M.I.T Media Laboratory Perceptual Computing Section Technical Report No. 286

Coding of Segmented Image Sequences

by

Ujjaval Yogesh Desai

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degrees of

Master of Engineering

and

Bachelor of Science in Electrical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1994

Author . . . .
Department of Electrical Engineering and Computer Science
May 13, 1994

Certified by . . . .
E. H. Adelson
Associate Professor, M.I.T. Media Lab
Thesis Supervisor

Accepted by . . . .
F. R. Morgenthaler
Chair, Departmental Committee on Graduate Students

Coding of Segmented Image Sequences
by
Ujjaval Yogesh Desai

Submitted to the Department of Electrical Engineering and Computer Science
on May 13, 1994, in partial fulfillment of the
requirements for the degrees of
Master of Engineering
and
Bachelor of Science in Electrical Engineering

Abstract

Standard image sequence coding methods, such as MPEG, use block-based motion compensation and transform techniques to reduce temporal and spatial redundancies. Higher compression, however, can be achieved by object-based image sequence representation. In an object-based representation scheme, the image sequence is decomposed into objects on the basis of motion, texture, structure, etc. By efficiently encoding these arbitrarily shaped objects, significantly lower bit rates can be achieved. We propose an object coding method which separately encodes the intensity and shape of objects. A compact description of the shape is obtained by using a contour encoding algorithm, while efficient transform coding techniques are used to compress the intensity. We describe three such techniques: smooth filling, cosine filling, and region dependent transform coding. We compare these methods with the KLT.

Thesis Supervisor: E. H. Adelson
Title: Associate Professor, M.I.T. Media Lab

Acknowledgements

First of all, I would like to thank Ted Adelson, my advisor, for his great ideas, constant support, and understanding. He taught me many things about research and life. Working with him was a privilege.

Many thanks to John Wang for good discussions and fine advice. His insight and enthusiasm amazed me. I am grateful to him for reading my thesis carefully and for providing helpful suggestions. This work would not have been possible without him.

Thank you, Steve and Funmi for the patience to read the thesis. Thanks to Kris, Baback, Alex, Chris, Stephen, Thad, Lee, Sourabh, George, and all other members of Vismod for their support. Special thanks to my friends, Daniel, Jerome, and Naimish, for their companionship.

Finally, I would like to thank my family for their love and support.

This research was sponsored in part by the Television of Tomorrow project at the MIT Media Laboratory.

Contents

1 Introduction **6**
- 1.1 Thesis organization . . . 8

2 Background **9**
- 2.1 Lossless source coding . . . 9
  - 2.1.1 Huffman coding . . . 9
  - 2.1.2 Run length coding . . . 10
  - 2.1.3 Lempel-Ziv Algorithm . . . 10
- 2.2 Transform coding . . . 11
  - 2.2.1 KLT . . . 12
  - 2.2.2 DCT . . . 13
  - 2.2.3 JPEG . . . 14
- 2.3 Image sequence coding . . . 15
  - 2.3.1 Block-based sequence coding . . . 16
  - 2.3.2 Model-based sequence coding . . . 16
  - 2.3.3 Object-based sequence coding . . . 16

3 Approach **19**
- 3.1 Object representation . . . 19
- 3.2 A simple method for encoding objects . . . 20
- 3.3 Our approach . . . 21

4 Coding of alpha maps **22**
- 4.1 Boundary contour extraction . . . 22
- 4.2 Lossless contour encoding . . . 22
- 4.3 Lossy contour encoding . . . 26
  - 4.3.1 Morphological smoothing . . . 26

5 Coding of intensity maps **28**
- 5.1 Smooth filling . . . 28
  - 5.1.1 Advantages of smooth filling . . . 30
  - 5.1.2 A better compression scheme . . . 30
- 5.2 Cosine filling . . . 30
  - 5.2.1 Motivation . . . 30
  - 5.2.2 The filling problem . . . 31
  - 5.2.3 Direct approach to the filling problem . . . 32
  - 5.2.4 Solution . . . 34
  - 5.2.5 Improvements to cosine filling . . . 35

5.2.6 Illustrations of cosine filling . . . 36
5.3 Region dependent transform coding . . . 36
5.3.1 Motivation . . . 36
5.3.2 Previous work . . . 39
5.3.3 Solution . . . 40
5.3.4 Advantages of region dependent transform coding . . . 42

6 Compression results **44**
6.1 Analysis of edge block coding methods . . . 44
6.1.1 Edge block coding results . . . 44
6.1.2 Comparison of edge block coding methods . . . 45
6.2 Summary of the object coding algorithm . . . 46
6.2.1 Encoder . . . 46
6.2.2 Decoder . . . 46
6.3 Object coding results . . . 46

7 Conclusions and future work **48**
7.1 Conclusions . . . 48
7.2 Suggestions for future work . . . 49

Bibliography **49**

Chapter 1

Introduction

The objective of image compression is to represent an image as compactly as possible for efficient transmission and storage. The compression is achieved by exploiting the spatial redundancy present in the image and the limitations of the human visual system. In addition to spatial redundancy, there is a significant temporal correlation between frames of an image sequence. Therefore, instead of encoding the frames separately, it is highly advantageous to compress them jointly.

The spatial correlation in an image can be reduced by using techniques such as transform and waveform coding [14, 16, 19, 21], whereas the temporal correlation can be reduced by motion compensation [14, 16]. An example of coding with motion compensation is the MPEG coder, which is the current industry standard. MPEG is considered to be a block-based coding technique because it performs motion compensation on small image blocks. A detailed description of MPEG will be given in the next chapter.

Even though MPEG reduces spatial as well as temporal redundancies, it fails to provide very high compression for sequences with occlusion. Occlusion is encountered when one object temporarily hides another by moving over it. Block-based motion compensation techniques are quite inefficient in dealing with occlusion. The simple block model is unable to describe the multiple motions arising from an occlusion. Object-based coding methods, on the other hand, do not have this problem.

In object-based coding, an image sequence is segmented into objects using characteristics such as motion, intensity, shape, etc. [18]. A general block diagram for an object-based coder is presented in figure 1-1. Wang and Adelson [1, 24, 25] have proposed *the layered representation* for efficient image sequence decomposition. In the layered representation, an image sequence is decomposed into a set of overlapping objects, which are referred to as layers, on the basis of their motion(see chapter 2 for details). Figure 1-2 shows 3 input frames of the MPEG flower garden sequence. The layered representation enables us to describe 30 frames of this sequence with the layers shown in figure 1-3. This reduction in data clearly shows the high compression potential of this representation.

The object coding module in figure 1-1 encodes the arbitrarily shaped objects that result from segmentation. Several efficient methods for encoding these objects will be presented in this thesis. It is important to note that our object coding algorithms are not specific to the layered representation. They can be used in any of the numerous methods that produce objects, for example model-based image coding and coding of still-images using segmentation based on intensity or texture [10]. Also, it is widely believed that the next generation of image coders will utilize mid-level vision techniques that include segmentation and object representation. These image coders will inevitably require coding of arbitrarily shaped objects, making the algorithms described in this thesis highly applicable.

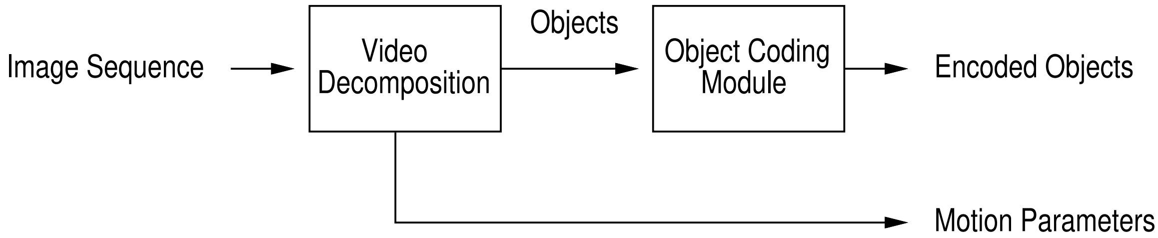


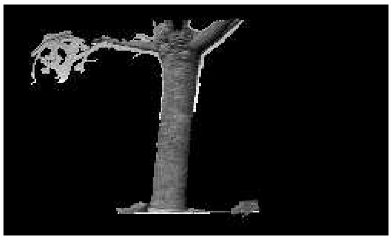
Figure 1-1: A general block diagram of an object-based coder

(a) Frame 1

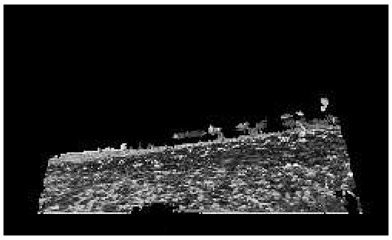
(b) Frame 15

(c) Frame 30

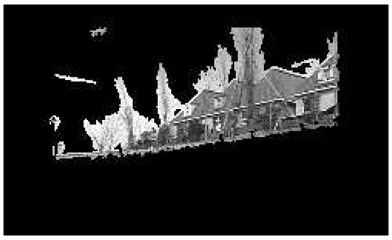
Figure 1-2: Three frames of the flower garden sequence.



(a) Tree Layer



(b) Flower bed layer



(c) House layer

Figure 1-3: This figure shows three layers obtained by segmenting the flower garden sequence. The unsupported regions are shown in black. The background sky layer is not shown.

1.1 Thesis organization

Chapter 2 describes fundamentals of image coding. It consists of three sections: entropy coding; transform coding; block-based and object-based image sequence coding. Chapter 3 outlines our approach to object coding. In our representation, an object consists of two components: an alpha map and an intensity map. The alpha map represents the region mask of the object, while the intensity map represents the intensity profile in the supported region of the object. Efficient algorithms for alpha map and intensity map encoding are presented in chapters 4 and 5, respectively. Compression results are given in chapter 6. Finally, chapter 7 presents conclusions and ideas for future work.

Chapter 2

Background

This chapter reviews some of the standard techniques for coding still-images and image sequences. The methods described here provide background for understanding the object coding algorithms presented in this thesis. Detailed description of image coding fundamentals can be found in [14, 16, 19]. This chapter is divided into three main sections. Section 2.1 covers various lossless source coding techniques. Transform coding methods are described in section 2.2. Finally, section 2.3 presents methods for image sequence compression.

2.1 Lossless source coding

In lossless coding, the input signal is encoded without any distortion. The theoretical bound for lossless compression is given by Shannon's *noiseless coding theorem*. Consider a source that produces L *independent* symbols with probabilities p_i $(i = 0, 1, ..., L-1)$. The entropy of this source is defined to be:

$$H = -\sum_{i=0}^{L-1} p_i \log_2 p_i \quad \text{bits per symbol} \tag{2.1}$$

According to Shannon's theorem, it is possible to represent a source of entropy H bits per symbol with $H + \epsilon$ $(\epsilon > 0)$ bits per symbol, where ϵ can be made arbitrarily small.

2.1.1 Huffman coding

Huffman coding is a theoretically optimal algorithm for lossless source coding. It is a fixed-to-variable length code, that is, it maps fixed length input symbols to variable length codewords. It reduces the average codeword length by assigning shorter codewords to highly frequent symbols and longer codewords to rarely occurring symbols. The algorithm is as follows [12]:

1. List all input symbols and consider them as leaf nodes of a binary tree.

2. Create a parent node for the two nodes with smallest probability. Assign the parent node a probability equal to the sum of the probabilities of the two children. Arbitrarily assign 1 and 0 to the branches connecting the parent node to the children.

3. Replace the two children in the list by their parent node. Repeat step 2 while there is more than one node in the list.

4. The sequence of bits associated with the branches in the path from the root node to a leaf is the codeword for the input symbol at that leaf.

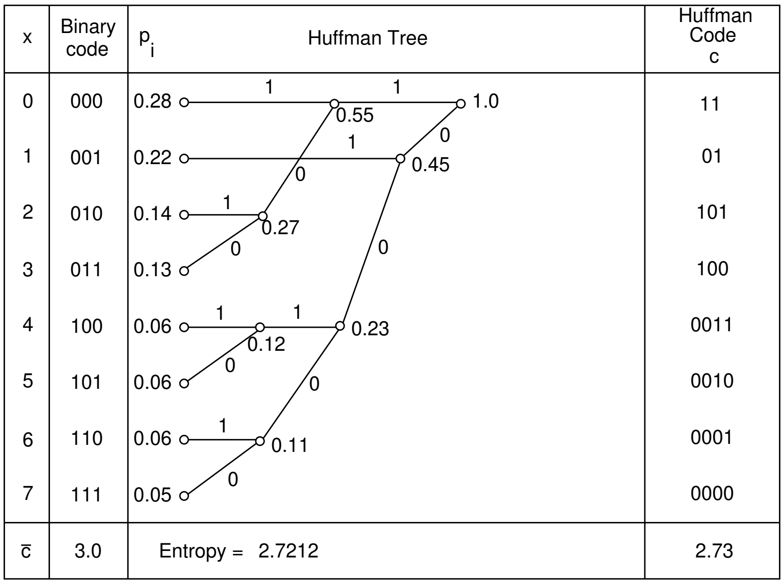


Figure 2-1: An illustration of huffman coding. $\overline{c}$ represents the average codeword length.

This algorithm can be used to generate codewords for any input symbol probabilities. An illustration of Huffman coding is given in figure 2-1. Huffman encoding and decoding simply require a table look-up. Because of the simplicity of this algorithm, Huffman codes are extremely popular for lossless source coding. An alternative to Huffman coding is *arithmetic coding*, which is described in [28].

An image can be viewed as a source if we consider the intensity at each pixel to be a source symbol. Huffman coding can then be used on this model of the image. Unfortunately, Huffman coding is not very useful for compressing grey-scale images because the symbols usually have uniform distribution. However, it can be of great use in transform coding and in coding binary data.

2.1.2 Run length coding

Huffman coding assumes that the input symbols are independent. This is not a good assumption for real data, which usually has considerable correlation between symbols. Run length coding can be used to exploit the inter-symbol redundancy. In run length coding of a binary source, the number of 0s between two successive 1s is coded. If the source has long runs of 0s, run length coding can be very effective. Binary images such as weather maps, line drawings, and text are good examples of sources that can be efficiently compressed with run length coding. Further improvement in compression can be obtained by using Huffman coding in conjunction with run length coding.

2.1.3 Lempel-Ziv Algorithm

The Lempel-Ziv algorithm is a universal data compression scheme [6]. It is asymptotically optimal and has a simple implementation. Consider a binary sequence that needs to be encoded. The algorithm sequentially parses the sequence into strings that have not been seen before. The parsing is done by looking, starting at the current bit, for the shortest string in the sequence that has not been encountered so far. This string can be represented as a prefix, which has occurred already, and the last bit of the string. The string can now be coded by encoding the index of the prefix and the value of the last bit. This process is repeated until we reach the end of the sequence. For example, the binary sequence “001011000110 ...” will be parsed as “0, 01, 011, 00, 0110, ...”.

Let c be the number of strings in the sequence. Therefore, we can encode each string with $log(c) + 1$ bits, where $log(c)$ is for the prefix and 1 bit is for the value of the last bit in the string. Since this algorithm maps variable length input symbols to fixed length codewords, it can be considered as a variable-to-fixed length code. An implementation of the Lempel-Ziv algorithm as a variable-to-variable length code can be found in [4, 27].

