MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

A.I. Memo No. 1584 November, 1996

Edge and Mean Based Image Compression

Ujjaval Y. Desai, Marcelo M. Mizuki, Ichiro Masaki, and Berthold K.P. Horn

Abstract

In this paper, we present a static image compression algorithm for very low bit rate applications. The algorithm reduces spatial redundancy present in images by extracting and encoding edge and mean information. Since the human visual system is highly sensitive to edges, an edge-based compression scheme can produce intelligible images at high compression ratios. We present good quality results for facial as well as textured, 256 x 256 color images at 0.1 to 0.3 bpp. The algorithm described in this paper was designed for high performance, keeping hardware implementation issues in mind. In the next phase of the project, which is currently underway, this algorithm will be implemented in hardware, and new edge-based color image sequence compression algorithms will be developed to achieve compression ratios of over 100, i.e., less than 0.12 bpp from 12 bpp. Potential applications include low power, portable video telephones.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for this research was provided in part by the Sharp Corporation. Marcelo Mizuki was supported by an NSF Graduate Fellowship.

1 Introduction

Image data compression plays an important role in the transmission and storage of image information. The goal of image compression is to obtain a representation that minimizes bit rate with respect to some distortion constraint. Typical compression techniques achieve bit rate reduction by exploiting correlation between pixel intensities. Standard compression methods include predictive or waveform coding, in which pixel correlation is reduced directly in the spatial domain, and transform coding, in which the intensity values are transformed into another domain such that most information is contained in a few parameters. Most of these methods use the mean square error (MSE) distortion measure so as to make mathematical analysis tractable.

For low bit rate applications, standard compression methods fail to provide satisfactory performance because of their inadequate distortion criterion (MSE). As the number of bits to be transmitted is reduced, it becomes increasingly crucial to preserve the perceptually significant parts of the image. Since the mean square error measure is not based on properties of the human visual system, algorithms using this measure tend to produce unacceptable results at low bit rates.

One of the most interesting properties of human visual perception is its sensitivity to edges [1, 2]. This suggests that edges can provide an efficient image representation, making edge-based compression techniques very useful, even at high compression ratios. There has been some previous work on edge-based compression, in particular, by Graham, Kocher, and Carlsson. Graham [3] considers an image as being composed of low and high frequency parts, which are encoded separately. The high frequency part corresponds to contours in the image, and the low frequency part corresponds to smooth areas between contours. Kocher et al. [4, 5] use region growing to segment the image into regions. The texture of each of these regions is encoded separately. Carlsson [6] proposes a sketch-based coding scheme for grey level images. The work described in this paper is based on the ideas presented by Carlsson. The representation, however, has been modified, and the encoding process has been simplified to suit hardware implementation, while providing similar performance. In particular, our algorithm uses: (a) 1-D linear interpolation, as opposed to iterative minimum variation interpolation, for efficient implementation; (b) intensity line-fitting for improved performance; (c) efficient coding of contour intensity data; and (d) mean coding, as opposed to Laplacian pyramid coding, for encoding the residual error. Detailed description of the algorithm can be found in the next section.

Another important advantage of our compression approach is that it can be used with edge-based motion estimation to encode moving images. The reader is referred to [7] for preliminary results on edge-based image sequence coding.

2 Compression Algorithm

The edge-based representation of an image consists of its binary edge map and the intensity information on both sides of the contours in the edge map. This representation is sufficient for obtaining an intelligible reconstructed image. The reconstruction is performed by simply interpolating the intensity information between contours of the edge map. As with many compression methods, the quality of reconstruction can be improved by residual coding, which in this case involves encoding the difference between the input image and the interpolated image. The compression algorithm, therefore, consists of edge detection, contour coding, contour intensity data extraction and coding, and residual coding. This section describes the algorithmic details of all these operations.

It is important to point out that our overall approach has two phases. In the first phase, which is presented in this paper, a new algorithm is developed with hardware implementation in mind, and in the second phase, which is currently being studied, the algorithm is modified for efficient hardware implementation. Even if the algorithm is modified in the second phase to suit hardware implementation, the research described in this paper should remain useful.

