This is to certify that the
dissertation entitled

OCTREE-BASED ANIMATED GEOMETRY COMPRESSION

presented by

Jinghua Zhang

has been accepted towards fulfillment
of the requirements for the

Ph.D. degree in Computer Science and Engineering

Major Professor's Signature

Feb 17 2005
Date

OCTREE –BASED ANIMATED GEOMETRY COMPRESSION

By

Jinghua Zhang

A DISSERTATION

Submitted to
MICHIGAN STATE UNIVERSITY
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Computer Science and Engineering

2005

ABSTRACT

OCTREE-BASED ANIMATED GEOMETRY COMPRESSION

By

Jinghua Zhang

Geometry compression is the compression of the 3D geometric data that provides a computer graphics system with the scene description necessary to render images. Geometric data is quite large and, therefore, needs effective compression methods to decrease the transmission and storage requirements. A large amount of research has been focused on static geometry compression, but only limited research has addressed *animated geometry compression*, the compression of temporal sequences of geometric data. The goal of this thesis is to represent 3D animation sequences with a reduced set of motion vectors that take advantage of the large data coherence in space and time. An octree-based motion representation method is proposed. In this approach, a small set of motion vectors are generated for each frame by accessing two consecutive frames at a time. These motion vectors represent the differential motion from the previous frame to the current frame. They are used to predict the vertex positions for each frame. The octree motion representation process generates a hierarchical octree structure for each frame in the sequence. Based on this approach, a hybrid coding method is proposed that combines the octree-based approach and delta coding method. Given the same threshold requirement, the hybrid approach performs better than the octree only approach in terms of compression ratio. Both the octree and hybrid approaches can represent 3D

animated sequences with high compression factors while maintaining reasonable quality. These two approaches are easy to implement and have a low cost encoding process and a fast decoding process, which make them very suitable for real time application.

DEDICATION

To my parents, my husband (Jinsheng), my son (Haiming) and my sister (Songhua).

ACKNOWLEDGMENTS

I would like to express my appreciation to my adviser, Dr. Owen for his generous guidance and encouragement during my Ph.D. study. I also thank him for valuable suggestions and comments on the thesis. I also would like to thank my other committee members, Dr. Ofria, Dr. Hughes and Dr. Stapleton for their support and comments on the thesis.

I would like to thank my entire family members for their support and love.

I want to thank Andrew Glassner for providing access to the Chicken data and MIT CSAIL Graphics Lab for the Dance sequence used in this thesis. I also want to thank Professor Craig Gotsman for providing the TG coder.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

1 Introduction

3D graphics are rapidly achieving mainstream success, both in motion pictures and computer gaming. The confluence of fast real-time hardware and advanced modeling and rendering capabilities is opening the door to a world where the animated films of the future will be rendered and viewed on the desktop with varying viewpoints, high resolutions, and with output devices ranging from cell-phones to theatre-sized projection. But, as these models grow more complex, more detailed, and longer format, it becomes increasingly difficult to efficiently store and transmit the huge 3D models that form the basic input for these future rendering systems. Geometric data is quite large and, therefore, needs effective compression methods to decrease the transmission and storage requirements.

The focus of this thesis is the design of efficient and powerful algorithms for compressing and transmitting animated geometric data. Geometry compression is the compression of the 3D geometric data that provides a computer graphics system with the scene description necessary to render images. Animated 3D models require even larger memory and transmission bandwidth since a sequence consists of a large number of frames and every frame is an already large static 3D object. Therefore, efficient compression techniques need to be applied to the 3D animation sequence before distributing it over the network.

Animated geometry compression is the compression of temporal sequences of geometric data. Compressed animated geometry models make possible the incorporation of rich animations in the interactive applications of the future. As a simple example, an efficient and powerful mechanism for animated geometry compression allows for the delivery of an animated motion picture such as *Shrek* as a 3D model using the least possible bandwidth in real time on demand or as a multicast. Rather than rendering the film to video frames in advance, the rendering can be done at the delivery point. Symmetric Multi-Rendering with Dynamic Load Balancing mechanism proposed by nVidia may accelerate rendering process dramatically [1]. Delivering content as a 3D model rather than as video frames opens up a new realm of applications. A user can then view the film from different and arbitrary viewpoints, in stereo, or in an immersive Virtual Reality environment such as *Improvisational Theater Space*. The Improvisational Theater Space is an interactive theatre where the users are put into the characters and perform together with virtual actors [2]. To achieve this data delivery in real time, a method that can achieve high compression ratios while maintaining good quality is required.

Animation systems typically represent animation with a relatively small parameter space. The motion of a limb may be represented by only two discrete motion points modeled on a Bezier path, with the vertex motion being interpolated using algorithms sensitive to the skeletal properties of the animated character. Hence, animation should lend itself to high levels of compression since this underlying parameterization implies a great deal of correlation among vertex data. Indeed, if the underlying parameterizations were available, an encoder could likely create highly

efficient representations. However, this data is generally not available for distribution and likely to remain unavailable to rendering systems for several reasons. Often the animation systems incorporate modeling techniques that are proprietary in nature, using algorithms subject to intellectual property protection. The mechanisms necessary to conversion of parameterized graphical models to animated meshes are often complex, particularly in relation to physical modeling systems such as water, fire, or explosions [3, 4] and online rendering is neither practical or economical. And, distribution of structural model data and associated animation parameterization greatly simplifies unauthorized manipulation of the models, aiding greatly those who would steal the complex model designs in these systems for unauthorized derivative works.

Indeed it is likely that distribution will be limited to simple polygon or triangle meshes, rather than complete hierarchical scene graphs so as to limit user modifications of sequences. Hence, this thesis assumes the mostly simplistic representation of the geometric data: a temporal sequence of polygonal meshes for each discrete frame of the animated sequence.

1.1 Background

The basis for this thesis is the transmission of animated geometric data. Clearly, the genesis for this work is the efficient compression of static data. The general field of geometry compression is concerned with efficient compression of 3D data, be it animated or static. Most initial work has been focused on static geometry. This section provides a short overview of the current state of the art in geometry compression. More detail on compression methods is provided in later chapters.

Deering first proposed geometry compression techniques in 1995 with compression ratios of 6-10 to 1 [5]. Since that time, various strategies for geometry compression have been reported [6-9]. *Progressive compression* provides for partial data transmission at reduced resolution and is particularly useful when varying levels of detail are required. Progressive compression techniques have been discussed since Hoppe's progressive meshes [10-18]. Boosen provides a comprehensive overview of the reported geometry compression techniques [19]. Another geometry compression survey paper was written by Shikhare [20]. Luebke wrote a survey on polygonal simplification algorithms [21]. All these techniques focus on static geometry. Less attention has been paid to animated geometry compression [22-26].

Lengyel presented a prediction compression method that splits vertices into sets and uses affine transformations to approximate the vertex paths [22]. His method is effective only when the animation is well represented by the supported transformations. Alexa and Müller proposed an interpolation predictor to represent animations by principal components [23]. *Principal Component Analysis* (PCA) makes their approach expensive. It requires significant memory usage and processing time. The PCA approach cannot achieve high compression ratios if the number of frames is significantly smaller than the number of vertices. Ibarria and Rossignac proposed a time-space predictor for all the vertices and all the frames [24]. Briceño et al. proposed an algorithm to generate geometry video from 3D animated meshes [25]. They extended the geometry images proposed by Gu et al.[27]. To construct a geometry video, all the frames in the animation sequence are replaced by the corresponding geometry images. Conventional video compression techniques are applied to encode geometry video.

The TG coder proposed by Touma and Gotsman can compress a typical input to approximately 9 bits per vertex [8]. In the raw data, each vertex has 3 coordinates and each coordinate needs 32 bit floating point number to represent it, so the typical compression ratio is about 10 to 1. If this technique is used to compress an animation sequence, the compression ratio it can achieve is still about 10 to 1. The size of an animation sequence depends on the number of vertices in each frame and the number of frames in the sequence. The raw data size of an animation sequence is numframes * numvertices * 96 bits. The PCA approach can achieve 39.8 to 1 with reasonable quality, but it has too many limitations. Therefore a better approach is needed to compress animated geometry.

1.2 Thesis statement

The goal of this thesis is to design new compression algorithms for animated geometry that improve on existing methods, both in compression ratios and quality. The methods presented herein represent 3D animation sequences with a reduced set of motion vectors that take advantage of the large vertex data coherence in space and time.

This thesis focuses on the compression of vertex positions for each frame in the animation sequence and the connectivity is assumed to be the same for all the frames. An octree-based motion representation method is proposed that hierarchically represents 3D animation [28]. This method only needs to access two consecutive frames at a time to generate differential motion between frames. It controls the quality of every frame in the reconstructed sequence. It achieves high compression ratios with reasonable quality. This approach is easy to implement and has low cost features. An additional delta coding

method is also presented that is effective for selected data sequences, but is not generally the best solution. To further increase the performance of the octree-based motion representation method, a hybrid coding method is proposed that combines the octree-based and delta coding approaches [29]. The hybrid coding method achieves higher compression ratios given similar quality requirements.

1.3 Major contributions

The contribution of this thesis is listed as follows:

- A novel new octree-based approach is proposed and implemented that efficiently represents the data coherence in a 3D animation sequence.
- A novel new delta coding mechanism is proposed and implemented that represents the data coherence in a 3D animation sequence and is efficient as a compression mechanism on some pair-wise frame sequences.
- A hybrid coding approach is proposed and evaluated that combines the octree-based approach and delta approach to improve the performance over the octree only approach.
- All the systems are evaluated in terms of compression ratio and L^2 norm.
- The octree-based approach is compared to a pure static compression mechanism and the existing PCA animated geometry approach. The systems are compared in terms of L^2 norm.
- The delta approach is compared to the octree-based approach in terms of the overall performance and the frame-wise performance.

- The hybrid approach is compared to the octree-based approach and the delta approach in terms of compression ratio.
- A systems-layer is proposed for animated geometry compression that supports practical application of these methods in distribution systems including stream joining, random access in sequences, merging with additional multimedia streams and playback controls.

1.4 Thesis structures

This thesis is organized as follows. Chapter 2 reviews previous work on text and media compression, providing the basic context for all compression methods. In particular the relation of the mature field of video compression to animated geometry compression is examined. Chapter 3 reviews existing work on static and animated geometry compression. Chapter 4 presents a new representation of 3D animation using an octree-based motion representation method. The chapter includes evaluation results between the octree-based approach and PCA approach. Brief comparisons with other approaches are also included. Chapter 5 describes a delta coding approach to represent animated geometric data. It presents evaluation results between the octree-based approach and the delta coding approach in terms of compression ratio. Chapter 6 presents a hybrid coding approach that combines the octree-based approach and delta coding approach. By selectively incorporating these two mechanisms, the method is able to achieve better performance than any of these methods individually. Evaluation results for the mechanism are presented in detail. Chapter 7 describes the system implementation of a frame-based geometry system with particular emphasis on support for practical

playback requirements including stream joining, playback controls, and error correction. The details of such a system are contrasted to the mature MPEG systems layer designs. Chapter 8 includes the conclusions of this thesis and describes possible extensions of the current work.

Chapter 2

2 Data compression techniques

Data compression is the conversion of an input data stream into another data stream that has smaller size [30]. Data compression is of great importance because it allows users to save storage space and to transmit data efficiently. The use of data compression makes it possible to store more data on a disk and transmit more data in an available bandwidth. This chapter illustrates some basic compression concepts by presenting how they are utilized in text and media data compression. Many elements of these techniques form the basic tools for the implementation of 3D geometry compression.

Compression techniques can be classified in many ways [31]. One such partitioning is to subdivide the various methods into lossless techniques and lossy techniques. *Lossless compression* refers to data compression methods where the data can be completely recovered after decompression and the recovered data is bit-wise identical to the original data. *Lossy compression* refers to data compression methods where the data cannot be completely recovered after decompression and the recovered data is not identical to the original data because some information has been lost. Typically, the information that is lost is chosen so as to be imperceptible. The advantage of lossy compression is that it typically achieves much greater compression ratios than the best lossless methods. Lossy techniques take advantage of the fact that, in many applications, exact data representation may not be necessary. If the data after compression is

perceptually indistinguishable from the original in the application context, it does not matter if the data is bit-wise identical. This is particularly the case for media data such as images and audio. Consequently, lossy methods are especially useful in compressing images, movies or sounds but rarely suitable for text compression.

This chapter is organized as follows: Section 2.1 reviews text compression techniques. The concept of entropy and entropy coding methods such as Huffman coding and arithmetic doing are introduced. Section 2.2 reviews media compression techniques. Transform coding methods are illustrated by presenting how they are used in media data compression, such as Fourier transform and wavelet transform. JPEG and MPEG are also included in this section as an illustration of how these methods are applied in static image compression and moving sequence compression. The goal of this chapter is to introduce tools and concepts that will be further developed in the context of geometry compression.

2.1 Text compression and Entropy

Methods that represent text using fewer bits or bytes are referred to as *text compression* [32]. Text data is highly sensitive to bit-wise modifications; even a single bit change will change a letter in context and modify the meaning. Hence, text compression methods must be lossless, i.e. able to reconstruct the original text exactly. Most modern data compression algorithms break the compression process into two parts, a predictive modeler and an entropy encoder [32]. The predictive modeler assigns probabilities to symbols by analyzing the pattern of usage in the input data and the encoder translates the symbols into a sequence of bits representing a compressed version of the original message. These two processes are almost completely independent. The encoder does not

need to know the prediction process, so different prediction models can be applied in various applications. The encoder and modeler development have proceeded independently.

As an example, assume a text data set consisting only of the symbols a, b, and c. A predictive modeler will decide the symbols actually subject to encoding and their probabilities in the data stream. In this case, assume that the letter a has probability 0.5 and b and c have probabilities 0.25. This is the determination a predictive modeler seeks to make. This determination might have been made by analyzing the input data or known in advance by analysis of a standard corpus of source data with similar distributions.

In this example, an encoder may choose to assign the bit sequence 0 to represent a, 01 to represent b, and 00 to represent c. This assignment takes advantage of the fact that the symbol a is more frequent and, therefore, should be represented with fewer bits.

Static modeling assumes that the encoder and decoder agree in advance on a fixed model. In the previous example, drawing probabilities from a standard corpus would be an example of static modeling. In *adaptive modeling*, the code used for particular data is dependent on the data actually being transmitted [33]. In the above example, drawing the probabilities from the input data would be an example of adaptive modeling. It is common that adaptive modeling will all continuously adjust the symbol probabilities as the data is streamed, thereby adapting to changing characteristics of the input streams.

Numerous methods exist for predictive modeling of text data [31]. Descriptions of these methods are beyond the scope of this thesis. However, it is illustrative to discuss the concept of symbol assignment in text compression. There is no reason that a symbol must be a single character. Indeed, methods for text compression have been proposed

based on single character symbols, bi-grams, tri-grams, and complete word units as symbols. An important element in the design of any predictive modeler is the choice of symbols.

2.1.1 Entropy

Entropy is one of the most important terms in data compression. It is a measure of the actual information content of data [32]. It plays a central role in information theory as a measure of information, choice and uncertainty. If there is considerable redundancy in a data set then the value of entropy is small because redundancy implies copies of data, not new data. If a data set is highly predictable, the entropy is low, since predictability implies a lesser amount of actual information content.

As an example, the entropy of a stream of heads/tails selections of a coin, where the coin has equal probability of each result, will be n, where n is the number of coin tosses. The events are independent, so each new coin toss cannot be predicted; each new coin toss represents a new datum. However, if the coin has a 90% chance of coming up heads, the entropy will be much less, since each coin toss can be predicted with a high degree of certainty. The only real data in this case are the number of tosses and the cases where the coin does not come up heads. The entropy of this stream turns out to be 0.47n bits, as will be demonstrated shortly.

Entropy is intimately related to data compression. In an ideal (theoretically optimum) case, the encoded message length would be equal to its entropy.

Suppose that there is a set of n possible events with known probabilities $p_1, p_2 \ldots p_n$ that sum to 1. The entropy of this set measures how much choice is involved in the selection of the event and how uncertain we are of the outcome.

Shannon postulated that the entropy $E(p_1, p_2 ... p_n)$ should have the following required properties [34]:

1) E should be continuous in p_i;
2) If each event is equally likely, E should be a steadily increasing function of n.
3) If a choice is broken down into successive choices, the original E should be the weighted sum of individual values of E.

Shannon demonstrated that the only function E that satisfies the above three requirements is in the form of Equation 2.1, the equation for entropy.

$$E(p_1, p_2 ... p_n) = -k \sum_{i=1}^{n} p_i \log p_i \tag{2.1}$$

In Equation 2.1, k is a positive constant and governs the unit in which entropy is measured. In computer applications this unit is typically the "bit", where k=1 and logs are taken with base 2. In this case, the entropy function is in the form of Equation 2.2.

$$E(p_1, p_2 ... p_n) = -\sum_{i=1}^{n} p_i \log_2 p_i \tag{2.2}$$

The minus sign guarantees the result for entropy to be a positive value. This means that the message with greater probability contains less "information". Therefore the overall entropy is the average of the entropy of the individual decisions involved.

In the previous coin toss example, if the coin has a 90% chance of coming up heads and 10% chance of coming up tails, the entropy of each coin toss is shown in Equation 2.3 after applying Equation 2.2. The minimum number of bits needed to present

the toss result of a biased coin turns out to be 0.47 bits. The entropy of the coin toss stream turns out to be 0.47n bits, where n is the number of tosses.

$$-0.1 * \log_2 0.1 - 0.9 * \log_2 0.9 \tag{2.3}$$

2.1.2 Huffman coding

D.A. Huffman discovered a way to achieve compression by constructing codes based on a set of message probabilities. This is the well-known coding method called *Huffman coding*. It is a variable length coding method. The basic idea is that shorter codes are used to encode symbols that occur frequently and longer codes are used to encode symbols that seldom occur.

Huffman's algorithm takes a list of probabilities and constructs a full binary tree whose leaves are labeled with the probabilities associated with the source messages. Initially there is a forest of single node trees, one for each probability in the list. In each step, locate two trees corresponding to the smallest probabilities, p_i and p_j. They are selected to construct a new tree with a combined probability of $p_i + p_j$ with the previous trees becoming subtrees of the new tree. The probabilities p_i and p_j are replaced by $p_i + p_j$ for the new tree. This process is repeated until only one value exists in the list. The final tree is the Huffman coding tree. This is best illustrated by an example. Given five symbols with probabilities as shown in Table 2.1, the Huffman tree is shown in Figure 2.1 and the final Huffman code is shown in Table 2.2. Therefore CAD can be encoded using 9 bits (1011 100 11).

Table 2.1: Probability of each symbol in the message.

Symbols	Probabilities (p)
A	0.13
B	0.40
C	0.10
D	0.29
E	0.08
Total	1.00

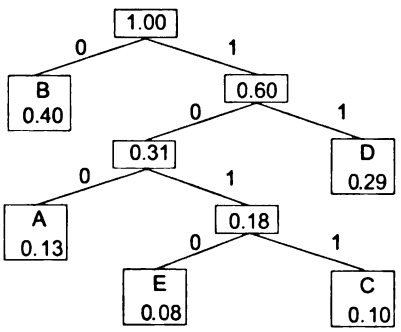


Figure 2.1: Huffman tree.

Table 2.2: Huffman codes.

Symbols	Binary Codes
A	100
B	0
C	1011
D	11
E	1010

Huffman coding provides the optimal solution if the optimum number of bits per symbol is integral. In this case, all symbol probabilities are exact powers of $\frac{1}{2}$. But this is seldom practical. To make Huffman coding more effective, symbols can be blocked into

n-grams and these n-grams can be used as the coding units. This is the shortcoming of Huffman coding.

2.1.3 Arithmetic Coding

Arithmetic coding was invented by Rissanen [35]. It is also an entropy-based lossless coding method. However, it is much more flexible than Huffman coding. Arithmetic coding guarantees that any message can be encoded in the number of bits dictated by its entropy. Arithmetic coding encodes symbols using non-integral numbers of bits. This coding method is computationally efficient and computes the code incrementally, one symbol at a time. In arithmetic coding, a message is represented by an interval for real numbers between 0 and 1. The longer the message, the more precision is required for the real numbers. It means the number of bits needed to specify the range increases. Before anything is processed, the range for the message is [0,1). As each symbol is processed, the range is narrowed to the portion allocated to the symbol. The arithmetic coding process is illustrated in Figure 2.2. The symbol probabilities are shown in Table 2.1. Any value in the final range can be used as an encoding of CAD.

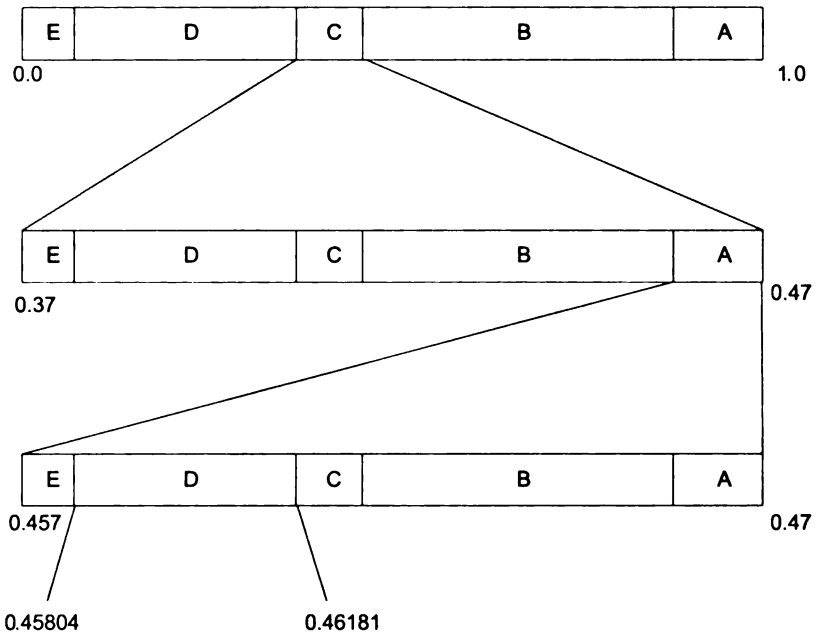


Figure 2.2: Arithmetic coding process.

