }
                ++(ipiv[icol]);
```

```
            if (irow != icol)
                  for (int l=0;l<4;l++)
                  {
                        FloatSwap(dupe[irow*4+l],dupe[icol*4+l]);
                        FloatSwap(Inverse[irow*4+l],Inverse[icol*4+l]);
                  }

            indxr[i] = irow;
            indxc[i] = icol;
            fpivinv = 1.0f / dupe[icol*4+icol];
            dupe[icol*4+icol] = 1.0f;
            for (int l=0;l<4;l++)
            {
                  dupe[icol*4+l] *= fpivinv;
                  Inverse[icol*4+l] *= fpivinv;
            }
            for (j=0;j<4;j++)
                  if (j!=icol)
                  {
                        float temp = dupe[j*4+icol];
                        dupe[j*4+icol] = 0.0f;
                        for (l=0;l<4;l++)
                        {
                              dupe[j*4+l] -= dupe[icol*4+l] * temp;
                              Inverse[j*4+l] -= Inverse[icol*4+l] * temp;
                        }
                  }
      }
      for (int j=3;j>=0;j--)
      {
            if (indxr[j] != indxc[j])
                  for (int k=0;k<4;k++)
                        FloatSwap(dupe[k*4+indxr[j]],dupe[k*4+indxc[j]]);
      }
}

void TransposeMatrix(float *Src)
{
      FloatSwap(Src[1],Src[4]);
      FloatSwap(Src[2],Src[8]);
      FloatSwap(Src[3],Src[12]);
      FloatSwap(Src[6],Src[9]);
      FloatSwap(Src[7],Src[13]);
      FloatSwap(Src[11],Src[14]);
}

void VectorCrossProduct(float *Vec0, float *Vec1, float *VecOut)
{
      VecOut[0] = Vec0[1]*Vec1[2] - Vec0[2]*Vec1[1];
      VecOut[1] = Vec0[2]*Vec1[0] - Vec0[0]*Vec1[2];
      VecOut[2] = Vec0[0]*Vec1[1] - Vec0[1]*Vec1[0];
}

void VectorNormalize(float *Vector)
{
      float sum = *(Vector) * *(Vector) + *(Vector+1) * *(Vector+1) +
*(Vector+2) * *(Vector+2);
      float factor = 1.0f / fsqrt(sum);
```

```
      *(Vector  ) *=factor;
      *(Vector+1) *=factor;
      *(Vector+2) *=factor;
}

void ConvertAAtoQuat(float angle, float *axis, float *quat)
{
      quat[0] = axis[0] * sinf(angle*0.5f);
      quat[1] = axis[1] * sinf(angle*0.5f);
      quat[2] = axis[2] * sinf(angle*0.5f);
      quat[3] = cosf(angle*0.5f);
}

float Ease( float t, float a, float b)
{
// page 355 of Watt & Watt outlines this as speed control
      float k;
      float s = a+b;

      if (s == 0.0f) return t;
      if (s > 1.0f) {
            a = a/s;
            b = b/s;
      }
      k = 1.0f/(2.0f-a-b);
      if (t < a) return ((k/a)*t*t);
      else  {
            if (t < 1.0f-b)   {
                  return (k*(2*t - a));
            }     else {
                  t = 1.0f-t;
                  return (1.0f-(k/b)*t*t);
            }
      }
}

void qexp( float *q, float *dest ) // Calculate quaternion`s exponent.
{
      float d,d1;
      d = sqrtf( *(q) * *(q) + *(q+1)* *(q+1) + *(q+2)* *(q+2) );
      if (d > 0.0f) d1 = sinf(d)/d; else d1 = 1.0f;
      dest[3] = cosf(d);
      dest[0] = q[0]*d1;
      dest[1] = q[1]*d1;
      dest[2] = q[2]*d1;
}

//    Calculates quaternion product dest = q1 * q2.
void qmul( float* q1, float* q2, float* dest )
{
      dest[0] = q1[3]*q2[3] - q1[0]*q2[0] - q1[1]*q2[1] - q1[2]*q2[2];
      dest[1] = q1[3]*q2[0] + q1[0]*q2[3] + q1[1]*q2[2] - q1[2]*q2[1];
      dest[2] = q1[3]*q2[1] + q1[1]*q2[3] + q1[2]*q2[0] - q1[0]*q2[2];
      dest[3] = q1[3]*q2[2] + q1[2]*q2[3] + q1[0]*q2[1] - q1[1]*q2[0];
}

// Returns dot product of normilized quternions q1*q2
float qdotunit( float* q1, float* q2 )
```

```
{
      return q1[0]*q2[0] + q1[1]*q2[1] + q1[2]*q2[2] + q1[3]*q2[3];
}

float qmod( float* q )  // Returns modul of quaternion
{
      float d;
      d = sqrtf(q[0]*q[0] + q[1]*q[1] + q[2]*q[2] + q[3]*q[3]);
      if (d == 0) d = 1;      // for some case.
      return d;
}

void qnegate( float *q )      // Negates q;
{
      float d;
      d = 1.0f/qmod(q);
      q[3] *=  d;
      q[0] *= -d;
      q[1] *= -d;
      q[2] *= -d;
}

void qinv( float *q, float *dest )  //  Multiplicative inverse of q.
{
      float d;
      d = q[0]*q[0] + q[1]*q[1] + q[2]*q[2] + q[3]*q[3];
      if (d != 0.0f) d = 1.0f/d;    else  d = 1.0f;
      dest[3] =  q[3]* d;
      dest[0] = -q[0]* d;
      dest[1] = -q[1]* d;
      dest[2] = -q[2]* d;
}

void qlog( float* q, float *dest )  // Calculate quaternion`s logarithm.
{
      float d;
      d = sqrtf( q[0]*q[0] + q[1]*q[1] + q[2]*q[2] );
      if (q[3] != 0.0)  d = atanf(d/q[3]);      else  d = FLOAT_PI*0.5f;
      dest[3] = 0.0;
      dest[0] = q[0]*d;
      dest[1] = q[1]*d;
      dest[2] = q[2]*d;
}


// Calculate logarithm of the relative rotation from p to q
void qlndif( float* p, float* q, float *dest )
{
      float inv[4],dif[4];
      float d,d1;
      float s;

      qinv( p,inv );                  // inv = -p;
      QuaternionMultiply(inv,q,dif);
                                          // dif = -p*q

      d = sqrtf( dif[0]*dif[0] + dif[1]*dif[1] + dif[2]*dif[2] );
      s = p[0]*q[0] + p[1]*q[1] + p[2]*q[2] + p[3]*q[3];
```

```
        if (s != 0.0f)    d1 = atanf(d/s);  else  d1 = FLOAT_PI*0.5f;
        if (d != 0.0f)  d1 /= d;

        dest[3] = 0;

        dest[0] = dif[0]*d1;
        dest[1] = dif[1]*d1;
        dest[2] = dif[2]*d1;
}
```

D.2.6 Texture.h

```
// Texture.h: interface for the CTexture class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_TEXTURE_H__8DBA47E1_A09A_11D3_9016_00AA00B9C442__INCLUDED_)
#define AFX_TEXTURE_H__8DBA47E1_A09A_11D3_9016_00AA00B9C442__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "../Compressor/Opcodes.h"

class CTexture
{
public:
	float fShininess;
	float SpecularColor[4];
	BOOL bTextureInUse;
	unsigned int iTextureBindNum;
	BOOL bTextureUnbound;
	unsigned char * MipMaps[MAXTEXTUREPOWER+1];
	int iCurrentMipLevel;
	float fTextureMax;
	float fTextureMin;
	int iTextureTransformType;
	CTexture *next;
	DumpMipMap(char *filename, unsigned int power);
	SetTextureType(int type, float min, float max);
	CTexture();
	virtual ~CTexture();
};

#endif                                                                    //
!defined(AFX_TEXTURE_H__8DBA47E1_A09A_11D3_9016_00AA00B9C442__INCLUDED_)
```

D.2.7 Texture.cpp

```
// Texture.cpp: implementation of the CTexture class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "decompressor.h"
#include "Texture.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CTexture::CTexture()
{
	bTextureInUse = FALSE;
	iTextureTransformType = TEXTYPE_NONE;
	iCurrentMipLevel = MINTEXTUREPOWER;
	for (int i=0;i<MAXTEXTUREPOWER+1;i++)
		MipMaps[i] = NULL;
	bTextureUnbound = TRUE;
	iTextureBindNum = NULL;
	fShininess = 5.0f;
	SpecularColor[0] = 1.0f;
	SpecularColor[1] = 1.0f;
	SpecularColor[2] = 1.0f;
	SpecularColor[3] = 0.5f;
	next = NULL;
}

CTexture::~CTexture()
{
	for (int i=0;i<MAXTEXTUREPOWER+1;i++)
		if (MipMaps[i])
			delete[] MipMaps[i];
}

CTexture::SetTextureType(int type, float min, float max)
{
	iTextureTransformType = TEXTYPE_HAAR;
	fTextureMin = min;
	fTextureMax = max;
}

CTexture::DumpMipMap(char *filename, unsigned int power)
{
	FILE *fout;
	if ((fout = fopen(filename, "wb")) == NULL)
	{
		fprintf (stderr, "Cannot open output file.\n");
```

```
            return 0;
        }
        unsigned char *D = MipMaps[power];
        unsigned int size = 1;
        for (unsigned int count=0;count<power;count++) size*=2;

        for (unsigned int y=0;y<size;y++)
            for (unsigned int x=0;x<size;x++)
                for (unsigned int c=0;c<3;c++)
                    fputc(*(D++),fout);
        fflush(fout);
        fclose(fout);
        return 0;
}
```

D.2.8 Logger.h

```
#if !defined(AFX_LOGGER_H__7F0C9309_6D3C_11D3_8FF2_00AA00B9C442__INCLUDED_)
#define AFX_LOGGER_H__7F0C9309_6D3C_11D3_8FF2_00AA00B9C442__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// Logger.h : header file
//

#include "resource.h"

/////////////////////////////////////////////////////////////////////////////
// CLogger dialog

class CLogger : public CDialog
{
// Construction
public:
      BOOL Create();
      CLogger(CWnd* pParent = NULL);   // standard constructor
      CWnd* m_pParent;
      int m_nID;

// Dialog Data
      //{{AFX_DATA(CLogger)
      enum { IDD = IDD_LOGWINDOW };
            // NOTE: the ClassWizard will add data members here
      //}}AFX_DATA


// Overrides
      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CLogger)
      protected:
      virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
      //}}AFX_VIRTUAL

// Implementation
protected:

      // Generated message map functions
      //{{AFX_MSG(CLogger)
            // NOTE: the ClassWizard will add member functions here
      //}}AFX_MSG
      DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.

#endif                                                                    //
!defined(AFX_LOGGER_H__7F0C9309_6D3C_11D3_8FF2_00AA00B9C442__INCLUDED_)
```

D.2.9 Logger.cpp

```
// Logger.cpp : implementation file
// This allows us to log data to a dialog window

#include "stdafx.h"
#include "decompressor.h"
#include "Logger.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CLogger dialog


CLogger::CLogger(CWnd* pParent /*=NULL*/)
      : CDialog(CLogger::IDD, pParent)
{
      //{{AFX_DATA_INIT(CLogger)
            // NOTE: the ClassWizard will add member initialization here
      //}}AFX_DATA_INIT
      m_pParent = pParent;
      m_nID = CLogger::IDD;
}


void CLogger::DoDataExchange(CDataExchange* pDX)
{
      CDialog::DoDataExchange(pDX);
      //{{AFX_DATA_MAP(CLogger)
            // NOTE: the ClassWizard will add DDX and DDV calls here
      //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CLogger, CDialog)
      //{{AFX_MSG_MAP(CLogger)
            // NOTE: the ClassWizard will add message map macros here
      //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CLogger message handlers

BOOL CLogger::Create()
{
      return CDialog::Create(m_nID, m_pParent);
}
```

D.2.10 GetUrl.h

```
#if !defined(AFX_GETURL_H__A4F36359_C3EB_11D1_87F8_00AA00B9C442__INCLUDED_)
#define AFX_GETURL_H__A4F36359_C3EB_11D1_87F8_00AA00B9C442__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// GetUrl.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CGetUrl dialog - retrieves an url from a user

class CGetUrl : public CDialog
{
// Construction
public:
      CGetUrl(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
      //{{AFX_DATA(CGetUrl)
      enum { IDD = IDD_GETURL };
      CString     m_UrlName;
      //}}AFX_DATA


// Overrides
      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CGetUrl)
      protected:
      virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
      //}}AFX_VIRTUAL

// Implementation
protected:

      // Generated message map functions
      //{{AFX_MSG(CGetUrl)
            // NOTE: the ClassWizard will add member functions here
      //}}AFX_MSG
      DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

#endif                                                                    //
!defined(AFX_GETURL_H__A4F36359_C3EB_11D1_87F8_00AA00B9C442__INCLUDED_)
```

D.2.11 GetUrl.cpp

```
// GetUrl.cpp : implementation file
//

#include "stdafx.h"
#include "Decompressor.h"
#include "GetUrl.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CGetUrl dialog


CGetUrl::CGetUrl(CWnd* pParent /*=NULL*/)
      : CDialog(CGetUrl::IDD, pParent)
{
      //{{AFX_DATA_INIT(CGetUrl)
      m_UrlName = _T("");
      //}}AFX_DATA_INIT
}


void CGetUrl::DoDataExchange(CDataExchange* pDX)
{
      CDialog::DoDataExchange(pDX);
      //{{AFX_DATA_MAP(CGetUrl)
      DDX_Text(pDX, IDC_EDIT1, m_UrlName);
      //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CGetUrl, CDialog)
      //{{AFX_MSG_MAP(CGetUrl)
            // NOTE: the ClassWizard will add message map macros here
      //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CGetUrl message handlers
```

D.2.12 GetDownloadSpeed.h

```
#if
!defined(AFX_GETDOWNLOADSPEED_H__97EBBCA3_F9EE_11D3_909C_00AA00B9C442__INCLUDE
D_)
#define
AFX_GETDOWNLOADSPEED_H__97EBBCA3_F9EE_11D3_909C_00AA00B9C442__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// GetDownloadSpeed.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CGetDownloadSpeed dialog

class CGetDownloadSpeed : public CDialog
{
// Construction
public:
      int iSpeedSelector;
      CGetDownloadSpeed(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
      //{{AFX_DATA(CGetDownloadSpeed)
      enum { IDD = IDD_SPEED };
            // NOTE: the ClassWizard will add data members here
      //}}AFX_DATA

// Overrides
      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CGetDownloadSpeed)
      protected:
      virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
      //}}AFX_VIRTUAL

// Implementation
protected:

      // Generated message map functions
      //{{AFX_MSG(CGetDownloadSpeed)
      afx_msg void OnRadio1();
      afx_msg void OnRadio2();
      afx_msg void OnRadio3();
      afx_msg void OnRadio4();
      afx_msg void OnRadio5();
      virtual BOOL OnInitDialog();
      //}}AFX_MSG
      DECLARE_MESSAGE_MAP()
};
//{{AFX_INSERT_LOCATION}}
#endif                                                                    //
!defined(AFX_GETDOWNLOADSPEED_H__97EBBCA3_F9EE_11D3_909C_00AA00B9C442__INCLUDE
D_)
```

D.2.13 GetDownloadSpeed.cpp

```
// GetDownloadSpeed.cpp : implementation file
//

#include "stdafx.h"
#include "decompressor.h"
#include "GetDownloadSpeed.h"
#include "../Compressor/Opcodes.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CGetDownloadSpeed dialog

CGetDownloadSpeed::CGetDownloadSpeed(CWnd* pParent /*=NULL*/)
      : CDialog(CGetDownloadSpeed::IDD, pParent)
{
      //{{AFX_DATA_INIT(CGetDownloadSpeed)
            // NOTE: the ClassWizard will add member initialization here
      //}}AFX_DATA_INIT

      iSpeedSelector = SPEED_288;
}


void CGetDownloadSpeed::DoDataExchange(CDataExchange* pDX)
{
      CDialog::DoDataExchange(pDX);
      //{{AFX_DATA_MAP(CGetDownloadSpeed)
            // NOTE: the ClassWizard will add DDX and DDV calls here
      //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CGetDownloadSpeed, CDialog)
      //{{AFX_MSG_MAP(CGetDownloadSpeed)
      ON_BN_CLICKED(IDC_RADIO1, OnRadio1)
      ON_BN_CLICKED(IDC_RADIO2, OnRadio2)
      ON_BN_CLICKED(IDC_RADIO3, OnRadio3)
      ON_BN_CLICKED(IDC_RADIO4, OnRadio4)
      ON_BN_CLICKED(IDC_RADIO5, OnRadio5)
      //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CGetDownloadSpeed message handlers

BOOL CGetDownloadSpeed::OnInitDialog()
{
      CDialog::OnInitDialog();

      CButton* pCRB1 = (CButton*) GetDlgItem(IDC_RADIO1);
```

```
	CButton* pCRB2 = (CButton*) GetDlgItem(IDC_RADIO2);
	CButton* pCRB3 = (CButton*) GetDlgItem(IDC_RADIO3);
	CButton* pCRB4 = (CButton*) GetDlgItem(IDC_RADIO4);
	CButton* pCRB5 = (CButton*) GetDlgItem(IDC_RADIO5);

	switch (iSpeedSelector)
	{
	case SPEED_288:
		pCRB1->SetCheck(1);
		break;
	case SPEED_56k:
		pCRB2->SetCheck(1);
		break;
	case SPEED_ISDN:
		pCRB3->SetCheck(1);
		break;
	case SPEED_ADSL:
		pCRB4->SetCheck(1);
		break;
	case SPEED_T1:
		pCRB5->SetCheck(1);
		break;
	}

	return TRUE;  // return TRUE unless you set the focus to a control
		      // EXCEPTION: OCX Property Pages should return FALSE
}

void CGetDownloadSpeed::OnRadio1()
{
	iSpeedSelector = SPEED_288;
}

void CGetDownloadSpeed::OnRadio2()
{
	iSpeedSelector = SPEED_56k;
}

void CGetDownloadSpeed::OnRadio3()
{
	iSpeedSelector = SPEED_ISDN;
}

void CGetDownloadSpeed::OnRadio4()
{
	iSpeedSelector = SPEED_ADSL;
}

void CGetDownloadSpeed::OnRadio5()
{
	iSpeedSelector = SPEED_T1;
}
```

D.2.14 DialogLog.h

```
#if !defined(AFX_DIALOGLOG_H__7F0C930A_6D3C_11D3_8FF2_00AA00B9C442__INCLUDED_)
#define AFX_DIALOGLOG_H__7F0C930A_6D3C_11D3_8FF2_00AA00B9C442__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// DialogLog.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CDialogLog dialog - this shows the actual log window

class CDialogLog : public CDialog
{
// Construction
public:
      Clear();
      Log(char *text);
      BOOL Create();
      CDialogLog(CWnd* pParent = NULL);   // standard constructor
      CWnd* m_pParent;
      int m_nID;

// Dialog Data
      //{{AFX_DATA(CDialogLog)
      enum { IDD = IDD_DIALOGTEST };
      float m_fps;
      long  m_fsize;
      int         m_polys;
      int         m_verts;
      //}}AFX_DATA


// Overrides
      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CDialogLog)
      protected:
      virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
      //}}AFX_VIRTUAL

// Implementation
protected:

      // Generated message map functions
      //{{AFX_MSG(CDialogLog)
      virtual BOOL OnInitDialog();
      //}}AFX_MSG
      DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
#endif                                                                    //
!defined(AFX_DIALOGLOG_H__7F0C930A_6D3C_11D3_8FF2_00AA00B9C442__INCLUDED_)
```

D.2.15 DialogLog.cpp

```
// DialogLog.cpp : implementation file
//

#include "stdafx.h"
#include "decompressor.h"
#include "DialogLog.h"
#include "../Compressor/Opcodes.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CDialogLog dialog


CDialogLog::CDialogLog(CWnd* pParent /*=NULL*/)
      : CDialog(CDialogLog::IDD, pParent)
{
      //{{AFX_DATA_INIT(CDialogLog)
      m_fps = 0.0f;
      m_fsize = 0;
      m_polys = 0;
      m_verts = 0;
      //}}AFX_DATA_INIT
      m_pParent = pParent;
      m_nID = CDialogLog::IDD;
}


void CDialogLog::DoDataExchange(CDataExchange* pDX)
{
      CDialog::DoDataExchange(pDX);
      //{{AFX_DATA_MAP(CDialogLog)
      DDX_Text(pDX, IDC_FPS, m_fps);
      DDX_Text(pDX, IDC_FSIZE, m_fsize);
      DDX_Text(pDX, IDC_POLYS, m_polys);
      DDX_Text(pDX, IDC_VERTS, m_verts);
      //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CDialogLog, CDialog)
      //{{AFX_MSG_MAP(CDialogLog)
      //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CDialogLog message handlers

BOOL CDialogLog::Create()
{
      return CDialog::Create(m_nID, m_pParent);
```

```
}

CDialogLog::Log(char *text)
{
#ifndef DO_NOT_LOG
      CString cs(text);
      CRichEditCtrl * pCEOutput = (CRichEditCtrl*) GetDlgItem(IDC_EDITTEXT);
      int endc = pCEOutput->GetWindowTextLength();
      pCEOutput->SetSel(endc,endc);
      pCEOutput->ReplaceSel(cs);
      pCEOutput->LineScroll(1);
#endif
}

BOOL CDialogLog::OnInitDialog()
{
      CDialog::OnInitDialog();
      return TRUE;
}

CDialogLog::Clear()
{
      CString cs("");
      CRichEditCtrl * pCEOutput = (CRichEditCtrl*) GetDlgItem(IDC_EDITTEXT);
      int endc = pCEOutput->GetWindowTextLength();
      pCEOutput->SetSel(0,-1);
      pCEOutput->ReplaceSel(cs);
      pCEOutput->Clear();
      endc = pCEOutput->GetWindowTextLength();
}
```

D.2.16 Animation.h

```
// This file defines all of the animation database subsystems
// for the project, including structures for objects, lights,
// and cameras.

#if !defined(ANIMATION_HEADER)
#define ANIMATION_HEADER

#include <gl\gl.h>
#include <gl\glu.h>
#include <gl\glaux.h>
#include "..\Compressor\OpCodes.h"
#include "Texture.h"

class CKeyFrame
{
public:
      float angleaxis[4];
      float scale[3];
      BOOL bChecked;
      CKeyFrame();
      ~CKeyFrame();
      int Type;
      float fTime;
      int iTime;
      float continuity, tension, bias;
      float easeIn, easeOut;
      float inTan[4], outTan[4], pos[3], rot[4], roll;
      CKeyFrame* prev;
      CKeyFrame* next;
};

class CVisualObject
// This class defines visible objects
{
public:
      UpdateBoundaries(int From);
      UpdateRoll(float fTotalTime);
      SetOffset(float *offsetarray);
      CalculateTCBQuatDeriv(CKeyFrame *prev,CKeyFrame *curr,CKeyFrame *next);
      void CheckComplete() {bCompleted = ObjectComplete();}
      BOOL AddTexCoords(int Number, float floatarr[]);
      BOOL AddVertColors(int Number, unsigned char colorlist[ ]);
      TextDumpFaceList(unsigned int total, int* iList);
      BOOL AddNormals(float * NormArrary, int total);
      BOOL AddTCBScaleKey(int frame, float *scale, float tens, float cont,
float bias, float easeIn, float easeOut, float invTicksPerFrame);
      BOOL AddBezierScaleKey(int frame, float *scale, float *inTan, float
*outTan, float invTicksPerFrame);
      BOOL AddLinearScaleKey(int frame, float x, float y, float z,float
invTicksPerFrame);
      unsigned int iDisplayList;
      BOOL bHasArrays;
      SetScale(float x, float y, float z);
      AddBezierRollKey(int frame,float roll, float invTPF);
```

```
      BOOL WithinFustrum(float *BackTransform,float zNear,float zFar);
      GenerateVertexNormals();
      Reset();
      CVisualObject* Target;
      SetOrientation(float *quaternion);
      SetPosition(float x,float y, float z);
      unsigned char solidRed, solidGreen, solidBlue;
      BOOL bSolidColor;
      AssignSolidColor(unsigned char red, unsigned char green, unsigned char
blue);
      BOOL AddTCBRotationKey(int frame, float *rot,
          float tens, float cont, float bias, float easeIn, float easeOut,
            float invTicksPerFrame);
      BOOL AddTCBEnhRotationKey(int frame, short count,
            float angle, float *axis, float tens, float cont, float bias,
            float easeIn, float easeOut, float invTicksPerFrame);
      BOOL AddLinearRotationKey(int frame, float *rot,
            float invTicksPerFrame);
      BOOL AddBezierRotationKey(int frame, float *rot,
            float invTicksPerFrame);
      BOOL AddTCBPositionKey(int frame, float *pos, float tens, float cont,
            float bias, float easeIn, float easeOut,float invTicksPerFrame);
      BOOL AddBezierPositionKey(int frame, float *pos, float *inTan, float
*outTan,
            float invTicksPerFrame);
      BOOL AddLinearPositionKey(int frame, float x, float y, float z,
            float invTicksPerFrame);
      float MinX, MinY, MinZ;
      float MaxX, MaxY, MaxZ;
      float GetMinX() {return MinX;}
      float GetMaxX() {return MaxX;}
      float GetMinY() {return MinY;}
      float GetMaxY() {return MaxY;}
      float GetMinZ() {return MinZ;}
      float GetMaxZ() {return MaxZ;}
      void SetMaxX(float f) {MaxX = f;}
      void SetMinX(float f) {MinX = f;}
      void SetMaxY(float f) {MaxY = f;}
      void SetMinY(float f) {MinY = f;}
      void SetMaxZ(float f) {MaxZ = f;}
      void SetMinZ(float f) {MinZ = f;}
      UpdateTransform(float CurTime);
      UpdateScale(float fTotalTime);
      UpdateRotation(float fTotalTime);
      UpdatePosition(float fTotalTime);

      CVisualObject* ParentObject;
      BOOL AddPatch(int *intlist);
      BOOL AddFaces(int Number,int facelist[]);
      BOOL AddVertices(int number,float floatarr[]);
      CVisualObject(int VCount,int PCount, int FCount);
      CVisualObject();
      virtual ~CVisualObject();
      float             *VertexArray;
        // an array of xyz values for vertices
```

```
        float             *NormalArray;
          // an array of normal values for the vertices
        unsigned char     *VertexColorArray;
          // an array of vertex color indices that correspond to vertex numbers
        int                       *PatchArray;
          // an array of patches, each patch is a sub array of 16 values
          // that contain vertex numbers
        int                       *FaceArray;
        float             *TexCoordArray;

        int TotalFaces;
        int TotalVertices;
        int CurrentTotalVertices;
        int CurrentTotalNormals;
        int CurrentTotalColors;
        int CurrentTotalPatches;
        int CurrentTotalFaces;
        int CurrentTotalTexCoords;
        int TotalPatches;
        float Transform[16];
        float TimeLastUpdated;
        // used for constructing the offset matrix, to adjust the pivot
        float OffsetMatrix[16];

        BOOL bCompleted;
          // when bCompleted is true, we know the object is completely defined
          // because of this, we can then compile the polygon information
          // into vertex arrays.  Vertex arrays can speed up rendering by quite
          // a bit by removing the function call overhead from defining each
          // vertex

        float SclX, SclY, SclZ;
        float PosX, PosY, PosZ;
        float OriW, OriX, OriY, OriZ;
        float Roll;

        CKeyFrame*  RotKeyframes;
        CKeyFrame*  PosKeyframes;
        CKeyFrame*  RollKeyframes;
        CKeyFrame*  SclKeyframes;
        CKeyFrame*  CurrPosKeyframe;
        CKeyFrame*  CurrRotKeyframe;
        CKeyFrame*  CurrRollKeyframe;
        CKeyFrame*  CurrSclKeyframe;
        CTexture*   TextureMap;
        CVisualObject* next;
private:
        BOOL ObjectComplete();
};

class CLight : public CVisualObject
{
public:
        CLight();
        virtual ~CLight();
```

```
        BOOL bEnabled;  // is this light on?
        float Color[4];
        int Type;
        float fSpotCutoff;
        float fSpotExponent;
};

class CCamera : public CVisualObject
{
public:
        CCamera();
        float fFieldOfView;
        float fNearPlane;
        float fFarPlane;
};

class CAnimation
// The animation database that controls the entire animation
{
public:
        float BackgroundColor[3];
        Reset();
        CTexture* FindTexture(int Num);
        CTexture* AddTexture();
        CTexture* TextureListHead;
        int iTargetX;
        int iTargetY;
        BOOL bSimpleShading;
        int iSourceTPF;
        int iSourceFPS;
        unsigned int MajorVersion;
        unsigned int MinorVersion;
        BOOL bMaterialQuality;
        BOOL bVertexNormals;
        BOOL bBeginningNewAnimation;
        CVisualObject* FindObject(int Num);
        CVisualObject* AddVisualObject(int VCount, int PCount, int FCount);
        CAnimation();
        virtual ~CAnimation();

        int ReadyTillFrame;
        int FinalFrame;
        BOOL bTimeNotSet;

        CVisualObject* ObjectListHead;
        int TotalKeyFrames;

        BOOL LightingEnabled;
        CLight Ambient;
        CLight SceneLights[GL_MAX_LIGHTS-1];
        int iLightsInUse;

        float FramesSoFar;

        CCamera SceneCamera;

        DWORD OldTime;
```

```
        DWORD CurTime;
        DWORD StartTime;
};

#endif
```

D.2.17 Animation.cpp

```
//////////////////////////////////////////////////
// Animation.cpp

#include "stdafx.h"
#include "Animation.h"
#include "../Compressor/Opcodes.h"
#include "Quaternion.h"
#include "math.h"

//////////////////////////////////////////////////////////////////////
// CAnimation Class
//////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CAnimation::CAnimation()
{
	LightingEnabled = FALSE;

	ReadyTillFrame = -2;
	FinalFrame = 0x7fffffff;

	ObjectListHead = NULL;
	bTimeNotSet = TRUE;

	bBeginningNewAnimation = TRUE;

	TextureListHead = NULL;

	BackgroundColor[0] = 0.0f;
	BackgroundColor[1] = 0.0f;
	BackgroundColor[2] = 0.0f;
}

CAnimation::~CAnimation()
{
	CTexture *ctex = NULL;
	if (TextureListHead)
		ctex = TextureListHead->next;
	while(ctex)
	{
		delete TextureListHead;
		TextureListHead = ctex;
		ctex = ctex->next;
	}
	if (TextureListHead)
		delete TextureListHead;

	CVisualObject *curr = NULL;
	if (ObjectListHead)
		curr = ObjectListHead->next;
	while(curr)
	{
```

```
            delete ObjectListHead;
            ObjectListHead = curr;
            curr = curr->next;
      }
      if (ObjectListHead)
            delete ObjectListHead;
}

////////////////////////////////////////////////////////////////////////
// Global effects to animated sequences
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
// Material Handling
////////////////////////////////////////////////////////////////////////

CTexture* CAnimation::AddTexture()
// adds a texture to the animation set
{
      CTexture *CT = new CTexture;
      if (!TextureListHead)
            TextureListHead = CT;
      else
      {
            CTexture *curr = TextureListHead;
            while(curr->next)
                  curr = curr->next;
            curr->next = CT;
      }
      return CT;
}

CTexture* CAnimation::FindTexture(int Num)
{
      CTexture* curr = TextureListHead;
      int i = 0;
      while (i++<Num && curr)
      {
            curr = curr->next;
      }
      return curr;
}

////////////////////////////////////////////////////////////////////////
// Object Handling
////////////////////////////////////////////////////////////////////////

CVisualObject* CAnimation::AddVisualObject(int VCount, int PCount, int FCount)
// adds an object to the animation set.
{
      CVisualObject *CVO = new CVisualObject(VCount,PCount,FCount);
      if (!ObjectListHead)
            ObjectListHead = CVO;
      else
      {
            CVisualObject *curr = ObjectListHead;
            while(curr->next)
```

```
                curr = curr->next;
            curr->next = CVO;
        }
        return CVO;
}

CVisualObject* CAnimation::FindObject(int Num)
// given the object ID number, returns a pointer to that object
{
        CVisualObject* curr = ObjectListHead;
        int i = 0;
        while (i++<Num && curr)
        {
            curr = curr->next;
        }
        return curr;
}


////////////////////////////////////////////////////////////////////////
// CVisualObject Class
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
// Construction/Destruction
////////////////////////////////////////////////////////////////////////

CVisualObject::CVisualObject()
{
        TimeLastUpdated = -1.0f;
        TotalPatches = 0;
        TotalVertices = 0;
        TotalFaces = 0;
        NormalArray = NULL;
        VertexArray = NULL;
        VertexColorArray = NULL;
        TexCoordArray = NULL;
        PatchArray = NULL;
        bSolidColor = FALSE;
        FaceArray = NULL;

        next = NULL;
        ParentObject = NULL;
        bCompleted = FALSE;
        bHasArrays = FALSE;
        CurrentTotalNormals = 0;
        CurrentTotalColors = 0;
        CurrentTotalVertices = 0;
        CurrentTotalPatches = 0;
        CurrentTotalFaces = 0;
        CurrentTotalTexCoords = 0;

        CurrPosKeyframe = NULL;
        CurrRotKeyframe = NULL;
        CurrRollKeyframe = NULL;
        CurrSclKeyframe = NULL;
        PosKeyframes = NULL;
        RotKeyframes = NULL;
        RollKeyframes = NULL;
```

```
	SclKeyframes = NULL;

	SclX = 1.0f; SclY = 1.0f; SclZ = 1.0f;
	PosX = 0.0f; PosY = 0.0f; PosZ = 0.0f;
	OriX = 0.0f; OriY = 0.0f; OriZ = 0.0f; OriW = 1.0f;
	Target = NULL;

	OffsetMatrix[0] = 1.0f; OffsetMatrix[1] = 0.0f;
	OffsetMatrix[2] = 0.0f; OffsetMatrix[3] = 0.0f;
	OffsetMatrix[4] = 0.0f; OffsetMatrix[5] = 1.0f;
	OffsetMatrix[6] = 0.0f; OffsetMatrix[7] = 0.0f;
	OffsetMatrix[8] = 0.0f; OffsetMatrix[9] = 0.0f;
	OffsetMatrix[10]= 1.0f; OffsetMatrix[11]= 0.0f;
	OffsetMatrix[12]= 0.0f; OffsetMatrix[13]= 0.0f;
	OffsetMatrix[14]= 0.0f; OffsetMatrix[15]= 1.0f;

	TextureMap = NULL;
}

CVisualObject::CVisualObject(int VCount, int PCount, int FCount)
{
	TimeLastUpdated = -1.0f;
	TotalPatches = PCount;
	TotalVertices = VCount;
	TotalFaces = FCount;

	if (FCount > 0)
		FaceArray = new int[FCount*3];
	else FaceArray = NULL;

	if (VCount > 0)
	{
		NormalArray = new float[VCount*3];
		VertexArray = new float[VCount*3];
		VertexColorArray = new unsigned char[VCount*3];
		TexCoordArray = new float[VCount*2];
	}
	else
	{
		NormalArray = NULL;
		VertexArray = NULL;
		VertexColorArray = NULL;
		TexCoordArray = NULL;
	}

	if (PCount > 0)
		PatchArray = new int[PCount*16];
	else PatchArray = NULL;

	bSolidColor = FALSE;
	next = NULL;
	ParentObject = NULL;
	bCompleted = FALSE;
	bHasArrays = FALSE;
	CurrentTotalNormals = 0;
	CurrentTotalColors = 0;
	CurrentTotalTexCoords = 0;
	CurrentTotalVertices = 0;
```

```
	CurrentTotalPatches = 0;
	CurrentTotalFaces = 0;
	CurrentTotalTexCoords = 0;

	CurrPosKeyframe = NULL;
	CurrRotKeyframe = NULL;
	CurrRollKeyframe = NULL;
	CurrSclKeyframe = NULL;
	PosKeyframes = NULL;
	RotKeyframes = NULL;
	RollKeyframes = NULL;
	SclKeyframes = NULL;

	SclX = 1.0f; SclY = 1.0f; SclZ = 1.0f;
	PosX = 0.0f; PosY = 0.0f; PosZ = 0.0f;
	OriX = 0.0f; OriY = 0.0f; OriZ = 0.0f; OriW = 1.0f;
	Target = NULL;

