LOSSY COMPRESSION OF IMAGES USING PIECEWISE-LINEAR APPROXIMATION

Vasudev Bhaskaran, Balas K. Natarajan, and Konstantinos Konstantinides

Hewlett-Packard Laboratories
Multimedia Systems Department
Palo Alto, CA

Abstract

We explore the lossy compression of images using an optimal algorithm for the approximation of waveforms by piecewise linear functions. We also present the hardware implementation of a modified version of the optimal algorithm. The modified algorithm is simpler and better suited for real-time applications. Furthermore, it retains all properties of the optimal algorithm, except that the compression ratios maybe at most twice the optimal number. Rate-distortion plots show that the proposed algorithms perform favorably in comparison with the JPEG standard.

1. Introduction

Sources such as speech and images are characterized by large amounts of data with properties varying widely over space and time. For such sources, there is often a need to compress the data to minimize transmission time or storage requirements. In many applications, a lossy compression scheme is acceptable, e.g. voice-mail, image browser. Traditional lossy compression methods rely on spatial or temporal redundancies in the data. For example, in Differential Pulse Code Modulation (DPCM), inter-sample correlations are exploited to predict future samples and the prediction errors are then stored with reduced resolution so as to yield some compression. On the other hand, block based compression schemes such as vector-quantization (VQ) first examine several samples within a block to derive a representative pattern for the block, and then assign a symbol to represent this block.

An alternate viewpoint is to consider the data as samples of a one-dimensional waveform. The waveform can then be compressed by approximating it by a piecewise linear function within a prescribed error tolerance, with the break points of the piecewise linear function composing the compressed representation. Image compression schemes based on this approach are reported in Rosenberg [1] and in Walach [2], although to be precise the latter paper takes a fractal-geometric approach. These methods have yielded good quality images at compression ratios of approximately 8:1. However, these methods are not provably good in that they do not offer an *a priori* estimate of the compression achieved.

In Section 2 of this paper, we describe a compression scheme based on the idea of approximating a waveform by a piecewise linear function. Unlike existing methods for this problem, our method is *provably good* in that for specified error criterion, the number of pieces in the approximation is guaranteed to be *optimal*. Furthermore, the error can be set independently at each sample point, allowing the algorithm to be used adaptively. The algorithm makes no assumptions on the nature of the source and can be used in a variety of contexts such as speech, medical data, and images.

In Section 3 we describe the use of this algorithm for compressing image data. A simplification to the methods described in Section 2 and Section 3 is described in Section 4. A hardware implementation of the simplified algorithm is presented in Section 5. Performance figures for the optimal and the simplified waveform compression methods are presented in Section 6. Proposals for future extensions of this algorithm are discussed in Section 7.

2. The Basic Algorithm

Let $\mathbf{R}$ be the set of reals and let $[0,1]$ denote the interval on $\mathbf{R}$. We consider piecewise linear functions $F:[0,1]\rightarrow\mathbf{R}^k$, for some fixed natural number k. For instance, in the case of a color image, F maps each pixel to three color values and hence can be viewed as a function from $[0,1]$ to $\mathbf{R}^3$. F is specified as a sequence of points from $[0,1]\times\mathbf{R}^k$ and is the function described by the line segments joining successive points in the given sequence.

Problem: Given a piecewise linear function $F : [0, 1] \rightarrow \mathbf{R}^k$ specified as the interpolant of N sample points, and an error tolerance $\varepsilon \in \mathbf{R}^k$, construct a piecewise linear function G such that $G \underset{\varepsilon}{\sim} F$, i.e., for all $x \in [1, N]$, $\| F(x) - G(x) \|_\infty \leq \varepsilon$ and G consists of the fewest number of segments over all such functions.