2.2 Media data compression

Media data often exhibits high degrees of redundancy. As an example, the value of a pixel in an image can be predicted with high probability based on the value of neighboring pixels. This predictability often extends of large areas of media data. Hence, simple first order predictions or models are often insufficient. What is desired is a way to decrease the redundancy in the media data as a unit rather than attempt to predict this redundancy.

Transform coding methods convert source data to an alternative representation that decorrelates the data, thereby reducing the redundancy. Fourier Transforms, Wavelet Transforms, Vector Quantization and Fractal are important transform techniques used in media data compression. The main reason for using transform-coding methods on media data is to take advantage of correlations among media data. For example, image compression can be achieved by transforming its pixels to a decorrelated representation.

An original image is represented as pixels in a spatial domain. If the image is subject to a Fourier transform, the data is transformed to the Fourier domain. This domain exhibits information packing, meaning the information in the image is packed into fewer actual data points, each of which is orthogonal to all other data points. JPEG and MPEG are discussed in this section as an illustration of how these methods are applied in static image compression and moving sequence compression.

2.2.1 Fourier Transform (FT)

The *Fourier transform* is used to transform a continuous time signal into the frequency domain [36, 37]. Joseph Fourier showed that any function could be expressed as an infinite sum of sinusoids of different frequencies. Sine waves of different frequencies are orthogonal. The transformed signal contains exactly the same information as that of the original function and they differ only in the information presentation [38]. The Fourier transform is an invertible transform. The transformed frequency domain signal can be converted back to the original time domain signal and vice versa.

Continuous Fourier transform

For a continuous function, the Fourier transform pair is given by the following two formulas. Equation 2.4 is the forward transform. Equation 2.5 is the reverse transform.

$$G(f) = \int_{-\infty}^{+\infty} g(t) e^{-j2\pi f t} dt \tag{2.4}$$

$$g(t) = \int_{-\infty}^{+\infty} G(f) e^{j2\pi f t} df \tag{2.5}$$

In the above equations, j is the square root of -1. $g(t)$ is a continuous function in the time domain and $G(f)$ is the corresponding FT of the function in the frequency domain.

Discrete Fourier transform

In the case of sampled discrete functions that are represented by values at equally spaced points, the discrete Fourier transform is utilized. It is an invertible linear transformation and widely used for the analysis and design of signals and systems.

Although the Fourier transform has been widely used in many areas, it has its limitations. This transform is not a suitable technique for non-stationary signals. A *Non-stationary signal* is a signal that has different characteristics at different times.

2.2.2 Wavelets

Wavelets are functions that meet certain mathematical requirements and are utilized to represent other functions and data [39]. The main idea of wavelet transform is to select a *mother wavelet*, a wavelet prototype function, and use it to explore the properties of a nonzero function in a small interval. The mother wavelet is then translated to another interval of t and used in the same way. Different frequency resolutions are explored by scaling the mother wavelet with a scale factor. The original signal can then be represented using coefficients in a linear combination of the wavelet functions. If some coefficients are ignored based on a predefined threshold, the original data is represented by a small number of coefficients. This makes wavelets an excellent tool in the field of data compression.

Wavelet transforms are often classified into two categories, namely the continuous wavelet transform (CWT) and the discrete wavelet transform (DWT). The continuous transform is defined as a continuous, multi-resolution transform built up from a mother wavelet. It decomposes a function into a set of basis functions. More detailed information about CWT can be found in the book written by Chui [40]. The discrete wavelet transform is applied to discrete data sets and generates discrete outputs. DWT is more commonly used in practice than the CWT because the information being transformed is usually discrete. The discrete wavelet transform provides sufficient information both for analysis and synthesis of the original signal. The signal is passed through a series of high pass filters to analyze the high frequencies and a series of low pass filters to analyze the low frequencies [41].

Wavelet transforms are especially useful for compressing image data. An image can be represented as sets of real coefficients after applying the wavelet transform. Most of the wavelet coefficients of a typical image are close to zero, so the image thus is well approximated with a small number of large wavelet coefficients. Wavelet compression is achieved by quantizing and encoding wavelet coefficients.

The Haar transform is an example of the simplest wavelet transform. An example of applying Haar wavelet transform to a gray image will be demonstrated shortly. The Haar wavelet transform uses a scale function and a wavelet to represent a large number of functions. Calculating averages and differences is the principle of the Haar transform. Each step of Haar transform computes a set of wavelet coefficients (differences) and a set of averages. For a n-element data set, there will be $\frac{n}{2}$ averages and $\frac{n}{2}$ wavelet coefficient values after the first step. The averages will be used as inputs for the next wavelet

calculation step until a single average and a single wavelet coefficient are computed. This replaces the original data set with an average followed by a set of differences.

It is easy to explain the Haar transform using matrix multiplication. Suppose there is a simple grayscale 8 by 8 image and the values of the first row pixels are (8, 7, 6, 5, 4, 3, 2, 1). Matrix A1, A2 and A3 as shown in Equation 2.6 are used in the first, second and third steps of the Haar transform respectively. For this data set it only needs 3 steps to get Haar wavelet transform of the original data items. The result is shown in Equation 2.7.

$$A1=\begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & -\frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & -\frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & -\frac{1}{2} \end{pmatrix} A2=\begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ \frac{1}{2} & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & -\frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} A3=\begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad (2.6)$$

$$A3*A2*A1*\begin{pmatrix} 8 \\ 7 \\ 6 \\ 5 \\ 4 \\ 3 \\ 2 \\ 1 \end{pmatrix}=\begin{pmatrix} 4.5 \\ 2 \\ 1 \\ 1 \\ 0.5 \\ 0.5 \\ 0.5 \\ 0.5 \end{pmatrix} \quad (2.7)$$

The above example shows that the Haar transform can be achieved by building matrices and multiplying them to get a result matrix W. The matrix W is used to convert

the original data set to a set of wavelet coefficients. To compute the Haar transform of a complete two-dimensional image, the standard algorithm starts by applying W to all the rows of the original image and then it applies W to all the columns of the transformed image. After applying the Harr wavelet transform, the pixel values of final image should be smaller than the corresponding values in the original image. The final values can be compressed using some basic coding methods. Lossy wavelet image compression can be achieved by discarding some small coefficients.

2.2.3 Vector Quantization

Due to the fact that adjacent data items in an image or digitized voice are often highly correlated, *vector quantization* (VQ) is commonly used to compress images and digitized sound [42]. It is a lossy data compression method. A vector quantizer is an approximator. As a simple illustration, a one-dimensional two-bit VQ is shown in Figure 2.3 and a formalized representation of this quantizer in Equation 2.8.

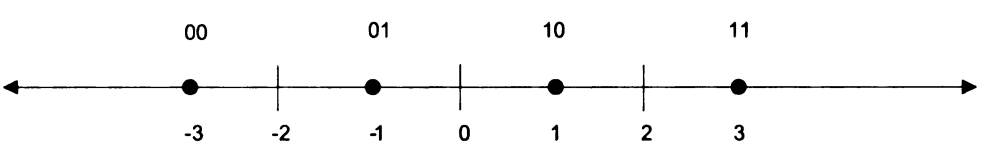


Figure 2.3: One-dimensional VQ.

$$VQ(x)=\begin{cases} -3 & x \le -2 \\ -1 & -2 \le x \le 0 \\ 1 & 0 \le x \le 2 \\ 3 & x \ge 2 \end{cases} \tag{2.8}$$

A two-dimensional VQ is illustrated in Figure 2.4. A two-dimensional quantizer assigns any input point (a pair of number) in the plane to one of a particular set of 4 black

dots in the plane. In this example, there are 4 regions and 4 black dots. This is a two-dimensional, two-bit VQ. In these two examples, the black dots are called code vectors and the region associated with each dot is called an encoding region. The codebook consists of all the code vectors and all encoding regions form a space partition.

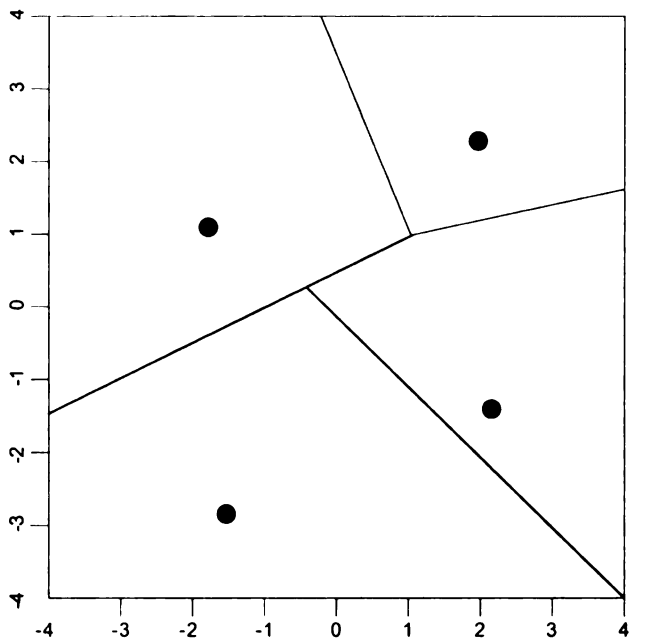


Figure 2.4: Two-dimensional VQ.

When the VQ method is applied to image compression, the image is first divided into small blocks of pixels, typically 2x2 or 4x4. The compression is achieved by compressing the pixel blocks instead of individual pixels. The encoder maintains a codebook. For each image block, the encoder outputs a pointer that points to the corresponding entry in the codebook and writes it on the compressed stream. Linde et al. proposed an efficient algorithm (LBG) for designing an optimal vector quantizer for a

given signal source. The LBG algorithm is the basis of many vector quantization methods used in image and voice compression [43].

Fractal compression is a form of vector quantization that utilizes a virtual codebook [44]. Compression is achieved by locating self-similar sections of an image, then using a fractal algorithm to generate the sections. This technique generates lossy compression. Compression is slow but decompression is fast. Fractal techniques are widely used. They are used in generating terrain world, image compression and so on.

2.2.4 JPEG

JPEG stands for Joint Photographic Experts Group and is a standardized image compression mechanism. It is designed for compressing continuous-tone still images [45]. JPEG utilizes the fact that the human eyes perceive small chrominance changes less accurately than small luminance changes. Thus, JPEG compressed images are suitable to be looked at by humans but not for machine image analysis. JPEG provides several operation modes so the user can choose different modes to compress the image according to different needs. JPEG is a lossy compression mechanism. JPEG allows the user to achieve a desired compression/quality tradeoff by adjusting compression parameters. Storing full color information (24 bits/pixel) is also an advantage of JPEG.

Discrete Cosine Transform (DCT) is a variant of the discrete Fourier transform. It can transform an image from the spatial domain to the frequency domain. Given a n by n image, the general equation for its two dimensional DCT is defined by the Equation 2.9. JPEG standard uses n equal to 8.

$$D(i,j) = \frac{1}{\sqrt{2n}} C(i)C(j) \sum_{x=0}^{n-1} \sum_{y=0}^{n-1} p(x,y) \cos\left(\frac{\pi(2x+1)i}{2n}\right) \cos\left(\frac{\pi(2y+1)j}{2n}\right) \qquad (2.9)$$

where $C(i) = \begin{cases} \frac{1}{\sqrt{2}} & for\ i = 1 \\ 1 & otherwise \end{cases}$ and $0 \le i, j \le n-1$.

In Equation 2.9, $p(x, y)$ is the intensity of the pixel in row x and column y; $D(i, j)$ is the DCT coefficient in row i and column j of the DCT matrix. For most images, much of the image information lies at low frequencies. The upper left values of the DCT matrix represent lower frequencies. The higher frequencies are represented by the lower right values. The lower right values are often small, so the DCT makes it possible to achieve image compression by neglecting those values with little visible distortion.

Figure 2.5 shows the JPEG compression diagram. The color images are transformed from RGB color space to luminance/chrominance color space and organized in groups of 8 x 8 pixel blocks. The JPEG standard applies the two dimensional DCT to each 8 x 8 pixel block to generate an 8 x 8 coefficient matrix for the corresponding frequency components. Each DCT coefficient is divided by its quantization coefficient (QC) and rounded to an integer. The 64 quantized DCT coefficients of each pixel block are encoded using entropy encoder to generate the compressed bit stream. JPEG is a symmetric compression method because the decoder performs the reverse steps.

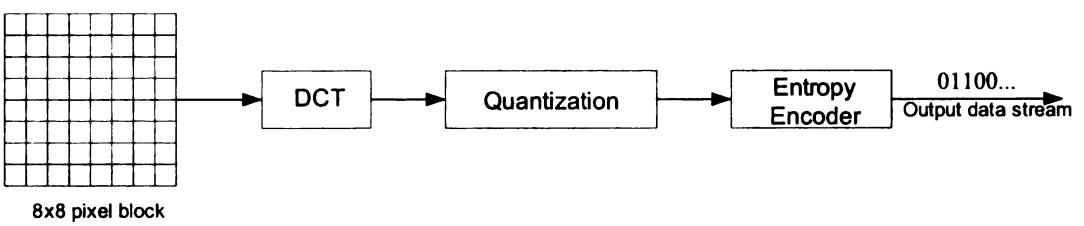


Figure 2.5: JPEG compression diagram.

2.2.5 MPEG

MPEG stands for Moving Pictures Experts Group. The MPEG group has devised numerous standards for compression and representations of digital video and multimedia data. The MPEG standards for compressed video data include MPEG-1, MPEG-2 and MPEG-4 [46]. MPEG-1 is intended for intermediate data rates, on the order of 1.5 M bits/s. MPEG-2 is intended for high data rates of at least 10M bits/s. MPEG-4 is intended for very low data rates of less than 64 K bits/s.

MPEG represents video sequence using three types of frames: I, P and B-frames. I stands for intra-coded frames. I-frames are coded independently. I-frames are basically encoded using JPEG compression algorithm. A P-frame is inter-coded using its predecessor. A P-frame is a differential representation of a frame based on the previously reconstructed I or P-frame as shown in Figure 2.6. A B-frame is inter-coded based on both past and future I-frame and P-frame as shown in Figure 2.7. MPEG utilizes differentially encoded frames to take advantages of interframe correlation of digital video.

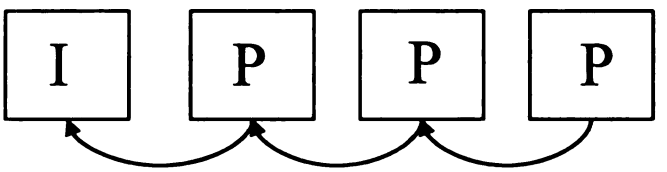


Figure 2.6: Illustration of an inter-coded P frame.

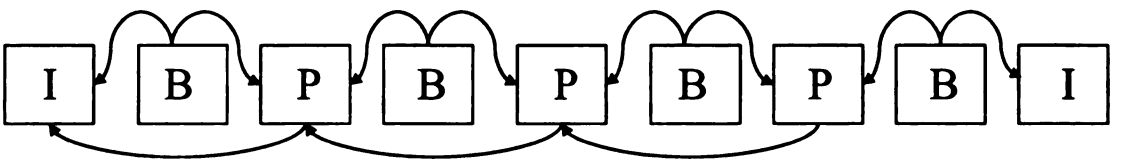


Figure 2.7: Illustration of an inter-coded B frame.

The macroblock is the basic building block of an MPEG picture. It is composed of a 16 x 16 block of luminance samples and two 8 x 8 chrominance sample blocks, therefore a macroblock has six 8 x 8 blocks of samples. To compress a macroblock, the MPEG standard applies a DCT to each 8 x 8 sample block to create decorrelated values, quantizes the DCT coefficients and encodes the quantized results. The main difference from JPEG is that MPEG uses different quantization tables, different code tables and different rounding methods for I-frames and non I-frames.

Motion Compensation is an important element in MPEG inter compression. It can only be used for non I-frame coding. In this mode, the *pels* are predicted by the previous reference frame. The smallest detail that can be reproduced in a picture is about the size of a picture element, referred to as a pel in video terminology [47]. Before applying DCT, the pels are subtracted to get the difference. The results are quantized and encoded. Motion is normally the main cause of differences between the current picture and the previous reference picture. Matching a region in the current picture with different regions in the previous reference picture can provide a template for the region that is a good prediction of the actual data values. Macroblocks are used as an elementary region in motion compensation. A P-frame uses an earlier I-frame or P-frame as a reference picture, so P-frame uses *forward prediction*. Forward prediction means that a target macroblock to be encoded is matched with a set of displaced macroblocks of the same size in a past reference picture. A B-frame may use forward or *backward prediction*, or both. Backward prediction means that a target macroblock can be predicted by a prediction macroblock from the future reference picture.

For decompression, MPEG reconstructs the pels of the entire video sequence. It reads the codes of a block from the compressed data stream, decodes them, recovers the quantized DCT coefficients and takes the inverse DCT. For P and B-frames, motion compensated prediction should be added to the result of inverse DCT. The entire video sequence is decoded picture by picture and macroblock by macroblock for each picture.

2.3 Summary

This chapter reviewed some standard compression techniques for both text and multimedia data. These methods are not directly applicable to 3D geometry. The text compression methods are ineffective because the actual symbol properties in geometric data are not well distributed. There is a much larger set of symbols and they are used with small frequency. Hence, predictive methods based on symbol probabilities are not generally effective. Pure transform techniques have been applied to geometry compression [48] and animated geometry compression [23, 26], but the methods that will be presented in this thesis are more closely related to wavelet compression methods.

The system design features of MPEG take advantage of the temporal redundancy in video sequence and are well developed in their support of playback controls and stream joining. Many of the general concepts of MPEG will be applicable to 3D geometry compression, though with considerable changes, as illustrated in Chapter 4 and Chapter 7.

Chapter 3

3 Related work on 3D compression

The standard compression techniques discussed in the previous chapter are effective for text and media compression. However, these methods are not well adapted to compression of 3D geometric data. This chapter reviews existing work on static and animated geometry compression and provides the foundation for new development in animated geometry compression. Section 3.1 reviews the related work on geometry compression and the animated geometry compression techniques are reviewed in Section 3.2.

Geometry compression can be divided into static and animated methods. Static methods encode a single geometry model for transmission and are typically applied to walkthroughs and applications where the animation is provided locally (such as some gaming applications). They also provide the intraframe coding necessary for any practical animated geometry compression solution, such as the basic concepts of JPEG compression are used to provide the intra-coded frame solution in MPEG compression.

Animated geometry compression encodes a sequence of geometry models and is suitable for delivery of fully animated content such as movies or interactive games.

3.1 Static geometry compression

This section reviews existing static geometry compression techniques. Section 3.1.2 presents different ways to represent 3D geometry. Static geometry compression

techniques can be grouped into two categories, vertex compression techniques and connectivity compression techniques. Section 3.1.3 and Section 3.1.4 review vertex compression and connectivity compression techniques respectively.

3.1.1 Introduction

With the increasing popularity of 3D graphics, large amounts of 3D data are routinely accessed over the Internet, while whole new applications for 3D graphics await more efficient ways to access the data. Due to the increasing complexity of 3D graphics, it becomes more difficult to efficiently store and transmit the massive 3D models that form the input for any rendering system. Geometric data is quite large and, therefore, it is valuable to develop effective compression methods to decrease the transmission and storage requirements.

Geometry compression has become a rapidly growing field since Deering proposed a geometry compression technique [5]. Various strategies have been developed for geometry compression since then. Geometry compression has initially focused on non-progressive techniques. Progressive geometry compression has become an active topic since Hoppe introduced progressive meshes [10]. Progressive techniques provide for partial data transmission at reduced resolution. It is good to compress 3D models in progressive fashion to meet the need of progressive transmission, display and Level of Detail (LOD) control. For progressive compression, information is compressed at different resolutions. The coarsest version of the model is encoded first and followed by finer detailed versions of the model. Most common geometry compression techniques are in a non-progressive fashion [5-9, 49-51]. Researchers have proposed progressive compression techniques [10-18, 52, 53].

View-independent methods compute LODs for each object in the scene. At run time the rendering system selects a proper LOD for each object based on the distance. As the object is further away from the viewer, the coarser LOD for this object is used. *View-dependent* approaches use a dynamic representation of the model [53-56]. It is continuously queried at run time to produce an appropriate representation of the model to the user's current viewpoint. View-dependent approaches address the problem of view-dependent LOD control. They find and update a mesh according to the viewing parameters. View-dependent methods have some advantages over view-independent approaches. An object can be rendered at multiple LODs. It is a very useful tool for real time rendering complex 3D environments. Hoppe's work represents an arbitrary triangle mesh as a hierarchy of geometrically optimized refinement transformations [54, 55]. This representation makes it easier to retrieve accurate approximating meshes based on view parameters. Increased run-time computation is one drawback of view-dependent approaches.

3.1.2 Geometric data representations

Geometric data is used to represent 3D objects in a computer. It provides foundations for computer graphics, computer-aided geometric design and visualization. 3D models can be represented in many different ways. These representations can be classified into four categories: *raw data, surfaces, solids* and *high-level structures* [57]. Raw data is unstructured data acquired directly from 3D modeling devices. It includes *point clouds, range images* and *polygon soups*. Point clouds are obtained from range scanners as a set of unstructured 3D point samples. Range images are obtained from a range scanner and are a set of points mapping to pixels as a depth image. Unstructured

sets of polygons are called polygon soups. *Mesh, subdivision surfaces, parametric surfaces* and *implicit surfaces* are in the surface category. Connected sets of polygons are called *Mesh surfaces*. The polygon mesh is the most common modeling method for 3D computer graphics. Surfaces can also be built by using coarse meshes and subdivision rules that specify how resolution is increased on the mesh. This representation is referred to as subdivision surfaces. Parametric surfaces describe surfaces using a small set of parameters such as control points. Spline and Bezier patches are examples of parametric surfaces. A surface can also be defined in an implicit way, an implicit surface, using a function to describe all the points on a surface. An equation for a sphere defines an implicit surface.