2.1 Edge Detection

Various edge detection methods can be found in the image processing literature. One of the simplest and most widely-used methods for digital implementation is the Sobel operator. Since in this phase of the project, we are not interested in optimizing hardware implementation, we chose the Sobel operator for edge detection just because of its simplicity and good performance. Any other efficient edge detection method can be used without significantly affecting the performance of the overall algorithm.

In the Sobel edge detection method, the input image is convolved with two filters: h_x and h_y.

$$h_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \qquad h_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

The filter h_x corresponds to horizontal differentiation (and responds to vertical edges), while h_y corresponds to vertical differentiation (and responds to horizontal edges). The results of the two convolutions are combined in the following way to obtain the Sobel edge-intensity map (S_{mag}):

$$S_x = I * h_x, \quad S_y = I * h_y \tag{1}$$

$$S_{mag} = \sqrt{{S_x}^2 + {S_y}^2} \tag{2}$$

where I is the input image.

S_{mag} is then thresholded to create the edge map for the input image. The edges obtained using this algorithm are usually very sharp. In order to obtain slowly-varying edges, we use Level edge detection in conjunction with the Sobel edge detection algorithm. Level edges are detected in exactly the same way as Sobel edges, but with different filters:

$$L_x = I * \hat{h}_x, \quad L_y = I * \hat{h}_y \tag{3}$$

$$L_{mag} = \sqrt{{L_x}^2 + {L_y}^2} \tag{4}$$

where

$$\hat{h}_x = \begin{pmatrix} -1 & 0 & 0 & 0 & 1 \\ -2 & 0 & 0 & 0 & 2 \\ -1 & 0 & 0 & 0 & 1 \end{pmatrix} \qquad \hat{h}_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

The final edge-intensity map is obtained by weighted averaging of the two individual edge-intensity maps. The weight w is chosen empirically.

$$I_{mag} = w \times S_{mag} + (1 - w) \times L_{mag} \tag{5}$$

To facilitate simple implementation of our edge detection algorithm, the expressions for S_{mag} and L_{mag} have to be modified because they contain the square root operation, which is difficult to implement. We have found that acceptable edge maps are obtained if the expressions for S_{mag} and L_{mag} are approximated by using the absolute value operation. This modification simplifies the implementation without sacrificing the edge map quality. The resultant expression for I_{mag} is given below.

$$I_{mag} = I_x + I_y \tag{6}$$

where

$$I_x = w \times |S_x| + (1 - w) \times |L_x| \tag{7}$$

$$I_y = w \times |S_y| + (1 - w) \times |L_y| \tag{8}$$

Notice that the above expression for I_{mag} requires 4 convolutions. The number of convolutions can be reduced by realizing that in most cases, S_x has the same sign as L_x, and S_y has the same sign as L_y. Therefore, I_x and I_y can be written as:

$$I_x = |I * g_x|, \quad I_y = |I * g_y| \tag{9}$$

where

$$g_x = w \times h_x + (1 - w) \times \hat{h}_x \tag{10}$$

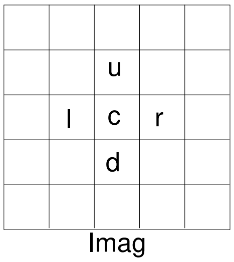
$$g_y = w \times h_y + (1 - w) \times \hat{h}_y \tag{11}$$

2.1.1 Thresholding and Thinning

The edge-intensity map produced by the aforementioned filtering operations has to be tested against a threshold in order to create a binary edge map. Since the number of edge pixels in the edge map is directly dependent on the threshold level, the bit rate can be controlled by varying the threshold.

It turns out that the binary edge map obtained by thresholding the edge-intensity map is thick and hence cannot be encoded efficiently. A thinning operation must be performed to produce edges that are single pixel wide. We use a very simple thinning algorithm in which every pixel in the edge-intensity map that is over the threshold is checked for a local maximum in the horizontal and vertical directions. If the current pixel is a local maximum in the horizontal direction but not the vertical, then it is considered an edge pixel only if $I_x > I_y$. Similarly, if the current pixel is a local maximum in the vertical direction but not the horizontal, then it is considered an edge pixel only if $I_y > I_x$. This processing eliminates false edges [9].