Due to the speed and efficiency of the Lempel-Ziv algorithm, it has become a standard for compressing text and facsimile. In chapter 4, we will use it for alpha map encoding.

2.2 Transform coding

So far we have only considered lossless compression of images. In real systems, however, it is acceptable to trade-off some image quality for improvement in compression. The *rate-distortion function*, $R(D)$, provides theoretical bounds on the performance of lossy compression schemes. One commonly used distortion measure is the *root-mean-squared error*($RMSE$):

$$RMSE = \sqrt{\frac{1}{NM}\sum_{i=1}^{N}\sum_{j=1}^{M}[f(i,j) - \hat{f}(i,j)]^2} \tag{2.2}$$

where $f(i,j)$ and $\hat{f}(i,j)$ are the input and reconstructed images, respectively. The images are assumed to be of dimension NxM.

Although there are numerous lossy compression techniques, we will only concentrate on transform coding. The principles of transform coding will be of great importance in understanding the object coding techniques presented later.

A general transform coding scheme involves three steps: transformation, quantization, and entropy encoding. The goal of transformation is to reduce the redundancy in the input signal. This is achieved by transforming the input into coefficients that have as little correlation as possible. Uncorrelated coefficients are usually compact, which means that most energy is packed in a few coefficients. Compaction is necessary for high compression because for a compact set of coefficients, only the high energy coefficients need to be encoded. The transforms are usually chosen to be invertible, so that the decoder can reconstruct the input signal. Therefore, the transformation step is lossless.

The distortion is, however, introduced in the quantization process, which discards coefficients with low energy. The quantized coefficients are then losslessly encoded using entropy coding methods such as Huffman coding and run length coding. For high compression, the entropy encoder must exploit the statistical properties of the quantized coefficients. Although there are interesting issues in designing optimal quantizers and entropy encoders, we will only consider them in the case of JPEG, which is a standard image transform coding scheme. We now present some of the important image transforms.

Let $\underline{u}$ be the Mx1 vector containing the input signal. A linear transformation, T (MxM), maps $\underline{u}$ into a vector of coefficients, $\underline{v}$, according to

$$\underline{v} = T\underline{u} \tag{2.3}$$

There are two interpretations of linear transforms:

1. From the above equation, it is clear that transforms can be considered as rotations of the coordinate axes. Consider an input vector that has a uniform distribution of energy among its elements. By rotating the coordinate axes, it is possible to redistribute energy so that most of it is present is very few coefficients. This is one of the reasons for using transforms for compresion.

2. Transforms can also be considered as basis function decompositions. The inverse transform relationship is given by

$$\underline{u} = T^{-1}\underline{v}. \tag{2.4}$$

T^{-1} exists because the transform is chosen to be invertible.
The above equation can also be written as:

$$\underline{u} = \sum_{i=0}^{M-1} v(i)\underline{t_i} \tag{2.5}$$

where $\underline{t_i}$ is the i^{th} column of T^{-1}.
Equation 2.5 suggests that the input can be expressed as a linear combination of the $\underline{t_i}$s. Therefore, the $\underline{t_i}$s are the basis functions of the M-dimensional space, $\mathcal{S}$, of all possible inputs.

The $\underline{t_i}$s, in general, are linearly independent vectors, and they span $\mathcal{S}$. However, an orthonormal basis is desirable because such a basis is necessary for decorrelating the coefficients, and it has fast implementations. An orthonormal basis has an additional property that the energy in the coefficients is equal to the energy in the input. The orthonormality condition is:

$$\underline{t_i}^T \underline{t_j} = \begin{cases} 0, & \text{if} \quad i \neq j \\ 1, & \text{if} \quad i = j \end{cases} \tag{2.6}$$

A transform with orthonormal basis is called a *unitary transform*. From the definition of orthonormality, it follows that

$$T^{-1} = T^T \tag{2.7}$$

As we mentioned before, an efficient transform must produce coefficients that are uncorrelated and compact. Next, we show that the *Karhunen-Loeve Transform* is the optimum transform that satisfies these properties.

2.2.1 KLT

The KLT has the property that its coefficients are uncorrelated and have maximum compaction. An elegant derivation for the KLT is given in [14]. It can be proven that the basis functions of the KLT are the eigenvectors of the input covariance matrix.

Let $\underline{u}$ be the vector of input random variables with an MxM covariance matrix, Λ_u. Let T be a unitary transformation and $\underline{v}$ be the coefficients. Λ_u is defined as

$$\Lambda_u = E[(\underline{u} - \underline{u_m})(\underline{u} - \underline{u_m})^T] \tag{2.8}$$

where $\underline{u_m}$ is the mean of $\underline{u}$. Λ_v, the covariance matrix of $\underline{v}$, can be shown to be

$$\Lambda_v = T\Lambda_u T^T \tag{2.9}$$

Now, suppose that the columns of T^T are the eigenvectors of Λ_u.[1] Then,

$$\Lambda_u t_i = \lambda_i t_i \tag{2.10}$$

where t_i is the i^{th} column of T^T and λ_i is the corresponding eigenvalue.

From this it follows that

$$\Lambda_u T^T = T^T D \tag{2.11}$$

[1] Λ_u has a complete set of orthonormal eignvectors beacuse it is symmetric.

where D is the diagonal matrix with λ_is on the diagonal:

$$D = diag(\lambda_0, \lambda_1, ..., \lambda_{M-1}). \tag{2.12}$$

By substituting equation 2.11 into equation 2.9, we obtain

$$\Lambda_v = TT^T D \tag{2.13}$$

Since T is a unitary matrix, the above equation simplifies to

$$\Lambda_v = D. \tag{2.14}$$

The KLT coefficients are uncorrelated because, as equation 2.14 suggests, their covariance matrix is diagonal. The proof that the KLT is optimal in energy compaction can be found in [14].

Although the KLT is optimal, it is not very practical for two reasons:

- The basis functions have to computed before transformation because the basis functions depend on the input. This significantly increases the amount of computation that has to be performed.
- The basis functions have to be transmitted along with the coefficients because the decoder needs to know the basis functions to reconstruct the input.

Due to these drawbacks of the KLT, practical systems use several sub-optimal transforms. The *Discrete Cosine Transform*(DCT) is one of them.

2.2.2 DCT

The forward DCT of an NxM image is defined as

$$v(i,j) = C(i,j) \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} u(n,m) cos[\frac{(2n+1)i\pi}{2N}] cos[\frac{(2m+1)j\pi}{2M}] \tag{2.15}$$

and the inverse DCT is defined as

$$u(n,m) = \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} C'(i,j) v(i,j) cos[\frac{(2n+1)i\pi}{2N}] cos[\frac{(2m+1)j\pi}{2M}] \tag{2.16}$$

where $C(i,j)$ and $C'(i,j)$ are the normalization constants.

The performance of the DCT has been shown to be virtually indistinguishable from the KLT. For certain images, the DCT basis functions approach the KLT basis functions [20]. The complete set of the 8 x 8 DCT basis functions are shown in figure 2-2. As one can see, the basis functions are cosines of various frequencies. The DCT is similar to the Discrete Fourier Transform (DFT) in that they both map spatial domain to frequency domain. However, there is an important difference between the two transforms. The DCT gives very high compaction, which makes it much more useful than the DFT. The reason for this difference in compaction can be understood if we interprete the DFT and the DCT in terms of the Discrete-Time Fourier Transform (DTFT).

The DFT is a frequency sampled version of the Discrete-Time Fourier Transform (DTFT). Since sampling in frequency corresponds to replication in space, the DFT is implicitly equal to the DTFT of a periodic sequence. This replication of the input usually results in discontinuities in the spatial domain, which correspond to spurious frequencies and, hence, to poor compaction.

The definition of the DCT in equation 2.15 can be shown to be consistent with the interpretation of the DCT as the DFT of the extended input which is created by reflecting the original input about the vertical axis. The

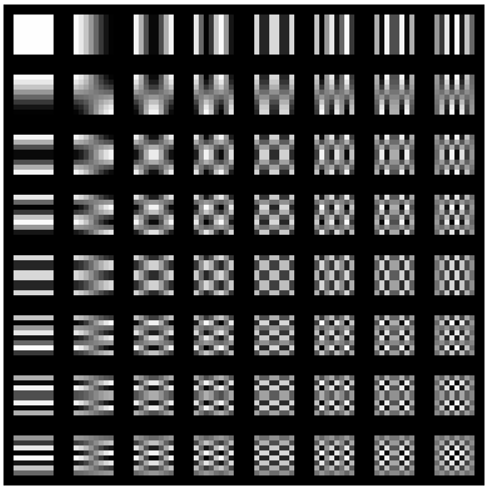


Figure 2-2: The basis functions of the 8 x 8 DCT

DFT of the extended signal has fewer discontinuities than the DFT of the original signal. Thus, the DCT has better compaction than the DFT. Furthermore, since the extended signal is even, the DCT requires only real computations. Due to these advantages, the DCT is highly popular.

2.2.3 JPEG

As we mentioned earlier, a general transform coding method consists of: transformation; quantization; and entropy encoding. We have already looked two important transforms, the KLT and the DCT. In this section, we describe the JPEG algorithm, which is a DCT-based transform coding method. JPEG is the current industry standard for compression of still-images.

The JPEG algorithm consists of three stages:

1. The original image is partitioned into 8 x 8 blocks and each block is transformed using the 8 x 8 DCT. Block-based transforms are used because they offer considerable reduction in computation and storage without much reduction in compression.

2. The transform coefficients are quantized using a user selected normalization table.

3. The quantized coefficients are entropy coded for improved compression.

Quantization

In the quantization stage, the transform coefficients are first normalized (weighted) according to a user-defined normalization table. The normalized coefficients are then "rounded-off" to the nearest integer. Let $v(i,j)$ and $v^*(i,j)$ be the transform coefficients and the normalized coefficients, respectively. Let $N(i,j)$ be the normalization factor. Then,

$$v^*(i,j) = round(\frac{v(i,j)}{N(i,j)}) \tag{2.17}$$

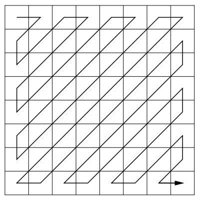
The normalization table directly controls the amount of distortion introduced in each coefficient during encoding. The properties of the *human visual system* are used to design the basic normalization table. The user can control the distortion and, hence, the compression by scaling the normalization table. A default JPEG normalization table is given in figure 2-3(a). From the table, it is clear that high frequencies are quantized more coarsely than low frequencies. This is because low frequencies are perceptually more important.

Entropy coding

The quantized coefficients usually have a large fraction of energy in very few coefficients, especially in low frequencies. The JPEG algorithm exploits this property of the coefficients by arranging them in a zigzag ordering (see figure 2-3(b)). The zigzag ordering arranges the quantized coefficients in approximately decreasing order of average energy. The ordered coefficients are then encoded using pre-defined huffman codes and run length codes. Since only the first few ordered coefficients need to be encoded, significant reduction in bit rate can be achieved.

$$N(i,j) = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

(a)



(b)

Figure 2-3: (a) The default normalization table; (b)zigzag ordering

The advantages of JPEG can be summarized as follows:

- Fast implemenation exists for the DCT.
  1. It is an $\mathcal{O}(nlog(n))$ transform (like the DFT).
  2. It only requires real computations.
- The DCT has an excellent energy compaction property.
- The JPEG quantizer exploits properties of the human visual system.

2.3 Image sequence coding

The lossless source coding methods described in section 2.1 can be considered as 1-D image coding techniques because they do not exploit the 2-D spatial redundancies present in real images. Transform coding , on the other hand, can be used for significant reduction in 2-D spatial correlation. The methods described in this section can be considered as 3-D coding techniques since they exploit the spatial as well as temporal redundancies that exist in image sequences. The spatial correlation is reduced by *intraframe coding*, whereas the temporal correlation is reduced by *interframe coding*.

The frames of an image sequence usually differ by some motion transformation. Consequently, the temporal correlation can be reduced if we can accurately estimate this motion. Unfortunately, the problem of estimating motion between two frames is extremely difficult. Real sequences are usually composed of objects that have different motion. In order to estimate the motion of these objects, accurate segmentation is required. On the other hand, it is difficult to perform good segmentation without any motion information. Hence, motion estimation and segmentation are highly interrelated problems that have to be solved simultaneously. There are three main classes of image sequence coding methods: block-based; model-based; and object-based. These three methods are presented in the rest of this chapter.

2.3.1 Block-based sequence coding

Block-based methods segment the sequence into small blocks and perform motion estimation on these blocks, assuming that each block has a single translational motion. This kind of motion estimation technique is known as *block matching*. In block matching for each block in the current frame, the closest match in the previous frame is found. Let $I(x, y, t)$ and $I(x, y, t-1)$ be the frames at times t and $t-1$, respectively. Then the motion vector (v_x, v_y) is found by minimizing

$$\sum_{x,y \in \mathcal{R}} [I(x, y, t) - I(x - v_x, y - v_y, t-1)]^2 \tag{2.18}$$

where $\mathcal{R}$ denotes the region of the block in the current frame.

One example of block-based sequence coding is MPEG, which is the current industry standard for video coding. It uses motion-compensated predictive coding for reducing temporal redundancy and the DCT for reducing spatial redundancy. The main idea behind motion-compensated prediction is to predict the current frame using the previous frame and the motion between the current and previous frames. The prediction is obtained by simply displacing the blocks in the previous frame according to the estimated motion. If the estimated motion is close to the true motion, the prediction error is very small. This error can then be coded using still-image coding techniques. Since MPEG applies prediction to blocks of data, it is considered to be a block-based sequence coding method. Once the motion is estimated, it finds the prediction error and encodes it using the DCT. The details of the algorithm can be found in [13].

The disadvantage of MPEG is that for many real sequences, block matching fails to give good estimates of motion. Block matching performs poorly if the block falls on object boundary or if the motion in the block is not translational (shear, rotational, and zoom). However, the object-based methods, to be discussed later, are capable of dealing with these problems.

2.3.2 Model-based sequence coding

In model-based coding, the objects in the sequence are represented with pre-specified models. The encoder compares the input with the model and extracts a set of parameters. Since the image is being represented by only a few parameters, high compression can be achieved. The parameters are updated frequently in order to represent moving objects. The decoder simply reconstructs the objects using the parameters and the model. Clearly, model-based coding can only be used for certain special sequences where the video content satisfies some fixed models. Examples of such sequences include the sequences used by video-conferencing systems. Even in the case of these special sequences, noise, occlusion, and changes in illumination make the extraction of parameters an extremely difficult problem. Due to its difficult implementation and limited applicability, model-based coding is not used in general image sequence compression.

2.3.3 Object-based sequence coding

As was mentioned earlier, block-based motion estimation techniques fail to model object motion adequately. Object-based methods have been suggested for more accurate motion estimation [18]. In this section, we will provide a brief description of the *layered representation*, put forth by Wang and Adelson. This section is only meant to highlight the main ideas behind the representation. For details, refer to [1, 24, 25, 26]. The figures included in this section are taken from those papers.

Layered representation

The layered representation decomposes an image sequence into a set of overlapping objects, which are referred to as layers, based on their motion. The algorithm has two main steps: local motion estimation and motion segmentation.