Solids categories include *Voxels* and *BSP trees*. As an example, voxels obtained from CAT or MRI are a uniform grid of volumetric samples. From polygonal representations, binary space partitions with solid cells labeled can be constructed. Scene graphs and skeletons are high-level structures to represent 3D objects. A scene graph is a hierarchical representation of a graphical scene.

Although many representations have been proposed for 3D models, the polygon mesh (and the more constrained triangle mesh) is the most prevalent representation. A polygon mesh is a very flexible and simple means of describing a 3D surface. 3D graphics adapters optimized for triangles reinforce this trend and, in fact, encourage the trend to make the polygons all triangles. Since all other representations can be converted to triangle meshes, triangle mesh representation has become a popular format to represent 3D models.

A *triangle mesh* is represented by vertex data and connectivity information. Vertex data consists of coordinates of all the vertices and (optionally) vertex normal vectors and texture coordinates. Connectivity information is simply a triangle list where each triangle is a list of vertices. Figure 3.1 illustrates a 3D model and its representation. The *PLY file format* is an example of this data structure [58]. A PLY file consists of a header followed by a list of vertices and a list of polygons.

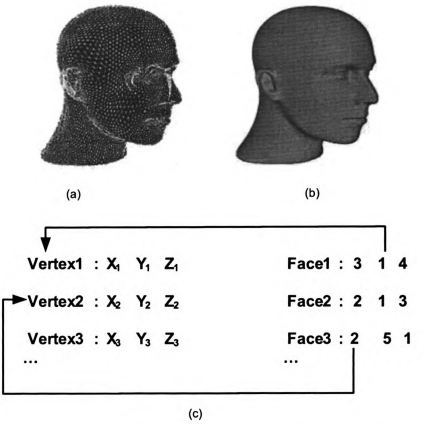


Figure 3.1: This head model has 11,703 vertices and 23,402 triangle faces. (a) Wire frames. (b) Shaded Illustration. (c) Triangle mesh representation.

3.1.3 Vertex compression techniques

Vertex data and connectivity information are the main elements of the geometry. The standard geometry representation used in modern graphics hardware is a set of

vertices containing xyz coordinates, and a set of faces containing indices referring to the vertices. Geometry compression techniques can be classified into two categories: vertex compression and connectivity compression techniques.

Vertex data consists of coordinates of all the vertices and optional normal vectors and textures. Each coordinate component of a vertex is usually a floating-point number. According to the survey by Gotsman et al. [59], there are three vertex compression techniques in the literature, namely quantization, prediction methods and spectral methods.

3.1.3.1 Quantization

Quantization is a common technique to compress numerical data by discarding excess resolution. It is a lossy compression method because the recovered data after decoding is not identical to the original data. *Uniform quantization* is a typical form of quantization, though the vector quantization methods described in Chapter 2 are all applicable. If the input range is divided into levels with equal spacing, a uniform quantizer is described; otherwise it is non-uniform quantization. They both have advantages and disadvantages. Uniform quantization is easy to implement and fast to quantize/dequantize, but the quality of reconstructed data is not optimal. A non-uniform quantizer has higher computation cost, but the reconstructed data has optimal quality.

Deering used a uniform quantizer to reduce the number of bits for each coordinate component of each vertex to 16 bits [5]. After quantization, vertex locations and colors were delta coded followed by modified Huffman compression. He used a table-based approach for normal vectors. In this method, any normal can be represented by 18-bit index and many normals can be represented with 8 bits or smaller index deltas.

3.1.3.2 Prediction methods

Some researchers have proposed a prediction idea that improved Deering's method by considering the features in the geometry. Predictive encoders work better than pure quantization methods. A predictive encoder takes the advantage of the correlation between the position of a vertex and the position of its neighboring vertices. Predictive encoders are based on vertex estimation that is derived from the incidence graph and from previously decoded vertex positions. Taubin and Rossignac used a linear predictive encoder [6]. Vertex coordinate components are first quantized to a fixed number of bits (8,10 or12 bits) by bounding the interval in which the coordinates lie. A vertex-spanning tree is used to predict the geometry of each vertex as a linear combination of the geometry of its ancestors in the tree. The geometry of a new vertex can be expressed as $\lambda_1 v_1 + \lambda_2 v_2 + \lambda_3 v_3 + e$, where λ_1, λ_2 and λ_3 are estimated by minimizing the prediction errors over the entire mesh, where v_1, v_2 and v_3 are the immediately proceeding ancestors of this new vertex in the vertex spanning tree, and where e represents the prediction error. Geometry is predictively encoded and the prediction errors are encoded with standard lossless entropy coding techniques. Other properties, such as normals, colors and texture coordinates are encoded in a similar way. This method can compress the geometry to approximately 12 bits per vertex. Touma and Gotsman proposed a more sophisticated prediction scheme to achieve a more accurate prediction [8]. The "parallelogram" rule was used to predict vertex positions. Using this rule, the geometry of a vertex is predicted from the geometry of vertices surrounding it on the mesh surface. This rule is based on the assumption that the two adjacent triangles in a smooth mesh can approximately form a parallelogram, so the geometry of fourth vertex may be predicted from the other three

vertices. The vertex d of the triangle incident on the edge bc can be predicted by $d^p = c + b - a$, where a is the third vertex of the other previously processed triangle incident upon edge bc, as shown in Figure 3.2 [59]. For typical inputs, their method can achieve approximately 9 bits per vertex.

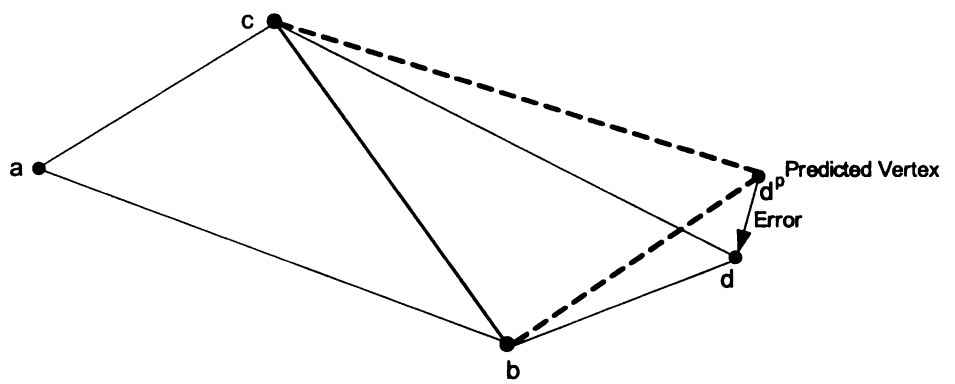


Figure 3.2: Parallelogram Rule.

Pajarola and Rossignac used a butterfly prediction method to estimate the original non-collapsed vertex positions [11]. The basic idea of their approach is to predict the position of a vertex by computing the weighted sum of surrounding vertices at topological distance 1 or 2 on the triangulation graph.

3.1.3.3 Spectral methods

Spectral methods are similar to the transform coding methods used for image and signal compression. Kami and Gotsman [48] first employed mesh spectral analysis to compress vertex coordinates of polygonal meshes. They built a set of geometric data that represents basis functions for the geometry of the entire mesh. Transform coding methods work by transforming a function (in this case the discrete vertex locations) to an

alternative domain such that the original data can be reconstructed as a weighted sum of the basis functions in the domain.

Their method computes the spectral coefficients by building a mesh Laplacian matrix from the topology information of the mesh. The Laplacian matrix is a n by n matrix, where n is the number of vertices in the mesh, and each element in the matrix is computed using Equation 3.1, where d_i is the degree of vertex i. The next step of their approach is to compute the eigenvectors of this matrix to obtain a set of basis functions. The spectrum of the geometry can be obtained from the spectral basis functions. For meshes containing more than 1000 vertices, it is not practical to compute eigenvectors. In this case, an efficient partitioning algorithm must be used to generate submeshes and mesh spectral analysis can be applied to each submesh separately [48].

$$L_{i,j} = \begin{cases} 1 & \text{if } i \text{ and } j \text{ are the same vertex} \\ \dfrac{-1}{d_i} & i \text{ and } j \text{ are neighbors} \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

3.1.4 Connectivity compression techniques

A large part of geometry compression research has been focused on connectivity information compression. Connectivity compression aims to find a way to reduce repeated references to vertices shared by many triangles. Most of these techniques compress connectivity information in a lossless way [6, 8, 9, 49]. Vertex positions can be subject to some loss, but actual geometry loss must be controlled carefully so as to create an object that is perceptually identical to the original.

Some existing connectivity compression methods assume specific structures in the geometric data such as triangle strips [5] or spanning trees [6, 49]. Rossignac's

Edgebreaker is a technique suitable for triangle *manifold meshes* [9]. In a manifold mesh, the neighborhood of each vertex can be continuously deformed to a disk and each edge is shared by no more than two triangles [60]. In their method, they used a preprocessing stage to renumber the vertices of the mesh and represent the model in a half edge data structure, then traverse triangles, recording the history in terms of 5 *op-codes*. The op-code is used to describe the topological relation between the current triangle and the boundary of the remaining part of the mesh [9]. Connectivity is encoded in 1.5 to 2 bits per triangle. Isenburg and Snoeyink extended the edgebreaker approach proposed by Rossignac to compress polygonal manifold meshes [50]. King et al. have worked on compressing irregular quadrilateral meshes [51]. Some researchers also extended earlier efforts to handle non-manifold meshes. Till now, connectivity compression techniques have been well developed. Connectivity representation and several significant previous published connectivity compression techniques are summarized in the following sections.

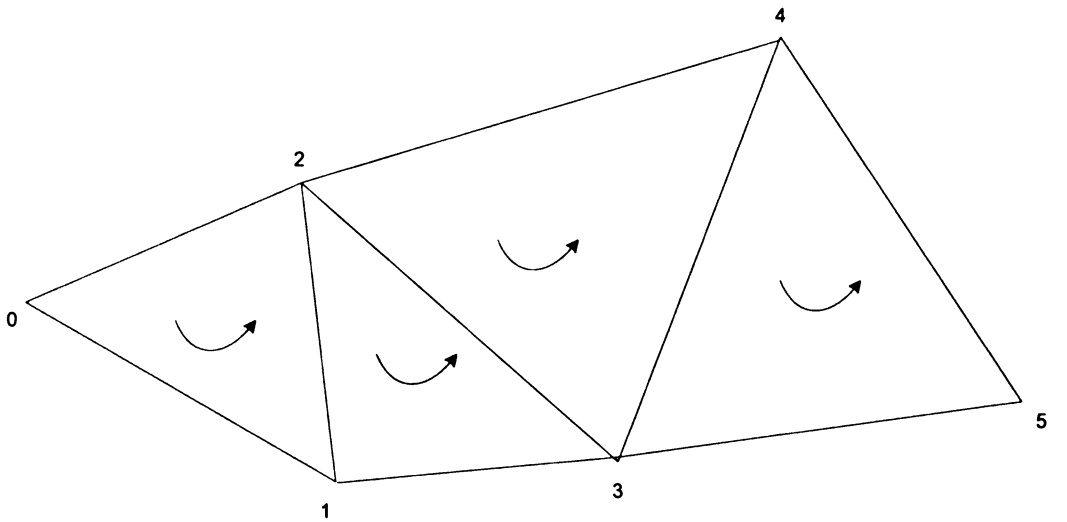
3.1.4.1 Connectivity representation

The simplest representation of each triangle is to store a list of vertex coordinates that consist of three coordinates of each vertex in the triangle. For this approach, each triangle is stored independently and the location of a vertex is repeated 6 times on average, which turns out to be very inefficient. To avoid storing the same vertex multiple times, the vertices are stored in an indexed list and the connectivity (each triangle) is stored as a list of indices into the vertex list. Each triangle can be represented using 3 integer numbers and $3\log_2(|V|)$ bits are needed for each triangle, where |V| stands for number of vertices in the model [9]. An improved method is to combine the advantages of the above two methods by storing only the triangles, each represented by 3 vertex

descriptors [9]. In this scheme, each vertex descriptor begins with one bit switch indicating a new vertex or previously seen vertex. If it is a new vertex, the switch bit is followed by the three coordinates of this vertex; else if this is a known vertex, the switch bit is followed by the $\lceil \log_2(p) \rceil$bits that identify one of the P previous seen vertices. Using this method, connectivity cost is $2.5\log_2(|V|)+0.5$ bits for each triangle [9]. Later on, Bar-Yehuda and Gotsman proposed a method to visit the triangles in an order that guarantees no more than $13(|V|)^{0.5}$ vertices are exposed at any given time [61]. Connectivity cost can be lowered down to $1.25\log_2(|V|)+9.75$ bits per triangle with this improved method.

3.1.4.2 Triangle-strip based technique

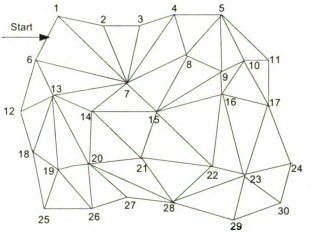
Deering presented the idea of triangle strips. A mesh representation based on this idea is supported by OpenGL [62] and other graphics libraries. This representation aims to reduce the number of times the same vertex is transferred and processed by the graphics subsystem. In a triangle strip, each triangle is represented by combining a new vertex with two previous seen vertices. To achieve this, ordering the triangles is needed to make consecutive triangles adjacent to each other. Each new triangle shares an edge with the previous triangle in the strip. Therefore a triangle strip is a series of connected triangles. The Figure 3.3 is an example of a triangle strip that consists of 4 triangles: 0 1 2; 2 1 3; 2 3 4; 4 3 5



0 1 2 3 4 5 ...

Figure 3.3: Triangle strip.

Some geometry, such as that in Figure 3.3, can be represented by one triangle strip, but geometry such as that in Figure 3.4 is not suitable to represent using a single triangle strip. The interior vertices will appear twice in the strip, which is undesirable. Deering used a generalized technique to avoid references to old data. Figure 3.4 is an illustration of generalized triangle mesh [5].



Generalized Triangle Strip:
R6, O1, O7, O2, O3, M4, M8, O5, O9, O10, M11, M17, M16, M9, O15, O8, O7, M14, O13, M6, O12, M18, M19, M20, M14, O21, O15, O22, O16, O23, O17, O24, M30, M29, M28, M22, O21, M20, M17, O26, M19, O25, O18

Generalized Triangle Mesh:
R6p, O1, O7p, O2, O3, M4, M8p, O5, O9p, O10, M11, M17p, M16p, M-3, O15p, O-5, O6, M14p, O13p, M-9, O12, M18p, M19p, M20p, M-5, O21p, O-7, O22p, O-9, O23, O-10, O-7, M30, M29, M28, M-1, O-2, M-3, M27, O26, M-4, O25, O-5

Legend:
First Letter: R=Restart, O=Replace Oldest, M= Replace Middle
Trailing "p"=push into mesh buffer
Number is vertex number,-number is mesh buffer reference where -1 is most recent pushed vertex.

Figure 3.4: Generalized triangle strip.

Deering's method is achieved by encoding geometry using generalized triangle strips and by using lossy techniques to encode vertex coordinates and other properties [5]. In the first step, the triangle data are converted into generalized triangle strips which are a near-optimal representation of triangle mesh given fixed storage. After that, quantization techniques are used to quantize vertex coordinates, color and vertex normals. The quantized results are delta encoded between neighbors and the last step is to encode the delta values using modified Huffman encoding method. Compression ratios obtained by using this technique are between 6 and 10 to 1 based on the original representation format

and the desired result quality. Deering's compressed format is designed for reducing the bandwidth requirements of the rendering hardware [5]. His work has been incorporated into the Java3d API by Sun Microsystems.

3.1.4.3 Topological surgery approach

Taubin and Rossignac's work on geometry compression through topology surgery [6] improved on Deering's [5] earlier results. Their method is not directly suitable for hardware rendering with limited on board memory. It aims to achieve high compression efficiency for transmission over networks. This approach can preserve the connectivity information. It is a single-resolution mesh compression scheme. It is derived from the body of work on encoding of planar graphs for traversal of manifold meshes in a particular fashion to obtain a vertex spanning tree. To encode the connectivity information, a binary triangle spanning tree of the mesh is constructed by cutting through the edges of the vertex spanning tree. The triangle spanning tree can be encoded in the same way as the vertex spanning tree.

3.1.4.4 Progressive approaches

This section summarizes the multi-resolution mesh compression schemes. Compressing polygonal meshes in progressive fashion can meet the need of progressive transmission, display and LOD control. It is a requirement for progressive transmission. These techniques can further improve the performance of graphics rendering hardware. The mesh with reduced resolution is used when it is far away from the camera, and the high-resolution mesh is used when it is closer to the camera. Progressive compression is used to create a compressed representation for geometry models so that the compressed

file consists of coarse version of the model followed by more detailed versions of the objects. Progressive techniques provide for partial data transmission at reduced resolution. Another advantage of this representation is that the user can view the coarse model before the entire object is decoded.

Hoppe proposed progressive meshes representation [10]. It can be used to transmit a 3D mesh progressively, starting from a coarse mesh and refining the mesh by inserting new vertices one by one. He applied a vertex insertion operation, which is the inverse of the edge collapse operation used in many mesh simplification methods [63]. Progressive mesh representation is a scheme for storing and transmitting arbitrary triangle meshes. The advantage of this representation is that it keeps the geometry of the original mesh and its overall appearance.

The first step to generate a progressive mesh is edge collapse transformations. An edge collapse transformation merges two adjacent vertices into a single vertex and two (one if boundary edge) adjacent faces disappear in the process as illustrated in Figure 3.5. The process starts from the original mesh and does a sequence of edge collapse transformations until it reaches a much coarser mesh. Because edge collapse transformations are invertible, the original mesh can be recovered from the coarser mesh using a sequence of vertex insert operations as shown in Figure 3.6.

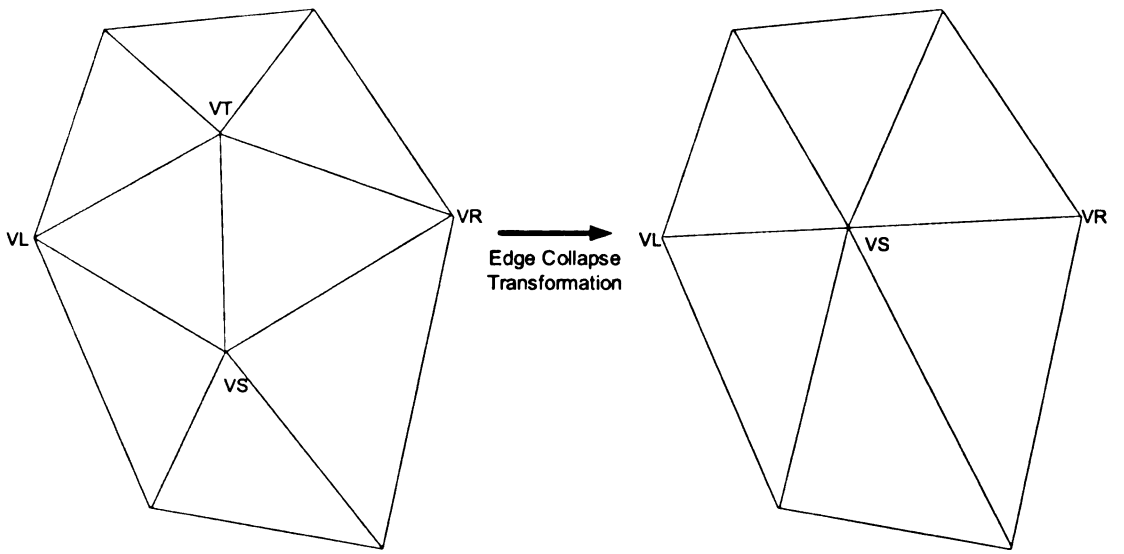


Figure 3.5: Edge collapse operation.

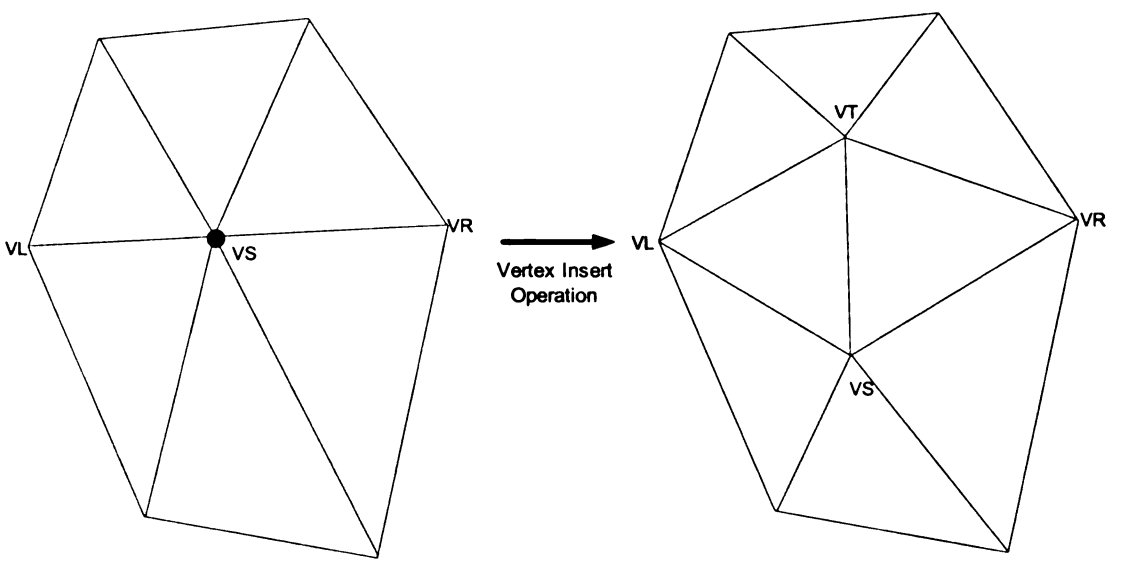


Figure 3.6: Vertex insert operation.

The key point of this approach is to determine which edge to collapse during the process. If this edge is randomly selected, the progressive mesh will not be able to keep its appearance during reconstruction. The author improved the energy function proposed in his earlier paper and used that as the metric [63]. Each edge selection attempts to

minimize the energy function. Therefore, the overall quality of progressive meshes is decided by the energy function.