	TextureMap = NULL;

	OffsetMatrix[0] = 1.0f; OffsetMatrix[1] = 0.0f;
	OffsetMatrix[2] = 0.0f; OffsetMatrix[3] = 0.0f;
	OffsetMatrix[4] = 0.0f; OffsetMatrix[5] = 1.0f;
	OffsetMatrix[6] = 0.0f; OffsetMatrix[7] = 0.0f;
	OffsetMatrix[8] = 0.0f; OffsetMatrix[9] = 0.0f;
	OffsetMatrix[10]= 1.0f; OffsetMatrix[11]= 0.0f;
	OffsetMatrix[12]= 0.0f; OffsetMatrix[13]= 0.0f;
	OffsetMatrix[14]= 0.0f; OffsetMatrix[15]= 1.0f;
}

CVisualObject::~CVisualObject()
{
	if (FaceArray)
//		delete[] FaceArray;
		delete[] FaceArray;
	if (NormalArray)
		delete[] NormalArray;
	if (VertexArray)
		delete[] VertexArray;
	if (VertexColorArray)
		delete[] VertexColorArray;
	if (TexCoordArray)
		delete[] TexCoordArray;
	if (PatchArray)
		delete[] PatchArray;
/*	while (PositionEvents)
	{
		CAnimEvent *curr = PositionEvents->next;
		delete PositionEvents;
		PositionEvents = curr;
	}*/
	while (PosKeyframes)
	{
		CKeyFrame *curr = PosKeyframes->next;
		delete PosKeyframes;
		PosKeyframes = curr;
	}
```

```
	while (RotKeyframes)
	{
		CKeyFrame *curr = RotKeyframes->next;
		delete RotKeyframes;
		RotKeyframes = curr;
	}
}

////////////////////////////////////////////////////////////////////////
// Object Handling
////////////////////////////////////////////////////////////////////////

// Visual component management

BOOL CVisualObject::AddVertices(int number, float floatarr[])
{
	if (number+CurrentTotalVertices > TotalVertices)
		return FALSE;
	float * VArray = VertexArray + CurrentTotalVertices*3;
	float * fList = floatarr;
	for(int i=0;i<number;i++)
	{
		*(VArray)++ = *(fList++);
		*(VArray)++ = *(fList++);
		*(VArray)++ = *(fList++);
//		VertexArray[CurrentTotalVertices*3+1] = floatarr[i*3+1];
//		VertexArray[CurrentTotalVertices*3+2] = floatarr[i*3+2];
	}
	CurrentTotalVertices+=number;
	if (ObjectComplete()) bCompleted = TRUE;
	return TRUE;
}

BOOL CVisualObject::AddNormals(float *NormArray, int total)
{
	if (total+CurrentTotalNormals > TotalVertices)
		return FALSE;
	float * NArray = NormalArray + CurrentTotalNormals*3;
	float * fList = NormArray;
	for(int i=0;i<total;i++)
	{
		*(NArray)++ = *(fList++);
		*(NArray)++ = *(fList++);
		*(NArray)++ = *(fList++);
	}
	CurrentTotalNormals+=total;
	if (ObjectComplete()) bCompleted = TRUE;
	return TRUE;
}

BOOL CVisualObject::AddFaces(int Number, int facelist [ ])
{
	if (Number+CurrentTotalFaces > TotalFaces)
		return FALSE;
	int * FArray = FaceArray+CurrentTotalFaces*3;
	int * pList = facelist;
	for(int i=0;i<Number;i++)
```

```
        {
                *(FArray++) = *(pList++);
                *(FArray++) = *(pList++);
                *(FArray++) = *(pList++);
//              FaceArray[CurrentTotalFaces*3+0] = facelist[i*3+0];
//              FaceArray[CurrentTotalFaces*3+1] = facelist[i*3+1];
//              FaceArray[CurrentTotalFaces*3+2] = facelist[i*3+2];
                CurrentTotalFaces++;
        }
        if (ObjectComplete())
                bCompleted = TRUE;
        return TRUE;
}

BOOL CVisualObject::AddPatch(int * intlist)
{
        if (16+CurrentTotalPatches > TotalPatches)
                return FALSE;
        for(int i=0;i<16;i++)
                PatchArray[CurrentTotalPatches*16+i] = intlist[i];
        CurrentTotalPatches++;
        if (ObjectComplete()) bCompleted = TRUE;
        return TRUE;
}

BOOL CVisualObject::AddVertColors(int Number, unsigned char colorlist[])
{
        if (Number+CurrentTotalColors > TotalVertices)
                return FALSE;
        unsigned char * CArray = VertexColorArray+CurrentTotalColors*3;
        unsigned char * cList = colorlist;
        for(int i=0;i<Number;i++)
        {
                *(CArray++) = *(cList++);
                *(CArray++) = *(cList++);
                *(CArray++) = *(cList++);
        }
        CurrentTotalColors+=Number;
        if (ObjectComplete())
                bCompleted = TRUE;
        return TRUE;
}

BOOL CVisualObject::AddTexCoords(int Number, float floatarr[])
{
        if (Number+CurrentTotalTexCoords > TotalVertices)
                return FALSE;
        float * TArray = TexCoordArray+CurrentTotalTexCoords*2;
        float * tList = floatarr;
        for(int i=0;i<Number;i++)
        {
                *(TArray++) = *(tList++);
                *(TArray++) = *(tList++);
        }
        CurrentTotalTexCoords+=Number;
        if (ObjectComplete())
                bCompleted = TRUE;
```

```
	return TRUE;
}

BOOL CVisualObject::ObjectComplete()
{

	return	(
		((CurrentTotalColors == TotalVertices || bSolidColor) ||
		 (CurrentTotalColors == 0 && !bSolidColor)) &&
		(CurrentTotalTexCoords == TotalVertices || !TextureMap ||
		(TextureMap &&
			TextureMap->iTextureTransformType == TEXTYPE_NONE)) &&
		CurrentTotalFaces == TotalFaces &&
		CurrentTotalPatches == TotalPatches &&
		CurrentTotalVertices == TotalVertices
			);
}

//////////////////////////////////////////////////////////////////////
// CLight Class
//////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CLight::CLight()  : CVisualObject()
{
	bEnabled = FALSE;
	Color[3] = 1.0f;
	Type = LIGHT_POINT;
	fSpotCutoff = 180.0f;
	fSpotExponent = 0.0f;
}

CLight::~CLight()
{
}

///  New Keyframing structure

CKeyFrame::CKeyFrame()
{
	prev = NULL;
	next = NULL;
	Type = CNT_LINPOS;
	bChecked = FALSE;
	inTan[0] = 0.0f;
	inTan[1] = 0.0f;
	inTan[2] = 0.0f;
	inTan[3] = 0.0f;
	outTan[0] = 0.0f;
	outTan[1] = 0.0f;
	outTan[2] = 0.0f;
	outTan[3] = 0.0f;
	easeIn = 0.0f;
	easeOut = 0.0f;
	bias = 0.0f;
	continuity = 0.0f;
```

```
	tension = 0.0f;
	angleaxis[0] = angleaxis[1] = angleaxis[2] = angleaxis[3] = 0.0f;
}

CKeyFrame::~CKeyFrame()
{
}

BOOL CVisualObject::AddLinearPositionKey(int frame, float x, float y, float z,
float invTicksPerFrame)
{
	CKeyFrame *newKey = new CKeyFrame;
	newKey->Type = CNT_LINPOS;
	newKey->iTime = frame;
	newKey->fTime = ((float) frame) * invTicksPerFrame;
	newKey->pos[0] = x;
	newKey->pos[1] = y;
	newKey->pos[2] = z;
	CKeyFrame *curr = PosKeyframes;
	if (!curr)
	{
		PosKeyframes = newKey;
		CurrPosKeyframe = PosKeyframes;
	}
	else
	{
		while (curr->next)
			curr = curr->next;
		curr->next = newKey;
		newKey->prev = curr->next;
	}
	return TRUE;
}

BOOL CVisualObject::AddBezierPositionKey(int frame, float *pos, float *inTan,
                                                                    float
*outTan,float invTicksPerFrame)
{
	CKeyFrame *newKey = new CKeyFrame;
	newKey->Type = CNT_BEZPOS;
	newKey->iTime = frame;
	newKey->fTime = ((float) frame) * invTicksPerFrame;
	newKey->pos[0] = pos[0];
	newKey->pos[1] = pos[1];
	newKey->pos[2] = pos[2];
	newKey->inTan[0] = inTan[0];
	newKey->inTan[1] = inTan[1];
	newKey->inTan[2] = inTan[2];
	newKey->outTan[0] = outTan[0];
	newKey->outTan[1] = outTan[1];
	newKey->outTan[2] = outTan[2];
	CKeyFrame *curr = PosKeyframes;
	if (!curr)
	{
		PosKeyframes = newKey;
		CurrPosKeyframe = PosKeyframes;
	}
	else
```

```
        {
                while (curr->next)
                        curr = curr->next;
                curr->next = newKey;
                newKey->prev = curr->next;
        }
        return TRUE;
}

BOOL CVisualObject::AddTCBPositionKey(int frame, float *pos,
                                                        float  tens,  float
cont, float bias,
                                                      float  easeIn,  float
easeOut,
                                                                      float
invTicksPerFrame)
{
        CKeyFrame *newKey = new CKeyFrame;
        newKey->Type = CNT_TCBPOS;
        newKey->iTime = frame;
        newKey->fTime = ((float) frame) * invTicksPerFrame;
        newKey->pos[0] = pos[0];
        newKey->pos[1] = pos[1];
        newKey->pos[2] = pos[2];
        newKey->tension = tens;
        newKey->continuity = cont;
        newKey->bias = bias;
        newKey->easeIn = easeIn;
        newKey->easeOut = easeOut;
        CKeyFrame *curr = PosKeyframes;
        if (!curr)
        {
                PosKeyframes = newKey;
                CurrPosKeyframe = PosKeyframes;
        }
        else
        {
                while (curr->next)
                        curr = curr->next;
                curr->next = newKey;
                newKey->prev = curr->next;
        }
        if (newKey->prev && newKey->prev->prev)
        {
                // if there are three nodes, calculate the inTan/outTan
                // values for the middle node

                // calculate the tangent vector
                float dsA = (1.0f - newKey->prev->tension) *
                        (1.0f     -     newKey->prev->continuity)     *     (1.0f     +
newKey->prev->bias);
                float dsB = (1.0f - newKey->prev->tension) *
                        (1.0f     +     newKey->prev->continuity)     *     (1.0f     -
newKey->prev->bias);
                float dsAdjust = (newKey->prev->fTime - newKey->prev->prev->fTime)
/
                        (newKey->prev->fTime - newKey->prev->prev->fTime);
```

```
            float ddA = (1.0f - newKey->prev->tension) *
                  (1.0f    +    newKey->prev->continuity)    *    (1.0f    +
newKey->prev->bias);
            float ddB = (1.0f - newKey->prev->tension) *
                  (1.0f    -    newKey->prev->continuity)    *    (1.0f    -
newKey->prev->bias);
            float ddAdjust = (newKey->fTime - newKey->prev->fTime) /
                  (newKey->prev->fTime - newKey->prev->prev->fTime);

            float v1,v2;
            v1 = newKey->prev->pos[0] - newKey->prev->prev->pos[0];
            v2 = newKey->pos[0] - newKey->prev->pos[0];
            newKey->prev->inTan[0] = dsAdjust*( dsA*v1 + dsB*v2 );
            newKey->prev->outTan[0] = dsAdjust*( ddA*v1 + ddB*v2 );

            v1 = newKey->prev->pos[1] - newKey->prev->prev->pos[1];
            v2 = newKey->pos[1] - newKey->prev->pos[1];
            newKey->prev->inTan[1] = dsAdjust*( dsA*v1 + dsB*v2 );
            newKey->prev->outTan[1] = dsAdjust*( ddA*v1 + ddB*v2 );

            v1 = newKey->prev->pos[2] - newKey->prev->prev->pos[2];
            v2 = newKey->pos[2] - newKey->prev->pos[2];
            newKey->prev->inTan[2] = dsAdjust*( dsA*v1 + dsB*v2 );
            newKey->prev->outTan[2] = dsAdjust*( ddA*v1 + ddB*v2 );

            if (!newKey->prev->prev->prev)
            {
                  // if there are only three nodes in the list,
                  // we know that new->p->p is the original first
                  // node.  Thus we can accurately calculate its
                  // value.

                  // a more accurate tangent value for the 1st node,
                  // that replaces the simpler calculation listed below
                  float f20,f10,v20,v10;
                  f20 = newKey->fTime - newKey->prev->prev->fTime;
                  f10 = newKey->prev->fTime - newKey->prev->prev->fTime;

                  v20 = newKey->pos[0] - newKey->prev->prev->pos[0];
                  v10 = newKey->prev->pos[0] - newKey->prev->prev->pos[0];
                  newKey->prev->prev->outTan[0]        =        (1.0f        -
newKey->prev->prev->tension) *
                        (v20*(0.25f - f10/(2*f20)) + (v10 - v20*0.5f)*1.5f +
v20*0.5f);

                  v20 = newKey->pos[1] - newKey->prev->prev->pos[1];
                  v10 = newKey->prev->pos[1] - newKey->prev->prev->pos[1];
                  newKey->prev->prev->outTan[1]        =        (1.0f        -
newKey->prev->prev->tension) *
                        (v20*(0.25f - f10/(2*f20)) + (v10 - v20*0.5f)*1.5f +
v20*0.5f);

                  v20 = newKey->pos[2] - newKey->prev->prev->pos[2];
                  v10 = newKey->prev->pos[2] - newKey->prev->prev->pos[2];
                  newKey->prev->prev->outTan[2]        =        (1.0f        -
newKey->prev->prev->tension) *
                        (v20*(0.25f - f10/(2*f20)) + (v10 - v20*0.5f)*1.5f +
v20*0.5f);
```

```
                    // Pretend the current key is the last key. if its not,
this'll be
                    // overwritten anyway.
                    float v;
                    v = newKey->pos[0] - newKey->prev->pos[0];
                    newKey->inTan[0] = v*(1.0f - newKey->tension);
                    v = newKey->pos[1] - newKey->prev->pos[1];
                    newKey->inTan[1] = v*(1.0f - newKey->tension);
                    v = newKey->pos[2] - newKey->prev->pos[2];
                    newKey->inTan[2] = v*(1.0f - newKey->tension);
              }
        }
        else if (newKey->prev)
        {
              // this is a case where we're dealing with the 2nd
              // Key in a list.. so we process the special case of
              // the beginning of the list here 'newKey->prev' and
              // the special case of last node in the list
              float v;
              v = newKey->pos[0] - newKey->prev->pos[0];
              newKey->prev->outTan[0] = v*(1.0f - newKey->prev->tension);
              newKey->inTan[0] = v*(1.0f - newKey->tension);

              v = newKey->pos[1] - newKey->prev->pos[1];
              newKey->prev->outTan[1] = v*(1.0f - newKey->prev->tension);
              newKey->inTan[1] = v*(1.0f - newKey->tension);

              v = newKey->pos[2] - newKey->prev->pos[2];
              newKey->prev->outTan[2] = v*(1.0f - newKey->prev->tension);
              newKey->inTan[2] = v*(1.0f - newKey->tension);
        }
        // otherwise we only have one node.. in which case there's no
        // need for tangents
        return TRUE;
}

BOOL    CVisualObject::AddBezierRotationKey(int    frame,    float    *rot,float
invTicksPerFrame)
{
        CKeyFrame *newKey = new CKeyFrame;
        newKey->Type = CNT_BEZROT;
        newKey->iTime = frame;
        newKey->fTime = ((float) frame) * invTicksPerFrame;
        newKey->rot[0] = rot[1];
        newKey->rot[1] = rot[2];
        newKey->rot[2] = rot[3];
        newKey->rot[3] = rot[0];
        CKeyFrame *curr = RotKeyframes;
        if (!curr)
        {
              RotKeyframes = newKey;
              CurrRotKeyframe = RotKeyframes;
        }
        else
        {
              while (curr->next)
                    curr = curr->next;
```

```
            curr->next = newKey;
            newKey->prev = curr->next;
      }
      return TRUE;
}

BOOL   CVisualObject::AddLinearRotationKey(int   frame,   float   *rot,float
invTicksPerFrame)
{
      CKeyFrame *newKey = new CKeyFrame;
      newKey->Type = CNT_LINROT;
      newKey->iTime = frame;
      newKey->fTime = ((float) frame) * invTicksPerFrame;
      newKey->rot[0] = rot[1];
      newKey->rot[1] = rot[2];
      newKey->rot[2] = rot[3];
      newKey->rot[3] = rot[0];

      CKeyFrame *curr = RotKeyframes;
      if (!curr)
      {
            RotKeyframes = newKey;
            CurrRotKeyframe = RotKeyframes;
      }
      else
      {
            while (curr->next)
                  curr = curr->next;
            curr->next = newKey;
            newKey->prev = curr->next;
      }
      return TRUE;
}

static void CompElementDeriv (float pp,  float p,   float pn,
                              float *ds, float *dd, float ksm,
                              float ksp, float kdm, float kdp)
{
  float delm, delp;

  delm = p - pp;
  delp = pn - p;
  *ds  = ksm*delm + ksp*delp;
  *dd  = kdm*delm + kdp*delp;
}

BOOL CVisualObject::AddTCBRotationKey(int frame, float *rot,
                                                    float  tens,  float
cont, float bias,
                                                   float easeIn,  float
easeOut,
                                                                  float
invTicksPerFrame)
{
      CKeyFrame *newKey = new CKeyFrame;
      newKey->Type = CNT_TCBROT;
      newKey->iTime = frame;
      newKey->fTime = ((float) frame) * invTicksPerFrame;
```

```
        newKey->angleaxis[0] = rot[1];
        newKey->angleaxis[1] = rot[2];
        newKey->angleaxis[2] = rot[3];
        newKey->angleaxis[3] = rot[0];

        float scalefactor = sinf(newKey->angleaxis[3]*0.5f);
        float tempquat[4];
        tempquat[0] = newKey->angleaxis[0]*scalefactor;
        tempquat[1] = newKey->angleaxis[1]*scalefactor;
        tempquat[2] = newKey->angleaxis[2]*scalefactor;
        tempquat[3] = cosf(newKey->angleaxis[3]*0.5f);

        newKey->tension = tens;
        newKey->bias = bias;
        newKey->continuity = cont;
        newKey->easeIn = easeIn;
        newKey->easeOut = easeOut;
        CKeyFrame *curr = RotKeyframes;
        if (!curr)
        {
              RotKeyframes = newKey;
              CurrRotKeyframe = RotKeyframes;
        }
        else
        {
              while (curr->next)
                    curr = curr->next;
              curr->next = newKey;
              newKey->prev = curr;
        }

        // Rotations used must be FINAL orientation, though only
        // relative rotations have been transmitted.  Since
        // quaternions refer to orientation positions and don't
        // easily account for the number of revolutions used
        // to achieve that orientation, we transmit AngleAxis
        // values rather than quaternions for TCB rots.

        // Thus, we previously converted the angle-axis value
        // into its relative quaternion equivilent (tempquat).
        // Once we've established the order or the current
        // rotation key, we now determine the true final local
        // orientation of this key by going backwards and
        // premultiplying consectutive older rotations.
        if (newKey->prev)
              QuaternionMultiply(tempquat,newKey->prev->rot,newKey->rot);
        else
        {
              newKey->rot[0] = tempquat[0]; newKey->rot[1] = tempquat[1];
              newKey->rot[2] = tempquat[2]; newKey->rot[3] = tempquat[3];
        }

        if (newKey->prev && newKey->prev->prev)
              // Calculate the quaternion derivatives for the middle frame of
the
              // known existing three frames, with the current frame being the
              // last of the three.
              CalculateTCBQuatDeriv(newKey->prev->prev,newKey->prev,newKey);
```

```
        else if (newKey->prev)
                // We know this is the second frame in the series. thus, we
                // calculate the values for the first frame
                CalculateTCBQuatDeriv(NULL,newKey->prev,newKey);

        // pretend the current key added is the last
        if (newKey->prev)
                CalculateTCBQuatDeriv(newKey->prev,newKey,NULL);
        return TRUE;
}

BOOL CVisualObject::AddTCBEnhRotationKey(int frame, short count,
                                                                float   angle,
float *axis,
                                                        float tens, float
cont, float bias,
                                                          float   easeIn,
float easeOut,
                                                                  float
invTicksPerFrame)
{
// THIS FUNCTION IS CURRENTLY NOT BEING USED

        // with angles over 180', we need to invoke a special case solution
        // and build a series of intermediate keys to interpolate thru (every
        // 90').  The reason behind this is that unlike raw angles, which can
        // specify multiple resolutions, unit quaternions are limited in that
        // they specify an absolute position on a hypersphere, not a relative
        // positioning allowing for multiple revolutions.  This is especially
        // true when they're created by converting from an angle axis
        // specification due to the calculations involved.
        // So what we do is if the angle is >= 180', we know that we'll end
        // up rotating the wrong direction if we simply try to slerp to the new
        // orientation and we're guaranteed to not account for multiple
        // revolutions.  So to get around this, we split the rotation into a
        // group of successive rotations of 90', spaced in time so they
        // correspond to the angle amount.
        float mult; float sign = 1.0f;
        if (count < 0)
        {
                mult=FLOAT_HALFPI;
                count = -count;
        }
        else
        {
                mult =-FLOAT_HALFPI;
                sign =-sign;
        }
        CKeyFrame *lastKey = NULL;

        float lastQuat[4] =
        {
                0.0f, 0.0f, 0.0f, 1.0f,
        };

        float RotQuat[4];
```

```
	CKeyFrame *curr = RotKeyframes;
	if (curr)
	{
		while (curr)
		{
			lastKey = curr;
			curr = curr->next;
		}

		CKeyFrame *OrigKey = lastKey;
		float oldtime = lastKey->fTime;
		float newtime = ((float) frame) * invTicksPerFrame;

		float timesumrecip = 1.0f / (fabsf(angle) + FLOAT_HALFPI *
(float)count);
		float diff = fabsf((newtime - oldtime)*timesumrecip);

		ConvertAAtoQuat(mult,axis,RotQuat);

		for (int i = 0;i<count;i++)
		{
			CKeyFrame *newKey = new CKeyFrame;
			newKey->Type = CNT_TCBROT;
			newKey->fTime = oldtime + (FLOAT_HALFPI*(float)(i+1)) *
diff;

			// these 5 values may need to be adjusted
			newKey->tension = tens;
			newKey->bias = bias;
			newKey->continuity = cont;
			newKey->easeIn = easeIn;
			newKey->easeOut = easeOut;

			lastQuat[0] = lastKey->rot[0];
			lastQuat[1] = lastKey->rot[1];
			lastQuat[2] = lastKey->rot[2];
			lastQuat[3] = lastKey->rot[3];

			QuaternionMultiply(RotQuat,lastQuat,newKey->rot);

			lastKey->next = newKey;
			newKey->prev = lastKey;
			lastKey = newKey;
		}
		ConvertAAtoQuat(angle*sign,axis,RotQuat);
		CKeyFrame *newKey = new CKeyFrame;
		newKey->Type = CNT_TCBROT;
		newKey->iTime = frame;
		newKey->fTime = ((float) frame) * invTicksPerFrame;
		newKey->tension = tens;
		newKey->bias = bias;
		newKey->continuity = cont;
		newKey->easeIn = easeIn;
		newKey->easeOut = easeOut;
		lastQuat[0] = lastKey->rot[0];
		lastQuat[1] = lastKey->rot[1];
		lastQuat[2] = lastKey->rot[2];
```

```
            lastQuat[3] = lastKey->rot[3];
            QuaternionMultiply(lastQuat,RotQuat,newKey->rot);
            lastKey->next = newKey;
            newKey->prev = lastKey;
      }
      else
      {
            // this is the original keyframe... so we can basically just
straight convert
            // the axis angle value to a quaternion
            CKeyFrame *newKey = new CKeyFrame;
            newKey->Type = CNT_TCBROT;
            newKey->iTime = frame;
            newKey->fTime = ((float) frame) * invTicksPerFrame;
            newKey->tension = tens;
            newKey->bias = bias;
            newKey->continuity = cont;
            newKey->easeIn = easeIn;
            newKey->easeOut = easeOut;

            ConvertAAtoQuat(angle*sign+count*mult,axis,RotQuat);
/*          if (curr)
            {
                  while (curr)
                  {
                        lastKey = curr;
                        curr = curr->next;
                  }
                  lastQuat[0] = lastKey->rot[0];
                  lastQuat[1] = lastKey->rot[1];
                  lastQuat[2] = lastKey->rot[2];
                  lastQuat[3] = lastKey->rot[3];
                  lastKey->next = newKey;
            }*/
            QuaternionMultiply(lastQuat,RotQuat,newKey->rot);
            newKey->prev = lastKey;
            RotKeyframes = newKey;
            CurrRotKeyframe = newKey;
      }
      return TRUE;
}

CVisualObject::AddBezierRollKey(int frame, float roll, float invTicksPerFrame)
{
      CKeyFrame *newKey = new CKeyFrame;
      newKey->Type = CNT_BEZRLL;
      newKey->iTime = frame;
      newKey->fTime = ((float) frame) * invTicksPerFrame;
      newKey->roll = roll;
      CKeyFrame *curr = RollKeyframes;
      if (!curr)
      {
            RollKeyframes = newKey;
            CurrRollKeyframe = RollKeyframes;
      }
      else
      {
            while (curr->next)
```

```
                curr = curr->next;
            curr->next = newKey;
            newKey->prev = curr->next;
        }
        return TRUE;
}

BOOL CVisualObject::AddLinearScaleKey(int frame, float x,
                                                    float y, float z,
                                                                float
invTicksPerFrame)
{
        CKeyFrame *newKey = new CKeyFrame;
        newKey->Type = CNT_LINSCL;
        newKey->iTime = frame;
        newKey->fTime = ((float) frame) * invTicksPerFrame;
        newKey->scale[0] = x;
        newKey->scale[1] = y;
        newKey->scale[2] = z;
        CKeyFrame *curr = SclKeyframes;
        if (!curr)
        {
            SclKeyframes = newKey;
            CurrSclKeyframe = SclKeyframes;
        }
        else
        {
            while (curr->next)
                curr = curr->next;
            curr->next = newKey;
            newKey->prev = curr->next;
        }
        return TRUE;
}

BOOL CVisualObject::AddBezierScaleKey(int frame, float *scale,
                                                  float  *inTan,  float
*outTan,
                                                                float
invTicksPerFrame)
{
        CKeyFrame *newKey = new CKeyFrame;
        newKey->Type = CNT_BEZSCL;
        newKey->iTime = frame;
        newKey->fTime = ((float) frame) * invTicksPerFrame;
        newKey->scale[0] = scale[0];
        newKey->scale[1] = scale[1];
        newKey->scale[2] = scale[2];
        newKey->inTan[0] = inTan[0] * 30.f;
        newKey->inTan[1] = inTan[1] * 30.f;
        newKey->inTan[2] = inTan[2] * 30.f;
        newKey->outTan[0] = outTan[0] * 30.f;
        newKey->outTan[1] = outTan[1] * 30.f;
        newKey->outTan[2] = outTan[2] * 30.f;
        CKeyFrame *curr = SclKeyframes;
        if (!curr)
        {
            SclKeyframes = newKey;
```

```
            CurrSclKeyframe = SclKeyframes;
        }
        else
        {
            while (curr->next)
                curr = curr->next;
            curr->next = newKey;
            newKey->prev = curr->next;
        }
        return TRUE;
}

BOOL CVisualObject::AddTCBScaleKey(int frame, float *scale,
                                   float tens, float cont,
float bias,
                                   float easeIn, float
easeOut, float invTicksPerFrame)
{
        CKeyFrame *newKey = new CKeyFrame;
        newKey->Type = CNT_TCBSCL;
        newKey->iTime = frame;
        newKey->fTime = ((float) frame) * invTicksPerFrame;
        newKey->scale[0] = scale[0];
        newKey->scale[1] = scale[1];
        newKey->scale[2] = scale[2];
        newKey->tension = tens;
        newKey->continuity = cont;
        newKey->bias = bias;
        newKey->easeIn = easeIn;
        newKey->easeOut = easeOut;
        CKeyFrame *curr = SclKeyframes;
        if (!curr)
        {
            SclKeyframes = newKey;
            CurrSclKeyframe = SclKeyframes;
        }
        else
        {
            while (curr->next)
                curr = curr->next;
            curr->next = newKey;
            newKey->prev = curr->next;
        }
        return TRUE;
}

CVisualObject::AssignSolidColor(unsigned char red, unsigned char green,
unsigned char blue)
{
        bSolidColor = TRUE;
        solidRed = red;
        solidGreen = green;
        solidBlue = blue;
}


CCamera::CCamera() : CVisualObject()
{
```

```
	fFarPlane = 1000.0f;
	fNearPlane = 1.0f;
	fFieldOfView = 90.0f;
	Roll = 0.0f;
}

CVisualObject::SetPosition(float x, float y, float z)
{
	PosX = x; PosY = y; PosZ = z;
}

CVisualObject::SetOrientation(float *quaternion)
{
	OriX = quaternion[0];
	OriY = quaternion[1];
	OriZ = quaternion[2];
	OriW = quaternion[3];
}

CVisualObject::SetScale(float x, float y, float z)
{
	SclX = x; SclY = y; SclZ = z;
}

CVisualObject::Reset()
// reset's time dependant data in this object
{
	CurrRotKeyframe  = RotKeyframes;
	CurrPosKeyframe  = PosKeyframes;
	CurrRollKeyframe = RollKeyframes;
}

CVisualObject::GenerateVertexNormals()
{
	static int ContainingFaces[10];
	for (int i=0;i<TotalVertices*3;i++)
		NormalArray[i] = 0.0f;
	for (i=0;i<TotalFaces;i++)
	{
		float Normal[3];
		// calculate normal
		float Vec0[3] =
		{
			VertexArray[FaceArray[i*3+1]*3]                    -
VertexArray[FaceArray[i*3]*3],
			VertexArray[FaceArray[i*3+1]*3+1]                  -
VertexArray[FaceArray[i*3]*3+1],
			VertexArray[FaceArray[i*3+1]*3+2]                  -
VertexArray[FaceArray[i*3]*3+2]
		};
		float Vec1[3] =
		{
			VertexArray[FaceArray[i*3+1]*3]                    -
VertexArray[FaceArray[i*3+2]*3],
			VertexArray[FaceArray[i*3+1]*3+1]                  -
VertexArray[FaceArray[i*3+2]*3+1],
			VertexArray[FaceArray[i*3+1]*3+2]                  -
VertexArray[FaceArray[i*3+2]*3+2],
		};
```

```
            VectorCrossProduct(Vec0,Vec1,Normal);
            VectorNormalize(Normal);

            NormalArray[FaceArray[i*3  ]*3]   += Normal[0];
            NormalArray[FaceArray[i*3+1]*3]   += Normal[0];
            NormalArray[FaceArray[i*3+2]*3]   += Normal[0];

            NormalArray[FaceArray[i*3  ]*3+1] += Normal[1];
            NormalArray[FaceArray[i*3+1]*3+1] += Normal[1];
            NormalArray[FaceArray[i*3+2]*3+1] += Normal[1];

            NormalArray[FaceArray[i*3  ]*3+2] += Normal[2];
            NormalArray[FaceArray[i*3+1]*3+2] += Normal[2];
            NormalArray[FaceArray[i*3+2]*3+2] += Normal[2];
      }
      for (i=0;i<TotalVertices;i++)
            VectorNormalize(NormalArray+i*3);
      CurrentTotalNormals = TotalVertices;
}

BOOL CVisualObject::WithinFustrum(float *BackTransform, float zNear, float
zFar)
// check if the object's bounding box is within the view fustrum
{
      return TRUE;
      float FinalTransform[16];
      MatrixMultiply(BackTransform,Transform,FinalTransform);
      MatrixMultiply(Transform,BackTransform,FinalTransform);

/*
      // multiply the bounding box by the modelview matrix
      // since the bounding box is no longer axis aligned, generate a
      // new axis aligned bounding box
      float p[8][3];
      p[0][0] = p[2][0] = p[4][0] = p[6][0] = GetMinX();
      p[1][0] = p[3][0] = p[5][0] = p[7][0] = GetMaxX();

      p[0][1] = p[1][1] = p[4][1] = p[5][1] = GetMinY();
      p[2][1] = p[3][1] = p[6][1] = p[7][1] = GetMaxY();

      p[0][2] = p[1][2] = p[2][2] = p[3][2] = GetMinZ();
      p[4][2] = p[5][2] = p[6][2] = p[7][2] = GetMaxZ();

      float r[8][3];
      for (int i=0;i<8;i++)
            VecMatrixMultiply(p[i],FinalTransform,r[i]);
*/
      // this uses a bounding sphere technique.

