COMPRESSING MESH GEOMETRY USING SPECTRAL METHODS AND A SET PARTITIONING ALGORITHM

by

Umut Konur

B.S., Computer Engineering, Boğaziçi University, 2003

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering
Boğaziçi University

2006

ACKNOWLEDGMENTS

I would like to thank my thesis supervisor Prof. Fikret S. Gürgen for his invaluable support of any sort during my thesis study. His support has shown as guidance, patience and motivation. The steps that I have been following since my undergraduate years are mainly his contribution.

I must specially thank Assoc. Prof. Uluğ Bayazıt for whatever I cannot list here. He has been the idea behind this project, he has carried out and implemented an important part of the work dictated by the project framework, feeded me with the resources to benefit from, worried with me for any step to take and shown me the way to follow at all and all instants.

I would also like to thank Assist. Prof. Tunga Güngör for having participated in my thesis jury and providing precious criticisms and feedback.

Another piece of thanks is for my friend Özgür Orcay who took part in a piece of the work achieved so far.

My greatest gratefulness and thanks go to my family without which this work and all past and future moments would not find chance to come true.

ABSTRACT

COMPRESSING MESH GEOMETRY USING SPECTRAL METHODS AND A SET PARTITIONING ALGORITHM

The demand for visualizing 3D objects has been growing rapidly in recent years due to the use of 3D models in several contexts and an increasing number of applications such as computer aided design, manufacturing, defense, architecture, entertainment, etc. Accessing these models over networks is required in many cases and hence developing efficient encoding schemes has been an interesting research topic for a decade or two. VRML (Virtual Reality Modeling Language) and MPEG-4 (Motion Pictures Experts Group-4) are two multimedia standards developed for coding and displaying polygonal meshes.

Coding 3D object data can take place in different domains and be classified in different ways. What primarily concerns us in this study is the progressive nature of the developed scheme. That is, while some applications may do with one instance of an encoded model and reconstruction performed on this single instance, other applications may require intermediate stages of a model that is reconstructed from coarse to fine in time. The latter case amounts to progressivity where a model can be viewed with higher and higher precision.

In this work, we develop a progressive mesh geometry coder that operates on a frequently used family of polygonal meshes (i.e. triangle meshes). The tool operates on spectral coefficients obtained by the method of Karni and Gotsman and bit-plane codes these coefficients using ideas borrowed from a set partitioning sorting algorithm called SPECK (Set Partitioning Embedded Block Coder) formerly used for image coding purposes. The output code is truly embedded and our method has shown to produce results superior to those of the spectral method.

ÖZET

SPEKTRAL YÖNTEMLER VE BİR KÜME BÖLÜNTÜLEME ALGORİTMASIYLA 3B NESNE BİLGİLERİNİN SIKIŞTIRILMASI

3B modellerin bilgisayar destekli tasarım, üretim, savunma, mimari, eğlence, vs. gibi alanlarda kullanımının artmasıyla birlikte 3B nesne bilgilerinin gösterimi ve görselleştirilmesi gereksinimi giderek artmıştır. Birçok uygulamada, model bilgilerine iletişim ağları üzerinden erişilme zorunluluğu, verimli kodlama yöntemleri geliştirmeyi son 20 yıllık dönemde ilgi çekici bir araştırma konusu yapmıştır. VRML (Virtual Reality Modeling Language) ve MPEG-4 (Motion Pictures Experts Group-4) 3B nesne bilgilerinin kodlanması ve görüntülenmesi için geliştirilmiş standartlara örnek verilebilir.

3B nesne bilgilerini kodlamak farklı çerçevelerde ele alınabilir ve sınıflandırılabilir. Bu çalışmada bizi esas olarak ilgilendiren, geliştirilen kodlayıcının ilerleyici bir yapıya sahip olmasıdır. Kimi uygulamaların tek bir kodlanmış modelle çalışarak geri çatmayı bu model üzerinde başarması yeterliyken, kimi uygulamalarda modelin ara aşamalarının görülmesi gerekebilir. Bu durumda ilerleyicilik, yani modelin kabadan iyiye doğru geri çatılması ve zaman ilerledikçe artan kesinlikte görselleştirilmesi gerekir.

Bu çalışmada, çokgen ağ yapılarının sıkça kullanılan bir alt kümesi olan üçgen ağ yapısındaki nesneleri ilerleyici biçimde kodlayan bir araç geliştiriliyor. Kodlama aracı Karni ve Gotsman'ın önerdiği yöntemle elde edilen spektral katsayıları daha öncesinde imge kodlama amacıyla kullanılmış olan ve küme bölüntüleme yaklaşımıyla çalışan bir sıralama algoritması olan SPECK (Küme Bölüntüleyen Gömülü Blok Kodlayıcı) fikirlerini kullanarak bit-düzlemsel kodlamaktadır. Elde edilen bit katarı tamamen gömülü olup spektral yöntemden daha başarılı sonuçlar ürettiği gözlenmiştir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS/ABBREVIATIONS

A	Adjacency matrix
D	Diagonal matrix
L	Laplacian operator
I	Identity matrix
N	Number of mesh points
R^n	n-dimensional vector space
n	Quantization level (threshold level) or number of mesh points
$\prod_n(T)$	Significance function of a set of coefficients T at quantization level n
c_i	Spectral coefficient at location i
$GL(v_i)$	Geometric Laplacian at vertex v_i
I	A set of type I
S	A set of type S

CAD	Computer Aided Design
CSPECK	Color Set Partitioning Embedded Block Coder
DCT	Discrete Cosine Transform
EZW	Embedded Zerotrees of Wavelet Coefficients
JPEG	Joint Pictures Experts Group
LIP	List of Insignificant Pixels
LIS	List of Insignificant Sets
LSC	List of Significant Coefficients
LSP	List of Significant Pixels
MPEG-4	Motion Pictures Experts Group-4
MSE	Mean Square Error
SPECK	Set Partitioning Embedded Block Coder
SPIHT	Set Partitioning in Hierarchical Trees
VRML	Virtual Reality Modeling Language

1. INTRODUCTION

1.1. Motivation

By the last decade or two, representing 3D object models and transmitting them over communication channels to end users in the framework of certain application areas have attracted considerable attention and thus been the focus of many research efforts.

The research has specifically appeared in the area of compression of these 3D models due to the enormous size of certain models and the need for efficient use of transmission bandwidth. Our goal in this study is to provide a progressive object geometry coder for such models.

It is highly desirable for a progressive coder to possess the property of embeddedness. The term stands for each new bit arriving at the decoder contributing to a piece of information that is more significant (at least as significant as) than that of the bits that have not arrived yet but less significant (at most as significant as) than that of the bits that have already arrived. This means that the leading bits in an embedded bitstream carry more information. The decoder end of a progressive codec is expected to produce reconstruction results that appear finer and finer with each new bit. This is very natural because one expects to obtain higher quality of reconstruction as time progresses and to observe the greatest possible distortion reduction at any point in time. In addition, it is worth noting that the encoder part of a codec that works with embedded bitstreams can stop encoding at any time and reconstruction can be performed with the available bits. Equivalently, the decoder part can achieve reconstruction with the leading portion of a transmitted bitstream and ignore the rest.

We propose a mesh geometry coder that outputs truly embedded bitstreams. Our scheme utilizes the spectral method of [1] and bit-plane codes the coefficients obtained by this method. The SPECK algorithm [2] that we have adopted for our purpose employs set

partitioning and achieves bit-plane coding with right bit priorities. A block diagram of the geometry coder we have designed and implemented is shown in Figure 1.1. The partitions do not refer to the aforementioned set partitioning concept but to partitioning meshes to be coded into smaller submeshes. The distinction will be clear in the following sections.

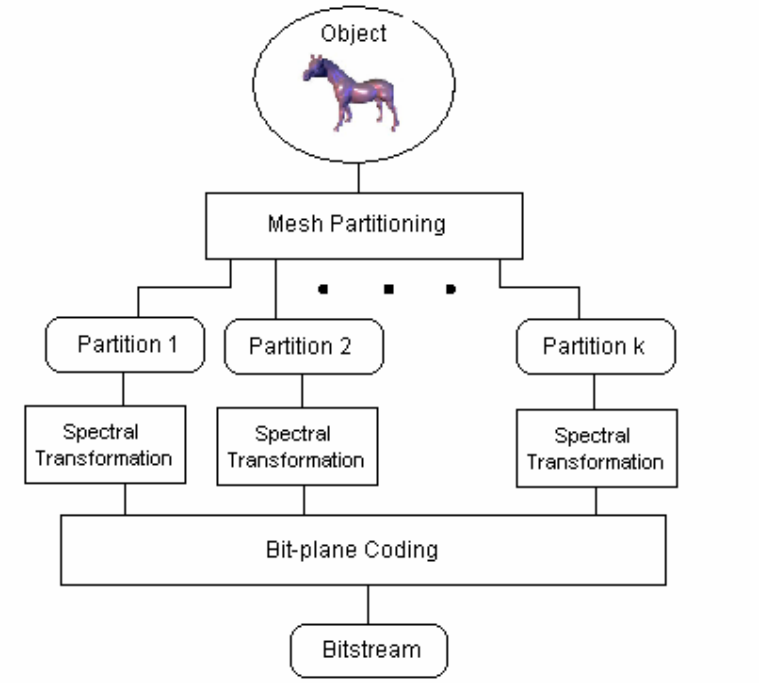


Figure 1.1. The block diagram of the geometry coder

1.2. Thesis Structure

The rest of the thesis document is organized as follows: Section 2 introduces the mesh compression problem. It starts with a discussion of 3D models pointing out where they are used and how they are represented as meshes. It then gives an overview of the standards used for mesh representation and coding them. Mesh compression schemes are classified depending on certain criteria and the overall mesh compression problem is decomposed into subproblems each one being explained. Section 3 presents a brief survey on mesh compression schemes proposed so far. Section 4 describes the spectral method that we have utilized and Section 5 is on the SPECK algorithm that has been used for bit-plane coding the spectral coefficients along with its frontiers and the bit-plane coding

concept. The application of SPECK to both image coding and mesh geometry compression is described with more detail and emphasis shared for the latter case. Section 6 discusses an error metric, called visual metric that has been used in [1] to measure the performance of mesh geometry coders. Section 7 presents the experimental results on two test models (the *horse* and *bunny* models) using both the geometry coder that we propose and the spectral coder that has already been proposed in [1] with a comparative viewpoint. The results that we report are those that have been obtained after running both schemes and applying entropy coding to the results. Section 8 concludes the thesis study providing a discussion on the rate-distortion performance and the bit-allocation policy of our geometry coder as well as making remarks on the expected future work and what has to be done in order to bring the proposed coder to a state that can serve useful at an application level.