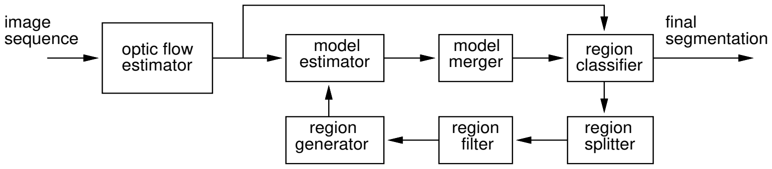


Figure 2-4: Block diagram of the motion segmentation algorithm.

Local motion estimation

The local motion between successive frame pairs is obtained with a multi-scale coarse-to-fine optic flow algorithm [17]. The optic flow is a gradient based motion estimation technique, given by

$$I(x - v_x(x,y), y - v_y(x,y), t-1) \approx I(x,y,t) \tag{2.19}$$

where $(v_x(x,y), v_y(x,y))$ is the motion at (x,y). The optic flow equation can be solved by estimating the linear least-squares solution for motion over a local region, R:

$$\min_{v_x, v_y} \sum_R [I(x - v_x(x,y), y - v_y(x,y), t-1) - I(x,y,t)]^2 \tag{2.20}$$

This equation can be simplified by linearizing it. The final solution can be shown to be :

$$\begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} \sum -I_x I_t \\ \sum -I_y I_t \end{bmatrix} \tag{2.21}$$

where I_x, I_y, and I_t are the partial derivatives of I w.r.t. to x, y, and t, respectively. The region R is chosen to be a 5 x 5 gaussian window. The optic flow algorithm estimates motion at each point of the image to produce a dense motion field. Figure 2-5(a) shows the optic flow from the first two frames of the flower garden sequence (see figure 1-2).

Motion segmentation

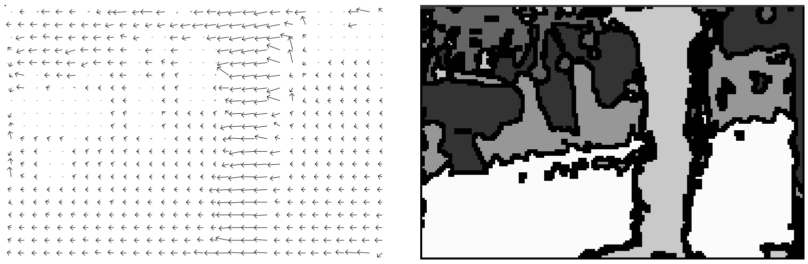
Once the optic flow has been estimated, layers can be created by grouping together pixels that are moving with similar motion. Each object's motion is represented with an affine model. The affine motion model is capable of describing various types of motion, including translation, rotation, zoom, and shear. The equations for the model are

$$v_x(x,y) = a_{x0} + a_{xx}x + a_{xy}y \tag{2.22}$$
$$v_y(x,y) = a_{y0} + a_{yx}x + a_{yy}y. \tag{2.23}$$

Using the affine model, the motion of an object can be described with only six parameters per frame, thereby providing reduction in bit rate.

The motion segmentation starts out by estimating affine models from the flow field. The affine models are initially estimated over small rectangular block regions. Since blocks corresponding to a single object will have similar affine parameters, they are merged together using a k-means clustering algorithm [23]. However, for some blocks that fall on object boundaries, the initial affine models do not represent their motion accurately. For better segmentation of these blocks, hypothesis testing is used to determine the correct model for each pixel. The correct model is the one that minimizes the error between the optic flow estimate and the motion given by the affine models. As a result of hypothesis testing and region assignment, we obtain arbitrarily shaped regions that have coherent motion. The segmentation can be improved by iterating through this procedure. After just a couple of iterations, a very stable and accurate segmentation is achieved. Figure 2-5(b) shows the result of segmenting the first frame of the flower garden sequence.

To maintain temporal coherence and stability, the segmentation result from the first two frames is used as initial conditions for the segmentation of the next pair of frames. This also achieves fast convergence. Once the whole sequence has been segmented, temporal accumulation of data is performed to obtain the layers (see figure 1-3). The images corresponding to a coherent motion region in different frames only differ by an affine transformation. By applying inverse transformations to all the frames, we can align the coherent motion regions. In this motion-compensated sequence, the pixels that appear to be stationary can be collected, and a layer can be formed.



(a) (b)

Figure 2-5: (a) Optic flow; (b) Segmentation

In this representation, a layer consists of an intensity profile and a region mask. The region mask describes the supported region of the layer. In the algorithm presented above, the region mask is binary; however, in sequences with transparent motion, the mask can be non-binary. In computer graphics, such a mask is referred to as an *alpha map*. We will use this terminology in the rest of this thesis.

The layered representation is superior to block-based motion estimation because it uses overlapping layers for handling occlusion and allows for non-translational motion by using affine models. An object-based sequence coder can utilize the layered representation for motion segmentation. The block diagram for such an object-based coder is shown in figure 2-6. For high compression, the objects produced by the layered representation have to compressed. This task is performed by the object coding module. An efficient implementation of the object coding module will be presented in the next chapter.

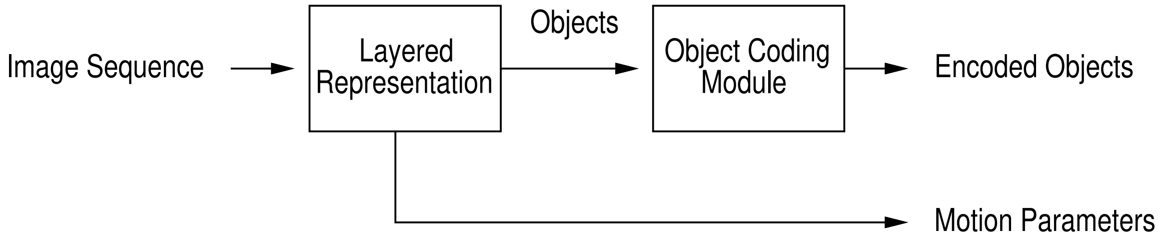


Figure 2-6: Block diagram of an object-based coder

Chapter 3

Approach

3.1 Object representation

An object consists of an intensity map, which represents its intensity profile, and an alpha map, which represents its shape and transparency. An alpha map, in general, can be either binary or non-binary, but for now we will only consider binary alpha maps. Figures 3-1 and 3-2 show the alpha and intensity maps of the tree object.

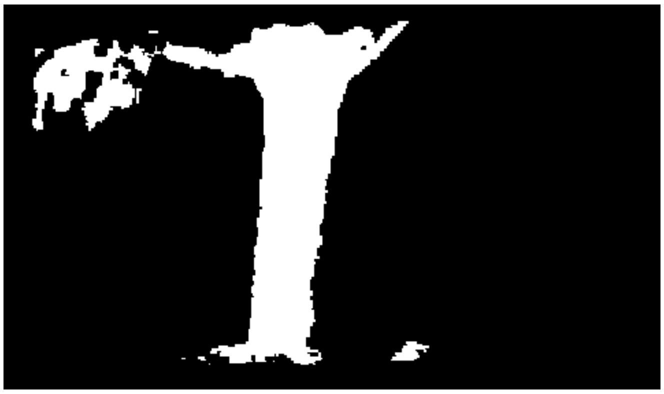


Figure 3-1: The binary alpha map of the tree layer

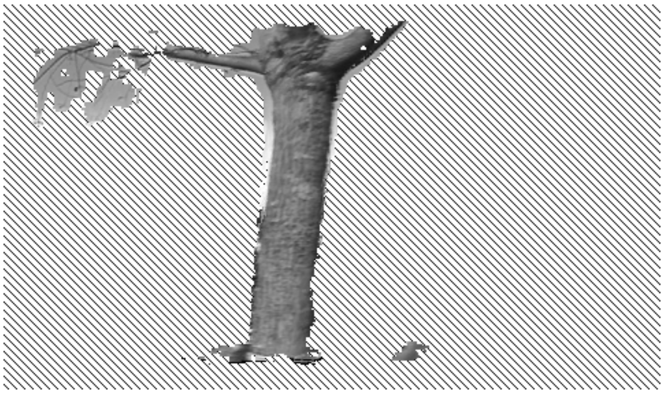


Figure 3-2: The intensity map of the tree layer. The hatched region represents the unsupported pixels.

3.2 A simple method for encoding objects

One way of encoding an object is to first convert it into an image and then apply standard still image coding schemes, such as JPEG, to compress the image. An object can be converted into an image, which we call a composite map, by setting all pixels in the unsupported region to a known constant, for instance zero. Figure 1-3 actually shows the composite maps of the objects obtained by decomposing the flower garden sequence.

Once the composite map is obtained, we can use JPEG to compress it. As discussed in chapter 2, JPEG uses the 8 x 8 DCT, which is a frequency-domain transform, to achieve good energy compaction. The DCT gives good compaction for rectangular images; however, it performs poorly on arbitrarily shaped images.

To better analyze the performance of the DCT, let's classify all possible 8 x 8 blocks of an object into three categories: interior blocks, in which all pixels are supported; exterior blocks, in which no pixels are supported; and edge blocks, in which a fraction of pixels are supported. Figure 3-3 shows these blocks.

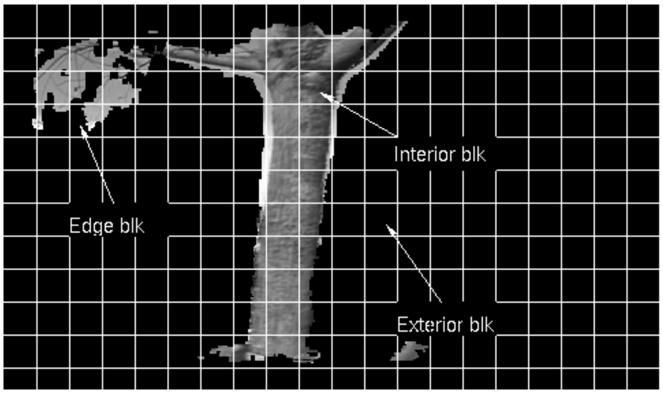
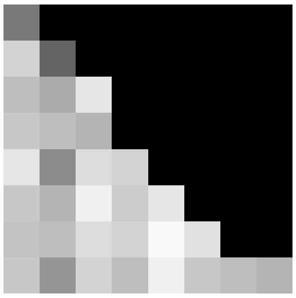
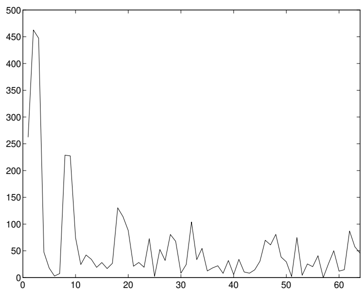


Figure 3-3: 8 x 8 blocks of the composite map of the tree layer

Since the DCT is a rectangular transform (i.e., has a rectangular region of support), JPEG can encode the exterior and interior blocks efficiently. However, since the boundary between the supported and unsupported regions of edge blocks creates large steps in intensity, which correspond to energy spread in the frequency domain, the energy compaction of the DCT is significantly reduced. A typical edge block of a composite map and its DCT are shown in figure 3-4. Poor compaction in the DCT coefficients is evident from the multiple peaks present in figure 3-4(b).



(a)



(b)

Figure 3-4: (a) An edge block; (b) The DCT of the edge block in (a). The dc component has been attenuated by a factor of 3 for proper displaying.

3.3 Our approach

A better method to compress objects is to separately encode the intensity and alpha maps. The intensity and alpha maps have distinct properties that can be exploited if they are encoded independently. The alpha map, which represents the shape of the object, is completely independent of the intensity profile, and hence, it is advantageous to compress it separately. The intensity map represents the intensity profile only of the supported region. The pixels in the unsupported region have no effect on the intensity map, and can take on any value. Since JPEG performs poorly on edge blocks because of the boundary between supported and unsupported regions, one good way of modifying the pixels in the unsupported region is to assign them intensity values that reduce(or eliminate) this boundary. It is important to note that using the information provided by the alpha map, the decoder can reconstruct the original object from the modified intensity map, that is, the modified intensity map and the alpha map contain full information about the object.

Our object encoding algorithm has two components: Alpha map encoding and intensity map encoding. The next chapter describes our alpha map encoding algorithm. Chapter 5 presents several efficient techniques for compressing intensity maps.

Chapter 4

Coding of alpha maps

In this chapter, we will address the problem of compressing alpha maps. It was pointed out in chapter 2 that if an object exhibits transparent motion, its alpha map will be non-binary, otherwise it will just be a binary mask. For simplicity, we assume that the alpha maps are binary. Our lossless alpha map encoding algorithm is described in section 4.2. Section 4.3 presents a method for lossy compression of alpha maps.

The binary alpha map of an object is shown in figure 3-1. In chapter 2 we discussed simple methods for compressing binary masks, e.g., run length coding (RLC) and Lempel-Ziv coding. These techniques can certainly be used for coding alpha maps, but better compression can be obtained if we use the fact that an alpha map represents an object's shape. It is very important to realize that the shape of an object can be completely represented by its boundary contour. Therefore, the task of compressing binary alpha maps can be accomplished by using a contour encoding algorithm.

4.1 Boundary contour extraction

A simple algorithm to find the boundary of a binary mask is as follows: Create a binary image, say V, of the size of the mask and initialize it to zero. Scan the mask row by row. On each row, if the current pixel is 1 and if both the left and right pixels are not 1s, assign 1 to the current pixel of V. Now create another binary image, say H, in the same manner as V, but this time scan the image column by column. V has all the vertical edges of the mask and H has all of the horizontal ones. The boundary of the mask is just the binary sum of V and H. An illustration of this algorithm is given in figure 4-1. Figure 4-2 shows the boundary contour of the tree layer given in figure 1-3.

4.2 Lossless contour encoding

Our contour encoding algorithm uses chain coding [11] to obtain a compact description of the alpha map. In chain coding, a contour is represented by a starting point and a sequence of directionals that gives the direction of travel as the contour is traced. The starting point of a contour can be found by simply scanning the mask, row by row, for the first non-zero pixel. Once the starting point has been found, the following procedure is used:

- Find the next point by searching withing the 8 neighbors for a non-zero pixel in a clockwise fashion, starting from the pixel at the location given by the previous directional.
- Output a directional representing the direction of the next point relative to the current directional.
- Go to the next point.

	1	1	1	1	1	1
	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
	1	1	1	1	1	1
		1	1	1	1	1
			1	1	1	1
	1	1	1	1	1	1

MASK

	1	0	0	0	0	1
	1	0	0	0	0	1
1	0	0	0	0	0	1
1	0	0	0	0	0	1
	1	0	0	0	0	1
		1	0	0	0	1
			1	0	0	1
	1	0	0	0	0	1

V

	1	1	1	1	1	1
	0	0	0	0	0	0
1	0	0	0	0	0	0
1	0	0	0	0	0	0
	1	0	0	0	0	0
		1	0	0	0	0
			1	0	0	0
	1	1	1	1	1	1

H

	1	1	1	1	1	1
	1	0	0	0	0	1
1	0	0	0	0	0	1
1	0	0	0	0	0	1
	1	0	0	0	0	1
		1	0	0	0	1
			1	0	0	1
	1	1	1	1	1	1

Boundary

Figure 4-1: Illustration of boundary extraction

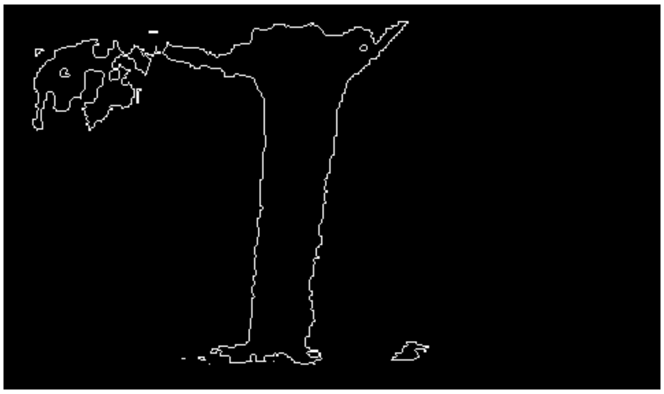


Figure 4-2: The boundary contour of the tree layer

This process is continued until the starting point is reached.