      // first, let's get the radius of the sphere
      float radius;
      float r1 = GetMaxX() * GetMaxX() + GetMaxY() * GetMaxY() +
            GetMaxZ() * GetMaxZ();
      float r2 = GetMinX() * GetMinX() + GetMinY() * GetMinY() +
            GetMinZ() * GetMinZ();
      if (r1 > r2)
            radius = sqrtf(r1);
```

```
      else radius = sqrtf(r2);

      float o[3] = {0.0f,0.0f,0.0f},
                  p[3];
      VecMatrixMultiply(o,FinalTransform,p);
      if (p[2] < p[0] - 1.4142136*radius ||
            p[2] <-p[0] - 1.4142136*radius ||
            p[2] < p[1] - 1.4142136*radius ||
            p[2] <-p[1] - 1.4142136*radius ||
            p[2] <-radius + zNear ||
            p[2] > radius + zFar)
            return FALSE;
      // clip the bounding box against the fustrum. if it is entirely
      // outside the fustrum, return FALSE
      return TRUE;
}



CVisualObject::UpdatePosition(float fTotalTime)
{
      while (CurrPosKeyframe && CurrPosKeyframe->next &&
               CurrPosKeyframe->next->fTime < fTotalTime &&
               CurrPosKeyframe->next->next)
            CurrPosKeyframe = CurrPosKeyframe->next;
      if (CurrPosKeyframe && CurrPosKeyframe->next &&
            CurrPosKeyframe->next->fTime <= fTotalTime)
            // if we get here, we can assume that we're in a special case of
            // being at the end of the keyframe list for this object where
'next'
            // is the last keyframe.   Thus, need to lock to the final
position
      {
            PosX = CurrPosKeyframe->next->pos[0];
            PosY = CurrPosKeyframe->next->pos[1];
            PosZ = CurrPosKeyframe->next->pos[2];
      }
      else
      {
            if (CurrPosKeyframe && !CurrPosKeyframe->next)
            {
                  PosX = CurrPosKeyframe->pos[0];
                  PosY = CurrPosKeyframe->pos[1];
                  PosZ = CurrPosKeyframe->pos[2];
            }
            else if (CurrPosKeyframe && CurrPosKeyframe->fTime <= fTotalTime
&&
                  CurrPosKeyframe->next->fTime >= fTotalTime)
            // first ensure that we're interpolating between the proper
frames
            {
                  // now switch based on which type of controller we're using
                  switch (CurrPosKeyframe->Type)
                  {
                        case CNT_LINPOS:
                        {
                              float  dif1  =  CurrPosKeyframe->next->fTime  -
```

```
CurrPosKeyframe->fTime;
                              float       dif2       =       fTotalTime       -
CurrPosKeyframe->fTime;
                              float perc = dif2 / dif1;
                              PosX = (CurrPosKeyframe->next->pos[0] -
                                    CurrPosKeyframe->pos[0]) * perc +
                                    CurrPosKeyframe->pos[0];
                              PosY = (CurrPosKeyframe->next->pos[1] -
                                    CurrPosKeyframe->pos[1]) * perc +
                                    CurrPosKeyframe->pos[1];
                              PosZ = (CurrPosKeyframe->next->pos[2] -
                                          CurrPosKeyframe->pos[2]) * perc +
                                          CurrPosKeyframe->pos[2];
                        } break;
                        case CNT_TCBPOS:
                        case CNT_BEZPOS:
                        {
/*
      X(t) =       ( 2t3 - 3t2 + 1)*P1x +
                   (-2t3 + 3t2    )*P4x +
                   (  t3 - 2t2 + t)*R1x +
                   (  t3 - t2     )*R4x
*************

****  Easily derived from Bezier and Bezier derivative def'n  ****

      Jon's Note : See page 353-356 of Watt & Watt

  or better yet

      Reference: Doris Kuchanek and Richard Bartels,
      "Interpolating  Splines  with  Local  Tension,  Continuity,  and  Bias
Control",
      Computer Graphics 18:3, pp. 33-41, July 1984 (SIGGRAPH 84)

*/
                              float dif1 = CurrPosKeyframe->next->fTime -
                                    CurrPosKeyframe->fTime;
                              float       dif2       =       fTotalTime       -
CurrPosKeyframe->fTime;
                              float perc = dif2 / dif1;
                              float                  newTime                  =
Ease(perc,CurrPosKeyframe->easeOut,
                                    CurrPosKeyframe->next->easeIn);
                              float  sqr  =  newTime*newTime;  float  cube  =
sqr*newTime;

                              PosX =
                                    (     2.0f*cube     -     3.0f*sqr     +
1.0f)*CurrPosKeyframe->pos[0] +
                                    (-2.0f*cube           +           3.0f*sqr
)*CurrPosKeyframe->next->pos[0] +
                                    (cube         -         2.0f*sqr         +
newTime)*CurrPosKeyframe->outTan[0] +
                                    (cube - sqr
)*CurrPosKeyframe->next->inTan[0];
                              PosY =
```

```
                                    (      2.0f*cube      -      3.0f*sqr      +
1.0f)*CurrPosKeyframe->pos[1] +
                                    (-2.0f*cube            +            3.0f*sqr
)*CurrPosKeyframe->next->pos[1] +
                                    (cube          -          2.0f*sqr          +
newTime)*CurrPosKeyframe->outTan[1] +
                                    (cube - sqr
)*CurrPosKeyframe->next->inTan[1];
                              PosZ =
                                    (      2.0f*cube      -      3.0f*sqr      +
1.0f)*CurrPosKeyframe->pos[2] +
                                    (-2.0f*cube            +            3.0f*sqr
)*CurrPosKeyframe->next->pos[2] +
                                    (cube          -          2.0f*sqr          +
newTime)*CurrPosKeyframe->outTan[2] +
                                    (cube - sqr
)*CurrPosKeyframe->next->inTan[2];
                        } break;
                  }
            }
            else if (PosKeyframes && CurrPosKeyframe == PosKeyframes &&
                        CurrPosKeyframe->fTime > fTotalTime)
            {
                  PosX = CurrPosKeyframe->pos[0];
                  PosY = CurrPosKeyframe->pos[1];
                  PosZ = CurrPosKeyframe->pos[2];
            }
      }
}

CVisualObject::UpdateRotation(float fTotalTime)
{
      while (CurrRotKeyframe && CurrRotKeyframe->next &&
                        CurrRotKeyframe->next->fTime    <    fTotalTime    &&
CurrRotKeyframe->next->next)
                  CurrRotKeyframe = CurrRotKeyframe->next;
      if (CurrRotKeyframe && CurrRotKeyframe->next &&
            CurrRotKeyframe->next->fTime <= fTotalTime)
            // if we get here, we can assume that we're in a special case of
            // being at the end of the keyframe list for this object where
'next'
            // is the last keyframe.   Thus, need to lock to the final
position
      {
            OriX = CurrRotKeyframe->next->rot[0];
            OriY = CurrRotKeyframe->next->rot[1];
            OriZ = CurrRotKeyframe->next->rot[2];
            OriW = CurrRotKeyframe->next->rot[3];
      }
      else
      {
            if (CurrRotKeyframe && !CurrRotKeyframe->next)
            {
                  OriX = CurrRotKeyframe->rot[0];
                  OriY = CurrRotKeyframe->rot[1];
                  OriZ = CurrRotKeyframe->rot[2];
                  OriW = CurrRotKeyframe->rot[3];
            }
```

```
            else if (CurrRotKeyframe &&
                  CurrRotKeyframe->fTime <= fTotalTime &&
                  CurrRotKeyframe->next->fTime >= fTotalTime)
                  // first ensure that we're interpolating between the proper
frames
            {
                  float rot[4],inTan[4];
                  rot[0] = CurrRotKeyframe->next->rot[0];
                  rot[1] = CurrRotKeyframe->next->rot[1];
                  rot[2] = CurrRotKeyframe->next->rot[2];
                  rot[3] = CurrRotKeyframe->next->rot[3];
                  inTan[0] = CurrRotKeyframe->next->inTan[0];
                  inTan[1] = CurrRotKeyframe->next->inTan[1];
                  inTan[2] = CurrRotKeyframe->next->inTan[2];
                  inTan[3] = CurrRotKeyframe->next->inTan[3];

                  // now switch based on which type of controller we're using
                  switch (CurrRotKeyframe->Type)
                  {
                  case CNT_LINROT:
                  case CNT_BEZROT:
                        {
                              float dot0 =
                                    (CurrRotKeyframe->next->rot[0]          -
CurrRotKeyframe->rot[0])*
                                    (CurrRotKeyframe->next->rot[0]          -
CurrRotKeyframe->rot[0]) +
                                    (CurrRotKeyframe->next->rot[1]          -
CurrRotKeyframe->rot[1])*
                                    (CurrRotKeyframe->next->rot[1]          -
CurrRotKeyframe->rot[1]) +
                                    (CurrRotKeyframe->next->rot[2]          -
CurrRotKeyframe->rot[2])*
                                    (CurrRotKeyframe->next->rot[2]          -
CurrRotKeyframe->rot[2]) +
                                    (CurrRotKeyframe->next->rot[3]          -
CurrRotKeyframe->rot[3])*
                                    (CurrRotKeyframe->next->rot[3]          -
CurrRotKeyframe->rot[3]);
                              float dot1 =
                                    (CurrRotKeyframe->next->rot[0]          +
CurrRotKeyframe->rot[0])*
                                    (CurrRotKeyframe->next->rot[0]          +
CurrRotKeyframe->rot[0]) +
                                    (CurrRotKeyframe->next->rot[1]          +
CurrRotKeyframe->rot[1])*
                                    (CurrRotKeyframe->next->rot[1]          +
CurrRotKeyframe->rot[1]) +
                                    (CurrRotKeyframe->next->rot[2]          +
CurrRotKeyframe->rot[2])*
                                    (CurrRotKeyframe->next->rot[2]          +
CurrRotKeyframe->rot[2]) +
                                    (CurrRotKeyframe->next->rot[3]          +
CurrRotKeyframe->rot[3])*
                                    (CurrRotKeyframe->next->rot[3]          +
CurrRotKeyframe->rot[3]);
                              if (dot1 < dot0)
                              {
                                    rot[0] = -rot[0];
```

```
                                    rot[1] = -rot[1];
                                    rot[2] = -rot[2];
                                    rot[3] = -rot[3];

                                    inTan[0] = -inTan[0];
                                    inTan[1] = -inTan[1];
                                    inTan[2] = -inTan[2];
                                    inTan[3] = -inTan[3];
                              }
                              // this makes sure that we're rotating in the
right
                              // direction around the circle by following the
                              // shorter path
                              float dif1 = CurrRotKeyframe->next->fTime -
                                    CurrRotKeyframe->fTime;
                              float      dif2      =      fTotalTime      -
CurrRotKeyframe->fTime;
                              float perc = dif2 / dif1;
                        // slerp between start and finish
                              QuaternionSlerp(perc,0,CurrRotKeyframe->rot,
                                                rot,OriX,OriY,OriZ,OriW)
;
                        } break;
                  case CNT_TCBROT:
                        {
                              float dif1 = CurrRotKeyframe->next->fTime -
                                    CurrRotKeyframe->fTime;
                              float      dif2      =      fTotalTime      -
CurrRotKeyframe->fTime;
                              float perc = dif2 / dif1;
                              float                newTime                =
Ease(perc,CurrRotKeyframe->easeOut,
                                    CurrRotKeyframe->next->easeIn);

                              float                 angle                 =
(CurrRotKeyframe->next->angleaxis[3] -
                                    CurrRotKeyframe->angleaxis[3]);
                              float spin;
                              if (angle > 0.0f)
                                    spin = floorf( angle/(2.0f*FLOAT_PI) );
                              else
                                    spin = ceilf( angle/(2.0f*FLOAT_PI) );
//                            angle = angle - (2.0f*FLOAT_PI)*spin;

                        // slerp between start and finish
                              float p[4],q[4],finaltime;

                              QuaternionSlerp(newTime,spin,CurrRotKeyframe->ro
t,
                                                rot,p[0],p[1],p[2],p[3])
;
                              QuaternionSlerp(newTime,spin,CurrRotKeyframe->ou
tTan,
                                                inTan,q[0],q[1],q[2],q[3
]);
                              finaltime = (((1.0f - newTime)*2.0f)*newTime);
                              QuaternionSlerp(finaltime,0,p,q,OriX,OriY,OriZ,O
riW);
```

```
                        } break;
                    }
                }
                else if (RotKeyframes && CurrRotKeyframe == RotKeyframes &&
                        CurrRotKeyframe->fTime > fTotalTime)
                {
                    OriX = CurrRotKeyframe->rot[0];
                    OriY = CurrRotKeyframe->rot[1];
                    OriZ = CurrRotKeyframe->rot[2];
                    OriW = CurrRotKeyframe->rot[3];
                }
        }
}

CVisualObject::UpdateRoll(float fTotalTime)
{
        // process rolling keyframes
        while (CurrRollKeyframe && CurrRollKeyframe->next &&
                CurrRollKeyframe->next->fTime < fTotalTime &&
                CurrRollKeyframe->next->next)
            CurrRollKeyframe = CurrRollKeyframe->next;
        if (CurrRollKeyframe && CurrRollKeyframe->next &&
            CurrRollKeyframe->next->fTime <= fTotalTime)
            // if we get here, we can assume that we're in a special case of
            // being at the end of the keyframe list for this object where
'next'
            // is the last keyframe.  Thus, need to lock to the final
position
            Roll = CurrRollKeyframe->next->roll;
        else
        {
            if (CurrRollKeyframe && !CurrRollKeyframe->next)
                Roll = CurrRollKeyframe->roll;
            else if (CurrRollKeyframe && CurrRollKeyframe->fTime <= fTotalTime
&&
                    CurrRollKeyframe->next->fTime >= fTotalTime)
                    // first ensure that we're interpolating between the proper
frames
            {
                    // now switch based on which type of controller we're using
                    switch (CurrRollKeyframe->Type)
                    {
                        case CNT_LINRLL:
                        {
                            float dif1 = CurrRollKeyframe->next->fTime -
                                CurrRollKeyframe->fTime;
                            float dif2 = fTotalTime -
CurrRollKeyframe->fTime;
                            float perc = dif2 / dif1;

                            Roll = (CurrRollKeyframe->next->roll -
                                CurrRollKeyframe->roll) * perc +
                                CurrRollKeyframe->roll;
                        } break;
                        case CNT_TCBRLL:
                        {
                            float dif1 = CurrRollKeyframe->next->fTime -
```

```
                                    CurrRollKeyframe->fTime;
                              float       dif2       =       fTotalTime       -
CurrRollKeyframe->fTime;
                              float perc = dif2 / dif1;
                              float                  newTime                  =
Ease(perc,CurrRollKeyframe->easeOut,
                                    CurrRollKeyframe->next->easeIn);
                              float  sqr  =  newTime*newTime;  float  cube  =
sqr*newTime;

                              Roll =
                                    (      2.0f*cube      -      3.0f*sqr      +
1.0f)*CurrRollKeyframe->roll +
                                    (-2.0f*cube           +           3.0f*sqr
)*CurrRollKeyframe->next->roll +
                                    (cube          -          2.0f*sqr          +
newTime)*CurrRollKeyframe->outTan[0] +
                                    (cube - sqr
)*CurrRollKeyframe->next->inTan[0];
                        } break;
                        case CNT_BEZRLL:
                        {
                              float dif1 = CurrRollKeyframe->next->fTime -
                                    CurrRollKeyframe->fTime;
                              float       dif2       =       fTotalTime       -
CurrRollKeyframe->fTime;
                              float perc = dif2 / dif1;
                              float sqr = perc*perc; float cube = sqr*perc;
                              Roll =
                                    (      2.0f*cube      -      3.0f*sqr      +
1.0f)*CurrRollKeyframe->roll +
                                    (-2.0f*cube           +           3.0f*sqr
)*CurrRollKeyframe->next->roll -
                                    (cube          -          2.0f*sqr          +
perc)*CurrRollKeyframe->outTan[0] -
                                    (cube - sqr
)*CurrRollKeyframe->next->inTan[0];
                        } break;
                  }
            }
            else if (RollKeyframes && CurrRollKeyframe == RollKeyframes &&
                        CurrRollKeyframe->fTime > fTotalTime)
                  Roll = CurrRollKeyframe->roll;
      }
}

CVisualObject::UpdateScale(float fTotalTime)
{
      // process scaling keyframes
      while (CurrSclKeyframe && CurrSclKeyframe->next &&
               CurrSclKeyframe->next->fTime < fTotalTime &&
               CurrSclKeyframe->next->next)
         CurrSclKeyframe = CurrSclKeyframe->next;
      if (CurrSclKeyframe && CurrSclKeyframe->next &&
            CurrSclKeyframe->next->fTime <= fTotalTime)
            // if we get here, we can assume that we're in a special case of
            // being at the end of the keyframe list for this object where
'next'
```

```
            // is the last keyframe.  Thus, need to lock to the final
position
      {
            SclX = CurrSclKeyframe->next->scale[0];
            SclY = CurrSclKeyframe->next->scale[1];
            SclZ = CurrSclKeyframe->next->scale[2];
      }
      else
      {
            if (CurrSclKeyframe && !CurrSclKeyframe->next)
            {
                  SclX = CurrSclKeyframe->scale[0];
                  SclY = CurrSclKeyframe->scale[1];
                  SclZ = CurrSclKeyframe->scale[2];
            }
            else if (CurrSclKeyframe && CurrSclKeyframe->fTime <= fTotalTime
&&
                  CurrSclKeyframe->next->fTime >= fTotalTime)
                  // first ensure that we're interpolating between the proper
frames
            {
                  // now switch based on which type of controller we're using
                  switch (CurrSclKeyframe->Type)
                  {
                        case CNT_LINSCL:
                        {
                              float dif1 = CurrSclKeyframe->next->fTime -
                                    CurrSclKeyframe->fTime;
                              float       dif2       =       fTotalTime       -
CurrSclKeyframe->fTime;
                              float perc = dif2 / dif1;

                              SclX = (CurrSclKeyframe->next->scale[0] -
                                    CurrSclKeyframe->scale[0]) * perc +
                                    CurrSclKeyframe->scale[0];
                              SclY = (CurrSclKeyframe->next->scale[1] -
                                    CurrSclKeyframe->scale[1]) * perc +
                                    CurrSclKeyframe->scale[1];
                              SclZ = (CurrSclKeyframe->next->scale[2] -
                                    CurrSclKeyframe->scale[2]) * perc +
                                    CurrSclKeyframe->scale[2];
                        } break;
                        case CNT_TCBSCL:
                        {
                              SclX = CurrSclKeyframe->scale[0];
                              SclY = CurrSclKeyframe->scale[1];
                              SclZ = CurrSclKeyframe->scale[2];
                        } break;
                        case CNT_BEZSCL:
                        {
                              float dif1 = CurrSclKeyframe->next->fTime -
                                    CurrSclKeyframe->fTime;
                              float       dif2       =       fTotalTime       -
CurrSclKeyframe->fTime;
                              float perc = dif2 / dif1;
                              float sqr = perc*perc; float cube = sqr*perc;
                              SclX =
                                    (     2.0f*cube      -      3.0f*sqr      +
```

```
1.0f)*CurrSclKeyframe->scale[0] +
                                    (-2.0f*cube          +          3.0f*sqr
)*CurrSclKeyframe->next->scale[0] +
                                    (cube          -          2.0f*sqr          +
perc)*CurrSclKeyframe->outTan[0] +
                                    (cube - sqr
)*CurrSclKeyframe->next->inTan[0];
                              SclY =
                                    (      2.0f*cube      -      3.0f*sqr      +
1.0f)*CurrSclKeyframe->scale[1] +
                                    (-2.0f*cube          +          3.0f*sqr
)*CurrSclKeyframe->next->scale[1] +
                                    (cube          -          2.0f*sqr          +
perc)*CurrSclKeyframe->outTan[1] +
                                    (cube - sqr
)*CurrSclKeyframe->next->inTan[1];
                              SclZ =
                                    (      2.0f*cube      -      3.0f*sqr      +
1.0f)*CurrSclKeyframe->scale[2] +
                                    (-2.0f*cube          +          3.0f*sqr
)*CurrSclKeyframe->next->scale[2] -
                                    (cube          -          2.0f*sqr          +
perc)*CurrSclKeyframe->outTan[2] -
                                    (cube - sqr
)*CurrSclKeyframe->next->inTan[2];
                        } break;
                  }
            }
            else if (SclKeyframes && CurrSclKeyframe == SclKeyframes &&
                        CurrSclKeyframe->fTime > fTotalTime)
            {
                  SclX = CurrSclKeyframe->scale[0];
                  SclY = CurrSclKeyframe->scale[1];
                  SclZ = CurrSclKeyframe->scale[2];
            }
      }
}


CVisualObject::UpdateTransform(float CurTime)
{
      // this allows us to recursively update the transformation matrices of
all
      // objects in the database
      if (CurTime != TimeLastUpdated)
      {
            UpdatePosition(CurTime);
            UpdateRotation(CurTime);
            UpdateScale(CurTime);

            float TransformationMatrix[16];

            float LocalSclMatrix[16] =
            {     SclX, 0.0f, 0.0f, 0.0f,
                  0.0f, SclY, 0.0f, 0.0f,
                  0.0f, 0.0f, SclZ, 0.0f,
                  0.0f, 0.0f, 0.0f, 1.0f,
            };
            float LocalRotMatrix[16] =
```

```
            {     1.0f, 0.0f, 0.0f, 0.0f,
                  0.0f, 1.0f, 0.0f, 0.0f,
                  0.0f, 0.0f, 1.0f, 0.0f,
                  0.0f, 0.0f, 0.0f, 1.0f,
            };
            float LocalTrnMatrix[16] =
            {     1.0f, 0.0f, 0.0f, 0.0f,
                  0.0f, 1.0f, 0.0f, 0.0f,
                  0.0f, 0.0f, 1.0f, 0.0f,
                  PosX, PosY, PosZ, 1.0f,
            };
            QuaternionToMatrix(OriW,OriX,OriY,OriZ,LocalRotMatrix);
            static float TempMatrix1[16],TempMatrix2[16],TempMatrix3[16];
            MatrixMultiply(LocalSclMatrix,LocalRotMatrix,TempMatrix1);
                  MatrixMultiply(TempMatrix1,LocalTrnMatrix,Transform);
//          MatrixMultiply(TempMatrix1,LocalTrnMatrix,TempMatrix2);
//          MatrixMultiply(OffsetMatrix,TempMatrix2,Transform);
      //          MatrixMultiply(TempMatrix2,OffsetMatrix,Transform);
            if (ParentObject)
            {
                  ParentObject->UpdateTransform(CurTime);
                  MatrixMultiply(Transform,ParentObject->Transform,Transformat
ionMatrix);
//                MatrixMultiply(ParentObject->Transform,TempMatrix2,Transform
ationMatrix);
                  MatrixCopy(TransformationMatrix,Transform);
            }

            if (Target)
            {
                  Target->UpdateTransform(CurTime);
                  // here we replace the rotation portion of the
transformation
                  // matrix with a targetting one.
                  float vector[3] =
                  {
                        Target->PosX - PosX,
                        Target->PosY - PosY,
                        Target->PosZ - PosZ,
                  };
                  VectorNormalize(vector);

                  // now we construct the B Matrix as listed on Page 9 of
Watt & Watt
                  float dot = vector[2];
                  float view[3] =
                  {
                        - dot * vector[0],
                        - dot * vector[1],
                        1.0f - dot * vector[2],
                  };
                  float up[3];
                  VectorCrossProduct(vector,view,up);
                  float B[16] =
                  {
                        up[0], view[0], -vector[0], 0,
                        up[1], view[1], -vector[1], 0,
                        up[2], view[2], -vector[2], 0,
```

```
                        0, 0, 0, 1,
                    };
                    float LocalRollMatrix[16] =
                    {     cosf(Roll), sinf(Roll), 0.0f, 0.0f,
                          -sinf(Roll), cosf(Roll), 0.0f, 0.0f,
                          0.0f, 0.0f, 1.0f, 0.0f,
                          0.0f, 0.0f, 0.0f, 1.0f,
                    };
                    float NewTrnMatrix[16] =
                    {     1.0f, 0.0f, 0.0f, 0.0f,
                          0.0f, 1.0f, 0.0f, 0.0f,
                          0.0f, 0.0f, 1.0f, 0.0f,
                          Transform[12], Transform[13],
                          Transform[14], 1.0f,
                    };
                    MatrixMultiply(B,LocalRollMatrix,TempMatrix1);
                    MatrixMultiply(TempMatrix1,NewTrnMatrix,Transform);
              }

              TimeLastUpdated = CurTime;
        }
}

CVisualObject::UpdateBoundaries(int From)
{
      for (int i=From;i<CurrentTotalVertices;i++)
      {
            if (GetMaxX() < VertexArray[i*3  ])
                  SetMaxX(VertexArray[i*3  ]);
            if (GetMaxY() < VertexArray[i*3+1])
                  SetMaxY(VertexArray[i*3+1]);
            if (GetMaxZ() < VertexArray[i*3+2])
                  SetMaxZ(VertexArray[i*3+2]);

            if (GetMinX() > VertexArray[i*3  ])
                  SetMinX(VertexArray[i*3  ]);
            if (GetMinY() > VertexArray[i*3+1])
                  SetMinY(VertexArray[i*3+1]);
            if (GetMinZ() > VertexArray[i*3+2])
                  SetMinZ(VertexArray[i*3+2]);
      }
}

CVisualObject::TextDumpFaceList(unsigned int total, int *iList)
{
      static int count = 0;
      static int tric = 0;
      char buffer[200];
      sprintf(buffer,"***     New     Face     List     #%d***\r\n",++count);
OutputDebugString(buffer);
      sprintf(buffer,"%d Faces Added.\r\n",total); OutputDebugString(buffer);

      for (unsigned int i=0;i<total;i++)
      {
            sprintf(buffer,"Triangle          %d:          %d,          %d,
%d.\r\n",tric++,iList[i*3],iList[i*3+1],iList[i*3+2]);
            OutputDebugString(buffer);
      }
```

```
	if (ObjectComplete())
		OutputDebugString("####Object is now Completed\r\n");
	OutputDebugString("");
	OutputDebugString("===Done List======Done List======Done List======Done
List======Done List======Done List===\r\n");
}



CAnimation::Reset()
{
	// Cleanup the previous animation now.
	CTexture *ctex = NULL;
	if (TextureListHead)
		ctex = TextureListHead->next;
	while(ctex)
	{
		delete TextureListHead;
		TextureListHead = ctex;
		ctex = ctex->next;
	}
	if (TextureListHead)
		delete TextureListHead;
	TextureListHead = NULL;

	CVisualObject *curr = NULL;
	if (ObjectListHead)
		curr = ObjectListHead->next;
	while(curr)
	{
		delete ObjectListHead;
		ObjectListHead = curr;
		curr = curr->next;
	}
	if (ObjectListHead)
		delete ObjectListHead;
	LightingEnabled = FALSE;

	ReadyTillFrame = -2;

	ObjectListHead = NULL;
	bTimeNotSet = TRUE;
	bBeginningNewAnimation = TRUE;

	// reset the camera
	SceneCamera.PosX = 0.0f;
	SceneCamera.PosY = 0.0f;
	SceneCamera.PosZ = 0.0f;
	SceneCamera.OriX = 0.0f;
	SceneCamera.OriY = 0.0f;
	SceneCamera.OriZ = 0.0f;
	SceneCamera.OriW = 1.0f;

	while (SceneCamera.PosKeyframes)
	{
		CKeyFrame *curr = SceneCamera.PosKeyframes->next;
		delete SceneCamera.PosKeyframes;
```

```
            SceneCamera.PosKeyframes = curr;
        }
        SceneCamera.PosKeyframes = NULL;
        SceneCamera.CurrPosKeyframe = NULL;

        while (SceneCamera.RotKeyframes)
        {
            CKeyFrame *curr = SceneCamera.RotKeyframes->next;
            delete SceneCamera.RotKeyframes;
            SceneCamera.RotKeyframes = curr;
        }
        SceneCamera.RotKeyframes = NULL;
        SceneCamera.CurrRotKeyframe = NULL;

        while (SceneCamera.RollKeyframes)
        {
            CKeyFrame *curr = SceneCamera.RollKeyframes->next;
            delete SceneCamera.RollKeyframes;
            SceneCamera.RollKeyframes = curr;
        }
        SceneCamera.RollKeyframes = NULL;
        SceneCamera.CurrRollKeyframe = NULL;

        SceneCamera.fFarPlane = 1000.0f;
        SceneCamera.fNearPlane = 1.0f;
        SceneCamera.fFieldOfView = 90.0f;

        iLightsInUse = 0;

        BackgroundColor[0] = 0;
        BackgroundColor[1] = 0;
        BackgroundColor[2] = 0;
}

CVisualObject::CalculateTCBQuatDeriv(CKeyFrame *prev, CKeyFrame *curr,
CKeyFrame *next)
// Given the three keyframes, this calculates the the incoming and outgoing
// tangents.  Note that this compensates for either prev or next not
existing.
// In other words, cases of the first and last frames, which are exceptional.

// Note that this function currently appears to work perfectly.  Quaternion
values
// need to be precalculated before this function is used, using FINAL
quaternions
// for a given keyframe, not the values straight-converted from AngleAxis
values
// which would merely be RELATIVE quaternions.  We presume this preprocessing
// step has been performed when we arrive here.

// This code is adapted from the IPAS sample code for KXP information
provided
// by Autodesk.  Original source information can be found at
// http://www.autodesk.com/support/techdocs/fax700/fax746.htm
{
        float qm[4];
        float qp[4];
```

```
        if( prev != NULL ) {
              if (fabs(curr->angleaxis[3]) > 2.0f*FLOAT_PI-.00001)
              {
                    float q[4];
                    q[0] = curr->angleaxis[0];
                    q[1] = curr->angleaxis[1];
                    q[2] = curr->angleaxis[2];
                    q[3] = 0;
                    qlog( q,qm );
              }
              else
              {
                    float qprev[4];
                    qprev[0] = prev->rot[0];
                    qprev[1] = prev->rot[1];
                    qprev[2] = prev->rot[2];
                    qprev[3] = prev->rot[3];
                    if (qdotunit( qprev,curr->rot ) < 0.0f)
                          qnegate( qprev );
                    qlndif( qprev, curr->rot, qm );
              }
        }

        if( next != NULL ) {
              if     (fabs(next->angleaxis[3]     -     curr->angleaxis[3])     >
2.0f*FLOAT_PI-.001)
              {
                    float q[4];
                    q[0] = next->angleaxis[0];
                    q[1] = next->angleaxis[1];
                    q[2] = next->angleaxis[2];
                    q[3] = 0;
                    qlog( q,qp );
              }
              else
              {
                    float qnext[4];
                    qnext[0] = next->rot[0];
                    qnext[1] = next->rot[1];
                    qnext[2] = next->rot[2];
                    qnext[3] = next->rot[3];
                    if (qdotunit( qnext,curr->rot ) < 0.0f)
                          qnegate( qnext );
                    qlndif( curr->rot, qnext, qp );
              }
        }

        if (!prev)
        {     qm[0] = qp[0];    qm[1] = qp[1];    qm[2] = qp[2];    qm[3]      =
qp[3];      }
        if (!next)
        {     qp[0] = qm[0];    qp[1] = qm[1];    qp[2] = qm[2];    qp[3]      =
qm[3];      }

        float fp = 1.0f, fn = 1.0f, cm = 1.0f - curr->continuity;
        if (next && prev)
        {
//            float dt = 1.0f/(0.5f * (float)(next->fTime - prev->fTime));
```

```
//          fp = ((float)(curr->fTime - prev->fTime))*dt;
//          fn = ((float)(next->fTime - curr->fTime))*dt;
            float dt = 1.0f/(0.5f * (float)(next->iTime - prev->iTime));
            fp = ((float)(curr->iTime - prev->iTime))*dt;
            fn = ((float)(next->iTime - curr->iTime))*dt;
            float c = fabsf( curr->continuity );
            fp = fp + c - c * fp;
            fn = fn + c - c * fn;
      }

      float tm = 0.5f*(1.0f - curr->tension);
      float cp = 2.0f - cm;
      float bm = 1.0f - curr->bias;
      float bp = 2.0f - bm;
      float tmcm = tm * cm;
      float tmcp = tm * cp;
      float ksm  = 1.0f - tmcm * bp * fp;
      float ksp  = -tmcp * bm * fp;
      float kdm  = tmcp * bp * fn;
      float kdp  = tmcm * bm * fn - 1.0f;

      float qa[4], qb[4];
      for (int i=0;i<4;i++)
      {
            qa[i]= 0.5f * ( kdm * qm[i]+ kdp * qp[i]);
            qb[i]= 0.5f * ( ksm * qm[i]+ ksp * qp[i]);
      }

      float qae[4], qbe[4];
      qexp( qa, qae );
      qexp( qb, qbe );

      QuaternionMultiply( curr->rot, qae, curr->inTan );
      QuaternionMultiply( curr->rot, qbe, curr->outTan );
}

CVisualObject::SetOffset(float *offsetarray)
{
      for (int i=0;i<4;i++)
            for (int j=0;j<3;j++)
                  OffsetMatrix[i*4+j] = *(offsetarray+i*3+j);
}
```

D.2.18 AnimationFile.h

```
#if
!defined(AFX_ANIMATIONFILE_H__A4F36380_C3EB_11D1_87F8_00AA00B9C442__INCLUDED_)
#define AFX_ANIMATIONFILE_H__A4F36380_C3EB_11D1_87F8_00AA00B9C442__INCLUDED_

#include "Animation.h"  // Added by ClassView
#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// AnimationFile.h : This abstracts the actual animation database, and deals
// entirely with the retrieval of data, and the processing of the opcodes
// received.

float inline FABS(float blah) // calculates the a quick floating point abs.
{
      // courtesy of Crom
      unsigned int* a = (unsigned int *) &blah;
      unsigned int b = 0x7fffffff & *(a);
      float r = *((float *) &b);
      return (r);
};

/////////////////////////////////////////////////////////////////////////////
// CAnimationFile command target
#define BUFFERSIZE 512

class CAnimationFile : public CAsyncMonikerFile
{
// Attributes
public:

// Operations
public:
      CAnimationFile();
      virtual ~CAnimationFile();

private:
      unsigned int iCompressedCount;
      unsigned int iUnCompressedCount;
      unsigned int bitlength;
      unsigned int GetCode(int &codepointer, BYTE * CodeBuffer, int bitlength,
            int &input_bit_count,unsigned int &input_bit_buffer);
public:
      long lBufferRequired;
      int   iSpeedType;
      int iSpeed;
      BOOL bPreBuffering;
      long lTotalFileLength;
      BOOL bDecodeComplete;
      unsigned char *Dictionary;
      Reset();
// Overrides
      int DecodeHeader();
      HaarInv(float a[], float source[], unsigned int size);
      ConvertColor16To24(unsigned int source,unsigned char &red, unsigned char
```

```
&green,
            unsigned char &blue);
      float invTPF;
      float ScaleUp(unsigned int val, float min, float max, int bitdepth);
      unsigned char DecodeByte(unsigned char * Buffer, int & pointer);
      unsigned short DecodeWord(unsigned char * Buffer, int & pointer);
      unsigned int DecodePartialWord(unsigned char *Buffer, int &pointer,
            unsigned int bitlength, unsigned int &input_bit_buffer,
            unsigned int &input_bit_count, int &total);
      float DecodeFloat(unsigned char * Buffer, int & pointer);
      BOOL bAbortAnimation;
      ProcessVersionNumber();
      int iBitsPerVertex;
      BOOL bCompressedNormals;
      BOOL bDownloadComplete;
      BOOL bAnimationInProgress;
      BOOL   Decode3FloatList(int   count,   unsigned   char   *Buffer,   float
*floatlist, int &pointer);
      BOOL bReadyToDisplay;
      CAnimation Animation;
      BOOL DecodeCode(int &codepointer, BYTE *CodeBuffer, int &respointer,
BYTE *ResBuffer,
            BOOL &Reset);
      int AnimPos;
      void DecodeAnimation(int HeaderLength,BOOL &Reset,int HoldBack);
      void ParseAnimation();
      BYTE Buffer[BUFFERSIZE*2];
      BYTE AnimBuffer[BUFFERSIZE*48];
      int BufferPos;
      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CAnimationFile)
      protected:
      virtual void OnDataAvailable(DWORD dwSize, DWORD bscfFlag);
      virtual void OnProgress(ULONG ulProgress, ULONG ulProgressMax,
            ULONG ulStatusCode, LPCTSTR szStatusText);
      //}}AFX_VIRTUAL

      // Generated message map functions
      //{{AFX_MSG(CAnimationFile)
            // NOTE - the ClassWizard will add and remove member functions
here.
      //}}AFX_MSG

// Implementation
protected:
private:
      unsigned int DecodeShortInt(unsigned char* Buffer, int &pointer);
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

#endif                                                                    //
```

```
!defined(AFX_ANIMATIONFILE_H__A4F36380_C3EB_11D1_87F8_00AA00B9C442__INCLUDED_)
```

D.2.19 AnimationFile.cpp

```
// AnimationFile.cpp : implementation file
//

#include "stdafx.h"
#include "mmsystem.h"
#include "math.h"
#include "Quaternion.h"
#include "Decompressor.h"
#include "AnimationFile.h"
#include "../Compressor/Opcodes.h"
#include "../Compressor/PixelPeano.h"

#include "Animation.h"

#include "DialogLog.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

extern CStatusBar *m_pwndStatusBar;
extern CSliderCtrl* pSlider;
extern BOOL bSliderCreated;
extern CDialogLog *pLog;
extern CWnd *pWin;

/////////////////////////////////////////////////////////////////////////////
// CAnimationFile

CAnimationFile::CAnimationFile()
{
      iSpeedType = 0;
      bPreBuffering = FALSE;
      bAnimationInProgress = FALSE;
      bAbortAnimation = FALSE;
      bReadyToDisplay = FALSE;
      bDecodeComplete = FALSE;
      Dictionary = new unsigned char [LZSSWINDOW_LENGTH];
}

CAnimationFile::~CAnimationFile()
{
      delete[] Dictionary;
}


// Do not edit the following lines, which are needed by ClassWizard.
#if 0
BEGIN_MESSAGE_MAP(CAnimationFile, CAsyncMonikerFile)
      //{{AFX_MSG_MAP(CAnimationFile)
      //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
#endif      // 0

/////////////////////////////////////////////////////////////////////////////
// CAnimationFile member functions

void CAnimationFile::OnDataAvailable(DWORD dwSize, DWORD bscfFlag)
{
      // fill the buffer with a maximum of 256 characters, and then parse it
for commands.
      static long m_dwReadBefore;
      static BOOL bReset;

      // do this on the first data recieved from the connection
      if ((bscfFlag & BSCF_FIRSTDATANOTIFICATION) != 0)
      {
            m_dwReadBefore = 0;
            BufferPos = 0;
            AnimPos = 0;
            bDownloadComplete = FALSE;
            bAbortAnimation = FALSE;
            Animation.bBeginningNewAnimation = TRUE;
            bDecodeComplete = FALSE;
            switch (iSpeedType)
            {
                  case SPEED_288:          iSpeed = 2500; break;
                  case SPEED_56k:          iSpeed = 5000; break;
                  case SPEED_ISDN:   iSpeed = 5700; break;
                  case SPEED_ADSL:   iSpeed = 11400; break;
                  case SPEED_T1:           iSpeed = 134000; break;
            }
            bPreBuffering = TRUE;
            lBufferRequired = 2147483647;
      }

      pLog->m_fsize = dwSize;
      pLog->UpdateData(FALSE);