2. MESH COMPRESSION

The mesh compression problem, as it name implies, is the problem of reducing the size of 3D object information represented and displayed via various tools. The problem is worth exploring and providing solutions, because 3D models employed in most applications have tremendous size and this makes the storage and transmission of these models quite infeasible.

2.1. 3D Models in General

2.1.1. Uses and Requirements

Representing and visualizing 3D object information has been a necessity for a few decades since the application areas in which such information is required have shown a great increase since then. Specific examples of these areas include manufacturing industry, entertainment, military industry, computer aided design (CAD) and architecture.

Representing 3D model information requires a way that describes how this information is extracted from a data representation. Since in many cases this data representation is of huge size and causes inefficient utilization of storage devices, compressing 3D model information is an action that has to be followed strictly.

The most important of all, compression of 3D models is critical for efficient use of transmission bandwidth of communication channels to reduce transmission time and not to annoy frustrated end users who deal with online applications where these models are used.

Visualizing 3D objects is done by rendering software, which operates on uncompressed data. Compressing the 3D object data at one end and decompressing it at the other end requires a coder-decoder (codec) system where the coder part resides in one

computer and the decoder part resides in another remote computer which is connected to the former host by a network.

2.1.2. The Mesh Representation

A 3D object is usually defined through its surface description. 3D objects can be represented in various ways. Bézier patches [3] and B-splines are examples of parametric representations of surfaces.

Another most widely used method of representing 3D models is the *mesh representation*. A mesh is a graphical representation of an object where the 3D points on the object surface correspond to the nodes of this graph and the incidence relations between the points are the edges of the graph. If a node is connected to another via an edge, there exists an edge between the two points (vertices) in space. That is, they are connected.

The mesh representation describes an object specifying its surface(s) in terms of 3D point coordinates and all of the incidence relations between pairs of these points. The incidence relations form polygons (also called faces) in space and the coordinates of each point is a 3-vector. Figure 2.1 shows an example of a mesh. It is obvious that for most objects, the mesh representation discretizes and approximates the true (smooth) surface information.

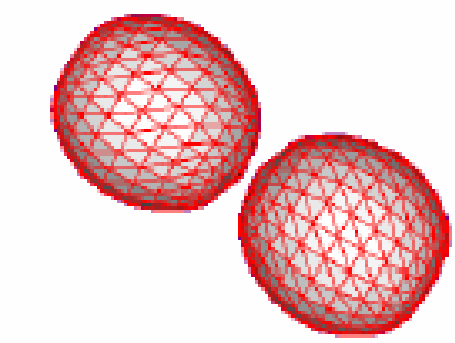


Figure 2.1. A mesh with two connected components

As far as considered, meshes have been defined in terms of point coordinates and the connectivity relations between points. Mesh definition is not constrained to these two features. Since meshes are models of 3D objects, properties of objects must also be contained in a mesh specification. Examples of properties of objects are texture values and material attributes.

2.2. Standards

Recent multimedia standards such as MPEG-4 (Motion Pictures Experts Group-4) and VRML (Virtual Reality Modeling Language) have embodied polygonal mesh coding and representation methods. Representing an object using the VRML standard is by listing the 3D coordinates of the points of the object mesh and the polygons that these points construct. Figure 2.2 shows the VRML representation of a simple tetrahedron (Note that there are four polygons and all are triangles).

```
#VRML V2.0 utf8
Shape {
  geometry IndexedFaceSet {
    coord Coordinate {
      point [
         0.00 -0.00  2.00
        -1.63 -0.94 -0.66
         0.00  1.88 -0.66
         1.63 -0.94 -0.66
      ] }
    coordIndex [
      0 1 3 -1
      3 1 2 -1
      3 2 0 -1
      0 2 1 -1
    ] } }
```

Figure 2.2. The VRML representation of a tetrahedron object

The encoder that we have designed and implemented expects the input 3D object data to be represented in VRML format.

2.3. Classification of Mesh Compression Schemes

2.3.1. Recovery-based Classification

Recovery-based classification of mesh compression algorithms is two-fold: *lossless* and *lossy*.

If there is no loss of information that the encoder introduces to the coded bitstream, the compression scheme is said to be *lossless*. On the other hand; if the encoder outputs a bitstream that cannot support reconstruction of the original model at the decoder site, the scheme is said to be *lossy*.

Lossy compression schemes are used when slight loss can be tolerated. In reality, lossless compression is impossible due to the loss introduced by quantization employed by almost every coder. When one talks about a lossless compression algorithm, what he means is a nearly lossless one.

2.3.2. Resolution-based Classification

Mesh compression algorithms are classified as *single-resolution* and *multi-resolution* depending on the way they operate.

Single-resolution schemes encode the entirety of the model in full resolution mode and the decoder can reconstruct the model once it has access to the whole bitstream. Multi-resolution algorithms encode a crude model first and transmit it to the receiving party. Following this, enhancement layers are encoded and progressively transmitted. Users can start to navigate the crude model immediately and as new layers arrive, they will see finer reconstructions of the model.

The method that we have implemented resembles multi-resolution algorithms in that it is progressive and reconstructed models are finer as time flows. However, classifying it as an exact example of a multi-resolution method is not true, because the numbers of points

and faces do not change with time. They are fixed. This is not the case in other multi-resolution methods. What is performed at the decoder site of our coder is actually reconstructing the model with all its entities of information but retrieving better and better approximations.

2.4. Decomposing the Process into Pieces

Compressing a polygonal mesh is a process that can be divided into pieces: Coding the connectivity and geometry information are the two legs.

2.4.1. Connectivity Coding

Coding the connectivity information of meshes refers to expressing the topology of objects with a sequence of symbols. This topological information specifies the connectivity relations of object points. Stating that *vertex i* is connected to *vertex j* is a piece of connectivity information and graphs that these pieces of information construct reveal which edges exist and thus which polygons construct the topology of a mesh.

In many compression schemes, the connectivity information ensures orderly processing of vertex data and provides a ground for more efficient vertex coordinate compression [4, 5].

Connectivity coding is an area on which research has taken a long way and performance has approximated optimality. State-of-the-art connectivity coding methods can code the connectivity information of typical meshes at a rate of 1-3 *bpv* (*bits per vertex*).

2.4.2. Geometry Coding

Compressing 3D coordinates of mesh points is known as *geometry coding*. A usual representation for vertex coordinates is in the form of classical floating point numbers.

Quantization is a useful tool to reduce the precision of floating point numbers and almost all coders benefit from this tool.

Mesh coders frequently use their connectivity coding methods in co-operation with a prediction rule, quantize prediction errors and code symbols that denote these errors. Predicting positions of new vertices of connectivity coding traversal from positions of previously coded vertices increases the efficiency of the geometry code.

2.4.3. Other Considerations

Although mesh compression is generally considered to consist of coding connectivity and geometry information, [6] emphasizes that there exists a third type of information and builds its progressive geometry coder exploiting this fact. The third type of information is *parameter information*, which captures where the sample locations are within the surface while geometry information captures the geometry independent of the used sample locations.

3. A SURVEY ON COMPRESSION OF 3D MESHES

3.1. Single-resolution Schemes

Mesh compression efforts have mostly dealt with triangle meshes up to date, because the *triangle* is the basic geometric primitive for graphics rendering hardware and for many simulation algorithms.

A worst-case bound of 4 *bpv* for the connectivity code is given by the popular *Edgebreaker* method [7]. [8, 9, 10] are other methods developed on top of Edgebreaker. Other approaches code the connectivity of triangle meshes transforming it into a sequence of valence codes [4, 11]. The average valence of six of regular triangle mesh vertices provides an advantage during entropy coding of this sequence because of low dispersion around the average. [5, 12] generalize the approach of [4] to arbitrary polygonal meshes relying on the key concept of *duality* and following the *degree/valence approach*. Figure 3.1 illustrates the duality concept. The vertex valences in the original mesh become the face degrees in the dual mesh and vice versa.

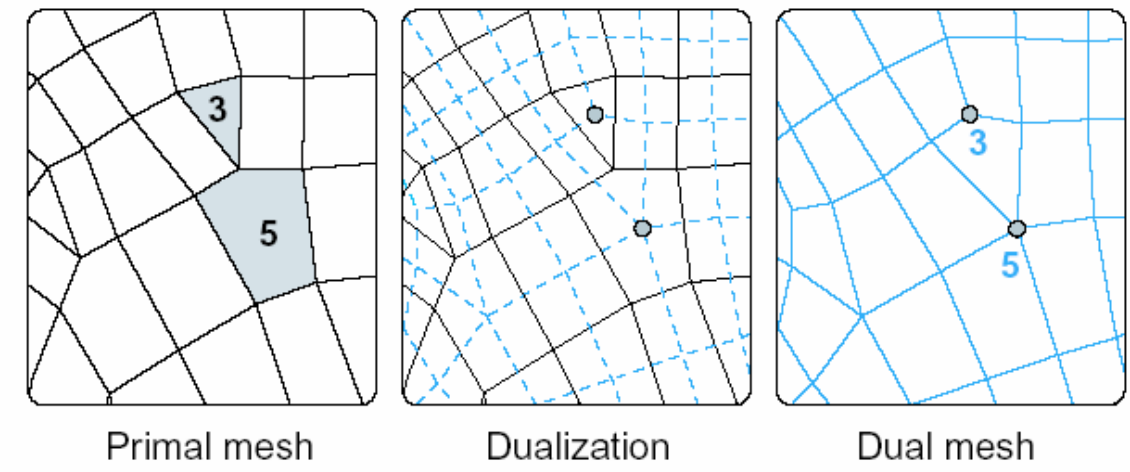


Figure 3.1. Mesh and its dual

Geometry compression of mesh data involves quantization that causes a reduction of precision in 3D coordinate data. The symbols obtained after quantization are then entropy-coded following the application of a prediction rule relying on smoothness assumptions.

Each vertex position can be quantized uniformly in Euclidian space [4, 13, 14]. Vector quantization approaches [15, 16] have also been proposed. [1] shows that quantization is possible in the space of spectral coefficients.