Since the contour is assumed to be 8-connected, 8 distinct directionals are possible, as shown in figure 4-3. These directionals can be represented with only 3 bits. Therefore, an n point contour can be losslessly described with $3n$ bits along with a few bits for the starting point. Chain codes, however, can be further compressed because the directionals are usually correlated. Quite often, a contour consists of a line or a smooth, slowly varying curve. In this case, the chain codes have a long run of a particular directional. We can use run length coding or Lempel-Ziv coding to reduce this redundancy in the directionals. Also, some directionals have a higher frequency of occurrence than others. Figure 4-4 shows a histogram of the directionals for the tree layer. We can exploit this unequal distribution by using huffman coding to assign codewords to input symbols depending on their probability of occurrence.

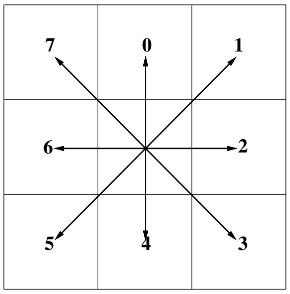


Figure 4-3: The directionals used in chain coding

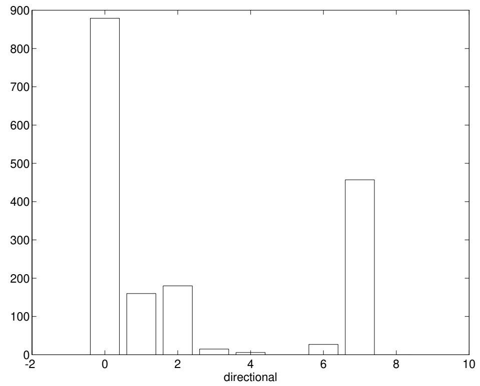


Figure 4-4: A histogram of the directionals for the tree layer

The encoding process, thus, involves extracting the boundary contour of the alpha map and then performing chain coding to compress the contour. If the boundary of an alpha map consists of several contours, which can happen if the object has holes or islands in it, each contour is encoded using the algorithm described above. The decoding process is even simpler because it only requires reconstructing the contour(s) from the chain codes and then filling the region within the boundary to obtain the original alpha map.

In most cases, obtaining the alpha map by filling the region within its boundary is very simple. We use a causal algorithm for region filling which scans the contour in a raster format and at each pixel, uses the previous pixels to determine whether or not the current pixel is within the boundary. However, for certain contours, such as open and linked ones, determining whether or not a pixel is within the boundary becomes quite ambiguous. The causal approach to region filling is extremely sensitive to errors. For example, if a pixel is filled incorrectly, this error propagates throughout the rest of the alpha map. Hence, an error correction algorithm is required.

We use a feedback algorithm for correcting errors in region filling. The algorithm is as follows: Once the boundary has been extracted by the encoder, it is filled and compared, as the filling progresses, with the original mask. If any error is detected, it gets recorded in an *error map*, and the corresponding pixel is corrected right away. Since the pixels that are still to be filled depend on this pixel(due to causality), correcting the error during filling prevents the error from propagating. The error map is encoded using an efficient method which is described below. The decoder can now reconstruct the original mask using the chain codes and the error information.

The error map contains information about the pixels where the boundary filling algorithm fails. As mentioned before, the boundary filling algorithm produces an error in the case of open or linked contours. Since the error map is just a binary map where the error pixels are indicated by 1s, a simple way to encode it is to use run length coding. However, higher compression can be achieved if we jointly encode the error map and the contour.

Joint encoding is possible because in the boundary filling algorithm, an error can occur only at a pixel that has a contour point as its neighbor. In fact, since the boundary filling algorithm scans the contour row by row, an error pixel must follow a contour point. Based on this important observation, we can modify the chain coding procedure to encode the contour along with the error map. Recall that in chain coding, the contour is traced to produce a stream of directionals. The modified chain coding procedure also traces the contour, but at each contour point, it checks the neighboring pixel(pixel on the right) in the error map for an error. If an error is found, a special codeword is emitted. Consequently, the resultant chain codes contain the stream of directionals, interspersed with the error information.

Our error correction algorithm has two advantages:

- It reduces the number of errors that occur in region filling because it prevents the error from propagating. Consequently, there is less error information to encode.

- It uses an efficient chain coding algorithm to encode the error information together with the contour.

The block diagrams of the encoder and decoder are given in figures 4-5 and 4-6, respectively. Using our binary alpha map encoding algorithm, we have been able to obtain a bit rate of about 2 bits/contour point for complex 8-connected contours. This bit rate includes the bits that represent the error information required for boundary filling. Eden *et al.* [7] have shown compression results of 1.2 bits/contour point for simple 4-connected contours (without boundary filling). Detailed results will be presented in the chapter on compression results.

Figure 4-5: The block diagram of the lossless alpha map encoder

Figure 4-6: The block diagram of the alpha map decoder

4.3 Lossy contour encoding

The contour encoding method described in the previous section performs lossless compression of alpha maps. It uses chain coding to obtain a compact representation of the boundary contours. However, if higher compression is desired, we have to tolerate some reduction in quality. In this section, we present a lossy contour encoding scheme called *morphological smoothing*. This method performs smoothing on the boundary contours before encoding them.

4.3.1 Morphological smoothing

Consider the boundary contour shown in figure 4-2. Clearly, the contour is not smooth, and there are several holes and islands in it. Such a contour is difficult to compress efficiently with chain codes. Significant improvement in compression can be obtained if we can smooth the contour and eliminate small holes and islands. Morphological operations can be effectively used for this purpose.

Morphological operations

Most morphological operations can be defined in terms of *erosion* and *dilation*. Let the sets X and G represent an object and a structuring element, respectively. Let G_x be the translation of G so that its origin is at x.
Then the erosion of X with respect to G is defined as the set of all points x such that G_x is a subset of X:

$$X \ominus G = \{x : G_x \subset X\} \tag{4.1}$$

and the dilation of X with respect to G is defined as the set of all points x such that X and G have a non-empty intersection:

$$X \oplus G = \{x : G_x \cap X \neq \phi\} \tag{4.2}$$

An illustration of erosion and dilation is given in figure 4-7. This figure demonstrates that erosion is a shrinking operation and dilation is an expansion operation.

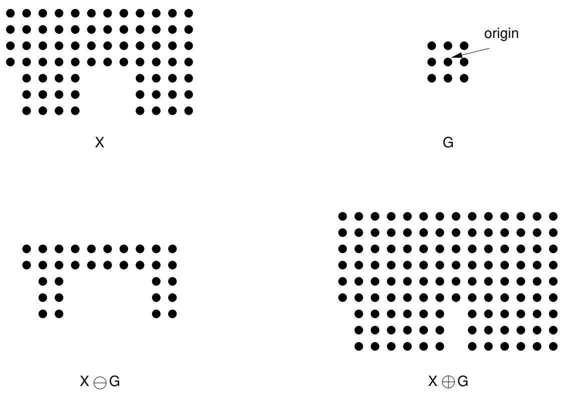


Figure 4-7: Erosion and dilation operations.

Erosion and dilation can be used to build more complicated morphological operations such as *open, close, etc.* [14].
The opening of X with respect to G is defined as a cascade of erosion and dilation operations on X with respect to G:

$$X \circ G = (X \ominus G) \oplus G. \tag{4.3}$$

Closing is the dual of opening:

$$X \odot G = (X \oplus G) \ominus G. \tag{4.4}$$

The open operation has the property of smoothing contours and suppressing small islands, whereas the close operation has the property of blocking narrow channels. Based on these properties, we define a smoothing operation which is a cascade of closing and opening:

$$\text{smoothing:} \quad (X \odot G) \circ G. \tag{4.5}$$

where G is the 3 x 3 structuring element shown in figure 4-7.

Our contour smoothing algorithm, performs this smoothing operation on the original alpha map to obtain a modified map. The boundary contour of this modified map is the smooth contour required for efficient chain coding. The result of applying the smoothing algorithm on the tree layer is given in figure 4-8. As one can see from this figure, the algorithm is very effective in smoothing the contour and in supressing holes and islands. Chain coding of this smooth contour results in a bit rate of about 1.4 bits/contour point. This is significantly lower than the 2 bits/contour point bit rate of the original contour.

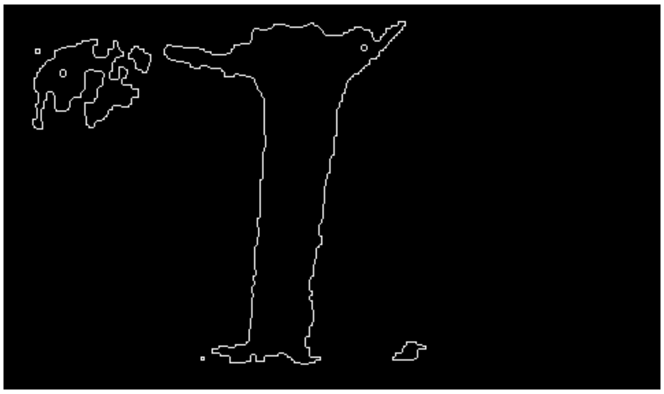


Figure 4-8: The tree contour obtained by morphological smoothing.

In this chapter, we presented lossless as well as lossy alpha map encoding algorithms. The lossless algorithm uses chain coding to encode the boundary contour of an alpha map. For lossy compression, the contour is smoothed with morphological operations and then encoded using chain coding. The lossy algorithm produces significant reduction in the bit rate; however, this reduction is achieved at the cost of increased distortion. In addition, the amount of distortion introduced by this kind of smoothing can not be controlled. A more efficient lossy scheme is to perform distortion controlled smoothing on the boundary contours. The method smooths only those contour pixels for which the distortion due to smoothing is less than some desired threshold. The distortion can be computed by measuring the error between the original and reconstructed frames. This method can potentially provide higher compression at acceptable quality loss. It, however, is computationally intensive. More research needs to be done for developing such an algorithm.

Recall that our object coding approach compresses the alpha and intensity maps of an object independently. Once the alpha map is encoded efficiently, we need to compress the intensity map. The next chapter describes several techniques for intensity map encoding.

Chapter 5

Coding of intensity maps

This chapter describes algorithms for compressing the intensity map of an object. The goal of intensity map encoding is to represent the intensities of the supported region with as few bits as possible. Since our object encoding algorithm encodes the alpha map, we have the flexibility to modify the unsupported pixels for improved intensity map encoding. We have already seen that JPEG can efficiently encode the interior and exterior blocks of the intensity map. The edge blocks, however, require special coding techniques. We have developed three methods for encoding the edge blocks: smooth filling, cosine filling, and region dependent transform coding. Smooth filling and cosine filling are both JPEG compliant because they produce DCT coefficients. Region dependent transform coding, on the other hand, is not JPEG compliant. It encodes each edge block using a transform that is designed based on of the shape of its supported region.

5.1 Smooth filling

As we saw in chapter 3, if we set the intensities of the unsupported pixels to a constant, for example zero, application of DCT results in poor compaction because of the edge between the supported and unsupported regions. Clearly, higher compression can be achieved by filling the unsupported region with values that reduce this edge. By using a filling approach, we can obtain high compression with JPEG. The process involves modifying the edge blocks by assigning proper intensities to the unsupported pixels and then encoding the modified blocks with JPEG.

One simple way of filling the unsupported pixels is to smoothly extend the supported region over to the unsupported region. This is the main idea behind smooth filling. We will see later that smooth filling not only improves compaction, but also provides reduction in the distortion introduced by quantization.

The algorithm recursively assigns each unsupported pixel the average of the neighboring supported pixel intensities. As a result of this algorithm, the modified block contains a smooth transition from the supported to the unsupported region, thereby improving compaction in the transform coefficients.

The smooth filling algorithm can be summarized as follows:

For each unsupported pixel located at position P_i,

- Find the average intensity of the supported pixels in a 3 x 3 window centered at P_i.
- Set the intensity at P_i to this average.
- If there are no supported pixels in the window, set the intensity of P_i to the intensity of the nearest supported pixel.

Iterate through this process until the unsupported region obtains a smooth intensity assignment.

Figure 5-1 gives a 1-D example of smooth filling. The first plot shows a composite input signal, which is obtained by setting the unsupported pixels to zero. The second plot shows the result of smooth filling. A 2-D illustration of smooth filling is shown in figure 5-2. Figure 5-3 shows the DCTs of the original and modified edge blocks. The improvement in compaction due to smooth filling is evident from figure 5-3(b) because most of the energy is present in low frequencies. Other simple filling techniques, such as filling the unsupported region with the mirror image of the supported region, can also be used. However, our experiments show that our smooth filling algorithm performs better than those techniques. The application of smooth filling to a real object, the tree layer, is shown in figure 5-4. Compression results will be presented in the next chapter.

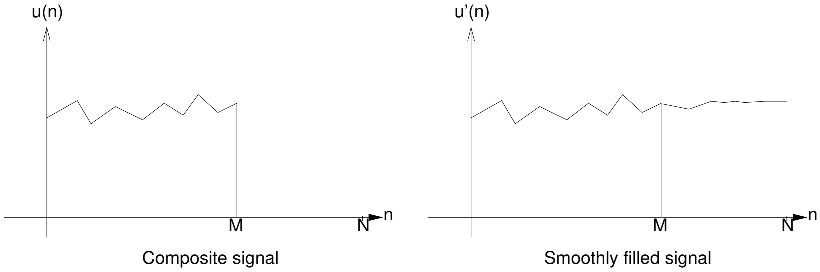


Figure 5-1: 1-D illustration of smooth filling

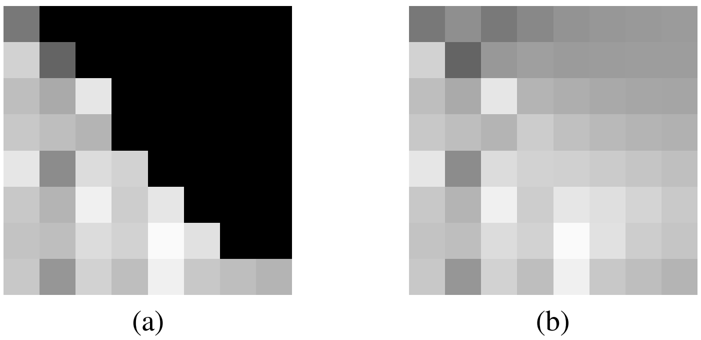


Figure 5-2: 2-D illustration of smooth filling. (a) composite edge block; (b) smoothly filled block

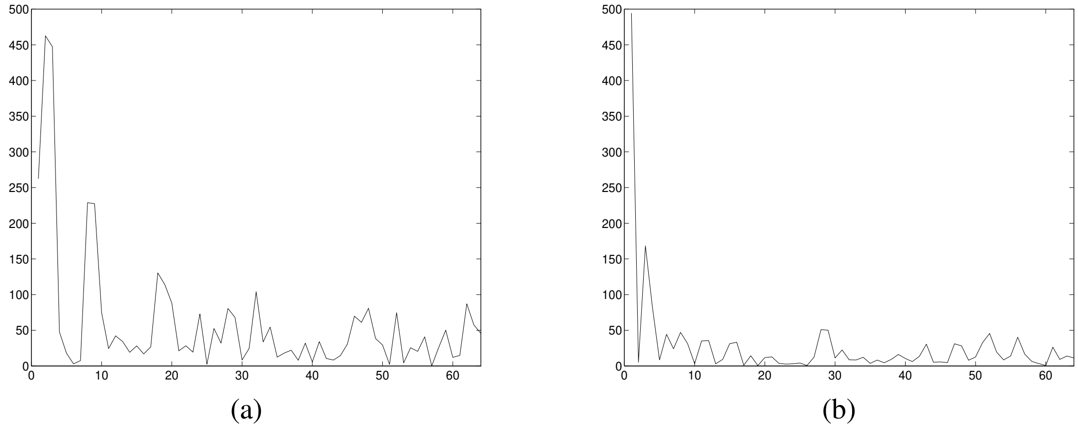


Figure 5-3: (a) DCT of the composite edge block; (b) DCT of the smoothly filled block. Note: the dc component has been attenuated by a factor of 3 for proper displaying.