This problem has been studied extensively in the literature for the case $k = 1$. See for example Sklansky and Gonzalez [3], Roberge [4], Ishijama et al. [5], and Rosenberg [1]. The methods reported in these papers are heuristics, and are not accompanied by provable guarantees of the optimality of the number of segments. Ihm and Naylor [6], Toussaint [7], and Imai and Iri [8], offer optimal solutions to a restricted version of the problem, where the approximation G must be defined as the interpolant of a subset of the sample points defining F. For the case of a function in one dimension ($k = 1$), Imai and Iri [9] give an optimal algorithm that runs in $O(N)$ time. The same algorithm can also be deduced from the general results of Suri [10] on visibility polygons, as noted in Natarajan [11]. The latter paper also generalizes this algorithm for $k > 1$. The algorithm is approximate but provably good in the following sense: for vector valued functions $F : [0, 1] \rightarrow \mathbf{R}^k$ the algorithm runs in time $O(kN)$ and produces an approximation G consisting of no more than twice as many segments as the optimal approximation.

The optimal algorithms [9], [11], are based on visibility techniques and work as follows. For each point (x, y) defining F, the algorithm constructs the points $(x, y + \varepsilon)$ and $(x, y - \varepsilon)$.

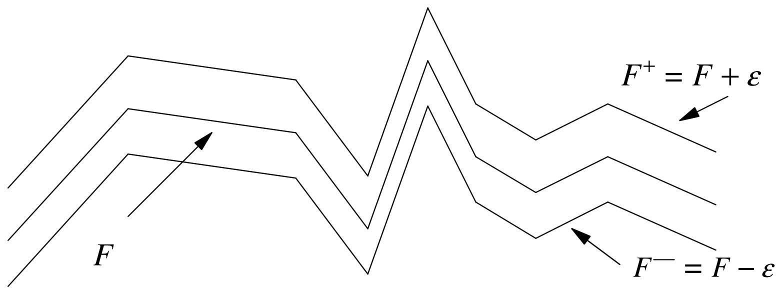

Fig. 1: Error tunnel for function F and error ε.

This creates an error tunnel as shown in Fig. 1. Note that Fig. 1 assumes that F is a scalar valued function, resulting in a tunnel in two dimensions. In general, F will be vector valued in k dimensions with an error tunnel in $k + 1$ dimensions. The algorithm then constructs a piecewise linear path connecting the two ends of the tunnel. For scalar valued functions, this path will consist of a minimum number of segments over all such paths, and will therefore result in an optimal piecewise linear approximation to given function F. For vector valued functions, the number of segments in the path will be within a factor of two of the minimum, regardless of the dimension. In either case, the run time of the algorithm will be $O(kN)$.

In fact, the above mentioned algorithm can solve the more general problem where each input point of F has an error tolerance that is set independently of the other points. In essence, the error tunnel mentioned above is given as the input. Formally, we can state the problem as follows.

Problem: Given two piecewise linear functions F^+ and F^- such that for all $x \in [0,1]$, $F^-(x) \le F^+(x)$, construct a piecewise linear function G such that for all $x \in [0,1]$, $F^-(x) \le G(x) \le F^+(x)$ and G consists of the fewest number of segments over all such functions.

This generalization allows the error tolerance to be set adaptively. The run time of the algorithm remains $O(kN)$ for functions in k dimensions. A complete description of the optimum waveform compression algorithm is given in the Appendix.

Fig. 2 is a schematic showing the steps involved in applying the algorithm to waveform com-

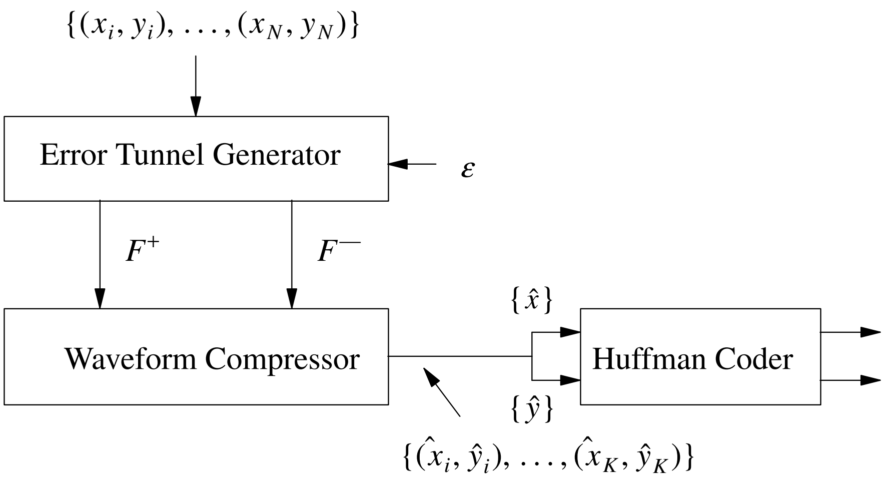

Fig. 2: The waveform compression scheme.