The choices for a prediction rule may involve linear coding [4, 14], or a more sophisticated parallelogram rule for triangle [4] and polygonal [17] meshes.

Some geometry coders [18, 19] rely on the approach of remeshing the original mesh to transform it to a more regular mesh expected as input. Remeshing stands for modifying topology information prior to coding.

3.2. Progressive Schemes

The concept of refinement is the general notion employed by progressive mesh coders. Figure 3.2 shows the nature of a progressive coder and intermediary stages during decoding.

In a progressive scheme, the coded refinement bits are decoded and the implied changes are applied to a coarser model to turn it into a finer one. As time flows, the distortion in the reconstructed object decreases. The challenge is to achieve this goal as efficiently as possible and obtain the best rate-distortion performance.

The method of [20] expresses a triangle mesh as a stream of refinements. During encoding, the mesh is treated with edge collapses and the operations are coded. The decoder solves the coded symbols and the changes are undone employing vertex splits. In [21], groups of edge collapses are kept as independent sets each corresponding to a level of detail. [22] defines a different methodology to locate independent sets of vertices to

decimate. [23] improves previous approaches by generating an alternation of independent sets.

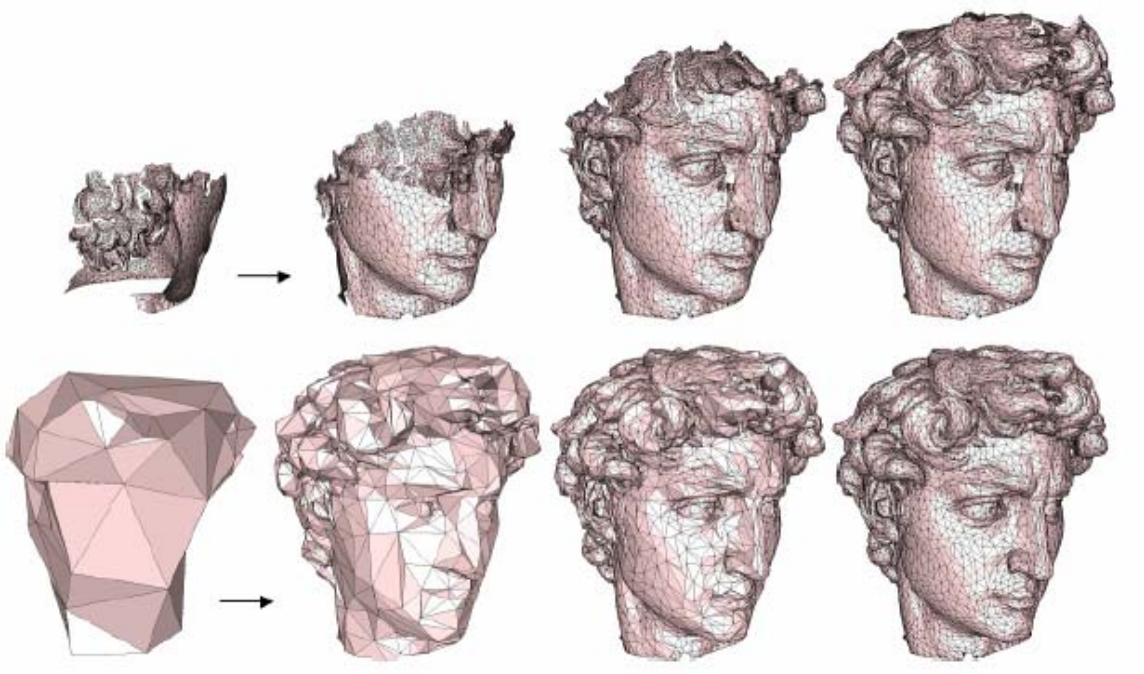


Figure 3.2. The operation of a progressive coder

4. SPECTRAL METHOS

Spectral methods are frequently used to compress media information such as images. In these methods, transform coefficients are computed through a transform technique and the data can be expressed as a linear combination of orthonormal basis functions, where each of these basis functions is characterized by one specific transform coefficient. Each basis function (and hence each coefficient) corresponds to information that is of a certain frequency. The JPEG (Joint Pictures Experts Group) method that relies on Discrete Cosine Transform (DCT) is a spectral image compression algorithm. Spectral methods form a useful ground for implementing progressive coders. This is because of the fact that each transform coefficient contributes to some information content. As more and more coefficients are employed in the reconstruction process, a better approximation is obtained.

If the progressivity of a spectral coder is to be achieved via using more and more coefficients in time, a natural question that arises is related to which coefficients must have the priority to be coded and transmitted. The answer is giving highest priority to the coefficient(s) that will result in maximum distortion reduction at all instants. The coefficients that contribute more to the information, thus are more significant and thus are expected to produce the most rate-distortion gain are the *low frequency* (or equivalently *high energy*) coefficients. Those coefficients that are inversely defined carry detail information and are known as *high frequency* (or equivalently *low energy*) coefficients. A progressive coder with the right policy must give the correct bit priority to the bits of coefficients.

4.1. Mesh Spectra

The classical Fourier analysis can be extended to general graph topologies and to 3D mesh data.

For a 2D signal (graph), the Fourier basis functions are the eigenvectors of the so-called Laplacian matrix of the 2D graph. The Laplacian operator L is computed using the adjacency relation matrix A describing the graph. A is a square matrix of size nxn, where n is the number of nodes of the graph. If node i and node j are adjacent nodes, A_{ij} is 1, 0 otherwise and

$$L = I - \frac{1}{4} A \tag{4.1}$$

where I is the nxn identity matrix.

Extending the theory further, as in [24], we have the following formalism for the spectra of a 3D mesh:

The adjacency matrix defined by the topology of an n-point mesh is

$$A_{ij} = \begin{cases} 1 & \text{if vertex i and vertex j are adjacent} \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

Let D be an nxn diagonal matrix such that $D_{ii} = 1/d_i$, where d_i is the degree of the i^{th} vertex. Then the Laplacian operator for the mesh is

$$L = I - DA \tag{4.3}$$

and the elements of L become

$$L_{ij} = \begin{cases} 1 & \text{if i = j} \\ -\frac{1}{d_i} & \text{if i and j are neighbors} \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

L_{ii} is set to 0 if vertex i has no neighbors.

An orthogonal basis for R^n is given by the eigenvectors of L and the associated eigenvalues can be considered *frequencies*.

4.2. Spectral Transformation

Spectral transformation consists of computing the eigenvectors of the Laplacian operator L of the mesh and then projecting the mesh geometry onto these orthonormal basis vectors that span R^n.

The projection results in three n-dimensional vectors, containing the spectral coefficients for x, y and z coordinates of the mesh points. The spectra for x, y and z are separate and their behavior may be different depending on directional geometric properties of the mesh.

Figure 4.1 shows an example of computing mesh spectra on a mesh with only 5 points. The spectral coefficients can be computed by multiplying the coordinate vectors of x, y and z with the matrix whose columns consist of the eigenvectors of L.

(a)

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

(b)

$$\begin{bmatrix} 1.00 & -0.25 & -0.25 & -0.25 & -0.25 \\ -0.25 & 1.00 & -0.25 & -0.25 & -0.25 \\ -0.25 & -0.25 & 1.00 & -0.25 & -0.25 \\ -0.33 & -0.33 & -0.33 & 1.00 & 0 \\ -0.33 & -0.33 & -0.33 & 0 & 1.00 \end{bmatrix}$$

(c)

$$\begin{bmatrix} -0.447 & 0.000 & -0.817 & 0.000 & 0.366 \\ -0.447 & 0.000 & 0.408 & -0.707 & 0.366 \\ -0.447 & 0.000 & 0.408 & 0.707 & 0.366 \\ -0.447 & 0.707 & 0.000 & 0.000 & -0.548 \\ -0.447 & -0.707 & 0.000 & 0.000 & -0.548 \end{bmatrix}$$

(d)

$$[0 \ 1 \ 1.25 \ 1.25 \ 1.51]$$

(e)

Figure 4.1. Computing mesh spectra

(a) The mesh, (b) Adjacency matrix, (c) Laplacian, (d) Eigenvectors and (e) Eigenvalues

4.3. Mesh Partitioning

Computing the eigenvectors of the mesh Laplacian L can be quite costly when the number of vertices n is large. This operation that naturally requires $O(n^3)$ time can be achieved in $O(n)$ time using multi-resolution methods since L is sparse. However, as n gets bigger the complexity cost is still not small enough to ignore and the available methods turn numerically unstable because of adjacent eigenvectors being too close.

In practice, it is impossible to compute the eigenvectors of L for meshes containing more than 1,000 points. If the mesh is too big, it must be partitioned into smaller submeshes with each submesh (partition) processed independently. Partitioning a mesh introduces *edge-effects* and reduces reconstruction quality along the boundaries. On the other hand, local geometric properties are better captured.

The METIS software package [25] provides a partitioning solution for meshes containing up to 100,000 points. METIS somehow prefers minimizing edge-cut to balancing the produced submeshes. The result is that the obtained partitions are reasonably uniform within submesh boundaries and pairs of submeshes possess little statistical dependence. It is also observed that METIS runs reasonably fast on certain test models. Figure 4.2 shows the partitions generated by METIS for the original *bunny* model.

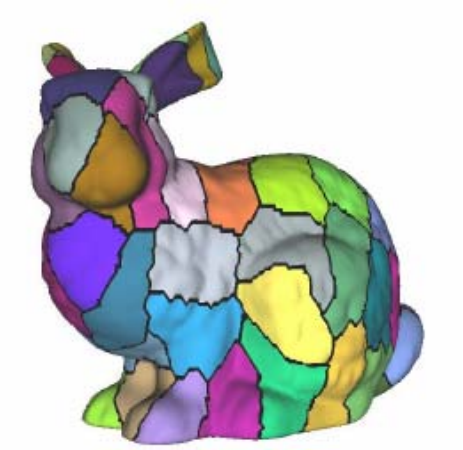


Figure 4.2. Partitioning with METIS

5. PROGRESSIVE GEOMETRY CODING USING SPECK

The SPECK (Set Partitioning Embedded Block Coder) algorithm [2] is a set-partitioning algorithm, which is used for progressively coding images. It is an embedded, block-based algorithm, which codes coefficients of an image wavelet transform. The hierarchical structure of an image transform and energy clustering in frequency and in space are exploited and transform coefficients are bit-plane coded. Coding many zeroes of a bit plane with a single symbol greatly improves the efficiency of the code and the coder is of low complexity.