Hoppe's progressive mesh representation is naturally a compact encoding method. For a mesh with n vertices and approximately 2n faces, the author estimated that the size of vertex insert records requires about (log(n) + 5)n bits. Using a similar method used in Deering's method with the progressive mesh representation, the geometry of the mesh can be encoded in 31n to 50n bits.

Taubin et al. proposed a method to group Hoppe's insertions into refinements [15]. A mesh is split into a forest of spanning trees of vertex runs. The forest split operation can be seen as a grouping of several consecutive edge split operations into a set. This decomposition is invertible so that the complete connectivity can be used to reconstruct the original mesh. A high compression ratio can be achieved by using this method when the size grows exponentially with LOD.

Li and Kuo combined progressive transmission of connectivity refinements with progressive transmission of vertex coordinates [12]. A vertex decimation scheme is used to produce series of operations that decrease the LOD of the model. Their approach leads to $0.5\log_2(|V|)+5$ bits for each triangle. Pajarola and Rossignac attempted to compactly encode progressive mesh representations for triangle meshes [11]. It extends Hoppe's idea by grouping simplifications and refinements to increase connectivity compression efficiency. Khodakovsky et al. proposed a wavelet based scheme for progressive geometry compression that works well on regular and semi-regular connectivity meshes [17]. The limitation of their method is that it does not work effectively on arbitrary

meshes. They defined wavelet basis functions over manifold surfaces for multi-resolution analysis. The multi-resolution technique provides LOD progressively.

3.2 Animated geometry compression

A large body of geometry compression research has been focused on static geometry, while only limited research has addressed animated geometry compression [22-26, 64-66]. This section aims to review some notable compression techniques used for 3D animation sequences.

3.2.1 Introduction

Animated geometry compression is the compression of temporal sequences of geometric data. Animated geometry compression makes possible the delivery of an animated motion picture as a 3D model using the least possible bandwidth in real time on demand or as a multicast. Animated 3D geometry models require larger memory and transmission bandwidth since a sequence consists of a large number of frames and every frame is an already large static 3D object. Therefore, efficient compression techniques need to be applied to the 3D animation sequence before distributing it in the network. The following sections give a survey of recent notable work in animated geometry compression.

3.2.2 Time-dependent geometry compression

Lengyel's work aims to transmit and play-back large time-dependent geometry in real time using the minimum required bandwidth [22]. His method is based on the idea that with proper mesh representation, the gross movement of a geometric mesh can be

coded with a small set of controls and the residual differences can be quantized and coded in a low bit rate. He presents a new view of time-dependent geometry as a media stream and presents a new technique for taking advantage of the large amount of coherence in time. This method exploits the coherence across animated frames by using a predictive encoder and describes a method to compress a generic animated geometry stream by solving for few-parameter models and encoding the residual. This prediction compression method splits vertices into sets and uses affine transformations to approximate the vertex paths.

This method is effective only when the animation is well represented by the supported transformations.

3.2.3 Principle component analysis approach

Alexa and Müller proposed an interpolation predictor to represent animations by principal components [23]. Their compression scheme works as follows: Given a 3D animated mesh with n vertices and m frames, all shapes are translated so the center of the mass is in the original, and then an affine transformation from each frame to the first frame is computed using the linear least square fit to normalize all the frames. They then build a 3n by m matrix A using the result frames from the normalization step, where each column of the matrix A is the geometry of one frame. Singular value decomposition is performed on matrix A to obtain the eigenvectors of AA^T. These eigenvectors are stored in the result matrix U. It is used to define a new basis replacing the original frames. This is a transform coding method, where the basis functions are determined directly from the data set.

Principal components analysis makes their approach expensive. The approach requires significant memory usage and processing time. It cannot achieve high compression ratios if the number of frames is significantly smaller than the number of vertices. It is also hard to control the quality of individual frames in the animation sequence. Since the method is global over a large range of frames, some frames may exhibit considerably larger errors than others or the average. It is good in terms of smoothness of the shape.

Among those notable approaches, PCA can achieve higher compression ratios. In order to compare PCA approach to the octree-based approach, PCA approach was reimplemented in this thesis. The detailed comparison results can be found in Chapter 4.

3.2.4 Dynapack

Ibarria and Rossignac proposed a time-space predictor for all the vertices and all the frames [24]. The geometry of a vertex is predicted based on the geometry of its neighbors in the same frame and the geometry of corresponding vertices in the previous frame.

The dynapack encoder runs very fast [26]. Additionally, it is predominantely lossless in nature. The only lossy element of the algorithm is caused by quantization, which can be ignored by using 12-bit quantization. This approach requires re-ordering the vertices of the mesh to meet the needs of a particular configuration of neighboring vertices in the current and previous frame. The compression ratio this approach can achieve is approximately 10-22 to 1.

3.2.5 Geometry Video

Briceño et al. proposed an algorithm to generate geometry video from a 3D animation sequence [25]. This approach is based on the geometry image representation proposed by Gu et al. [27]. A geometry image unwraps a polygon mesh surface onto a 2D color image that can then be subjected to conventional image compression methods. The method exploits conventional image compression methods indirectly by converting geometric data into images.

A geometry video consists of a sequence of geometry images. To generate a geometry video from a 3D animation sequence, they extend the three major steps of constructing geometry image: cut, parameterization and compression. The algorithm works as follows: They first slice the mesh using the cut algorithm in order to create a surface with the topology of a disk. Slicing converts a manifold surface into a surface. Given a single cut that is the same for all the frames in the sequence, they aim to seek a single parameterization that minimizes the error across all the frames. Once they have determined a cut and a parameterization that work well across all the frames, each frame in the animation sequence can be transformed into a geometry image. All the frames in the animation sequence are replaced by the corresponding geometry images. The resulting geometry image sequence is called geometry video. They apply standard video compression techniques such as MPEG to encode the geometry video.

Their approach supports LOD because it is easy to get a coarse version of the mesh by removing every other pixel in each direction of the geometry images.

3.2.6 Soft-body animation compression

Karni and Gotsman presented a technique to compress soft-body animation sequences [26]. Their approach is based on the principle component analysis idea proposed by Alexa and Müller [23]. Linear prediction coding is applied to the PCA coefficients in order to achieve further code reduction by capturing the temporal behavior present in the sequence. Their compression scheme works as follows: they first applied singular value decomposition to a 3n by m matrix A, where n is the number of vertices and m is the number of frames in the animated 3D mesh. The eigenvectors of matrix AA^T are stored in matrix U. Coefficient matrix (U^TA) is obtained by projecting the original animation onto the new basis. Using only low frequency parts results a new k by m coefficient matrix. In the end they apply linear prediction coding on the resulting coefficients.

3.3 Summary

This chapter reviewed geometry compression techniques, both static and animated. Static geometry compression techniques can be divided into two categories: vertex compression and connectivity compression techniques. These techniques are suitable for static 3D meshes.

Connectivity compression is not the focus of this dissertation. This dissertation is focused on compressing animated polygonal meshes with fixed connectivity. Compared to the huge amount of geometry information in each frame, connectivity information is negligible because it needs to be transmitted only once. A straightforward way to compress 3D animated meshes is to apply static geometry compression algorithm

individually to each frame in the animation sequence, but this does not capture the temporal coherence present in the animation sequence. Animation compression techniques need to exploit not only the space coherence, but also the temporal coherence existed in the sequence.

The chapter also reviewed recent efforts to compress animated 3D meshes. All these approaches assume fixed connectivity. Each approach has its advantages and disadvantages. The time-dependent geometry compression scheme needs to split the dataset to fit different transforms. PCA approach is computationally expensive for large meshes. Dynapack requires re-ordering of the vertices of the mesh. The soft-body animation compression scheme also suffers from significant memory usage and processing time because it is based on PCA approach. Geometry videos, while interesting and colorful, are not really competitive as animated geometry compression methods.

In the next chapter, a novel octree-based animated geometry compression approach is presented. This approach can achieve high compression ratios and quality with no restriction on the data set. A simple and a low cost encoding process and a fast decoding process make this approach quite suitable for real time applications.

Chapter 4

4 Octree-based animated geometry compression

This chapter presents a new algorithm for animated geometry compression. The algorithm uses an octree-based motion representation with motion vectors associated with the corners of the octree cells. The motion vectors capture the motion coherence within spatial localities. This new method efficiently compresses 3D animated geometric data.

Section 4.1 introduces the data representation for 3D animation. In this thesis, three different data sets are used to evaluate the performance of new algorithms in relation to the existing methods. Section 4.2 presents these data sets. Section 4.3 describes the octree-based animated geometry compression in detail. Section 4.4 presents the compression results, the comparison between the octree and PCA approaches and the comparison between the octree approach and a simplistic approach that compresses each frame in the sequence using static geometry compression.

4.1 Data representation

This chapter assumes the mostly simplistic representation of the geometric data: triangle meshes for each discrete frame in the animated sequence. The connectivity information is assumed to remain unchanged for each frame in the animation sequence. Any polygonal mesh can be converted into a triangle mesh through triangulation [67, 68]. A triangle mesh is represented by its vertex data and connectivity information. Vertex data consists of coordinates of all the vertices and, optionally, vertex normal vectors and

texture coordinates. Connectivity information is simply a triangle list. Each triangle contains indices referring to the vertices. Figure 4.1 shows the data representation for 3D animation.

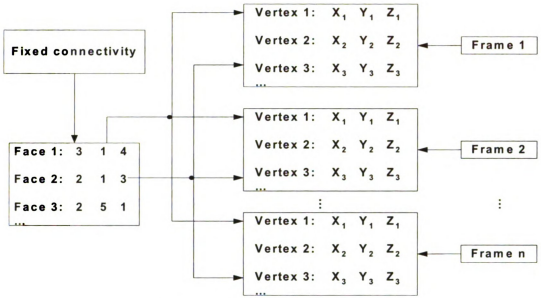


Figure 4.1: 3D animation representation.

This thesis is focused on compressing vertex positions for all the frames. Polyhedral simplification techniques are not suitable to compress animated geometric data because these methods cannot maintain the exact connectivity of the model. The straightforward way to compress 3D animated meshes is to apply vertex data compression techniques [5, 6, 8, 11] individually to each frame in the animation sequence, but this approach does not take advantage of the temporal coherence present in the animation sequence. Animation compression techniques need to exploit not only the spatial coherence, but also the temporal coherence in the sequence.

To show the data coherence in the animation sequence, the following tests are conducted on the dance sequence. For each vertex, compare the motion of the vertex to the average motion of its nearest four neighboring vertices in terms of vector length and

angle between these two vectors. Figure 4.2 shows the histogram of vector length differences between the motion of the vertex and the average motion of the nearest four neighboring vertices. Figure 4.3 shows the histogram of vector angle differences. These tests show that the motion of a vertex is predictable from its neighbors. There is a great deal of coherence among moving vertices.

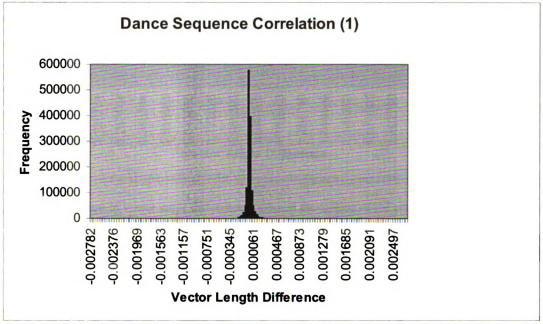


Figure 4.2: Histogram of vector length differences between the motion of the vertex and the average motion of the nearest four neighboring vertices.

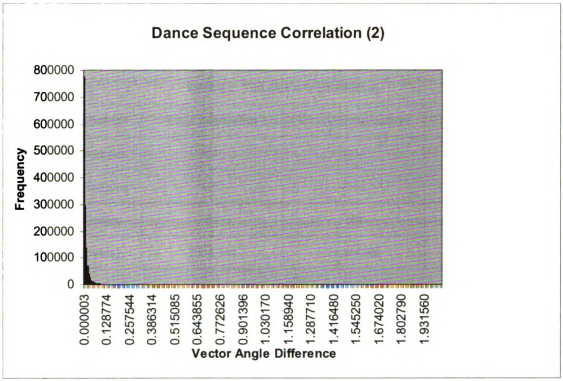


Figure 4.3: Histogram of vector angle differences between the motion of the vertex and the average motion of the nearest four neighboring vertices.

4.2 Test data sets

In this thesis, three different animation datasets are used to evaluate the performance of the octree-based approach and compare to other approaches. The data sets include "Chef", "Chicken Crossing" and "Dance". The Chef data was generated from a 3D Studio Max animation. It was used as an initial test data set. "Chicken Crossing" was created by Andrew Glassner et al.[69]. It was used as a test data set in the dynapack approach proposed by Ibarria and Rossignac [24] and PCA approach proposed by Alexa and Müller [23]. The "Dance" sequence was created by the MIT CSAIL Graphics Lab. It was used in the geometry video approach proposed by Briceño et al.[25]. To make it easier to compare the octree-based approach to other approaches in the literature, "Chicken Crossing" and "Dance" sequence were chosen as standard test data sets. Table

4.1 shows the properties of "Chef", "Chicken Crossing" and "Dance" animation sequences.

Table 4.1: Test data sets information.

Data sets Properties	Chef animation	Chicken animation	Dance animation
Number of vertices	4241	3030	7061
Number of triangles	8162	5664	14118
Number of frames	75	400	201
Total size (bytes)	3,816,900	14,544,000	17,031,132

Figure 4.4 shows the first frame and every 5^{th} frame of the original Chef animation sequence. Figure 4.5 shows the first frame and every 20^{th} frame of the original Chicken animation sequence. Figure 4.6 shows the first frame and every 10^{th} frame of the original Dance animation sequence.

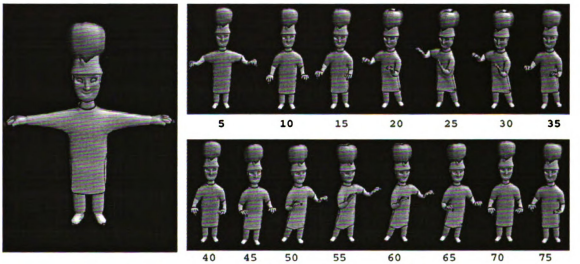


Figure 4.4: Sample frames from original Chef animation sequence.

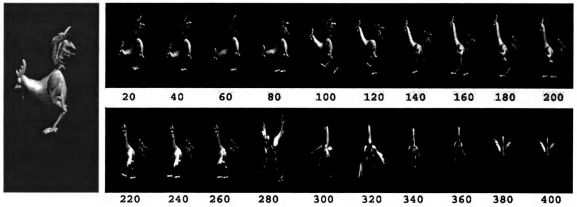


Figure 4.5: Sample frames from original Chicken animation sequence.

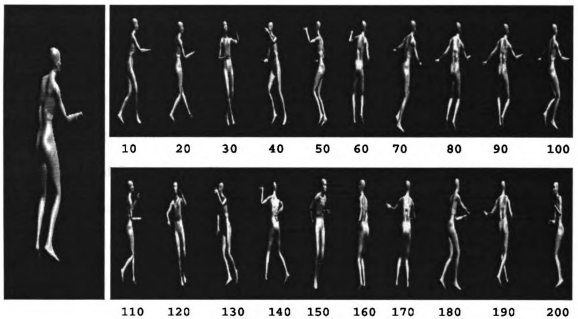


Figure 4.6: Sample frames from original Dance animation sequence.

4.3 Octree-based approach

The octree-based approach presented in this chapter takes advantage of the spatial and temporal coherence present in the data. Two consecutive frames are needed to generate a reduced set of motion vectors that represent the motion from the previous frame to the current frame. The motion vectors are used to predict the vertex positions in

the decoder side for each frame except for the first frame. The process generates a hierarchical octree motion representation for each frame.

The method can be classified as a predictive modeler in that it attempts to create a reduced set of data that can be used to predict the actual vertex motion over a localized area. The motion vectors are then subject to encoding using arithmetic coding.

4.3.1 Encoding process

In a raw animated data set, an animated 3D model is represented by the vertex positions for each frame. Connectivity is assumed to remain constant. This is an assumption in all of the current work on animated geometry compression, though future work must eventually address dynamic changes in topology. Chapter 7 discusses system issues including the interspersing of intra-coded frames which can support geometry additions or modifications as well as scene cuts.

In the motion modeling process, each frame except the first frame is represented by a set of motion vectors, which are stored in an octree. In a client–server scenario, when the server wishes to send an animated 3D model to the client, it will send the encoded first frame (using an intraframe encoding method) together with an encoded octree for each of the subsequent frames rather than sending vertex positions for each new frame. The encoded octree is significantly smaller than the corresponding original frame. The vertex locations for each frame are approximated by the corresponding octree on the client side.

The logical flow of the encoding process is illustrated in Figure 4.7. To encode the current frame, the previous frame needs to be available to the decoder, so the first frame is encoded using an intraframe coding technique. A complete system will include

occasional intra-coded frames so as to provide synchronization points, as in MPEG video and as discussed in Chapter 7. The previous and current frames are used to generate vertex delta values representing the differential motion between the frames. Motion vectors are obtained by using a Least Square Error estimation method to estimate the motion of enclosed vertices. The motion vectors are then subject to encoding using adaptive arithmetic coding. To continue encoding the following frames, a local decoder is called to generate a decoded version of the current frame as known to the receiving system, which can be used to encode the next frame at the encoder side. This process is repeated for each new frame.

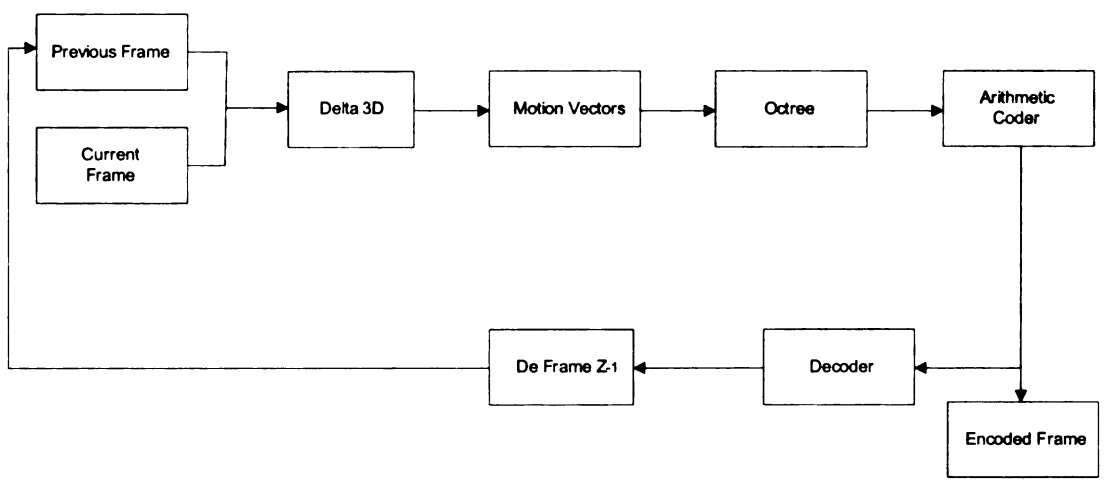


Figure 4.7: Logical flow of encoding process for compressing 3D animated frames.

4.3.2 Motion-modeling process

The most important step of the encoding process is the construction of an octree. This process is called the *motion-modeling process*. Jackins and Tanimoto showed that octrees can be used to represent 3D objects [70]. Ahuja and Nash improved the idea presented by Jackins and Tanimoto, and proposed a method to represent moving 3D objects using octree structures [71]. The octree representation of the occupancy of space

by objects is obtained by recursive decomposition of the space into octants. Octrees are a common method for modeling 3D data.

Instead of using an octree to represent a 3D object, this thesis uses an octree to represent motion within space. Constructing a minimum cube (axis parallel bounding box) that contains all the vertices of a 3D object as a starting box or *cell* is the initial step of motion-modeling process. A motion vector is associated with each corner of the cell as illustrated in Figure 4.8 (a). These eight motion vectors approximate the motion of the enclosed vertices using interpolation of the motion vectors over the space. If the motion of all vertices in the current cell is not well approximated by the motion vectors, the cell is subdivided into eight child octants as shown in Figure 4.8 (b) and the corresponding tree node generates eight children as shown in Figure 4.8 (c). Each octant has new motion vectors that are stored in the corresponding tree nodes. The splitting process continues until the motion of the enclosed vertices is estimated to be below a specified threshold. Therefore, any reconstructed vertex in any frame in the reconstructed animation sequence satisfies the requirement that the error between the reconstructed vertex and the corresponding original vertex is smaller than the threshold as illustrated in Equation 4.1. In this thesis, the *Max Distance* threshold measurement is examined for the splitting process. *Distance* for a vertex is defined as Euclidean error between the original vertex and the reconstructed vertex. Max Distance is the maximum error of all the enclosed vertices. The *threshold* is set to be the maximum allowed distance between the reconstructed vertex and its corresponding original vertex. It is represented by the percentage of the edge length of the initial cubic bounding box of the object as illustrated in the Equation 4.2.

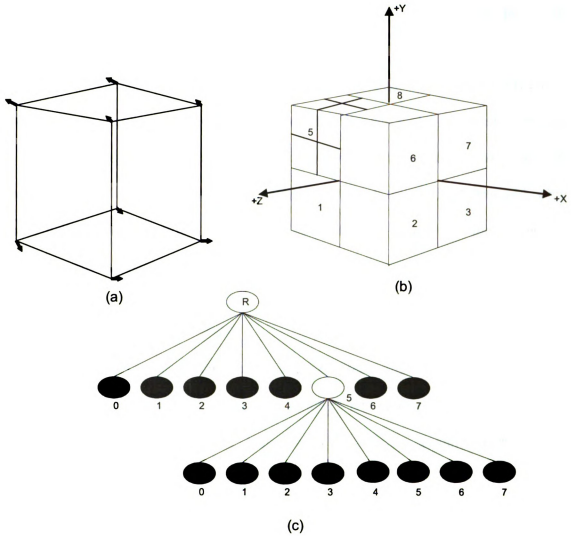


Figure 4.8: (a) Eight motion vectors. (b) The splitting process. (c) Corresponding octree representation.

$$\forall v \in current\ cell$$
$$\sqrt{(v_x - v_x')^2 + (v_y - v_y')^2 + (v_z - v_z')^2} < threshold \tag{4.1}$$

v and v' represent the original vertex and the corresponding reconstructed vertex.

$$Threshold = \%Length \tag{4.2}$$

Length is the edge length of the initial cubic bounding box. The chosen percentage influences the quality; as the percentage increases the quality decreases.