The complete thinning algorithm is illustrated in figure 1 below. Figure 6 shows the binary edge maps for the images in figure 5.



Let c, u, d, l, and r represent the edge-intensities at the current, up, down, left, and right pixels, respectively. The current pixel is an edge pixel if

$$[c > u \text{ AND } c \geq d \text{ AND } c > l \text{ AND } c \geq r] \text{ OR}$$
$$[(c > u \text{ AND } c \geq d \text{ AND } I_y > I_x) \text{ OR } (c > l \text{ AND } c \geq r \text{ AND } I_x > I_y)]$$

Figure 1: Illustration of thinning

2.2 Contour Coding

The edge map produced by edge detection has to be encoded for efficient transmission. The encoding process involves three steps:

- 8-connected map to 4-connected map
- Contour filtering
- Chain coding

2.2.1 8-connected map to 4-connected map

An 8-connected map is a binary graph in which each pixel has 8 neighbors, as shown in figure 2(a). Our analysis has shown that reduction in bit rate can be achieved if the 8-connected map is transformed into a 4-connected map. Also, the 4-connected map makes the implementation of the subsequent modules much simpler. Due to these reasons, our contour coding module converts the input edge map into a 4-connected map and then performs the encoding. An illustration of the conversion process is shown in figure 2(b).

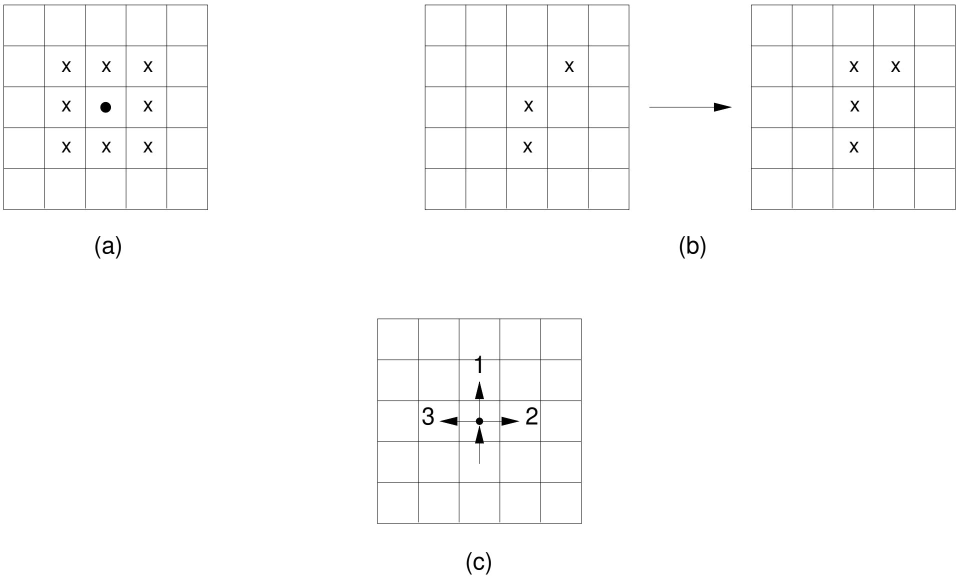


Figure 2: (a) An 8-connected edge map; (b) 8-connected to 4-connected transformation; (c) Possible directionals for a 4-connected map.

2.2.2 Contour filtering

The edge map for an image contains many contours, each representing an object present in the image. Typically, long contours correspond to important features, while short contours are usually produced by noise. Therefore, in order to encode the edge map efficiently, short contours must be eliminated. This simple step is referred to as contour filtering.