The frontiers of SPECK are EZW (Embedded Zerotrees of Wavelet Coefficients) [26] and SPIHT (Set Partitioning in Hierarchical Trees) [27] algorithms, which were also designed for image coding and relied on the same principles as SPECK did. SPECK employs a recursive set-partitioning procedure to order coefficients by magnitude with respect to thresholds of powers of two.

In our work, we adapt the SPECK algorithm for our purpose of mesh geometry compression and the bit-plane coding concept differs in that the coded spectral coefficients vectors are 1D. It would be reasonable to name the new concept as *bit-line coding*, but this is not the tradition and we follow the convention of using the term *bit-plane coding*.

5.1. Bit-plane Coding

In a bit-plane coding scheme, the entities to be coded (coefficient values in our case) are ordered according to magnitude or by some other rule. The pattern that these entities form is originally a plane, hence the name *bit-plane*. For example, if an image is treated with a wavelet transform, each wavelet coefficient is located at a pixel of a 2D grid and the grid is a plane. After ordering is performed, the bits are coded and transmitted bit-plane by bit-plane. First, the bits in the bit plane of highest position are transmitted. Following, the

bits in the bit plane of the next highest position are transmitted, and so on. The bit plane in the highermost position corresponds to the most significant bits.

Figure 5.1 shows a schematic binary representation of magnitude-ordered coefficients. Since the coefficients are magnitude-ordered, the many zeroes of a particular bit plane can be coded and transmitted by means of single symbols indicating at which location the zeroes start to appear. The location of the last 1 bit is implicit from this information.

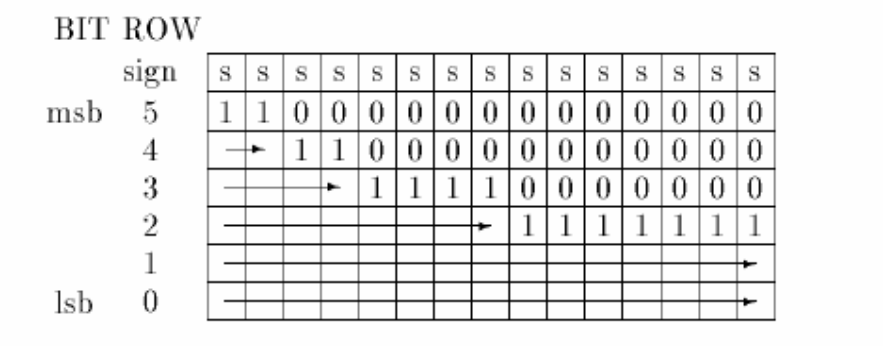


Figure 5.1. Bit-plane coding

5.2. The Way to SPECK

5.2.1. Embedded Zerotrees of Wavelet Coefficients (EZW)

[26] proposed an embedded image coder using zerotrees of wavelet coefficients (EZW). It is based on coding the wavelet coefficients of an image transform by exploiting the similarity inherent in images across scales. This similarity allows prediction of absence of significant information. The wavelet coefficients are quantized and coded at the levels of successive bit planes.

EZW is the first algorithm in the literature that demonstrated the invalidity of the belief that an inverse relation exists between the efficiency of any coder and its time complexity. While EZW is fast, it also provides rate-distortion performances that are

competitive with all image compression algorithms that had been known up to the appearance of EZW.

EZW bases its discussion on why the discrete wavelet transform was chosen as an image transform whose output wavelet coefficients are to be coded. The reason for this is to exploit the spatial similarity that is present in images.

A discrete wavelet transform of an image places a particular wavelet coefficient at all pixels of an image. Hence, the input image and the transformed image are of the same size. The coefficients may be classified as contributing to higher information and lower information content. *Low frequency* coefficients carry more information (such as the average gray level of an image) than *high frequency* coefficients. High frequency coefficients contribute to detail information (such as sharp contours and edges). Since an image has two spatial dimensions, signal frequency is considered in both horizontal and vertical directions.

Partitioning the $2^n x 2^n$ window of the transform image into four, the upper left quadrant is named *LL*, which means that both horizontal and vertical spatial frequencies characterized by those coefficients are low. The upper right quadrant is named *HL* indicating that the horizontal frequency is high and the vertical frequency is low. In turn, the lower left quadrant is *LH* and the lower right quadrant is *HL*. Figure 5.2 is a first stage for a discrete wavelet transform of an image and indicates how quadrants are named.

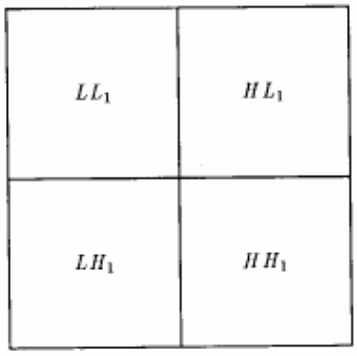


Figure 5.2. Frequency bands of an image wavelet transform

Figure 5.3 is a two-level discrete wavelet transform. Note that at the second level, the LL_1 band of the first level has been partitioned into another four (LL_2, HL_2, LH_2 and HH_2). At each transform level, the uppermost left band is partitioned into four. This property is useful in defining a hierarchical structure for image transforms and exploiting the similarity across image subbands.

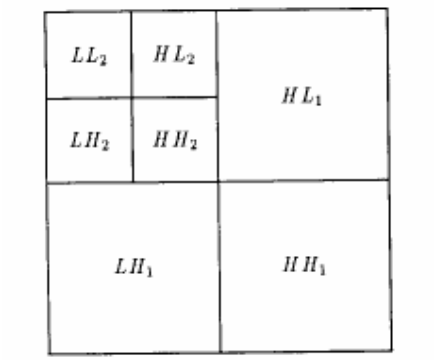


Figure 5.3. A two-level image wavelet transform

Bit-plane coding of decorrelated coefficients (achieved by the transform) is performed through the encoding of a significance map after coefficients are scanned in some order.

EZW defines a new data structure called a *zerotree* for improving the coding of significance maps. A coefficient x is significant if $|x| < T$ for a given threshold T and the assumption is that if a coefficient at a coarse scale is insignificant at a certain threshold level, then it is most likely that all coefficients of the same orientation at finer scales (describing the same spatial location) are insignificant at the same threshold. It is also true that with the exception of the highest frequency subbands, every coefficient at a given scale can be related to a set of coefficients at the next finer scale. This forms a tree structure and *parent-child* relationships between the nodes of this tree. The coefficients at coarser scales are *parent*s of the coefficients at finer scales.

Figure 5.4 is an illustration of the hierarchical dependency of subbands. HH_3 is a zerotree root. Each parent has four children except the lowest frequency subband where each parent has three children.

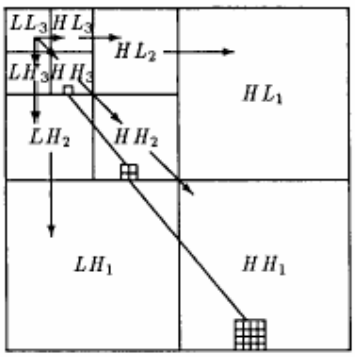


Figure 5.4. Hierarchical dependency of subbands

The scanning order of the coefficients is such that no child node is scanned before its parent is. Figure 5.5 shows the scanning order.

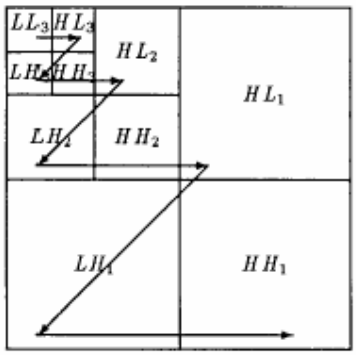


Figure 5.5. Scanning order of wavelet coefficients

Each coefficient is encoded with one of four symbols:

- Zerotree root
- Isolated zero
- Positive significant
- Negative significant

If an insignificant coefficient has no significant descendants, it is a zerotree root. If an insignificant coefficient has significant descendants, it is an isolated zero. Coding the signs of significant coefficients is useful for embeddedness. The flow chart for coding an input coefficient is shown in Figure 5.6.

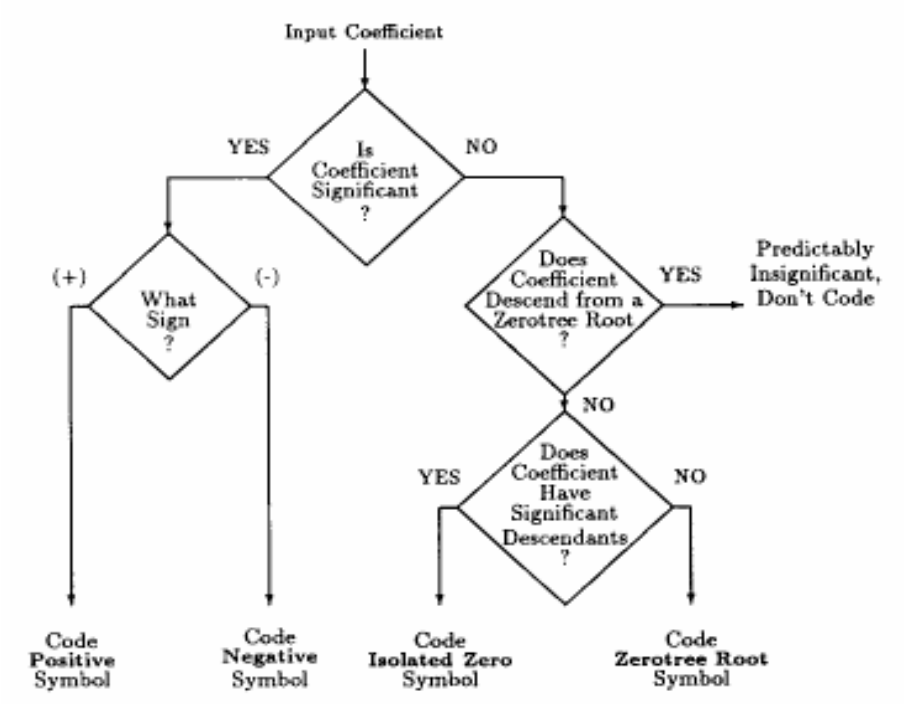


Figure 5.6. Coding an input coefficient

Achieving bit-plane coding is by means of successive-approximation-quantization. A sequence of decreasing thresholds is applied by the encoder (and the decoder). Two separate lists of coefficients are maintained. The *dominant* list contains those coefficients that have not been found to be significant. The s*ubordinate* list contains those coefficients that have been found to be significant. The coding proceeds as an alternating sequence of dominant and subordinate passes. In a dominant pass, if a coefficient is found to be significant, then its significance map is zerotree coded and the coefficient magnitude is appended to the subordinate list. A subordinate pass refines the magnitude of coefficients available to the decoder as dictated by coded bits.