pression. Referring to Fig. 2, to improve overall compression, the sequences $\{\hat{x}\}$ and $\{\hat{y}\}$ are entropy coded using Huffman coders. Decompression to recover the value of the function at any point is achieved by first decoding the Huffman encoding, and then linearly interpolating between the $\{\hat{x}_i, \hat{y}_i\}$. Note that $\{\hat{x}\}$ need not be a subset of the $\{x\}$, unlike previous waveform coders such as the one discussed in [1]. This additional degree of freedom allows the algorithm to achieve an optimal number of segments for a given ε.

For the compression method depicted in Fig. 2, we define compression ratio to be $Nb/(K\hat{b})$ where N is the number of input samples, K is number of output samples, and b and $\hat{b}$ are the number of bits required to represent each input and output sample respectively.

3. Waveform Compression - Images

By considering a two-dimensional image as a collection of independent one-dimensional scanlines, a waveform compression scheme can be directly adopted for image compression. In order to exploit the two-dimensional correlations of the image, it is necessary to scan the image in a somewhat more sophisticated fashion. While several schemes were studied, few offer significant advantages. For instance, the Peano scan performs poorly [1]. We adopt the following scanning method, which is simple but performs well in experiments. Let I be an $N \times N$ matrix of pixel entries defining the image. Let I_i stand for the i^{th} row (scanline), and $I_i(j)$ the (i, j)-th entry.

Compression
input: ε;
begin
$i = 0$;
while $i \leq N$ **do**
    compress I_i to tolerance ε.
    let J_i be the decompressed form of I_i.
    $j = i + 1$;

    while $\sum_{l=1}^{l=N} |I_j(l) - J_i(l)| \leq \varepsilon N$ **do**
        $j = j + 1$; /* scanline j ignored */
    end
    $i = j$;
  end
end

Decompression
begin
for each scanline i that was compressed **do**
    decompress to obtain J_i;
    end
for each scanline i that was not compressed **do**
    find closest enclosing scanlines that were compressed,
    linearly interpolate between them to obtain J_i.
    end
end

If the above scheme is used as is, the decompressed image tends to be blocky. To overcome this, we bound the number of consecutive unencoded scanlines. Simulations also indicated that a post-processing filter with a strong smoothing effect along diagonal axes improves image quality.

4. Description of the modified algorithm

While the optimal line compression algorithm presented in the previous section yields the minimum number of output segments, its arithmetic complexity preempts real-time performance on a general purpose processor or an efficient custom VLSI implementation. In [12], the authors presented a modified algorithm that is much better suited for real-time applications and retains all properties of the optimal algorithm, except that the number of output segments is at most twice the optimal number. The modified algorithm uses a simpler visibility test that allows the algorithm to be implemented in an efficient and pipelined manner and eliminates the need for the complex data structures required in the optimal algorithm. Furthermore, simulation results showed that in practice the simplified algorithm is only about 1.5 times worse than the optimal in compression ratios.

Consider the sequence of data samples $\{x_{i,}\ f_i^+\}$ and $\{x_{i,}\ f_i^-\}$, for $i = 1, 2, \ldots, N$, representing the upper and lower envelopes of the error tunnel. Denote by s the starting index point of the portion of the waveform under consideration at some stage p of the algorithm. Starting from (x_s, f_s), the algorithm tries to draw tangents to increasingly longer prefixes of the upper and lower envelopes. Specifically, initially the upper and lower envelopes are truncated to the first two points. The algorithm then draws a tangent to each of the envelopes from the point (x_s, f_s). It then seeks to extend each envelope to the next point, and update their tangents to include them. Denote by $k-1$ the index point representing the longest prefix of the error tunnel separable by tangents $a_h x + b_h$ and $a_l x + b_l$ passing through $(x_s,\ f_s)$ (see Fig. 3). The algorithm first checks if it is possible to include the next point, x_k, in the tunnel prefix by determining if f_k^+ is above the lower tangent and f_k^- is below the upper tangent. This is accomplished by testing the signs of

