Edge Based Video Image Compression for Low Bit Rate Applications

by

Marcelo Mikiyo Mizuki

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

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 1996

© Massachusetts Institute of Technology 1996. All rights reserved.

Author ........................................
Department of Electrical Engineering and Computer Science
August 9, 1996

Certified by ....................................
Berthold K. P. Horn
Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by .................................
Frederic R. Mogenthaler
Chairman, Departmental Committee on Graduate Students

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARCHIVES

OCT 1 5 1996
Edge Based Video Image Compression for Low Bit Rate Applications
by
Marcelo Mikiyo Mizuki

Submitted to the Department of Electrical Engineering and Computer Science on August 9, 1996, in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering

Abstract

This thesis examines an edge based video image compression scheme for low bit rate applications. In the proposed system, both the static coding and the interframe coding rely on contour information. In the case of the static coding algorithm, the contour method is used for image quality reasons, whereas in the interframe coding algorithm, contour information is used to reduce the computational requirements of the motion estimation procedure. This thesis will discuss the effectiveness of such methods for video image compression and some of their advantages and disadvantages.

Thesis Supervisor: Berthold K. P. Horn
Title: Professor of Computer Science and Engineering
Acknowledgments

I would first like to thank my thesis advisor Professor Berthold Horn for his extremely helpful advice and support during the completion of the research work. I am very grateful to Dr. Ichiro Masaki, who worked closely with me on the project and provided invaluable advice and encouragement. I would also like to thank Professors Anantha Chandrakasan and Charlie Sodini for their assistance with technical matters.

I wish to give special thanks to my friend Ujjaval Desai, who contributed greatly to the static algorithm and helped me solve my problems whenever he could. He not only provided great moral support but also took time to review this thesis. Jason Bergendahl, J.P. Mellor, and Stanislav Frumkin were also very willing to provide insight on various issues.

Many thanks to the staff members in the A.I. Lab who simplified tasks and helped me solve hardware and computer related problems. Ron Wilken helped me to find parts and to build essential setups. Bruce Walton sorted out computer and network related problems. Laurel Simmons helped me with equipment and procedural problems. In addition, many other members of the lab were very helpful and courteous.

I would like to extend my very deep gratitude to the National Science Foundation for supporting me under a Graduate Research Fellowship for the past two years.

Finally, I would like to thank my family. I would like to thank my brother Alex, for being a great friend and positive influence, and my parents, who have done so much for me and who continue to inspire me to become a better person.
Contents

1 Introduction .................................................. 13

2 Static Image Coding ........................................... 16
   2.1 The Algorithm ............................................ 18
   2.2 The Encoder ............................................... 19
      2.2.1 Edge Detection ....................................... 21
      2.2.2 Possible Improvements to the Edge Detection Procedure  24
      2.2.3 Contour Coding ....................................... 30
      2.2.4 Contour Intensity Data Extraction and Coding ............ 32
      2.2.5 Residual Coding ...................................... 37
      2.2.6 Entropy Coding ...................................... 38
      2.2.7 Encoder Block Diagram ............................... 39
   2.3 The Decoder ............................................... 39
   2.4 Algorithm Performance and Results ....................... 42
   2.5 Conclusions .............................................. 45

3 Interframe Coding ............................................. 51
   3.1 Introduction to Block Matching ............................ 52
   3.2 Binary Block Matching ................................... 54
   3.3 Experimental Results .................................... 59
   3.4 Practical Implementation of Block Matching ............... 61
   3.5 Design Issues ............................................ 65
   3.6 Circuit Components ...................................... 68
3.7 Comparison Between Conventional and Binary Block Matching  
3.8 Conclusions  

4 Video Algorithm Bit Rates  
4.1 Image Resolution and Frame Rate Constraints  
4.2 Issues in Static Coding  
4.3 Issues in Motion Coding  
4.4 Conclusions  

5 Conclusions  
5.1 Results and Conclusions  
5.2 Further work  

A Circuits Used in Design of Processing Elements
List of Figures

1-1 Illustration of the frames involved in the video coding process. .... 14

2-1 General illustration of the encoding procedure. .................. 20
2-2 General illustration of the decoding procedure. .................. 20
2-3 Illustration of the thinning procedure. ......................... 23
2-4 Location of pels in the gradient magnitude map \( G_{\text{mag}}(i,j) \) used in the thinning procedure. ....................... 24
2-5 Illustration of the effects of poor edge detection on image reconstruction. (a) Edge map using thinning and thresholding procedure described in Desai, Mizuki et al. [7]. PSNR = -28.3 dB. (b) Edge map using slightly relaxed thinning and thresholding procedure. PSNR = -26.7 dB. (c) Reconstructed image using edge map in (a). (d) Reconstructed image using edge map in (b). ......................... 25
2-6 Illustration of histogram equalization. (a) Original image. (b) Histogram preprocessed image. (c) Intensity histogram of the original image. (d) Histogram of the image in (b). ......................... 27
2-7 (a) Result of edge detection without histogram equalization. (b) Result of edge detection with histogram equalization. ......................... 28
2-8 Result of edge detection using adaptive thresholding technique. .... 29
2-9 Comparison of edge detection techniques. (a) Result of edge detection using histogram equalized image. (b) Result of edge detection using adaptive threshold. ......................... 29
2-10 (a) Possible directions for the nearest neighbors in 8-connected edge maps. (b) Possible directions for the nearest neighbors in 4-connected edge maps. 30

2-11 Conversion of an edge from 8-connected to 4-connected. 31

2-12 Example of regular chain coding procedure and differential chain coding procedure. Directional numbers follow the convention in figure 2-10(b). 31

2-13 Demonstration of points of interest in the encoding procedure. (a) Diagram illustrating the locations of interest. (b) Actual picture of points of interest in the image “Lenna” 32

2-14 Reason why pels immediately adjacent to the contour might not be representative of the points in the texture section between contours. 33

2-15 Determination of the intensity values of pels of interest. (a) Location of pels of interest with intensity values X and Y to be determined. (b) Horizontal pel values used in the determination of $X_H$. (c) Illustration of $X_H$ the value obtained from a linear combination of the horizontal points. (d) Vertical pel values used in the determination of $X_V$. (d) Illustration of $X_V$ the value obtained from a linear combination of the vertical points. 35

2-16 Alternative method used to obtain the intensity values of pels of interest. 36

2-17 Intensity coding procedure. 36

2-18 Use of block mean information at the decoder. 38

2-19 General illustration of the Huffman encoding procedure. 39

2-20 Block diagram of color static encoder. 40
2-21 Interpolation procedure. (a) Points of interest with intensity values X and Y used in the horizontal interpolation procedure. (b) Recovery of horizontal component of the intensity values between contours using the values X and Y. (c) Points of interest with intensity values X and Z used in the vertical interpolation procedure. (d) Recovery of vertical component of the intensity values between contours using the values X and Z.

2-22 Relative performance of the contour based static image coding algorithm versus JPEG. (a) Original 256×256 “Girl2” image. (b) Reconstructed image (PSNR = -21.5 dB, 0.16 bpp). (c) JPEG result (PSNR = -18.6 dB, 0.28 bpp). (d) JPEG result (PSNR = -19.5, 0.27 bpp).

2-23 Color 256×256 images. The left column contains the original pictures, the middle column contains the edge maps used for the static encoding process, and the right column contains the reconstructed color images.

2-24 (a) Compressed image “Lenna” at 0.24 bpp. (b) Compressed image “Girl2” at 0.095 bpp.

3-1 Illustration of the block matching algorithm. (a) Current block. (b) Search window and location of the minimum distortion position (from Pirsch [12]).

3-2 (a) Conventional block matching procedure. (b) Binary block matching procedure.

3-3 A conceptual block diagram of motion compensation using binary block matching (BBM).

3-4 Blocks used in the interpolation procedure and the notation for the vectors.
3-5 Images reconstructed from the motion vectors. (a) Original frame to be predicted. (b) Binary Block Matching prediction without interpolation of motion vectors of invalid blocks (normalized MAE = 10.0). (c) Binary Block Matching prediction with interpolation of motion vectors (normalized MAE = 9.46). ........................................ 60

3-6 (a) Original frame 1. (b) Original frame 4. (c) Motion compensated frame 4 from binary block matching (BBM). (d) Motion compensated frame 4 from conventional block matching (BM). (e) Error image from BBM. (f) Error image from BM. .......................... 61

3-7 (a) Diagram of normalized MAE for the "susie" head and shoulders image. (b) Diagram of normalized MAE for the "flower garden" sequence. 62

3-8 2-D array and PE structure used in the block matching procedure (from Pirsch [12].) .......................................................... 64

3-9 Comparison of the pipelined binary and conventional distortion computation modules (adapted from [12]). .......................... 67

3-10 Comparison of the binary and conventional processing elements. ........................................ 69

3-11 Layout of a pipelined Sobel edge detector of dimension 1496 $\lambda \times 1380$ $\lambda$. The left section is the horizontal kernel and the right section is the vertical kernel. ....................... 71

3-12 IRSIM verification for the conventional PE. ........................................ 72

3-13 IRSIM verification for the binary PE. ........................................ 73

4-1 Illustration of the performance of the algorithm for low resolution images. (a) Original 100 $\times$ 100 "Girl2" image. (b) Reconstructed "Girl2" (0.32 bpp). (c) Original 100 $\times$ 100 "author" image. (d) Reconstructed "author" (0.3 bpp). .............................. 77

4-2 Block diagram for video coding system. ........................................ 79

4-3 Original "flower garden" video sequence. ........................................ 81

4-4 "Flower garden" video sequence compressed to 35.25 Kbits/sec. 81

4-5 Original "author" image sequence. ........................................ 83
4-6 Illustration of compressed "author" sequence at 19Kbits/sec. Original 
image resolution used was 105×120. Compression ratio is 80.  

4-7 Illustration of compressed "author" sequence at 11Kbits/sec. Original 
image resolution used was 95×105. Compression ratio is 101.  

A-1 Copyright agreement for use of the low power library. 