2.2.3 Chain coding

Chain coding has been shown to be very efficient for encoding binary contours [8]. Each contour is represented with a starting point and a sequence of directionals that gives the direction of travel as the contour is traced. The starting point is found by simply scanning the edge map, row by row, for the first non-zero pixel. Once the starting point has been found, the following procedure is used:

- Find the next pixel on the contour by searching the neighbors clockwise for a non-zero pixel.
- Output the directional that represents the direction of the next pixel relative to the current direction.
- Delete the current contour pixel, go to the next pixel, and update the current direction.

This process is continued until the whole contour is traced.

Figure 2(c) shows the possible directionals for a 4-connected edge map. Each directional can be represented with $log_2(3)$ bits. If we use statistical coding on the sequence of directionals, an even lower bit rate of about 1.3 bits/contour point can be obtained [6]. There is also an overhead of 9 bits/contour to encode the starting point.

2.3 Contour Intensity Data Extraction and Coding

Edge-based image representation requires encoding of the binary edge map and the contour intensity data of the image. The contour intensity data typically consists of intensity values of pixels to the left and right of each contour pixel. The receiver reconstructs the image intensity by interpolating the contour intensity data. Since the number of contour pixels is much less than the total number of pixels in the image, edge-based representation can be used for low bit rate image compression.

The reconstructed image is obtained by averaging the results of horizontal and vertical interpolation. In horizontal interpolation, the binary edge map is scanned row by row, and the intensity values between two consecutive edge points on a row are obtained by linearly interpolating the corresponding contour intensity data. The same process is repeated for vertical interpolation, but this time the image is scanned column by column. We use linear interpolation, as opposed to other sophisticated interpolation techniques like iterative interpolation with minimum variation criterion [6], in order to simplify the hardware implementation of the algorithm.

As mentioned above, we would like to represent the image intensity between contours with a linear approximation. For good reconstruction, the contour intensity data must be chosen so as to obtain a near-optimal linear approximation. We could use the left and right intensities of contour pixels as contour intensity data, however, in many instances this leads to poor reconstruction, as illustrated in figures 3 (a) and (b).

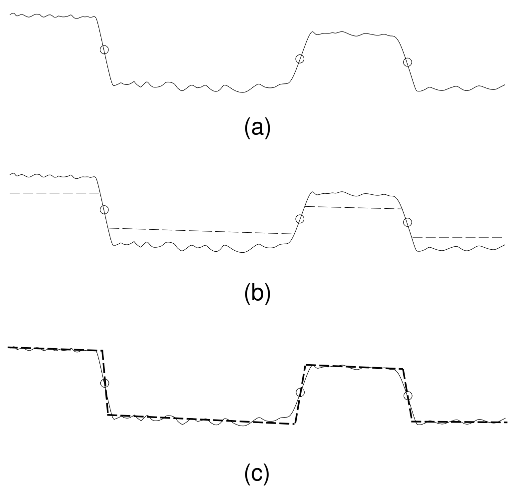


Figure 3: 1-D illustration of linear interpolation. (a) Original signal, 'o' marks the location of an edge; (b) reconstructed signal if the left and right intensities are chosen as contour intensity data; (c) reconstructed signal with line-fitting.

2.3.1 Intensity Line-fitting

A better approach, as shown in figure 3(c), is to find the best linear approximation for the image intensity between contours and then use these parameters as contour intensity data. The approximation is done horizontally as well as vertically. The horizontal approximation is obtained by fitting a line to the intensity values between two consecutive contour pixels on a row, for each row of the image. The end points of these lines are used as horizontal contour intensity data. Similar processing is done on the columns for vertical approximation. Finally, the results of the two approximations are combined, by averaging if needed, to obtain the required contour intensity data. Our simulation results indicate that the quality of reconstruction is improved significantly by the line-fitting approach. Since there are closed-form solutions available for fitting a line through data, this approach is simple to implement.