The tree representation of the subdivision process is shown in Figure 4.8 (c). Each node has eight motion vectors. Motion vectors of all the leaf nodes represent the motion of the entire object. Black nodes (leaf nodes) represent motion vectors that can approximate all the vertex movements enclosed in this cell to a defined threshold; white nodes indicate that the current cell needs to split into eight octants and the current tree node will generate eight children.

4.3.3 Tri-linear interpolation

Motion for vertices within a cell is approximated using tri-linear interpolation of the motion vectors. Given a cell of size s, lower left rear corner (minimum x,y,z) $<b_x,b_y,b_z>$, motion vectors $m_1,...,m_8$, and a vertex $v=<v_x,v_y,v_z>$ as shown in Figure 4.9, the ratio ρ_x in the x direction is computed using Equation 4.3.

$$\rho_x = \frac{v_x - b_x}{s} \tag{4.3}$$

In a similar way, the ratios ρ_y in the y direction and ρ_z in the z direction can be computed. The weight for each motion vector $w_1,...,w_8$ is computed using Equation 4.4.

$$
\begin{aligned}
w_1 &= (1-\rho_x)\rho_y\rho_z \\
w_2 &= (1-\rho_x)(1-\rho_y)\rho_z \\
w_3 &= \rho_x(1-\rho_y)\rho_z \\
w_4 &= \rho_x\rho_y\rho_z \\
w_5 &= (1-\rho_x)(1-\rho_y)(1-\rho_z) \\
w_6 &= \rho_x(1-\rho_y)(1-\rho_z) \\
w_7 &= \rho_x\rho_y(1-\rho_z) \\
w_8 &= (1-\rho_x)\rho_y(1-\rho_z)
\end{aligned}
\tag{4.4}
$$

The vertex motion Δv for the target point is then computed using tri-linear interpolation of the eight motion vectors of the cell [72], as illustrated in Equation 4.5.

$$
\Delta v = \sum_{i=1}^{8} w_i m_i \tag{4.5}
$$

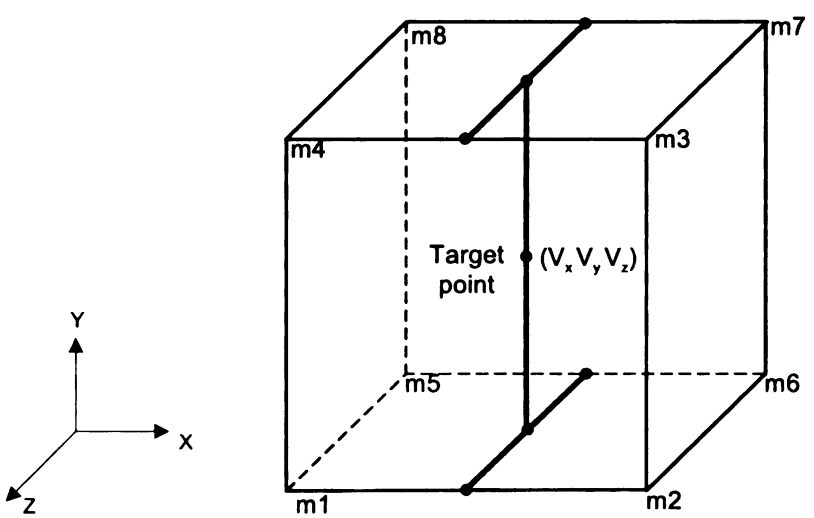


Figure 4.9: Tri-linear Interpolation.

The use of tri-linear interpolation of corner representational motion vectors for estimation of vertex motion accommodates a wide variety of motion styles including affine motion, the most common rigid motion in computer animation. Non-linear motion or discrete (independent) object motion is approximated using the octree subdivisions.

Other methods for motion estimation for a cell will be explored in later work including compact quaternion representations and tri-cubic interpolations.

4.3.4 Motion vector computation

Each node in the octree approximates the motion of enclosed vertices. The key point of the motion-modeling process is to compute the motion vectors. Motion vectors are assumed to approximate the motion of the enclosed vertices using tri-linear interpolation of the motion vectors over the space. Let n be the number of vertices enclosed in a given cell. Matrix A is a 3n by 24 matrix and is built as illustrated in Equation 4.6.

$$A = \begin{bmatrix} w_1 & 0 & 0 & w_2 & 0 & 0 & \cdots & w_8 & 0 & 0 \\ 0 & w_1 & 0 & 0 & w_2 & 0 & \cdots & 0 & w_8 & 0 \\ 0 & 0 & w_1 & 0 & 0 & w_2 & \cdots & 0 & 0 & w_8 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \tag{4.6}$$

Vector b, a 3n element vector is computed using Equation 4.7.

$$b = \begin{bmatrix} \Delta v_{x_1} \\ \Delta v_{y_1} \\ \Delta v_{z_1} \\ \Delta v_{x_2} \\ \Delta v_{y_2} \\ \Delta v_{z_2} \\ . \\ . \\ . \end{bmatrix} \tag{4.7}$$

$$\begin{bmatrix} w_1 & 0 & 0 & w_2 & 0 & 0 & \cdots & w_8 & 0 & 0 \\ 0 & w_1 & 0 & 0 & w_2 & 0 & \cdots & 0 & w_8 & 0 \\ 0 & 0 & w_1 & 0 & 0 & w_2 & \cdots & 0 & 0 & w_8 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} * \begin{bmatrix} m_{x_1} \\ m_{y_1} \\ m_{z_1} \\ m_{x_2} \\ m_{y_2} \\ m_{z_2} \\ . \\ . \\ . \end{bmatrix} \approx \begin{bmatrix} \Delta v_{x_1} \\ \Delta v_{y_1} \\ \Delta v_{z_1} \\ \Delta v_{x_2} \\ \Delta v_{y_2} \\ \Delta v_{z_2} \\ . \\ . \\ . \end{bmatrix} \quad (4.8)$$

The motion vectors for a given cell are then computed using least square estimation of x in the equation $Ax=b$ as shown in Equation 4.8, where x is a 24 element vector consisting of the $<x, y, z>$ components of eight motion vectors. This process finds the vector x that minimizes the sum of residual squares as illustrated in Equation 4.9. In the implemented solution, the least square estimation is computed using QR decomposition. The matrix A is decomposed into a 3n-by-24 orthogonal matrix Q and a 24-by-24 upper triangular matrix R so that A = QR. Template Numerical Toolkit (TNT) and JAMA/C++ linear algebra library are used to compute vector x [73]. A least square estimate of the motion vectors in this application minimizes the Euclidian distance between the motion vectors. This is identical to minimizing the Euclidian distances between the reconstructed and actual vertex locations after motion.

$$Residual\ sum = \sum_{i=1}^{3n} (A_i x - b_i)^2 \quad (4.9)$$

To avoid accumulative error, the previous decoded frame is utilized to compute the next set of motion vectors rather than the previous input frame on the encoder side. For example, to build an octree representation of frame 3, construct a minimum bounding

box that contains all the vertices of the 3D model as the first step. Given decoded frame 2 and original frame 3, vector b can be obtained easily; matrix A can be built by using the tri-linear interpolation factors for all the vertices in the current cell based on the reconstructed frame 2 locations. Using least square estimation, motion vectors are computed, which provide the best estimate of the motion of the enclosed vertices in the current cell. After obtaining the motion vectors for the current bounding cell, the accuracy of the estimation is evaluated. A distance threshold is used to measure accuracy of the current motion vectors. If maximum distance is larger than a specified threshold, the current cell is subdivided into eight octants, each with eight new motion vectors to estimate the motion of its enclosed vertices.

The splitting process continues until the motion estimated by the motion vectors is less than the threshold. Therefore all leaf nodes of the octree contain the motion vectors for the entire object. The octree is stored into two parts: the tree structure and the motion vectors. '0' is used to indicate an interior node and use '1' to indicate a leaf node. The tree structure is represented with prefix traversal of the tree. The tree structure can be restored uniquely using this representation. The motion vectors are stored in the leaf nodes. The motion vectors can be consecutively stored in the leaf nodes from left to right.

4.3.5 Adaptive arithmetic coding

Adaptive arithmetic coding is used to encode the octree. Adaptive coder has several advantages. It does not need two passes and there is no need to transmit the probabilities of the symbols. An adaptive model changes the probabilities of the symbols in the compression process in order to adapt to the changing contexts. Both the encoder and decoder assume some initial basic model. Then in the compression process, the

model gets adapted by the symbols transmitted before. The point is that the model gets adapted only by the symbols that have already been transmitted, since the decoder needs to mirror the encoder's operation later in the decompression process. The decoder can therefore use the symbols transmitted before to adapt the model in the same way as the encoder did.

The version implemented by Witten et al. is utilized in the octree coding [74]. An improved implementation of this arithmetic coding is described in the paper written by Moffat et al. [75].

4.4 Evaluations

This section presents an evaluation of the performance of the octree-based motion representation method in terms of the compression ratio and L^2 distances. PCA approach was reimplemented and compared to the octree-based approach in detail. This section includes brief comparison between the octree approach and pure static mechanism. A brief comparison of the octree approach to other animated geometry compression techniques is also included.

4.4.1 Octree results

Test results for the octree approach using all three test data sets are presented in this section.

4.4.1.1 Compression ratios

The compression ratio results for the Chef, Chicken Crossing and Dance animation are shown in Table 4.2, Table 4.3 and Table 4.4 respectively. A selected set of

reconstructed Chef, Chicken Crossing and Dance animations are shown in Figure 4.10, Figure 4.11 and Figure 4.12 respectively.

Table 4.2: Compression ratios of the Chef animation. The Chef animation has 75 frames and each frame has 8162 triangles and 4241 vertices. The object size is 440 units in x dimension, 148 units in y dimension and 455 units in z dimension. Original file size is 3,816,900 bytes. Max Distance threshold =p% *455 units, where p=1, 2, 3, ...7.

Chef Animation	Max Distance						
	1%	2%	3%	4%	5%	6%	7%
Encoded Size	309,593	173,853	115,689	90,752	77,101	50,474	41,112
Ratio	12:1	22:1	33:1	42:1	50:1	77:1	93:1

Table 4.3: Compression ratios of the Chicken Crossing animation. The Chicken Crossing animation has 400 frames and each frame has 5664 triangles and 3030 vertices. The object size is 2.556 units in x dimension, 2.23 units in y dimension and 1.07 units in z dimension. The original file size is 14,544,000 bytes. Max Distance threshold= p% *2.556 units, where p=2.5, 5, 7.5, 10, ... 20.

Chicken Animation	Max Distance							
	2.5%	5%	7.5%	10%	12.5%	15%	17.5%	20%
Encoded Size	1,540,423	820, 417	517,648	367,387	270,287	218,639	165,678	124,254
Ratio	9.4:1	18:1	28:1	40:1	54:1	67:1	88:1	117:1

Table 4.4: Compression ratios of the Dance animation. The Dance animation has 201 frames and each frame has 14118 triangles and 7061vertices. The object size is 0.0758 units in x dimension, 0.172 units in y dimension and 0.074 units in z dimension. Original file size is 17,031,132 bytes. Max Distance threshold=p%*0.172 units, where p=1, 2, 3, ... 8.

Dance Animation	Max Distance							
	1%	2%	3%	4%	5%	6%	7%	8%
Encoded Size	1,118,420	687,636	478,444	369,675	287,510	227,481	185,108	161,163
Ratio	15:1	25:1	36:1	46:1	59:1	75:1	92:1	106:1

Figure 4.10: A selected set of reconstructed Chef animation. The top row is the original animation. The reconstructed animation by using *Max Distance threshold* = 1%, 3%, 5%, 7% are shown in row 2, 3, 4, 5 respectively. The compression ratio is 12:1, 33:1, 50:1, and 93:1 in row 2, 3, 4, 5 respectively.

Figure 4.11: A selected set of reconstructed Chicken Crossing animation. The top row is the original one. The reconstructed animation by using *Max Distance threshold* = 5%, 10%, 15% and 20% are shown in row 2, 3, 4, 5 respectively. The compression ratio is 18:1, 40:1, 67:1 and 117:1 in row 2, 3, 4, 5 respectively.

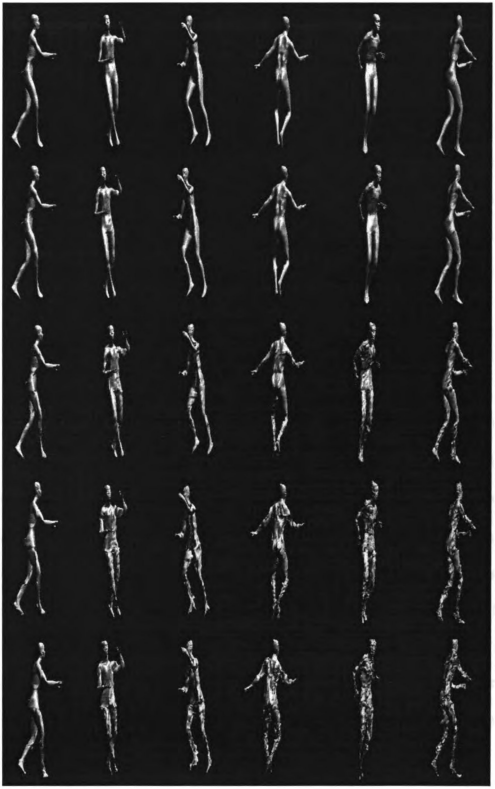


Figure 4.12: A selected set of reconstructed Dance animation. The top row is the original animation. The reconstructed animation by using *Max Distance threshold* = 1%, 3%, 5%, 7% are shown in row 2, 3, 4, 5 respectively. The compression ratio is 15:1, 36:1, 59:1, and 92:1 in row 2, 3, 4, 5 respectively.

4.4.1.2 Quality

Quantitatively evaluating the visual quality of the reconstructed 3D model is not easy, and assessing the visual quality of 3D animation is even more difficult. L^2 distance is widely used in the literature to measure the quality so this thesis uses it to evaluate the quality of reconstructed frames in the animation sequences. This is done in a similar way to Briceño et al. [25]. This L^2 metric is also used in the following section in order to compare the octree-based method to PCA approach proposed by Alexa and Müller [23] The L^2 distance from surface X to surface Y is defined as follows:

$$d(X,Y) = \left(\frac{1}{area(X)} \int_{x \in X} d(x,Y)^2 dx \right)^{\frac{1}{2}} \tag{4.7}$$

Where $d(x,Y)$ is defined as the minimum Euclidean distance from the point x to the surface Y. The distance computed from Equation 4.7 is not symmetric, so the L^2 distance of two surfaces is defined as the $\max(d(X,Y), d(Y,X))$. This thesis utilized the gtscompare functionality provided in the GTS library [76]. For the integral approximation, 0.5 percent of the side length of the initial bounding box diagonal is used for the delta step. For simplified surfaces, L^2 distance can be obtained by using Metro tool [77]. Figure 4.13 shows comparison of L^2 distance for some sample frames in the reconstructed Chef animation given different threshold settings. Similarly Figure 4.14 and Figure 4.15 show the comparison results for Chicken and Dance animation respectively.

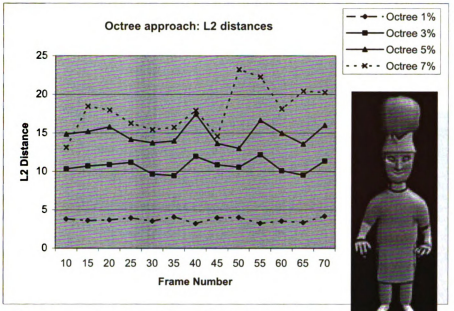


Figure 4.13: L^2 distances of some sample frames in the reconstructed Chicken animation by using *Max Distance threshold* = 1%, 3%, 5% and 7% respectively. The compression ratio is 12:1, 33:1, 50:1, and 93:1 respectively.

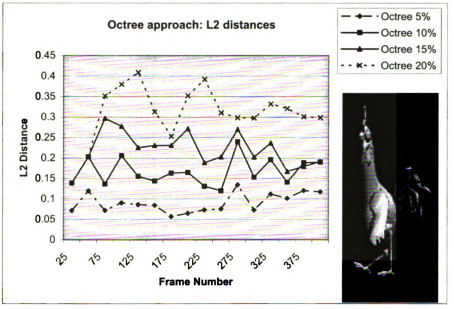


Figure 4.14: L^2 distances of some sample frames in the reconstructed Chicken animation by using *Max Distance threshold* = 5%, 10%, 15% and 20% respectively. The compression ratio is 18:1, 40:1, 67:1 and 117:1 respectively.

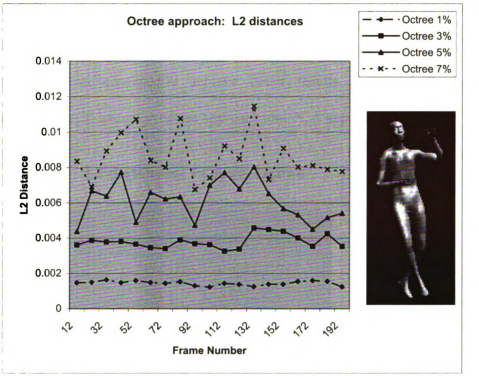


Figure 4.15: L^2 distances of some sample frames in the reconstructed Dance animation by using *Max Distance threshold* = 1%, 3%, 5% and 7% respectively. The compression ratio is 15:1, 36:1, 59:1, and 92:1 respectively.

4.4.2 Octree vs. PCA

To compare the octree approach with existing approaches in the literature, the PCA approach was reimplemented and evaluated. PCA was chosen for comparison because it can achieve relatively high compression ratios with reasonable quality. Three different animation data sets: "Chef", "Chicken Crossing" and "Dance" were used for comparison. The compressed size of the octree approach is the sum of encoded octree size for each frame. The compressed size of PCA approach should include the affine transformations for each frame, matrix U (number of bases * number of vertices * 12 bytes/vertex), matrix S (number of bases*4) and matrix V (number of bases * number of frames*12). Given approximately the same compression ratio, L^2 distances of two systems are compared. Some comparison results for the Chef animation sequence are

shown in Figure 4.16 and 4.17 respectively. In Figure 4.17, the average L^2 distance for the octree approach and PCA approach are 8.7 and 19.1 respectively. This means the octree approach can consistently achieve better results. Figure 4.18 shows the reconstructed frame 25 using the octree approach and PCA approach.

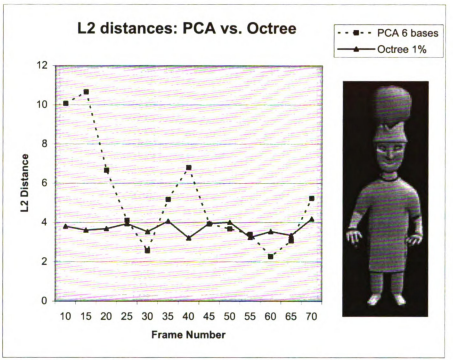


Figure 4.16: L^2 distance comparison between the octree and PCA approaches using the Chef animation. This comparison is based on approximately the same compression ratio (12:1). To get this ratio, threshold in the octree approach is set to be 1% of the initial cubic bounding box size. It corresponds to PCA with 6 bases.

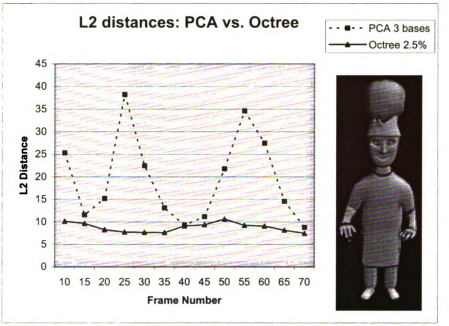


Figure 4.17: L^2 distance comparison between the octree and PCA approaches using the Chef animation. This comparison is based on approximately the same compression ratio. The compression ratio of PCA with 3 bases is 24:1 and the compression ratio of the octree with threshold 2.5% is 27:1.

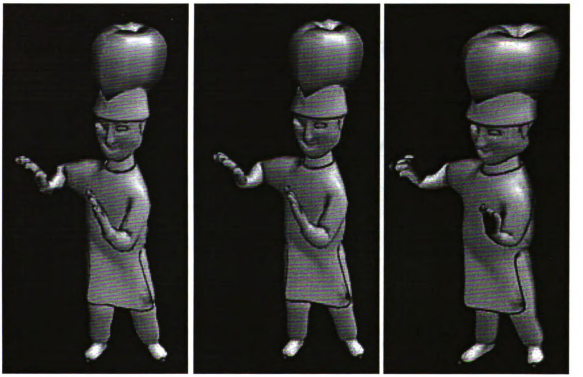


Figure 4.18: Image in the middle is the original frame 25. The reconstruction using the octree approach with threshold 2.5% and PCA with 3 bases are shown in the first and last image respectively.

L^2 distance comparison between the octree and PCA approach using the Chicken animation sequence are shown in Figure 4.19 and 4.20 respectively. In Figure 4.20, the average L^2 distance for the octree approach and PCA approach are 0.167 and 0.18 respectively. Figure 4.21 shows a reconstructed sample frame using the octree and PCA approaches. Figure 4.19 shows that the octree approach has much larger L^2 distance error for frame 300 than PCA approach. However the quality of the reconstructed image using PCA approach is similar to the quality of the octree approach as shown in Figure 4.21. L^2 distance could not exactly reflect the visual quality of the reconstruction. To get more accurate quality evaluation, the quality metric may take into account the effect of human visual system (HVS).