      DWORD dwArriving = dwSize - m_dwReadBefore;
      while (dwArriving > 0 && !bAbortAnimation && !bDecodeComplete)
      {
            int iHoldBack = 3;
            if (dwArriving > BUFFERSIZE - 1)
            {
                  Read(Buffer + BufferPos, BUFFERSIZE);
                  m_dwReadBefore += BUFFERSIZE;
                  dwArriving -= BUFFERSIZE;
                  BufferPos += BUFFERSIZE;
            }
            else
            {
                  Read(Buffer + BufferPos, dwArriving);
                  m_dwReadBefore += dwArriving;
                  BufferPos += dwArriving;
                  dwArriving = 0;
//                iHoldBack = 3;
            }
            if (m_dwReadBefore >= lBufferRequired)
                  bPreBuffering = FALSE;
```

```
            pWin->RedrawWindow(NULL,NULL,RDW_INVALIDATE);

            if (!Animation.bBeginningNewAnimation ||
                  BufferPos >= 14)
            {
                  // if we're beginning, parse the header
                  // otherwise, decompress the animation
                  // data
                  if (Animation.bBeginningNewAnimation)
                  {
                        bReset = TRUE;
                        DecodeAnimation(DecodeHeader(),bReset,iHoldBack);
                        ParseAnimation();
                  }
                  else
                  {
                        DecodeAnimation(0,bReset,iHoldBack);
                        ParseAnimation();
                  }
            }
      }

      CAsyncMonikerFile::OnDataAvailable(dwSize, bscfFlag);
}

void CAnimationFile::OnProgress(ULONG ulProgress, ULONG ulProgressMax,
            ULONG ulStatusCode, LPCTSTR szStatusText)
{
      CString s;
      switch (ulStatusCode)
      {
            case BINDSTATUS_FINDINGRESOURCE:
            {
                  s.Format("Looking up %s",szStatusText);
                  m_pwndStatusBar->SetPaneText(0,(LPCSTR)s,TRUE);
            } break;
            case BINDSTATUS_USINGCACHEDCOPY:
            {
                  s.Format("Grabbing local copy from cache",szStatusText);
                  m_pwndStatusBar->SetPaneText(0,(LPCSTR)s,TRUE);
            } break;
            case BINDSTATUS_CONNECTING:
            {
                  s.Format("Site found... connecting to %s",szStatusText);
                  m_pwndStatusBar->SetPaneText(0,(LPCSTR)s,TRUE);
            } break;
            case BINDSTATUS_BEGINDOWNLOADDATA:
            {
                  s.Format("Beginning transfer..",szStatusText);
                  m_pwndStatusBar->SetPaneText(0,(LPCSTR)s,TRUE);
            } break;
            case BINDSTATUS_DOWNLOADINGDATA:
            {
                  s.Format("%d completed",
//                      (int)(100.0*(float)ulProgress                    /
(float)ulProgressMax),
                        ulProgress);//, ulProgressMax);
```

```
                m_pwndStatusBar->SetPaneText(0,(LPCSTR)s,TRUE);
            } break;
            case BINDSTATUS_ENDDOWNLOADDATA:
            {
                s.Format("Download Completed",szStatusText);
                m_pwndStatusBar->SetPaneText(0,(LPCSTR)s,TRUE);
               bDownloadComplete = TRUE;
            }
      }

      CAsyncMonikerFile::OnProgress(ulProgress,ulProgressMax,
            ulStatusCode,szStatusText);
}

int CAnimationFile::DecodeHeader()
{
      pLog->m_fsize = 0;
      pLog->m_polys = 0;
      pLog->m_verts = 0;
      pLog->Clear();
      char buffer[200];

      int i=0;
      // parse animation header
      unsigned char b0 = Buffer[0];
      unsigned char b1 = Buffer[1];
      unsigned char b2 = Buffer[2];
      unsigned char b3 = Buffer[3];

      pLog->Log("Parsing file header...\r\n");

      // Process bools from the header
      iBitsPerVertex = b0 & 0x03;

      bCompressedNormals = b0 & 0x80;
      if (bCompressedNormals)
            pLog->Log("- Compressed normals included\r\n");
      else pLog->Log("- Compressed normals not included\r\n");

      Animation.bVertexNormals = b0 & 0x40;
      if (Animation.bVertexNormals)
            pLog->Log("- Vertex normals included\r\n");
      else pLog->Log("- Vertex normals not included\r\n");

      Animation.bMaterialQuality = b0 & 0x20;
      if (Animation.bMaterialQuality )
            pLog->Log("- Texturemaps used.\r\n");
      else pLog->Log("- Object or vertex colors only used.\r\n");

      i+=4;
      if ((b0 & 0x10) == 0)
            Animation.iSourceFPS = 30;
      else
            Animation.iSourceFPS = DecodeShortInt(Buffer,i);

      sprintf(buffer,"- Original Framerate = %d\r\n",Animation.iSourceFPS);
      pLog->Log(buffer);
```

```
if ((b0 & 0x08) == 0)
      Animation.iSourceTPF = 160;
else
      Animation.iSourceTPF = DecodeShortInt(Buffer,i);
sprintf(buffer,"- Original Tickrate = %d\r\n",Animation.iSourceTPF);
pLog->Log(buffer);

if ((b0 & 0x04) == 0)
{
      Animation.bSimpleShading = FALSE;
      pLog->Log("- Simple shading not being used.\r\n");
}
else
{
      Animation.bSimpleShading = TRUE;
      pLog->Log("- Simple shading being used.\r\n");
}

invTPF = 1.0f / (float) (Animation.iSourceTPF * Animation.iSourceFPS);
// Get Version number
Animation.MajorVersion= b2;
Animation.MinorVersion= b3;
ProcessVersionNumber();
sprintf(buffer,"- File version number = %d.%d\r\n",
      Animation.MajorVersion,Animation.MinorVersion);
pLog->Log(buffer);

unsigned int frame = 0;
unsigned char* ucTime = (unsigned char*) &frame;
ucTime[0] = DecodeByte(Buffer,i);
ucTime[1] = DecodeByte(Buffer,i);
ucTime[2] = DecodeByte(Buffer,i);

Animation.FinalFrame = frame;
sprintf(buffer,"- Total internal frame count = %d\r\n",
      Animation.FinalFrame);
pLog->Log(buffer);

if (bSliderCreated)
{
      pSlider->SetRange(0,Animation.FinalFrame*100,TRUE);
      pSlider->SetPos(0);
}

unsigned int res = 0;
unsigned char *ucRes = (unsigned char*) &res;
ucRes[0] = DecodeByte(Buffer,i);
ucRes[1] = DecodeByte(Buffer,i);
ucRes[2] = DecodeByte(Buffer,i);

Animation.iTargetX = (res & 0x00000FFF);
Animation.iTargetY = (res >> 12);

int val = 0;
int cx = (::GetSystemMetrics(SM_CXSCREEN) >> 1) - Animation.iTargetX/2;
int cy = (::GetSystemMetrics(SM_CYSCREEN) >> 1) - Animation.iTargetY/2;
if (!bAbortAnimation)
{
```

```
            pWin->MoveWindow(cx,cy,Animation.iTargetX+12,Animation.iTargetY+98
);
            pWin->RedrawWindow();//NULL,NULL,RDW_INVALIDATE | RDW_UPDATENOW);
      }
      sprintf(buffer,"- Animation resolution = %dx%d\r\n",
            Animation.iTargetX,Animation.iTargetY);
      pLog->Log(buffer);

      unsigned int totalLen = 0;
      unsigned char *ucLen = (unsigned char*) &totalLen;
      ucLen[0] = DecodeByte(Buffer,i);
      ucLen[1] = DecodeByte(Buffer,i);
      ucLen[2] = DecodeByte(Buffer,i);
      ucLen[3] = DecodeByte(Buffer,i);

      if (totalLen != 0)
            lTotalFileLength = totalLen;
      else lTotalFileLength = 1;

      sprintf(buffer,"- Total file length before LZSS = %d\r\n",
            lTotalFileLength);
      pLog->Log(buffer);

      float       multiplier       =       (float)Animation.FinalFrame       /
(float)Animation.iSourceFPS;
      lBufferRequired = lTotalFileLength - (long)( (float)iSpeed*multiplier);
      if (lBufferRequired < 0)
            lBufferRequired = 0;

      bPreBuffering = TRUE;
      pLog->Log("Header Parsed.\r\n");

      return (i);
}

void CAnimationFile::ParseAnimation()
{
      int i=0;
      BOOL bDone = FALSE;
      static float floatlist[ITEMSPERBLOCK*3];
      static int   intlist[256*3];
      static unsigned char charlist[256*3];

      // note that this little group of variables are used for decoding
texture mipmaps
      static BOOL bDecodingTextures;
      static CTexture *DecodingTex;
      static int CoefficientIndex;
      static int CoefficientBound;
      static float CoefficientMatrix[MAXTEXTURERES*MAXTEXTURERES*3];
      static CPixelPeano *pPeano;
      static int CoefficientColor;
      static int CoefficientInc;
      static float TextureDelta;
      static float PreviousCoeff;

      if (Animation.bBeginningNewAnimation)
```

```
	{
		// reset local variables for new file
		Animation.bBeginningNewAnimation = FALSE;

		bDecodingTextures = FALSE;
		DecodingTex = NULL;
		CoefficientIndex = 0;
		CoefficientBound = 0;
		CoefficientColor = 0;
		pPeano = NULL;
		TextureDelta = 0.0f;
		PreviousCoeff = 0.0f;
	}

//	AnimPos-=2;

	while (i<AnimPos && !bDone && !bAbortAnimation)
	// parse individual opcodes
	{
		int tempi;
		tempi = i;

		// If we are decoding texture maps, we do that here & now.
		if (bDecodingTextures)
		{
			float tempStore;
			// add texture components until the current mipmap is
filled.
			if (DecodingTex->iCurrentMipLevel == MINTEXTUREPOWER)
			for (int loop = tempi;CoefficientIndex<CoefficientBound &&
i<AnimPos;loop++)
			{
				unsigned char databyte = DecodeByte(AnimBuffer,i);
				if (databyte != 0xFF)
				{
					*(CoefficientMatrix+CoefficientColor*MAXTEXTURER
ES*MAXTEXTURERES+
						pPeano->coordx+MAXTEXTURERES*pPeano->coord
y) =
						(float)(0.0f	+	(float)databyte	*
1.00393700787f);

					while(pPeano->Process());
					CoefficientIndex++;
				}
				else
				{
					// here we have our zero decoder
					int count = DecodeByte(AnimBuffer,i);
					if (count == 0) count = 256;
					CoefficientIndex+=count;
					for (int
innerloop=0;innerloop<count;innerloop++)
					{
						*(CoefficientMatrix+CoefficientColor*MAXTE
XTURERES*MAXTEXTURERES+
							pPeano->coordx+MAXTEXTURERES*pPeano-
>coordy) = 0.0f;
```

```
                                    while(pPeano->Process());
                              }
                        }
                  }
                  else
                  for (int loop = tempi;CoefficientIndex<CoefficientBound &&
i<AnimPos;loop++)
                  {
                        unsigned char databyte = DecodeByte(AnimBuffer,i);
                        if (databyte != 0xFF)
                        {
                              tempStore = (float)(DecodingTex->fTextureMin + (
float)databyte * TextureDelta) +
                                    PreviousCoeff;
                              *(CoefficientMatrix+CoefficientColor*MAXTEXTURER
ES*MAXTEXTURERES+
                                    pPeano->coordx+MAXTEXTURERES*pPeano->coord
y) = tempStore;
                              //                    PreviousCoeff = tempStore; // uncomment to
enable delta decoding

                              while(pPeano->Process());
                              CoefficientIndex++;
                        }
                        else
                        {
                              // here we have our zero decoder
                              int count = DecodeByte(AnimBuffer,i);
                              if (count == 0) count = 256;
                              CoefficientIndex+=count;
                              for                                          (int
innerloop=0;innerloop<count;innerloop++)
                              {
                                    *(CoefficientMatrix+CoefficientColor*MAXTE
XTURERES*MAXTEXTURERES+
                                          pPeano->coordx+MAXTEXTURERES*pPeano-
>coordy) = PreviousCoeff;
                                    while(pPeano->Process());
                              }
                        }
                  }

                  if (CoefficientIndex == CoefficientBound)
                  {
                        if (++CoefficientColor == 3)
                        {
                              // now clean up a bit
                              bDecodingTextures = FALSE;
                              if (pPeano)
                              {
                                    delete pPeano;
                                    pPeano = NULL;
                              }
                              if       (DecodingTex->iCurrentMipLevel       ==
MINTEXTUREPOWER)
                              {
                                    // Copy the texture data into the the
```

```
object's texture holder for
                                    // this mipmap level.
                                    DecodingTex->MipMaps[MINTEXTUREPOWER] =
                                          new                                unsigned
char[MINTEXTURERES*MINTEXTURERES*3];
                                    for                                          (int
countx=0;countx<MINTEXTURERES;countx++)
                                          for                                    (int
county=0;county<MINTEXTURERES;county++)
                                                for                              (int
color=0;color<3;color++)
                                          *(DecodingTex->MipMaps[MINTEXTUREPOW
ER]+3*(MINTEXTURERES*county+countx)+color) =
                                                                 (unsigned
char)*(CoefficientMatrix+color*MAXTEXTURERES*MAXTEXTURERES+
                                                                 county*MAXTEXTURERES+cou
ntx);
                                    // Average & sample the MIN texture size
to construct the rest of the
                                    // lower level mipmaps down to a 1x1 map.
                                    int power=MINTEXTUREPOWER-1;
                                    for     (int     val=(MINTEXTURERES     >>
1);val>0;val=val>>1)
                                    {
                                          DecodingTex->MipMaps[power]   =   new
unsigned char[val*val*3];
                                          for (int color=0;color<3;color++)
                                                      for          (int
countx=0;countx<val;countx++)
                                                for                          (int
county=0;county<val;county++)
*(DecodingTex->MipMaps[power]+3*(county*val+countx)+color) = (
                                                                 *(DecodingTex->MipMaps[p
ower+1]+3*((county<<1)*val+(countx<<1))+color) +
                                                                 *(DecodingTex->MipMaps[p
ower+1]+3*((county<<1)*val+(countx<<1)+1)+color) +
                                                                 *(DecodingTex->MipMaps[p
ower+1]+3*(((county<<1)+1)*val+(countx<<1))+color) +
                                                                 *(DecodingTex->MipMaps[p
ower+1]+3*(((county<<1)+1)*val+(countx<<1)+1)+color))>>2;
                                          power--;
                                    }
                              }
                              else
                              {
                                    // first let's begin by calculating the
mipmap resolution
                                    int val = MINTEXTURERES;
                                    for                                          (int
count=MINTEXTUREPOWER;count<DecodingTex->iCurrentMipLevel-1;count++) val*=2;

                                    // Copy the previous texture data as the
approximation
                                    //   for   the   current   mipmap   level's
coefficients.
```

```
                                    for (int countx=0;countx<val;countx++)
                                          for                                (int
county=0;county<val;county++)
                                                for                          (int
color=0;color<3;color++)
                                                      *(CoefficientMatrix+colo
r*MAXTEXTURERES*MAXTEXTURERES+
                                                            county*MAXTEXTURER
ES+countx) =
                                                            *(DecodingTex->Mip
Maps[DecodingTex->iCurrentMipLevel-1]+
                                                            3*(val*county+coun
tx)+color);

                                    val*=2;
                                    // Do a wavelet decode
                                    float DataArray[MAXTEXTURERES];
                                    float TempArray[MAXTEXTURERES];
                                    for (int color=0;color<3;color++)
                                    {
                                          for (count=0;count<val;count++)
                                          {
                                                for                          (int
count2=0;count2<val;count2++)
                                                      DataArray[count2]=*(Coef
ficientMatrix+color*MAXTEXTURERES*MAXTEXTURERES+
                                                            count2*MAXTEXTURER
ES+count);
                                                switch(DecodingTex->iTextureTr
ansformType)
                                                {
                                                      //     insert     inverse
wavelet transforms here
                                                      case        TEXTYPE_HAAR:
HaarInv(TempArray,DataArray,val); break;
                                                }
                                                for
(count2=0;count2<val;count2++)
                                                      *(CoefficientMatrix+colo
r*MAXTEXTURERES*MAXTEXTURERES+
                                                            count2*MAXTEXTURER
ES+count) = TempArray[count2];
                                          }
                                          for (count=0;count<val;count++)
                                          {
                                                for                          (int
count2=0;count2<val;count2++)
                                                      DataArray[count2]=*(Coef
ficientMatrix+color*MAXTEXTURERES*MAXTEXTURERES+
                                                            count*MAXTEXTURERE
S+count2);
                                                TempArray[MAXTEXTURERES];
                                                switch(DecodingTex->iTextureTr
ansformType)
                                                {
                                                      //     insert     inverse
wavelet transforms here
                                                      case        TEXTYPE_HAAR:
```

```
HaarInv(TempArray,DataArray,val); break;
                                                    }
                                                    for
(count2=0;count2<val;count2++)
                                                          *(CoefficientMatrix+colo
r*MAXTEXTURERES*MAXTEXTURERES+
                                                          count*MAXTEXTURERES+coun
t2) = TempArray[count2];
                                              }
                                        }

                                        // Copy the texture data into the the
object's texture holder for
                                        // this mipmap level.
                                        DecodingTex->MipMaps[DecodingTex->iCurrent
MipLevel] = new unsigned char[val*val*3];
                                        for (color=0;color<3;color++)
                                          for (int countx=0;countx<val;countx++)
                                              for                                 (int
county=0;county<val;county++)
                                                    for                           (int
color=0;color<3;color++)
                                                    *(DecodingTex->MipMaps[DecodingTex->
iCurrentMipLevel]+3*(val*county+countx)+color) =
                                                          (unsigned
char)*(CoefficientMatrix+color*MAXTEXTURERES*MAXTEXTURERES+
                                                          county*MAXTEXTURERES+cou
ntx);
                                  }
                                  pLog->Log("MipMap decoded successfully.\r\n");

// Diagnostic code that simply dumps the current mipmap to a file
//                                char buf[200];
//                                sprintf(buf,"_mip%d",DecodingTex->iCurrentMipLev
el);
//                                DecodingTex->DumpMipMap(buf,DecodingTex->iCurren
tMipLevel);

                                  DecodingTex->iCurrentMipLevel++;
                                  DecodingTex->bTextureUnbound = TRUE;
                                  DecodingTex->bTextureInUse = TRUE;
                            }
                            else
                            {
                                  CoefficientBound                              =
(CoefficientColor+1)*CoefficientInc;
//                                PreviousCoeff = 0.0f;
                                  pPeano->Reset();
                            }
                      }
                }

                // Process opcodes
                else switch (DecodeShortInt(AnimBuffer,i))
                {
                case OPCODE_ADDVERTS:
                      // add vertices to an object
                      {
```

```
                        if (i+4 >= AnimPos)
                        {
                              i = tempi;
                              bDone = TRUE;
                              break;
                        }

                        unsigned short which;
                        unsigned short total;
                        which = DecodeShortInt(AnimBuffer,i);
                        total = DecodeShortInt(AnimBuffer,i);

                        CVisualObject* VisObj = Animation.FindObject(which);

                        switch (iBitsPerVertex)
                        {
                              case BPC_32:
                              {
                                    if (i+total*4*3 >= AnimPos)
                                    {
                                          i = tempi;
                                          bDone = TRUE;
                                          break;
                                    }
                                    // unpack the bytes into a group of
floats
                                    // floats are packed (for vertices A,B,C)
as
                                    //
XA0_XB0_XC0_XA1_XB1_XC1_XA2_XB2_XC2_XA3_XB3_XC3_....
                                    //
YA0_YB0_YC0_YA1_YB1_YC1_YA2_YB2_YC2_YA3_YB3_YC3_....
                                    //
ZA0_ZB0_ZC0_ZA1_ZB1_ZC1_ZA2_ZB2_ZC2_ZA3_ZB3_ZC3

                                    Decode3FloatList(total,AnimBuffer,floatlis
t,i);
                              } break;
                              case BPC_16:
                              {
                                    if (i+total*2*3 >= AnimPos)
                                    {
                                          i = tempi;
                                          bDone = TRUE;
                                          break;
                                    }

                                    for (int loop=0;loop < total;loop++)

                                          floatlist[loop*3 ] =
                                                ScaleUp(DecodeWord(AnimB
uffer,i),
                                                      VisObj->GetMinX(),
                                                      VisObj->GetMaxX(),
                                                      BPC_16);
                                    for (loop=0;loop < total;loop++)

                                          floatlist[loop*3+1] =
```

```
                                                ScaleUp(DecodeWord(AnimB
uffer,i),
                                                      VisObj->GetMinY(),
                                                      VisObj->GetMaxY(),
                                                      BPC_16);
                                    for (loop=0;loop < total;loop++)

                                                floatlist[loop*3+2] =
                                                      ScaleUp(DecodeWord(AnimB
uffer,i),
                                                      VisObj->GetMinZ(),
                                                      VisObj->GetMaxZ(),
                                                      BPC_16);

                              } break;
                              case BPC_10:
                              {
                                    if (i+total*1.25*3+1 >= AnimPos)
                                    {
                                          i = tempi;
                                          bDone = TRUE;
                                          break;
                                    }

                                    unsigned int bbuf=0, bcount=0;
                                    int pcount = total*3;

                                    for   (int   loop=0;loop   <   total;loop++)

                                                floatlist[loop*3   ] =
                                                      ScaleUp(DecodePartialWor
d(AnimBuffer,i,10,bbuf,bcount,pcount),
                                                      VisObj->GetMinX(),
                                                      VisObj->GetMaxX(),
                                                      BPC_10);
                                    for (loop=0;loop < total;loop++)

                                                floatlist[loop*3+1] =
                                                      ScaleUp(DecodePartialWor
d(AnimBuffer,i,10,bbuf,bcount,pcount),
                                                      VisObj->GetMinY(),
                                                      VisObj->GetMaxY(),
                                                      BPC_10);
                                    for (loop=0;loop < total;loop++)

                                                floatlist[loop*3+2] =
                                                      ScaleUp(DecodePartialWor
d(AnimBuffer,i,10,bbuf,bcount,pcount),
                                                      VisObj->GetMinZ(),
                                                      VisObj->GetMaxZ(),
                                                      BPC_10);

                              } break;
                              case BPC_8:
                              {
                                    if (i+total*3 >= AnimPos)
                                    {
                                          i = tempi;
```

```
                                    bDone = TRUE;
                                    break;
                              }

                              for (int loop = 0;loop < total;loop++)

                                    floatlist[loop*3  ] =
                                          ScaleUp(DecodeByte(AnimBuffer,
i),
                                                      VisObj->GetMinX(),
                                                      VisObj->GetMaxX(),
                                                      BPC_8);
                              for (loop = 0;loop < total;loop++)

                                    floatlist[loop*3+1] =
                                          ScaleUp(DecodeByte(AnimBuffer,
i),
                                                      VisObj->GetMinY(),
                                                      VisObj->GetMaxY(),
                                                      BPC_8);
                              for (loop = 0;loop < total;loop++)

                                    floatlist[loop*3+2] =
                                          ScaleUp(DecodeByte(AnimBuffer,
i),
                                                      VisObj->GetMinZ(),
                                                      VisObj->GetMaxZ(),
                                                      BPC_8);

                        } break;
                  }
                  if (!bDone)
                  {
                        if (VisObj)
                        {
                              if (!VisObj->AddVertices(total,floatlist))
                              {
                                    AfxMessageBox("\
Error - Attempt to add more vertices than the \n\
set maximum for the given model allows.");
                                    bAbortAnimation = TRUE;
                              }
                              else
                              {
                                    if (iBitsPerVertex == BPC_32)
                                          VisObj->UpdateBoundaries(VisOb
j->CurrentTotalVertices);

                                    char buffer[200];
                                    sprintf(buffer,
                                          "%d Vertices Added to object
#%d.\r\n",total,which);
                                    pLog->Log(buffer);
                                    if (VisObj->bCompleted)
                                    {
                                          char buffer[200];
                                          sprintf(buffer,
                                                "Object                #%d
```

```
completed.\r\n",which);
                                                pLog->Log(buffer);
                                        }
                                }
                        }
                        else
                        {
                                AfxMessageBox("\
An Object reference found for an object not yet added to the\n\
Animation database.");
                                bAbortAnimation = TRUE;
                        }
                }
        } break;
    case OPCODE_ADDNORMS:
        // add a block of vector normals to an object
        {
                if (i+4 >= AnimPos)
                {
                        i = tempi;
                        bDone = TRUE;
                        break;
                }
                int which;
                int total;
                which = DecodeShortInt(AnimBuffer,i);
                total = DecodeShortInt(AnimBuffer,i);
                CVisualObject *Curr = Animation.FindObject(which);
                if (!Curr)
                {
                        AfxMessageBox("\
An object reference was found refering to an object not yet\n\
added to the Animation database.");
                        bAbortAnimation = TRUE;
                        break;
                }
                if (bCompressedNormals)
                {
                        if (i+total*3 >= AnimPos)
                        {
                                i = tempi;
                                bDone = TRUE;
                                break;
                        }
                        // Instead of 6 bytes per normal, we use 3
bytes per
                        // normal.   These 3 bytes specify two 12b
angles,
                        // used to compute a rotation.
                        // We convert from spherical coordinates to
cartesian
                        // coordinates, assuming a fixed vector length
                        // of one.
                        unsigned int combo = 0;
                        unsigned char * pcombo = (unsigned char *)
&combo;
                        unsigned int theta, psi;
```

```
                        for (int loop = 0;loop < total;loop++)
                        {
                              pcombo[0] = DecodeByte(AnimBuffer,i);
                              pcombo[1] = DecodeByte(AnimBuffer,i);
                              pcombo[2] = DecodeByte(AnimBuffer,i);
                              theta = ((combo & 0x00FFF000) >> 12);
                              psi = (combo & 0x00000FFF);
                              float fTheta = ScaleUp(theta,
-FLOAT_HALFPI,FLOAT_HALFPI,BPC_12);
                              float fPsi   = ScaleUp(psi,
-FLOAT_HALFPI,FLOAT_HALFPI,BPC_12);
                              floatlist[loop*3  ] =
                                    cosf(fTheta) * sinf(fPsi);
                              floatlist[loop*3+1] =
                                    sinf(fTheta) * sinf(fPsi);
                              floatlist[loop*3+2] =
                                    cosf(fPsi);
                        }
                  }
                  else
                  {
                        if (i+total*6 >= AnimPos)
                        {
                              i = tempi;
                              bDone = TRUE;
                              break;
                        }
                        for (int loop = 0;loop < total;loop++)

                              floatlist[loop*3  ] =
                                    ScaleUp(DecodeWord(AnimBuffer,i),
                                                -1.0f,
                                                1.0f,
                                                BPC_16);
                        for (loop = 0;loop < total;loop++)

                              floatlist[loop*3+1] =
                                    ScaleUp(DecodeWord(AnimBuffer,i),
                                                -1.0f,
                                                1.0f,
                                                BPC_16);
                        for (loop = 0;loop < total;loop++)

                              floatlist[loop*3+2] =
                                    ScaleUp(DecodeWord(AnimBuffer,i),
                                                -1.0f,
                                                1.0f,
                                                BPC_16);

                  }
                  if (!Curr->AddNormals(floatlist,total))
                  {
                        AfxMessageBox("\
Error - Attempt was made to add more normals than the\n\
set maximum for a specific object model allows.");
                        bAbortAnimation = TRUE;
```

```
                        }
                        else
                        {
                              char buffer[200];
                              sprintf(buffer,
                                    "%d     Normals     Added     to     object
#%d.\r\n",total,which);
                              pLog->Log(buffer);
                              if (Curr->bCompleted)
                              {
                                    char buffer[200];
                                    sprintf(buffer,
                                          "Object #%d completed.\r\n",which);
                                    pLog->Log(buffer);
                              }
                        }
                  } break;
            case OPCODE_ADDFACES:
                  // add a block of faces to an object
                  {
                        if (i+4 >= AnimPos)
                        {
                              i = tempi;
                              bDone = TRUE;
                              break;
                        }
                        int which;
                        unsigned int total;
                        which = DecodeShortInt(AnimBuffer,i);
                        total = DecodeShortInt(AnimBuffer,i);
                        CVisualObject *Curr = Animation.FindObject(which);
                        int multiplier;
                        if (Curr && Curr->TotalFaces < 200)
                              multiplier = 1;
                        else multiplier = 2;

                        unsigned int loop;
                        int * iList = intlist;

                        // because the size of the facelist is not static and
is dependant
                        // on the vertex numbers referenced, we check if we
have enough data
                        // retrieved at the head of every face

                        for (loop = 0;loop < total;loop++)
                        {
                              if (i+3*multiplier >= AnimPos)
                              {
                                    i = tempi;
                                    bDone = TRUE;
                                    break;
                              }
                              *(iList++) = DecodeShortInt(AnimBuffer,i);
                              *(iList++) = DecodeShortInt(AnimBuffer,i);
                              *(iList++) = DecodeShortInt(AnimBuffer,i);
                        }
```

```
                        if (bDone) break;
                        if (Curr)
                        {

                              if (!Curr->AddFaces(total,intlist))
                              {
                                    AfxMessageBox("\
Error - Attempt was made to add more faces than the\n\
set maximum for a specific object model allows.");
                                    bAbortAnimation = TRUE;
                              }
                              else
                              {
                                    char buffer[200];
                                    sprintf(buffer,
                                          "%d    Faces    Added    to    object
#%d.\r\n",total,which);
                                    pLog->Log(buffer);
                                    if (Curr->bCompleted)
                                    {
                                          char buffer[200];
                                          sprintf(buffer,
                                                "Object                    #%d
completed.\r\n",which);
                                          pLog->Log(buffer);
                                    }
                              }
                        }
                        else
                        {
                              AfxMessageBox("\
An object reference was found refering to an object not yet\n\
added to the Animation database.");
                              bAbortAnimation = TRUE;
                        }
                  } break;
            case OPCODE_ADDVERTCOLORS:
                  {
                        if (i+4 >= AnimPos)
                        {
                              i = tempi;
                              bDone = TRUE;
                              break;
                        }
                        int which;
                        int total;
                        which = DecodeShortInt(AnimBuffer,i);
                        total = DecodeShortInt(AnimBuffer,i);
                        CVisualObject *Curr = Animation.FindObject(which);
                        if (i+total*2 >= AnimPos)
                        {
                              i = tempi;
                              bDone = TRUE;
                              break;
                        }
                        unsigned char * cList = charlist;
                        for (int loop = 0;loop < total;loop++)
                        {
```

```
                        unsigned long color = DecodeWord(AnimBuffer,i);
                        ConvertColor16To24(color,*(cList),*(cList+1),*(c
List+2));
                        cList+=3;
                  }
                  if (Curr)
                  {

                        if (!Curr->AddVertColors(total,charlist))
                        {
                              AfxMessageBox("\
Error - Attempt was made to add more vertex colors than the\n\
set maximum for a specific object model allows.");
                              bAbortAnimation = TRUE;
                        }
                        else
                        {
                              char buffer[200];
                              sprintf(buffer,
                                    "%d Vertex Colors Added to object
#%d.\r\n",total,which);
                              pLog->Log(buffer);
                              if (Curr->bCompleted)
                              {
                                    char buffer[200];
                                    sprintf(buffer,
                                          "Object                    #%d
completed.\r\n",which);
                                    pLog->Log(buffer);
                              }
                        }
                  }
                  else
                  {
                        AfxMessageBox("\
An object reference was found refering to an object not yet\n\
added to the Animation database.");
                        bAbortAnimation = TRUE;
                  }
            } break;
      case OPCODE_ADDTEXCOORDS:
            {
                  if (i+4 >= AnimPos)
                  {
                        i = tempi;
                        bDone = TRUE;
                        break;
                  }
                  int which;
                  int total;
                  which = DecodeShortInt(AnimBuffer,i);
                  total = DecodeShortInt(AnimBuffer,i);
                  CVisualObject *Curr = Animation.FindObject(which);
                  if (i+total*3 >= AnimPos)
                  {
                        i = tempi;
                        bDone = TRUE;
                        break;
```

```
                        }
                        float * fList = floatlist;
                        for (int loop = 0;loop < total;loop++)
                        {
                              unsigned int holder = 0,ucoord,vcoord;
                              unsigned     char     *pHolder     =     (unsigned
char*)&holder;
                              pHolder[0] = DecodeByte(AnimBuffer,i);
                              pHolder[1] = DecodeByte(AnimBuffer,i);
                              pHolder[2] = DecodeByte(AnimBuffer,i);
                              ucoord = ((holder & 0x00FFF000) >> 12);
                              vcoord = (holder & 0x00000FFF);
                              float fU = ScaleUp(ucoord,
                                          -8.0f,8.0f,BPC_12);

                              float fV = ScaleUp(vcoord,
                                          -8.0f,8.0f,BPC_12);

                              floatlist[loop*2  ] = fU;
                              floatlist[loop*2+1] = fV;
                        }
                        if (Curr)
                        {

                              if (!Curr->AddTexCoords(total,floatlist))
                              {
                                    AfxMessageBox("\
Error - Attempt was made to add more vertex colors than the\n\
set maximum for a specific object model allows.");
                                    bAbortAnimation = TRUE;
                              }
                              else
                              {
                                    char buffer[200];
                                    sprintf(buffer,
                                          "%d  Texture  Coordinates  Added  to
object #%d.\r\n",total,which);
                                    pLog->Log(buffer);
                                    if (Curr->bCompleted)
                                    {
                                          char buffer[200];
                                          sprintf(buffer,
                                                "Object                    #%d
completed.\r\n",which);
                                          pLog->Log(buffer);
                                    }
                              }
                        }
                        else
                        {
                              AfxMessageBox("\
An object reference was found refering to an object not yet\n\
added to the Animation database.");
                              bAbortAnimation = TRUE;
                        }
                  } break;
            case OPCODE_ASSIGNOBJECT_SOLIDCOLOR:
                  {
```

```
                        if (i+4 >= AnimPos)
                        {
                              i = tempi;
                              bDone = TRUE;
                              break;
                        }
                        unsigned short which;
                        which = DecodeShortInt(AnimBuffer,i);
                        CVisualObject  *curr  =  Animation.FindObject(which);

                        if (!curr)
                        {
                              AfxMessageBox("\
An object reference was found refering to an object not yet\n\
added to the Animation database.");
                              bAbortAnimation = TRUE;
                              break;
                        }
                        unsigned int sourcecolor = DecodeWord(AnimBuffer,i);
                        unsigned char red,green,blue;
                        ConvertColor16To24(sourcecolor,red,green,blue);
                        curr->AssignSolidColor(red,green,blue);
                        char buffer[200];
                        sprintf(buffer,
                              "Solid  Color  (%d,%d,%d)  assigned  to  object
#%d.\r\n",
                              red,green,blue,which);
                        pLog->Log(buffer);
                  } break;
            case OPCODE_ADDOBJECT:
                  // add a new object
                  {
                        int bytelength = 6;
                        if (iBitsPerVertex != BPC_32) bytelength += 24;
                        if (i+bytelength >= AnimPos)
                        {
                              i = tempi;
                              bDone = TRUE;
                              break;
                        }
                        unsigned short TVcount;
                        unsigned short TPcount = 0;
                        unsigned short TFcount;
                        TVcount = DecodeShortInt(AnimBuffer,i);
                        TFcount = DecodeShortInt(AnimBuffer,i);
                        CVisualObject*                visObj                =
Animation.AddVisualObject(TVcount,TPcount,TFcount);
                        // If the object is defined using relative vertices,
                        // get the bounding box to use for scaling.
                        if (iBitsPerVertex != BPC_32)
                        {
                              visObj->SetMinX(DecodeFloat(AnimBuffer,i));
                              visObj->SetMinY(DecodeFloat(AnimBuffer,i));
                              visObj->SetMinZ(DecodeFloat(AnimBuffer,i));
                              visObj->SetMaxX(DecodeFloat(AnimBuffer,i));
                              visObj->SetMaxY(DecodeFloat(AnimBuffer,i));
                              visObj->SetMaxZ(DecodeFloat(AnimBuffer,i));
                        }
```