A-2 TSPCL latch dnf401.mag schematic. 

A-3 Multiplexer muxf301.mag schematic. 

A-4 Schematic for 1 bit adder faf001.mag and 1 bit subtracter fsf001.mag.
List of Tables

2.1 Breakdown in the number of bits for the color 256×256 “Lenna” image at 0.35 bpp. Total: 22,627 bits. .............................................. 44
2.2 Breakdown in the number of bits for the color 256×256 “Girl2” image at 0.14 bpp. Total: 9,095 bits. .............................................. 44
2.3 Breakdown in the number of bits for the color 256×256 “Cablecar” image at 0.53 bpp. Total: 34,826 bits. .............................................. 44
2.4 Breakdown in the number of bits for the color 256×256 “Sailboat” image at 0.47 bpp. Total: 30,486 bits. .............................................. 45

3.1 Comparison of normalized mean absolute error (MAE) for prediction using thinned edges and thick edges (resolution of images is 176×112). .......................................................... 56
3.2 A comparison of hardware requirements for binary block matching (BBM) and conventional block matching (BM). ...................................... 63
3.3 Propagation delays for the 8-bit and 16-bit adders at power supply voltages of 1.0 V, 1.5 V and 3.0 V. .............................................................. 66

4.1 Breakdown in the number of bits for the 100×100 “Girl2” image. Total: 3,203 bits ................................................................. 76
4.2 Breakdown in the number of bits for the 100×100 “author” image. Total: 3,073 bits. ................................................................. 76
Chapter 1

Introduction

The significant advances in semiconductor technology over the past decade have greatly improved information transmission and storage capabilities. With the increased complexity achievable in electronic systems, a whole new range of visual information processing applications has become possible. In these applications, various compression techniques are employed to make efficient use of bandwidth and memory resources.

In order to facilitate hardware interoperability, standards for visual information compression such as the Joint Photographic Experts Groups (JPEG) still frame coding standard, the Motion Picture Experts Group (MPEG) video coding standard, and the H.261 video telephony standard have been established. Although these standards play an important role today, they are not suitable for a large range of applications where bandwidth requirements are stringent. In analog phone line video telephony, for instance, relatively high compression ratios (on the order of 100:1 and higher) are required, as the maximum bit rate supported by the Public Switched Telephone Network (PSTN) is at most 40 kilobits per second for both audio and video [1]. Another application that requires high image compression is mobile video telephony, where Radio Frequency (RF) bandwidth has become an extremely important resource. In these situations, a number of tradeoffs with respect to frame rate and image resolution are required. An attempt must also be made to preserve subjectively important features within the image.
This thesis describes the performance of an edge based, low bit rate video compression scheme designed with complexity and power constraints in mind. The algorithm can naturally be divided into the following sections:

- Static Coding
- Interframe Coding

The static portion of a video coding algorithm seeks to reduce the spatial redundancy in an image frame without using information from previous or subsequent frames (static frames are labeled “intraframes” in figure 1-1 (see [2])). The static compression algorithm relies on edge based interpolative techniques that have been found in practice to give superior subjective quality to mean squared error criterion methods. The reduction of temporal or interframe redundancy in the video sequence is achieved by encoding information about the “predicted” frames in figure 1-1 using motion compensated prediction. The interframe coding algorithm uses a block based motion estimation method with a binary edge matching criterion.

![Diagram of intraframe and predicted frames](image)

Figure 1-1: Illustration of the frames involved in the video coding process.

The goal of this thesis is to summarize the performance of the video compression algorithm and to examine its advantages and limitations. Chapter II describes the
static compression algorithm and its performance. Chapter III describes the algorithm for interframe coding and its performance. Because motion estimation is the most computationally expensive portion of video image compression, this chapter discusses possible reductions in power and silicon area derived from using an edge based distortion criterion over conventional grayscale methods. Chapter IV discusses bit rates attainable by the algorithm and presents low bit rate results for face images. The final chapter discusses the results and suggestions for further work.
Chapter 2

Static Image Coding

The objective of static image coding is to reduce the amount of data required to represent a single image frame. The coding is usually done to meet bandwidth and memory constraints subject to an image quality requirement. Various different techniques have been proposed for static image compression. The main groups are

- predictive
- transform
- interpolative/extrapolative
- statistical

In predictive coding, an effort is made to predict the value of a pel (or group of pels) at the decoder using temporally or spatially close pels that have already been received. The prediction is also usually done at the encoder, which then computes a prediction error, quantizes the error and transmits it to the receiver. This method is also known as Differential Pulse Code Modulation (DPCM). In transform coding, the pel data is processed and converted to a different domain (ex. frequency), where the image is characterized by transform coefficients. Transforms with good energy compaction properties are desirable in coding, since they are capable of condensing a large amount of information into a small number of coefficients. Compression is achieved by discarding coefficients with small values. In interpolative and extrapolative coding,
only some of the pels in the image are transmitted and the remainder of the pels are obtained either through interpolation of the transmitted pels or extrapolation (prediction using the previous pels). Statistical coding is lossless and includes Huffman coding, arithmetic coding, and other entropy based methods. These classifications are by no means extensive, as some other coding methods such as Vector Quantization and run length coding do not belong to any of the above four categories\(^1\). In many image compression systems, a combination of different methods is used depending on performance and complexity requirements. See [1] for a thorough discussion of basic image compression techniques.

One of the difficulties associated with the image coding problem is that it is not a simple matter to come up with a meaningful image quality criterion that incorporates human visual psychophysics. For this reason, some compression schemes use mathematically tractable image quality criteria such as minimizing the Mean Squared Error (MSE) of a reconstructed image. A measure of image quality that is often used and which will be used in this thesis as a quantitative measure of image quality is the Peak Signal-to-Noise Ratio (PSNR) which is defined in dB as follows [3]:

\[
\varepsilon_{\text{psnr}} = \frac{\sum_{i=1}^{M} \sum_{j=1}^{N} (I(i,j) - \hat{I}(i,j))^2}{(\max[I(i,j)])^2}
\]

(2.1)

\[
\text{PSNR} = -10 \log_{10} \varepsilon_{\text{psnr}}
\]

(2.2)

In equation 2.1, \(I(i,j)\) is the original \(M \times N\) digitized image and \(\hat{I}(i,j)\) is the reconstructed image. Although reducing MSE or increasing SNR or PSNR is often done for simplicity, methods based on these criteria tend to break down for high compression ratios. In transform coding, for instance, it is common to try to minimize the energy in the reconstruction error \(\varepsilon_R\) (an MSE measure), which is defined as

\[
\varepsilon_R = \sum_{i=1}^{M} \sum_{j=1}^{N} (I(i,j) - I_R(i,j))^2
\]

(2.3)

\(^1\)Coding algorithms are also subdivided into adaptive or non adaptive depending on whether system parameters change with the local statistics of the image.
R in equation 2.3 is the number of coefficients retained and \( I_R(i, j) \) is the reconstruction obtained using those R coefficients\(^2\). The most commonly used linear transform in image coding applications is the Discrete Cosine Transform (DCT), because of its good energy compaction properties and relative ease of implementation. The problem with the DCT is that for low bit compression (below 0.5 bits per pel), coarse quantization of the lower frequency coefficients is necessary. This tends to produce visually unpleasant block edge effects in low detail areas of the picture.

Psychovisual studies have shown that the human eye is very sensitive to edges or sharp luminance transitions [4, 5]. In fact, the presence of edges also tends to mask noise in their vicinity [1]. For these reasons, the algorithm investigated in this research combines edge or contour based techniques with interpolative and entropy coding to encode still images at low bit rates. The overall algorithm is described in more detail in the following section.

### 2.1 The Algorithm

In many edge based compression schemes, the image \( I(i, j) \) is decomposed into a contour component \( C(i, j) \) and a texture component \( T(i, j) \) which are coded separately and transmitted to the receiver [1].

\[
I(i, j) = C(i, j) + T(i, j)
\]  
(2.4)

The coding of edge information \( C(i, j) \) can be done very economically using two level (binary) edge maps and chain codes [6]. Although there are several methods for encoding the texture information, the performance of many of these methods heavily depends on good segmentation, which is not always achieved in practice. For this reason, a combination of interpolation using pels near the edges and a simple residual coding scheme was used in an effort to efficiently and economically encode \( T(i, j) \).

A general illustration of the static compression algorithm is shown in

\(^2\)With this criterion, the Karhunen-Loeve transform achieves the best energy compaction of any linear transform.
Figures 2-1 and 2-2. Figure 2-1 summarizes the encoding procedure, which consists of the following steps:

- Edge Detection and Contour Coding
- Intensity and Color Coding for Selected Points
- Residual Coding
- Entropy Coding

The decoding procedure, which is illustrated in figure 2-2, consists of the following steps:

- Entropy Decoding
- Edge Map Reconstruction
- Placement of Selected Points at Appropriate Locations
- Interpolation (Linear)
- Addition of Residual Information

The proposed static image compression algorithm is based on an approach described by Carlsson [6] and modified in Desai, Mizuki, et al. [7]. Some of the modifications included simplification of the intensity coding and the residual coding process. Emphasis was placed on encoder and decoder simplicity and on achieving high compression ratios.

2.2 The Encoder

This section discusses the details of the edge based encoding procedure and some alternatives to improve image quality. A block diagram of the encoder is presented after all the individual portions of the static algorithm are described.
Figure 2-1: General illustration of the encoding procedure.

Figure 2-2: General illustration of the decoding procedure.
2.2.1 Edge Detection

Edge detection is an extremely important part of the static algorithm. If edges are missed or parts of edges are broken, a perceptually unpleasant blurring of texture from objects in the foreground and the features in the background will occur (for an example, see figure 2-5). This subsection introduces the basics of the edge detection method and discusses some of the modifications that can be made to enhance edge quality.