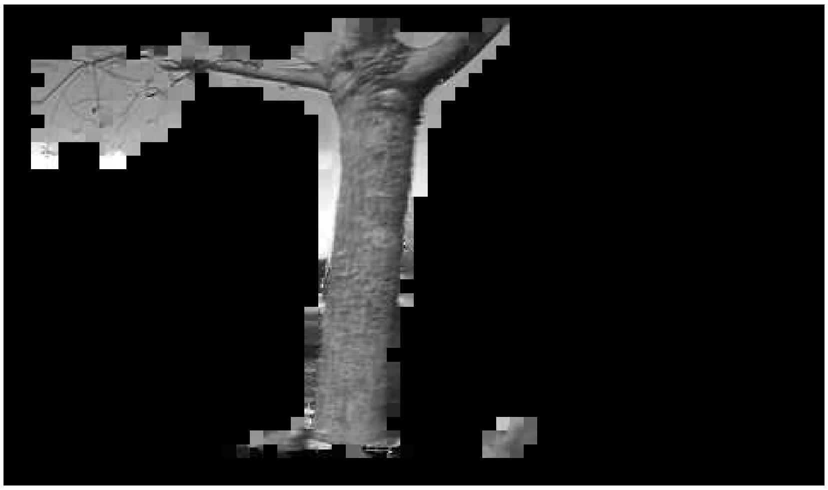


Figure 5-4: Smoothly filled tree layer

5.1.1 Advantages of smooth filling

Smooth filling improves compression of edge blocks in two ways:

1. By reducing the edge between the supported and unsupported regions, smooth filling reduces the energy spread in the transform domain. When the DCT is applied to the smoothly filled block, compaction is greatly improved because, as shown in figure 5-3, most of the energy is present in low frequencies.

2. Smooth filling tends to shift energy from high to low frequencies due to edge reduction, thus producing lower distortion when the filled block is encoded with JPEG. Recall that the JPEG algorithm involves two steps: transformation and quantization. In the first step, compaction is achieved with the DCT, while in the second step, bit rate reduction is achieved with quantization. Since the quantizer used in JPEG quantizes high frequencies more coarsely than low frequencies, smooth filling results in lower distortion.

5.1.2 A better compression scheme

We showed above that smooth filling can obtain a lower bit rate than that obtained by compressing composite maps with JPEG. There is, however, room for further improvement. Smooth filling has two important disadvantages:

- Filling is based only on neighboring pixels, that is, only local processing is performed. On the other hand, if all the supported pixels in the edge block are used in filling, higher compression can be achieved.

- The structure of the basis functions is not directly utilized in filling. If the filling method takes the structure of the basis functions into account, it might be possible to represent the filled block with only a few basis functions, thereby achieving high compaction.

The next section describes an improved compression scheme that does not have these drawbacks.

5.2 Cosine filling

5.2.1 Motivation

As we saw in the previous section, smooth filling provides a simple way of assigning intensities to the unsupported pixels that is compatible with standard coding schemes such as JPEG. However, it does not use the basis functions for filling and thus, does not provide very high compaction. Figure 5-5 illustrates this problem. In this example, the supported region consists of a single frequency cosine wave. The result of smooth filling is a smooth extension of this cosine wave. Unfortunately, this filling does not provide any improvement in compaction. On the contrary, it arbitrarily introduces energy in low frequencies.

In this example, we see that maximum compaction can be achieved if all of the energy is packed in a single frequency by extending the cosine wave itself. Figure 5-5(c) shows this extension. This example suggests a new filling approach; an approach that harmonically extends the supported region to the unsupported region. Since the DCT basis functions are cosines(see figure 2-2), harmonic extension can potentially provide better compaction.

It is important to note that the filling approach to be described in this section is very general. Given any transform, it provides a way to assign intensities to the unsupported region so as to achieve very high compaction. Even though the derivation given below is based on the DCT, solutions for other transforms can be obtained easily.

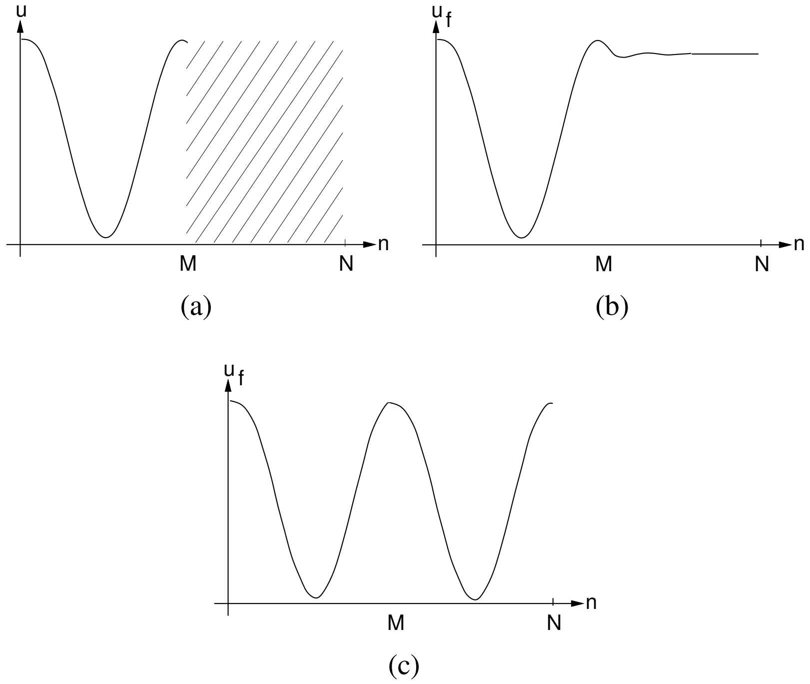


Figure 5-5: 1-D illustration of cosine filling (a) Input signal. The hatched region indicates the unsupported region; (b) smoothly filled signal; (c) cosine filled signal.

5.2.2 The filling problem

In order to develop a filling method that performs harmonic extension, let us first consider the filling problem.

Figure 5-6 shows the block diagram of the filling process. In the figure, u is the input edge block, u_f is the filled block, and v_f is the set of tranform coefficients. The problem can be stated as:

Given u, obtain u_f such that v_f is compact. [1]

[1]Actually, the general problem is to obtain u_f such that the entropy coder outputs as few bits as possible. This problem will be considered at the end of this chapter.

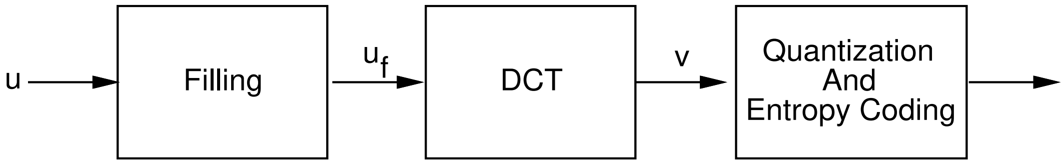


Figure 5-6: Block diagram of the filling process.

Here compaction represents the idea of "packing most of the energy in a few coefficients."

One conceptually simple method for solving this general problem is to exhaustively search the space of all possible u_fs. However, this method is obviously quite inefficient.

5.2.3 Direct approach to the filling problem

Consider the following problem:

Find v_f, a set of coefficients that is compact, such that its inverse DCT, u_f, is equal to the input, u, over the region of support. (5.1)

Although this problem is equivalent to the general filling problem, it represents a more direct approach because it tries to obtain the solution in the transform(frequency) domain rather than searching the space of all filled inputs and choosing the one that gives maximum compaction. It also allows for direct control of the distortion due to quantization.

Formulation of the direct approach

Let u be the 8 x 8 input edge block with M supported pixels ($0 < M < 64$). Let R be the region of support represented as a set of ordered pairs.

Let $\mathcal{B} = \{b_0, b_1, ..., b_{63}\}$ be the set of standard DCT basis functions, that is, b_is are cosines of different frequencies. Then u can be decomposed into b_is as:

$$u(n,m) = \sum_{i=0}^{63} v_i b_i(n,m) \tag{5.2}$$

where $v_i = i^{th}$ DCT coefficient. Since the DCT is a unitary transform, v_i can be obtained by

$$v_i = \sum_{n=0}^{7} \sum_{m=0}^{7} u(n,m) b_i(n,m) \tag{5.3}$$

Now consider the following set of functions:

$$\mathcal{B}' = \{b'_0, b'_1, ..., b'_{63}\} \tag{5.4}$$

where $$b'_i(n,m) = \begin{cases} b_i(n,m), & \text{when } (n,m) \in R \\ 0, & \text{otherwise} \end{cases}$$

Thus, $\mathcal{B}'$ is the set of masked DCT basis functions. Figure 5-7 shows a masked DCT basis function for a given region of support.

The problem stated in equation 5.1 can be formulated as:

Given u and $\mathcal{B}$, obtain v_f such that:

$$\underline{u} = B'\underline{v_f} \tag{5.5}$$

AND

$$\underline{v_f} \quad \text{is compact} \tag{5.6}$$

where

- B' is an Mx64 matrix with $\underline{b'_i}$ as columns.
- $\underline{u}$ and $\underline{b'_i}$ are Mx1 vectors containing the supported pixels of u and b'_i, respectively.
- $\underline{v_f}$ is the 64x1 vector of transform coefficients.

Note that equation 5.5 represents the constraint that u_f, the inverse DCT of v_f, should be equal to u over the region of support. Since B' is an Mx64 matrix ($M < 64$), equation 5.5 describes an under-determined system of linear equations, which has infinitely many solutions.[2] It is interesting to realize that these solutions correspond to the infinitely many ways of filling u.[3]

Before proceeding any further, we need an expression for compactness. As mentioned earlier, a set of coefficients is compact if most of the energy is present in a few coefficients.

Definition 1 *The compactness, $\mathcal{C}_k$, of $\underline{v}$ is the ratio of the energy in its k largest elements to its total energy.*

Let $\underline{v_s}$ = sort($\underline{|v|}$) (therefore, $|v_s(i)| \geq |v_s(j)|$; for $j > i$)

$$\mathcal{C}_k = \frac{\sum_{i=0}^{k-1} v_s(i)^2}{\sum_{i=0}^{63} v_s(i)^2} \tag{5.7}$$

Now, the problem in equation 5.1 can be stated as:

$$\max_{\underline{v_f}} \quad \mathcal{C}_k \tag{5.8}$$

under the constraint: $\underline{u} = B'\underline{v_f}$

If we can solve this constrained optimization problem, we will have an efficient solution to the general filling problem. Unfortunately, the optimization approach is not feasible because of the following reasons:

- The expression in equation 5.8 that needs to be maximized is non-linear. Since the gradient of the expression does not exist, there are no fast methods for finding the maximum.
- The presence of local minima makes the solution difficult to find.

We have developed a solution that employs an iterative algorithm for obtaining a valid, compact set of coefficients. While an optimal solution requires finding all the coefficients simultaneously, our iterative algorithm uses a greedy approach to estimate one coefficient at a time.

[2] Any system, of the form $\underline{u} = B'\underline{v}$, representing M equations and N unknowns ($M < N$) has infinitely many solutions if the rank of B' is M. We will show in the next section that the B' in equation 5.5 has rank M because it has M independent columns.

[3] We are not restricting the unsupported pixel intensities of u_f to be between 0 and 255.

5.2.4 Solution

Before presenting our algorithm, it is helpful to summarize the problem that we are solving. We want to perform harmonic filling of an edge block, u, that has M supported pixels. This problem is equivalent to directly finding 64 transform coefficients that, in the spatial domain, correspond to the M supported pixel intensities. Since $M < 64$, this requires solving an under-determined system, which always has infinitely many solutions. The solution that we are looking for is the one that has good energy compaction. We obtain the solution by using an iterative algorithm, which is described below.

Pseudocode:

$\underline{x} := \underline{u}$ *(initialization)*
repeat
$\underline{\tilde{x}} := c(i)\underline{b'_i}$ *(Find the $\underline{b'_i}$ that best approximates $\underline{x}$)*
$\underline{x} := \underline{x} - \underline{\tilde{x}}$ *(Update $\underline{x}$, get residue)*
until (energy($\underline{x}$)$< T$) *(check energy with threshold)*
where $:=$ stands for assignment.

As before, $\underline{u}$ and $\underline{b'_i}$ are vectors containing the supported pixels of u and b'_i, respectively. $\underline{c}$ is the resultant vector of transform coefficients.

Our iterative algorithm starts out by initializing the residue to equal the input. In each iteration, the masked basis function that best estimates the residue is found, and then the residue is updated. This process is repeated until the residual energy is less than some threshold. The frequency that best estimates the residue can be found as follows:

Let $e(j)$ be the error in representing the residue, $\underline{x}$, with only one function, $\underline{b'_j}$.

$$\text{Therefore,} \qquad e(j) = \min_{d(j)} ||\underline{x} - d(j)\underline{b'_j}||^2 \tag{5.9}$$

We can solve for $d(j)$ by expressing $e(j)$ as

$$e(j) = \min_{d(j)} \quad (\underline{x} - d(j)\underline{b'_j})^T(\underline{x} - d(j)\underline{b'_j}) \tag{5.10}$$

and then by setting the derivative of $e(j)$ w.r.t. $d(j)$ to 0.
After some simplifications, it can be shown that $d(j)$ is just the projection of $\underline{x}$ onto $\underline{b'_j}$:

$$d(j) = \frac{\underline{b'_j}^T\underline{x}}{\underline{b'_j}^T\underline{b'_j}} \tag{5.11}$$

The function that best approximates the residue is the one that has the minimum error. Let

$$i = \arg\min_j e(j) \tag{5.12}$$

then $c(i) := d(i)$, and $c(i)\underline{b'_i}$ is the best estimate of $\underline{x}$.

This iterative algorithm for obtaining transform coefficients provides high compaction because, at each stage, it uses a greedy approach of putting as much energy into one coefficient as possible. The algorithm does require much more computation than smooth filling, but fast implementations exist.