```
                        CVisualObject *curr = Animation.ObjectListHead;
                        int ObjCount = 0;
                        while(curr->next)
                        {
                              curr = curr->next;
                              ObjCount++;
                        }
                        char buffer[200];
                        sprintf(buffer,
                              "New object #%d has been created composed of %d
faces and %d vertices.\r\n",
                              ObjCount,TFcount,TVcount);
                        pLog->Log(buffer);
                        pLog->m_polys+= TFcount;
                        pLog->m_verts+= TVcount;
                        pLog->UpdateData(FALSE);
                  } break;
            case OPCODE_SETPARENTCHILD:
                  // establishes a parent/child relationship between the two
                  // specified objects
                  {
                        if (i+4 >= AnimPos)
                        {
                              i = tempi;
                              bDone = TRUE;
                              break;
                        }
                        unsigned short cnode,pnode;
                        cnode = DecodeShortInt(AnimBuffer,i);
                        pnode = DecodeShortInt(AnimBuffer,i);

                        CVisualObject  *Child  =  Animation.FindObject(cnode);

                        CVisualObject  *Parent  =  Animation.FindObject(pnode);

                        if (!Child || !Parent)
                        {
                              AfxMessageBox("\
An object reference was found refering to an object not yet\n\
added to the Animation database.");
                              bAbortAnimation = TRUE;
                              break;
                        }
                        Child->ParentObject = Parent;
                        CVisualObject *curr = Animation.ObjectListHead;
                        int iParent = 0,iChild = 0;
                        while(curr->next && curr != Parent)
                        {
                              curr = curr->next;
                              iParent++;
                        }
                        curr = Animation.ObjectListHead;
                        while(curr->next && curr != Child)
                        {
                              curr = curr->next;
                              iChild++;
                        }
                        char buffer[200];
```

```
                        sprintf(buffer,
                              "New object #%d has been named a child of object
#%d.\r\n",
                              iChild,iParent);
                        pLog->Log(buffer);
                  } break;
            case OPCODE_KEYFRAME_TCBPOS:
                  // add a new TCB position keyframe to the scene
                  {
                        if (i+2 >= AnimPos)
                        {
                              i = tempi;
                              bDone = TRUE;
                              break;
                        }
                        unsigned short which;
                        which = DecodeShortInt(AnimBuffer,i);
                        CVisualObject  *curr  =  Animation.FindObject(which);

                        if (!curr)
                        {
                              AfxMessageBox("\
An object reference was found refering to an object not yet\n\
added to the Animation database.");
                              bAbortAnimation = TRUE;
                              break;
                        }
                        if (i+25 >= AnimPos)
                        {
                              i = tempi;
                              bDone = TRUE;
                              break;
                        }

                        int frame = 0;
                        unsigned char* ucTime = (unsigned char*) &frame;
                        ucTime[0] = DecodeByte(AnimBuffer,i);
                        ucTime[1] = DecodeByte(AnimBuffer,i);
                        ucTime[2] = DecodeByte(AnimBuffer,i);
                        ucTime[3] = DecodeByte(AnimBuffer,i);

                        float pos[3] =
                        {
                              DecodeFloat(AnimBuffer,i),
                              DecodeFloat(AnimBuffer,i),
                              DecodeFloat(AnimBuffer,i),
                        };
                        float  tension  =  ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16);
                        float continuity = ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16);
                        float   bias    =   ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16);
                        float   easeIn   =   ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16);
                        float  easeOut   =   ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16);
```

```
                        curr->AddTCBPositionKey(frame,pos,tension,continuity,b
ias,easeIn,easeOut,invTPF);
                        char buffer[256];
                        sprintf(buffer,
                        "Object #%d has been assigned a TCB position
keyframe\r\n\
|     taking place at frame %d and position (%f,%f,%f) with the
parameters:\r\n\
|     tension = %f, bias = %f, continuity = %f, ease in = %f, ease out =
%f\r\n",
                              which,frame,pos[0],pos[1],pos[2],tension,continu
ity,bias,easeIn,easeOut);
                        pLog->Log(buffer);
                  } break;
            case OPCODE_KEYFRAME_LINPOS:
                  // add a new Linear Interpolation position keyframe to the
scene
                  {
                        if (i+2 >= AnimPos)
                        {
                              i = tempi;
                              bDone = TRUE;
                              break;
                        }
                        unsigned short which;
                        which = DecodeShortInt(AnimBuffer,i);
                        CVisualObject  *curr  =  Animation.FindObject(which);

                        if (!curr)
                        {
                              AfxMessageBox("\
An object reference was found refering to an object not yet\n\
added to the Animation database.");
                              bAbortAnimation = TRUE;
                              break;
                        }
                        if (i+15 >= AnimPos)
                        {
                              i = tempi;
                              bDone = TRUE;
                              break;
                        }

                        int frame = 0;
                        unsigned char* ucTime = (unsigned char*) &frame;
                        ucTime[0] = DecodeByte(AnimBuffer,i);
                        ucTime[1] = DecodeByte(AnimBuffer,i);
                        ucTime[2] = DecodeByte(AnimBuffer,i);
                        ucTime[3] = DecodeByte(AnimBuffer,i);

                        float pos[3] =
                        {
                              DecodeFloat(AnimBuffer,i),
                              DecodeFloat(AnimBuffer,i),
                              DecodeFloat(AnimBuffer,i),
                        };

                        curr->AddLinearPositionKey(frame,pos[0],pos[1],pos[2],
```

```
invTPF);
                        char buffer[256];
                        sprintf(buffer,
                              "Object #%d has been assigned a TCB position
keyframe\r\n\
|     taking place at frame %d and position (%f,%f,%f)\r\n",
                                    which,frame,pos[0],pos[1],pos[2]);
                        pLog->Log(buffer);
                  } break;
            case OPCODE_KEYFRAME_BEZPOS:
                  // add a new TCB position keyframe to the scene
                  {
                        if (i+2 >= AnimPos)
                        {
                              i = tempi;
                              bDone = TRUE;
                              break;
                        }
                        unsigned short which;
                        which = DecodeShortInt(AnimBuffer,i);
                        CVisualObject  *curr  =  Animation.FindObject(which);

                        if (!curr)
                        {
                              AfxMessageBox("\
An object reference was found refering to an object not yet\n\
added to the Animation database.");
                              bAbortAnimation = TRUE;
                              break;
                        }
                        if (i+39 >= AnimPos)
                        {
                              i = tempi;
                              bDone = TRUE;
                              break;
                        }

                        int frame = 0;
                        unsigned char* ucTime = (unsigned char*) &frame;
                        ucTime[0] = DecodeByte(AnimBuffer,i);
                        ucTime[1] = DecodeByte(AnimBuffer,i);
                        ucTime[2] = DecodeByte(AnimBuffer,i);
                        ucTime[3] = DecodeByte(AnimBuffer,i);

                        float pos[3] =
                        {
                              DecodeFloat(AnimBuffer,i),
                              DecodeFloat(AnimBuffer,i),
                              DecodeFloat(AnimBuffer,i),
                        };
                        float inTan[3] =
                        {
                              DecodeFloat(AnimBuffer,i),
                              DecodeFloat(AnimBuffer,i),
                              DecodeFloat(AnimBuffer,i),
                        };
                        float outTan[3] =
                        {
```

```
                              DecodeFloat(AnimBuffer,i),
                              DecodeFloat(AnimBuffer,i),
                              DecodeFloat(AnimBuffer,i),
                        };

                        curr->AddBezierPositionKey(frame,pos,inTan,outTan,invT
PF);
                        char buffer[256];
                        sprintf(buffer,
                        "Object #%d has been assigned a Bezier position
keyframe\r\n\
|     taking place at frame %d and position (%f,%f,%f) with the
parameters:\r\n\
|     inTan = (%f,%f,%f),  outTan = (%f,%f,%f).\r\n",
                              which,frame,pos[0],pos[1],pos[2],inTan[0],inTan[
1],
                              inTan[2],outTan[0],outTan[1],outTan[2]);
                        pLog->Log(buffer);
                  } break;
            case OPCODE_KEYFRAME_TCBROT:
                  // add a new TCB rotation keyframe to the scene
                  {
                        if (i+2 >= AnimPos)
                        {
                              i = tempi;
                              bDone = TRUE;
                              break;
                        }
                        unsigned short which;
                        which = DecodeShortInt(AnimBuffer,i);

                        if (i+11 >=AnimPos)
                        {
                              i = tempi;
                              bDone = TRUE;
                              break;
                        }

                        int frame = 0;
                        unsigned char* ucTime = (unsigned char*) &frame;
                        ucTime[0] = DecodeByte(AnimBuffer,i);
                        ucTime[1] = DecodeByte(AnimBuffer,i);
                        ucTime[2] = DecodeByte(AnimBuffer,i);
                        ucTime[3] = DecodeByte(AnimBuffer,i);
                        CVisualObject  *curr  =  Animation.FindObject(which);

                        if (!curr)
                        {
                              AfxMessageBox("\
An object reference was found refering to an object not yet\n\
added to the Animation database.");
                              bAbortAnimation = TRUE;
                        }
                        else
                        {
                              float rot[4] =
                              {
                                    DecodeFloat(AnimBuffer,i),
```

```
                                    -ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16),
                                    -ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16),
                                    -ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16),
                              };
                              float                tension                =
ScaleUp(DecodeWord(AnimBuffer,i),-1.0f,1.0f,BPC_16);
                              float              continuity               =
ScaleUp(DecodeWord(AnimBuffer,i),-1.0f,1.0f,BPC_16);
                              float bias = ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16);
                              float                easeIn                 =
ScaleUp(DecodeWord(AnimBuffer,i),-1.0f,1.0f,BPC_16);
                              float               easeOut                 =
ScaleUp(DecodeWord(AnimBuffer,i),-1.0f,1.0f,BPC_16);
                              curr->AddTCBRotationKey(frame,rot,tension,contin
uity,
                                    bias,easeIn,easeOut,invTPF);
                              char buffer[256];
                              sprintf(buffer,
                                    "Object #%d has been assigned a TCB
rotation keyframe\r\n\
|     taking place at frame %d with Angle Axis value: %f degrees
(%f,%f,%f)\r\n",
                              which,frame,rot[0]*FLOAT_RAD_TO_DEG,rot[1],rot[2
],rot[3]);
                              pLog->Log(buffer);
                        }
                  } break;
            case OPCODE_KEYFRAME_LINROT:
                  // add a new Linear rotation keyframe to the scene
                  {
                        if (i+2 >= AnimPos)
                        {
                              i = tempi;
                              bDone = TRUE;
                              break;
                        }
                        unsigned short which;
                        which = DecodeShortInt(AnimBuffer,i);

                        if (i+11 >=AnimPos)
                        {
                              i = tempi;
                              bDone = TRUE;
                              break;
                        }

                        int frame = 0;
                        unsigned char* ucTime = (unsigned char*) &frame;
                        ucTime[0] = DecodeByte(AnimBuffer,i);
                        ucTime[1] = DecodeByte(AnimBuffer,i);
                        ucTime[2] = DecodeByte(AnimBuffer,i);
                        ucTime[3] = DecodeByte(AnimBuffer,i);
                        CVisualObject *curr = Animation.FindObject(which);
```

```
                        if (!curr)
                        {
                              AfxMessageBox("\
An object reference was found refering to an object not yet\n\
added to the Animation database.");
                              bAbortAnimation = TRUE;
                        }
                        else
                        {
                              float rot[4] =
                              {
                                    ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16),
                                    ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16),
                                    ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16),
                                    ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16),
                              };
                              curr->AddLinearRotationKey(frame,rot,invTPF);
                              char buffer[256];
                              sprintf(buffer, "Object #%d has been assigned a
Linear rotation keyframe\r\n\
|     taking place at frame %d with quaternion (%f,%f,%f,%f)\r\n",
                              which,frame,rot[0],rot[1],rot[2],rot[3]);
                              pLog->Log(buffer);
                        }
                  } break;
/*          case OPCODE_KEYFRAME_BEZROT:  Unnecessary
                  // add a new Bezier rotation keyframe to the scene
                  {
                        if (i+2 >= AnimPos)
                        {
                              i = tempi;
                              bDone = TRUE;
                              break;
                        }
                        unsigned short which;
                        which = DecodeShortInt(AnimBuffer,i);

                        if (i+11 >=AnimPos)
                        {
                              i = tempi;
                              bDone = TRUE;
                              break;
                        }

                        int frame = 0;
                        unsigned char* ucTime = (unsigned char*) &frame;
                        ucTime[0] = DecodeByte(AnimBuffer,i);
                        ucTime[1] = DecodeByte(AnimBuffer,i);
                        ucTime[2] = DecodeByte(AnimBuffer,i);
                        ucTime[3] = DecodeByte(AnimBuffer,i);
                        CVisualObject   *curr   =   Animation.FindObject(which);

                        if (!curr)
```

```
                        {
                                AfxMessageBox("\
An object reference was found refering to an object not yet\n\
added to the Animation database.");
                                bAbortAnimation = TRUE;
                        }
                        else
                        {
                                float rot[4] =
                                {
                                        ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16),
                                        ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16),
                                        ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16),
                                        ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16),
                                };
                                curr->AddBezierRotationKey(frame,rot,invTPF);
                                char buffer[256];
                                sprintf(buffer,
                                "Object #%d has been assigned a Bezier rotation
keyframe\r\n\
|       taking place at frame %d with quaternion (%f,%f,%f,%f)\r\n",
                                which,frame,rot[0],rot[1],rot[2],rot[3]);
                                pLog->Log(buffer);
                        }
                } break;*/
        case OPCODE_KEYFRAME_TCBSCL:
                // add a new TCB scale keyframe to the scene
                {
                        if (i+2 >= AnimPos)
                        {
                                i = tempi;
                                bDone = TRUE;
                                break;
                        }
                        unsigned short which;
                        which = DecodeShortInt(AnimBuffer,i);
                        CVisualObject   *curr   =   Animation.FindObject(which);

                        if (!curr)
                        {
                                AfxMessageBox("\
An object reference was found refering to an object not yet\n\
added to the Animation database.");
                                bAbortAnimation = TRUE;
                                break;
                        }
                        if (i+25 >= AnimPos)
                        {
                                i = tempi;
                                bDone = TRUE;
                                break;
                        }
```

```
                        int frame = 0;
                        unsigned char* ucTime = (unsigned char*) &frame;
                        ucTime[0] = DecodeByte(AnimBuffer,i);
                        ucTime[1] = DecodeByte(AnimBuffer,i);
                        ucTime[2] = DecodeByte(AnimBuffer,i);
                        ucTime[3] = DecodeByte(AnimBuffer,i);

                        float scale[3] =
                        {
                              DecodeFloat(AnimBuffer,i),
                              DecodeFloat(AnimBuffer,i),
                              DecodeFloat(AnimBuffer,i),
                        };
                        float  tension  =  ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16);
                        float continuity = ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16);
                        float   bias    =   ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16);
                        float   easeIn   =   ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16);
                        float   easeOut   =   ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16);

                        curr->AddTCBScaleKey(frame,scale,tension,continuity,bi
as,easeIn,easeOut,invTPF);
                        char buffer[256];
                        sprintf(buffer,
                        "Object   #%d   has   been   assigned   a   TCB   scale
keyframe\r\n\
|     taking place at frame %d and scale (%f,%f,%f) with the parameters:\r\n\
|     tension = %f, bias = %f, continuity = %f, ease in = %f, ease out =
%f\r\n",
                              which,frame,scale[0],scale[1],scale[2],tension,c
ontinuity,bias,easeIn,easeOut);
                        pLog->Log(buffer);
                  } break;
            case OPCODE_KEYFRAME_LINSCL:
                  // add a new Linear Interpolation scale keyframe to the
scene
                  {
                        if (i+2 >= AnimPos)
                        {
                              i = tempi;
                              bDone = TRUE;
                              break;
                        }
                        unsigned short which;
                        which = DecodeShortInt(AnimBuffer,i);
                        CVisualObject  *curr  =  Animation.FindObject(which);

                        if (!curr)
                        {
                              AfxMessageBox("\
An object reference was found refering to an object not yet\n\
added to the Animation database.");
                              bAbortAnimation = TRUE;
                              break;
```

```
                        }
                        if (i+15 >= AnimPos)
                        {
                              i = tempi;
                              bDone = TRUE;
                              break;
                        }

                        int frame = 0;
                        unsigned char* ucTime = (unsigned char*) &frame;
                        ucTime[0] = DecodeByte(AnimBuffer,i);
                        ucTime[1] = DecodeByte(AnimBuffer,i);
                        ucTime[2] = DecodeByte(AnimBuffer,i);
                        ucTime[3] = DecodeByte(AnimBuffer,i);

                        float scale[3] =
                        {
                              DecodeFloat(AnimBuffer,i),
                              DecodeFloat(AnimBuffer,i),
                              DecodeFloat(AnimBuffer,i),
                        };

                        curr->AddLinearScaleKey(frame,scale[0],scale[1],scale[
2],invTPF);
                        char buffer[256];
                        sprintf(buffer,
                              "Object #%d has been assigned a TCB scale
keyframe\r\n\
|     taking place at frame %d and scale (%f,%f,%f)\r\n",
                                    which,frame,scale[0],scale[1],scale[2]);
                        pLog->Log(buffer);
                  } break;
            case OPCODE_KEYFRAME_BEZSCL:
                  // add a new Bezier scaling keyframe to the scene
                  {
                        if (i+2 >= AnimPos)
                        {
                              i = tempi;
                              bDone = TRUE;
                              break;
                        }
                        unsigned short which;
                        which = DecodeShortInt(AnimBuffer,i);
                        CVisualObject  *curr  =  Animation.FindObject(which);

                        if (!curr)
                        {
                              AfxMessageBox("\
An object reference was found refering to an object not yet\n\
added to the Animation database.");
                              bAbortAnimation = TRUE;
                              break;
                        }
                        if (i+39 >= AnimPos)
                        {
                              i = tempi;
                              bDone = TRUE;
                              break;
```

```
                        }

                        int frame = 0;
                        unsigned char* ucTime = (unsigned char*) &frame;
                        ucTime[0] = DecodeByte(AnimBuffer,i);
                        ucTime[1] = DecodeByte(AnimBuffer,i);
                        ucTime[2] = DecodeByte(AnimBuffer,i);
                        ucTime[3] = DecodeByte(AnimBuffer,i);

                        float scale[3] =
                        {
                              DecodeFloat(AnimBuffer,i),
                              DecodeFloat(AnimBuffer,i),
                              DecodeFloat(AnimBuffer,i),
                        };
                        float inTan[3] =
                        {
                              DecodeFloat(AnimBuffer,i),
                              DecodeFloat(AnimBuffer,i),
                              DecodeFloat(AnimBuffer,i),
                        };
                        float outTan[3] =
                        {
                              DecodeFloat(AnimBuffer,i),
                              DecodeFloat(AnimBuffer,i),
                              DecodeFloat(AnimBuffer,i),
                        };

                        curr->AddBezierScaleKey(frame,scale,inTan,outTan,invTP
F);

                        char buffer[256];
                        sprintf(buffer,
                        "Object  #%d  has  been  assigned  a  Bezier  scaling
keyframe\r\n\
|     taking place at frame %d and scale (%f,%f,%f) with the parameters:\r\n\
|     inTan = (%f,%f,%f),  outTan = (%f,%f,%f).\r\n",
                              which,frame,scale[0],scale[1],scale[2],inTan[0],
inTan[1],
                              inTan[2],outTan[0],outTan[1],outTan[2]);
                        pLog->Log(buffer);
                  } break;
// CAMERA STUFF
            case OPCODE_CAMERA_KEYFRAME_TCBPOS:
                  // add a new TCB position keyframe to the scene
                  {
                        if (i+25 >= AnimPos)
                        {
                              i = tempi;
                              bDone = TRUE;
                              break;
                        }

                        int frame = 0;
                        unsigned char* ucTime = (unsigned char*) &frame;
                        ucTime[0] = DecodeByte(AnimBuffer,i);
                        ucTime[1] = DecodeByte(AnimBuffer,i);
                        ucTime[2] = DecodeByte(AnimBuffer,i);
                        ucTime[3] = DecodeByte(AnimBuffer,i);
```

```
                        float pos[3] =
                        {
                              DecodeFloat(AnimBuffer,i),
                              DecodeFloat(AnimBuffer,i),
                              DecodeFloat(AnimBuffer,i),
                        };
                        float  tension  =  ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16);
                        float continuity = ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16);
                        float   bias    =   ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16);
                        float   easeIn   =   ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16);
                        float   easeOut   =   ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16);

                        Animation.SceneCamera.AddTCBPositionKey(frame,pos,tens
ion,continuity,bias,easeIn,easeOut,invTPF);
                        char buffer[256];
                        sprintf(buffer,
                        "Camera has been assigned a TCB position keyframe\r\n\
|     taking   place   at   frame   %d   and   position   (%f,%f,%f)   with   the
parameters:\r\n\
|     tension = %f, bias = %f, continuity = %f, ease in = %f, ease out =
%f\r\n",
                              frame,pos[0],pos[1],pos[2],tension,continuity,bi
as,easeIn,easeOut);
                        pLog->Log(buffer);
                  } break;
            case OPCODE_CAMERA_KEYFRAME_LINPOS:
                  // add a new Linear Interpolation position keyframe to the
scene
                  {
                        if (i+15 >= AnimPos)
                        {
                              i = tempi;
                              bDone = TRUE;
                              break;
                        }

                        int frame = 0;
                        unsigned char* ucTime = (unsigned char*) &frame;
                        ucTime[0] = DecodeByte(AnimBuffer,i);
                        ucTime[1] = DecodeByte(AnimBuffer,i);
                        ucTime[2] = DecodeByte(AnimBuffer,i);
                        ucTime[3] = DecodeByte(AnimBuffer,i);

                        float pos[3] =
                        {
                              DecodeFloat(AnimBuffer,i),
                              DecodeFloat(AnimBuffer,i),
                              DecodeFloat(AnimBuffer,i),
                        };

                        Animation.SceneCamera.AddLinearPositionKey(frame,pos[0
],pos[1],pos[2],invTPF);
```

```
                        char buffer[256];
                        sprintf(buffer,
                        "Camera has been assigned a TCB position keyframe\r\n\
|     taking place at frame %d and position (%f,%f,%f)\r\n",
                              frame,pos[0],pos[1],pos[2]);
                        pLog->Log(buffer);
                  } break;
            case OPCODE_CAMERA_KEYFRAME_BEZPOS:
                  // add a new TCB position keyframe to the scene
                  {
                        if (i+40 >= AnimPos)
                        {
                              i = tempi;
                              bDone = TRUE;
                              break;
                        }

                        int frame = 0;
                        unsigned char* ucTime = (unsigned char*) &frame;
                        ucTime[0] = DecodeByte(AnimBuffer,i);
                        ucTime[1] = DecodeByte(AnimBuffer,i);
                        ucTime[2] = DecodeByte(AnimBuffer,i);
                        ucTime[3] = DecodeByte(AnimBuffer,i);

                        float pos[3] =
                        {
                              DecodeFloat(AnimBuffer,i),
                              DecodeFloat(AnimBuffer,i),
                              DecodeFloat(AnimBuffer,i),
                        };
                        float inTan[3] =
                        {
                              DecodeFloat(AnimBuffer,i),
                              DecodeFloat(AnimBuffer,i),
                              DecodeFloat(AnimBuffer,i),
                        };
                        float outTan[3] =
                        {
                              DecodeFloat(AnimBuffer,i),
                              DecodeFloat(AnimBuffer,i),
                              DecodeFloat(AnimBuffer,i),
                        };

                        Animation.SceneCamera.AddBezierPositionKey(frame,pos,i
nTan,outTan,invTPF);
                        char buffer[256];
                        sprintf(buffer,
                              "Camera has been assigned a Bezier position
keyframe\r\n\
|     taking place at frame %d and position (%f,%f,%f) with the
parameters:\r\n\
|     inTan = (%f,%f,%f),  outTan = (%f,%f,%f).\r\n",
                              frame,pos[0],pos[1],pos[2],inTan[0],inTan[1],inT
an[2],outTan[0],outTan[1],outTan[2]);
                        pLog->Log(buffer);
                  } break;
            case OPCODE_CAMERA_KEYFRAME_TCBROT:
                  // add a new TCB rotation keyframe to the scene
```

```
                {
                        if (i+11 >=AnimPos)
                        {
                                i = tempi;
                                bDone = TRUE;
                                break;
                        }

                        int frame = 0;
                        unsigned char* ucTime = (unsigned char*) &frame;
                        ucTime[0] = DecodeByte(AnimBuffer,i);
                        ucTime[1] = DecodeByte(AnimBuffer,i);
                        ucTime[2] = DecodeByte(AnimBuffer,i);
                        ucTime[3] = DecodeByte(AnimBuffer,i);
                        float rot[4] =
                        {
                                DecodeFloat(AnimBuffer,i),
                                ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16),
                                ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16),
                                ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16),
                        };
                        float  tension  =  ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16);
                        float continuity = ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16);
                        float   bias    =   ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16);
                        float   easeIn   =   ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16);
                        float   easeOut   =   ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16);
                        Animation.SceneCamera.AddTCBRotationKey(frame,rot,tens
ion,continuity,
                                        bias,easeIn,easeOut,invTPF);
                        char buffer[256];
                        sprintf(buffer,
                        "Camera has been assigned a TCB rotation keyframe\r\n\
|     taking   place   at   frame   %d   with   Angle   Axis   value:   %f   degrees
(%f,%f,%f)\r\n",
                                frame,rot[0]*FLOAT_RAD_TO_DEG,rot[1],rot[2],rot[
3]);
                        pLog->Log(buffer);
                } break;
        case OPCODE_CAMERA_KEYFRAME_LINROT:
                // add a new TCB rotation keyframe to the scene
                {
                        if (i+11 >=AnimPos)
                        {
                                i = tempi;
                                bDone = TRUE;
                                break;
                        }

                        int frame = 0;
                        unsigned char* ucTime = (unsigned char*) &frame;
```

```
                        ucTime[0] = DecodeByte(AnimBuffer,i);
                        ucTime[1] = DecodeByte(AnimBuffer,i);
                        ucTime[2] = DecodeByte(AnimBuffer,i);
                        ucTime[3] = DecodeByte(AnimBuffer,i);
                        float rot[4] =
                        {
                              ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16),
                              ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16),
                              ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16),
                              ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16),
                        };
                        Animation.SceneCamera.AddLinearRotationKey(frame,rot,i
nvTPF);
                        char buffer[256];
                        sprintf(buffer, "Camera has been assigned a Linear
rotation keyframe\r\n\
|     taking place at frame %d with quaternion (%f,%f,%f,%f)\r\n",
                              frame,rot[0],rot[1],rot[2],rot[3]);
                        pLog->Log(buffer);
                  } break;
/*          case OPCODE_CAMERA_KEYFRAME_BEZROT: // Unnecessary
                  // add a new TCB rotation keyframe to the scene
                  {
                        if (i+11 >=AnimPos)
                        {
                              i = tempi;
                              bDone = TRUE;
                              break;
                        }

                        int frame = 0;
                        unsigned char* ucTime = (unsigned char*) &frame;
                        ucTime[0] = DecodeByte(AnimBuffer,i);
                        ucTime[1] = DecodeByte(AnimBuffer,i);
                        ucTime[2] = DecodeByte(AnimBuffer,i);
                        ucTime[3] = DecodeByte(AnimBuffer,i);
                        float rot[4] =
                        {
                              ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16),
                              ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16),
                              ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16),
                              ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16),
                        };
                        Animation.SceneCamera.AddBezierRotationKey(frame,rot,i
nvTPF);
                        char buffer[256];
                        sprintf(buffer,
                        "Camera   has   been   assigned   a   Bezier   rotation
keyframe\r\n\
```

```
|     taking place at frame %d with quaternion (%f,%f,%f,%f)\r\n",
                              frame,rot[0],rot[1],rot[2],rot[3]);
                        pLog->Log(buffer);
                  } break;*/
            case OPCODE_CAMERA_KEYFRAME_BEZRLL:
                  {
                        if (i+7 >=AnimPos)
                        {
                              i = tempi;
                              bDone = TRUE;
                              break;
                        }

                        int frame = 0;
                        unsigned char* ucTime = (unsigned char*) &frame;
                        ucTime[0] = DecodeByte(AnimBuffer,i);
                        ucTime[1] = DecodeByte(AnimBuffer,i);
                        ucTime[2] = DecodeByte(AnimBuffer,i);
                        ucTime[3] = DecodeByte(AnimBuffer,i);
                        float roll = DecodeFloat(AnimBuffer,i);
                        char buffer[256];
                        sprintf(buffer,
                        "Camera has been assigned a Bezier roll keyframe\r\n\
|     taking place at frame %d with angle %f\r\n",
                              frame,roll);
                        pLog->Log(buffer);
                  } break;
// END CAMERA STUFF
            case OPCODE_ADDCAMERA:  // specifies a camera field of view, near
& far planes. Absolutes,
                              // and not attached to a keyframe, so it's
recommended this be done
                              // as early as possible if the defaults aren't
used.
                  {
                        if (i+12 >= AnimPos)
                        {
                              i = tempi;
                              bDone = TRUE;
                              break;
                        }
                        Animation.SceneCamera.fFieldOfView = ScaleUp(
                              DecodeWord(AnimBuffer,i),0.0f,FLOAT_PI,BPC_16);
                        Animation.SceneCamera.fNearPlane                    =
DecodeFloat(AnimBuffer,i);
                        if (Animation.SceneCamera.fNearPlane == 0.0f)
                              Animation.SceneCamera.fNearPlane = 1.0f;
                        Animation.SceneCamera.fFarPlane                     =
DecodeFloat(AnimBuffer,i);
                        char buffer[256];
                        sprintf(buffer,
                              "Camera added to scene with FOV = %f, near plane
= %f, far plane = %f\r\n",
                              Animation.SceneCamera.fFieldOfView*FLOAT_RAD_TO_
DEG,Animation.SceneCamera.fNearPlane,
                              Animation.SceneCamera.fFarPlane);
                        pLog->Log(buffer);
                  } break;
```

```
            case OPCODE_ADDCAMERA_WITHLOCATION:
                        // Specifies a camera field of view, near & far
planes.
                        // This also adds in the location and orientation
of the
                        // camera as this camera has no keyframes attached.
                  {
                        if (i+30 >= AnimPos)
                        {
                              i = tempi;
                              bDone = TRUE;
                              break;
                        }
                        Animation.SceneCamera.fFieldOfView = ScaleUp(
                              DecodeWord(AnimBuffer,i),0.0f,FLOAT_PI,BPC_16);
                        Animation.SceneCamera.fNearPlane                    =
DecodeFloat(AnimBuffer,i);
                        if (Animation.SceneCamera.fNearPlane == 0.0f)
                              Animation.SceneCamera.fNearPlane = 1.0f;
                        Animation.SceneCamera.fFarPlane                     =
DecodeFloat(AnimBuffer,i);

                        Animation.SceneCamera.PosX                          =
DecodeFloat(AnimBuffer,i);
                        Animation.SceneCamera.PosY                          =
DecodeFloat(AnimBuffer,i);
                        Animation.SceneCamera.PosZ                          =
DecodeFloat(AnimBuffer,i);

                        Animation.SceneCamera.OriX = 0.0f;
                        Animation.SceneCamera.OriY = 0.0f;
                        Animation.SceneCamera.OriZ = 0.0f;
                        Animation.SceneCamera.OriW = 1.0f;

                        char buffer[256];
                        sprintf(buffer, "Camera added to scene with FOV = %f,
near plane = %f, far plane = %f\r\n\
|     Position = (%f,%f,%f)\r\n",
                              Animation.SceneCamera.fFieldOfView*FLOAT_RAD_TO_
DEG,
                              Animation.SceneCamera.fNearPlane,
                              Animation.SceneCamera.fFarPlane,Animation.SceneC
amera.PosX,
                              Animation.SceneCamera.PosY,Animation.SceneCamera
.PosZ);
                        pLog->Log(buffer);
                  } break;
            case OPCODE_SETCAMERA_ORIENTATION:
                  {
                        if (i+8 >= AnimPos)
                        {
                              i = tempi;
                              bDone = TRUE;
                              break;
                        }
                        Animation.SceneCamera.OriX = ScaleUp(
                              DecodeWord(AnimBuffer,i),-1.0f,1.0f,BPC_16);
                        Animation.SceneCamera.OriY = ScaleUp(
```

```
                        DecodeWord(AnimBuffer,i),-1.0f,1.0f,BPC_16);
                    Animation.SceneCamera.OriZ = ScaleUp(
                        DecodeWord(AnimBuffer,i),-1.0f,1.0f,BPC_16);
                    Animation.SceneCamera.OriW = ScaleUp(
                        DecodeWord(AnimBuffer,i),-1.0f,1.0f,BPC_16);

                    char buffer[256];
                    sprintf(buffer, "Camera set to initial orientation
(%f,%f,%f,%f).\r\n",
                        Animation.SceneCamera.OriX,Animation.SceneCamera
.OriY,
                        Animation.SceneCamera.OriZ,Animation.SceneCamera
.OriW);
                    pLog->Log(buffer);
                } break;
            case OPCODE_SETCAMERA_OFFSET:
                {
                    if (i+48 >= AnimPos)
                    {
                        i = tempi;
                        bDone = TRUE;
                        break;
                    }
                    float offset[12];
                    for (int loop=0;loop<12;loop++)
                        offset[loop] = DecodeFloat(AnimBuffer,i);

                    Animation.SceneCamera.SetOffset(offset);

                    char buffer[200];
                    sprintf(buffer,
                        "SceneCamera has been offset to
[[%f,%f,%f][%f,%f,%f][%f,%f,%f][%f,%f,%f]].\r\n",
                        offset[0],offset[1],offset[2],offset[3],
                        offset[4],offset[5],offset[6],offset[7],
                        offset[8],offset[9],offset[10],offset[11]);
                    pLog->Log(buffer);
                } break;
            case OPCODE_SETOFFSET:
                {
                    if (i+2 >= AnimPos)
                    {
                        i = tempi;
                        bDone = TRUE;
                        break;
                    }
                    unsigned short which;
                    which = DecodeShortInt(AnimBuffer,i);
                    if (i+48 >= AnimPos)
                    {
                        i = tempi;
                        bDone = TRUE;
                        break;
                    }
                    CVisualObject *curr = Animation.FindObject(which);

                    if (!curr)
                    {
```

```
                                    AfxMessageBox("\
An object reference was found refering to an object not yet\n\
added to the Animation database while trying to set an Offset.");
                                    bAbortAnimation = TRUE;
                              }
                              else
                              {
                                    float offset[12];
                                    for (int loop=0;loop<12;loop++)
                                          offset[loop] = DecodeFloat(AnimBuffer,i);

                                    curr->SetOffset(offset);

                                    char buffer[200];
                                    sprintf(buffer,
                                          "Object   #%d   has   been   offset   to
[[%f,%f,%f][%f,%f,%f][%f,%f,%f][%f,%f,%f]].\r\n",
                                          which,offset[0],offset[1],offset[2],offset
[3],
                                          offset[4],offset[5],offset[6],offset[7],
                                          offset[8],offset[9],offset[10],offset[11])
;
                                    pLog->Log(buffer);
                              }
                        } break;
                  case OPCODE_SETPOSITION:
                        // sets an absolute position for a given object
                        {
                              if (i+2 >= AnimPos)
                              {
                                    i = tempi;
                                    bDone = TRUE;
                                    break;
                              }
                              unsigned short which;
                              which = DecodeShortInt(AnimBuffer,i);
                              if (i+12 >= AnimPos)
                              {
                                    i = tempi;
                                    bDone = TRUE;
                                    break;
                              }
                              CVisualObject  *curr  =  Animation.FindObject(which);

                              if (!curr)
                              {
                                    AfxMessageBox("\
An object reference was found refering to an object not yet\n\
added to the Animation database.");
                                    bAbortAnimation = TRUE;
                              }
                              else
                              {
                                    float pos[3] =
                                    {
                                          DecodeFloat(AnimBuffer,i),
                                          DecodeFloat(AnimBuffer,i),
                                          DecodeFloat(AnimBuffer,i),
```

```
                        };

                        curr->SetPosition(pos[0],pos[1],pos[2]);

                        char buffer[200];
                        sprintf(buffer,
                              "Object  #%d  has  been  positioned  at
(%f,%f,%f).\r\n",
                              which,pos[0],pos[1],pos[2]);
                        pLog->Log(buffer);
                  }
            } break;
      case OPCODE_SETROTATION:
            // sets an absolute orientation for a given object
            {
                  if (i+10 >= AnimPos)
                  {
                        i = tempi;
                        bDone = TRUE;
                        break;
                  }
                  unsigned short which;
                  which = DecodeShortInt(AnimBuffer,i);
                  CVisualObject  *curr  =  Animation.FindObject(which);

                  if (!curr)
                  {
                        AfxMessageBox("\
An object reference was found refering to an object not yet\n\
added to the Animation database.");
                        bAbortAnimation = TRUE;
                  }
                  else
                  {
                        float rot[4] =
                        {
                              ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16),
                              ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16),
                              ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16),
                              ScaleUp(DecodeWord(AnimBuffer,i),-
1.0f,1.0f,BPC_16),
                        };