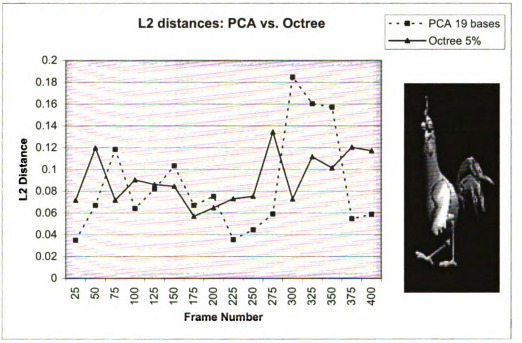


Figure 4.19: L^2 distance comparison between the octree and PCA approaches using the Chicken animation. This comparison is based on approximately the same compression ratio. The compression ratio of PCA with 19 bases is 19:1 and the compression ratio of the octree with threshold 5% is 18:1.

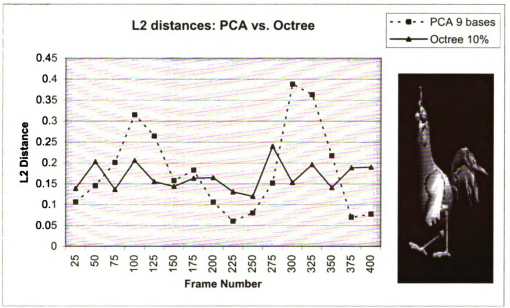


Figure 4.20: L^2 distance comparison between the octree and PCA approaches using the Chicken animation. This comparison is based on approximately the same compression ratio (40:1). To get this ratio, threshold in the octree approach is 10%. It corresponds to PCA with 9 bases.

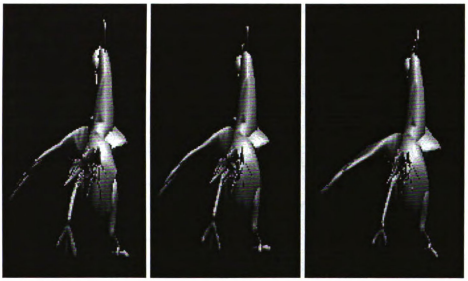


Figure 4.21: Image in the middle is the original frame 300. The reconstruction using the octree approach with threshold 5% and PCA with 19 bases are shown in the first and last image respectively.

L^2 distance comparison between the octree and PCA approaches using the Dance animation are shown in Figure 4.22 and 4.23 respectively. In Figure 4.22, the average L^2 distance for the octree approach is 0.0014 and the average L^2 distance for PCA approach is 0.019.

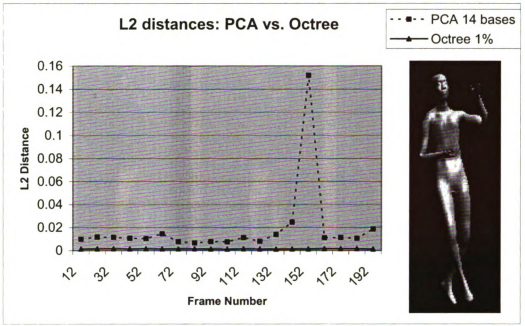


Figure 4.22: L^2 distance comparison between the octree and PCA approaches using the Dance animation. This comparison is based on approximately the same compression ratio. The compression ratio of PCA with 14 bases is 14:1 and the compression ratio of the octree with threshold 1% is 15:1.

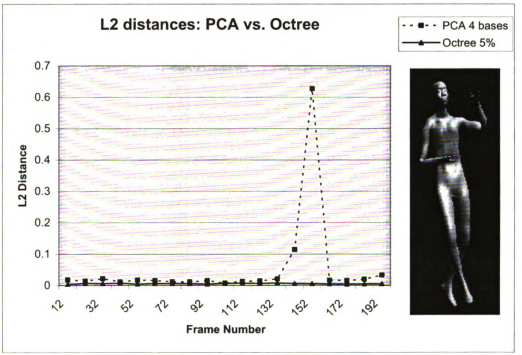


Figure 4.23: L^2 distance comparison between the octree and PCA approaches using the Dance animation. This comparison is based on approximately the same compression ratio. The compression ratio of PCA with 4 bases is 48:1 and the compression ratio of the octree with threshold 5% is 59:1.

Figure 4.24: Image in the middle is the original frame 150. The reconstruction using the octree approach with threshold 1% and PCA with 14 bases are shown in the first and last image respectively.

Figure 4.24 shows a reconstructed sample frame using the octree and PCA approaches.

Compared to PCA approach, the octree-based approach has some advantages. PCA approach requires significant memory usage and processing time. The octree-based approach has a much smaller memory requirement and runs much faster. The octree-based approach compresses the animation sequence equally well no matter how many frames are in the animation sequences. PCA approach cannot achieve high compression ratios if the number of frames is significantly smaller than the number of vertices. The PCA approach can not control the quality of an individual frame in the animation

sequence. Some reconstructed frames have outrageous error. However, the octree approach can control the quality of each frame in the sequence by setting the threshold. The average L^2 distance comparison shows that the octree approach is better than PCA approach in terms of result consistency. PCA approach is good in terms of smoothness of the shape.

4.4.3 Octree vs. pure static compression mechanism

Another way to compress animated 3D models is to apply static geometry compression algorithm individually to each frame in the animation sequence, but this does not take advantage of the temporal coherence present in the animation sequence. Additionally this approach is expensive for large models. This approach is tested using the "parallelogram" approach proposed by Touma and Gotsman [8]. For the test data sets used in this thesis, the TG coder can compress the animation with a compression ratio of 9 to 1, not a competitive animation result. Compared to this approach, the octree-based approach exploits not only the space coherence, but also the temporal coherence existed in the sequence. It can achieve higher compression ratios and better reconstruction quality.

4.4.4 Octree vs. other approaches

According to the results interpreted by Ibarria and Rossignac, dynapack approach can compress the entire chicken animation with 13.7:1 compression ratio when it uses 11-bit quantization and Lengyel's approach can achieve 14:1 compression ratio with comparable accuracy [24]. To get more accurate results, Lengyel's approach can get 2:1 compression ratio and dynapack can achieve 10.8:1 with 15-bit quantization [24]. Ibarria

and Rossignac claimed their approach could achieve undistinguishable result from the original using 13-bit quantization. In this case, the compressed ratio is 10:8 to 13.7: 1. The octree approach can achieve 18:1 compression ratio with quality undistinguishable from the original animation. When less accuracy is allowed, the octree-based approach can achieve 100:1 compression ratio. Compared to the results obtained by previously published animation compression techniques [22-24], the octree approach can achieve higher compression ratios with comparable animation quality.

4.5 Conclusions

This chapter described the octree-based motion representation method in detail. This method presents a novel way to implement predictive, differential frame compression. It achieves the goal of this thesis: represent 3D animation sequence with a reduced set of motion vectors that take advantage of the large data coherence in space and time. This approach presents a novel way to represent 3D animation. It is easy to implement and has a low cost encoding process and fast decoding process. Compared to other approaches, it can achieve higher compression ratios and at the same time achieve very good quality.

Chapter 5

5 Delta approach

This chapter presents a new *delta coding approach* and evaluation of the performance of this method for animated geometry compression. The delta coding approach generates 3D delta values from two consecutive frames. The method achieves the compression by quantizing the 3D delta values. Different frames in the animation sequence can use different quantization levels. Quantized values are coded using arithmetic coding to achieve further data reduction. This method is simple and easy to implement.

During the course of this research, the delta coding was explored as an alternative to the octree-based approach. This chapter will show that the delta coding method does not achieve the consistent level of the performance demonstrated by the octree approach. However, in limited cases the delta approach does outperform. Hence, it is explored here in isolation, and then incorporated into a hybrid compression approach in Chapter 6.

5.1 Delta encoding process

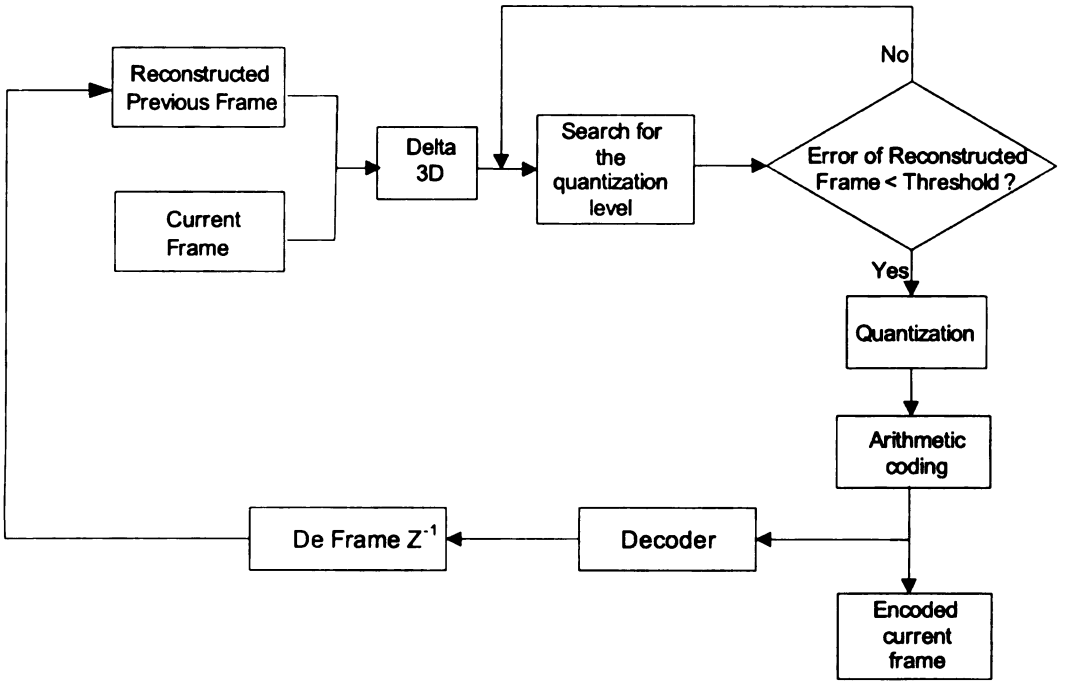


Figure 5.1: Logical flow of the encoding process of the delta method.

The logical flow of the encoding process is illustrated in Figure 5.1. To encode the current frame, the previous reconstructed frame needs to be available in the encoder side. Therefore the first frame is encoded differently from other frames in the sequence. It is encoded using an intraframe coding method. The first step in the delta encoding process is to generate delta 3D values between two consecutive frames, namely the reconstructed previous frame and the current original frame. Delta 3D data presents the differential motion between two frames. After the first step, the delta approach seeks the best quantization level with the minimum number of bits for the current delta 3D that satisfies a predefined threshold. After the number of bits for the quantization is determined, the *linear quantization* over the range of each possible vertex is applied to the delta 3D as illustrated in Equation 5.1. The quantized values are encoded using

adaptive arithmetic coding to further achieve data reduction. To this step, the current frame is completely delta encoded. In the local decoding step, a *linear inverse quantization* is applied to the quantized samples as shown in Equation 5.2.

$$\bar{x} = \left\lfloor 2^q \left(\frac{x - x_{\min}}{x_{\max} - x_{\min}} \right) \right\rfloor \tag{5.1}$$

$$x = \frac{(x_{\max} - x_{\min}) * \bar{x}}{2^q} + x_{\min} \tag{5.2}$$

The above steps are repeated for every frame except for the first frame in the animation sequence.

5.2 Evaluations

In order to compare the results of different approaches, the same test data sets as those in octree approach are used to evaluate the performance of the delta approach. Compression ratio and the L^2 distance results of the delta approach for different animation sequences are presented. This section also includes the overall and frame-wise compression ratio comparison between the delta approach and octree approach.

5.2.1 Delta approach results

This section presents the compression ratio results for Chef, Chicken Crossing and Dance animation sequences given different threshold settings. Table 5.1 shows the results for Chef animation. Similarly Table 5.2 and Table 5.3 show the results for the Chicken Crossing and Dance animation respectively.

Table 5.1: Compression ratios of the Chef animation using the delta approach. The Chef animation has 75 frames and each frame has 8162 triangles and 4241 vertices. The object size is 440 units in x dimension, 148 units in y dimension and 455 units in z dimension. Original file size is 3,816,900 bytes. Max Distance threshold =p% *455 units, where p=1, 2, 3, ...6.

Delta approach Chef sequence	Max Distance					
	1%	2%	3%	4%	5%	6%
Compression Ratio	17:1	21:1	31:1	32:1	32:1	34:1

Table 5.2: Compression ratios of the Chicken animation using the delta approach. The Chicken Crossing animation has 400 frames and each frame has 5664 triangles and 3030 vertices. The object size is 2.556 units in x dimension, 2.23 units in y dimension and 1.07 units in z dimension. The original file size is 14,544,000 bytes. Max Distance threshold= p% *2.556 units, where p=2.5, 5, 7.5, 10, 15, 20.

Delta approach Chicken sequence	Max Distance					
	2.5%	5%	7.5%	10%	15%	20%
Compression Ratio	21:1	26:1	28:1	30:1	33:1	35:1

Table 5.3: Compression ratios of the Dance animation using the delta approach. The Dance animation has 201 frames and each frame has 14118 triangles and 7061vertices. The object size is 0.0758 units in x dimension, 0.172 units in y dimension and 0.074 units in z dimension. Original file size is 17,031,132 bytes. Max Distance threshold=p%*0.172 units, where p=1, 2, 3, 4, 5, 6.

Delta approach Dance sequence	Max Distance					
	1%	2%	3%	4%	5%	6%
Compression Ratio	10:1	15:1	18:1	19:1	21:1	23:1

L^2 distance results for the test data sets given different threshold settings are also included in this section. Figure 5.2 shows the L^2 distance results for the Chef animation. Similarly L^2 distance results for the Chicken animation and Dance animation are shown in Figure 5.3 and Figure 5.4 respectively.

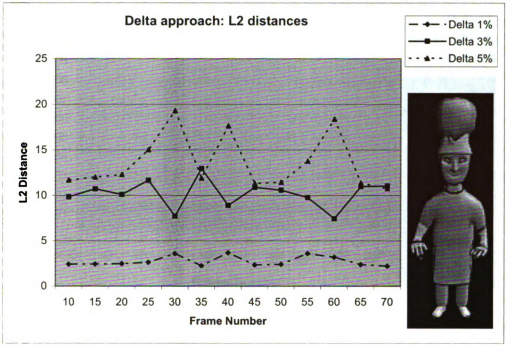


Figure 5.2: L^2 distances of some sample frames in the reconstructed Chef animation using the delta approach given different threshold settings.

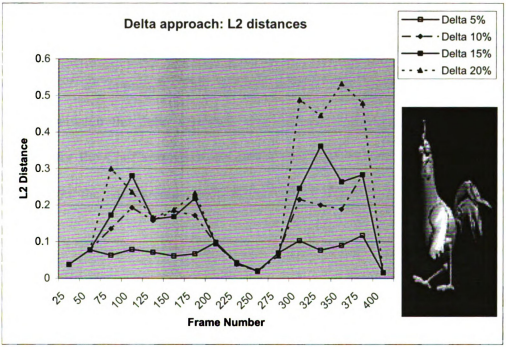


Figure 5.3: L^2 distances of some sample frames in the reconstructed Chicken animation using the delta approach given different threshold settings.

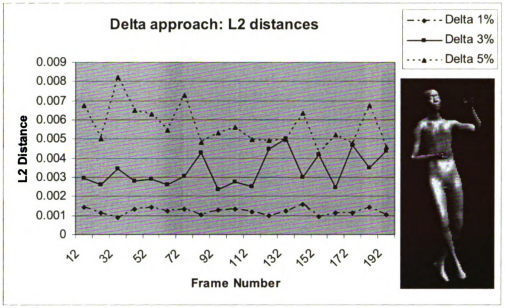


Figure 5.4: L^2 distances of some sample frames in the reconstructed Dance animation using the delta approach given different threshold settings.

5.2.2 Overall compression ratio comparison: delta vs. octree

Compression ratio is one of the most important measurements to evaluate the performance of an approach. This section presents the overall compression ratio comparison between the delta approach and the octree approach. Compression ratio comparisons for the Chef animation are shown in Figure 5.5. Similarly Figure 5.6 and Figure 5.7 show the comparison results for the Chicken Crossing animation and Dance animation.

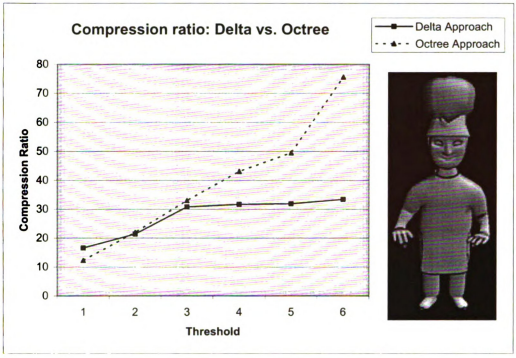


Figure 5.5: Compression ratio comparison between the delta and octree approach using the Chef animation.

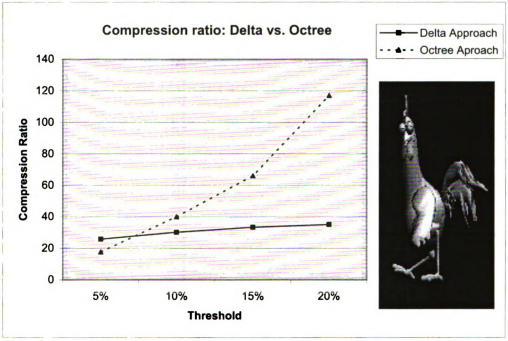


Figure 5.6: Compression ratio comparison between the delta and octree approach using the Chicken animation.

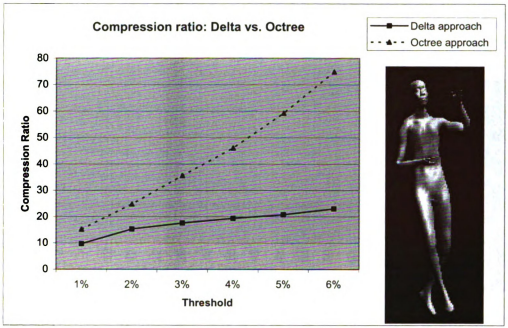


Figure 5.7: Compression ratio comparison between the delta and octree approach using the Dance animation.

5.2.3 Frame-wise compression ratio comparison: delta vs. octree

The overall compression ratio comparison results show that the delta coding method does not achieve the consistent level of performance demonstrated by the octree approach. However, in limited cases the delta approach does outperform. The frame-wise compression ratio comparisons were conducted. Figure 5.8 shows the frame-wise compression ratio comparison between the delta and octree approach given the same threshold settings for the Chef animation. In this case, the delta approach is better than the octree approach for 18% frames. Figure 5.9 shows the frame-wise comparison results for the Chicken Crossing animation. For 22% frames, the delta approach can achieve higher compression ratios given the same threshold settings. Figure 5.10 shows the comparison results for the Dance animation. The delta approach is better than the octree approach for 9% frames.

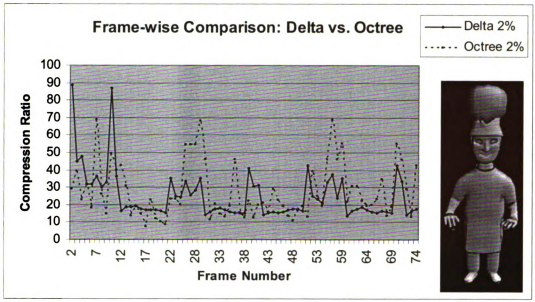


Figure 5.8: Frame-wise compression ratio comparison between the delta and octree approach using the Chef animation given the same threshold settings. (threshold=2%).

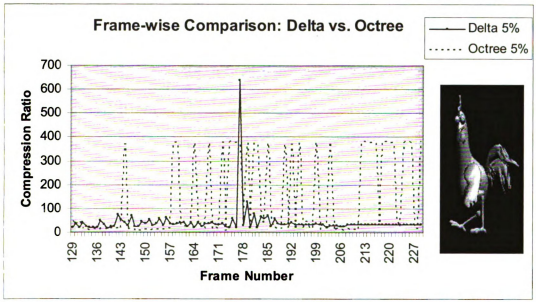


Figure 5.9: Frame-wise compression ratio comparison between the delta and octree approach using the Chicken animation given the same threshold settings. (threshold=5%).

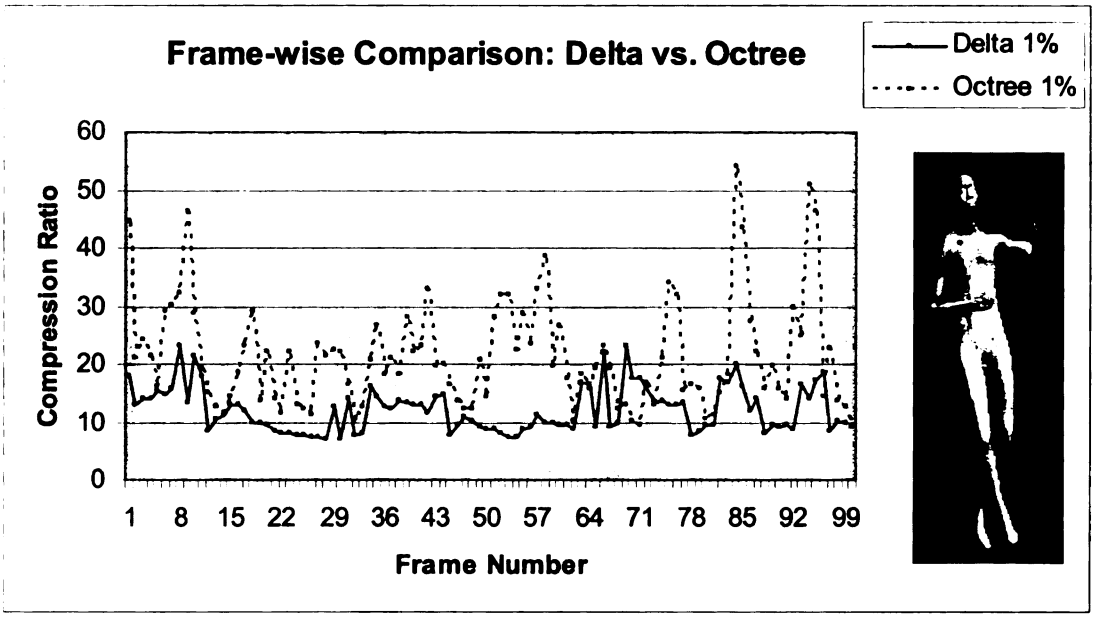


Figure 5.10: Frame-wise compression ratio comparison between the delta and octree approach using the Dance animation given the same threshold settings. (threshold=1%).