5.2.2. Set Partitioning in Hierarchical Trees (SPIHT)

The Set Partitioning in Hierarchical Trees (SPIHT) algorithm [27] builds on the principles of EZW and provides better performance. [27] explains the principles for the excellent performance of EZW and SPIHT in terms of partial ordering by magnitude, ordered bit-plane transmission and exploiting self-similarity across scales of an image wavelet transform.

The terminology of SPIHT defines parent-child dependencies among subbands of an image wavelet transform as a *spatial orientation tree*.

The significance function that outputs if a certain wavelet coefficient is significant at a threshold level *n* is

$$S_n(T) = \begin{cases} 1 & \text{if } \max_{(i,j)\in T}\{|c_{i,j}|\} \geq 2^n \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

The SPIHT algorithm is shown and described in Figure 5.7 after an introduction of a piece of notation related to the spatial orientation tree as follows:

<u>*Notation:*</u>

- *O(i,j)*: set of coordinates of all direct descendants (offspring) of node *(i,j)*
- *D(i,j)*: set of coordinates of all descendants of node *(i,j)*
- *H:* set of coordinates of all spatial orientation tree roots (nodes at the highest level)
- *L(i,j) = D(i,j) – O(i,j)*
- *LIS*: List of insignificant sets
- *LIP*: List of insignificant pixels
- *LSP*: List of significant pixels
- *Type A entry*: An *LIS* entry representing *D(i,j)*
- *Type B entry*: An *LIS* entry representing *L(i,j)*

Algorithm:

1. **Initialization:** output $n = \left\lfloor \log_2 \left(\max_{(i,j)} \{|c_{i,j}|\} \right) \right\rfloor$; set the LSP as an empty list, and add the coordinates $(i,j) \in \mathcal{H}$ to the LIP, and only those with descendants also to the LIS, as type A entries.

2. **Sorting pass:**

   2.1. for each entry (i,j) in the LIP do:

      2.1.1. output $S_n(i,j)$;

      2.1.2. if $S_n(i,j) = 1$ then move (i,j) to the LSP and output the sign of $c_{i,j}$;

   2.2. for each entry (i,j) in the LIS do:

      2.2.1. if the entry is of type A then

      - output $S_n(\mathcal{D}(i,j))$;
      - if $S_n(\mathcal{D}(i,j)) = 1$ then
        - for each $(k,l) \in \mathcal{O}(i,j)$ do:
          - output $S_n(k,l)$;
          - if $S_n(k,l) = 1$ then add (k,l) to the LSP and output the sign of $c_{k,l}$;
          - if $S_n(k,l) = 0$ then add (k,l) to the end of the LIP;
        - if $\mathcal{L}(i,j) \neq \emptyset$ then move (i,j) to the end of the LIS, as an entry of type B, and go to Step **2.2.2**; else, remove entry (i,j) from the LIS;

      2.2.2. if the entry is of type B then

      - output $S_n(\mathcal{L}(i,j))$;
      - if $S_n(\mathcal{L}(i,j)) = 1$ then
        - add each $(k,l) \in \mathcal{O}(i,j)$ to the end of the LIS as an entry of type A;
        - remove (i,j) from the LIS.

3. **Refinement pass:** for each entry (i,j) in the LSP, except those included in the last sorting pass (i.e., with same n), output the n-th most significant bit of $|c_{i,j}|$;

4. **Quantization-step update:** decrement n by 1 and go to Step **2**.

Figure 5.7. The SPIHT algorithm

5.3. Image Coding using SPECK

Image coding using SPECK can be applied to monochrome images as well as to color images. The latter version is called CSPECK.

5.3.1. SPECK for Monochrome Images

As EZW and SPIHT do, the SPECK algorithm operates by coding the location information of wavelet coefficients of an image transform at successive bit planes.

We start by restating the definition of the significance test of a set *T* of coefficients. *T* is significant with respect to *n* if

$$\max_{(i,j)\in T}\{|\, c_{i,j}\,|\} \geq 2^n \tag{5.2}$$

where *n* is the quantization level (or threshold level). The significance of *T* can be written as

$$\prod_n(T) = \begin{cases} 1 & \text{if } 2^n \leq \max\limits_{(i,j)\in T}\{|\, c_{i,j}\,|\} \leq 2^{n+1} \\ 0 & \text{otherwise} \end{cases} \tag{5.3}$$

This significance test also implies for which quantization level a set has been found significant.

Rectangular regions of the image transform are used in SPECK and these regions are referred to as sets of type S. S sets can be of varying sizes depending on at which layer of the hierarchical structure the set is and, of course, the size of the input image. The number of elements in set S (cardinality of S) is defined as

$$size(S) = C(S) = |\, S\,| \tag{5.4}$$

Sets of varying sizes will be formed as the algorithm runs. A set of size 1 corresponds to a single coefficient.

SPECK also makes use of sets of type I. These sets are obtained by cutting a small square region from the top left part of a larger square region. Figure 5.8 shows a set of type I.

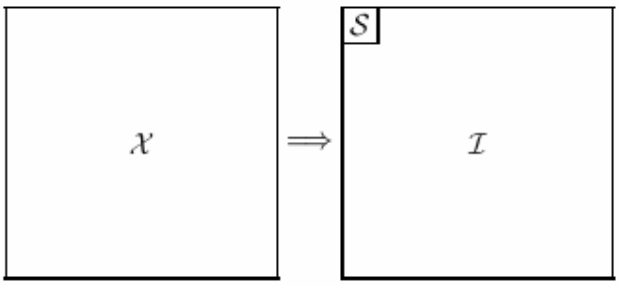


Figure 5.8. Sets of type S and I

I sets are partitioned into S sets and a new I set as the algorithm specifies following a recursive path so as to progressively code the image and move from coarser to finer resolutions in an embedded fashion. Coding is always done on sets of type S.

The difference of SPECK from SPIHT is in the sorting pass step of the algorithms. Instead of using the spatial orientation trees to determine the coefficients for significance testing, sets of type S are used. The idea is to exploit energy clustering in image wavelet transforms. Type S sets concentrate on areas of high energy and high information content is coded first.

SPECK maintains two lists: LIS – List of Insignificant Sets, and LSP – List of Significant Pixels. The former keeps sets of type S that have not yet tested significant with respect to none of the threshold levels *n*, and the latter keeps coefficients (pixels) that have been found to be significant.

Prior to coding, the image is transformed and partitioned into two sets of type S and I as in Figure 5.8. The SPECK algorithm for coding monochrome images consists of four steps and is illustrated in the following:

Algorithm:

1) **Initialization**
   a. Partition image transform X into two sets: $S \equiv$ root, and $I \equiv X\text{-}S$
   b. Output $n_{\max} = \lfloor \log_2 (\max_{(i,j)\in X} | c_{i,j} |) \rfloor$
   c. Add S to LIS and set $LSP = \emptyset$
2) **Sorting pass**
   a. In increasing order of size $|S|$ of sets (smaller sets first)
      i. For each set S in LIS do ProcessS (S)
   b. If $I \neq \emptyset$, ProcessI ()
3) **Refinement pass**
   a. For each (i,j) in LSP, except those included in the last sorting pass, output the n^{th} most significant bit of $|c_{i,j}|$
4) **Quantization step**
   a. Decrement n by 1 and go to step 2

The four functions called by the algorithm are described in Figure 5.9.

Procedure `ProcessS`($\mathcal{S}$)

1) output $\Gamma_n(\mathcal{S})$
2) if $\Gamma_n(\mathcal{S}) = 1$
   - if $\mathcal{S}$ is a pixel, then output sign of $\mathcal{S}$ and add $\mathcal{S}$ to LSP
   - else `CodeS`($\mathcal{S}$)
   - if $\mathcal{S} \in$ LIS, then remove $\mathcal{S}$ from LIS
3) else
   - if $\mathcal{S} \notin$ LIS, then add $\mathcal{S}$ to LIS
4) return

Procedure `CodeS`($\mathcal{S}$)

1) Partition $\mathcal{S}$ into four equal subsets $\mathcal{O}(\mathcal{S})$
2) for each set $\mathcal{S}_i \in \mathcal{O}(\mathcal{S})$ $(i = 0, 1, 2, 3)$
   - output $\Gamma_n(\mathcal{S}_i)$
   - if $\Gamma_n(\mathcal{S}_i) = 1$
     - if $\mathcal{S}_i$ is a pixel, output its sign and add $\mathcal{S}_i$ to LSP
     - else `CodeS`($\mathcal{S}_i$)
   - else
     - add $\mathcal{S}_i$ to LIS
3) return

Procedure `ProcessI`()

1) output $\Gamma_n(\mathcal{I})$
2) if $\Gamma_n(\mathcal{I}) = 1$
   - `CodeI`()
3) return

Procedure `CodeI`()

1) Partition $\mathcal{I}$ into four sets—three $\mathcal{S}_i$ and one $\mathcal{I}$
2) for each of the three sets $\mathcal{S}_i$ $(i = 0, 1, 2)$
   - `ProcessS`($\mathcal{S}_i$)
3) `ProcessI()`

Figure 5.9. The functions used by the SPECK algorithm

During the course of the algorithm, sets of type S and I are partitioned by different partitioning schemes (in CodeS () and CodeI () functions). If a set S tests significant against a threshold *n*, it is partitioned into four sets of type S whose sizes are one-fourth of the original set. This *quadtree-partitioning* is illustrated in Figure 5.10.