                        curr->SetOrientation(rot);

                        char buffer[200];
                        sprintf(buffer,
                              "Object  #%d  has  been  oriented  to
(%f,%f,%f,%f).\r\n",
                              which,rot[0],rot[1],rot[2],rot[3]);
                        pLog->Log(buffer);
                  }
            } break;
      case OPCODE_SETSCALE:
            // sets an absolute scale for a given object
```

```
                {
                        if (i+14 >= AnimPos)
                        {
                                i = tempi;
                                bDone = TRUE;
                                break;
                        }
                        unsigned short which;
                        which = DecodeShortInt(AnimBuffer,i);
                        CVisualObject   *curr   =   Animation.FindObject(which);

                        if (!curr)
                        {
                                AfxMessageBox("\
An object reference was found refering to an object not yet\n\
added to the Animation database.");
                                bAbortAnimation = TRUE;
                        }
                        else
                        {
                                float scale[3] =
                                {
                                        DecodeFloat(AnimBuffer,i),
                                        DecodeFloat(AnimBuffer,i),
                                        DecodeFloat(AnimBuffer,i)
                                };

                                curr->SetScale(scale[0],scale[1],scale[2]);

                                char buffer[200];
                                sprintf(buffer,
                                        "Object    #%d    has    been    scaled    to
(%f,%f,%f).\r\n",
                                        which,scale[0],scale[1],scale[2]);
                                pLog->Log(buffer);
                        }
                } break;
//
// ADD LIGHTS HERE
//
            case OPCODE_ADDLIGHT_OMNI:
                {
                        if (i+2 >= AnimPos)
                        {
                                i = tempi;
                                bDone = TRUE;
                                break;
                        }
                        unsigned short sourcecolor = DecodeWord(AnimBuffer,i);
                        unsigned char red,green,blue;
                        if (Animation.iLightsInUse < GL_MAX_LIGHTS)
                        {
                                ConvertColor16To24(sourcecolor,red,green,blue);
                                Animation.SceneLights[Animation.iLightsInUse].Co
lor[0] =
                                        (float) red / 255.0f;
                                Animation.SceneLights[Animation.iLightsInUse].Co
lor[1] =
```

```
                                    (float) green / 255.0f;
                                Animation.SceneLights[Animation.iLightsInUse].Co
lor[2] =
                                    (float) blue / 255.0f;
                                Animation.SceneLights[Animation.iLightsInUse].Co
lor[3] = 1.0f;
                                Animation.SceneLights[Animation.iLightsInUse++].
Type = LIGHT_POINT;
                                char buffer[200];
                                sprintf(buffer,
                                    "Point Light #%d has been added to the
scene with color (%f,%f,%f).\r\n",
                                    Animation.iLightsInUse,red,green,blue);
                                pLog->Log(buffer);
                            }
                        } break;
                    case OPCODE_ADDLIGHT_SPOT:
                        {
                            if (i+2 >= AnimPos)
                            {
                                i = tempi;
                                bDone = TRUE;
                                break;
                            }
                            unsigned short sourcecolor = DecodeWord(AnimBuffer,i);
                            unsigned char red,green,blue;
                            if (Animation.iLightsInUse < GL_MAX_LIGHTS)
                            {
                                ConvertColor16To24(sourcecolor,red,green,blue);
                                Animation.SceneLights[Animation.iLightsInUse].Co
lor[0] =
                                    (float) red / 255.0f;
                                Animation.SceneLights[Animation.iLightsInUse].Co
lor[1] =
                                    (float) green / 255.0f;
                                Animation.SceneLights[Animation.iLightsInUse].Co
lor[2] =
                                    (float) blue / 255.0f;
                                Animation.SceneLights[Animation.iLightsInUse].Co
lor[3] = 1.0f;
                                Animation.SceneLights[Animation.iLightsInUse++].
Type = LIGHT_POINT;
                                char buffer[200];
                                sprintf(buffer,
                                    "SpotLight #%d has been added to the scene
with color (%f,%f,%f).\r\n",
                                    Animation.iLightsInUse,red,green,blue);
                                pLog->Log(buffer);
                            }
                        } break;
                    case OPCODE_ADDLIGHT_DIRECTIONAL:
                        {
                            if (i+2 >= AnimPos)
                            {
                                i = tempi;
                                bDone = TRUE;
                                break;
                            }
```

```
                        unsigned short sourcecolor = DecodeWord(AnimBuffer,i);
                        unsigned char red,green,blue;
                        if (Animation.iLightsInUse < GL_MAX_LIGHTS)
                        {
                              ConvertColor16To24(sourcecolor,red,green,blue);
                              Animation.SceneLights[Animation.iLightsInUse].Co
lor[0] =
                                    (float) red / 255.0f;
                              Animation.SceneLights[Animation.iLightsInUse].Co
lor[1] =
                                    (float) green / 255.0f;
                              Animation.SceneLights[Animation.iLightsInUse].Co
lor[2] =
                                    (float) blue / 255.0f;
                              Animation.SceneLights[Animation.iLightsInUse].Co
lor[3] = 1.0f;
                              Animation.SceneLights[Animation.iLightsInUse++].
Type = LIGHT_POINT;
                              char buffer[200];
                              sprintf(buffer,
                                    "Directional Light #%d has been added to
the scene with color (%f,%f,%f).\r\n",
                                    Animation.iLightsInUse,red,green,blue);
                              pLog->Log(buffer);
                        }
                  } break;
            case OPCODE_ADDTARGET_TO_CAMERA:
                  {
                        if (i+2 >= AnimPos)
                        {
                              i = tempi;
                              bDone = TRUE;
                              break;
                        }
                        unsigned int which = DecodeShortInt(AnimBuffer,i);
                        CVisualObject  *curr  =  Animation.FindObject(which);

                        if (!curr)
                        {
                              AfxMessageBox("\
An object reference was found refering to an object not yet\n\
added to the Animation database.");
                              bAbortAnimation = TRUE;
                        }
                        else
                        {
                              Animation.SceneCamera.Target = curr;
                              char buffer[200];
                              sprintf(buffer,
                                    "SceneCamera has been assigned object #%d
as a target.\r\n",
                                    which);
                              pLog->Log(buffer);
                        }
                  } break;
            case OPCODE_ADDTARGET_TO_OBJECT:
                  {
                        if (i+4 >= AnimPos)
```

```
                        {
                              i = tempi;
                              bDone = TRUE;
                              break;
                        }
                        unsigned int source = DecodeShortInt(AnimBuffer,i);
                        unsigned int target = DecodeShortInt(AnimBuffer,i);
                        CVisualObject  *scurr  =  Animation.FindObject(source);

                        CVisualObject  *tcurr  =  Animation.FindObject(target);

                        if (!scurr || !tcurr)
                        {
                              AfxMessageBox("\
An object reference was found refering to an object not yet\n\
added to the Animation database.");
                              bAbortAnimation = TRUE;
                        }
                        else
                        {
                              scurr->Target = tcurr;
                              char buffer[200];
                              sprintf(buffer,
                                    "Object #%d has been assigned object #%d
as a target.\r\n",
                                    source,target);
                              pLog->Log(buffer);
                        }
                  } break;
            case OPCODE_ASSIGN_MAPTOOBJ:
                  {
                        if (i+4 >= AnimPos)
                        {
                              i = tempi;
                              bDone = TRUE;
                              break;
                        }
                        unsigned int whichobj = DecodeShortInt(AnimBuffer,i);
                        unsigned int whichtex = DecodeShortInt(AnimBuffer,i);
                        CTexture* ctex = Animation.FindTexture(whichtex);
                        CVisualObject* cobj = Animation.FindObject(whichobj);
                        if (!ctex)
                        {
                              AfxMessageBox("\
A texture reference was found refering to a texture not yet\n\
added to the Animation database.");
                              bAbortAnimation = TRUE;
                        }
                        else if (!cobj)
                        {
                              AfxMessageBox("\
An object reference was found refering to an object not yet\n\
added to the Animation database.");
                              bAbortAnimation = TRUE;
                        }
                        else
                        {
                              cobj->TextureMap = ctex;
```

```
                        char buffer[200];
                        sprintf(buffer,
                              "Texture #%d has been assigned object
#%d.\r\n",
                              whichtex,whichobj);
                        pLog->Log(buffer);
                  }
            } break;
      case OPCODE_ADDTEXMAPTYPE_HAAR:
            {
                  if (i+11 >= AnimPos)
                  {
                        i = tempi;
                        bDone = TRUE;
                        break;
                  }
                  CTexture* curr = Animation.AddTexture();
                  float min = DecodeFloat(AnimBuffer,i);
                  float max = DecodeFloat(AnimBuffer,i);
                  curr->fShininess = ScaleUp(
                        DecodeByte(AnimBuffer,i),0.0f,1.0f,BPC_8)      *
128.0f;

                  unsigned int sourcecolor = DecodeWord(AnimBuffer,i);
                  unsigned char destcolor[3];
                  ConvertColor16To24(sourcecolor,destcolor[0],
                        destcolor[1],destcolor[2]);
                  const float recip = 1.0f / 255.0f;
                  curr->SpecularColor[0]=recip*(float)destcolor[0];
                  curr->SpecularColor[1]=recip*(float)destcolor[1];
                  curr->SpecularColor[2]=recip*(float)destcolor[2];

                  curr->SetTextureType(TEXTYPE_HAAR,min,max);
                  char buffer[200];
                  sprintf(buffer,
                        "New texture add of type Haar, with a
quantization\r\n   range of (%f,%f)\r\n",
                        min,max);
                  pLog->Log(buffer);
                  sprintf(buffer,
                        "Texture has shininess value of %f with a
specular color of (%f,%f,%f)\r\n",
                        curr->fShininess,curr->SpecularColor[0],curr->Sp
ecularColor[1],
                        curr->SpecularColor[2]);
                  pLog->Log(buffer);
            } break;
      case OPCODE_ADDMIPMAP:
            {
                  if (i+2 >= AnimPos)
                  {
                        i = tempi;
                        bDone = TRUE;
                        break;
                  }
                  unsigned int which = DecodeShortInt(AnimBuffer,i);
                  CTexture *curr = Animation.FindTexture(which);
                  if (!curr)
```

```
                        {
                              AfxMessageBox("\
A texture reference was found refering to a texture not yet\n\
added to the Animation database.");
                              bAbortAnimation = TRUE;
                        }
                        else
                        {
                              // prepare the Object for addition of a texture
mipmap.
                              bDecodingTextures = TRUE;
                              DecodingTex = curr;
                              if (curr->iCurrentMipLevel == MINTEXTUREPOWER)
                              {
                                    CoefficientInc                              =
MINTEXTURERES*MINTEXTURERES;
                                    pPeano                  =                   new
CPixelPeano(MINTEXTUREPOWER,FALSE);
                              }
                              else
                              {
                                    unsigned int val = 1;
                                    for (int j=0;j<curr->iCurrentMipLevel;j++)
val*=2;
                                    CoefficientInc = val*val - (val*val >> 2);
                                    pPeano                  =                   new
CPixelPeano(curr->iCurrentMipLevel,TRUE);
                              }
                              CoefficientIndex = 0;
                              PreviousCoeff = 0.0f;
                              CoefficientBound = CoefficientInc;
                              CoefficientColor = 0;
                              TextureDelta      =      (curr->fTextureMax      -
curr->fTextureMin) / 254.0f;
                              char buffer[200];
                              sprintf(buffer,
                                    "Decoding a texture Mipmap of power %d for
texture %d\r\n",
                                    curr->iCurrentMipLevel,which);
                              pLog->Log(buffer);
                        }
                  } break;
            case OPCODE_SETBACKGROUNDCOLOR:
                  {
                        if (i+2 > AnimPos)
                        {
                              i = tempi;
                              bDone = TRUE;
                              break;
                        }
                        unsigned int sourcecolor = DecodeWord(AnimBuffer,i);
                        unsigned char destcolor[3];
                        ConvertColor16To24(sourcecolor,destcolor[0],destcolor[
1],destcolor[2]);
                        const float recip = 1.0f/255.0f;
                        Animation.BackgroundColor[0] = destcolor[0] * recip;
                        Animation.BackgroundColor[1] = destcolor[1] * recip;
                        Animation.BackgroundColor[2] = destcolor[2] * recip;
```

```
                } break;
            case OPCODE_COMPLETETOFRAME:
                        // indicates that all data is valid until the
specified
                        // keyframe.  Allows checking of animation.
                {
                        // we setup a simple debug operation.    If the
ReadyTillFrame == 0,
                        // since that doesn't technically make any sense
(since we BEGIN at frame 0)
                        if (i+3 > AnimPos)
                        {
                              i = tempi;
                              bDone = TRUE;
                              break;
                        }
                        unsigned int frame = 0;
                        unsigned char* ucTime = (unsigned char*) &frame;
                        ucTime[0] = DecodeByte(AnimBuffer,i);
                        ucTime[1] = DecodeByte(AnimBuffer,i);
                        ucTime[2] = DecodeByte(AnimBuffer,i);
                        if (frame != 0)
                              Animation.ReadyTillFrame = frame;
                        else Animation.ReadyTillFrame = 16777216;    // since
this is beyond our max frame #,

            // everything will now appear as it's ready
                        bReadyToDisplay = TRUE;

//                      Animation.FramesSoFar = 0.0f;


                        // play with time values here?
                        if (Animation.bTimeNotSet)
                        {
                              Animation.OldTime = timeGetTime();
                              Animation.StartTime = Animation.OldTime;
                              Animation.bTimeNotSet = FALSE;
                        }

                        pWin->RedrawWindow();

                        char buffer[200];
                        sprintf(buffer, "***Complete to frame marker placed at
");
                        pLog->Log(buffer);
                        sprintf(buffer,     "%d     and     now     ready     to
display***\r\n",Animation.ReadyTillFrame);
                        pLog->Log(buffer);
                  } break;
             case OPCODE_ENDOFFILEMARKER:
                  {
                        // END of the data file has been reached
                        pLog->Log("END OF FILE marker reached.\r\n");
                        pLog->Log("--------------------------------\r\n");
                        bDone = TRUE;
                        bDecodeComplete = TRUE;
                        Beep(1000,15);
```

```
                } break;
            default:
                char buffer[200];
                sprintf(buffer,
                    "Invalid opcode %d found in source file's decoded
stream.\r\n",
                    DecodeShortInt(AnimBuffer,tempi));
                pLog->Log(buffer);
                OutputDebugString(buffer);
                AfxMessageBox("Invalid opcode discovered.");
                bAbortAnimation = TRUE;
            }
        }

//      AnimPos+=2;
        for (int j=i;j<AnimPos;j++)
            AnimBuffer[j-i] = AnimBuffer[j];
        AnimPos-=i;

        // if aborted, stop downloading!
        if (bAbortAnimation)
        {
            IBinding* pIBind = GetBinding();
            pIBind->Abort();
        }
}

void    CAnimationFile::DecodeAnimation(int    HeaderLength,BOOL    &Reset,int
HoldBack)
{
        int curPointer = HeaderLength;
        static int curMode = 0;
        BOOL bDone = FALSE;
        for (int j=0;j<HoldBack;j++)
            Buffer[BufferPos+j] = 0;

        while (curPointer < (BufferPos-HoldBack) && !bDone)
            // translate whatever we have in the buffer,
            // and dump it into the AnimBuffer
            bDone   =   DecodeCode(curPointer,Buffer,AnimPos,AnimBuffer,Reset);

        for (int i=0;curPointer<BufferPos;i++,curPointer++)
            Buffer[i]=Buffer[curPointer];
        BufferPos = i;
        if (bDone)
            bDecodeComplete = TRUE;
}

BOOL CAnimationFile::DecodeCode(int &codepointer, BYTE * CodeBuffer,
                                                int    &respointer,    BYTE    *
ResBuffer,
                                                BOOL &Reset)
{
        // decode the current code listed at the codepointer location, and
        // place it's true value into ResBuffer, adjusting codepointer and
        // respointer as we go.

        // if adding the current code's result to the result buffer will
```

```
	// overflow the buffer, return false, and adjust back the two pointers.

	// Decompress a single LZSS codeword from the codebuffer, and place
	// the result in the resbuffer.

	static int input_bit_count;
	static unsigned int input_bit_buffer;
	static unsigned int pDict;

	if (Reset)
	{
		input_bit_count  = 0;
		input_bit_buffer = 0;
		pDict = 0;
		for (int i=0;i<LZSSWINDOW_LENGTH;Dictionary[i++] = 0);

		iCompressedCount = 0;
		iUnCompressedCount = 1;
		Reset = FALSE;
	}

	unsigned char HeaderBit = GetCode(codepointer,CodeBuffer,1,
		input_bit_count,input_bit_buffer);
	if (HeaderBit == 1)
	{
		unsigned char newchar = GetCode(codepointer,CodeBuffer,8,
			input_bit_count,input_bit_buffer);
		Dictionary[pDict] = newchar;
		pDict = (pDict+1) % LZSSWINDOW_LENGTH;
		ResBuffer[respointer++]=newchar;
	}
	else
	{
		// get the long code
		unsigned int word =
GetCode(codepointer,CodeBuffer,LZSSBITLEN+LZSSBITPOS,
			input_bit_count,input_bit_buffer);
		// if the code is 2^21-1 we've reached our stop code, so stop.
		unsigned int test = ((1 << (LZSSBITLEN+LZSSBITPOS))-1);
		if (word == test)
			return TRUE;
		unsigned int pos = word & ((1 << LZSSBITPOS)-1);
		unsigned int len = word  >> LZSSBITPOS;
		for (unsigned int i=0;i<len+3;i++)
			ResBuffer[respointer++] = Dictionary[(pos + i) %
LZSSWINDOW_LENGTH];
		for (i=0;i<len+3;i++)
			Dictionary[(pDict + i) % LZSSWINDOW_LENGTH] =
ResBuffer[respointer-len-3+i];
		pDict = (pDict + len+3) % LZSSWINDOW_LENGTH;
	}
	return FALSE;
}

float CAnimationFile::ScaleUp(unsigned int val, float min, float max, int
bitdepth)
{
```

```
        float scale;
        float fval = (float) val;
        switch (bitdepth)
        {
                case BPC_16: scale = 65535.0f; break;
                case BPC_12: scale = 4095.0f; break;
                case BPC_10: scale = 1023.0f; break;
                case BPC_8:  scale = 255.0f; break;
        }
        float perc = fval / scale;
        return ( min + perc * (max - min)  );
}

unsigned int CAnimationFile::DecodeShortInt(unsigned char * Buffer, int &
pointer)
// extracts a dynamic "short" integer value from the buffer, which can either
be
// 8 or 16 bits long, depending on what the value is
{
        unsigned char val1 = DecodeByte(Buffer,pointer);
        if (val1 > 199)
                return (DecodeByte(Buffer,pointer) + (val1-199)*200);
        else return val1;
}

float CAnimationFile::DecodeFloat(unsigned char *Buffer, int &pointer)
// extracts a 32 bit float value from the buffer
{
        float val;
        unsigned char * fakearr = (unsigned char *) &val;
        memcpy(&val,Buffer+pointer,4);
        pointer+=4;
        return val;
}

unsigned short CAnimationFile::DecodeWord(unsigned char *Buffer, int &pointer)
// extracts a 16 bit word value from the buffer
{
        unsigned short val;
        memcpy(&val,Buffer+pointer,2);
        pointer+=2;
        return val;
}

unsigned char CAnimationFile::DecodeByte(unsigned char *Buffer, int &pointer)
// extracts an 8 bit byte value from the buffer
{
        unsigned char val;
        memcpy(&val,Buffer+pointer,1);
        pointer++;
        return val;
}

unsigned int CAnimationFile::DecodePartialWord(unsigned char *Buffer, int
&pointer,
                                                                unsigned
int bitlength,unsigned int &input_bit_buffer,
```

```
                                                                  unsigned
int &input_bit_count, int &total)
{
      unsigned int return_value;

      while (input_bit_count < bitlength)
      {
            input_bit_buffer |=
                  (unsigned    int)    DecodeByte(Buffer,pointer)    <<    (24-
input_bit_count);
            input_bit_count += 8;
      }
      return_value = input_bit_buffer >> (32-bitlength);
      input_bit_buffer <<= bitlength;
      input_bit_count -= bitlength;
      total--;
      return(return_value);
}

BOOL CAnimationFile::Decode3FloatList(int count, unsigned char * Buffer, float
* floatlist, int & pointer)
{
      int loop;
      unsigned char chararr[256*3];

      for(loop=0;loop<12;loop++)
            for (int loop2=0;loop2<count;loop2++)
//                memcpy(&chararr[loop*count+loop2],
Buffer+pointer+loop+count*loop2,1);
                  chararr[loop2*12+loop] = Buffer[pointer++];
//    pointer+=count * 3 * 4;
      unsigned int temp;
      for (loop=0;loop<count;loop++)
      {
            memcpy(&temp,&chararr[loop*3*4],4);
            temp = _rotl(temp,1);
            memcpy(&floatlist[loop*3],&temp,4);

            memcpy(&temp,&chararr[loop*3*4+4],4);
            temp = _rotl(temp,1);
            memcpy(&floatlist[loop*3+1],&temp,4);

            memcpy(&temp,&chararr[loop*3*4+8],4);
            temp = _rotl(temp,1);
            memcpy(&floatlist[loop*3+2],&temp,4);
      }
      return TRUE;
}

CAnimationFile::ProcessVersionNumber()
{
      if (Animation.MajorVersion != MAJORVERSION || Animation.MinorVersion !=
MINORVERSION)
      {
            AfxMessageBox("\
Error! The version of the animation file is incompatible with\n\
this version of the decompression engine."
                  );
```

```
            bAbortAnimation = TRUE;
            bPreBuffering = FALSE;
      }

}

CAnimationFile::ConvertColor16To24(unsigned int source, unsigned char &red,
                                                    unsigned char &green,
unsigned char &blue)
{
      unsigned int temp;
      temp = (source & 0x0000F800) >> 11; red = (unsigned char) temp << 3;
      temp = (source & 0x000007E0) >> 5;  green = (unsigned char) temp << 2;
      temp = (source & 0x0000001F);       blue = (unsigned char) temp << 3;
}

CAnimationFile::HaarInv(float a[], float source[], unsigned int size)
{
      const unsigned int halfsize = size >> 1;

      for (unsigned int i=0,j=0;i<halfsize;i++)
      {
            a[j++] = source[i] + source[i+halfsize]*0.5f;
            a[j++] = source[i] - source[i+halfsize]*0.5f;
      }
}


unsigned int CAnimationFile::GetCode(int &codepointer, BYTE *CodeBuffer, int
bitlength,
                                                                    int
&input_bit_count,unsigned int &input_bit_buffer)
{
      unsigned int return_value;

      while (input_bit_count < bitlength)
      {
            input_bit_buffer |=
                  (unsigned    int)    CodeBuffer[codepointer++]    <<    (24-
input_bit_count);
            input_bit_count += 8;
            iCompressedCount++;
      }
      return_value = input_bit_buffer >> (32-bitlength);
      input_bit_buffer <<= bitlength;
      input_bit_count -= bitlength;
      return(return_value);
}

CAnimationFile::Reset()
{
      BufferPos = 0;
      AnimPos = 0;
      bDownloadComplete = FALSE;
      bAbortAnimation = FALSE;
      bAnimationInProgress = FALSE;
      bReadyToDisplay = FALSE;
      bDecodeComplete = FALSE;
```

```
        bPreBuffering = FALSE;
        Animation.Reset();
}
```

D.2.20 MainFrm.h

```
// MainFrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_MAINFRM_H__A4F36375_C3EB_11D1_87F8_00AA00B9C442__INCLUDED_)
#define AFX_MAINFRM_H__A4F36375_C3EB_11D1_87F8_00AA00B9C442__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CStyleBar : public CToolBar
{
public:
      CSliderCtrl m_Slider;
};

class CMainFrame : public CFrameWnd
{
protected: // create from serialization only
      CMainFrame();
      DECLARE_DYNCREATE(CMainFrame)

// Attributes
public:

// Operations
public:

// Overrides
      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CMainFrame)
      virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
      //}}AFX_VIRTUAL

// Implementation
public:
      virtual ~CMainFrame();
#ifdef _DEBUG
      virtual void AssertValid() const;
      virtual void Dump(CDumpContext& dc) const;
#endif

public:  // control bar embedded members
      int width;
      BOOL CreateStyleBar();
      CStatusBar  m_wndStatusBar;
      CStyleBar   m_wndPlayBar;

// Generated message map functions
protected:
      //{{AFX_MSG(CMainFrame)
      afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
            // NOTE - the ClassWizard will add and remove member functions
```

```
here.
            //    DO NOT EDIT what you see in these blocks of generated code!
      //}}AFX_MSG
      DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_MAINFRM_H__A4F36375_C3EB_11D1_87F8_00AA00B9C442__INCLUDED_)
```

D.2.21 MainFrm.cpp

```
// MainFrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "Decompressor.h"

#include "MainFrm.h"
#include "resource.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

CStatusBar *m_pwndStatusBar;
CSliderCtrl *pSlider;
CStyleBar *pBar;
BOOL bSliderCreated = FALSE;
CWnd *pWin = NULL;

/////////////////////////////////////////////////////////////////////////////
// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
      //{{AFX_MSG_MAP(CMainFrame)
            // NOTE - the ClassWizard will add and remove mapping macros
here.
            //    DO NOT EDIT what you see in these blocks of generated code
!
      ON_WM_CREATE()
      //}}AFX_MSG_MAP
END_MESSAGE_MAP()

static UINT indicators[] =
{
      ID_SEPARATOR,           // status line indicator
      ID_INDICATOR_CAPS,
      ID_INDICATOR_NUM,
      ID_INDICATOR_SCRL,
};

static UINT BASED_CODE ButtonStyles[] =
{
      // same order as in the bitmap 'styles.bmp'
      ID_ANIMATION_PLAY,
      ID_ANIMATION_PAUSE,
      ID_ANIMATION_STOP,
      ID_SEPARATOR,
      ID_SEPARATOR,           // for pullbar (placeholder)
};

/////////////////////////////////////////////////////////////////////////////
```

```
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
      AfxInitRichEdit();
      pWin = this;
}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
      if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
            return -1;

      if (!m_wndStatusBar.Create(this) ||
            !m_wndStatusBar.SetIndicators(indicators,
              sizeof(indicators)/sizeof(UINT)))
      {
            TRACE0("Failed to create status bar\n");
            return -1;      // fail to create
      }

      m_pwndStatusBar = &m_wndStatusBar;

      CreateStyleBar();

      return 0;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
      // this style prevents the window from being resized
    cs.style = WS_OVERLAPPED | WS_SYSMENU | WS_BORDER |
                        WS_MINIMIZEBOX;
      cs.cx = 500+12;
      cs.cy = 375+98;
      width = cs.cx;
    cs.y = (::GetSystemMetrics(SM_CYSCREEN) >> 1) - 270;
    cs.x = (::GetSystemMetrics(SM_CXSCREEN) >> 1) - 320;

      return CFrameWnd::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
      CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
      CFrameWnd::Dump(dc);
```

```
}

#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CMainFrame message handlers

BOOL CMainFrame::CreateStyleBar()
{
      const int nDropHeight = 100;

      if (!m_wndPlayBar.Create(this, WS_CHILD|WS_VISIBLE|CBRS_BOTTOM|
                  CBRS_TOOLTIPS|CBRS_FLYBY, IDW_STYLES) ||
            !m_wndPlayBar.LoadBitmap(IDB_PLAYBUTTONS) ||
            !m_wndPlayBar.SetButtons(ButtonStyles, sizeof(ButtonStyles)/sizeof
(UINT)))
      {
            TRACE0("Failed to create Play Bar\n");
            return FALSE;       // fail to create
      }

      // Create the combo box
      m_wndPlayBar.SetButtonInfo(4,  IDW_PLAYBUTTONS,  TBBS_SEPARATOR,  width -
90);
      CRect rect;
      m_wndPlayBar.GetItemRect(4, &rect);
      rect.top = 0;
      rect.bottom = rect.top + 20;
      if (!m_wndPlayBar.m_Slider.Create(
                  WS_VISIBLE|WS_TABSTOP,
                  rect, &m_wndPlayBar, IDW_PLAYBUTTONS))
      {
            TRACE0("Failed to create Slider\n");
            return FALSE;
      }
      pSlider = &(m_wndPlayBar.m_Slider);
      pBar = &(m_wndPlayBar);
      bSliderCreated = TRUE;
      return TRUE;
}
```

D.2.22 Decompressor.h

```
// Decompressor.h : main header file for the DECOMPRESSOR application
//

#if
!defined(AFX_DECOMPRESSOR_H__A4F36371_C3EB_11D1_87F8_00AA00B9C442__INCLUDED_)
#define AFX_DECOMPRESSOR_H__A4F36371_C3EB_11D1_87F8_00AA00B9C442__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#ifndef __AFXWIN_H__
      #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"       // main symbols

/////////////////////////////////////////////////////////////////////////////
// CDecompressorApp:
// See Decompressor.cpp for the implementation of this class
//

class CDecompressorApp : public CWinApp
{
public:
      CDecompressorApp();

// Overrides
      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CDecompressorApp)
      public:
      virtual BOOL InitInstance();
      virtual int Run();
      //}}AFX_VIRTUAL

// Implementation

      //{{AFX_MSG(CDecompressorApp)
      afx_msg void OnAppAbout();
            // NOTE - the ClassWizard will add and remove member functions
here.
            //    DO NOT EDIT what you see in these blocks of generated code
!
      //}}AFX_MSG
      DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////
//{{AFX_INSERT_LOCATION}}

                                                                          //
#endif
!defined(AFX_DECOMPRESSOR_H__A4F36371_C3EB_11D1_87F8_00AA00B9C442__INCLUDED_)
```

D.2.23 Decompressor.cpp

```
// Decompressor.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "Decompressor.h"

#include "MainFrm.h"
#include "DecompressorDoc.h"
#include "DecompressorView.h"

#include "gl\gl.h"
#include "gl\glu.h"
#include "gl\glaux.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

extern unsigned int TitleScreenTexture;
extern unsigned char * TitleGFXarray;
extern unsigned int BufferingTexture;
extern unsigned char * Bufferarray;

/////////////////////////////////////////////////////////////////////////////
// CDecompressorApp

BEGIN_MESSAGE_MAP(CDecompressorApp, CWinApp)
      //{{AFX_MSG_MAP(CDecompressorApp)
      ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
            // NOTE - the ClassWizard will add and remove mapping macros
here.
            //    DO NOT EDIT what you see in these blocks of generated code!
      //}}AFX_MSG_MAP
      // Standard file based document commands
      ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
      ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CDecompressorApp construction

CDecompressorApp::CDecompressorApp()
{
      // TODO: add construction code here,
      // Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CDecompressorApp object

CDecompressorApp theApp;
```

```
/////////////////////////////////////////////////////////////////////////////
// CDecompressorApp initialization

BOOL CDecompressorApp::InitInstance()
{
	// Standard initialization
	// If you are not using these features and wish to reduce the size
	//  of your final executable, you should remove from the following
	//  the specific initialization routines you do not need.

#ifdef _AFXDLL
	Enable3dControls();			// Call this when using MFC in a shared DLL
#else
	Enable3dControlsStatic();	// Call this when linking to MFC statically
#endif

	// Change the registry key under which our settings are stored.
	// You should modify this string to be something appropriate
	// such as the name of your company or organization.
	SetRegistryKey(_T("Local AppWizard-Generated Applications"));

	LoadStdProfileSettings();  // Load standard INI file options (including MRU)

	// Register the application's document templates.  Document templates
	//  serve as the connection between documents, frame windows and views.