5.2.5 Improvements to cosine filling

As we mentioned before, transform coding of data involves two steps: transforming the input into uncorrelated (compact) coefficients; quantizing and entropy encoding these coefficients for actual reduction in bit rate. In our discussion of cosine filling, we have not yet addressed the issue of quantization. As one might expect, compression can be improved if the quantization process is integrated with cosine filling. For simplicity, let us restrict ourselves to the JPEG quantizer, keeping in mind that this may not be the best quantizer to use with cosine filling.

The JPEG quantizer, which is described in detail in Chapter 2, quantizes high frequencies more coarsely than low frequencies. The quantized coefficients are encoded using huffman and run length coding. Therefore, high compression is achieved for blocks that have most energy in low frequencies. The problem with the cosine filling algorithm is that it achieves compaction (most energy in few coefficients) at the cost of significantly increasing the randomness in the location of high energy coefficients. Since the high energy coefficients need to be transmitted (after quantization), increase in the entropy in the location of these coefficients directly translates into reduction in compression.

The performance of cosine filling can be improved if we utilize the properties of the JPEG quantizer to perform filling. This can be done in the following two ways.

1. Combine quantization and filling
   By integrating the quantization and filling processes, reduction in distortion can be obtained because it prevents the quantization error from accumulating. At each stage of this new algorithm, a greedy approach that finds the coefficient that best approximates the *quantized* residual is used. The algorithm is shown below:

   $\underline{x} := \underline{u}$ *(initialization)*
   repeat
   $\widehat{\underline{x}} := c(i)\underline{b'_i}$ *(Find the $\underline{b'_i}$ that best approximates $\underline{x}$)*
   $\underline{x} := \underline{x} - \widehat{\underline{x}}$ *(Update $\underline{x}$, get quantized residue)*
   until (energy($\underline{x}$)$< T$) *(check energy with threshold)*

   The $\underline{b'_i}$ that best approximates $\underline{x}$ can be obtained as:

   $$e(j) = \min_{d(j)} ||\underline{x} - d(j)\underline{b'_j}||^2 \tag{5.13}$$

   As before, $d(j)$ is the projection of x onto b'_j.

   $$d(j) = \frac{\underline{b'_j}^T \underline{x}}{\underline{b'_j}^T \underline{b'_j}} \tag{5.14}$$

   Let $\widehat{d(j)}$ be the quantized $d(j)$:

   $$\widehat{d(j)} = Q(d(j)) \tag{5.15}$$

   where Q is the quantizer.
   Now, let $\widehat{e(j)}$ be the quantized error:

   $$\widehat{e(j)} = ||\underline{x} - \widehat{d(j)}\underline{b'_j}||^2 \tag{5.16}$$

   and let

   $$i = \arg\min_j \widehat{e(j)} \tag{5.17}$$

then $c(i) := \widehat{d(i)}$, and $c(i)\underline{b'_i}$ is the best estimate of $\underline{x}$.

Apart from reducing the quantization error, this algorithm also provides us significant control over the filling process. At the end of each iteration, we know exactly how much mean square error (MSE) is introduced by the encoding process, and if the MSE is less than some threshold, we can terminate cosine filling right away, thereby reducing the processing time.

2. Reduce randomness in quantized coefficients
   This can be achieved by introducing quantization costs in the iterative cosine filling algorithm.

   Replace equation 5.17 in the above algorithm with:

$$i = \text{arg } \min_j w(j)\widehat{e(j)} \tag{5.18}$$

   where w is the cost function.
   Since we are using the JPEG quantizer, w can be chosen to be an increasing function of frequency (j). This effectively shifts energy towards low frequencies, thus improving JPEG compression.

The two modifications suggested above provide about 4% compression improvement over the original iterative algorithm. Detailed compression results for cosine filling will be given in the next chapter.

5.2.6 Illustrations of cosine filling

An example of cosine filling is shown in figure 5-8. The input edge block was obtained by masking an original block that only contained two frequencies. Smooth and cosine filled blocks are also shown in the same figure. Figure 5-9 shows the corresponding DCTs. Figure 5-10 shows the result of cosine filling the edge block in figure 5-2. As can be seen from all these figures, harmonic filling provides greater compaction than smooth filling. The result of cosine filling the tree layer is shown in figure 5-11.

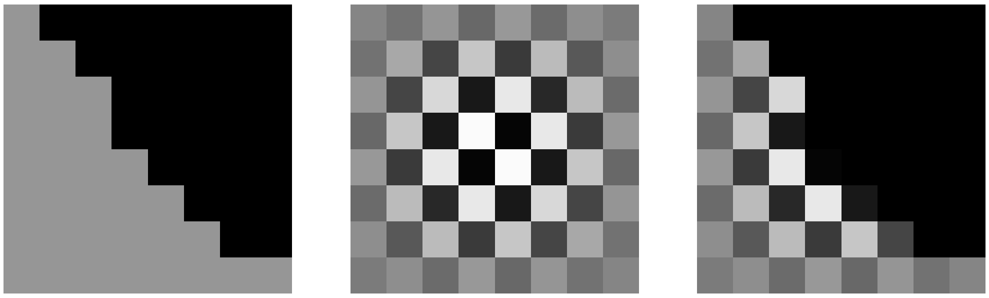
Recall that the objective of cosine filling is to obtain a compact [4] set of coefficients under the constraint that they represent the intensities in the supported region. We can evaluate the performance of our iterative cosine filling algorithm by using a compaction curve. From the definition of compactness, it is clear that the compactness of a method can be measured by computing the distortion that is introduced if only the k largest coefficients are used to reconstruct the input block. Therefore, a compaction curve is a plot of this distortion (averaged over all blocks) versus the number of coefficients used for reconstruction. Figure 5-12 shows the compaction curves for cosine filling, smooth filling, and JPEG on the composite edge block(edge block with unsupported pixels set to zero). The curves indicate that our iterative algorithm does indeed achieve very high compaction.

5.3 Region dependent transform coding

5.3.1 Motivation

As discussed in the previous sections, smooth filling and cosine filling modify the unsupported pixel intensities to obtain higher JPEG compression. These methods can be considered to be JPEG compliant in that they use the 8 x 8 DCT as transform. We can, however, approach this coding problem in a different way: instead of using 8 x 8 DCT, we can try to find an appropriate transform that only exists over the supported region of the input. The important idea is that such a transform does not require any filling of the edge block. To obtain a better understanding of the problem, let's consider the following 1-D case.

[4] Compactness($\mathcal{C}_k$) is defined to be the energy in the k largest coefficients

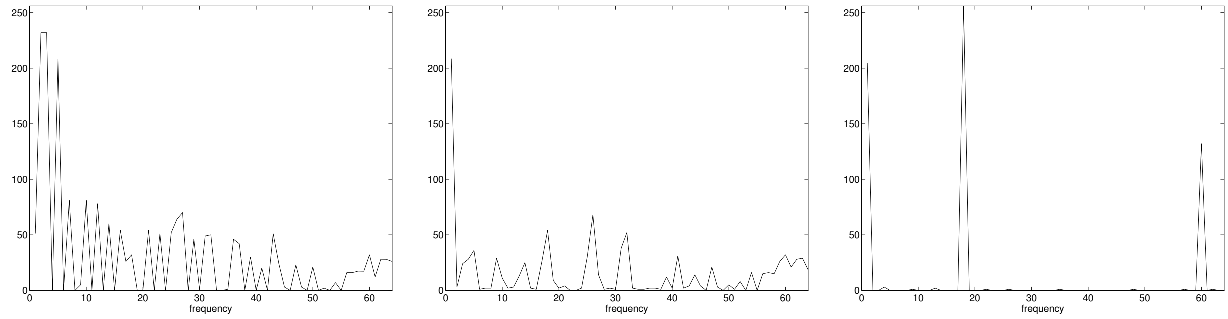


(a) (b) (c)

Figure 5-7: (a) Region of support; (b) b_{63}; (c) b'_{63}

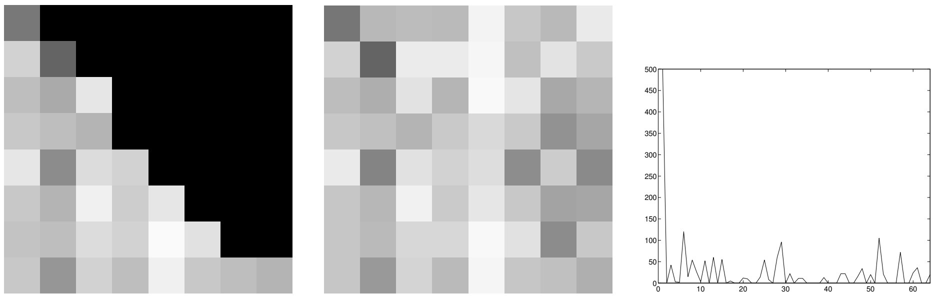
(a) (b) (c)

Figure 5-8: Illustration of cosine filling. (a) input edge block; (b) smoothly filled block; (c) cosine filled block



(a) (b) (c)

Figure 5-9: (a) DCT of edge block; (b) DCT of smoothly filled block; (c) DCT of block with harmonic extension. Note: the dc component has been attenuated by a factor of 5 for proper displaying.



(a) (b) (c)

Figure 5-10: Illustration of cosine filling. (a) input edge block; (b) cosine filled block; (c) DCT of the cosine filled block

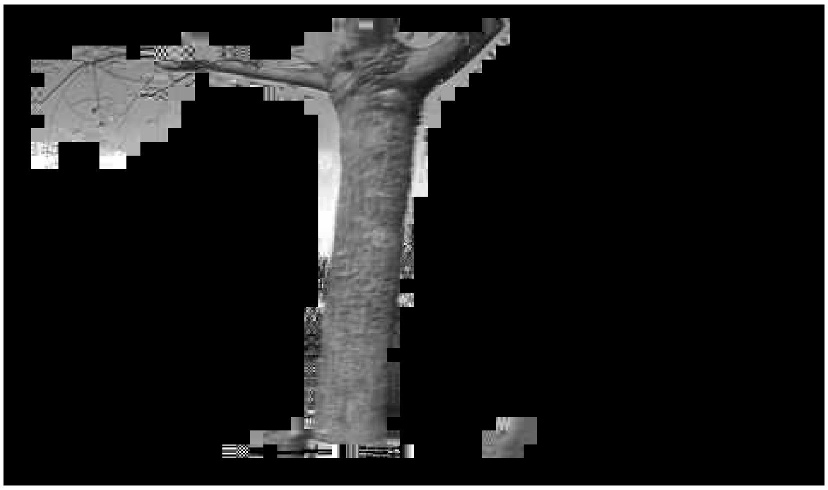


Figure 5-11: Cosine filled tree layer

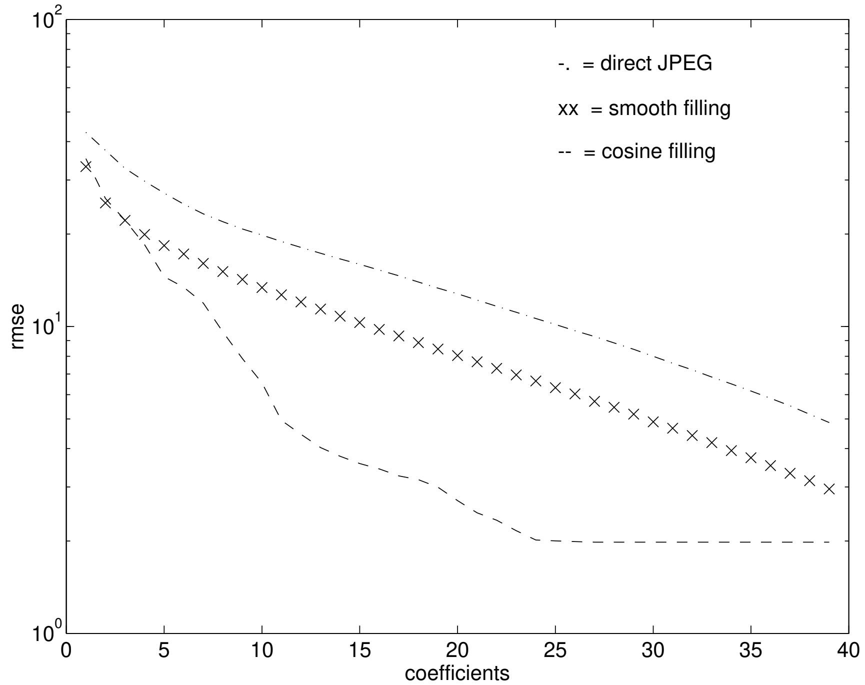


Figure 5-12: Compaction curve

1-D input

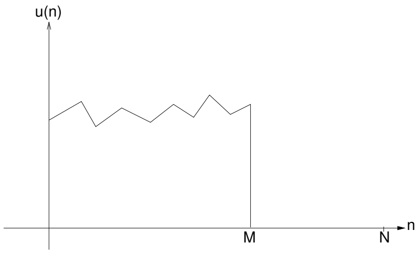


Figure 5-13: 1-D input signal

As shown in figure 5-13, the support [5] of the input is M points long. Assume that JPEG uses the N point DCT ($N > M$). Now, if we use smooth filling or cosine filling to encode this input, we will have to first fill the unsupported $(N - M)$ pixels and then perform the N pt. DCT on the filled input. Filling is required in this case because the support of the N pt. DCT is much longer than that of the input, and therefore, the region of analysis includes the intensity step between the supported and unsupported pixels of the input. The correct transform for this 1-D problem is the M pt. DCT. The basis functions, b_i, of the M pt. DCT are:

$$b_i(n) = a(i)cos(\frac{(2n+1)\pi i}{2M}), \qquad 0 < i, n < M - 1 \tag{5.19}$$

where $a(i)$ is the normalization constant.

2-D input

This idea of using DCT with support equal to that of the input can be extended easily to 2-D edge blocks with rectangular support. The edge block in figure 5-14(a) has an M_1xM_2 rectangular support. The correct transform for coding this block is the M_1xM_2 pt. DCT, and not the 8 x 8 DCT that JPEG uses. The basis functions, $b_{i,j}$, of the M_1xM_2 pt. DCT are:

$$b_{i,j}(n,m) = a(i,j)cos(\frac{(2n+1)\pi i}{2M_1})cos(\frac{(2m+1)\pi j}{2M_2}) \tag{5.20}$$

where $0 < i, n < M_1$ and $0 < j, m < M_2$. $a(i,j)$ is the normalization constant.
Note that it is always possible to obtain a DCT with separable support. Unfortunately, the edge blocks that we are dealing with have arbitrary, non-separable support (see figure 5-14(b)).

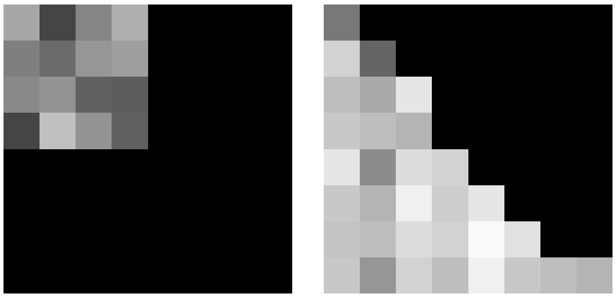
Problem: Develop a transform with arbitrary support.

Before discussing the solution, we will present some previous work on edge block coding.