Although it is possible to use second order edge detectors such as the Laplacian of Gaussian (see Marr and Hildreth [8]), a first order edge detection approach was chosen for hardware complexity reasons and the fact that the computed gradients can readily be used in the edge based motion estimation algorithm. A simple first order edge detection method relies on the generation of an image gradient magnitude map $G_{\text{mag}}(i,j)$ from the luminance component of a color image, which is represented here by $L(i,j)$. Locations of large gradient magnitude indicate large luminance transitions in the image and help to determine the location of edges. $G_{\text{mag}}(i,j)$ can be obtained from gradients formed in two orthogonal directions (which in this case will be the horizontal and vertical directions). The equation for the gradient magnitude map can be given by [3]:

$$G_{\text{mag}}(i,j) = \sqrt{G_x(i,j)^2 + G_y(i,j)^2}$$  \hspace{1cm} (2.5)

where $G_x(i,j)$ represents a horizontal gradient map of $L(i,j)$ and $G_y(i,j)$ represents a vertical gradient map of $L(i,j)$. For implementation purposes, the above formula can be approximated by

$$G_{\text{mag}}(i,j) = |G_x(i,j)| + |G_y(i,j)|$$  \hspace{1cm} (2.6)

The vertical and horizontal gradients can be formed through the following expressions:
\[ G_x(i, j) = L(i, j) \ast h_x(i, j) \]  
\[ G_y(i, j) = L(i, j) \ast h_y(i, j) \]  

where \( h_x(i, j) \) and \( h_y(i, j) \) represent kernels for the computation of the horizontal (determines vertical edges) and vertical (determines horizontal edges) gradients. In vector notation, these kernels will be represented by \( h_x \) and \( h_y \) respectively. Many first order kernels such as the Prewitt, pixel difference and Sobel kernels are commonly used for edge detection. Due to simplicity of implementation and reasonable performance, modified Sobel kernels were chosen for the algorithm. For a quantitative discussion on the performance of edge detectors, see [3]. The \( 3 \times 3 \) Sobel kernels \( s_x \) and \( s_y \) are given by

\[
s_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \quad s_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}
\]

One of the problems observed in Desai, Mizuki et al. [7] is that the \( 3 \times 3 \) Sobel kernels are too small to capture slowly varying edges. In order to capture more slowly varying intensity patterns, it is suggested that a simple alteration be made to the Sobel kernels. The modified Sobel kernels used in the algorithm are:

\[
h_x = \begin{pmatrix} -1 & -1 & 0 & 1 & 1 \\ -2 & -2 & 0 & 2 & 2 \\ -1 & -1 & 0 & 1 & 1 \end{pmatrix} \quad h_y = \begin{pmatrix} -1 & -2 & -1 \\ -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \\ 1 & 2 & 1 \end{pmatrix}
\]

Once the gradient magnitude map \( G_{\text{mag}}(i, j) \) is obtained through the described procedure, a thresholding operation can be carried out in order to determine locations in the image that are likely to contain edges. The magnitude of the threshold can be determined either empirically or through an analysis of the statistics of the image as suggested in [3]. The threshold level affects the sensitivity of the edge detector and
therefore the number of edges in the contour map \( C(i, j) \).

In addition to thresholding, it is useful to thin the edge map to facilitate coding and to decrease the number of bits required to code the contour map. An example of thinning is shown in figure 2-3. A combined thinning and thresholding procedure described by Lim [9] and modified in Desai, Mizuki et al. [7] makes use of the gradient magnitude map \( G_{\text{mag}}(i, j) \) to determine whether an edge point belongs on the thinned edge map or should be eliminated. Figure 2-4 illustrates the points on \( G_{\text{mag}}(i, j) \) that are used to determine whether the point at location \( C \) should be accepted as an edge point. A pel location is accepted as an edge point if the gradient at that location (Point \( C \) in figure 2-4) exceeds the threshold value and the gradients in and around that point satisfy ONE of the following conditions:

- \( C \) is a local maximum: \( (C > U) \ AND \ (C \geq D) \ AND \ (C > L) \ AND \ (C \geq R) \)

- \( C \) is a maximum on the horizontal direction: \( (C > L) \ AND \ (C \geq R) \ AND \ (|G_x| > |G_y|) \)

- \( C \) is a maximum on the vertical direction: \( (C > U) \ AND \ (C \geq D) \ AND \ (|G_y| > |G_x|) \)

A problem with this algorithm is that edges that are at approximately 45° angles may be lost due to the fact that the magnitudes of \( G_x \) and \( G_y \) are very close to each other at those angles. The edge map obtained from using the thinning procedure on an artificially generated 256×256 cartoon-like image is illustrated in figure 2-5(a).
Figure 2-4: Location of pels in the gradient magnitude map \((G_{\text{mag}}(i,j))\) used in the thinning procedure.

The image reconstruction (using the static algorithm) that corresponds to this edge map is shown in figure 2-5(c). The omission of diagonal edges results in a visually unpleasant image reconstruction with significant blurring near the broken edges.

A simple solution to the problem caused by the proximity of \(G_x\) and \(G_y\) at angles close to 45° is to relax the directional gradient comparisons of the last two conditions in the thinning algorithm. For instance, the inequalities that need to be satisfied can be modified to \((|G_x| > |G_y| - \alpha)\) and \((|G_y| > |G_x| - \alpha)\) where \(\alpha\) is a small constant value. Figure 2-5(b) demonstrates the improved edge map and figure 2-5(d) displays the much improved reconstruction of the cartoon-like image. Although it is possible to introduce tests in the diagonal directions, it is much simpler to relax the gradient conditions in a practical implementation of the algorithm.

### 2.2.2 Possible Improvements to the Edge Detection Procedure

The edge detection procedure outlined in Section 2.2.1 proves to be quite satisfactory for images with salient edge features such as cartoon images, but its performance tends to degrade as the image size decreases and when edges features are more subtle. Two
Figure 2-5: Illustration of the effects of poor edge detection on image reconstruction. (a) Edge map using thinning and thresholding procedure described in Desai, Mizuki et al. [7]. PSNR = -28.3 dB. (b) Edge map using slightly relaxed thinning and thresholding procedure. PSNR = -26.7 dB. (c) Reconstructed image using edge map in (a). (d) Reconstructed image using edge map in (b).
approaches that can increase the quality of the edge detection have been examined for the compression algorithm. The first approach relies on histogram equalization to enhance the image contrast in dark regions of the image (as proposed in [3]), while the second method makes use of a simple adaptive thresholding procedure.

As illustrated in figure 2-6, the histogram equalization method forces the distribution of intensity levels to be uniform. Figure 2-6(c) is a plot of the histogram for the original image in figure 2-6(a) while figure 2-6(d) is the equalized histogram that corresponds to the enhanced image in figure 2-6(b). The results of edge detection with and without histogram equalization preprocessing are shown in figure 2-7. Figure 2-7(a) is generated from the original image in figure 2-6(a) and figure 2-7(b) is generated from the enhanced image in figure 2-6(b). The same threshold for edge detection was used in both cases.

As it can be observed, the histogram equalization method enhances edge features, but certain characteristics of the image are altered. For instance, the edge detection after histogram preprocessing in figure 2-7(b) fails to detect the light post near the truck appropriately. In addition, the visibility of the clouds is artificially altered. A possible solution to the undesirable modification of properties in the original image may be adaptive histogram equalization, where different sections of the image are histogram equalized separately. The biggest problem with this approach is that it is computationally very expensive [3]. Even the non-adaptive equalization procedure can be quite expensive in terms of hardware complexity and silicon area [10]. For this reason, the histogram equalization methods are better suited for special applications where hardware resources are not a primary concern.

The second approach to enhancing edge characteristics is to use an adaptive edge detection technique with thresholds that change as a function of the local gradients in the image. As suggested by Netravali [1], a measure of spatial activity can be the computed by obtaining the weighted sum of slopes in a $3 \times 3$ region. A simplified measure of such an activity factor $AF(i,j)$ might be given as
Figure 2-6: Illustration of histogram equalization. (a) Original image. (b) Histogram preprocessed image. (c) Intensity histogram of the original image. (d) Histogram of the image in (b).
Figure 2-7: (a) Result of edge detection without histogram equalization. (b) Result of edge detection with histogram equalization.

\[ AF(i, j) = \frac{1}{9} \sum_{k=-1}^{1} \sum_{l=-1}^{1} G_{\text{mag}}(i + k, j + l) \]  

(2.9)

where \( G_{\text{mag}}(i, j) \) is the gradient magnitude map computed in the previous subsection. Once \( AF(i, j) \) is determined, it is possible to increase or decrease the edge detection threshold according to the local activity of the image. In areas where the activity factor is high, a large number of edges will be present and the threshold should be increased to reject a larger number of candidate points for the edge map. In regions of lower activity function, the threshold should be decreased to accept the more subtle transitions as parts of the edge map. A simple threshold selection criterion might vary the threshold at each candidate point by setting it equal to the activity function at that point. The performance of this adaptive edge detection technique is illustrated in figure 2-8.

A comparison of the edge maps obtained using the adaptive edge detection method and the histogram equalization preprocessing method is shown in figure 2-9. Figure 2-9(a) illustrates the result of edge detection after histogram equalization preprocessing and figure 2-9(b) illustrates the edge map obtained using the adaptive method. It can be observed that the adaptive technique captures the light post that was deleted using the histogram equalization procedure.
Figure 2-8: Result of edge detection using adaptive thresholding technique.

Figure 2-9: Comparison of edge detection techniques. (a) Result of edge detection using histogram equalized image. (b) Result of edge detection using adaptive threshold.
Figure 2-10: (a) Possible directions for the nearest neighbors in 8-connected edge maps. (b) Possible directions for the nearest neighbors in 4-connected edge maps.

Implementation of the adaptive edge detection method is very simple relative to histogram equalization procedures. Only additions within a small $3 \times 3$ window and one multiplication by the coefficient $\frac{1}{6}$ are required to determine the local threshold. Another advantage of the adaptive threshold method is that empirical determination of a global threshold is not required.