$$S_1(k) = f_k^+ - (a_l\ x_k + b_l) = f_k^+ - f_s\ - a_l\ (x_k - x_s)\ \ , \tag{1a}$$

$$S_2(k) = f_k^- - (a_h\ x_k + b_h) = f_k^- - f_s - a_h\ (x_k - x_s)\ \ . \tag{1b}$$

If any of these conditions is not met, then the algorithm terminates and outputs the coordinate pair $(x_{k-1},\ g_p(x_{k-1}))$, where

$$g_p(x_{k-1}) = a_l\ x_{k-1} + b_l = a_l\ (x_{k-1} - x_s) + f_s\ \ . \tag{2}$$

This pair then becomes the starting point for the next stage of the algorithm. If none of the termination tests fails, then the algorithm continues by determining if any of the tangent slopes needs to be updated. The upper tangent is updated if f_k^+ is below the upper tangent and the lower tangent is updated if f_k^- is above the lower tangent. Thus, the algorithm tests the signs of

$$S_3(k) = f_k^+ - (a_h\ x_k + b_h) = f_k^+ - f_s - a_h\ (x_k - x_s)\ \ , \tag{3a}$$

$$S_4(k) = f_k^- - (a_l\ x_k + b_l) = f_k^- - f_s - a_l\ (x_k - x_s)\ \ . \tag{3b}$$

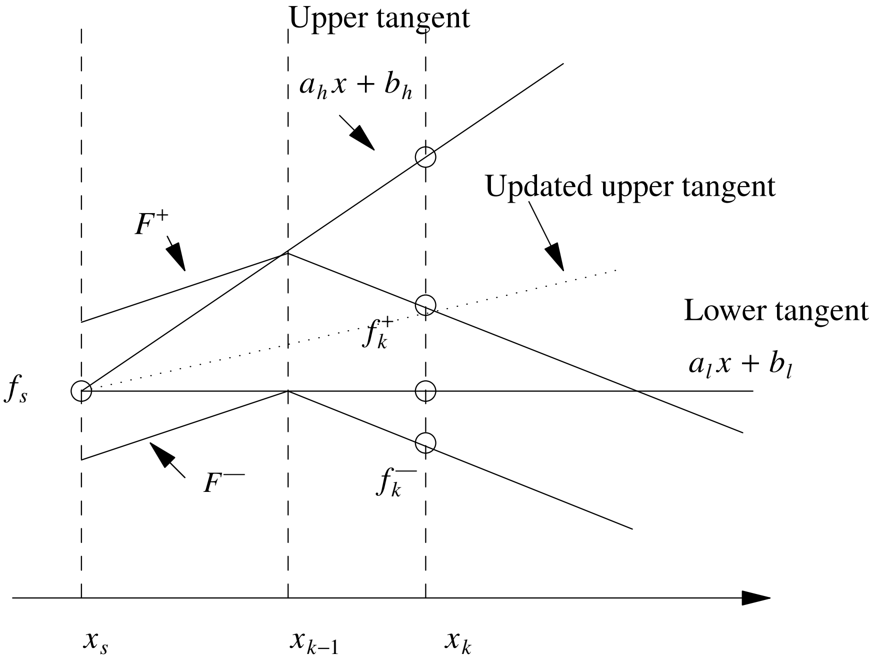

Fig. 3: Upper and lower tangents in the line-compression algorithm.