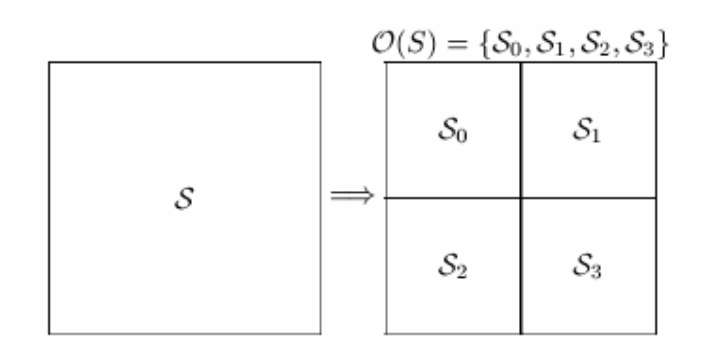


Figure 5.10. Partitioning of set S

After all sets of type S are processed and coded at a particular threshold level *n*, the set I is partitioned into four sets by another partitioning scheme: *octave-band-partitioning*. This produces three sets of type S and a new set I. The size of the new S sets is identical to the already chopped portion of the transform. Octave-band partitioning is shown in Figure 5.11.

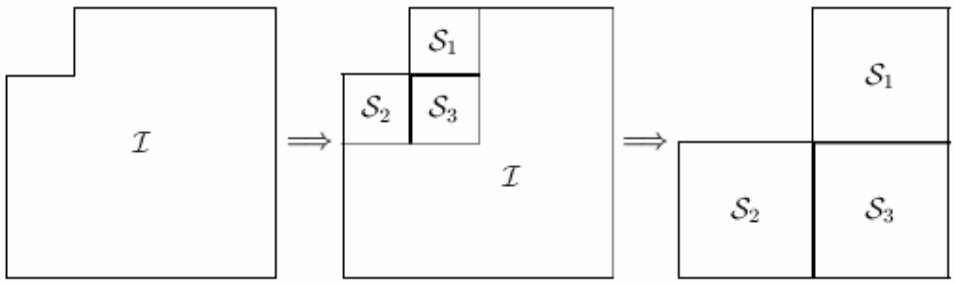


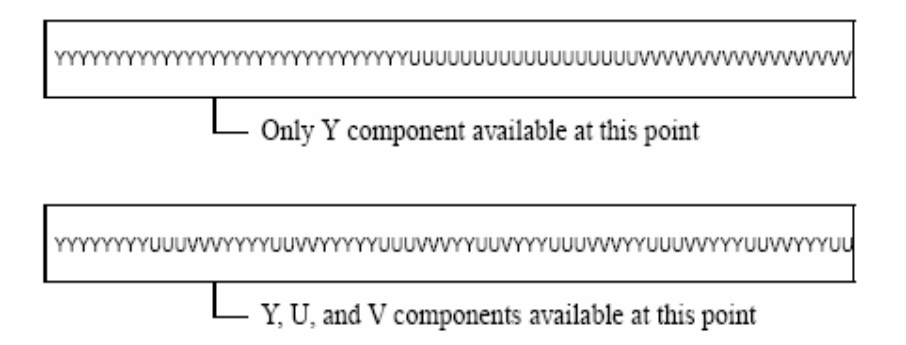
Figure 5.11. Partitioning of set I

The partitioning schemes that are used aim to exploit the hierarchical pyramidal structure of the subband decomposition of the image transform. Energy is concentrated at the topmost levels of the pyramid and partitioning S achieves climbing higher in this hierarchy and processing high energy coefficients. Partitioning sets of type S runs recursively until significant coefficients are located and coded. Partitioning the I set into four creates three new S sets that are close to the top level in the hierarchy containing high energy and they are to be coded before the coefficients that remain in the I set.

SPECK processes S sets in an order of increasing size, because small S sets are more likely to contain and arrive at a significant coefficient.

5.3.2. CSPECK

Generalizing SPECK to code color images must ensure that bits are fairly allocated among color components to achieve full color embeddedness. Hence, coding coefficients of each color component separately and concatenating the bitstreams would not meet this requirement. CSPECK codes color images in the YUV format and generates one mixed bitstream for all color component planes. Figure 5.12 shows a conventional color bitstream and the bitstream produced by CSPECK.



5.12. Compressed color bitstreams

The color space YUV consists of three planes, where U and V are each one-quarter of Y. Each plane is wavelet-transformed by its own to the same number of decomposition levels. The initialization partitions each plane into one set of type S and one set of type I, as in the monochrome SPECK. Each of the transform planes is accompanied by an *LIS* but there is only one *LSP* for all planes. In the scheme of CSPECK, the sorting passes at a threshold level *n* are executed in turn for the Y, U and V components and significant coefficients are mixed in the single *LSP*. After the sorting passes are over, the refinement pass runs on *LSP* and the process goes on with the next threshold level.

5.4. Geometry Coding using SPECK

Our adaptation of SPECK to the geometry coding problem is applied on the spectral coefficients obtained through the method of [1]. Each of *x*, *y* and *z* coordinates of a transformed mesh produces a coefficients vector of size *N*, where *N* is the number of vertices. In the following subsection, geometry coding for a single coefficients vector using the approaches of SPECK is described. The procedure of bit allocation to the three components (*x*, *y* and *z*) is described in the later subsection that follows.

5.4.1. Coding 1D Coefficients Vectors

The significance test for a coefficient takes the form

$$\prod_n(T) = \begin{cases} 1 & \text{if } 2^n \le \max_{(i\in T)}\{|c_i|\} \le 2^{n+1} \\ 0 & \text{otherwise} \end{cases} \tag{5.5}$$

Note that the coefficient locations are determined by a single index, since the adapted algorithm is on linear regions of 1D coefficients vectors. To facilitate the concept of energy clustering, the spectral coefficients are sorted in increasing order of eigenvalues associated with the corresponding eigenvector of each coefficient. Because of this sorting action, low frequency coefficients take place at the beginning and high frequency coefficients are located at the end.

Like the image coding case, sets of type S and I exist with the geometry coder. The cardinality of a set is similarly defined and the size of S sets depends on subband level and the length of the coefficients vector.

The initialization step involves forming the initial sets of type S and I, outputting the maximum threshold level and adding S to *LIS*. Figure 5.13 shows how the coefficients vector is initially partitioned.

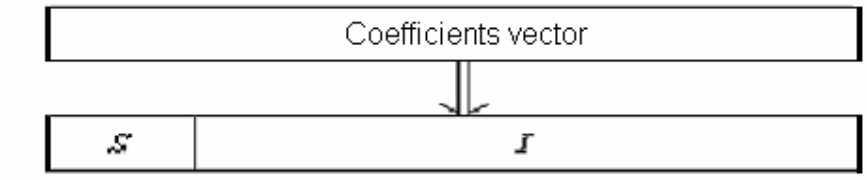


Figure 5.13. Partitioning the coefficients vector to S and I sets

Coding is performed on sets of type S. Lists *LIS* and *LSC* (was named *LSP* before) that are maintained are defined as in the image coding case. *LIS* contains sets of type S that have not been found to be significant and *LSC* keeps coefficients that have tested significant against some threshold *n*.

Our modified algorithm for a single coefficients vector (e.g. *x* component) is as follows:

Algorithm:

1) **Initialization**
   a. Apply spectral transformation on the input mesh and compute spectral coefficients
   b. Sort the coefficients in ascending order of eigenvalues
   c. Partition the coefficients vector X into two sets: $S \equiv$ root, and $I \equiv X\text{-}S$
   d. Output $n_{\max} = \lfloor \log_2 (\max_{i \in X} |c_i|) \rfloor$
   e. Add S to *LIS* and set $LSC = \emptyset$
2) **Sorting pass**
   a. In increasing order of size $|S|$ of sets (smaller sets first)
      i. For each set S in *LIS* do ProcessS (S)
   b. If $I \neq \emptyset$, ProcessI ()
3) **Refinement pass**
   a. For each i in *LSC*, except those included in the last sorting pass, output the n^{th} most significant bit of $|c_i|$
4) **Quantization step**
   a. Decrement n by 1 and go to step 2

The codes for the ProcessS (), CodeS (), ProcessI () and CodeI () are identical to the code presented in Figure 5.9 except that CodeS () partitions S into two new sets of type S and CodeI () partitions I into one set of type S and one set of type I.

If a set is found significant for some threshold *n*, it is partitioned into two smaller sets of type S, each being half the size of the original set. We call this partitioning method *binary-tree partitioning* illustrated in Figure 5.14.

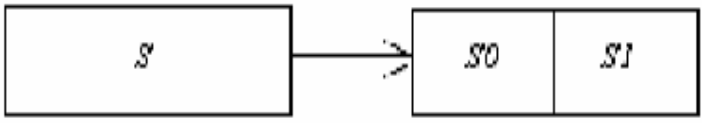


Figure 5.14. Partitioning sets of type S

Partitioning sets of type I consists of obtaining a set of type S and the new I set. The size of the new S set is the same as the size of the chopped portion. This partitioning scheme is illustrated in Figure 5.15.

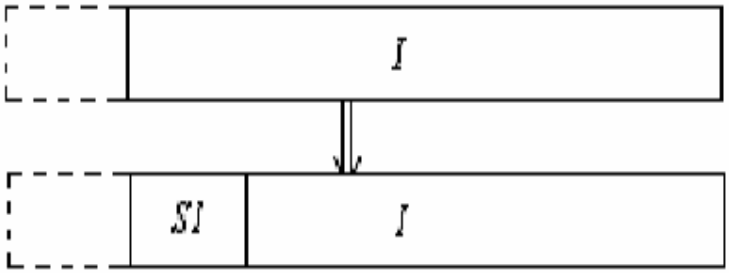


Figure 5.15. Partitioning sets of type I

S sets are processed in order of increasing size and before the I set at each quantization level. A truly embedded code is generated since the principle of energy clustering is successfully exploited.

5.4.2. Generalization to CSPECK

Adopting the CSPECK algorithm to code the three 1D coefficients vectors, the SPECK algorithm is applied separately (the sorting pass) for each component (x, y and z) at a threshold level n and the execution goes on with the next sorting pass after the coefficients in *LSC* have been refined at the corresponding bit plane. Following the approach of CSPECK achieves a fair bit allocation among all components and an embedded geometry coder. If the mesh to be coded consists of a large number of vertices (>500), it is partitioned (by METIS) into smaller submeshes and CSPECK runs on each submesh transform (covering all submeshes in a particular order) for a specific bit-plane. The next group of sorting passes (three in total for each geometry component) takes place at the level of the next bit plane in the same order for all submeshes.