2.2.3 Contour Coding

Contour coding involves representing the binary edge map in a compact manner for transmission to the receiver. An efficient approach to contour coding is to use chain codes [1, 6]. In chain coding, each edge is represented by the starting point and a series of vectors indicating the direction of the pels that follow. The starting location and the direction vectors completely specify a contour and allow for its perfect reconstruction at the receiver. The possible directions for a chain code depend upon whether the edge map is 4-connected or 8-connected. Figure 2-10 illustrates the possible directions for the nearest neighbor pels for the 8-connected (figure 2-10(a)) and 4-connected (figure 2-10(b)) maps [1].

Because the 4-connected edge map has fewer states to contend with, it is simpler to implement in hardware and its use facilitates the intensity coding step that follows chain coding. The edge detection procedure produces 8-connected contours, so a conversion to 4-connected contours is necessary. An example of a conversion for a simple contour is illustrated in figure 2-11.
Figure 2-11: Conversion of an edge from 8-connected to 4-connected.

![Conversion of an edge from 8-connected to 4-connected.](image)

Figure 2-12: Example of regular chain coding procedure and differential chain coding procedure. Directional numbers follow the convention in figure 2-10(b).

<table>
<thead>
<tr>
<th>BASIC CHAIN CODE DIRECTIONALS</th>
<th>DIFFERENTIAL CHAIN CODE DIRECTIONALS</th>
</tr>
</thead>
<tbody>
<tr>
<td>3,3,3,1,1,1,0,0,2</td>
<td>1,0,0,2,0,0,2,0,2</td>
</tr>
</tbody>
</table>

Once a 4-connected map is available, the chain coding process can begin. The image is scanned from left to right until the starting point of the first contour is obtained. The starting point is coded, and the direction of subsequent pels in the contour is found and transmitted until the contour ends. Often, it is advantageous to use a differential chain coding procedure, where the directions become relative to the last pel. The advantage of the differential approach is that instead of having 4 possible directions (North, South, East, West), only 3 directions will exist (Forward, Right and Left). Figure 2-12 illustrates the codes for an example contour using normal chain coding and differential chain coding. The numbers used for the directionals follow the convention of figure 2-10(b). Each of these directionals generally has a different probability of occurrence, and for this reason, a variable length Huffman code can be assigned.
2.2.4 Contour Intensity Data Extraction and Coding

Once the edge map information is obtained, intensity information for selected points on both sides of the edges is calculated. An illustration of the points of interest around a given contour is shown in the diagram in figure 2-13(a). The picture in figure 2-13(b) displays these points for the "Lenna" image. The determination of the intensity for these pels of interest must be done with care, since they will be used at the decoder to recover the remainder of the pels between the contours using a process of linear interpolation. Although the following discussion only explains how to obtain the luminance or intensity values of these pels, the chrominance of the pels are obtained in an analogous fashion.

![Only the Locations of Pel Intensities to Be Coded Are Shown](image)

Figure 2-13: Demonstration of points of interest in the encoding procedure. (a) Diagram illustrating the locations of interest. (b) Actual picture of points of interest in the image "Lenna".

At first, it would seem that the values of adjacent pels immediately to the right and left of the edge should be used. One problem with this approach is that the location of the edge might be slightly offset due to imperfections of the edge detection and thinning process. Another problem is that the transition zone for an edge could be slowly varying as shown in figure 2-14, which illustrates intensity values of pels that
describe two vertical edge locations. Both of these problems could lead to the use of intensity values that are grossly in error and which are not representative of the remainder of the pels in the area between the contours.

Figure 2-14: Reason why pels immediately adjacent to the contour might not be representative of the points in the texture section between contours.

For this reason, the value of the pel of interest is determined from intensity values of various pels surrounding the contour. Consider the intensity value $X$ of the point of interest to the right of contour 1 in Figure 2-15(a). In order to determine its value, a linear combination of pel values on both the horizontal and vertical directions can be used.

$$X = \frac{1}{2}(X_H + X_V) \tag{2.10}$$

where

$$X_H = \frac{1}{M} \sum_{i=1}^{M} \alpha_i H_i \tag{2.11}$$

$$X_V = \frac{1}{N} \sum_{j=1}^{N} \beta_j V_j \tag{2.12}$$

Figure 2-15(b) illustrates the horizontal points with intensity values $H_i$ ($i = 1, ..., M$) in the original image and figure 2-15(d) displays the vertical points with intensity values $V_j$ ($j = 1, ..., N$). Usually, the intensity values of pels that are close to edges such as $H_1, H_2, H_{M-1}$ and $H_M$ in figure 2-15(b) are not representative of the
region between the contours (also see figure 2-14) and should have small weighting coefficients \( \alpha_i \) (possibly set them to zero). Although values of other adjacent points can be used to determine the value \( X \), only pels along the same row and column were used, as the reconstruction process uses horizontal and vertical interpolations of the coded intensity values.

More complicated interpolation and intensity determination procedures can also be used at the expense of increased encoder and decoder complexity. A linear interpolation method to determine the intensity values was proposed in Desai, Mizuki et al. [7] and is illustrated in figure 2-16. A line is fitted through the "representative" points between contours, which are chosen after eliminating pels too close to the contours. The intensity value of the left endpoint of the line is chosen for the value \( X_H \) and the intensity of the right endpoint of the line is assigned to the value \( Y_H \) to the left of contour 2 (see figure 2-15(c) for the location of these points).

This interpolation algorithm increases hardware complexity, but it improves the reconstruction quality, since it fits the best line through the intensity values and the reconstruction process at the decoder uses linear interpolation. More complicated approaches to encoding and reconstructing the intensity values between contours are feasible, but in general they tend to produce images at higher bit rates (see [6]). In applications such as compression of cartoon images, this interpolation is not necessary, since the intensity values between contours are constant.

After the intensity data for the points of interest are obtained, the redundancy in intensities of adjacent pels of interest can be exploited. Experimentation suggests that it is possible to break up the edges every \( N \) adjacent points and to replace the values of the calculated intensities by the mean value of those intensities (see Desai, Mizuki et al. [7]). Figure 2-17 illustrates the locations of the points involved in the procedure for \( N = 10 \).

\[
R_M = \frac{1}{N} \sum_{i=1}^{N} R_i \quad (2.13)
\]
Figure 2-15: Determination of the intensity values of pels of interest. (a) Location of pels of interest with intensity values X and Y to be determined. (b) Horizontal pel values used in the determination of $X_H$. (c) Illustration of $X_H$ the value obtained from a linear combination of the horizontal points. (d) Vertical pel values used in the determination of $X_V$. (d) Illustration of $X_V$ the value obtained from a linear combination of the vertical points.
Figure 2-16: Alternative method used to obtain the intensity values of pels of interest.

Figure 2-17: Intensity coding procedure.
\[ L_M = \frac{1}{N} \sum_{j=1}^{N} L_j \]  

(2.14)

All the points \( R_i \) \((i = 1, \ldots, N)\) to the right of the contour are replaced with \( R_M \) and all the points \( L_j \) \((j = 1, \ldots, N)\) to the left of the contour are replaced with \( L_M \) (see figure 2-17). Compression is achieved by transmitting one mean intensity value instead of \( N \) individual intensity values. The parameter \( N \) is adjustable and can be used to control the bit rate of the coder.

### 2.2.5 Residual Coding

Imperfections in the edge detection process can lead to some serious distortion problems. Lines appear in the interpolated regions when improper breaks in the contour occur. These lines are caused by "leakage", as incorrect intensity values from one region in the picture (ex. background) are used in the interpolation process for another region in the picture (ex. an object in the foreground). Examples of this are shown in Desai, Mizuki et al. [7]. A simple approach (to deal with this problem) that has been found to produce subjectively acceptable results is to partition the original image frame into blocks, to calculate the mean value of pels in those blocks and to quantize and transmit only the mean values for those blocks that fully fit between contours. That is, if any edges are present within a block region, a null symbol is sent indicating that the block should not be used.

At the receiver, the mean values of the valid blocks transmitted by the encoder are pasted onto the areas where no contours are present (see figure 2-18). A low pass filtering operation is then performed in regions without edges to restore smoothness. A loss of texture occurs, but at low bit rates, this loss is visually more acceptable than a degradation in the edges of the image. Using small blocks results in higher quality reconstruction of the image at the expense of a higher bit rate (see Section 2.3.1).
Figure 2-18: Use of block mean information at the decoder.

2.2.6 Entropy Coding

Entropy coding can be used to further reduce the redundancy in the data to be sent to the transmitter. The data includes the following:

- Contour Information (starting locations and directional vectors use separate codebooks).

- Intensity (and color) values of pels of interest.

- Residual coding values (mean information).

Each type of data generally uses a separate codebook, since the data are of a very different nature. For instance, the directional vector data used in chain coding contains long patterns with 3 values (forward, left and right), whereas intensity data can have the full dynamic range of 0 to $2^N - 1$ (assuming N bit pulse code modulated data for the original image). The block diagram in figure 2-19 illustrates how a Huffman coding procedure can be used in the static algorithm. See Cover [11] for a thorough explanation of Huffman codes and other forms of entropy coding.
2.2.7 Encoder Block Diagram

The block diagram of the complete static encoder is shown in figure 2-20. The input is a color static frame in the YIQ format, and the output is a Huffman coded bitstream of edge, intensity and mean information. The dashed lines in the figure indicate that contour coding information needs to be used in the intensity coding, intensity extraction, and mean coding procedures.

2.3 The Decoder

The static decoding procedure consists of decoding the Huffman coded serial bit stream, recovering the edge map, performing linear interpolations, adding mean values in regions without edges, and low pass filtering those regions without edges. Recovering the chain coded edge map is straightforward and can be done with a state machine. The intensity values of pels between contours is obtained from the average of the values resulting from a horizontal and vertical interpolation procedure using the points of interest transmitted by the encoder. Figure 2-21(b) shows horizontal interpolation using the values X and Y of the points of interest, while figure 2-21(d) shows the vertical interpolation using the values X and Z. The points $H_i \ (i = 1, ..., M)$
Figure 2-20: Block diagram of color static encoder.
and $V_j (j = 1, \ldots, N)$ are obtained from a simple linear interpolation procedure after making $H_1 = X$, $H_M = Y$, $V_1 = X$ and $V_N = Z$. The resulting horizontal and vertical values obtained at each pel location are then averaged. Once this is done, the residual blocks are added in areas that are large enough to fit full blocks (see figure 2-18), these areas are smoothed, and the decoding procedure is complete.