If any of these tests fails (for example, in Fig. 3, f_k^+ is below the upper tangent), then it computes new tangents

$$\hat{a}_h = \frac{f_k^+ - f_s}{x_k - x_s} \quad , \qquad \hat{a}_l = \frac{f_k^- - f_s}{x_k - x_s} \quad . \tag{4}$$

Equations (1)-(4) represent the main computation requirements for the modified compression algorithm. Note that for uniform sampled intervals (as in image data), (4) can be simplified as in

$$\hat{a}_h = (\, f_k^+ - f_s \,)\, C_k \quad , \qquad \hat{a}_l = (\, f_k^- - f_s \,)\, C_k \quad , \tag{5}$$

where $C_k = 1 / (x_k - x_s) = 1 \,/\, d\ (k - s)$ can be determined from an internal ROM table. The size of the ROM table depends on the maximum value of $(k - s)$. However, in practice, it does not need to be larger than the maximum expected compression ratio.

5. Processor architecture

Following the description of the image compression algorithm in section 3, its hardware implementation needs to address three major computation steps: (a) the compression of an image line I_i, (b) the generation of its decompressed approximation J_i, and (c) the evaluation of $\sum_{l=1}^{l=N} \left| I_j(l) - J_i(l) \right|$ to determine if the next image line needs to be compressed or not.

5.1 Image line compression

There are many possible implementations of equations (1)-(4), depending on the speed requirements and cost constraints of our application. For example, implementations may range from programming the algorithm into a general purpose processor, to a fully parallel and custom architecture with two multipliers and eight adders. In this section, we present a hardware implementation that is simple enough for a single-chip implementation, but adequately fast for many real-time applications.

Fig. 4 shows a block diagram of the arithmetic processing unit and the register file for the

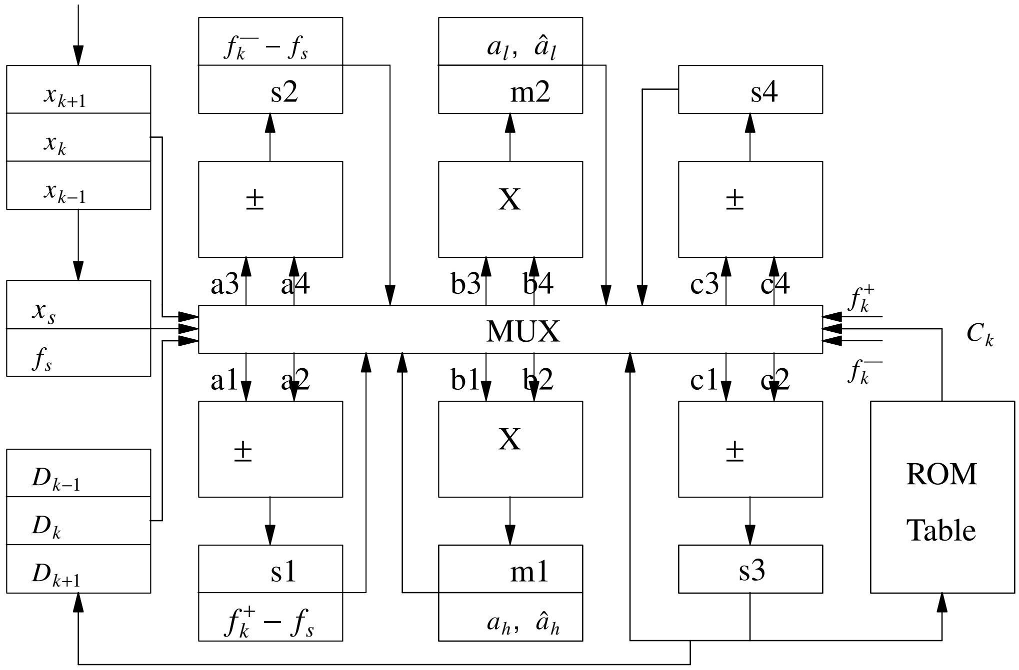

Fig. 4: Block diagram of the arithmetic and register file unit.

implementation of the modified compression algorithm. It consists of two multiplier units, four adder/subtracter units, one look-up ROM table, data registers, and input multiplexors.