	CSingleDocTemplate* pDocTemplate;
	pDocTemplate = new CSingleDocTemplate(
		IDR_MAINFRAME,
		RUNTIME_CLASS(CDecompressorDoc),
		RUNTIME_CLASS(CMainFrame),       // main SDI frame window
		RUNTIME_CLASS(CDecompressorView));
	AddDocTemplate(pDocTemplate);

	// Parse command line for standard shell commands, DDE, file open
	CCommandLineInfo cmdInfo;
	ParseCommandLine(cmdInfo);

	// Dispatch commands specified on the command line
	if (!ProcessShellCommand(cmdInfo))
		return FALSE;

	HGLOBAL TitleGFXHandle, BufferHandle;
	::glEnable(GL_TEXTURE_2D);
	HRSRC hrc = FindResource(NULL,MAKEINTRESOURCE(IDR_RAWTITLE),"RAWMAP");
	HRSRC hrc2= FindResource(NULL,MAKEINTRESOURCE(IDR_RAWBUFFER),"RAWMAP");

	TitleGFXHandle = LoadResource(AfxGetInstanceHandle(),hrc);
	BufferHandle   = LoadResource(AfxGetInstanceHandle(),hrc2);

	TitleGFXarray = (unsigned char*) LockResource(TitleGFXHandle);
	Bufferarray = (unsigned char*) LockResource(BufferHandle);

	::glPixelStorei(GL_UNPACK_ALIGNMENT,1);
```

```
	::glGenTextures(1, &TitleScreenTexture);
	::glBindTexture(GL_TEXTURE_2D,TitleScreenTexture);
	::glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S, GL_REPEAT);
	::glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T, GL_REPEAT);
	::glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER, GL_LINEAR);
	::glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER, GL_LINEAR);

	::glTexImage2D( GL_TEXTURE_2D, 0, GL_RGB, 256, 256, 0, GL_RGB,
				GL_UNSIGNED_BYTE, TitleGFXarray );

	::glPixelStorei(GL_UNPACK_ALIGNMENT,1);
	::glGenTextures(1, &BufferingTexture);
	::glBindTexture(GL_TEXTURE_2D,BufferingTexture);
	::glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S, GL_REPEAT);
	::glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T, GL_REPEAT);
	::glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER, GL_LINEAR);
	::glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER, GL_LINEAR);

	::glTexImage2D( GL_TEXTURE_2D, 0, GL_RGB, 256, 256, 0, GL_RGB,
				GL_UNSIGNED_BYTE, Bufferarray );

	// The one and only window has been initialized, so show and update it.
	m_pMainWnd->ShowWindow(SW_SHOW);
	m_pMainWnd->UpdateWindow();

	return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
	CAboutDlg();

// Dialog Data
	//{{AFX_DATA(CAboutDlg)
	enum { IDD = IDD_ABOUTBOX };
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAboutDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	//{{AFX_MSG(CAboutDlg)
		// No message handlers
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
```

```
      //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
      CDialog::DoDataExchange(pDX);
      //{{AFX_DATA_MAP(CAboutDlg)
      //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
      //{{AFX_MSG_MAP(CAboutDlg)
            // No message handlers
      //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CDecompressorApp::OnAppAbout()
{
      CAboutDlg aboutDlg;
      aboutDlg.DoModal();
}

/////////////////////////////////////////////////////////////////////////////
// CDecompressorApp commands

int CDecompressorApp::Run()
{
      BOOL bDoingBackgroundProcessing = TRUE;
      while ( bDoingBackgroundProcessing )
      {
            MSG msg;
            while ( ::PeekMessage( &msg, NULL, 0, 0, PM_NOREMOVE ) )
            {
                  if ( !PumpMessage( ) )
                  {
                        bDoingBackgroundProcessing = FALSE;
                          ::PostQuitMessage(0 );
                            break;
                  }
            }
            // let MFC do its idle processing
          LONG lIdle = 0;
            while ( AfxGetApp()->OnIdle(lIdle++ ) )
                  ;
          // Perform some background processing here
            // using another call to OnIdle
            CView * pActiveView = NULL;
            CFrameWnd * pFrm = (CFrameWnd*) AfxGetMainWnd( );
            if (pFrm) pActiveView = pFrm->GetActiveView();
            CDecompressorView* pClone = (CDecompressorView *) pActiveView;
//          if (pActiveView && pClone && !pClone->bStartingUpLevel1)
            if (pActiveView && pClone)
                  pClone->Invalidate(FALSE);
      }
      return CWinApp::Run();
}
```

D.2.24 DecompressorDoc.h

```
// DecompressorDoc.h : interface of the CDecompressorDoc class
//
/////////////////////////////////////////////////////////////////////////////

#if
!defined(AFX_DECOMPRESSORDOC_H__A4F36377_C3EB_11D1_87F8_00AA00B9C442__INCLUDED
_)
#define AFX_DECOMPRESSORDOC_H__A4F36377_C3EB_11D1_87F8_00AA00B9C442__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#include "AnimationFile.h"

class CDecompressorDoc : public CDocument
{
protected: // create from serialization only
      CDecompressorDoc();
      DECLARE_DYNCREATE(CDecompressorDoc)

// Attributes
public:

// Operations
public:

// Overrides
      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CDecompressorDoc)
      public:
      virtual BOOL OnNewDocument();
      virtual void Serialize(CArchive& ar);
      //}}AFX_VIRTUAL

// Implementation
public:
      virtual ~CDecompressorDoc();
#ifdef _DEBUG
      virtual void AssertValid() const;
      virtual void Dump(CDumpContext& dc) const;
#endif

public:
      CAnimationFile* AnimF;

protected:
// Generated message map functions
protected:
      //{{AFX_MSG(CDecompressorDoc)
            // NOTE - the ClassWizard will add and remove member functions
here.
            //    DO NOT EDIT what you see in these blocks of generated code
!
```

```
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

                                                                          //
#endif
!defined(AFX_DECOMPRESSORDOC_H__A4F36377_C3EB_11D1_87F8_00AA00B9C442__INCLUDED
_)
```

D.2.25 DecompressorDoc.cpp

```
// DecompressorDoc.cpp : implementation of the CDecompressorDoc class
// Just a placeholder for the animation database

#include "stdafx.h"
#include "Decompressor.h"

#include "DecompressorDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif


#ifdef _DEBUG
#pragma warning(disable : 4074)
      #pragma init_seg(compiler)

//forward fn. decl.
BOOL AFXAPI AllocHook(size_t nSize, BOOL bObject, LONG lRequestNumber);

//set various sys hooks
static AFX_ALLOC_HOOK PrevAllocHook = AfxSetAllocHook(AllocHook);

LONG g_lFirstReqCaught = 0;

BOOL AFXAPI AllocHook(size_t nSize, BOOL bObject, LONG lRequestNumber)
{
      if(!g_lFirstReqCaught)
      {
            g_lFirstReqCaught = lRequestNumber;
            TRACE("L_Afx:  Caught  first  alloc  request  number  {%d}\n",
g_lFirstReqCaught);

            //turn off memory alloc tracking
            PrevAllocHook = AfxSetAllocHook(PrevAllocHook);

            //set memory allocation breakpoints here
      }
      return TRUE;
}
#endif


////////////////////////////////////////////////////////////////////////////
// CDecompressorDoc

IMPLEMENT_DYNCREATE(CDecompressorDoc, CDocument)

BEGIN_MESSAGE_MAP(CDecompressorDoc, CDocument)
      //{{AFX_MSG_MAP(CDecompressorDoc)
            // NOTE - the ClassWizard will add and remove mapping macros
here.
            //    DO NOT EDIT what you see in these blocks of generated code!
```

```
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CDecompressorDoc construction/destruction

CDecompressorDoc::CDecompressorDoc()
{
	AnimF = new CAnimationFile;
}

CDecompressorDoc::~CDecompressorDoc()
{
	delete AnimF;
}

BOOL CDecompressorDoc::OnNewDocument()
{
	if (!CDocument::OnNewDocument())
		return FALSE;
	return TRUE;
}


/////////////////////////////////////////////////////////////////////////////
// CDecompressorDoc serialization

void CDecompressorDoc::Serialize(CArchive& ar)
{
	if (ar.IsStoring())
	{}
	else
	{}
}

/////////////////////////////////////////////////////////////////////////////
// CDecompressorDoc diagnostics

#ifdef _DEBUG
void CDecompressorDoc::AssertValid() const
{
	CDocument::AssertValid();
}

void CDecompressorDoc::Dump(CDumpContext& dc) const
{
	CDocument::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CDecompressorDoc commands
```

D.2.26 DecompressorView.h

```
// DecompressorView.h : interface of the CDecompressorView class
// Displays the animations - all OpenGL specific rendering is
// here (with the exception of some allocation in CDecompressor
//////////////////////////////////////////////////////////////////////////

#if
!defined(AFX_DECOMPRESSORVIEW_H__A4F36379_C3EB_11D1_87F8_00AA00B9C442__INCLUDE
D_)
#define
AFX_DECOMPRESSORVIEW_H__A4F36379_C3EB_11D1_87F8_00AA00B9C442__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#include "gl\gl.h"
#include "gl\glu.h"
#include "gl\glaux.h"
#include <afxadv.h>
#include "DialogLog.h"  // Added by ClassView

class CDecompressorView : public CView
{
protected: // create from serialization only
      CDecompressorView();
      DECLARE_DYNCREATE(CDecompressorView)

// Attributes
public:
      CDecompressorDoc* GetDocument();
      float aspect_ratio;
      float aspect_ratio_inv;
      int m_cx;
      int m_cy;
      CDC* m_pDC;
private:
      HGLRC m_hRC;

// Operations
public:
      BOOL InitializeOpenGL();
      void SetupFlags();
      BOOL RenderScene();
      BOOL SetupPixelFormat();

// Overrides
      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CDecompressorView)
      public:
      virtual void OnDraw(CDC* pDC);  // overridden to draw this view
      virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
      protected:
      //}}AFX_VIRTUAL
```

```
// Implementation
public:
	PlayAnimation();
	BOOL bCurrentlyDrawing;
	ReBindTexture(CTexture *tex);
	BOOL bLogActive;
	CDialogLog LogWindow;
	BOOL bEndedNaturally;
	DWORD dwFrames;
	DWORD dwTimePerDivision;
	float fNextBlock;
	BOOL bIsPlayAble;
	BOOL bStopped;
	BOOL bStartingRendering;
	BOOL bStartingUpLevel1;
	BOOL bStartingUpLevel2;
	BOOL bStartingUpLevel3;
	BOOL bStartingUpLevel4;
	void SetupCamera();
	void FileProcess(int number);
	void  RenderPatch(CVisualObject  *  Obj,  int  *  PatchVertices,  float
TesRate);
	void RenderObjects();
	void SetupObjects();
	void SetupLighting();
	float fTime;
	float fTotalTime;
	virtual ~CDecompressorView();
#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
	//{{AFX_MSG(CDecompressorView)
	afx_msg void OnFileOpenlocation();
	afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
	afx_msg void OnDestroy();
	afx_msg void OnSize(UINT nType, int cx, int cy);
	afx_msg BOOL OnEraseBkgnd(CDC* pDC);
	afx_msg void OnFileMruFile1();
	afx_msg void OnFileMruFile2();
	afx_msg void OnFileMruFile3();
	afx_msg void OnFileMruFile4();
	afx_msg void OnAnimationPlay();
	afx_msg void OnUpdateAnimationPlay(CCmdUI* pCmdUI);
	afx_msg void OnAnimationPause();
	afx_msg void OnUpdateAnimationPause(CCmdUI* pCmdUI);
	afx_msg void OnAnimationStop();
	afx_msg void OnUpdateAnimationStop(CCmdUI* pCmdUI);
	afx_msg void OnViewDiagnostics();
	afx_msg void OnAnimationDownloadspeed();
	//}}AFX_MSG
```

```
    DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG  // debug version in DecompressorView.cpp
inline CDecompressorDoc* CDecompressorView::GetDocument()
   { return (CDecompressorDoc*)m_pDocument; }
#endif

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

                                                                          //
#endif
!defined(AFX_DECOMPRESSORVIEW_H__A4F36379_C3EB_11D1_87F8_00AA00B9C442__INCLUDE
D_)
```

D.2.27 DecompressorView.cpp

```
// DecompressorView.cpp : implementation of the CDecompressorView class
//

#include "stdafx.h"
#include <mmsystem.h>
#include <math.h>
#include "MainFrm.h"
#include "Decompressor.h"

#include "DecompressorDoc.h"
#include "DecompressorView.h"

#include "GetUrl.h"
#include "AnimationFile.h"

#include "Quaternion.h"
#include "../Compressor/Opcodes.h"

#include "DialogLog.h"

#include "GetDownloadSpeed.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

unsigned int TitleScreenTexture;
unsigned char * TitleGFXarray;
unsigned int BufferingTexture;
unsigned char * Bufferarray;

extern CSliderCtrl* pSlider;
extern CStyleBar *pBar;
extern BOOL bSliderCreated;
extern CWnd *pWin;

CDialogLog *pLog;
/////////////////////////////////////////////////////////////////////////////
// CDecompressorView

IMPLEMENT_DYNCREATE(CDecompressorView, CView)

BEGIN_MESSAGE_MAP(CDecompressorView, CView)
      //{{AFX_MSG_MAP(CDecompressorView)
      ON_COMMAND(ID_FILE_OPENLOCATION, OnFileOpenlocation)
      ON_WM_CREATE()
      ON_WM_DESTROY()
      ON_WM_SIZE()
      ON_WM_ERASEBKGND()
      ON_COMMAND(ID_FILE_MRU_FILE1, OnFileMruFile1)
      ON_COMMAND(ID_FILE_MRU_FILE2, OnFileMruFile2)
      ON_COMMAND(ID_FILE_MRU_FILE3, OnFileMruFile3)
```

```
	ON_COMMAND(ID_FILE_MRU_FILE4, OnFileMruFile4)
	ON_COMMAND(ID_ANIMATION_PLAY, OnAnimationPlay)
	ON_UPDATE_COMMAND_UI(ID_ANIMATION_PLAY, OnUpdateAnimationPlay)
	ON_COMMAND(ID_ANIMATION_PAUSE, OnAnimationPause)
	ON_UPDATE_COMMAND_UI(ID_ANIMATION_PAUSE, OnUpdateAnimationPause)
	ON_COMMAND(ID_ANIMATION_STOP, OnAnimationStop)
	ON_UPDATE_COMMAND_UI(ID_ANIMATION_STOP, OnUpdateAnimationStop)
	ON_COMMAND(ID_VIEW_DIAGNOSTICS, OnViewDiagnostics)
	ON_COMMAND(ID_ANIMATION_DOWNLOADSPEED, OnAnimationDownloadspeed)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CDecompressorView construction/destruction

CDecompressorView::CDecompressorView()
{
	bStartingUpLevel1 = TRUE;
	bStartingUpLevel2 = FALSE;
	bStartingUpLevel3 = FALSE;
	bStartingUpLevel4 = FALSE;
	fTotalTime = 0.0f;
	fNextBlock = 1.0f;
	bStopped = FALSE;
	bStartingRendering = TRUE;
	bIsPlayAble = FALSE;
	dwFrames = 0;
	dwTimePerDivision = 0;
	bEndedNaturally = TRUE;
	LogWindow.Create();
	bLogActive = FALSE;
	pLog = &(LogWindow);
}

CDecompressorView::~CDecompressorView()
{
	LogWindow.DestroyWindow();
}

BOOL CDecompressorView::PreCreateWindow(CREATESTRUCT& cs)
{
	// TODO: Modify the Window class or styles here by modifying
	//  the CREATESTRUCT cs

	return CView::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CDecompressorView drawing

void CDecompressorView::OnDraw(CDC* pDC)
{
	CDecompressorDoc* pDoc = GetDocument();
	ASSERT_VALID(pDoc);

	if (bStartingUpLevel2 && !pDoc->AnimF->bPreBuffering)
	{
		bStartingUpLevel1 = FALSE;
		bStartingUpLevel2 = FALSE;
```

```
            bStartingUpLevel3 = TRUE;
      }

      if (bStartingUpLevel1)
      {
            bStartingUpLevel2 = FALSE;
            ::glMatrixMode(GL_PROJECTION);
            ::glLoadIdentity();
            ::glMatrixMode(GL_MODELVIEW);
            ::glLoadIdentity();

            CRect Region;
            GetWindowRect(Region);
            ::glViewport(0,0,Region.right-Region.left,Region.bottom-
Region.top);

            ::glClearColor(0.0f,0.0f,0.0f,1.0f);
            ::glClear(GL_DEPTH_BUFFER_BIT);
            ::glEnable(GL_DEPTH_TEST);
            // display splash background
            ::glEnable(GL_TEXTURE_2D);
            if (pDoc->AnimF->bPreBuffering)
            {
                  ::glBindTexture(GL_TEXTURE_2D,BufferingTexture);
                  bStartingUpLevel2 = TRUE;
            }
            else
                  ::glBindTexture(GL_TEXTURE_2D,TitleScreenTexture);
            ::glColor3f(1.0f,1.0f,1.0f);
            ::glBegin(GL_QUADS);
                  ::glTexCoord2f(0.0f, 0.0f);
                  ::glVertex2f(-1.0f,-1.0f);
                  ::glTexCoord2f(1.0f, 0.0f);
                  ::glVertex2f( 1.0f,-1.0f);
                  ::glTexCoord2f(1.0f, 1.0f);
                  ::glVertex2f( 1.0f, 1.0f);
                  ::glTexCoord2f(0.0f, 1.0f);
                  ::glVertex2f(-1.0f, 1.0f);
            ::glEnd();
            SwapBuffers(m_pDC->GetSafeHdc() );
      }

      ::glDisable(GL_TEXTURE_2D);
      if (!pDoc->AnimF->bReadyToDisplay || pDoc->AnimF->bPreBuffering)
            return;
      if (pDoc->AnimF->bAbortAnimation )
      {
            bStartingUpLevel1 = TRUE;
            return;
      }
      if (bStartingUpLevel1)
      {
            bStartingUpLevel1 = FALSE;
            return;
      }
      if (bStartingUpLevel3)
      {
            bStartingUpLevel3 = FALSE;
```

```
            bStartingUpLevel4 = TRUE;
            OnAnimationStop();
            return;
      }

      ::glClearColor(pDoc->AnimF->Animation.BackgroundColor[0],
            pDoc->AnimF->Animation.BackgroundColor[1],
            pDoc->AnimF->Animation.BackgroundColor[2],1.0f);
      ::glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
      ::glEnable(GL_DEPTH_TEST);

      ::glColor3f(1.0f,1.0f,1.0f);

      if (RenderScene())
      {
            ::glMatrixMode(GL_MODELVIEW);
            ::glFinish();

            if (!::SwapBuffers(m_pDC->GetSafeHdc() ) )
            {
                  TRACE("Error while double buffering\n");
            }

            if (bStartingUpLevel4)
            {
                  OnAnimationPlay();
                  bStartingUpLevel4 = FALSE;
            }
      }
}

////////////////////////////////////////////////////////////////////////////
// CDecompressorView diagnostics

#ifdef _DEBUG
void CDecompressorView::AssertValid() const
{
      CView::AssertValid();
}

void CDecompressorView::Dump(CDumpContext& dc) const
{
      CView::Dump(dc);
}

CDecompressorDoc* CDecompressorView::GetDocument() // non-debug version is
inline
{
      ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CDecompressorDoc)));
      return (CDecompressorDoc*)m_pDocument;
}
#endif //_DEBUG

////////////////////////////////////////////////////////////////////////////
// CDecompressorView message handlers

void CDecompressorView::OnFileOpenlocation()
{
```

```
        CDecompressorDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);

        CGetUrl dlg;
        if (dlg.DoModal())
        {
                if (pDoc->AnimF)
                {
                        delete pDoc->AnimF;
                        pDoc->AnimF = new CAnimationFile;
                }
                // this doesn't actually play the animation, it just
                // sets all the internal variables that allow the
                // engine to playback an animation.  That's why
                // its set before the Open() command, as the file
                // opening must be completed before the function
                // returns.
                bStartingUpLevel1 = TRUE;
                bIsPlayAble = TRUE;
                bEndedNaturally = TRUE;
                PlayAnimation();
                if (!pDoc->AnimF->Open(dlg.m_UrlName))
                {
                        AfxMessageBox("Unable to Open specified URL");
                        bIsPlayAble = FALSE;
                        pDoc->AnimF->bAnimationInProgress = FALSE;
                        return;
                }
                AfxGetApp()->AddToRecentFileList(dlg.m_UrlName);
        }
}

int CDecompressorView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
        if (CView::OnCreate(lpCreateStruct) == -1)
                return -1;

        if (!InitializeOpenGL())
        {
                TRACE("OpenGL failed to init.\n");
                AfxMessageBox("Error initializing OpenGL");
        }

        return 0;
}

BOOL CDecompressorView::InitializeOpenGL()
{
        m_pDC = new CClientDC(this);

        if (NULL == m_pDC) // failure to capture DC
        {
                TRACE("Error capturing DC\n");
                return FALSE;
        }

        if (!SetupPixelFormat() )
        {
```

```
            TRACE("PixelFormat failed\n");
            return FALSE;
      }

      if (0 == (m_hRC =
            ::wglCreateContext (m_pDC->GetSafeHdc() ) ) )
      {
            TRACE("Failed to create OpenGL context\n");
            return FALSE;
      }

      if (FALSE ==
            ::wglMakeCurrent( m_pDC->GetSafeHdc(), m_hRC ) )
      {
            TRACE("Failed to make OpenGL context the current context\n");
            return FALSE;
      }

      SetupFlags();

      return TRUE;
}

BOOL CDecompressorView::SetupPixelFormat()
{
      PIXELFORMATDESCRIPTOR pfd =
      {
            sizeof(PIXELFORMATDESCRIPTOR),
                  1,
                  PFD_DRAW_TO_WINDOW |
                  PFD_SUPPORT_OPENGL |
                  PFD_DOUBLEBUFFER,
                  PFD_TYPE_RGBA,
                  24,
                  0, 0, 0, 0, 0, 0,
                  0,
                  0,
                  0,
                  0, 0, 0, 0,
                  16,
                  0,
                  0,
                  PFD_MAIN_PLANE,
                  0,
                  0, 0, 0
      };
      int pixelformat;

      if (0 == (pixelformat =
            ::ChoosePixelFormat(m_pDC->GetSafeHdc(), &pfd)) )
      {
            TRACE("Failed to choose a pixel format");
            return FALSE;
      }

      if (FALSE ==
            ::SetPixelFormat(m_pDC->GetSafeHdc(), pixelformat, &pfd) )
      {
```

```
            TRACE("Failed to set chosen pixel format");
            return FALSE;
      }

      return TRUE;
}

////////////////////////////////////////////////////////////////////////
///
///   Animation Display Routines
///

void CDecompressorView::SetupCamera()
{
      CDecompressorDoc* pDoc = GetDocument();
      ASSERT_VALID(pDoc);
      CCamera* SceneCamera = &(pDoc->AnimF->Animation.SceneCamera);

      // first, let's deal with any keyframes the camera might have

      SceneCamera->UpdateTransform(fTotalTime);

      ::glMatrixMode(GL_PROJECTION);
      ::glLoadIdentity();

      float xMin, xMax, yMin, yMax;

/*    float tdist = pDoc->AnimF->Animation.SceneCamera.fNearPlane * 0.5f;
      if (SceneCamera->Target)
      {
            float deltaX = SceneCamera->Target->PosX - SceneCamera->PosX;
            float deltaY = SceneCamera->Target->PosY - SceneCamera->PosY;
            float deltaZ = SceneCamera->Target->PosZ - SceneCamera->PosZ;
            tdist = sqrtf(deltaX*deltaX + deltaY*deltaY + deltaZ*deltaZ);
      }*/

      xMax = pDoc->AnimF->Animation.SceneCamera.fNearPlane *
//    xMax = 2.0f * tdist *
            tanf(pDoc->AnimF->Animation.SceneCamera.fFieldOfView    *    0.5f);

//          0.5f * tanf(pDoc->AnimF->Animation.SceneCamera.fFieldOfView)
//          tanf(pDoc->AnimF->Animation.SceneCamera.fFieldOfView * 0.5f) *
//          tanf(pDoc->AnimF->Animation.SceneCamera.fFieldOfView);
      xMin = -xMax;

      yMax = xMax * aspect_ratio_inv;
      yMin = xMin * aspect_ratio_inv;

    ::glFrustum( xMin, xMax, yMin, yMax,
            pDoc->AnimF->Animation.SceneCamera.fNearPlane,
            pDoc->AnimF->Animation.SceneCamera.fFarPlane );
/*    ::gluPerspective(
            pDoc->AnimF->Animation.SceneCamera.fFieldOfView,
            aspect_ratio,
            pDoc->AnimF->Animation.SceneCamera.fNearPlane,
            pDoc->AnimF->Animation.SceneCamera.fFarPlane);*/

/*          float GLMatrix0[16];
```

```
            // use me to find the matrix values
           ::glGetFloatv(GL_PROJECTION_MATRIX,GLMatrix0);
*/

      if (pDoc->AnimF->Animation.SceneCamera.ParentObject)
            ::glMultMatrixf(pDoc->AnimF->Animation.SceneCamera.
                  ParentObject->Transform);


      float Rotation[16] =
      {
            1.0f, 0.0f, 0.0f, 0.0f,
            0.0f, 1.0f, 0.0f, 0.0f,
            0.0f, 0.0f, 1.0f, 0.0f,
            0.0f, 0.0f, 0.0f, 1.0f,
      };
      float Translation[16] =
      {
            1.0f, 0.0f, 0.0f, 0.0f,
            0.0f, 1.0f, 0.0f, 0.0f,
            0.0f, 0.0f, 1.0f, 0.0f,
            0.0f, 0.0f, 0.0f, 1.0f,
      };

      ::glMultMatrixf(pDoc->AnimF->Animation.SceneCamera.OffsetMatrix);

      // ... perform Quaternion to Matrix translation, and store in our
Matrix
      // this way we generate our orientation
      QuaternionToMatrix(pDoc->AnimF->Animation.SceneCamera.OriW,
            pDoc->AnimF->Animation.SceneCamera.OriX,pDoc->AnimF->Animation.Sce
neCamera.OriY,
            pDoc->AnimF->Animation.SceneCamera.OriZ,Rotation);
      Translation[12] = -pDoc->AnimF->Animation.SceneCamera.PosX;
      Translation[13] = -pDoc->AnimF->Animation.SceneCamera.PosY;
      Translation[14] = -pDoc->AnimF->Animation.SceneCamera.PosZ;

      if (pDoc->AnimF->Animation.SceneCamera.Target)
      {
            float vector[3] =
            {
                  pDoc->AnimF->Animation.SceneCamera.Target->PosX -
                        pDoc->AnimF->Animation.SceneCamera.PosX,
                  pDoc->AnimF->Animation.SceneCamera.Target->PosY -
                        pDoc->AnimF->Animation.SceneCamera.PosY,
                  pDoc->AnimF->Animation.SceneCamera.Target->PosZ -
                        pDoc->AnimF->Animation.SceneCamera.PosZ,
            };
            VectorNormalize(vector);

            // now we construct the B Matrix as listed on Page 9 of Watt &
Watt
            float dot = vector[2];
            float view[3] =
            {
                  - dot * vector[0],
                  - dot * vector[1],
                  1.0f - dot * vector[2],
```

```
            };
            float up[3];
            VectorCrossProduct(vector,view,up);
            float B[16] =
            {
                  up[0], view[0], -vector[0], 0,
                  up[1], view[1], -vector[1], 0,
                  up[2], view[2], -vector[2], 0,
                  0, 0, 0, 1,
            };
            glMultMatrixf(B);
            glRotatef(pDoc->AnimF->Animation.SceneCamera.Roll*180.0f/FLOAT_PI,
0.0f,0.0f,1.0f);
      }
      else ::glMultMatrixf(Rotation);

      float m[16]; glGetFloatv(GL_PROJECTION_MATRIX,m);
      ::glMultMatrixf(Translation);

      ::glMatrixMode(GL_MODELVIEW);
}

void CDecompressorView::SetupLighting()
// Sets up and configures the scene's lights based on animation info
{
      CDecompressorDoc* pDoc = GetDocument();
      // setup lighting for scene

      ::glMatrixMode(GL_MODELVIEW);
      ::glLoadIdentity();

      if (pDoc->AnimF->Animation.LightingEnabled)
      {
/*          while (pDoc->AnimF->Animation.Ambient.CurrPositionEvent &&
                        pDoc->AnimF->Animation.Ambient.CurrPositionEvent->EndF
rame <
                  pDoc->AnimF->Animation.CurrKeyFrameNumber)
            { // increment the Ambient light's keyframe until we have an
event that ends on or
              // after the current keyframe
                  pDoc->AnimF->Animation.Ambient.CurrPositionEvent =
                        pDoc->AnimF->Animation.Ambient.CurrPositionEvent->next
;
            }


            // setup each enabled light one at a time
            if (pDoc->AnimF->Animation.Ambient.bEnabled)
                                    glLightiv(GL_LIGHT0,       GL_AMBIENT,
pDoc->AnimF->Animation.Ambient.color);
            for (int i=0;i<GL_MAX_LIGHTS-1;i++)
                  if (pDoc->AnimF->Animation.SceneLights[i].bEnabled)
                  {
                        glLightfv(GL_LIGHT1+i, GL_POSITION,
                              pDoc->AnimF->Animation.SceneLights[i].Pos);
                        glLightiv(GL_LIGHT1+i, GL_DIFFUSE,
                              pDoc->AnimF->Animation.SceneLights[i].color);
```

```
            }

          // enable each light
        glEnable(GL_LIGHTING);
          if (pDoc->AnimF->Animation.Ambient.bEnabled)
              glEnable(GL_LIGHT0);
          for (i=0;i<GL_MAX_LIGHTS-1;i++)
                if (pDoc->AnimF->Animation.SceneLights[i].bEnabled)
                      glEnable(GL_LIGHT1+i);*/
          int ActiveCount = 0;
          for (int i=0;i<pDoc->AnimF->Animation.iLightsInUse;i++)

                if (pDoc->AnimF->Animation.SceneLights[i].bEnabled)
                {
                      CLight                    *CurrLight                    =
&(pDoc->AnimF->Animation.SceneLights[i]);
                      float Pos[4] = {0.0f, 0.0f, 0.0f, 1.0f,};
                      CurrLight->UpdateTransform(fTotalTime);

                      if (CurrLight->Type == LIGHT_SPOTLIGHT)
                      {
                            Pos[0] = CurrLight->Transform[12];
                            Pos[1] = CurrLight->Transform[13];
                            Pos[2] = CurrLight->Transform[14];
                            ::glLightfv(GL_LIGHT0+ActiveCount,  GL_POSITION,
Pos);
                            if (CurrLight->Target)
                            {
                                  float Dir[3] =
                                  {
                                        CurrLight->Target->Transform[12]    -
CurrLight->Transform[12],
                                        CurrLight->Target->Transform[13]    -
CurrLight->Transform[13],
                                        CurrLight->Target->Transform[14]    -
CurrLight->Transform[14],
                                  };
                                  ::glLightfv(GL_LIGHT0+ActiveCount,
GL_SPOT_DIRECTION, Dir);
                            }
                            else
                            {
                                  float DirMatrix[16] =
                                  {
                                        CurrLight->Transform[0],
CurrLight->Transform[1],
                                        CurrLight->Transform[2], 0.0f,
                                        CurrLight->Transform[4],
CurrLight->Transform[5],
                                        CurrLight->Transform[6], 0.0f,
                                        CurrLight->Transform[8],
CurrLight->Transform[9],
                                        CurrLight->Transform[10], 0.0f,
                                        0.0f, 0.0f, 0.0f, 1.0f,
                                  };
                                  float DAxis[3] =
                                  {
                                        0.0f, 0.0f, 1.0f,
```

```
                                    };
                                    float Dir[3];
                                    VecMatrixMultiply(DAxis,DirMatrix,Dir);
                                    ::glLightfv(GL_LIGHT0+ActiveCount,
GL_SPOT_DIRECTION, Dir);
                              }
                              ::glLightf(GL_LIGHT0+ActiveCount,
GL_SPOT_EXPONENT,
                                    CurrLight->fSpotExponent);
                              ::glLightf(GL_LIGHT0+ActiveCount,
GL_SPOT_CUTOFF, CurrLight->fSpotCutoff);
                              ::glLightfv(GL_LIGHT0+ActiveCount, GL_DIFFUSE,
                                    CurrLight->Color);
                        }
                        else if (CurrLight->Type == LIGHT_DIRECTIONAL)
                        {
                              float Dir[3];
                              if (CurrLight->Target)
                              {
                                    Dir[0] = CurrLight->Target->Transform[12]
- CurrLight->Transform[12];
                                    Dir[1] = CurrLight->Target->Transform[13]
- CurrLight->Transform[13];
                                    Dir[2] = CurrLight->Target->Transform[14]
- CurrLight->Transform[14];
                              }
                              else
                              {
                                    float DirMatrix[16] =
                                    {
                                          CurrLight->Transform[0],
CurrLight->Transform[1],
                                          CurrLight->Transform[2], 0.0f,
                                          CurrLight->Transform[4],
CurrLight->Transform[5],
                                          CurrLight->Transform[6], 0.0f,
                                          CurrLight->Transform[8],
CurrLight->Transform[9],
                                          CurrLight->Transform[10], 0.0f,
                                          0.0f, 0.0f, 0.0f, 1.0f,
                                    };
                                    float DAxis[3] =
                                    {
                                          0.0f, 0.0f, 1.0f,
                                    };
                                    VecMatrixMultiply(DAxis,DirMatrix,Dir);
                              }
                              Pos[0] = Dir[0]; Pos[1] = Dir[1]; Pos[2] =
Dir[2];
                              Pos[3] = 0.0f;
                              ::glLightfv(GL_LIGHT0+ActiveCount, GL_POSITION,
Pos);
                              ::glLightfv(GL_LIGHT0+ActiveCount, GL_DIFFUSE,
                                    CurrLight->Color);
                        }
                        else if (CurrLight->Type == LIGHT_POINT)
                        {
                              Pos[0] = CurrLight->Transform[12];
                              Pos[1] = CurrLight->Transform[13];
```

```
                              Pos[2] = CurrLight->Transform[14];
                              ::glLightfv(GL_LIGHT0+ActiveCount,  GL_POSITION,
Pos);
                              ::glLightfv(GL_LIGHT0+ActiveCount, GL_DIFFUSE,
                                    CurrLight->Color);
                        }
                        else if (CurrLight->Type == LIGHT_AMBIENT)

                              ::glLightfv(GL_LIGHT0+ActiveCount,   GL_AMBIENT,
CurrLight->Color);


                        ::glEnable(GL_LIGHT0+ActiveCount);
                        ActiveCount++;
                  }

      }
      else if (pDoc->AnimF->Animation.bSimpleShading)
      {

            float Pos[4] = { -1.0f, 1.0f, 1.0f, 0.0f};
            float PosR[4] = { 1.0f, -1.0f, -1.0f, 0.0f};
            float ColorA[4] = {0.05f, 0.05f, 0.05f, 1.0f };
            float ColorD[4] = {1.0f, 1.0f, 1.0f, 1.0f };
            float CZero[4] = {0.0f, 0.0f, 0.0f, 0.0f };
            ::glLightfv(GL_LIGHT0, GL_POSITION, Pos);
            ::glLightfv(GL_LIGHT0, GL_DIFFUSE, ColorD);
            ::glLightfv(GL_LIGHT0, GL_AMBIENT, ColorA);
            ::glLightfv(GL_LIGHT1, GL_POSITION, PosR);
            ::glLightfv(GL_LIGHT1, GL_DIFFUSE, ColorD);
            ::glLightfv(GL_LIGHT1, GL_AMBIENT, ColorA);
            ::glShadeModel(GL_SMOOTH);
          ::glEnable(GL_LIGHTING);
            ::glEnable(GL_LIGHT0);
            ::glEnable(GL_LIGHT1);
      }
}

void CDecompressorView::SetupObjects()
// repositions and reorients objects based on animation info
{
      CDecompressorDoc* pDoc = GetDocument();
      // prepare objects for rendering
      CVisualObject *CurrObj = pDoc->AnimF->Animation.ObjectListHead;

      // for all objects...
      while (CurrObj)
      {
            CurrObj->UpdateTransform(fTotalTime);
            CurrObj = CurrObj->next;
      }
}

void CDecompressorView::RenderPatch(CVisualObject * Obj, int * PatchVertices,
float TesRate)
{
      // given an array of 16 vertex references into the Obj,
      // we tesselate a patch out based on the specified rate (which we
```

```
already figured
	// using LOD estimates)
	float Vertices[16][3];
	for (int i=0;i<16;i++)
	{
		Vertices[i][0] = Obj->VertexArray[PatchVertices[i]*3+0];
		Vertices[i][1] = Obj->VertexArray[PatchVertices[i]*3+1];
		Vertices[i][2] = Obj->VertexArray[PatchVertices[i]*3+2];
	}

	// strict OGL calls.  probably would be faster (since we'd be forcing a
4x4 special case)
	// if replaced with own code to do the  same thing for evaluating B-
patches.
	// If nurbs are required... well, maybe not...
	int iTesRate = (int)TesRate;
	glMap2f(GL_MAP2_VERTEX_3, 0,1,3,4, 0,1,12,4, &Vertices[0][0]);
	glEvalMesh2(GL_FILL,0,iTesRate,0,iTesRate);
}

void CDecompressorView::RenderObjects()
// draws the animation objects
{
	const float white_color[4] = { 1.0f, 1.0f, 1.0f, 0.5f };
	const float low_shininess[1] = { 50.0f };
	CDecompressorDoc* pDoc = GetDocument();

	// Begin by first determining the inverse tranform for the camera's
rotation
	// and transformation.  Multiply them together to construct the camera
inverse
	// needed to check for what's within the frustum
	float CameraInverse[16];
	MatrixInverse(pDoc->AnimF->Animation.SceneCamera.Transform,CameraInverse
);

	// now that we have the inverse camera matrix,
	// let's begin rendering the objects
	// prepare objects for rendering
	CVisualObject *CurrObj = pDoc->AnimF->Animation.ObjectListHead;
	while (CurrObj)
	{
		CurrObj->UpdateTransform(fTotalTime);
		::glLoadIdentity();
		::glMultMatrixf(CurrObj->Transform);
		::glMultMatrixf(CurrObj->OffsetMatrix);

		// now, since the object is inside, at least partially, we draw
it

		if (CurrObj->WithinFustrum(CameraInverse,
			pDoc->AnimF->Animation.SceneCamera.fNearPlane,
			pDoc->AnimF->Animation.SceneCamera.fFarPlane) )
		{
			CurrObj->CheckComplete();
			float normal[3];
			if (CurrObj->bCompleted && CurrObj->CurrentTotalNormals == 0
```

```
&&
                        !pDoc->AnimF->Animation.bVertexNormals)
                        CurrObj->GenerateVertexNormals();

                  if (CurrObj->bCompleted)
                  {
                        if (CurrObj->bSolidColor)
                        {
                        // these two commands cause the vertex color
                        // to be used as the object's material property,
                        // allowing for per-vertex color (which we
                        // generally want).
                              ::glEnable(GL_COLOR_MATERIAL);
                              ::glColorMaterial(GL_FRONT, GL_DIFFUSE);
                              ::glMaterialfv(GL_FRONT,          GL_SPECULAR,
white_color);
                              ::glMaterialfv(GL_FRONT,          GL_SHININESS,
low_shininess);
                              ::glColor3ub(CurrObj->solidRed,
                                    CurrObj->solidGreen,CurrObj->solidBlue);
                              ::glEnableClientState(GL_VERTEX_ARRAY);
                              ::glEnableClientState(GL_NORMAL_ARRAY);
                              ::glVertexPointer(3,GL_FLOAT,0,CurrObj->VertexAr
ray);
                              ::glNormalPointer(GL_FLOAT,0,CurrObj->NormalArra
y);
                              ::glDrawElements(GL_TRIANGLES,CurrObj->TotalFace
s*3,
                                    GL_UNSIGNED_INT,CurrObj->FaceArray);
                              ::glDisableClientState(GL_VERTEX_ARRAY);
                              ::glDisableClientState(GL_NORMAL_ARRAY);
                              ::glDisable(GL_COLOR_MATERIAL);
                        }
                        else
                        {
                              if (!pDoc->AnimF->Animation.bMaterialQuality)
                              {
                                    ::glEnable(GL_COLOR_MATERIAL);
                                    ::glColorMaterial(GL_FRONT, GL_DIFFUSE);
                                    ::glMaterialfv(GL_FRONT,      GL_SPECULAR,
white_color);
                                    ::glMaterialfv(GL_FRONT,      GL_SHININESS,
low_shininess);
                                    ::glEnableClientState(GL_VERTEX_ARRAY);
                                    ::glEnableClientState(GL_NORMAL_ARRAY);
                                    ::glEnableClientState(GL_COLOR_ARRAY);
                                    ::glVertexPointer(3,GL_FLOAT,0,CurrObj->Ve
rtexArray);
                                    ::glNormalPointer(GL_FLOAT,0,CurrObj->Norm
alArray);
                                    ::glColorPointer(3,GL_UNSIGNED_BYTE,0,Curr
Obj->VertexColorArray);
                                    ::glDrawElements(GL_TRIANGLES,CurrObj->Tot
alFaces*3,
                                          GL_UNSIGNED_INT,CurrObj->FaceArray);
                                    ::glDisableClientState(GL_VERTEX_ARRAY);
                                    ::glDisableClientState(GL_NORMAL_ARRAY);
                                    ::glDisableClientState(GL_COLOR_ARRAY);
                                    ::glDisable(GL_COLOR_MATERIAL);
```