6. ERROR METRIC

The common practice to measure the error between the original and a reconstructed version of a signal is through computing the MSE (Mean Square Error) that arises when corresponding discrete signal points are considered.

In the case of measuring the visual distance between a 3D model and one of its particular reconstructions, the same approach can be followed and the MSE computed using the squares of differences of the corresponding vertex coordinates or the square root of the MSE, known as RMS (Root Mean Square) error, can be reported.

The MSE is:

$$MSE = \frac{\sum_{i \in v} (v_i - v_i')^2}{N} \tag{6.1}$$

where *v* and v' are *the vertex sets* (the coordinates) of the two models and the RMS error is

$$RMS = \sqrt{MSE} \tag{6.2}$$

If the signal of a 3D object is considered solely of consisting of 3D points (a cloud of these points), the RMS error measure is the best that gives an intuition of the geometric closeness of these points. However, a signal of a 3D object is more than that. Besides the existence of points that build the mesh, there are also the connectivity relations between vertices. Since the signal for a 3D object is discrete, but in fact is the approximation of continuous surface information, the adjacency relations of vertices and faces must be considered and fairly treated for defining an error metric that captures the visual distance between two models (the original and its approximation). Such a metric takes into account the smoothness of the mesh at each vertex location.

The error metric proposed in [1] is referred to as the *visual metric* and we use the same metric to report our results. Because our intention is to compare our own results with those of [1], this approach is fair.

Defining the visual metric requires the computation of a geometric Laplacian at each vertex position of a mesh. The value of the geometric Laplacian at vertex v_i is

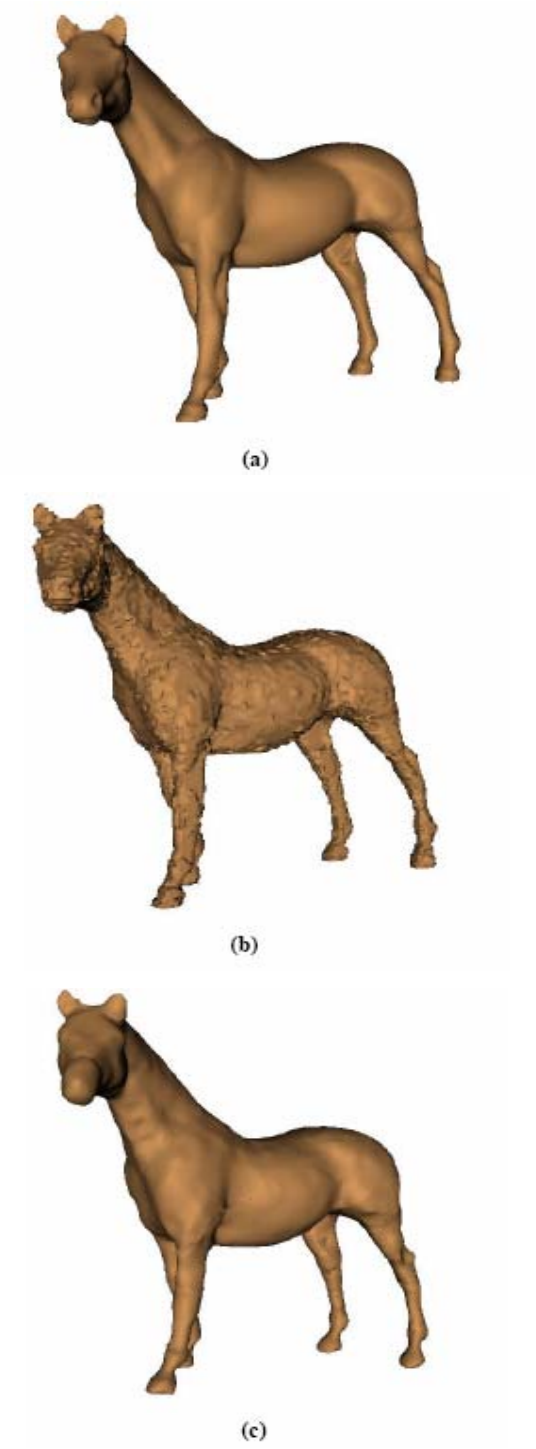
$$GL(v_i) = v_i - \frac{\sum_{j \in n(i)} l_{ij}^{-1} v_j}{\sum_{j \in n(i)} l_{ij}^{-1}} \tag{6.3}$$

where *n(i)* is the index set for vertices adjacent to v_i and l_{ij} is the geometric distance between v_i and v_j. It must be noted that more weight is given to vertices that are closer than to those that are farther. Since this is the sensible thing to do if smoothness is taken into account; given that M^1 and M^2 are two models with vertex sets v^1 and v^2, the visual metric (the visual distance between the two models) using the geometric Laplacian is defined as

$$\left\| M^1 - M^2 \right\| = \frac{1}{2n} \left(\left\| v^1 - v^2 \right\| + \left\| GL(v^1) - GL(v^2) \right\| \right) \tag{6.4}$$

Observe that the visual distance is the simple average norm of the geometric distance between models and the norm of the Laplacian difference.

Figure 6.1 shows the justification of using the visual metric instead of RMS error to measure the visual distance between models. (a) is the original *horse* model M^1, (b) is an approximation M^2 where $\left\| v^1 - v^2 \right\| = 0.10$ and (c) is an approximation M^3 where $\left\| v^1 - v^3 \right\| = 0.10$. It is obvious that M^3 is closer to M^1 than M^2 is. However, pure geometric distance does not capture this fact. Computing the visual distance with (6.4), it is observed that $\left\| M^1 - M^2 \right\| = 0.16$ and $\left\| M^1 - M^3 \right\| = 0.07$. Hence, the use of the visual metric is justified.



(a)

(b)

(c)

Figure 6.1. Visual metric

7. EXPERIMENTAL RESULTS

The results presented in this section are based on the comparative rate-distortion performance of the spectral geometry coder of [1] (referred to as the *KG coder* due to Karni and Gotsman) and the geometry coder of our implementation called the *CSPECK coder*.

The KG coder applies a spectral transformation to the geometry of the input mesh and obtains three coefficients vectors for the geometry components of x, y and z. Exploiting the fact that the energy of the geometry information is concentrated in low frequency coefficients (the ones that correspond to small eigenvalues), it expects to obtain a rather good approximation of the original model with a relatively small number of low-frequency coefficients. Due to this reason, it truncates the coefficients vectors and does not use a large number of high frequency coefficients in the reconstruction process. This saves from the bit budget because fewer coefficients are coded with fewer bits. The way followed in coding the truncated coefficients is to quantize all coefficients to a finite precision. Typical quantization levels are 12, 14 and 16-bit quantization. The coder can be viewed as a progressive coder because as more coefficients are used for reconstruction, the visual quality increases. However, the coder does not possess the property of embeddedness of desirable progressive coders, because some coefficients are not used at all and the coded coefficients are not bit-plane coded.

On the other hand, the CSPECK geometry coder is an embedded coder because it treats all bits at a bit-plane together in every pass and progresses to the lower bit planes and bit allocation to coordinates and partitions is implicitly performed.

In our comparative analysis of the rate-distortion performance of the coders, we have applied entropy coding and employed a universal lossless coding technique. This technique is known as *arithmetic coding* [28] and is the most efficient lossless coding technique known to date. Our application of arithmetic coding is adaptive.

We have used 12, 14 and 16-bit quantization levels to measure the performance of the KG coder. We have not copied any results that were mentioned in [1], but also implemented the spectral coder besides our own CSPECK coder and collected the results.

Our test models for comparing the performance of the CSPECK coder with that of the KG coder are the *horse* model that consists of 19,851 points and 36,698 faces (triangles) and the *bunny* model that consists of 34,835 points and 69,472 faces (also triangles). The *horse* mesh has been partitioned to 40 and the *bunny* mesh has been partitioned to 70 submeshes by METIS before coding has taken place. Figure 7.1. and Figure 7.2. demonstrate the rate-distortion curves for the two models of the CSPECK coder and the KG coder with different quantization levels.

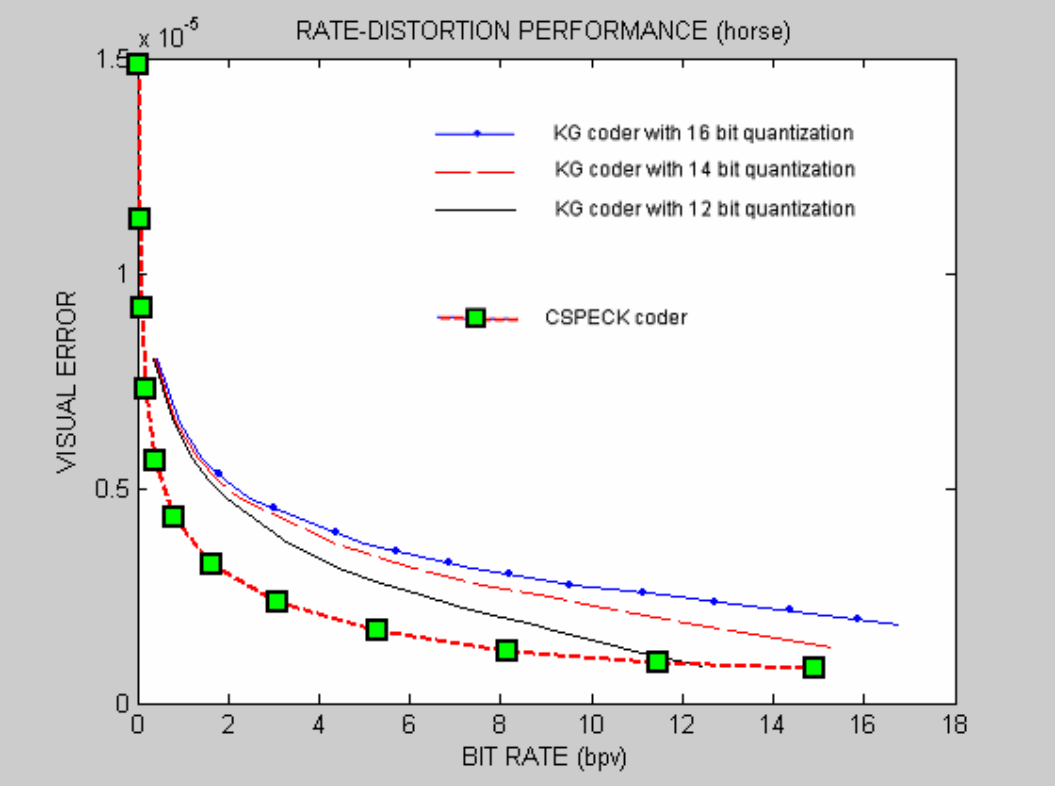


Figure 7.1. Rate-distortion performance (*horse*)