Figure 2-21: Interpolation procedure. (a) Points of interest with intensity values $X$ and $Y$ used in the horizontal interpolation procedure. (b) Recovery of horizontal component of the intensity values between contours using the values $X$ and $Y$. (c) Points of interest with intensity values $X$ and $Z$ used in the vertical interpolation procedure. (d) Recovery of vertical component of the intensity values between contours using the values $X$ and $Z.$
2.4 Algorithm Performance and Results

This section summarizes the results obtained using the static compression algorithm. As discussed in the introductory section for this chapter, the PSNR measure of image quality is not the best indicator of image quality. In many instances, the only adequate method of determining image quality is through subjective tests.

Figure 2-22(a) contains a $256 \times 256$ grayscale component of the original "Girl2" image. Figure 2-22(b) shows the result obtained using the edge based static algorithm. The image was encoded at 0.16 bits per pel (bpp) and its PSNR was found to be -21.5 dB. Figure 2-22(c) contains a JPEG compressed image with a PSNR of -18.6 dB, which is superior to the PSNR for the edge based algorithm. Subjectively, however, the result in figure 2-22(c) is not pleasing due to the block effects. The number of bits required for this image is also substantially higher, as the picture is encoded at 0.28 bpp. The result in figure 2-22(d) is an attempt to reduce the number of bits required for the JPEG coding scheme. This picture was encoded at 0.27 bpp and its PSNR was found to be -19.5 dB. The result is clearly unacceptable as the quantization has become too coarse. At these compression levels, the JPEG standard provides inadequate results and the PSNR measure is a very poor indicator of image quality.

Figure 2-23 illustrates subjectively high quality images obtained using the edge based compression algorithm on $256 \times 256$ color images. For all images in figure 2-23, the adaptive edge detection method was used, the contour break length for intensity coding was set to $N=15$, and the mean block sizes were $10 \times 10$. The breakdown of the number of bits for each of the compressed images was calculated. Table 2.1 illustrates the breakdown of the number of bits required for the “Lenna” image (encoded at 0.35 bpp), table 2.2 contains the breakdown for the “Girl2” image (encoded at 0.14 bpp), table 2.3 contains the breakdown for the “Cablecar” image (encoded at 0.53 bpp), and table 2.4 illustrates the breakdown for the “Sailboat” image (encoded at 0.47 bpp).
Figure 2-22: Relative performance of the contour based static image coding algorithm versus JPEG. (a) Original 256×256 “Girl2” image. (b) Reconstructed image (PSNR = -21.5 dB, 0.16 bpp). (c) JPEG result (PSNR = -18.6 dB, 0.28 bpp). (d) JPEG result (PSNR = -19.5, 0.27 bpp).
<table>
<thead>
<tr>
<th>Bit type</th>
<th>Number of Bits</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contour Starting Positions</td>
<td>3511</td>
<td>15.52</td>
</tr>
<tr>
<td>Contour Direction Vectors</td>
<td>10375</td>
<td>45.85</td>
</tr>
<tr>
<td>Mean Intensity and Color Data</td>
<td>1497</td>
<td>6.62</td>
</tr>
<tr>
<td>Intensity and Color Data</td>
<td>7244</td>
<td>32.01</td>
</tr>
</tbody>
</table>

Table 2.1: Breakdown in the number of bits for the color 256×256 “Lenna” image at 0.35 bpp. Total: 22,627 bits.

<table>
<thead>
<tr>
<th>Bit type</th>
<th>Number of Bits</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contour Starting Positions</td>
<td>795</td>
<td>8.74</td>
</tr>
<tr>
<td>Contour Direction Vectors</td>
<td>3583</td>
<td>39.40</td>
</tr>
<tr>
<td>Mean Intensity and Color Data</td>
<td>2296</td>
<td>25.24</td>
</tr>
<tr>
<td>Intensity and Color Data</td>
<td>2421</td>
<td>26.62</td>
</tr>
</tbody>
</table>

Table 2.2: Breakdown in the number of bits for the color 256×256 “Girl2” image at 0.14 bpp. Total: 9,095 bits.

<table>
<thead>
<tr>
<th>Bit type</th>
<th>Number of Bits</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contour Starting Positions</td>
<td>7699</td>
<td>22.11</td>
</tr>
<tr>
<td>Contour Direction Vectors</td>
<td>15653</td>
<td>44.95</td>
</tr>
<tr>
<td>Mean Intensity and Color Data</td>
<td>1119</td>
<td>3.21</td>
</tr>
<tr>
<td>Intensity and Color Data</td>
<td>10355</td>
<td>29.73</td>
</tr>
</tbody>
</table>

Table 2.3: Breakdown in the number of bits for the color 256×256 “Cablecar” image at 0.53 bpp. Total: 34,826 bits.

As it can be seen, the bit rate is heavily dependent on the image content. An approximate expression (see Desai, Mizuki et al. [7]) for the total number of bits required to code an $M \times N$ image as a function of the number of edge points $N_e$, the number of contours $N_c$ and the image resolution is given by

$$TotalBits = 1.3 \times N_e + 41 \times N_c + \frac{M \times N}{25} \quad (2.15)$$

In this expression, it is assumed that $10 \times 10$ mean coding blocks are used and quantization of intensity and mean values is done using 4-bits per value.

The head and shoulders images contained fewer contours than the natural scene images and consequently were encoded with fewer bits by the static compression
<table>
<thead>
<tr>
<th>Bit type</th>
<th>Number of Bits</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contour Starting Positions</td>
<td>6802</td>
<td>22.31</td>
</tr>
<tr>
<td>Contour Direction Vectors</td>
<td>12741</td>
<td>41.79</td>
</tr>
<tr>
<td>Mean Intensity and Color Data</td>
<td>1282</td>
<td>4.21</td>
</tr>
<tr>
<td>Intensity and Color Data</td>
<td>9661</td>
<td>31.69</td>
</tr>
</tbody>
</table>

Table 2.4: Breakdown in the number of bits for the color 256×256 “Sailboat” image at 0.47 bpp. Total: 30,486 bits.

algorithm using adaptive edge detection. Note that the best compression ratio 86:1 (0.14 bpp) was obtained for the relatively simple “Girl2” image (the original color images were stored using 12 bpp). In order to increase the compression ratio, it is necessary to reduce the number of edge points in the image by eliminating candidate edges for lower thresholds, to increase the contour coding length N and to increase size of the mean blocks. Higher compression ratio results obtained for the head and shoulders images are shown in figure 2-24. The contour break length was set to N=20 and the mean block size was 15×15. The best result obtained was 0.095 for the “Girl2” image, which corresponds to a static compression ratio of 126:1.

2.5 Conclusions

This section described the static compression algorithm in detail and presented results for a series of different images. Bit rates below 0.3 bpp were achieved for 256×256 head and shoulders color images at a subjectively reasonable quality. If quality can be further sacrificed, bit rates below 0.1 bpp are achievable. The heavy dependence of the total number of bits required to encode an image on scene content was also discussed. This scene content dependence and the irregularity of the decoder structure relative to block based encoding schemes (ex. JPEG) are the biggest drawbacks of the static coding algorithm.
Figure 2-23: Color $256 \times 256$ images. The left column contains the original pictures, the middle column contains the edge maps used for the static encoding process, and the right column contains the reconstructed color images.
Figure 2-24: (a) Compressed image “Lenna” at 0.24 bpp. (b) Compressed image “Girl2” at 0.095 bpp.
Chapter 3

Interframe Coding

Efficient coding of image sequences relies on motion estimation to reduce interframe redundancy. Extensive work has been done on this field and a more complete discussion can be found in [12]. Motion estimation involves the analysis of successive video frames to obtain the displacement of pels or blocks of pels. Several different algorithms have been proposed, and most of these algorithms make the following assumptions:

- Only translational motion occurs.
- Occlusion of one object by another is neglected.

The most popular method for motion estimation is block matching, due to its conceptual simplicity. Optical flow algorithms, which determine translation from estimates of spatio-temporal derivatives of brightness, have the disadvantage that several computations need to be performed for each pel. See Horn and Schunck [13] for a discussion of the first algorithm to determine optical flow from image brightness derivatives. Davis et al. [14] demonstrate the equivalence of block matching and spatio-temporal algorithms for subpixel displacements. More complicated algorithms involving translational motion for entire objects are hindered by difficulties encountered in segmentation.

This chapter investigates the use of a binary block matching motion estimation scheme in terms of quality of motion estimation and silicon area and power reduction.
relative to a conventional block matching approach. An introduction to the block matching procedure follows.

3.1 Introduction to Block Matching

Block matching involves partitioning the current frame into $M \times N$ blocks as shown in Figure 3-1(a) (in the MPEG standard and in the H.261 standard, $M=N=16$). For every block in the current frame, there is a corresponding search window in the previous frame (Figure 3-1(b)). The search window size is determined by the size of the current block and its maximum allowed displacement in the horizontal and vertical directions. In Figure 3-1(b), both the horizontal and vertical displacements are assumed to be in the range [-p, p]. The search window thus contains $(M+2p) \times (N+2p)$ pels.

![Figure 3-1: Illustration of the block matching algorithm. (a) Current block. (b) Search window and location of the minimum distortion position (from Pirsch [12]).](image)

The block in the current frame is matched to a block in the search window according to some distortion criterion. The most commonly used criterion for conventional block matching is the Mean-Absolute Error (MAE) between blocks. Letting the upper left hand corner of the current block be the origin of the coordinate system (point (0,0)), the distortion for the candidate block at position $(m,n)$ in the search window is given by
\[ D(m,n) = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} |x(i,j) - y(i+m,j+n)| \]  

(3.1)

where \(x(i,j)\) are pels in the current block and \(y(i,j)\) are pels in the search window.