5.3 Conclusions

This chapter presented the delta approach to compress animated geometric data. This approach is straight forward. It is easy to implement. But it has its limitation. The big problem is that it can not achieve consistently high compression ratios. From the compression ratio comparisons with the octree approach, the compression ratio curves for the delta approach stop growing after they achieve certain level. Obviously the delta approach could not be used alone to achieve animated geometry compression.

However, frame-wise compression ratio comparison results between the delta approach and octree approach showed that the delta approach could achieve higher compression ratios for some frames of the animation sequence. Although the delta approach does not have good performance when it is used alone, it may increase the performance of the octree approach if they are combined. This is not the case of the PCA method. Although it also exhibits a great deal of variability among frames, the PCA

method must work over an entire sequence and cannot be efficiently applied to a single differential frame reconstruction. The delta approach, like the octree approach, is a pair-wise approach and can be applied to frames in isolation. In the following chapter, a hybrid approach is presented based on the combination of these two mechanisms.

Chapter 6

6 Hybrid approach

This chapter presents a *hybrid coding* approach to compress animated geometric data by combining the octree and delta coding approaches. The motivation of this approach comes from the observation that the encoded octree size is sometimes larger than the encoded size of the delta encoding. Section 6.1 presents the encoding process of the hybrid approach and Section 6.2 presents the results of this approach and the comparison among the octree approach, delta approach and hybrid approach.

6.1 Hybrid encoding process

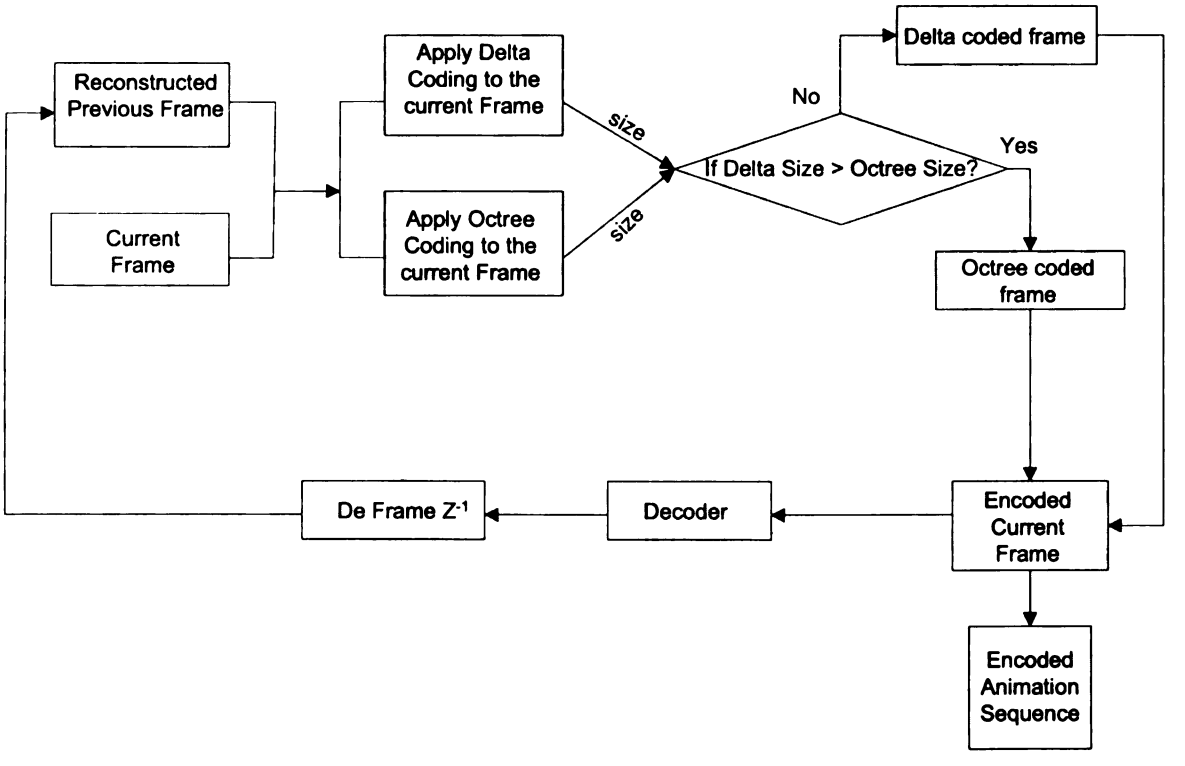


Figure 6.1: Logical flow of the hybrid encoding process for compressing 3D animation sequence.

To overcome the shortcomings of the octree approach, the hybrid approach is presented in this chapter. The logical flow of the encoding process using the hybrid approach is shown in Figure 6.1.

This approach uses a threshold to control the quality of the output. The threshold is used to control the subdividing process of the octree coding and the number of bits used in the quantization step of the delta coding. For the octree encoding process, if the error (Euclidean distance) of any reconstructed vertex is larger than the threshold, it will subdivide the current bounding box and repeat the motion vector computation step. For the delta encoding process, if the error of any reconstructed vertex is larger than the threshold, it will continue search for the best quantization level that drops the reconstruction error below the threshold. Therefore, any reconstructed vertex in any frame in the reconstructed animation sequence satisfies the criteria that the error between this vertex and the corresponding original vertex is smaller than the threshold as shown in Equation 4.1. The threshold in the hybrid coding method is set to be the maximum allowed distance between the reconstructed vertex and its corresponding original vertex. It is represented by the percentage of the edge length of the initial cubic bounding box of the object as illustrated in the Equation 4.2.

To encode each frame except the first frame in the sequence, both the octree coding method and delta coding method are applied in parallel. If the size of the encoded delta is smaller than the encoded octree, the encoded octree is replaced with the encoded delta. A single bit in the output data stream is used to indicate if it is an octree coding or delta coding.

6.2 Evaluations

To evaluate the performance of the hybrid approach, three different animation datasets: "Chef", "Chicken Crossing" and "Dance" are used in the evaluation as shown in Figure 4.4, Figure 4.5 and Figure 4.6 respectively. This section presents the compression ratio results of the hybrid approach, the reconstruction given different threshold settings and L^2 distance results. It also includes the compression ratio comparison among the hybrid, octree and delta approach given the same threshold settings.

6.2.1 Hybrid approach results

The hybrid method generates a mixed sequence of the octree encoded and delta encoded frames. Table 6.1 shows the percentage of the octree encoded and delta encoded frames for different animation sequences and different thresholds. The delta encoding contributes significantly to the hybrid method. It takes up to 41% of the encoded frames. However, as the threshold increases, the percentage of the delta coded frames in the encoded sequence decreases. This is because the performance of the octree approach is much better than the delta coding approach as the threshold increases.

Table 6.1: Percentage of the octree encoded and delta encoded frames in the hybrid method.

Hybrid	Threshold	Octree	Delta
Chef	1%	59%	41%
	2%	82%	18%
	3%	86%	14%
	4%	90%	10%
Chicken	2.5%	64%	36%
	5%	78%	22%
	7.5%	85%	15%
	10%	88%	12%
Dance	1%	91%	9%
	2%	92%	8%
	3%	95%	5%
	4%	97%	3%

This section also presents the compression ratio results for the hybrid approach. Table 6.2, 6.3 and 6.4 present the compression ratios of the Chef, Chicken Crossing and Dance animation sequence using the hybrid approach. Some reconstructed sample frames of the Chef, Chicken Crossing and Dance animation sequence using the hybrid approach are shown in Figure 6.2, 6.3 and 6.4 respectively. L^2 distance comparison results among different threshold settings using the hybrid approach are presented in Figure 6.5, 6.6 and 6.7 respectively.

Table 6.2: Compression ratios of the Chef animation using the hybrid approach. The Chef animation sequence has 75 frames and each frame has 8162 triangles and 4241 vertices. The object size is 440 units in x dimension, 148 units in y dimension and 455 units in z dimension. Original file size is 3,816,900 bytes. Max Distance threshold =p% *455 units, where p=1, 2, 3, ...6.

Hybrid approach Chef animation	Max Distance					
	1%	2%	3%	4%	5%	6%
Compression Ratio	22:1	32:1	41:1	65:1	78:1	96:1

Table 6.3: Compression ratios of the Chicken Crossing animation using the hybrid approach. The Chicken Crossing animation has 400 frames and each frame has 5664 triangles and 3030 vertices. The object size is 2.556 units in x dimension, 2.23 units in y dimension and 1.07 units in z dimension. The original file size is 14,544,000 bytes. Max Distance threshold= p% *2.556 units, where p=2.5, 5, 7.5, 10, 15, 20.

Hybrid approach Chicken animation	Max Distance					
	2.5%	5%	7.5%	10%	15%	20%
Compression Ratio	29:1	47:1	60:1	77:1	112:1	153:1

Table 6.4: Compression ratios of the Dance animation using the hybrid approach. The Dance animation has 201 frames and each frame has 14118 triangles and 7061vertices. The object size is 0.0758 units in x dimension, 0.172 units in y dimension and 0.074 units in z dimension. Original file size is 17,031,132 bytes. Max Distance threshold=p%*0.172 units, where p=1, 2, 3, 4, 5, 6.

Hybrid Approach Dance animation	Max Distance					
	1%	2%	3%	4%	5%	6%
Compression Ratio	17:1	28:1	37:1	48:1	62:1	72:1

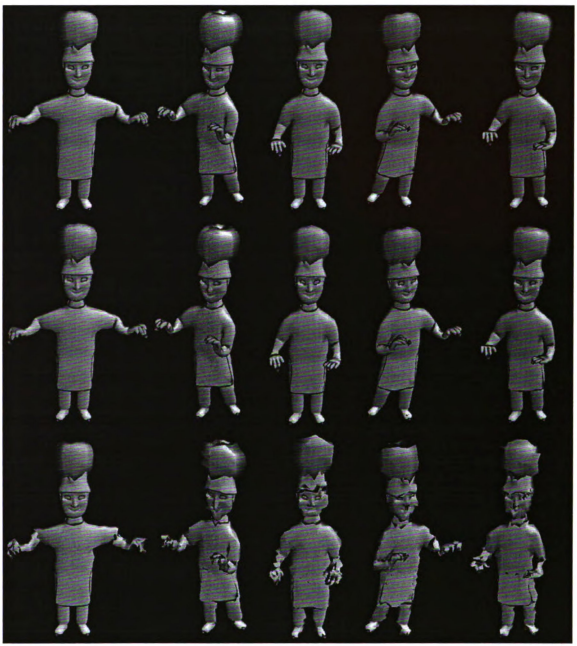


Figure 6.2: A selected set of reconstructed Chef animation using the hybrid approach. The top row is the original animation. The reconstructed animation by using *Max Distance threshold* = 1% and 5% are shown in row 2and 3 respectively. The compression ratio is 22:1 and 78:1 in row 2 and 3 respectively.

Figure 6.3: A selected set of reconstructed Chicken Crossing animation using the hybrid approach. The top row is the original one. The reconstructed animation by using *Max Distance threshold* = 5%, 10% and 15% are shown in row 2, 3, and 4 respectively. The compression ratio is 47:1, 77:1 and 112:1 in row 2, 3 and 4 respectively.

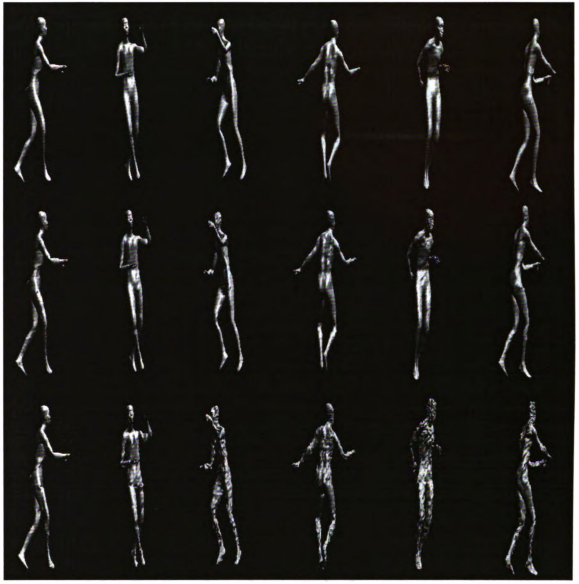


Figure 6.4: A selected set of reconstructed Dance animation using the hybrid approach. The top row is the original animation. The reconstructed animation by using max distance threshold = 1% and 3% are shown in row 2 and 3 respectively. The compression ratio is 17:1 and 37:1 in row 2 and 3 respectively.

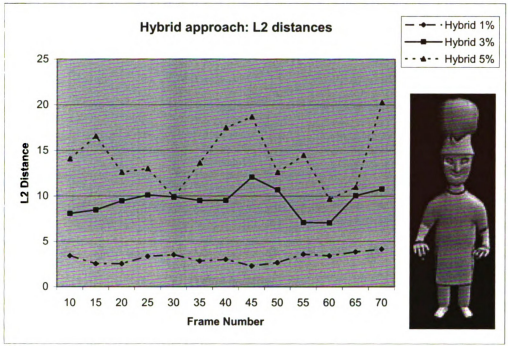


Figure 6.5: L^2 distances of some sample frames in the reconstructed Chef animation using the hybrid approach given different threshold settings.

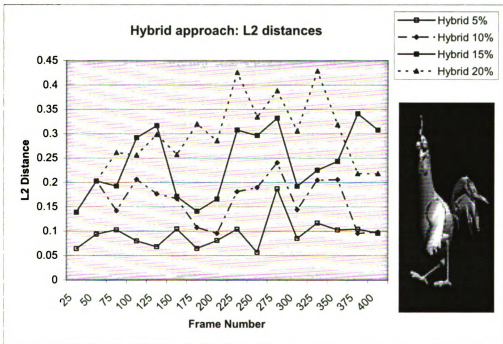


Figure 6.6: L^2 distances of some sample frames in the reconstructed Chicken animation using the hybrid approach given different threshold settings.

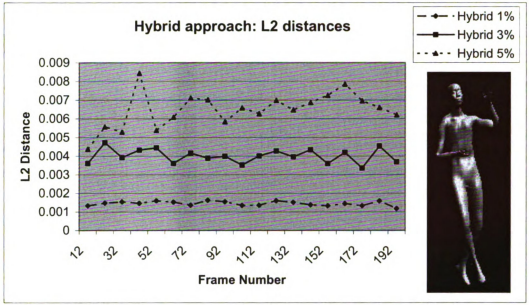


Figure 6.7: L^2 distances of some sample frames in the reconstructed Dance animation using the hybrid approach given different threshold settings.

6.2.2 Compression ratio comparison: hybrid, octree and delta

This section presents the compression ratio comparison results among the hybrid, octree and delta approaches. Figure 6.8 shows the results using the Chef animation. Similarly, Figure 6.9 and Figure 6.10 show the results using the Chicken Crossing and Dance animation respectively.

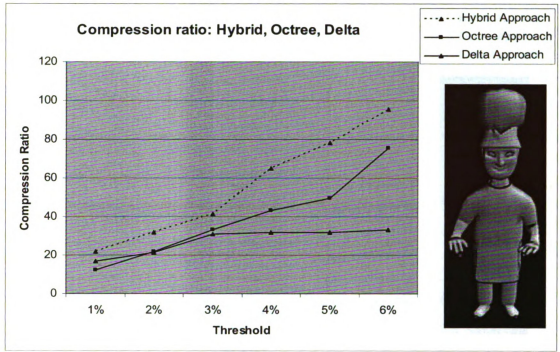


Figure 6.8: Compression ratio comparison among the hybrid, octree and delta approach using the Chef animation.

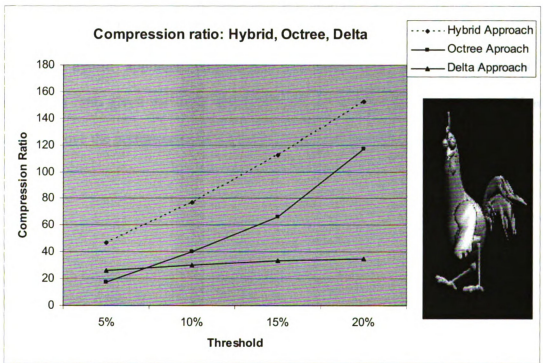


Figure 6.9: Compression ratio comparison among the hybrid, octree and delta approach using the Chicken animation.

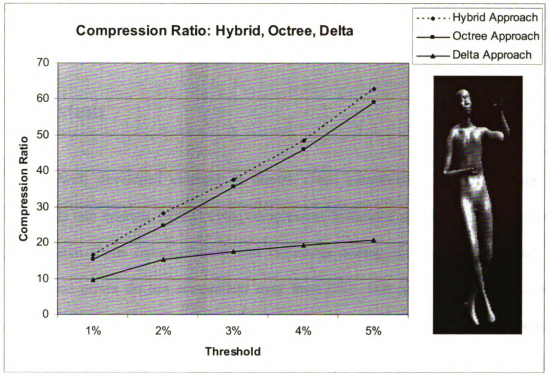


Figure 6.10: Compression ratio comparison among the hybrid, octree and delta approach using the Dance animation.

6.3 Conclusions

The hybrid approach outperforms the octree-only approach in terms of compression ratio given the same threshold settings. It provides a simple and efficient way to improve the performance of the octree-based motion representation method.

Chapter 7

7 System

A well developed concept in digital video compression is the layering of components into systems. MPEG video compression includes a structure for combination of differential frame representations and intraframe coding such that streams can be rejoined and errors do not accumulate over time [78]. This system, the well known combination of I, P, and B-frames allows MPEG to exploit the massive interframe correlations of digital video while still providing occasional points where the stream can be picked back up for decoding. In addition, the MPEG systems standard defines another layer of stream encoding that allows for the merging of a video stream with additional media streams such as audio [79].

In this chapter, a *systems-layer* is proposed for animated geometry compression that supports practical application of these methods in distributed systems including stream joining, random access in sequences, merging with additional multimedia streams, and playback controls. It describes the system implementation of a frame-based geometry system with particular emphasis on support for practical playback requirements including stream joining, playback controls, and error correction. The details of such a system are contrasted to the mature MPEG systems layer designs. A system block diagram is illustrated in Figure 7.1.

7.1 System design

Similar to MPEG, the complete hybrid compression system has I-frames and P-frames. I-frames (intra-coded frames) are coded independently using a static geometry compression method as discussed in Chapter 3. P-frames (octree or delta predicted frames) are inter-coded frames. P-frames are predicted from temporally preceding I-frames or P-frames in the animation sequence as described in Chapter 4, 5, and 6. P-frames are encoded using the hybrid coding method.

MPEG gains advantage from the recognition that motion is often continuous. Hence, the motion from one frame to another may be easily deduced by the location of content before and after the frame. The MPEG B-frames allow for interpolation of motion between preceding and succeeding video frames. Though this thesis does not explore the concept in detail, it is easy to propose the concept of a B-frame for animated geometry compression.

B-frames are predicted from the nearest preceding and following I or P-frames in the sequence. B-frames could be encoded with an interpolation method. As an example, a simple fraction can indicate the distance the vertices are expected to move along a linear or cubic path between the preceding and succeeding frame vertex locations. Then a differential of that motion (the error after the prediction) is simply encoded using the hybrid encoding method. This closely mirrors the concept of DCT compression of residuals in MPEG B-frames.

I-frames are required to support random access playback at the decoder side. Decompression must always begin with an I-frame for any sequence. I-frames also allow for stream joining, the acquisition in sequence of a broadcast stream, and recovery after

errors. It is common that broadcast multimedia streams utilize forward error correction (FER) to mask the effects of errors. If errors exceed the correction threshold, I-frames provide a later place to recover decoding following the error.

Increasing the quantity of I-frames will lower the compression ratio, since intra-coded frames are considerably larger than inter-coded frames. Having more inter-coded frames increases the compression ratio at the risk of reducing the reconstructed animation quality. The distribution of I, P, and B-frames in MPEG video is typically application dependent, allowing applications to choose the granularity of stream random access and the quality of the content for a given data rate. The same idea clearly applied to animated geometry compression.

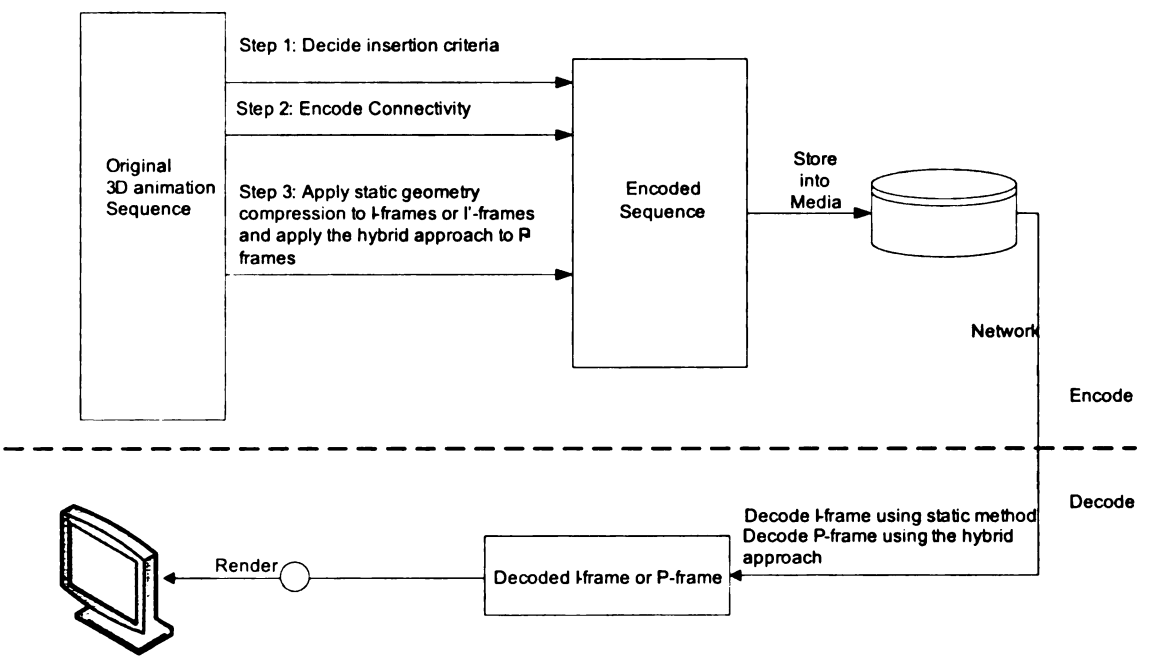


Figure 7.1: System Diagram.