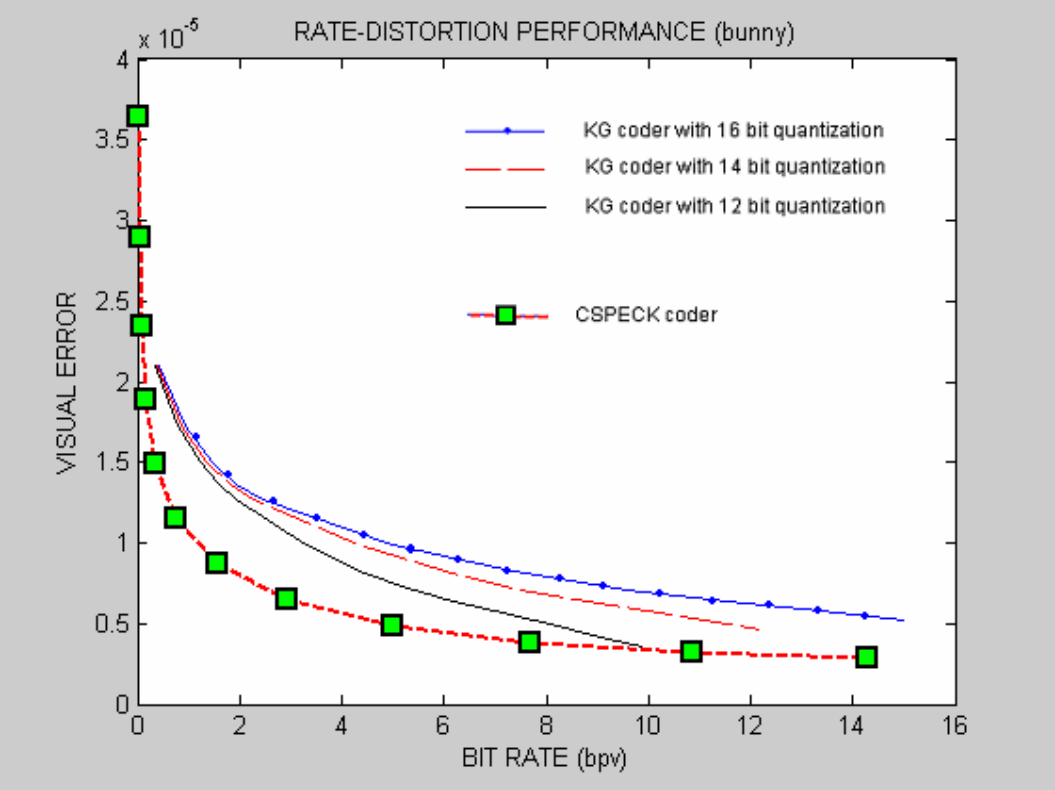


Figure 7.2. Rate-distortion performance (*bunny*)

It can clearly be observed that the performance of the CSPECK coder at all rates is much superior to that of the KG coder for both models.

8. CONCLUSIONS

We have presented a geometry coder that utilizes the CSPECK algorithm to code the spectral coefficients obtained via the spectral method of [1]. It must be noted that the KG coder and the CSPECK coder deal only with the geometry compression part of a mesh coder. For a full mesh coder, connectivity must be coded using another scheme. The coders that follow the spectral approach are different from most geometry coders because the coding of geometry is not dictated by the order of traversal of vertices implied by the connectivity code. Nevertheless, it cannot be stated that the connectivity information does not interact with the geometry code. The spectral coefficients that describe the geometry are obtained by using the mesh topology information (remember that the Laplacian matrix L used for the spectral transformation is constructed through the information available in the adjacency matrix A).

In the method of [1], a sub-vector of spectral coefficients that contain the most of the total energy is coded. In this scheme, more importance is given to the least significant bits of the high energy coefficients than the most significant bits of low energy coefficients and thus those bits are later transmitted or never transmitted. In other words, the right priority to the bits of coefficients is not given. Another problem that is caused by truncation is that the pattern of the bit allocation to be applied to unions of coordinates and partitions cannot be estimated and these unions are coded at different rate-distortion points. That is why, the rate-distortion performance is good at low bit rates but at medium and high rates it is worse than that of methods relying on prediction [16]. Our method realizes implicit bit allocation to partitions and coordinates, gives the right priority to the bits of coefficients, codes the zero bits of insignificant coefficients with a single symbol in one pass and thus the performance is better at also the medium and high bit rates.

As a last remark we put a word on some future work. The efficient geometry coding technique we have developed so far will further be improved upon newer ideas under the umbrella of a new project. The implementation of a decoder is essential for a full system.

REFERENCES

1. Karni, Z. and C. Gotsman, "Spectral Compression of Mesh Geometry", *ACM SIGGRAPH Conference Proceedings*, pp 279-286, 2000.

2. Pearlman, W. A., A. Islam, N. Nagaraj, and A. Said., "Efficient, Low-Complexity Image Coding with a Set-Partitioning Embedded Block Coder", *IEEE Trans. Circuits and Systems for Video Technology*, Vol. 14, pp. 1219-1235, November 2004.

3. Bézier, P., "Numerical Control: Mathematics and Applications", Wiley, Chichester, UK, 1972.

4. Touma, C. and C. Gotsman, "Triangle Mesh Compression", *Graphics Interface 98 Conference Proceedings*, pp. 26-34, 1998.

5. Isenburg, M., "Compressing Polygon Mesh Connectivity with Degree Duality Prediction", *In Graphics Interface '02 Conference Proceedimgs*, pp. 161-170, 2002.

6. Khodakovsky, A., P. Schröder, and W. Sweldens, "Progressive Geometry Compression," *Proceedings of ACM SIGGRAPH 2000*, pp. 271-278, 2000.

7. Rossignac, J., "Edgebreaker: Connectivity Compression for Triangle Meshes", *IEEE Transactions on Visualization and Computer Graphics*, 1999.

8. Gumhold, S., "New Bounds on the Encoding of Planar Triangulations", *Technical Report WSI-2000-1*, University of Tübingen, 2000.

9. Isenburg, M. and J. Snoeyink, "Spirale Reversi: Reverse Decoding of the Edgebreaker Encoding", *In Proceedings of 12th Canadian Conference on Computational Geometry*, pp. 247-256, 2000.

10. King, D. and J. Rossignac, "Guaranteed 3.67V bit Encoding of Planar Triangle Graphs", *In 11th Canadian conference on Computational Geometry*, pp. 146-149, 1999.

11. Isenburg, M. and J. Snoeyink, "Mesh Collapse Compression", *In Proceedings of SIBGRAPH'99*, Campinas, Brazil, pp. 27-28, 1999.

12. Khodakovsky, A., P. Alliez, M. Desbrun, and P. Schröder, "Near-Optimal Connectivity Encoding of 2-Manifold Polygon Meshes". *Graphical Methods, special issue*, 2002.

13. Deering, M., "Geometry Compression", *ACM SIGGRAPH Conference Proceedings*, pp. 13-20, 1995.

14. Taubin, G., W. Horn, J. Rossignac, and F. Lazarus, "Geometry Coding and VRML", *In Proceedings of the IEEE*, volume 86(6), pp. 1228-1243, June 1998.

15. Lee, E. and H. Ko, "Vertex Data Compression for Triangular Meshes", *In Proceedings of Pacific Graphics*, pp. 225-234, 2000.

16. Bayazıt, U., Ö. Orcay, U. Konur, and F. S. Gürgen, "Predictive Vector Quantization of 3-D Polygonal Mesh Geometry by Representation of Vertices in Local Coordinate Systems", *Proceedings of EUSIPCO 2005*, December 2005.

17. Isenburg, M. and P. Alliez, "Compressing Polygon Mesh Geometry with Parallelogram Prediction", *In IEEE Visualization Conference Proceedings*, pp. 141-146, 2002.

18. Attene, M., B. Falcidieno, M. Spagnuolo, and J. Rossignac, "SwingWrapper: Retiling Triangle Meshes for Better Edgebreaker Compression", *ACM Transactions on Graphics*, 2003.

19. Szymczak, A., J. Rossignac, and D. King, "Piecewise Regular Meshes: Construction and Compression", *Graphical Models*, 2003.

20. Hoppe, H., "Progressive Meshes", *In ACM SIGGRAPH Conference Proceedings*, pp. 99-108, 1996.

21. Pajarola, R. and J. Rossignac, "Compressed Progressive Meshes", *IEEE Transactions on Visualization and Computer Graphics*, 6(1), pp. 79-93, 2000.

22. Cohen-Or, D., D. Levin, and O. Remes, "Progressive Compression of Arbitrary Triangular Meshes", *In IEEE Visualization Conference Proceedings*, pp. 67-72, 1999.

23. Alliez, P. and M. Desbrun, "Progressive Encoding for Lossless Transmission of 3D Meshes", *In ACM SIGGRAPH Proceedings*, pp. 198-205, 2001.

24. Taubin, G., "A Signal Processing Approach to Fair Surface Design", *Proceedings of ACM SIGGRAPH '95*, pp. 351-358, 1995.

25. Karypis, G. and V. Kumar, "MeTiS: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-reducing Orderings of Sparse Matrices", Version 4.0, Univ. of Minnesota, Dept. Of Computer Science, 1998. Available at http://www.users.cs.umn.edu/~karypis/metis/metis.html

26. Shapiro, J. M., "Embedded Image Coding using ZeroTrees of Wavelet Coefficients", *IEEE Tranactions. on Signal Processing*, vol. 41, pp. 3445-3462, December 1993.

27. Said, A. and W. A. Pearlman, “A New, Fast and Efficient Image Codec based on Set Partitioning in Hierarchical Trees”, *IEEE Tranactions on Circuits and Systems for Video Technology*, vol. 6, pp. 243-250, June 1996.

28. Witten, I. H., R. M. Neal, and J. G. Cleary, “Arithmetic Coding for Data Compression”, *Communications of the ACM*, Volume 30, Number 6, pp. 520-540, June 1987.