The search window block with the minimum distortion is matched to the block in the current frame. The motion displacement between the current block and the selected minimum distortion block determines the motion vector for the current block.

\[ v = [m,n] D_{\text{min}} \]  

(3.2)

Motion vectors are computed for every block in the image in this fashion and are transmitted to the decoder as overhead. The receiver uses the previous or reference frame (it is assumed the receiver has a copy of the reference frame) and displaces each block according to the motion vectors to produce a prediction of the current frame. This cut and paste operation is usually also performed at the encoder end, and the error between the prediction and the reference frame is quantized and transmitted.

A potential drawback of a full search block matching procedure is its high computational requirement. If the maximum block displacement is allowed to be in the range [-p, +p] in both horizontal and vertical directions, there will be \((2p+1)^2\) candidate blocks in the search window. The number of operations per candidate position is equal to \(3 \times M \times N\), where the factor of 3 arises if it is assumed that subtraction, absolute value, and addition each count as one operation. For an \(H \times V\) image, where the number of horizontal pels is \(H\) and the number of vertical pels is \(V\), the number of blocks in the image is \(\frac{H \times V}{M \times N}\), and the total number of operations per second for a frame rate \(F\) is

\[ 3 \times H \times V \times (2p+1)^2 \times F \]  

(3.3)

Assuming \(p=8\), for \(512 \times 512\) video frames at a rate of 30 frames per second, the computational load is 6.8 Giga operations per second (GOPS). The objective of this chapter is to describe how it is possible to reduce the computational load of a
conventional full search block matching procedure and to reduce the area and power required for a silicon implementation of the block matching algorithm.

3.2 Binary Block Matching

There are three approaches to deal with the computationally intensive nature of full search conventional block matching. The first approach is to make use of specialized VLSI architectures to implement exhaustive search block matching [12, 15]. The second alternative is to reduce the number of pel-level distortion operations as described in [16, 17, 18, 19]. The third approach is to reduce the computational load by proposing a different pel-level distortion criterion [20, 21].

The objective of this research is to follow the third approach by combining contour based motion estimation techniques [22] and the block matching algorithm. Simplification of the criterion is done by going from a grayscale approach as shown in figure 3-2(a) to an edge based binary approach as in figure 3-2(b). Instead of using the MAE criterion, the binary correlation between blocks is used to perform matching [23, 21]. The distortion criterion becomes:

\[ D(m, n) = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} x(i, j) \oplus y(i + m, j + n) \]  

(3.4)

where \(x(i, j)\) and \(y(i, j)\) are binary edge maps and \(\oplus\) is the XOR operation. Minimizing this quantity is equivalent to maximizing the correlation.

Using contour edge maps that have not been thinned when performing the correlation procedure produces superior results [23]. An illustration of the difference in performance using the first order modified edge detector described in Chapter 2 is given in table 3.1 for six different predicted frames. The comparison measure is the normalized mean absolute error (MAE) between the frame predicted using motion compensation and the original frame. The normalized MAE is simply the total MAE between the images divided by the total number of pels in the image (the frame resolution in all these cases was 176×112 and the threshold level used was 100). Usually, it is also advantageous to precede the edge detection with a low pass filtering opera-
Figure 3-2: (a) Conventional block matching procedure. (b) Binary block matching procedure.
tion to reduce the noise level in the images. If the adaptive edge detection algorithm is used in the static encoder section, the activity factor of the image (which is a low pass filtered gradient magnitude map) can be used to generate edge maps for the binary matching procedure after appropriate thresholding.

<table>
<thead>
<tr>
<th>Image</th>
<th>Normalized MAE using thinned edges</th>
<th>Normalized MAE using thick edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>susie 1</td>
<td>6.19</td>
<td>6.01</td>
</tr>
<tr>
<td>susie 2</td>
<td>6.09</td>
<td>5.94</td>
</tr>
<tr>
<td>susie 3</td>
<td>7.50</td>
<td>6.86</td>
</tr>
<tr>
<td>flower garden 1</td>
<td>5.31</td>
<td>4.97</td>
</tr>
<tr>
<td>flower garden 2</td>
<td>7.94</td>
<td>7.19</td>
</tr>
<tr>
<td>flower garden 3</td>
<td>12.34</td>
<td>10.01</td>
</tr>
</tbody>
</table>

Table 3.1: Comparison of normalized mean absolute error (MAE) for prediction using thinned edges and thick edges (resolution of images is 176×112).

The block diagram for a binary block matching system is shown in figure 3-3. Edge detection is performed on both the current image and the reference image. The block matching is performed on the binary edge maps using the maximum block correlation criterion. The prediction using the calculated motion vectors is also performed at the encoder end and the error in the prediction is transform coded (as in MPEG standard), quantized and transmitted.

Although binary correlation procedures are often used in practice for stereo applications, the performance for video coding is generally not robust due to possible ambiguity in the contour maps [21]. For this reason, it is necessary to introduce some corrective measures to reduce the possibility of choosing motion vector values that are grossly in error. The first important step is to introduce a rejection criterion for motion vectors that are possibly unreliable. Dron [20] has proposed one such validation procedure for a stereo matching application. The criteria for a motion vector to be accepted as valid are the following:

- Number of edge pels in the current block must exceed a threshold $T_1$.
- The value of the minimum $D(m,n)$ must NOT exceed a threshold $T_2$. 

56
Figure 3-3: A conceptual block diagram of motion compensation using binary block matching (BBM).
Figure 3-4: Blocks used in the interpolation procedure and the notation for the vectors.

The thresholds $T_1$ and $T_2$ can be determined using methods described in [20] or empirically. These thresholding steps increase the robustness of the matching algorithm. An additional step that can be used to improve the accuracy of motion estimation is to try to determine the motion vectors of blocks that failed the validation procedure from an interpolation of motion vectors of adjacent blocks that passed the validation procedure. The adjacent blocks located around a block which will have its motion vector interpolated (labeled "Block of interest") are illustrated in figure 3-4. The notation for the vector $\mathbf{V}_{R1}$ corresponding to block $R1$ is shown in the figure. $\mathbf{V}_{RX1}$ is the horizontal component of $\mathbf{V}_{R1}$ and $\mathbf{V}_{RY1}$ is the vertical component of $\mathbf{V}_{R1}$. See [22] for an edge based algorithm using an interpolation procedure that does not involve blocks.

The horizontal and vertical components of the interpolated vector can be determined through:

$$M_x = \alpha \times \frac{1}{2} \times (V_{rx1} + V_{lx1}) + \beta \times \frac{1}{2} \times (V_{rx2} + V_{lx2})$$  \hspace{1cm} (3.5)

$$M_y = \alpha \times \frac{1}{2} \times (V_{uy1} + V_{dy1}) + \beta \times \frac{1}{2} \times (V_{uy2} + V_{dy2})$$  \hspace{1cm} (3.6)

where $M_x$ is the horizontal component and $M_y$ is the vertical component of the interpolated motion vector for the block of interest that did not pass validation.
possible selection of coefficients that gives greater weight to the immediately adjacent blocks \((R_1, L_1, U_1\text{ and }D_1)\) is \(\alpha = \frac{2}{3}\) and \(\beta = \frac{1}{3}\). A comparison of the difference between a non-interpolated prediction and an interpolated prediction for one frame in the "flower garden sequence" is shown in figure 3-5. The image to be predicted is shown in figure 3-5(a). Visually speaking, the non-interpolated prediction (figure 3-5(b)), which has a normalized MAE of 10 is far more unpleasant than the interpolated reconstruction (figure 3-5(c)), which has a normalized MAE of 9.46. Note that this interpolation procedure can be performed at the decoder if the locations of blocks that failed the validation procedure are known.

### 3.3 Experimental Results

In order to evaluate the performance of the binary block matching, tests were performed on the "susie" image sequence shown in figure 3-6 (a) and (b). Binary block matching and conventional block matching were used to predict the frame in figure 3-6(b) using the frame in figure 3-6(a). The results of these predictions are shown in figure 3-6 (c) and (d), respectively. To further illustrate the differences between the motion compensated frames, the error images in figure 3-6 (e) and (f) were created by subtracting the motion compensated frames from the original frame. In the binary block matching procedure, validation and interpolation eliminated motion vectors that were grossly in error in regions of uniform luminance (ex. the neck area). Although there are some differences in the results of the two methods, the subjective quality appears to be similar.

In addition to a subjective comparison, a quantitative comparison was also made. Figure 3-7(a) contains a comparison of the normalized MAE for the "susie" image sequence using frame differencing, conventional block matching and binary block matching to generate a predicted image. Frame differencing simply involves predicting the current image using the reference image (no motion estimation) and quantizing and transmitting the error. In this case, the normalized MAE is a measure of how the current and the reference images differ. The vertical axis indicates the frame
Figure 3-5: Images reconstructed from the motion vectors. (a) Original frame to be predicted. (b) Binary Block Matching prediction without interpolation of motion vectors of invalid blocks (normalized MAE = 10.0). (c) Binary Block Matching prediction with interpolation of motion vectors (normalized MAE = 9.46).
number of the prediction. Frame 1 immediately follows the reference frame, frame 2 immediately follows frame 1 and frame 3 immediately follows frame 2. Figure 3-7(b) contains the same information for the "flower garden sequence" (figure 3-5(a)).

As it can be seen from the comparison, the motion estimation obtained with binary block matching is worse (in the MAE sense) than that of the conventional method by approximately 10% in most cases. The performance of both algorithms is significantly better than frame differencing. This is especially true for the "flower garden sequence" where a significant amount of motion is present. These results illustrate the importance of carrying out some form of motion estimation for these sequences.