The main feature of the above design is the use of four adder/subtracter units operating in parallel. Evaluation of (1) and (3) requires a minimum of two multiplications and seven subtractions. This design allows, in pipelined mode, the computation of (1) and (3) in two cycles, with a three-cycle initial pipeline delay. Table 1 shows the data flow of operations for equations (1), (3), and (5). Note that $D_k = x_k - x_s$, for $k = 1, 2, \ldots, N$, and that D_{k+1} is computed in parallel with the other tests. Every two cycles, the controller can perform all sign tests and evaluate new tangent slopes, even though they may not be needed. If any of the continuation tests (S_1 or S_2) fails, then the rest of the computations in the pipeline are discarded, the controller initiates an output of the $g_p(\,.\,)$ value, and the computational cycle restarts for the remaining error tunnel. If any of the tangents needs to be updated, then the processor uses the updated value and computations continue with a one cycle pipe-delay. For example, if test $S_3(k)$ fails at time $n+2$ (see Table 1), then an extra cycle $(\hat{n}+2)$ is needed between the old $n+2$ and $n+3$ cycles, because a new upper tangent, with slope $\hat{a}_h$ needs to

Table 1

Pipelined Data Flow in Line Compression Algorithm

	Clock				
Inputs	n	n+1	n+2	n+3	n+4
a1 a2	f_k^+ f_s	s1 m1	f_{k+1}^+ f_s	s1 m1	
b1 b2	a_l D_k	s1 C_k	a_l D_{k+1}	s1 C_{k+1}	
c1 c2	x_{k+1} x_s	s1 m2	x_{k+2} x_s	s1 m2	
a3 a4	f_k^- f_s	s2 m2	f_{k+1}^- f_s	s2 m2	
b3 b4	a_h D_k	s2 C_k	a_h D_{k+1}	s2 C_{k+1}	
c3 c4		s2 m1		s2 m1	
Outputs					
s1		$f_k^+ - f_s$	$S_1(k)$	$f_{k+1}^+ - f_s$	$S_1(k+1)$
m1		$a_l D_k$	$\hat{a}_h$	$a_l D_{k+1}$	$\hat{a}_h$
s3		D_{k+1}	$S_3(k)$	D_{k+2}	$S_3(k+1)$
s2		$f_k^- - f_s$	$S_2(k)$	$f_{k+1}^- - f_s$	$S_2(k+1)$
m2		$a_h D_k$	$\hat{a}_l$	$a_h D_{k+1}$	$\hat{a}_l$
s4			$S_4(k)$		$S_4(k+1)$

be used. At $\hat{n}+2$, $a_h = \hat{a}_h$, and the product $a_h D_{k+1}$ is re-evaluated.

5.2 Image line decompression

After line I_i has been compressed, the next step is to generate its approximation J_i from the compressed samples. Given two consecutive output pairs $(x_a,\, g_p(x_a))$ and $(x_b,\, g_p(x_b))$ from the compression algorithm, the in-between samples are approximated using linear interpolation, i.e.,

$$g_p(x_j) = a_l\,(x_j - x_a) + g_p(x_a)\ , \quad \text{for } j = a+1,\, a+2,\, \ldots,\, b-1\ , \tag{6}$$

where $a_l = (g_p(x_b) - g_p(x_a))/(x_b - x_a) = (g_p(x_b) - g_p(x_a))\; C_{b-a}$. The arithmetic unit of Fig. 4 can then be used to evaluate the above expressions using one multiplier, two adders, and the ROM table. For example, Table 2 shows one possible pipeline flow of operations for evaluating (6). After an initial pipeline delay needed to compute the line slope a_l and $g_p(x_{a+1})$, a

Table 2

Pipeline Data Flow in Line Decompression Algorithm

	Clock			
Inputs	n	n+1	n+2	n+3
a1 a2		$g_p(x_a)$ m1	$g_p(x_a)$ m1	
b1 b2	a_l C_1	a_l C_2	a_l C_3	
c1 c2	x_{a+3} x_a	x_{a+4} x_a	x_{a+5} x_a	
Outputs				
s1			$g_p(x_{a+1})$	$g_p(x_{a+2})$
m1	a_l	a_l C_1	a_l C_2	a_l C_3
s3	D_2	D_3	D_4	D_5

new $g_p(\, . \,)$ value is available every single cycle. In Table 2, $D_k = x_{a+k} - x_a$.