5.3.2 Previous work

Earlier work on object coding was done by Eden *et al.*, Kocher and Leonardi, Gilge *et al.*, and Barnard *et al.*. Eden *et al.* [8] have described polynomial representations for object coding, but they only consider rectangular objects. Kocher and Leonardi [15] use polynomials of up to second order to represent arbitrarily shaped objects.

[5] The support of a function is defined to be the set on which the function assumes non zero values.



(a) (b)

Figure 5-14: Edge blocks with (a)rectangular region of support; (b) arbitrary region of support

Gilge *et al.* have presented region oriented generalized orthogonal transforms for object coding [5, 9, 10]. They suggest a way to orthogonalize polynomials of any order with respect to the shape of objects. These modified polynomials then act as basis functions for transform coding. Barnard *et al.* have presented a region-based discrete wavelet transform which uses an efficient signal extension algorithm for compressing arbitrary length signals [2, 3]. The method that we will describe in this section is similar to the one proposed by Gilge *et al.*.

5.3.3 Solution

We now describe a method for obtaining a transform with arbitrary support.

Notation

Let u be the input edge block with region of support, R, and let the number of pixels in R be M ($0 < M < 64$). Let $\mathcal{B} = \{b_0, b_1, ..., b_{63}\}$ be the set of standard DCT basis functions and $\mathcal{B}' = \{b'_0, b'_1, ..., b'_{63}\}$ be the set of masked DCT basis functions [6].
Let S_R be the M dimensional subspace of all possible inputs with support equal to R. Let S be the space of all 8 x 8 input blocks. Therefore,

$$u \in S_R \subset S \tag{5.21}$$

Region dependent basis

Claim 1 *$\mathcal{B}'$ spans S_R*

Proof: u can be decomposed into the DCT basis functions as :

$$\forall u \in S_R, \qquad u(n,m) = \sum_{i=0}^{63} v_i b_i(n,m) \tag{5.22}$$

where $v_i = i^{th}$ DCT coefficient.
Now, from the definition of $\mathcal{B}'$ (see equation 5.4), it follows that

$$\forall (n,m) \in R, \qquad b'_i(n,m) = b_i(n,m) \tag{5.23}$$

$$\forall (n,m) \notin R, \forall u \in S_R, \qquad b'_i(n,m) = u(n,m) = 0 \tag{5.24}$$

[6] Refer to equation 5.4 for definition of $\mathcal{B}'$.

Therefore, u can be represented by a linear combination of b_i's:

$$\forall u \in S_R, \qquad u(n,m) = \sum_{i=0}^{63} v_i b_i'(n,m) \tag{5.25}$$

∎

Claim 2 *Any spanning set for an M-dimensional subspace can be reduced to a basis by retaining any M independent elements.*

Proof: The proof for this standard linear algebra theorem can be found in [22]. ∎

From claims 1 and 2, it is follows that by retaining M independent elements of $\mathcal{B}'$, we obtain a basis, $\mathcal{B}_R$, for S_R.

Since $\mathcal{B}'$ has 64 elements, it has a large number of subsets with M independent elements. The natural question, then, is: which one of these subsets should be chosen as a basis? A good answer is to first assign elements to $\mathcal{B}'$ using the JPEG zigzag ordering(see figure 2-3(b)) and then choose the first M elements to form a basis for S_R. The JPEG zigzag ordering gives high priority to low frequency cosines. Implicit in this ordering is the assumption that input blocks usually have a large portion of energy in low frequencies.

The algorithm for obtaining $\mathcal{B}_R$ can be summarized as:

$$\begin{array}{cc} \mathcal{B} = \{b_0, b_1, \ldots, b_{63}\} & \text{(DCT basis in zigzag order)} \\ \downarrow & \\ \mathcal{B}' = \{b_0', b_1', \ldots, b_{63}'\} & \text{(Masked DCT basis functions)} \\ \downarrow & \\ \mathcal{B}_R & \text{(First } M \text{ indep. elements of } \mathcal{B}') \end{array}$$

Region dependent orthogonal basis

For high compression, it is desired to have as little correlation between transform coefficients as possible. An orthogonal basis is necessary for achieving this. $\mathcal{B}_R$ can be converted into an orthogonal basis, $\mathcal{Q}_R$, by using Gram-Schmidt orthogonalization.

Gram Schmidt Orthogonalization: Given $\mathcal{B}_R = \{b_0', b_1', \ldots, b_{M-1}'\}$, we construct $\mathcal{Q}_R = \{q_0, q_1, \ldots, q_{M-1}\}$ such that the q_is are orthogonal. The Gram-Schmidt procedure for constructing $\mathcal{Q}_R$ is as follows:

$$q_0 = b_0' \tag{5.26}$$

$$q_k = b_k' - (q_0^T b_k')q_0 - \ldots - (q_k^T b_k')q_k, \quad 0 < k < M \tag{5.27}$$

An orthonormal basis is obtained simply by normalizing the q_is.

The region dependent orthogonal basis for the region of support in figure 5-7(a) is shown in figure 5-15.

Region dependent orthogonal transform

From the region dependent orthogonal basis, $\mathcal{Q}_R$, we construct the region dependent orthogonal transform, Q_R, as follows:

Let $\underline{u}$ and $\underline{q_i}$ be the Mx1 vectors containing the supported pixels of u and q_i, respectively.

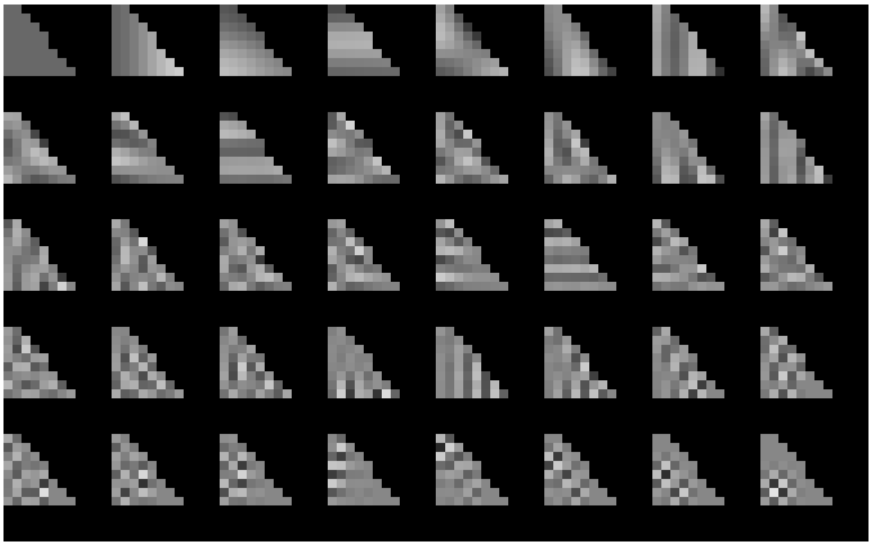


Figure 5-15: A region dependent orthogonal basis

The transform, Q_R, is the MxM matrix with the $\underline{q_i}$s as columns. The transform equations are

$$\underline{v} = Q_R^T \underline{u} \qquad \text{(forward transform)} \tag{5.28}$$

$$\underline{u} = Q_R \underline{v} \qquad \text{(reverse transform)} \tag{5.29}$$

where $\underline{v}$ is the Mx1 vector of transform coefficients. In the above equations, the basis is assumed to be orthonormal.

Figure 5-16 summarizes the procedure for obtaining Q_R from the DCT.

5.3.4 Advantages of region dependent transform coding

There are two main advantages of region dependent transforms:

- The support of the transform is exactly same as the support of the edge block, thus the intensity step between the supported and unsupported pixels is not encountered.
- The dimensionality of the transform is only M. This implies that a maximum of M coefficients will be required to be encoded instead 64(in the case of DCT), thereby providing higher compression.

Remark: The basis functions of the region dependent transform do not have to be transmitted because they can be obtained from the region of support of the input edge blocks, which is encoded in the form of alpha maps. The region dependent transform is, therefore, much more practical than the Karhunen-Loeve transform(KLT), which requires transmitting its basis functions.

To conclude this chapter, we summarize the three edge block coding methods. The edge blocks of a composite (alpha-intensity) map can not be directly compressed with JPEG because of the intensity step between the supported and unsupported regions. A simple way to improve compression is to perform smooth filling on the edge blocks. In smooth filling, the unsupported pixels are filled by smoothly extending the supported region. Further improvement can be achieved if we utilize the structure of the basis functions to perform filling. Since the DCT basis functions are cosines, a more efficient filling approach is to harmonically extend the supported region over the unsupported region. This is the main idea behind cosine filling. A completely different approach to edge block coding is to construct basis functions with support equal to that of the edge block. An elegant

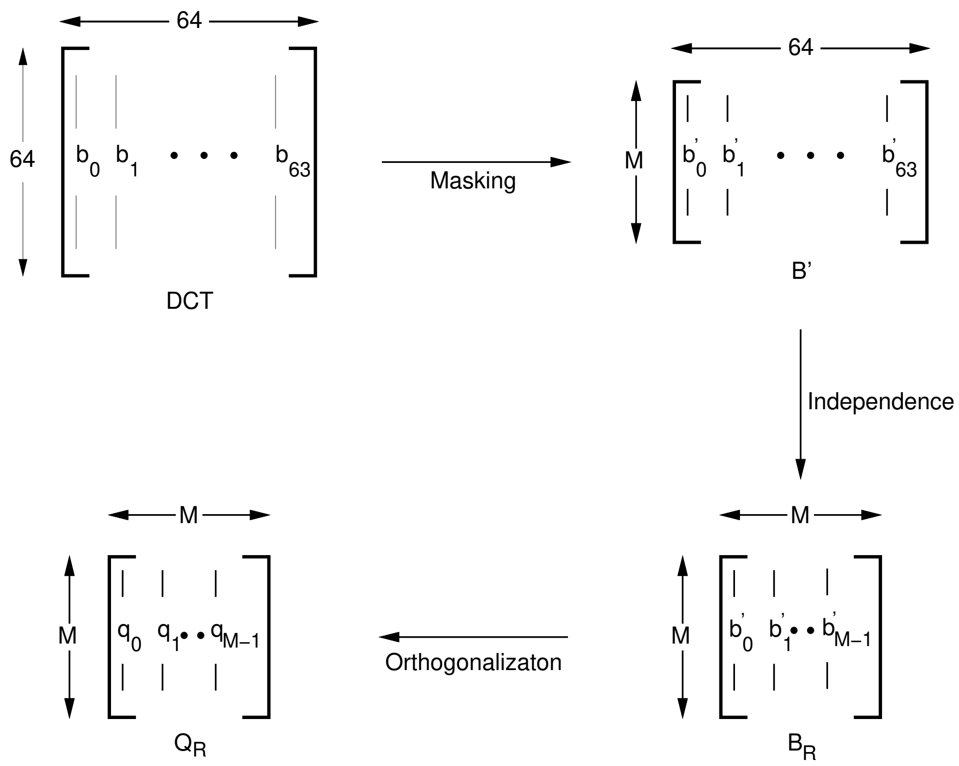


Figure 5-16: The procedure for obtaining the region dependent orthogonal transform from the DCT

algorithm for obtaining such a transform from the DCT gives rise to region dependent transform coding. Detailed analysis of these three techniques will be presented in the next chapter. Compression results for objects obtained from the flower garden sequence will also be provided.

Chapter 6

Compression results

This chapter is divided into three sections. In section 6.1 we analyze the performance of the edge block coding methods described in the previous chapter. We also compare these methods with the Karhunen-loeve transform(KLT). Section 6.2 provides an overall summary of our object coding algorithm. The results of applying our object coding algorithm on objects obtained from a real sequence (the flower garden sequence) are presented in section 6.3.

6.1 Analysis of edge block coding methods

We have put forth three edge block coding techniques: smooth filling, cosine filling, and region dependent transform coding. The theory behind these methods was presented in the previous chapter. In this section, we will measure the performance of these methods by considering some experimental results.

6.1.1 Edge block coding results

Figure 6-1 shows the compression curves for the three methods and the KLT. The y-axis indicates the average distortion(RMSE) and the x-axis indicates the average bits per pixel. As we mentioned in the background chapter, the KLT requires knowledge of the input covariance matrix. Since usually there is no *a priori* information about the input covariance matrix, it has to be computed from the input blocks. Due to this reason, an ensemble of blocks, each with the same (40 pixels wide) region of support, was used as input for generating the compression curves.

As expected, cosine filling performs better than smooth filling, but it does not perform as well as region dependent transform coding. This is because cosine filling achieves very high compaction at the cost of increased randomness in the location of high energy coefficients. This increase in randomness(or entropy) in the location of large coefficients adversely affects the compression performance of cosine filling. In section 5.2.5 we suggested some improvements to cosine filling such as using quantizer dependent cost functions in the iterative cosine filling algorithm. However, even with these improvements, we find that region dependent transform coding performs better cosine filling. The curves also indicate that the performance of region dependent transform coding is very similar to that of the KLT [1].

The compression curves for the edge blocks of the tree layer are shown in figure 6-2. Once again, region dependent transform coding achieves the highest compression.

[1]The curves indicate that for high bit rates, region dep. transform coding gives better results than the KLT. This is because we used the JPEG quantizer for all four methods. Before applying the JPEG quantizer, the KLT coefficients were arranged in decreasing order of their variances, which are given by the eigenvalues of the input covariance matrix. However, for a completely fair comparison, variance-based quantizers must be designed for each method.

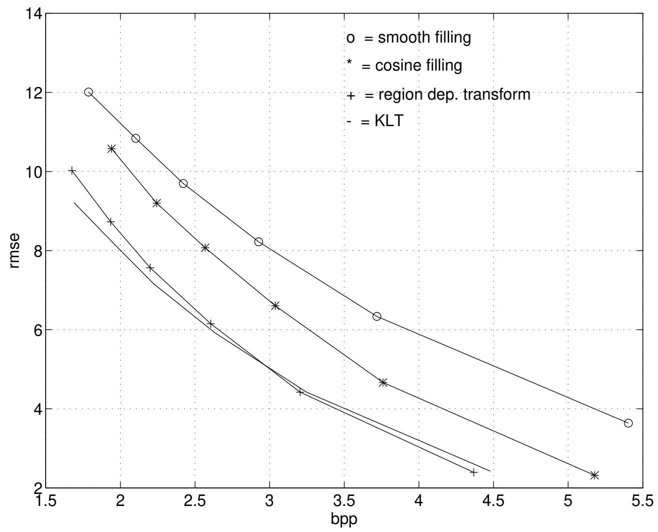


Figure 6-1: Compression curve

6.1.2 Comparison of edge block coding methods

So far we have only considered the actual bit rate produced by the edge block coding methods. However, the processing speed (and hence the complexity of implementation) is of great importance in a practical compression system. Obviously, there is a trade-off between the achievable compression and the processing speed of a compression system. In this section we will analyze this trade-off for various edge block coding methods.

Table 6.1 gives the relative compression rates and processing speeds for smooth filling, cosine filling, and region dependent transform coding. The compression rates in the table directly follow from the compression curves given earlier. As one might expect, the encoding process of smooth filling is the fastest. Cosine filling and region dependent transform coding have comparable encoding speeds; however, there is a vast difference in their decoding speeds. Recall that both smooth filling and cosine filling use the 8 x 8 DCT as transform. These two filling methods can be considered to be JPEG compliant because their decoding only requires JPEG decompression. Since the DCT has an efficient hardware implementation, JPEG decompression is extremely fast. On the other hand, for region dependent transform coding, the decoder is as computationally intensive as the encoder. The decoding process for region dependent transform coding is at least an order of magnitude slower than that for the two filling methods.