3.4 Practical Implementation of Block Matching

In practical block matching architectures, a 2-dimensional (2-D) array of processing elements and shift registers are used to reduce the number of data accesses. An architecture that achieves this I/O bandwidth reduction can be derived by realizing
Figure 3-7: (a) Diagram of normalized MAE for the "susie" head and shoulders image. (b) Diagram of normalized MAE for the "flower garden" sequence.
that each search window pel is used several times in the computation of the distortions. An efficient array architecture where each processing element (PE) computes the distortion of one candidate position for the search window is shown in Figure 3-8 [12]. In this case, \((2p + 1)^2\) PE's will be required. The PE structure for the architecture is also shown in Figure 3-8. Each PE and shift register in the array (with the exception of registers on the boundaries of the array) contains 3-way registers that are capable of shifting data up, down, and right. The operation of the array is described in detail in [12].

The architecture described can be used for both conventional and binary block matching. The only difference lies in the distortion computation modules of the PE's. In place of the subtracter and the absolute value circuitry, the binary block matching PE would contain a 1-bit XOR gate. Reduction in computation for BBM arises from the fact that the PE's operate on binary data instead of 8-bit gray scale data. Table 3.2 lists the components of the distortion modules for both the binary and conventional 8-bit grayscale PE's. The objective of the next section is to design low power processing elements for both the binary and conventional cases and to perform a comparative analysis in terms of power and silicon area.

<table>
<thead>
<tr>
<th></th>
<th>Processing Hardware per PE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Typical 8-bit BM</td>
<td>8-bit subtracter absolute value circuit 16-bit accumulator</td>
</tr>
<tr>
<td>Proposed BBM</td>
<td>XOR, 8-bit accumulator</td>
</tr>
</tbody>
</table>

Table 3.2: A comparison of hardware requirements for binary block matching (BBM) and conventional block matching (BM).
Figure 3-8: 2-D array and PE structure used in the block matching procedure (from Pirsch [12].)
3.5 Design Issues

The power of a given circuit in CMOS technology can be estimated as [24]:

\[ P = C_{\text{eff}} \times V_{\text{dd}}^2 \times f \]  
(3.7)

where \( V_{\text{dd}} \) is the power supply voltage, \( C_{\text{eff}} \) is the effective capacitance of the circuit and \( f \) is the frequency of operation. The effective capacitance is given by the following expression:

\[ C_{\text{eff}} = \alpha_{0 \rightarrow 1} \times C_{\text{phys}} \]  
(3.8)

The activity factor \( \alpha_{0 \rightarrow 1} \) accounts for the percentage of energy consuming transitions that occur in a given clock cycle (if \( \alpha_{0 \rightarrow 1} = 1 \) all circuit nodes undergo an energy consuming transition). The physical capacitance \( C_{\text{phys}} \) is the actual capacitance of all the nodes in the circuit that can be switched. In order to reduce the power consumption of a circuit, there are three alternatives:

- Reduce the physical capacitance of a circuit.
- Minimize the number of transitions, which amounts to reducing the activity factor \( \alpha_{0 \rightarrow 1} \).
- Reduce the power supply voltage.

Note that reducing the frequency is not an useful option since the average power will be reduced, but the total energy dissipated to perform the overall task will remain unchanged.

Before proceeding with the design, it is necessary to determine the throughput constraints on the processing elements\(^1\). In order to determine how stringent conditions are when using a 2-D array, it is possible to make a rough estimate. For \( 512 \times 512 \) video images at 30 frames per second, \( 1024 \times 16 \times 16 \) blocks need to be processed in

\(^1\)For some applications that do not require high throughput, it is possible to use linear array architectures rather than 2-D parallel arrays.
roughly 33 ms. Assuming [-7, +8] displacements in both horizontal and vertical directions, there will be 256 different distortions for each of those blocks. Each PE is dedicated to a single distortion position and will contain the final distortion value after 256 cycles. Thus, 256 cycles are necessary for each block. A total of 262,144 cycles are required in 33 ms, which results in a minimum clock frequency of 8MHz (125 ns cycle). For higher image resolutions, the required frequency needs to be scaled accordingly.

Once the throughput constraints are determined, the critical delay path for the processing elements must be small enough to meet those constraints. The critical delay for the conventional PE is determined by the delay through an 8-bit subtracter, an XOR gate (assuming a one's complement number representation) and a 16-bit accumulator. Although this delay can be lower than 100 ns for supply voltages over 3V (from table 3.3, it can be seen that at 3V, the delay would be around 150 ns, which is too slow), it is possible to trade off some area for power by using pipelining. By pipelining the distortion module as in Figure 3-9 [12], the critical delay will be determined by a single 16-bit accumulator. The reduction in the critical delay of the physical circuit enables the supply voltage to be lowered until the design constraints are met. SPICE simulations for a 8-bit accumulator show that a worst case delay of less than 100 ns can be obtained with $V_{dd} = 1.5$ V. This is illustrated in table 3.3 which shows propagation delays for the 16-bit and 8-bit accumulators at $V_{dd} = 3$ V, $V_{dd} = 1.5$ V and $V_{dd} = 1$ V.

<table>
<thead>
<tr>
<th>Adder Width</th>
<th>1.0 V Supply</th>
<th>1.5 V Supply</th>
<th>3.0 V Supply</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 bits</td>
<td>142 ns</td>
<td>73 ns</td>
<td>52 ns</td>
</tr>
<tr>
<td>16 bits</td>
<td>280 ns</td>
<td>141 ns</td>
<td>101 ns</td>
</tr>
</tbody>
</table>

Table 3.3: Propagation delays for the 8-bit and 16-bit adders at power supply voltages of 1.0 V, 1.5 V and 3.0 V.

These numbers determine how much the power supply voltage can be reduced for the conventional and binary processing elements before the circuits no longer meet the throughput constraints. For instance, if the delay requirement for the critical
Figure 3-9: Comparison of the pipelined binary and conventional distortion computation modules (adapted from [12]).
path were 100 ns, the conventional PE would have to be run slightly above 3 V (see table 3.3) whereas the binary processing element can run using a supply voltage of less than 1.5 V. Since the supply voltage dependence is quadratic, the reduction in power achieved would be by more than a factor of 4.

3.6 Circuit Components

From a power and wiring perspective, it was decided that a single phase clock would be used. The design was carried out using standard MAGIC cells from the Lager cell library. Every register in the design was implemented using a True Single Phase Clocked Logic (TSPCL) latch found in the standard cell library. The 3-way register on each PE was implemented using a 3-input multiplexer with two select inputs followed by a TSPCL latch. All the adders and subtracters were of the ripple carry variety using transmission gate static CMOS family circuits.

3.7 Comparison Between Conventional and Binary Block Matching

The layouts for the conventional and the binary PE’s are displayed in Figure 3-10. The actual dimensions given in wavelengths (\(\lambda\)) are as follows:

- Conventional PE: 1434 \(\lambda \times 753 \lambda\)

- Binary PE: 363 \(\lambda \times 668 \lambda\)

This corresponds to a silicon area reduction by a factor of 4.4. Using an IRSIM validation, the average effective capacitances over 10 valid blocks of two images in the “flower garden sequence” (figure 3-5) were found to be:

---

2MAGIC is the VLSI circuit layout tool used. The technology was 1.2 \(\mu\) CMOS.

3See appendix A for copyright details and for the schematics of the circuits used.

4Switch level simulator that can be used to estimate power in circuits.
Conventional

<table>
<thead>
<tr>
<th>Conventional</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1434 λ x 753 λ</td>
<td>363 λ x 668 λ</td>
</tr>
</tbody>
</table>

Figure 3-10: Comparison of the binary and conventional processing elements.

- Conventional $C_{eff} = 147.51$ pF
- Binary $C_{eff} = 30.11$ pF

If it is possible to provide the edges at a high rate of speed (by either raising the supply voltage for the edge detection circuitry or by making it simpler), gains in area reduction can be achieved with the binary processing element in addition to those already made with the physical circuitry (factor of 4.4). This is mainly due to the fact that the critical path of the conventional PE is a 16-bit accumulator whereas the critical path of the binary PE is an 8-bit accumulator. For the area reduction, it is possible to halve the number of required binary PE’s by adding an extra storage register after the accumulator and running the PE at twice the frequency (using time-multiplexing). This roughly halves the required area for the overall parallel array (see [12] for more details on time multiplexing of processing elements). The alternative to saving area is to run the binary processing element at an even lower voltage than is required for the conventional processing element and to save power. For instance, although the effective capacitance was reduced by a factor of 5.2, a binary PE can be run at a lower voltage for an equivalent throughput, making further
power gains possible. See table 3.3 for delay-voltage supply pairs. Since the power is proportional to the square of the supply voltage, significant power reduction (over a factor of 20 depending on the voltages in question) is possible.

A layout of a simple edge detector was done in order to verify that the added overhead would not destroy the area gains for a 2-D array. If a simple convolution kernel such as the Sobel operator is used, it is possible to build a compact pipelined structure that takes the symmetry of the kernel into account [25]. The layout of a Sobel edge detector (including horizontal and vertical kernels) is shown in figure 3-11. Its total area is 1496 $\lambda \times 1380$ $\lambda$ or roughly the size of two conventional processing elements. For the overall array of 256 PE's this changes the ratio of the computational structure from 4.4 to 4.3, an insignificant reduction.

As long as the array is relatively large (ex. 256 $\times$ 256), the edge detection overhead is not too significant and the power and area gains will remain significant. In the edge based video algorithm, however, it is possible to exploit the edge detection that already needs to be done for the static compression.

An IRSIM validation for the conventional PE is shown in Figure 3-12 and a validation for the binary PE is shown in Figure 3-13. The input X (INX) is the current block input pel and the input Y (INY) is the search window pel shifting through the PE. The distortion result is stored in the accumulator (ACC).
Figure 3-11: Layout of a pipelined Sobel edge detector of dimension $1496 \lambda \times 1380 \lambda$. The left section is the horizontal kernel and the right section is the vertical kernel.
Figure 3-12: IRSIM verification for the conventional PE.
Figure 3-13: IRSIM verification for the binary PE.
3.8 Conclusions

A binary block matching algorithm for motion estimation was examined and its performance was compared to that of the conventional matching algorithm. The quality of motion estimation using a normalized MAE criterion is about 10% worse in the binary case. If prediction performance can be sacrificed, the binary approach can be utilized to achieve significant silicon area and power reduction. The binary block matching architecture has processing elements that are 4.4 times smaller than the conventional architecture. The area can be reduced further through the use of time multiplexing techniques. If power reduction is the main concern, the power supply voltage can be lowered further for a binary array than for a conventional array (because of lower critical delay path). Because the power is proportional to the square of the voltage supply, this can result in significant power savings.
Chapter 4

Video Algorithm Bit Rates

This chapter discusses use of the edge based video algorithm to obtain very low bit rates (below 20 Kbits/sec). Image resolution and frame rate tradeoffs will be discussed along with compression issues for both static and motion coding.