2.3.2 Contour Intensity Data Coding

Once the contour intensity data is obtained, it has to be compressed for transmission. Efficient compression can be achieved if we exploit the redundancy in the contour intensity data for a contour. Since the intensity between contours is assumed to be smooth, there should be little variation in the contour intensity data on each side of a contour. This assumption suggests that the data on each side of a contour could be represented with just one parameter, namely, their average. In reality, however, this assumption does not hold because the edge extraction process is not perfect. Experimental results indicate that a more reasonable coding strategy is to partition the data for a contour into fixed-length pieces and then encode each piece with its average value.

2.4 Residual Coding

As described in the previous section, the reconstructed image is obtained by interpolating the contour intensity data. Because of imperfections in edge extraction, which lead to open contours, the interpolation of the contour intensity data diffuses the intensity of one object into its neighboring regions, causing reconstruction error. In order to reduce this error, the residual image, which is obtained by subtracting the interpolated image from the original image, must be encoded. Standard methods employed for residual coding include waveform coding techniques like Laplacian pyramid coding [6]. These coding techniques, however, are not feasible for very low bit rate applications.

We perform mean coding on the input image to reduce the reconstruction error and to keep the bit rate down. In mean coding, the input image is partitioned into 10 x 10 blocks, and the mean of each block is encoded and transmitted. The receiver uses the mean information and the interpolated image to obtain an acceptable reconstruction of the input image.

One simple way to use the mean information is to modify the mean of the interpolated image so that it matches the mean of the input image. Unfortunately, this method can not correct the line artifacts present in the interpolated image. Our experiments indicate that these artifacts can be reduced considerably if the intensity of each non-edge block in the reconstructed image is set to the mean of the corresponding block in the input image (see figures 7 and 8). As a result of this processing, all non-edge blocks in the reconstructed image become DC (constant intensity) blocks, and thereby lose their texture. This loss of texture in non-edge blocks is, however, not that crucial because texture information is not necessary in low bit rate applications.

Another significant advantage of mean coding over other residual coding techniques is that in mean coding, the encoder does not have to compute the interpolated image, and therefore, has a simpler structure.

2.5 Color Image Encoding

The algorithm described above can be easily extended for encoding color images. The input color image is assumed to be in the YIQ format. Edge detection and contour coding is performed on the luminance (Y) component alone, while contour intensity data extraction and coding is performed independently on all three components. The luminance component is then mean coded using 10 x 10 blocks, while the chrominance components are mean coded using 20 x 20 blocks. The complete block diagram of the encoder is shown in figure 4.

3 Bit Rate Estimation

Unlike fixed-bit-rate compression schemes like Vector Quantization (VQ), the bit rate of the edge-based compression technique depends on the content of the input image, of which we have no a priori knowledge. In this section, we first present the rate controlling parameters of our algorithm and then use these parameters for bit rate estimation.

Based on the compression algorithm described in the previous section, one can clearly see that the following parameters control the bit rate:

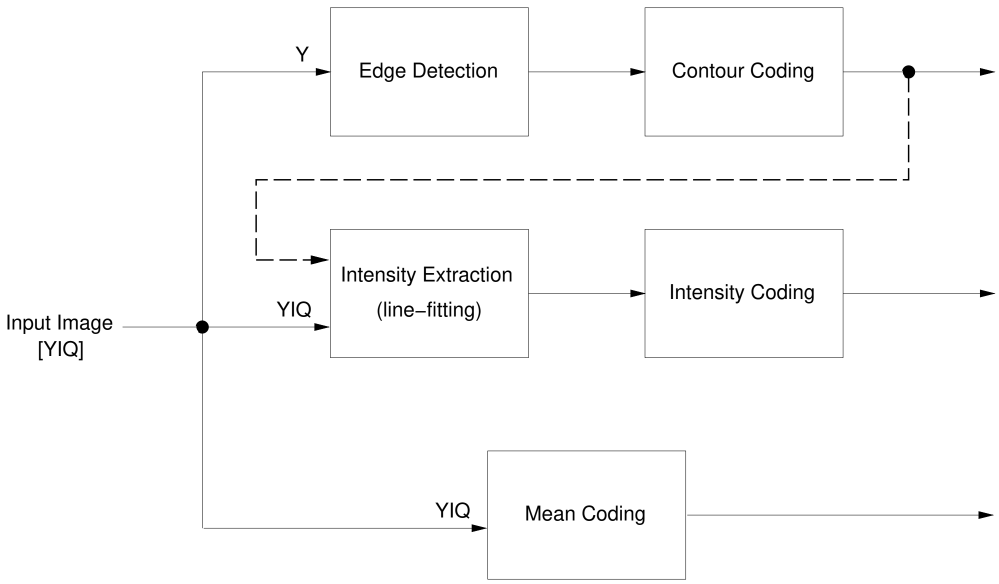


Figure 4: Block Diagram of Encoder

- Edge detection threshold, T;
- Edge weight, w, for combining Sobel and level edges;
- Minimum contour length, L_{min}, for contour coding;
- Maximum contour length, L_{max}, for intensity coding.

The edge weight,w, and minimum contour length, L_{min}, vary from image to image. But for most images, $w = 0.5$ and $L_{min} = 6$ to 10 produce good results. Similarly, L_{max} must be chosen experimentally. Our simulation results indicate that a reasonable value for L_{max} is 15 for the Y component of the input. This implies that the contour intensity data is encoded by partitioning each contour into 15 pixel long pieces and transmitting the average intensity on each side. Since the I and Q components are perceptually less important and have much less information than the Y component, they are encoded with $L_{max} = \infty$.

As opposed to the other parameters, T can not be chosen a priori. The only way to achieve a given fixed bit rate is to start with a predetermined T and then fine-tune it until the desired bit-rate is obtained. The fine-tuning, however, does not have a serious impact on the speed of the overall algorithm because our simulations suggest that the correct threshold level can be reached with very few iterations, each involving thresholding and thinning of the edge-intensity map and then estimating the bit rate. It is important to note that the edge-intensity map has to be computed only once.

Having looked at the rate controlling parameters, we now focus on the problem of estimating bit rate for measuring the performance of the compression algorithm. There are three main modules in the algorithm that contribute to the total bit rate: contour coding, intensity coding, and mean coding.

The performance of contour coding has been studied extensively by many researchers [6, 8]. From their results, we can estimate the number of bits required for contour coding to be $1.3L_c + 9N_c$, where L_c is the total number of contour pixels in the image and N_c is the number of distinct contours. For details on these estimates, the reader is referred to [6].

Intensity coding is performed by breaking each contour in the image into L_{max} pixel long pieces and then average encoding the intensity on each side. Our results indicate that for $L_{max} = 15$, which is the value chosen for the Y component, the total number of contour pieces equals $2N_c$ on the average. This implies that intensity coding for the Y component requires $2N_c \times 2$ parameters. On the other hand, since L_{max} is chosen to be ∞ for the I and Q components, each of them requires $N_c \times 2$ parameters. If each parameter is quantized to 4 bits, the total number of bits needed for intensity coding becomes $32N_c$.

Recall from the discussion on mean coding that the Y component is encoded with a block size of 10 x 10, while the I and Q components are encoded with 20 x 20 blocks. If the image size is $N \times M$, the number of mean values to be transmitted equals $(1.5NM)/100$. In our simulations, statistical coding of quantized mean values resulted in about 2.7 bits per mean. Thus, the number of bits required for mean coding can be estimated to be $(NM)/25$.

Combining all the above estimates, the total bit rate of our compression algorithm can be expressed as:

$$1.3L_c + 41N_c + \frac{NM}{25}.$$

4 Results

The edge-based static compression algorithm was evaluated using the four images shown in figure 5. The images are 256 x 256 and have 24 bits/pixel. Figure 6 shows the binary edge maps obtained by edge detection. The rate controlling parameters and the resulting bit rates for each of the images are given in table 4.

In order to illustrate the significance of mean coding, the interpolated images (without mean coding) are shown in figure 7 and the final reconstructed images are displayed in figure 8. These figures clearly show the effectiveness of mean coding in reducing line artifacts of an interpolated image.