Animated geometry compression methods actually induce two possible variants on the I-frame. The data for a given frame consists of vertex and connectivity

information. A complete intra-coded frame will include both elements, allowing for the stream to be joined in progress and all data for rendering decoded. However, in many cases the connectivity information will remain constant. This leads to a reduced I-frame version (the I'-frame) consisting only of vertex data. Such a frame does not support stream joining, but will allow error recovery if the geometry does not change during the stream loss and will support random access, since the geometric data can be stored in the file header or in relation of some grouping of frames.

A hybrid animated geometry compression system works as follows: given a 3D animated sequence, the user needs to specify some parameters in the beginning, namely I-frame, I'-frame, P-frame and B-frame interval or insertion criteria. Insertion criteria might include a maximum spacing and automatic insertion of I-frames following geometry changes (due to cuts or new object inclusion). Connectivity may be assumed to remain unchanged for the entire sequence, so it can be encoded and transmitted only once, though new connectivity can always be transmitted using an I-frame. The connectivity information is encoded with each I-frame where connectivity has changed or stream joining capabilities are to be supported. Other information like the number of vertices and number of frames is also stored with the I-frame.

According to those parameters, each frame can be encoded as an I-frame, I'-frame, P-frame or possibly a B-frame. Apply static geometry compression method for I-frames; apply the hybrid approach for the P-frames. After all the frames are encoded, the encoded I and P-frame sequence are stored into media. The encoded data could be transmitted through network to a decoder side. The decoder reads the stream data from the encoder side. It first reads the insertion criteria information to decide parameters such

as I-frame, P-frame and B-frame intervals. It also restores the connectivity information. It then retrieves an encoded I-frame or P-frame record and decodes it. An I-frame is decoded using static geometry compression method; a P-frame is decoded using the hybrid approach. After decoding a frame, it could be rendered on the screen or saved into a file. This process is repeated until all the frame records are retrieved from the network.

7.2 System features

As a complete hybrid compression system which is similar to the MPEG system [80], the system design described in this chapter can support several desired features for practical applications.

- Stream joining

  Stream joining feature allows joining the stream in progress when user switches the geometry channel. I-frames can be used to support this feature.

- Random access

  Random access requires that a compressed bit stream be accessible in its middle [80]. It is an essential feature that a frame based compression system should support. This feature can be achieved by providing access points within the sequence. Intra-coded frame can serve as access points for the random access. Any full system will need to support the transmission of intra-coded data in addition to the inter-coded data in order to support scene changes and new content entering the scene. I-frames are intra-coded. To support random access, the nearest I-frame to the requested access point is located and the decoder can directly jump to the record of the I-frame.

- Interleaved audio support

  It is possible to add audio support to the system. At the encoder, 3D animated data and audio are encoded separately and then interleaved using a system layer standard such as that defined for the MPEG standards. The decoder decodes audio and animation from the interleaved stream. Time Division Multiplexing can be used if the delay from the sender to the receiver is fixed. However, in a packetized system the delay is usually not fixed and the methods described in this thesis are fundamentally variable bit-rate mechanisms. Some means of synchronizing audio and animation packets are required. The solution used by MPEG-2 system can be adopted [47]. Decoding Time Stamps (DTS) indicates when a unit to be decoded; Presentation Time Stamps (PTS) indicates when a unit to be rendered, displayed or played. When the System Time Clock (STC) advances to the DTS, the unit is decoded. Likewise, when the STC advances to the PTS, the unit is presented. This relatively simple system has been applied to a variety of media types and compression methods.

- Error correction

  During the transmission of an encoded animation, errors may occur. The error may be propagated by differential hybrid decoding. However, I-frames will be able to correct the previous errors and restart the correct animation sequence. How soon the system can recover from an error depends on the frequency of I-frames.

- VCR controls

The hybrid compression system as described in this chapter can support VCR controls such as pause, fast forward. Fast forward is basically a more demanding form of random access. Several different levels of fast forward can be supported. The fastest level is done through decoding only I-frames and skipping all inter-coded frames. If B-frames are included, the next level is supported by decoding P-frames and skipping B-frames.

Chapter 8

8 Conclusions and Future work

This chapter presents some conclusions that can be drawn from this thesis and some possible future extensions to the current work. This thesis has presented three novel approaches to efficiently represent 3D animation sequences, an octree-based approach, delta coding approach and hybrid approach. These approaches have been compared and evaluated using the "Chef", "Chicken Crossing" and "Dance" animation sequences.

8.1 Conclusions

This thesis began with a review of the basic techniques for data compression, some notable efforts in geometry compression and animated geometry compression. It then described the implementation of a novel octree-based approach that efficiently represents the data coherence in a 3D animation sequence in detail. In order to evaluate this approach, PCA method was implemented and compared to the octree-based method in terms of compression ratio and L^2 distance. The results show that the octree-based approach is significantly better in L^2 distance given the same compression ratio. Compared to a purely static mechanism, the octree approach can achieve much higher compression ratios.

A delta coding mechanism was also introduced that represents the data coherence in a 3D animation sequence and is efficient as a compression mechanism on some pair-wise frame sequences. This approach was evaluated by comparing its performance to the

octree-based approach in terms of overall and frame-wise compression ratios. In some pair-wise frame cases it did out-perform the octree-based method. However, it is shown that it is not suitable as a general solution because the compression ratio does not scale with the number of vertices of a frame. Although the delta coding approach could not achieve good performance when it is used alone, it can increase the performance of the octree-based approach when they are combined.

A hybrid approach was then presented that combines the octree-based method and delta coding method. The results show that the hybrid approach outperforms both the octree-based and delta-encoding approaches in terms of compression ratio.

This thesis also described the system implementation of a frame-based geometry system with particular emphasis on support for practical playback requirements including stream joining, playback controls, and error correction. The details of such a system were contrasted to the mature MPEG systems layer designs.

8.2 Future work

This section presents some possible extensions to the current work.

- A better 3D visual quality measure

  Finding a good visual quality measure is a difficult task. In this thesis, L^2 distance was used as a distortion measurement. However, it is not an accurate visual quality measurement. For instance, there are many cases that one may have larger L^2 distance but have better visual quality. A better measure may need to take into account more factors, such as issues of the human visual system and characteristics of the rendered images. In addition to comparing the distortions of

individual frames in an animation sequence, there is also a need to measure the reconstructed motion quality. Good visual quality measurements may help the compression system adjust parameters to achieve improved ratio/distortion performance.

- Several variations on the octree design

  In order to increase the performance of the current design, the modification of the splitting criteria deserves more exploration. Other alternative ways to estimate and model motion for a cell may worth further exploration, such as a compact quaternion representation and cubic interpolations.

- Motion clustering

  The concept of clustering vertices so as to partition the data into groups with greater motion coherence is another area that deserves exploration. Compression methods take advantage of the motion coherence among vertices. This is clearly the case for vertices within an object, but may not be the case among independently moving objects. Clustering the data into groups that mimic independently moving objects may allow for better motion modeling within the groups, thereby increasing quality.

- Progressive compression

  To support a system that can provide for partial data transmission at reduced resolution, the octree-based implementation needs to be modified. One possible approach is to store the motion vectors not only at the leaf, but also at internal nodes of an octree. A child node constructs a motion vector based on the results of its parent's reconstructed frame. One research issue is how to traverse

the octree to provide a smooth and balanced increase of LOD for all parts of an object.

- Dynamic changes in topology

  Connectivity is assumed to remain constant in this thesis; future work needs to address dynamic changes in topology.

Animated geometry compression is a relatively new area, but one with great promise. Numerous applications will be enabled by practical and high quality solutions to the problem. This thesis presents methods that significantly improve on existing methods, particularly in the area of design simplicity, performance, and consistency from frame to frame and a preliminary description of how to integrate this work into a system is presented. It is not assumed that this is the final solution of the problem, but the general method is very promising as a framework for developing a solution with wide applicability.

References

[1] "Nvidia brings back SLI," http://www.gamespot.com/news/2004/06/28/news_6101476.html

[2] F. Sparacino, K. Hall, C. Wren, G. Davenport, and A. Pentland, "Improvisational Theater Space," *Proceedings of The Sixth Biennal Symposium for Arts and Technology*, New London, CT, 1997.

[3] N. Foster and R. Fedkiw, "Practical Animation of Liquids," *Proceedings of ACM SIGGRAPH'01*, 2001.

[4] N. Foster and D. Metaxas, "Modeling Water for Computer Animation," *Communications of the ACM*, vol. 43, no. 7, pp. 60-67, 2000.

[5] M. Deering, "Geometry Compression," *Proceedings of ACM SIGGRAPH'95*, pp. 13-20, 1995.

[6] G. Taubin and J. Rossignac, "Geometric compression through topological surgery," *ACM Transactions on Graphics*, vol. 17, no. 2, pp. 84-115, 1998.

[7] G. Taubin and J. Rossignac, "3D Geometry Compression," *ACM SIGGRPAH'98 Course Notes 21*, Orlando, Florida, 1998.

[8] C. Touma and C. Gotsman, "Triangle Mesh Compression.," *Proceedings of 24th Conference on Graphics Interface (GI-98)*, pp. 26-34, San Francisco, 1998.

[9] J. Rossignac, "Edgebreaker: Connectivity Compression for Triangle Meshes," *IEEE Transactions on Visualization and Computer Graphics*, vol. 5, no. 1, pp. 47-61, 1998.

[10] H. Hoppe, "Progressive meshes," *Proceedings of ACM SIGGRAPH'96*, vol. 30, pp. 99-108, New Orleans, Louisiana, 1996.

[11] R. Pajarola and J. Rossignac, "Compressed progressive meshes," *IEEE Transactions on Visualization and Computer Graphics*, vol. 6, no. 1, 2000.

[12] J. Li and C. C. Kuo, "Progressive coding of 3D graphic models," *Proceedings of IEEE Multimedia and Systems*, pp. 1052-1063, 1998.

[13] C. L. Bajaj, V. Pascucci, and G. Zhuang, " Progressive compression and transmission of Arbitrary triangle meshes," *Proceedings of IEEE Visualization '99*, pp. 67-72, 1999.

[14] D. Cohen-Or, D. Levin, and O. Remez, "Progressive compression of arbitrary triangular meshes," *Proceedings of IEEE Visualization'99*, pp. 67-72, Los Alamitos, California, 1999.

[15] G. Taubin, A. Gueziec, W. Horn, and F. Lazarus, "Progressive Forest Split Compression," *Proceedings of ACM SIGGRAPH'98*, pp. 123 - 132, 1998.

[16] P.-M. Gandoin and O. Devillers, "Progressive lossless compression of arbitrary simplicial complexes," *Proceedings of ACM SIGGRAPH'02*, pp. 372-379, Texas, 2002.

[17] A. Khodakovsky, P. Schröder, and W. Sweldens, "Progressive geometry compression," *Proceedings of ACM SIGGRAPH'00*, pp. 271- 278, 2000.

[18] J. Popovic and H. H. A. 1997, "Progressive simplicial complexes," *Proceedings of ACM SIGGRAPH'97*, pp. 217-224, 1997.

[19] F. Bossen, "On The Art Of Compressing Three-Dimensional Polygonal Meshes And Their Associated Properties," Ph.D. Thesis, cole Polytechnique Fdrale de Lausanne (EPFL), 1999.

[20] D. Shikhare, "State of the Art in Geometry Compression," National Centre for Software Technology, 2000.

[21] D. Luebke, "A survey of polygonal simplification algorithms," Dept. Computer Science, University of North Carolina, Chapel Hill, Tech. Report TR97-045, 1997.

[22] J. E. Lengyel, "Compression of Time-Dependent Geometry," *Proceedings of ACM Symposium on Interactive 3D Graphics*, pp. 89 -95, New York, ACM Press, 1999.

[23] M. Alexa and W. Müller, "Representing Animations by Principal Components," *Computer Graphics Forum*, vol. 19, no. 3, pp. 411-418, 2000.

[24] L. Ibarria and J. Rossignac, "Dynapack:Space-Time Compression of the 3D animations of triangle meshes with fixed connectivity," *Proceedings of Eurographics/SIGGRAPH Symposium on Computer Animation*, 2003.

[25] H. M. Briceño, P. V. Sander, L. McMillan, S. Gortler, and H. Hoppe, "Geometry Videos: A new representation for 3D Animations," *Proceedings of Eurographics/SIGGRAPH Symposium on Computer Animation(SCA03)*, San Diego, California, 2003.

[26] Z. Karni and C. Gotsman, "Compression of soft-body animation sequences," *Computers & Graphics*, vol. 28, pp. 25-34, 2004.

[27] X. Gu, S. Gortler, and H. Hoppe, "Geometry images," *Proceedings of ACM SIGGRAPH '02*, pp. 355--361, 2002.

[28] J. Zhang and C. B. Owen, "Octree-based Animated Geometry Compression," *Proceedings of Data Compression Conference (DCC'04)*, pp. 508-517, Snow Bird, UT, 2004.

[29] J. Zhang and C. B. Owen, "Hybrid Coding for Animated Polygonal Meshes: Combining Delta and Octree," *Proceedings of IEEE International Conference on Information Technology (ITCC'05)*, Las Vegas, NV, 2005.

[30] D. Salomon, *Data Compression: The Complete Reference*, 2nd, 2000.

[31] K. Sayood, *Introduction to Data Compression*, 2nd, Morgan Kaufmann Publishers, 2000.

[32] T. C. Bell, J. G. Cleary, and L. H. Witten, *Text Compression*, Prentice Hall, 1990.

[33] A. Moffat and A. Turpin, *Compression and Coding Algorithms*, Kluwer Academic Publishers, 2002.

[34] C. E. Shannon, "A Mathematical Theory of Communication," *The Bell System Technical Journal*, vol. 27, pp. 379-423 and 623-656, 1948.

[35] J. Rissanen and G. G. Langdon, "Arithmetic coding," *IBM J. Res. Develop*, vol. 23, no. 2, pp. 149-162, 1979.

[36] R. N. Bracewell, *The Fourier Transform & Its Applications*, 3rd, McGraw-Hill Science/Engineering/Math, 1999.

[37] E. Brigham, *Fast Fourier Transform and Its Applications*, 1st, Prentice Hall, 1988.

[38] O. Riol and M. Vetterli, "Wavelets and signal processing," *IEEE Signal Processing Magazine*, vol. 8, no. 4, pp. 14-38, 1991.

[39] A. Graps, "An Introduction to Wavelets," *IEEE Computational Science & Engineering*, vol. 2, no. 2, pp. 50-61, June 1995.

[40] C. K. Chui, *An Introduction to Wavelets: Wavelet Analysis and its Applications*, Academic Press, 1992.

[41] R. Polikar, "The Wavelet Tutorial," http://users.rowan.edu/~polikar/WAVELETS/WTtutorial.html

[42] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*, Kluwer Academic Press, 1992.

[43] Y. Linde, A. Buzo, and R. M. Gray, "An Algorithm for Vector Quantization Design," *IEEE Transactions on Communications*, vol. 28, pp. 84-95, 1980.

[44] J. Kominek, "Introduction to Fractal compression," http://www.faqs.org/faqs/compression-faq/part2/section-8.html

[45] W. B. Pennebaker and J. L. Mitchell, *Jpeg: Still Image Data Compression Standard*, 1st edition, 1992.

[46] J. L. Mitchell, W. B. Pennebaker, C. E. Fogg, and D. J. LeGall, *MPEG Video compression standard*, Chapman & Hall, 1996.

[47] B. G. Haskell, A. Puri, and A. N. Netravali, *Digital video: an introduction to MPEG-2*, Kluwer Academic Publishers, 1996.

[48] Z. Karni and C. Gotsman, "Spectral compression of mesh geometry," *Proceedings of ACM SIGGRAPH'00*, pp. 279-286, 2000.

[49] S. Gumhold and W. Strasser, "Real time compression of triangle mesh connectivity," *Proceedings of ACM SIGGRAPH'98*, pp. 133-140, 1998.

[50] M. Isenburg and J. Snoeyink, "Face Fixer: Compressing Polygon Meshes With Properties," *Proceedings of ACM SIGGRAPH'00*, pp. 263-270, New Orleans, Louisiana, 2000.

[51] D. King, J. Rossignac, and A. Szymczak, "Connectivity compression irregular quadrilateral meshes," Georgia Tech, Tech. Rep. TR-99-36,GVU, 1999.

[52] O. Staadt, M. Gross, and R. Weber, "Multiresolution Compression and Reconstruction," *Proceedings of IEEE Visualization '97*, pp. 337-364, 1997.

[53] C. Zach and K. Karner, "Progressive Compression of Visibility Data for View-dependent Multiresolution Meshes," *Proceedings of WSCG'2003*, Czech Republic, 2003.

[54] H. Hoppe, "View-dependent refinement of progressive meshes," *Proceedings of ACM SIGGRAPH'97*, pp. 189--198, 1997.

[55] H. Hoppe, "Smooth view-dependent level-of-detail control and its application to terrain rendering," *Proceedings of IEEE Visualization'98*, pp. 35-42, 1998.

[56] D. Schmalstieg and G. Schaufler, "Smooth Levels of Detail," *Proceedings of IEEE Virtual Reality Annual International Symposium (VRAIS'97)*, pp. 12-19, New Mexico, 1997.

[57] T. Funkhouser, "Advanced Computer Graphics Lecture Notes: Mesh Representations," http://www.cs.princeton.edu/courses/archive/fall02/cs526/lectures/reps.pdf

[58] "The Stanford 3D Scanning Repository," http://graphics.stanford.edu/data/3Dscanrep/

[59] C. Gotsman, S. Gumhold, and L. Kobbelt, "Simplification and compression of 3D meshes," *Proceedings of the European Summer School on Principles of Multiresolution in Geometric Modelling (PRIMUS)*, Munich, 2001.

[60] A. Guiziec, F. Bossen, G. Taubin, and C. Silva, "Efficient Compression of Non-manifold Polygonal Meshes," *Proceedings of IEEE Visualization'99*, 1999.

[61] R. Bar-Yehuda and C. Gotsman, "Time/space tradeoffs for polygon mesh rendering," *ACM Transactions on Graphics*, vol. 15, no. 2, pp. 141-152, 1996.

[62] J. Neider, T. Davis, and M. Woo, *OpenGL programming Guide*, Addison-Wesley, 1993.

[63] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Mesh optimization," *Proceedings of ACM SIGGRAPH'93*, pp. 19-26, 1993.

[64] M. Davis and M. Tuceryan, "Coding of Facial Image Sequences by Model-Based Optical Flow," *Proceedings of International Workshop on Synthetic-Natural Hybrid Coding and Three Dimensional Imaging (IWSNHC3DI'97)*, Greece, 1997.

[65] H. Tao, T. S. Huang, H. H. Chen, and T.-P. J. Shen, "Data compression for animated three dimensional objects," Rockwell Science Center,Inc., 1998.

[66] D. Cohen-Or, Y. Mann, and S. Fleishman, "Deep Compression for Streaming Texture Intensive Animations," *Proceedings of ACM SIGGRAPH'99*, 1999.

[67] A. Fournier and D. Y. Montuno, "Triangulating simple polygons and equivalent problems," *ACM Trans. on Graphics*, vol. 3, pp. 153-174, 1984.

[68] R. Seidel, "A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons," *Computational Geometry: Theory and Applications*, vol. 1, no. 1, pp. 51-64, 1991.

[69] A. Glassner, T. McClure, S. Benza, and M. V. Langeveld, "Chicken Crossing," *SIGGRAPH Video Review*, 1996.

[70] C. L. Jackins and S. L. Tanimoto, "Oct-tree and their use in representing three-dimensional objects," *Proceedings of Computer Graphics and Image Processing*, vol. 14, pp. 249-270, 1980.

[71] N. Ahuja and C. Nash, "Octree Representations of moving objects," *Proceedings of Computer Vision, Graphics and Image Processing*, vol. 26, pp. 207-216, 1984.

[72] " 3D Trilinear Interpolation," http://www.siggraph.org/education/materials/HyperVis/vised/VisTech/VisProcess/VisInterp/interp5.html

[73] R. Pozo, "Template Numerical Toolkit," http://math.nist.gov/tnt/index.html

[74] I. Witten, R. Neal, and J. Cleary, " Arithmetic Coding for data compression," *Communications of the ACM*, vol. 30, no. 6, pp. 520-540, 1987.

[75] A. Moffat, R. M. Neal, and I. H. Witten, "Arithmetic Coding Revisited," *ACM Transactions on Information Systems*, vol. 16, no. 3, pp. 256-294, 1998.

[76] "GNU Triangulated Surface Library v. 0.7.2," http://gts.sourceforge.net/

[77] P. Cignoni, C. Rocchini, and R. Scopigno, "Metro: Measuring Error on Simplified Surfaces," *Computer Graphics Forum*, vol. 17, no. 2, pp. 167-174, 1998.

[78] "Coding of moving pictures and associated audio for digital storage media at up to about 1.5Mbit/s-Part2: Video," ISO/IEC 11172-2, 1993.

[79] "Coding of moving pictures and associated audio for digital storage media at up to about 1.5Mbit/s-Part1: Systems," ISO/IEC 11172-1, 1993.

[80] D. L. Gall, "MPEG: A Video Compression Standard for Multimedia Applications.," *Communications of the ACM*, vol. 34, no. 4, pp. 46-58, 1991.