4.1 Image Resolution and Frame Rate Constraints

The expression for the number of bits per second for an $H \times V$ image where each pel is represented by B bits and the frame rate is F frames per second is:

$$\text{BitRate(bits/sec)} = H \times V \times B \times F$$  \hspace{1cm} (4.1)

Thus, $256 \times 256$ color images with 12 bits per pel at 30 frames per second would require a bit rate of 23.6 Mbits/sec and the compression ratio needed to achieve a bit rate of 20 Kbits/sec is over 1000. This is unreasonable, and some tradeoffs need to be made with respect to image resolution and frame rate. A reasonable combination of image resolution and frame rate is $100 \times 100$ frames at 10 frames per second. If each color pel requires 12 bits, the total bit rate is 1.2 Mbits/sec. If a compression ratio of over 100 can be obtained, a bit rate of less than 12 Kbits/sec can be achieved and transmission over PSTN lines is possible.
4.2 Issues in Static Coding

As observed in results for various images in Chapter 2, the number of bits required to code a static image is heavily influenced by the number of contour points in the image. The largest amount of compression was obtained for the head and shoulders image in which very little background detail was present. Results for a smaller $100 \times 100$ version of this image ("Girl2") and for another face image ($100 \times 100$ resolution) are illustrated in figure 4-1.

<table>
<thead>
<tr>
<th>Bit type</th>
<th>Number of Bits</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contour Starting Positions</td>
<td>422</td>
<td>13.18</td>
</tr>
<tr>
<td>Contour Direction Vectors</td>
<td>1310</td>
<td>40.90</td>
</tr>
<tr>
<td>Mean Intensity and Color Data</td>
<td>224</td>
<td>6.99</td>
</tr>
<tr>
<td>Intensity and Color Data</td>
<td>1247</td>
<td>38.93</td>
</tr>
</tbody>
</table>

Table 4.1: Breakdown in the number of bits for the $100 \times 100$ "Girl2" image. Total: 3,203 bits.

<table>
<thead>
<tr>
<th>Bit type</th>
<th>Number of Bits</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contour Starting Positions</td>
<td>537</td>
<td>17.47</td>
</tr>
<tr>
<td>Contour Direction Vectors</td>
<td>1320</td>
<td>42.95</td>
</tr>
<tr>
<td>Mean Intensity and Color Data</td>
<td>293</td>
<td>9.53</td>
</tr>
<tr>
<td>Intensity and Color Data</td>
<td>923</td>
<td>30.03</td>
</tr>
</tbody>
</table>

Table 4.2: Breakdown in the number of bits for the $100 \times 100$ "author" image. Total: 3,073 bits.

The 0.32 bpp obtained for the "Girl2" static frame would produce a 32 Kbits/sec bit rate (for $F = 10$ frames/sec) if no motion coding were to be done. Note that 0.32 bpp for an image resolution of $100 \times 100$ corresponds to 3,200 bits, while the 0.095 bpp obtained for the $256 \times 256$ picture in chapter 2 corresponds to 6,225 bits. The second image "author" required 0.3 bpp (bit rate of 30 Kbits/sec). These rates are still too high for transmission over PSTN lines, because audio also needs to be included.
Figure 4-1: Illustration of the performance of the algorithm for low resolution images. 
(a) Original $100 \times 100$ "Girl2" image. (b) Reconstructed "Girl2" (0.32 bpp). (c) Original $100 \times 100$ "author" image. (d) Reconstructed "author" (0.3 bpp).
4.3 Issues in Motion Coding

In order to further reduce the bit rate from that obtained in the static coding procedure, motion compensated prediction needs to be used. In this section, lower bounds for bit rates achievable with a simple video compression scheme will be given. In this system, only Huffman encoded motion vectors are transmitted to the decoder without regard to error information. The overall system is shown in figure 4-2.

Figure 4-2: Block diagram for video coding system.

Figure 4-3 and 4-4 illustrate how motion coding using the scheme in figure 4-2 reduces the bit rate for a natural scene. Figure 4-3 contains the original grayscale (represented using 8 bpp) "flower garden" sequence of $176 \times 112$ frames. If static frames are coded at 0.44 bpp (obtained using adaptive edge detection algorithm), a bit rate of 95 Kbits/sec will be required to transmit the sequence of static frames (for
a frame rate of 10 frames/sec). Figure 4-4 displays results of the video coding scheme with interframe coding of two frames between static frames. The leftmost column in figure 4-4 contains static coded frames, while the middle and right columns contain the prediction of one and two frames ahead, respectively (using the static frame in the same row). The bit rate obtained was 35.3 Kbits/sec and the compression ratio for the encoded video sequence was 49. Interframe coding reduces the bit rate by a factor of 2.7 for this particular sequence. The bit rate is unacceptable for PSTN lines, and this is mainly due to the fact that the static bit rate is high for this natural scene. An effort was made to verify if a compression ratio of 100 was possible for the simple “author” image sequence, which has a static bit rate of approximately 30 Kbits/sec.

The sequence in figure 4-5 consists of the original color “author” frame sequence. The block size for the motion estimation was chosen to be 15×15 in all the image sequences that follow. Figure 4-6 illustrates a compressed sequence of 105×120 images at an average bit rate of 19 Kbits/sec (compression ratio of 80). The pictures on the left hand columns are static frames and the pictures on the center and right hand columns are predicted from the static frame on the same row (the arrangement of frames is the same as that in figure 4-4). Figure 4-7 displays 95×105 compressed video frames at 11 Kbits/s (compression ratio of 101). A compression ratio of over 100 for simple face images is thus possible using the scheme in figure 4-2. The subjective image quality obtained for the 19 Kbits/sec case is superior to that of the 11 Kbits/s case. Note that for improved image quality, error information should be transmitted.

### 4.4 Conclusions

The conclusion that can be drawn from these experiments is that in order to achieve high compression ratios, the number of edges in the static images need to be minimized. A compression ratio of over 100 is possible for a very simple (low resolution) restricted class of images using the described static algorithm and a very simple block based motion estimation procedure. For improved image quality, more complex interframe coding procedures can be used.
Figure 4-3: Original "flower garden" video sequence.

Figure 4-4: "Flower garden" video sequence compressed to 35.25 Kbits/sec.
Figure 4-5: Original "author" image sequence.
Figure 4-6: Illustration of compressed "author" sequence at 19Kbits/sec. Original image resolution used was 105×120. Compression ratio is 80.
Figure 4-7: Illustration of compressed "author" sequence at 11Kbits/sec. Original image resolution used was $95 \times 105$. Compression ratio is 101.
Chapter 5

Conclusions

5.1 Results and Conclusions

This thesis describes details of an edge based video compression algorithm and its performance. The emphasis was on achieving the lowest possible bit rate at a subjectively acceptable image quality. As demonstrated in Chapter 2 for the static algorithm, the number of bits required to encode an image is heavily dependent on the number of edges detected for that image. Because static frames generate the bulk of the bits transmitted, this heavily influences the final bit rate.

The motion estimation method consisted of a binary block matching approach using validation and interpolation procedures to improve subjective image quality. An obvious advantage of using an edge based motion estimation process is that the preprocessing structure for the static algorithm can be used to advantage. The performance of the edge based motion estimation does not differ drastically from that of conventional block matching procedure and hardware savings are considerable as the silicon area can be reduced by a factor of 4.3 times (a factor of 8 is feasible if the edge detection process can be accelerated and power traded off). The power for the array can also be reduced by a factor of 5.2 without any voltage scaling. Further gains are possible with voltage scaling.

Overall compression ratios of over 100 were achieved for low resolution 95x105 color images (compressed sequence bit rate was 11 Kbits/sec). In order to obtain
high compression and bit rates that are reasonable for PSTN lines, the image class has to be restricted and quality has to be sacrificed. If higher bandwidth ISDN lines can be used, some of these constraints can be eased.

5.2 Further work

A few suggestions can be made with regards to further work on the compression algorithm. For the static compression, more efficient edge map coding schemes that do not compromise edge quality can be investigated. Although the chain coding methods are quite efficient, it would be desirable to reduce the contribution of the edge information to the overall static bit rate to well below the approximately 50% of number of bits required for static coding. For the motion estimation architecture, it may be possible to explore low power analog matching and interpolation circuits. From a bit rate standpoint, more complex static and motion algorithms (possibly next generation schemes) can be investigated.
Appendix A

Circuits Used in Design of Processing Elements

This appendix contains the schematics for the main components used in the design of the processing elements in Chapter 2. These components were obtained from the Lager Low Power library stdcell2.3lp. The standard MAGIC cells can be obtained from Berkeley at the following link


under the "libraries" directory.

The copyright agreement for use of the library and its documentation is shown in figure A-1.

In the pipelined PE structure, the True Single Phase Clocked Logic (TSPCL) latch dfnf401.mag was used. The schematic for the latch is shown in figure A-2. The multiplexer structure used in the three-way registers is illustrated in figure A-3. The schematics for the adder and subtracter are illustrated in figure A-4. For the full characterization of these cells and the other stdcell2.3lp cells, see the stdcell2.3lp.ps documentation file that can be obtained along with the library.
Figure A-1: Copyright agreement for use of the low power library.

Figure A-2: TSPCL latch dfnf401.mag schematic.
Figure A-3: Multiplexer muxf301.mag schematic.

Figure A-4: Schematic for 1 bit adder faf001.mag and 1 bit subtracter fsf001.mag.
Bibliography