Method	Compression	Encoding	Decoding
Smooth filling	low	fast	very fast
Cosine filling	high	slow	very fast
Region dep. transform coding	very high	slow	slow

Table 6.1: A comparison of the edge block coding methods

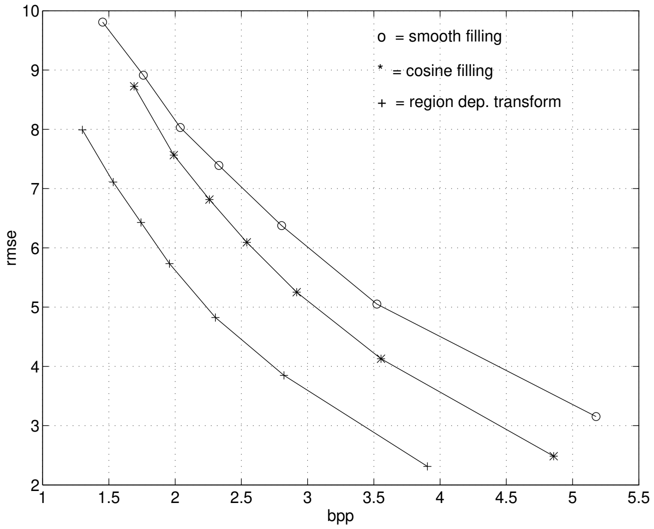


Figure 6-2: Compression curve for the edge blocks of the tree layer

6.2 Summary of the object coding algorithm

6.2.1 Encoder

The block diagram of the encoder is shown in figure 6-3. An object is represented by an alpha map and an intensity map. The alpha map is encoded with a contour encoding algorithm, which is based on chain coding. The intensity map is first partitioned into 8 x 8 blocks. The interior and exterior blocks are encoded directly with JPEG, while the edge blocks are encoded with special coding methods. There are three edge block coding methods: smooth filling, cosine filling, and region dependent transform coding. Either one of these methods can be used to encode the edge blocks. The overall performance of this object coding algorithm will be analyzed in the section on results.

6.2.2 Decoder

The decoding process is shown in figure 6-4. The chain codes are decoded using a contour decoding algorihm. The contour is then filled to obtain the alpha map. The alpha map used to decode the encoded intensity map. The exterior and interior blocks are simply decoded with the standard JPEG decompression algorithm. If smooth filling or cosine filling is used in the encoder, the decoding of edge blocks simply requires standard JPEG decompression; otherwise, region dependent transform decoding has to be performed. As one can see, the two filling methods have an extremely simple and fast decoder. A simple decoder is desirable in many coding applications, such as television broadcasting. For these applications, the filling methods are much more attractive than region dependent transform coding.

6.3 Object coding results

In our experiments, we encoded 30 frames of the MPEG flower garden sequence shown in figure 1-2. The sequence represents one second of color video at NTSC resolution. This sequence was first decomposed into objects shown in figure 1-3. The objects were then encoded with our object coding methods. Using the smooth filling algorithm, we obtained bit rates of 960 kbits and 560 kbits at average distortions of 5.7 rmse/pixel and 7.6 rmse/pixel, respectively. Applying JPEG directly to the composite maps, which are obtained by setting

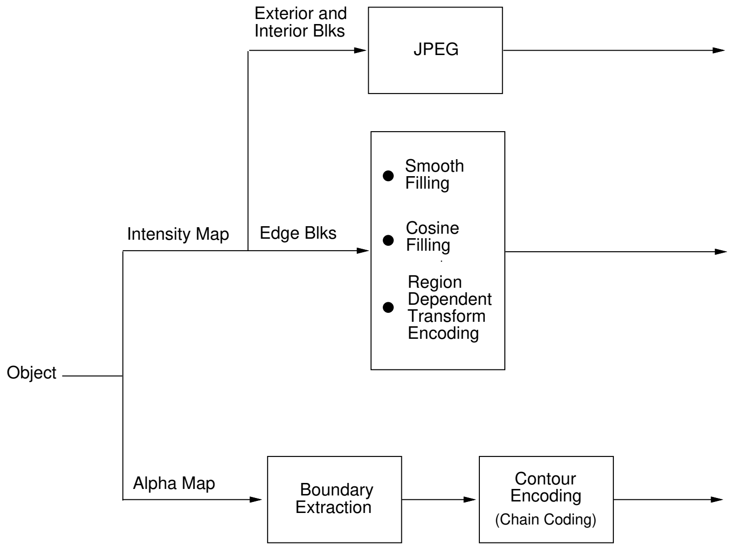


Figure 6-3: Block diagram of the encoder

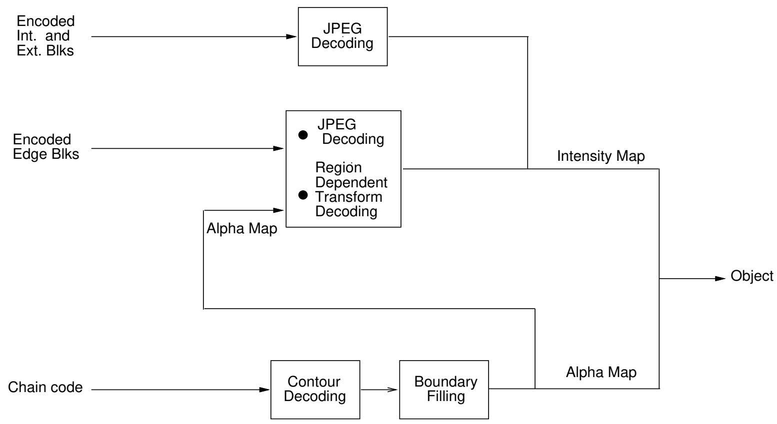


Figure 6-4: Block diagram of the decoder

the unsupported pixels to zero, resulted in bit rates of 1.54 Mbits and 1.04 Mbits at average distortions of 6.5 rmse/pixel and 9.5 rmse/pixel, respectively. Cosine filling gave about 6% improvement over smooth filling, whereas region dependent transform coding provided about 12% improvement over smooth filling.

The alpha maps were losslessly encoded at 50 kbits. As mentioned in chapter 4, we can reduce the bit rate by smoothing the alpha maps using morphological operations. Our morphological smoothing algorithm encoded the alpha maps at 25 kbits. Motion parameters for the sequence required only about 40 kbits.

The experimental results indicate the improvement in compression due to our three object encoding algorithms: smooth filling, cosine filling, and region dependent transform coding. These results can also be used to compare object-based sequence coding with MPEG. As we saw above, an object-based coder using the layered representation can compress the flower garden sequence at less than 1 Mbits/second. MPEG, on the other hand, requires about 2 Mbits/second at approximately the same distortion. These numbers clearly show that the object-based coder performs much better than MPEG.

Chapter 7

Conclusions and future work

7.1 Conclusions

This thesis proposes and describes techniques for compressing arbitrarily shaped objects. Efficient algorithms for coding objects are essential in many image coding systems that rely on mid-level vision techniques, such as segmentation. The object coding methods presented in this thesis can be of great of use in such systems.

In our representation, an object consists of two maps: an alpha map that represents the region of support and an intensity map that represents the intensity profile of the supported pixels. The alpha and intensity maps can be represented jointly as a composite map by setting the unsupported pixels in the intensity map to some constant, for example zero. When the composite map is encoded with standard still-image coding methods, such as JPEG, it results in poor compression. As we showed in chapter 3, this is because the intensity step between the supported and unsupported regions causes poor energy compaction in the transform domain. Our approach to encoding objects is to compress both the intensity map and the alpha map separately. This way we can exploit the properties of the two maps more efficiently.

Our alpha map encoding algorithm is described in chapter 4. We present lossless as well as lossy techniques for compressing alpha maps. Both of these techniques represent an alpha map with its boundary contour. The lossless algorithm encodes the boundary contour using chain coding. Chain coding provides a simple, compact description of the boundary contour. The decoder for this encoding algorithm is also very simple. It reconstructs the boundary contour from the chain codes and then fills the contour to obtain the original alpha map. However, in certain cases, it was found that the boundary contour is not sufficient for representing the alpha map, that is, the contour filling process produced some errors. We describe an error correction algorithm that encodes the error information along with the chain codes. Moreover, this algorithm reduces the amount of error information that needs to be transmitted.

Chapter 4 also describes our lossy alpha map encoding algorithm. This algorithm first performs smoothing on the boundary contour of the alpha map and then encodes the smooth contour with chain codes. Morphological operations, such as opening and closing, are used for smoothing the boundary contour.

In chapter 5, we present three intensity map encoding methods: smooth filling, cosine filling, and region dependent transform coding. All of these methods compress the interior and exterior blocks of the intensity map with JPEG. The methods, however, differ in their approach to coding of edge blocks. In smooth filling, the supported region is smoothly extended into the unsupported region. This improves compression by reducing the energy spread in the transform domain. Further improvement in compression is obtained by performing filling using the structure of the transform basis functions. This is the idea behind cosine filling. Cosine filling assigns intensities to the unsupported pixels by harmonically extending the supported region. Although the cosine filling algorithm derived in chapter 5 is based on the DCT, the algorithm is applicable with any other transform.

The third edge block coding method uses a completely different approach. Instead of using the DCT, it constructs transforms based on the shape of the supported region in edge blocks. This method is called region dependent transform coding. An elegant algorithm for obtaining such transforms is presented in chapter 5.

Finally, compression results are presented in chapter 6. Among the three methods, region dependent transform coding achieved the highest compression, followed by cosine filling. We also analyzed the three methods on the basis of their processing speed. As expected, the encoding process for smooth filling is faster than those for cosine filling and region dependent transform coding. Both smooth filling and cosine filling have an extremely simple decoding process because they are based on the DCT and are compliant with JPEG, which has very fast hardware implementations. On the other hand, the decoding process for region dependent transform coding is at least an order of magnitude slower than that for the two filling methods.

Many practical systems, for example video broadcasting, require a fast decoder. For these systems, the filling methods, cosine filling in particular, give the best trade-off between compression rate and processing speed.

7.2 Suggestions for future work

The work presented in this thesis can be continued in two ways. The lossy alpha map encoder in chapter 4 performs contour smoothing to improve compression. Unfortunately, the distortion caused by smoothing can not be controlled. A distortion controlled smoothing algorithm would not only improve compression, but also preserve quality. One way of implementing such an algorithm is to smooth only those pixels for which the distortion introduced by smoothing is below some threshold. One could measure distortion by computing the error between the original and synthesized frames.

Further research is required to investigate alternative techniques for improving the cosine filling algorithm. Our cosine filling algorithm fills the unsupported region in a way that makes the transform coefficients as compact as possible. However, the most general filling problem is to perform filling in a way that maximizes the compression. One solution to this problem is to extend the current implementation of cosine filling to include the quantization process in the iterative algorithm.

Bibliography

[1] E. H. Adelson. Layered representation for image coding. In *Vision and Modeling Technical Report 181*, The MIT Media Lab, 1991.

[2] H. Barnard, J. Weber, and J. Biemond. Efficient signal extension for subband/wavelet decomposition of arbitrary length signals. *Proc. SPIE: Visual Comm. and Image Proc.*, 2094:966–975, 1993.

[3] H. Barnard, J. Weber, and J. Biemond. A region-based discrete wavelet transform for image coding. *To appear in Eusipco*, 1994.

[4] T. C. Bell, J. G. Cleary, and I. H. Witten. *Text Compression*. Prentice Hall, Englewood Cliffs, NJ 07632, 1990.

[5] M. J. Biggar, O. J. Morris, and A. G. Constantinides. Segmented-image coding : performance comparison with the discrete cosine transform. *IEE Proc. Part F*, 135(2):121–132, April 1988.

[6] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. John Wiley and Sons, Inc., New York, 1991.

[7] M. Eden and M. Kocher. On the performance of a contour coding algorithm in the context of image coding. part i: Contour segment coding. *Signal Processing*, 8:381–386, 1985.

[8] M. Eden, M. Unser, and R. Leonardi. Polynomial representation of pictures. *Signal Processing*, 8(4):385–393, June 1986.

[9] U. Franke, R. Mester, and T. Aach. Constrained iterative restoration techniques: A powerful tool in region oriented texture coding. *Signal Processing IV: Theories and Applications*, pages 1145–1148, 1988.

[10] M. Gilge, T. Engelhardt, and R. Mehlan. Coding of arbitrarily shaped image segments based on a generalized orthogonal transform. *Signal Processing: Image Communication*, 1:153–180, 1989.

[11] R. C. Gonzalez and P. Wintz. *Digital Image Processing*, pages 244–265. Addison-Wesley Publishing Company, Reading, MA, 1977.

[12] D. A. Huffman. A method for the construction of minimum redundancy codes. In *Proceedings of the IRE*, volume 40, 1952.

[13] ISO-IEC/JTC1/SC29/WG11. Mpeg test model.

[14] A. K. Jain. *Fundamentals of Digital Image Processing*. Prentice Hall, Englewood Cliffs, NJ 07632, 1989.

[15] M. Kocher and R. Leonardi. Adaptive region growing technique using polynomial functions for image approximation. *Signal Processing*, 11(1):47–60, July 1986.

[16] J. S. Lim. *Two-Dimensinal Signal and Image Processing*. Prentice Hall, Englewood Cliffs, NJ 07632, 1990.

[17] B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Image Understanding Workshop*, pages 121–130, 1981.

[18] H. G. Mussman, M. Hotter, and J. Ostermann. Object-oriented analysis-synthesis coding of moving images. *Signal Processing:* Image Communication *1*, pages 117–138, 1989.

[19] Majid Rabbani and Paul W. Jones. *Digital Image Compression Techniques*. SPIE - The International Society for Optical Engineering, Washington, 1991.

[20] K. R. Rao and P. Yip. *Discrete Cosine Transform*. Academic Press Inc, 1990.

[21] A. Rosenfeld and A. C. Kak. *Digital Picture Processing*, pages 108–150. Academic Press, New York, 1976.

[22] Gilbert Strang. *Linear Algebra and Its Applications*. Academic Press, 1980.

[23] C. W. Therrien. *Decision estimation and classification*. John Wiley and Sons, New York, 1989.

[24] J. Y. Wang and E. H. Adelson. Layered representation for image sequence coding. In *IEEE ICASSP*, volume 5, pages 221–224, Minneapolis, Minnesota, April 1993.

[25] J. Y. Wang and E. H. Adelson. Layered representation for motion analysis. In *IEEE Conf. Computer Vision Pattern Recog.*, pages 361–366, New York, June 1993.

[26] J. Y. A. Wang, Edward H. Adelson, and Ujjaval Desai. Applying mid-level vision techniques for video data compression and manipulation. In *Proc. SPIE on Digital Video Compression on Personal Computers: Algorithms and Technologies*, volume 2187, San Jose, California, 1994. SPIE.

[27] T. A. Welch. A technique for high-performance data compression. 17:8–19, 1984.

[28] I. H. Witten, R. M. Neal, and J. G. Cleary. Arithmetic coding for data compression. 30(6):520–540, 1987.