As one can see from these results, our edge and mean based algorithm produces good quality images at very low bit rates. Since sharp edges are very well preserved, the resulting images convey almost all information that is perceptually important. There is, however, some loss of texture for images like Sailboat and Cablecar because the algorithm does not perform any texture coding.

Tests were also performed to determine the lowest possible bit rates for which the algorithm could produce acceptable quality results. These bit rates were found to be 0.16 bpp and 0.08 bpp for the Lenna and Girl2 images, respectively. Figures 9 (a) and (b) show the reconstructed Lenna image at 0.2 bpp and 0.16 bpp. Figures 9 (c) and (d) show the reconstructed Girl2 image at 0.1 bpp and 0.08 bpp.

Color image	Threshold(T)	L_{min}	L_c	N_c	bit rate
Lenna	180	6	5110	220	0.275 bpp
Girl2	75	7	2970	115	0.175 bpp
Cablecar	225	10	5100	215	0.275 bpp
Sailboat	230	9	4550	230	0.275 bpp

Table 1: Rate controlling parameters and bit rate estimates

(a) Lenna (b) Girl2 (c) Cablecar (d) Sailboat

Figure 5: Original color images displayed in monochrome

(a) (b) (c) (d)

Figure 6: Binary edge maps

(a) (b) (c) (d)

Figure 7: Interpolated color images (without mean coding) displayed in monochrome

(a) (b) (c) (d)

Figure 8: Reconstructed color images (with mean coding) displayed in monochrome

(a) (b) (c) (d)

Figure 9: Reconstructed color images (with mean coding) displayed in monochrome (a) Lenna at 0.2 bpp; (b) Lenna at 0.16 bpp; (c) Girl2 at 0.1 bpp; (d) Girl2 at 0.08 bpp

5 Conclusions

Most traditional compression techniques exploit the statistical characteristics of images to reduce bit rate. These techniques fail to provide acceptable quality at very low bit rates because they do not take into account the properties of human visual perception. We propose a compression algorithm that represents an image in terms of its binary edge map, mean information, and the intensity information on both sides of the edges. An edge-based representation is chosen because the human visual system is highly sensitive to edges.

The proposed algorithm is capable of delivering acceptable quality images at 0.1 to 0.3 bpp for 256 x 256 color images. Our simulation results suggests that the algorithm works well for images with limited amount of texture, such as facial images. This, however, is not a drawback in the case of many low bit rate applications, for example videophones, where the input images can be assumed to be facial.

The research presented here is part of a project whose goal is to develop a new edge-based image sequence compression technique for use in low power, portable videophones. Edge-based motion estimation and detailed hardware implementation of the static algorithm are currently being studied.

References

[1] T. Cornsweet, *Visual perception,* Academic Press, New York, 1970.

[2] F. Ratliff, *Contour and contrast*, Sci. Am., Vol. 226, No. 6, June 1972, pp. 99-101.

[3] D. Graham, *Image transmission by two-dimensional contour coding,* Proc. IEEE, Vol. 55, No. 3, March 1967, pp. 336-346.

[4] M. Kocher and M. Kunt, *Image data compression by contour texture modelling,* SPIE Int. Conf. on the Applications of Digital Image Processing, Geneva, Switzerland, April 1983, pp. 131-139.

[5] M. Kunt, A. Ikonomopoulos, and M. Kocher, *Second generation image coding techniques,* Proc. IEEE, Vol. 73, No. 4, April 1985, pp. 549-574.

[6] S. Carlsson, *Sketch based coding of grey level images,* Signal Processing, Vol. 15, 1988, pp. 57-83.

[7] M. Mizuki, U. Desai, I. Masaki, and A. Chandrakasan,*A binary block matching architecture with reduced power consumption and silicon area requirement,* Submitted to ICASSP 1996.

[8] M. Eden and M. Kocher, *On the performance of a contour coding algorithm in the context of image coding: Part 1. Contour segment coding,* Signal Processing, Vol. 8, No. 4, 1985, pp. 381-386.

[9] J. Lim, *Two-dimensional signal and image processing*, Prentice Hall, 1990.