```
                        }
                        else
                        {
                              float shine[1] = {      5.0f };
                              float spec[4] =
                              {
                                    1.0f, 1.0f, 1.0f, 1.0f
                              };
                              float *specular = spec;
                              // render with texture coords
                              ::glEnable(GL_COLOR_MATERIAL);
                              ::glColorMaterial(GL_FRONT, GL_DIFFUSE);
                              ::glColor3ub(255,255,255);
                              if          (CurrObj->TextureMap          &&
CurrObj->TextureMap->bTextureInUse)
                              {
                                    ::glEnable(GL_TEXTURE_2D);
                                    if
(CurrObj->TextureMap->bTextureUnbound)
                                          ReBindTexture(CurrObj->Texture
Map);
                                    ::glBindTexture(GL_TEXTURE_2D,CurrOb
j->TextureMap->iTextureBindNum);
                                    shine[0]                              =
CurrObj->TextureMap->fShininess;
                                    specular                              =
CurrObj->TextureMap->SpecularColor;
                              }
                              ::glMaterialfv(GL_FRONT,      GL_SPECULAR,
specular);
                              ::glMaterialfv(GL_FRONT,     GL_SHININESS,
shine);
                              ::glEnableClientState(GL_VERTEX_ARRAY);
                              ::glEnableClientState(GL_NORMAL_ARRAY);
                              ::glEnableClientState(GL_TEXTURE_COORD_ARR
AY);
                              ::glVertexPointer(3,GL_FLOAT,0,CurrObj->Ve
rtexArray);
                              ::glNormalPointer(GL_FLOAT,0,CurrObj->Norm
alArray);
                              ::glTexCoordPointer(2,GL_FLOAT,0,CurrObj->
TexCoordArray);
                              ::glDrawElements(GL_TRIANGLES,CurrObj->Tot
alFaces*3,
                                    GL_UNSIGNED_INT,CurrObj->FaceArray);
                              ::glDisable(GL_TEXTURE_2D);
                              ::glDisableClientState(GL_VERTEX_ARRAY);
                              ::glDisableClientState(GL_NORMAL_ARRAY);
                              ::glDisableClientState(GL_TEXTURE_COORD_AR
RAY);
                              ::glDisable(GL_COLOR_MATERIAL);
                        }
                  }
            }
            else
            // begin by drawing the straight triangle faces
            if (CurrObj->CurrentTotalNormals>0)
            {
                  if (CurrObj->CurrentTotalColors > 0)
```

```
                        {
                                ::glBegin(GL_TRIANGLES);
                                for (int i=0;i<CurrObj->CurrentTotalFaces;i++)
                                {
                                        normal[0]                                       =
(CurrObj->NormalArray[CurrObj->FaceArray[3*i]] +

CurrObj->NormalArray[CurrObj->FaceArray[3*i+1]] +

CurrObj->NormalArray[CurrObj->FaceArray[3*i+2]]) * 0.333333333f;
                                        normal[1]                                       =
(CurrObj->NormalArray[CurrObj->FaceArray[3*i]+1] +

CurrObj->NormalArray[CurrObj->FaceArray[3*i+1]+1] +

CurrObj->NormalArray[CurrObj->FaceArray[3*i+2]+1]) * 0.333333333f;
                                        normal[2]                                       =
(CurrObj->NormalArray[CurrObj->FaceArray[3*i]+2] +

CurrObj->NormalArray[CurrObj->FaceArray[3*i+1]+2] +

CurrObj->NormalArray[CurrObj->FaceArray[3*i+2]+2]) * 0.333333333f;

                                        ::glNormal3f(normal[0],normal[1],normal[2]
);
                                        ::glColor3ub(CurrObj->VertexColorArray[Cur
rObj->FaceArray[i*3]*3],

CurrObj->VertexColorArray[CurrObj->FaceArray[i*3]*3+1],

CurrObj->VertexColorArray[CurrObj->FaceArray[i*3]*3+2]);
                                        ::glVertex3f(CurrObj->VertexArray[CurrObj-
>FaceArray[i*3]*3],

CurrObj->VertexArray[CurrObj->FaceArray[i*3]*3+1],

CurrObj->VertexArray[CurrObj->FaceArray[i*3]*3+2]);
                                        ::glColor3ub(CurrObj->VertexColorArray[Cur
rObj->FaceArray[i*3+1]*3],

CurrObj->VertexColorArray[CurrObj->FaceArray[i*3+1]*3+1],

CurrObj->VertexColorArray[CurrObj->FaceArray[i*3+1]*3+2]);
                                        ::glVertex3f(CurrObj->VertexArray[CurrObj-
>FaceArray[i*3+1]*3],

CurrObj->VertexArray[CurrObj->FaceArray[i*3+1]*3+1],

CurrObj->VertexArray[CurrObj->FaceArray[i*3+1]*3+2]);
                                        ::glColor3ub(CurrObj->VertexColorArray[Cur
rObj->FaceArray[i*3+2]*3],

CurrObj->VertexColorArray[CurrObj->FaceArray[i*3+2]*3+1],

CurrObj->VertexColorArray[CurrObj->FaceArray[i*3+2]*3+2]);
                                        ::glVertex3f(CurrObj->VertexArray[CurrObj-
>FaceArray[i*3+2]*3],

CurrObj->VertexArray[CurrObj->FaceArray[i*3+2]*3+1],
```

```
CurrObj->VertexArray[CurrObj->FaceArray[i*3+2]*3+2]);
                                }
                                ::glEnd();
                        }
                        else if (CurrObj->bSolidColor)
                        {
                                ::glEnable(GL_COLOR_MATERIAL);
                                ::glColorMaterial(GL_FRONT, GL_DIFFUSE);
                                ::glColor3ub(CurrObj->solidRed,CurrObj->solidGre
en,CurrObj->solidBlue);
                                ::glBegin(GL_TRIANGLES);
                                for (int i=0;i<CurrObj->CurrentTotalFaces;i++)
                                {

                                        ::glNormal3f(CurrObj->NormalArray[CurrObj-
>FaceArray[i*3]*3],

CurrObj->NormalArray[CurrObj->FaceArray[i*3]*3+1],

CurrObj->NormalArray[CurrObj->FaceArray[i*3]*3+2]);
                                        ::glVertex3f(CurrObj->VertexArray[CurrObj-
>FaceArray[i*3]*3],

CurrObj->VertexArray[CurrObj->FaceArray[i*3]*3+1],

CurrObj->VertexArray[CurrObj->FaceArray[i*3]*3+2]);
                                        ::glNormal3f(CurrObj->NormalArray[CurrObj-
>FaceArray[i*3+1]*3],

CurrObj->NormalArray[CurrObj->FaceArray[i*3+1]*3+1],

CurrObj->NormalArray[CurrObj->FaceArray[i*3+1]*3+2]);
                                        ::glVertex3f(CurrObj->VertexArray[CurrObj-
>FaceArray[i*3+1]*3],

CurrObj->VertexArray[CurrObj->FaceArray[i*3+1]*3+1],

CurrObj->VertexArray[CurrObj->FaceArray[i*3+1]*3+2]);
                                        ::glNormal3f(CurrObj->NormalArray[CurrObj-
>FaceArray[i*3+2]*3],

CurrObj->NormalArray[CurrObj->FaceArray[i*3+2]*3+1],

CurrObj->NormalArray[CurrObj->FaceArray[i*3+2]*3+2]);
                                        ::glVertex3f(CurrObj->VertexArray[CurrObj-
>FaceArray[i*3+2]*3],

CurrObj->VertexArray[CurrObj->FaceArray[i*3+2]*3+1],

CurrObj->VertexArray[CurrObj->FaceArray[i*3+2]*3+2]);
                                }
                                ::glEnd();
                        }
                        else          if          (CurrObj->TextureMap          &&
CurrObj->TextureMap->bTextureInUse)
                        {
                                ::glEnable(GL_TEXTURE_2D);
                                ::glEnable(GL_COLOR_MATERIAL);
```

```
                                ::glColorMaterial(GL_FRONT, GL_DIFFUSE);
                                if (CurrObj->TextureMap->bTextureUnbound)
                                      ReBindTexture(CurrObj->TextureMap);
                                ::glBindTexture(GL_TEXTURE_2D,CurrObj->TextureMa
p->iTextureBindNum);

                                ::glBegin(GL_TRIANGLES);
                                ::glColor3ub( 255,255,255 );
                                for (int i=0;i<CurrObj->CurrentTotalFaces;i++)
                                {
                                      ::glNormal3f(CurrObj->NormalArray[CurrObj-
>FaceArray[i*3]*3],

CurrObj->NormalArray[CurrObj->FaceArray[i*3]*3+1],

CurrObj->NormalArray[CurrObj->FaceArray[i*3]*3+2]);
                                      ::glTexCoord2f(CurrObj->TexCoordArray[Curr
Obj->FaceArray[i*3]*2],

CurrObj->TexCoordArray[CurrObj->FaceArray[i*3]*2+1]);
                                      ::glVertex3f(CurrObj->VertexArray[CurrObj-
>FaceArray[i*3]*3],

CurrObj->VertexArray[CurrObj->FaceArray[i*3]*3+1],

CurrObj->VertexArray[CurrObj->FaceArray[i*3]*3+2]);

                                      ::glNormal3f(CurrObj->NormalArray[CurrObj-
>FaceArray[i*3+1]*3],

CurrObj->NormalArray[CurrObj->FaceArray[i*3+1]*3+1],

CurrObj->NormalArray[CurrObj->FaceArray[i*3+1]*3+2]);
                                      ::glTexCoord2f(CurrObj->TexCoordArray[Curr
Obj->FaceArray[i*3+1]*2],

CurrObj->TexCoordArray[CurrObj->FaceArray[i*3+1]*2+1]);
                                      ::glVertex3f(CurrObj->VertexArray[CurrObj-
>FaceArray[i*3+1]*3],

CurrObj->VertexArray[CurrObj->FaceArray[i*3+1]*3+1],

CurrObj->VertexArray[CurrObj->FaceArray[i*3+1]*3+2]);

                                      ::glNormal3f(CurrObj->NormalArray[CurrObj-
>FaceArray[i*3+2]*3],

CurrObj->NormalArray[CurrObj->FaceArray[i*3+2]*3+1],

CurrObj->NormalArray[CurrObj->FaceArray[i*3+2]*3+2]);
                                      ::glTexCoord2f(CurrObj->TexCoordArray[Curr
Obj->FaceArray[i*3+2]*2],

CurrObj->TexCoordArray[CurrObj->FaceArray[i*3+2]*2+1]);
                                      ::glVertex3f(CurrObj->VertexArray[CurrObj-
>FaceArray[i*3+2]*3],

CurrObj->VertexArray[CurrObj->FaceArray[i*3+2]*3+1],
```

```
CurrObj->VertexArray[CurrObj->FaceArray[i*3+2]*3+2]);
                                }
                                ::glEnd();
                        }
                }
                else
                {
                        if (CurrObj->CurrentTotalColors > 0)
                        {
                                ::glBegin(GL_TRIANGLES);
                                for (int i=0;i<CurrObj->CurrentTotalFaces;i++)
                                {
                                        ::glColor3ub(CurrObj->VertexColorArray[Cur
rObj->FaceArray[i*3]*3],

CurrObj->VertexColorArray[CurrObj->FaceArray[i*3]*3+1],

CurrObj->VertexColorArray[CurrObj->FaceArray[i*3]*3+2]);
                                        ::glVertex3f(CurrObj->VertexArray[CurrObj-
>FaceArray[i*3]*3],

CurrObj->VertexArray[CurrObj->FaceArray[i*3]*3+1],

CurrObj->VertexArray[CurrObj->FaceArray[i*3]*3+2]);
                                        ::glColor3ub(CurrObj->VertexColorArray[Cur
rObj->FaceArray[i*3+1]*3],

CurrObj->VertexColorArray[CurrObj->FaceArray[i*3+1]*3+1],

CurrObj->VertexColorArray[CurrObj->FaceArray[i*3+1]*3+2]);
                                        ::glVertex3f(CurrObj->VertexArray[CurrObj-
>FaceArray[i*3+1]*3],

CurrObj->VertexArray[CurrObj->FaceArray[i*3+1]*3+1],

CurrObj->VertexArray[CurrObj->FaceArray[i*3+1]*3+2]);
                                        ::glColor3ub(CurrObj->VertexColorArray[Cur
rObj->FaceArray[i*3+2]*3],

CurrObj->VertexColorArray[CurrObj->FaceArray[i*3+2]*3+1],

CurrObj->VertexColorArray[CurrObj->FaceArray[i*3+2]*3+2]);
                                        ::glVertex3f(CurrObj->VertexArray[CurrObj-
>FaceArray[i*3+2]*3],

CurrObj->VertexArray[CurrObj->FaceArray[i*3+2]*3+1],

CurrObj->VertexArray[CurrObj->FaceArray[i*3+2]*3+2]);
                                }
                                ::glEnd();
                        }
                        else
                        {
                                if (pDoc->AnimF->Animation.bSimpleShading)
                                {

                                        if (CurrObj->bSolidColor)
                                        {
                                                ::glEnable(GL_COLOR_MATERIAL);
```

```
::glColorMaterial(GL_FRONT,
GL_DIFFUSE);
::glColor3ub(CurrObj->solidRed,CurrO
bj->solidGreen,CurrObj->solidBlue);
::glBegin(GL_TRIANGLES);
for                                    (int
i=0;i<CurrObj->CurrentTotalFaces;i++)
{
      float Normal[3];
      // calculate normal
      float Vec0[3] =
      {
            CurrObj->VertexArray[Cur
rObj->FaceArray[i*3+1]*3] -
                  CurrObj->VertexArr
ay[CurrObj->FaceArray[i*3]*3],
            CurrObj->VertexArray[Cur
rObj->FaceArray[i*3+1]*3+1] -
                  CurrObj->VertexArr
ay[CurrObj->FaceArray[i*3]*3+1],
            CurrObj->VertexArray[Cur
rObj->FaceArray[i*3+1]*3+2] -
                  CurrObj->VertexArr
ay[CurrObj->FaceArray[i*3]*3+2]
      };
      float Vec1[3] =
      {
            CurrObj->VertexArray[Cur
rObj->FaceArray[i*3+1]*3] -
                  CurrObj->VertexArr
ay[CurrObj->FaceArray[i*3+2]*3],
            CurrObj->VertexArray[Cur
rObj->FaceArray[i*3+1]*3+1] -
                  CurrObj->VertexArr
ay[CurrObj->FaceArray[i*3+2]*3+1],
            CurrObj->VertexArray[Cur
rObj->FaceArray[i*3+1]*3+2] -
                  CurrObj->VertexArr
ay[CurrObj->FaceArray[i*3+2]*3+2],
      };
      VectorCrossProduct(Vec0,Vec1,N
ormal);
      VectorNormalize(Normal);
      ::glNormal3fv(Normal);
      ::glVertex3f(CurrObj->VertexAr
ray[CurrObj->FaceArray[i*3]*3],
CurrObj->VertexArray[CurrObj->FaceArray[i*3]*3+1],
CurrObj->VertexArray[CurrObj->FaceArray[i*3]*3+2]);
      ::glVertex3f(CurrObj->VertexAr
ray[CurrObj->FaceArray[i*3+1]*3],
CurrObj->VertexArray[CurrObj->FaceArray[i*3+1]*3+1],
CurrObj->VertexArray[CurrObj->FaceArray[i*3+1]*3+2]);
      ::glVertex3f(CurrObj->VertexAr
ray[CurrObj->FaceArray[i*3+2]*3],
```

```
CurrObj->VertexArray[CurrObj->FaceArray[i*3+2]*3+1],
CurrObj->VertexArray[CurrObj->FaceArray[i*3+2]*3+2]);
                                        }
                                        ::glEnd();
                                }
                                else
                                {
                                        if      (CurrObj->TextureMap      &&
CurrObj->TextureMap->bTextureInUse)
                                        {
                                                ::glEnable(GL_TEXTURE_2D);
                                                ::glEnable(GL_COLOR_MATERIAL);
                                                ::glColorMaterial(GL_FRONT,
GL_DIFFUSE);
                                                if
(CurrObj->TextureMap->bTextureUnbound)
                                                        ReBindTexture(CurrObj->T
extureMap);
                                                ::glBindTexture(GL_TEXTURE_2D,
CurrObj->TextureMap->iTextureBindNum);
                                                ::glBegin(GL_TRIANGLES);
                                                ::glColor3ub(255,255,255);
                                                for                     (int
i=0;i<CurrObj->CurrentTotalFaces;i++)
                                                {
                                                        float Normal[3];
                                                        // calculate normal
                                                        float Vec0[3] =
                                                        {
                                                                CurrObj->VertexArr
ay[CurrObj->FaceArray[i*3+1]*3] -
                                                                        CurrObj->Ver
texArray[CurrObj->FaceArray[i*3]*3],
                                                                CurrObj->VertexArr
ay[CurrObj->FaceArray[i*3+1]*3+1] -
                                                                        CurrObj->Ver
texArray[CurrObj->FaceArray[i*3]*3+1],
                                                                CurrObj->VertexArr
ay[CurrObj->FaceArray[i*3+1]*3+2] -
                                                                        CurrObj->Ver
texArray[CurrObj->FaceArray[i*3]*3+2]
                                                        };
                                                        float Vec1[3] =
                                                        {
                                                                CurrObj->VertexArr
ay[CurrObj->FaceArray[i*3+1]*3] -
                                                                        CurrObj->Ver
texArray[CurrObj->FaceArray[i*3+2]*3],
                                                                CurrObj->VertexArr
ay[CurrObj->FaceArray[i*3+1]*3+1] -
                                                                        CurrObj->Ver
texArray[CurrObj->FaceArray[i*3+2]*3+1],
                                                                CurrObj->VertexArr
ay[CurrObj->FaceArray[i*3+1]*3+2] -
                                                                        CurrObj->Ver
texArray[CurrObj->FaceArray[i*3+2]*3+2],
                                                        };
```

```
                                                    VectorCrossProduct(Vec0,
Vec1,Normal);
                                                    VectorNormalize(Normal);


                                                    ::glNormal3fv(Normal);
                                                    ::glTexCoord2f(CurrObj->
TexCoordArray[CurrObj->FaceArray[i*3]*2],
CurrObj->TexCoordArray[CurrObj->FaceArray[i*3]*2+1]);
                                                    ::glVertex3f(CurrObj->Ve
rtexArray[CurrObj->FaceArray[i*3]*3],
CurrObj->VertexArray[CurrObj->FaceArray[i*3]*3+1],
CurrObj->VertexArray[CurrObj->FaceArray[i*3]*3+2]);
                                                    ::glTexCoord2f(CurrObj->
TexCoordArray[CurrObj->FaceArray[i*3+1]*2],
CurrObj->TexCoordArray[CurrObj->FaceArray[i*3+1]*2+1]);
                                                    ::glVertex3f(CurrObj->Ve
rtexArray[CurrObj->FaceArray[i*3+1]*3],
CurrObj->VertexArray[CurrObj->FaceArray[i*3+1]*3+1],
CurrObj->VertexArray[CurrObj->FaceArray[i*3+1]*3+2]);
                                                    ::glTexCoord2f(CurrObj->
TexCoordArray[CurrObj->FaceArray[i*3+2]*2],
CurrObj->TexCoordArray[CurrObj->FaceArray[i*3+2]*2+1]);
                                                    ::glVertex3f(CurrObj->Ve
rtexArray[CurrObj->FaceArray[i*3+2]*3],
CurrObj->VertexArray[CurrObj->FaceArray[i*3+2]*3+1],
CurrObj->VertexArray[CurrObj->FaceArray[i*3+2]*3+2]);
                                                }
                                                ::glEnd();
                                            }
                                            else
                                            {
                                            ::glEnable(GL_COLOR_MATERIAL);
                                            ::glColorMaterial(GL_FRONT,
GL_DIFFUSE);
                                            ::glBegin(GL_TRIANGLES);
                                            for                                (int
i=0;i<CurrObj->CurrentTotalFaces;i++)
                                            {
                                                float Normal[3];
                                                // calculate normal
                                                float Vec0[3] =
                                                {
                                                    CurrObj->VertexArray[Cur
rObj->FaceArray[i*3+1]*3] -
                                                        CurrObj->VertexArr
ay[CurrObj->FaceArray[i*3]*3],
                                                    CurrObj->VertexArray[Cur
rObj->FaceArray[i*3+1]*3+1] -
                                                        CurrObj->VertexArr
```

```
ay[CurrObj->FaceArray[i*3]*3+1],
                                                CurrObj->VertexArray[Cur
rObj->FaceArray[i*3+1]*3+2] -
                                                      CurrObj->VertexArr
ay[CurrObj->FaceArray[i*3]*3+2]
                                          };
                                          float Vec1[3] =
                                          {
                                                CurrObj->VertexArray[Cur
rObj->FaceArray[i*3+1]*3] -
                                                      CurrObj->VertexArr
ay[CurrObj->FaceArray[i*3+2]*3],
                                                CurrObj->VertexArray[Cur
rObj->FaceArray[i*3+1]*3+1] -
                                                      CurrObj->VertexArr
ay[CurrObj->FaceArray[i*3+2]*3+1],
                                                CurrObj->VertexArray[Cur
rObj->FaceArray[i*3+1]*3+2] -
                                                      CurrObj->VertexArr
ay[CurrObj->FaceArray[i*3+2]*3+2],
                                          };
                                          VectorCrossProduct(Vec0,Vec1,N
ormal);
                                          VectorNormalize(Normal);
                                          ::glNormal3fv(Normal);
                                          ::glColor3ub( 255,0,0);
                                          ::glVertex3f(CurrObj->VertexAr
ray[CurrObj->FaceArray[i*3]*3],
CurrObj->VertexArray[CurrObj->FaceArray[i*3]*3+1],
CurrObj->VertexArray[CurrObj->FaceArray[i*3]*3+2]);
                                          ::glColor3ub( 0,255,0);
                                          ::glVertex3f(CurrObj->VertexAr
ray[CurrObj->FaceArray[i*3+1]*3],
CurrObj->VertexArray[CurrObj->FaceArray[i*3+1]*3+1],
CurrObj->VertexArray[CurrObj->FaceArray[i*3+1]*3+2]);
                                          ::glColor3ub( 0,0,255);
                                          ::glVertex3f(CurrObj->VertexAr
ray[CurrObj->FaceArray[i*3+2]*3],
CurrObj->VertexArray[CurrObj->FaceArray[i*3+2]*3+1],
CurrObj->VertexArray[CurrObj->FaceArray[i*3+2]*3+2]);
                                    }
                                    ::glEnd();
                                    }
                              }
                        }
                        else
                        {
                              ::glBegin(GL_TRIANGLES);
                              for                                    (int
i=0;i<CurrObj->CurrentTotalFaces;i++)
                              {
                                    ::glColor3ub( 255,0,0);
```

```
                                        ::glVertex3f(CurrObj->VertexArray[Cu
rrObj->FaceArray[i*3]*3],

CurrObj->VertexArray[CurrObj->FaceArray[i*3]*3+1],

CurrObj->VertexArray[CurrObj->FaceArray[i*3]*3+2]);
                                        ::glColor3ub( 0,255,0);
                                        ::glVertex3f(CurrObj->VertexArray[Cu
rrObj->FaceArray[i*3+1]*3],

CurrObj->VertexArray[CurrObj->FaceArray[i*3+1]*3+1],

CurrObj->VertexArray[CurrObj->FaceArray[i*3+1]*3+2]);
                                        ::glColor3ub( 0,0,255);
                                        ::glVertex3f(CurrObj->VertexArray[Cu
rrObj->FaceArray[i*3+2]*3],

CurrObj->VertexArray[CurrObj->FaceArray[i*3+2]*3+1],

CurrObj->VertexArray[CurrObj->FaceArray[i*3+2]*3+2]);
                                    }
                                    ::glEnd();
                                }
                            }
                        }
                }

                CurrObj = CurrObj->next;
        }
}

BOOL CDecompressorView::RenderScene()
{
        BOOL bWaiting = FALSE;

        if (!bStopped && !bWaiting) dwFrames++;

        CDecompressorDoc* pDoc = GetDocument();
        pDoc->AnimF->Animation.OldTime = pDoc->AnimF->Animation.CurTime;
        pDoc->AnimF->Animation.CurTime = timeGetTime();

        dwTimePerDivision+=          (pDoc->AnimF->Animation.CurTime          -
pDoc->AnimF->Animation.OldTime);

        if (bSliderCreated &&
                ((float)        pSlider->GetPos()        *        0.01f)        >=
pDoc->AnimF->Animation.ReadyTillFrame)
        {
//              pSlider->SetPos(pDoc->AnimF->Animation.ReadyTillFrame * 100);
                bWaiting = TRUE;
        }
        float fNewTime = (float) pSlider->GetPos() /
                ((float) pDoc->AnimF->Animation.iSourceFPS * 100.0f);
        float dif = FABS(fNewTime - fTotalTime);
        if (dif > 0.005)
        {
                fTotalTime     =     (float)     pSlider->GetPos()     /     ((float)
pDoc->AnimF->Animation.iSourceFPS * 100.0f);
                fNextBlock = ceilf(fTotalTime);
```

```
            dwFrames = 0;
            dwTimePerDivision = 0;
            pDoc->AnimF->Animation.SceneCamera.Reset();
            for (int i=0;i<GL_MAX_LIGHTS;i++)
                  pDoc->AnimF->Animation.SceneLights->Reset();
            CVisualObject* Curr = pDoc->AnimF->Animation.ObjectListHead;
            while (Curr)
            {
                  Curr->Reset();
                  Curr = Curr->next;
            }
            bStartingRendering = TRUE;
            pDoc->AnimF->Animation.FramesSoFar = fTotalTime *
                              (float) pDoc->AnimF->Animation.iSourceFPS ;
      }

// calculates overall framerate
      if (!(bStopped || bWaiting) && fTotalTime >= fNextBlock)
      {
            LogWindow.m_fps = (float) dwFrames / ( (float) dwTimePerDivision *
0.001f );
            LogWindow.UpdateData(FALSE);
// Uncomment to calculate current framerate
//          dwFrames = 0;
//          dwTimePerDivision = 0;
            fNextBlock+=1.0f;
      }

      fTime         =         (float)(pDoc->AnimF->Animation.CurTime         -
pDoc->AnimF->Animation.OldTime) / 1000.0f;
      if (!bStopped && !bWaiting)
            fTotalTime+=fTime;

      // if an initial "go ahead" hasn't been specified, we really don't need
      // to proceed
      if (pDoc->AnimF->Animation.bTimeNotSet)
            return FALSE;

      if  (!(bStopped  ||  bWaiting)  &&  pDoc->AnimF->Animation.FramesSoFar  >=
0.0f)
      {
            if      (fabsf(pDoc->AnimF->Animation.FramesSoFar+0.0005f)      >=
pDoc->AnimF->Animation.ReadyTillFrame)
            {
                  if         (pDoc->AnimF->Animation.ReadyTillFrame         >=
pDoc->AnimF->Animation.FinalFrame)
                  {
                        bStopped = TRUE;
                        bEndedNaturally = TRUE;
                  }
            }
            else
            {
                  int oldTime = pSlider->GetPos();
                  int newTime = (int) (fTotalTime *
                              (float)    pDoc->AnimF->Animation.iSourceFPS    *
100.0f);
                  pDoc->AnimF->Animation.FramesSoFar+=    fTime    *    (float)
```

```
pDoc->AnimF->Animation.iSourceFPS;
                    if (bSliderCreated && oldTime != newTime)
                        pSlider->SetPos(newTime);
                }
        }

        SetupObjects();

        SetupCamera();

        SetupLighting();

        if (bStartingRendering)
        {
                bStartingRendering = FALSE;
                return FALSE;
        }

        RenderObjects();
        return TRUE;
}

void CDecompressorView::SetupFlags()
{
        ::glClearColor (0.0f,0.0f,1.0f,0.0f);
        ::glClearDepth(1.0f);
        ::glEnable(GL_DEPTH_TEST);
}

void CDecompressorView::OnDestroy()
{
        CView::OnDestroy();

        if (FALSE == ::wglMakeCurrent(0, 0) )
        {
                TRACE("Error cleaning up #1");
        }

        if (FALSE == ::wglDeleteContext(m_hRC) )
        {
                TRACE("Error cleaning up #2");
        }

        if (m_pDC)
                delete m_pDC;
}

void CDecompressorView::OnSize(UINT nType, int cx, int cy)
{
        CView::OnSize(nType, cx, cy);

        if ( 0>= cx || 0>= cy)
                return;

        ::glViewport(0,0,cx,cy);
        aspect_ratio = (float)cx/(float)cy;
        aspect_ratio_inv = 1.0f / aspect_ratio;
```

```
	// this resizes the slider along the bottom of the screen
	CRect Region;
	if (bSliderCreated)
	{
		pSlider->GetWindowRect(Region);
		pSlider->MoveWindow(70,0,cx-70-10,20);
	}
}



BOOL CDecompressorView::OnEraseBkgnd(CDC* pDC)
{
	return TRUE;
}

void CDecompressorView::OnFileMruFile1()
{
	FileProcess(0);
}

void CDecompressorView::OnFileMruFile2()
{
	FileProcess(1);
}

void CDecompressorView::OnFileMruFile3()
{
	FileProcess(2);
}

void CDecompressorView::OnFileMruFile4()
{
	FileProcess(3);
}

CDecompressorView::PlayAnimation()
{
	CDecompressorDoc* pDoc = GetDocument();
	ASSERT_VALID(pDoc);
	bStopped = FALSE;
	pDoc->AnimF->bAnimationInProgress = TRUE;
	if (bEndedNaturally)
	{
		dwFrames = 0;
		dwTimePerDivision = 0;
		pDoc->AnimF->Animation.SceneCamera.Reset();
		for (int i=0;i<GL_MAX_LIGHTS;i++)
			pDoc->AnimF->Animation.SceneLights->Reset();
		CVisualObject* Curr = pDoc->AnimF->Animation.ObjectListHead;
		while (Curr)
		{
			Curr->Reset();
			Curr = Curr->next;
		}
		bStartingRendering = TRUE;
		pDoc->AnimF->Animation.FramesSoFar = 0.0f;
		pSlider->SetPos(0);
```

```
            fTotalTime = 0.0f;
            fNextBlock = 1.0f;
      }
}

void CDecompressorView::FileProcess(int number)
{
      CDecompressorDoc* pDoc = GetDocument();
      ASSERT_VALID(pDoc);

      // this doesn't actually play the animation, it just
      // sets all the internal variables that allow the
      // engine to playback an animation.  That's why
      // its set before the Open() command, as the file
      // opening must be completed before the function
      // returns.
      bStartingUpLevel1 = TRUE;
      bIsPlayAble = TRUE;
      bEndedNaturally = TRUE;
      PlayAnimation();

      if (pDoc->AnimF)
      {
            delete pDoc->AnimF;
            pDoc->AnimF = new CAnimationFile;
      }

      // now to grab the url name
      CRecentFileList MRU(1, "Recent File List", "File%d", 4);
      MRU.ReadList();

      if (!pDoc->AnimF->Open(MRU[number]))
      {
            AfxMessageBox("Unable to Open specified URL");
            bIsPlayAble = FALSE;
            return;
      }

}


void CDecompressorView::OnAnimationPlay()
{
      PlayAnimation();
}

void CDecompressorView::OnUpdateAnimationPlay(CCmdUI* pCmdUI)
{
      CDecompressorDoc* pDoc = GetDocument();
      ASSERT_VALID(pDoc);
//    pCmdUI->Enable(!pDoc->AnimF->bAnimationInProgress    &&    bIsPlayAble);

      pCmdUI->Enable(bStopped           &&           bIsPlayAble           &&
!pDoc->AnimF->bAbortAnimation);
}

void CDecompressorView::OnAnimationPause()
{
```

```
	CDecompressorDoc* pDoc = GetDocument();
	ASSERT_VALID(pDoc);
	bStopped = TRUE;
	bEndedNaturally = FALSE;
}

void CDecompressorView::OnUpdateAnimationPause(CCmdUI* pCmdUI)
{
	CDecompressorDoc* pDoc = GetDocument();
	ASSERT_VALID(pDoc);
	pCmdUI->Enable(!bStopped		&&		bIsPlayAble		&&
!pDoc->AnimF->bAbortAnimation);
}

void CDecompressorView::OnAnimationStop()
{
	CDecompressorDoc* pDoc = GetDocument();
	ASSERT_VALID(pDoc);
	bStopped = TRUE;
	fTotalTime = 0.0f;
	fNextBlock = 1.0f;
	dwFrames = 0;
	dwTimePerDivision = 0;
	pDoc->AnimF->Animation.SceneCamera.Reset();
	for (int i=0;i<GL_MAX_LIGHTS;i++)
		pDoc->AnimF->Animation.SceneLights->Reset();
	CVisualObject* Curr = pDoc->AnimF->Animation.ObjectListHead;
	while (Curr)
	{
		Curr->Reset();
		Curr = Curr->next;
	}
	bStartingRendering = TRUE;
	pDoc->AnimF->Animation.FramesSoFar = 0.0f;
	pSlider->SetPos(0);
	bEndedNaturally = FALSE;
//	pDoc->AnimF->Animation.bTimeNotSet = FALSE;
}

void CDecompressorView::OnUpdateAnimationStop(CCmdUI* pCmdUI)
{
	CDecompressorDoc* pDoc = GetDocument();
	ASSERT_VALID(pDoc);
//	pCmdUI->Enable(pDoc->AnimF->bAnimationInProgress);
	pCmdUI->Enable(!bStopped		&&		bIsPlayAble		&&
!pDoc->AnimF->bAbortAnimation);
}

void CDecompressorView::OnViewDiagnostics()
{
	if (bLogActive)
	{
		LogWindow.ShowWindow(SW_HIDE);
		bLogActive = FALSE;
	}
	else
	{
		LogWindow.ShowWindow(SW_SHOW);
		bLogActive = TRUE;
```

```
        }
}

CDecompressorView::ReBindTexture(CTexture *tex)
{
        // This function constructs the OpenGL bindings for the generation
        // of a mipmapped texturemap.
        if (tex->iTextureBindNum)
                ::glDeleteTextures(1, &(tex->iTextureBindNum));

        unsigned int val = MINTEXTURERES;
        for (int i = MINTEXTUREPOWER;i<tex->iCurrentMipLevel-1;i++) val*=2;

        ::glGenTextures(1, &(tex->iTextureBindNum));
        ::glBindTexture(GL_TEXTURE_2D, tex->iTextureBindNum);

        ::glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
        ::glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
        ::glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
        ::glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                        GL_LINEAR_MIPMAP_LINEAR);
        for (i=tex->iCurrentMipLevel-1;val>0;val>>=1,i--)
                ::glTexImage2D(GL_TEXTURE_2D,  tex->iCurrentMipLevel-1-i,  GL_RGB,
val, val,
                        0, GL_RGB, GL_UNSIGNED_BYTE, tex->MipMaps[i]);
    ::glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);

        tex->bTextureUnbound = FALSE;
}

void CDecompressorView::OnAnimationDownloadspeed()
{
        CDecompressorDoc* pDoc = GetDocument();
        CGetDownloadSpeed dlg;
        dlg.iSpeedSelector = pDoc->AnimF->iSpeedType;
        if (dlg.DoModal() == IDOK)
                pDoc->AnimF->iSpeedType = dlg.iSpeedSelector;
}
```