5.3 Scanline compression test

According to the compression algorithm, scanline j is ignored if

$$S(j) = \sum_{l=1}^{l=N} \left| I_j(l) - J_i(l) \right| \leq \varepsilon \, N. \tag{7}$$

Again, our arithmetic unit is very well suited for that operation using three of the available adders. Two of the adders can be used to evaluate the absolute value by computing both $I_j(l) - J_i(l)$, and $J_i(l) - I_j(l)$, and selecting the positive output, while the third adder evaluates the running sum

$$S(j) = S(j) + \left| I_j(l) - J_i(l) \right| \ , \qquad l = 1, 2, \ldots, N \ . \tag{8}$$

Hence, (8) can be computed in $N + 2$ cycles (including the initial two-cycle pipeline delay).

5.4 Processor architecture

Fig. 5 shows a block diagram of the complete design for the image compression algorithm. In addition to the arithmetic unit, the design includes a PLA for control and address generation, an I/O unit for interfacing with external memory and a host microprocessor, and three local memories. The I memory is used for storing a scan line of the input image. The G memory contains the compressed data, and the J memory is used to store the reconstructed scan line from the compressed data. A dual adder/subtracter is used to generate on the fly the $I + \varepsilon$ (F^{+}) and $I - \varepsilon$ (F^{-}) data needed in the compression algorithm. This adder could be integrated into the arithmetic unit, or could be eliminated, provided those data is input directly from the host processor.

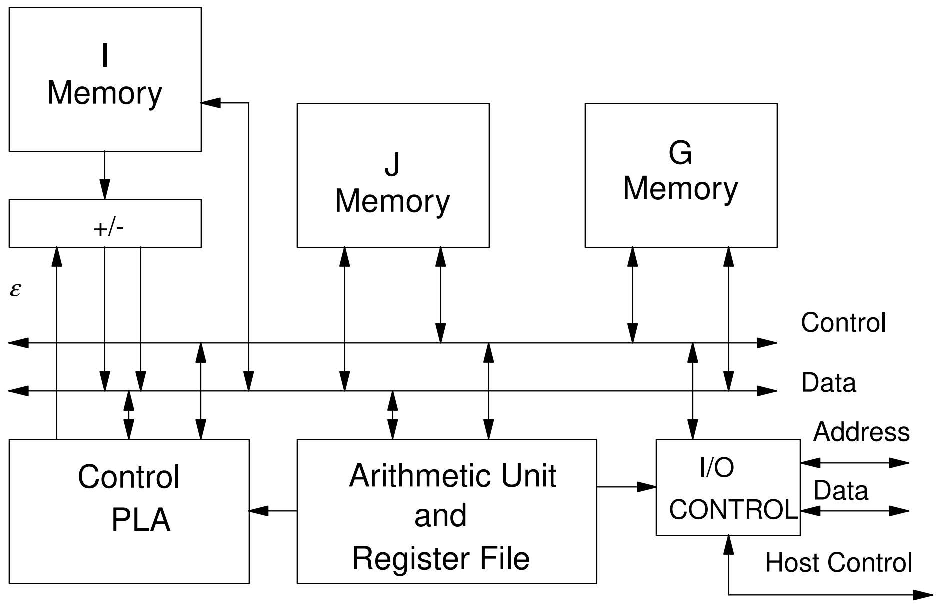

Fig. 5: Block diagram of the image compression architecture.

6. Simulation Results

For purposes of illustration, as a test image, we have used a 512 x 512 8 bits/pixel grayscale image referred to as *Lena* in the image compression literature. We have performed tests with other grayscale images and the results are similar to that obtained with *Lena*. Fig. 6, shows the rate-distortion performance for the proposed compression scheme. For distortion, we use the Peak to Peak Signal to Noise Ratio (PSNR), which is defined as the ratio of the peak-signal to the rms-error. Results for the optimal waveform compression method of Section 3 and the suboptimal scheme of Section 4 are depicted. Note that the suboptimal scheme is slightly inferior to the optimal scheme - typically the compression ratio is worsened by 20 - 30 %. For comparison, we also show the performance of the JPEG standard for still-image compression. The JPEG scheme achieves higher compression for same PSNR as the waveform coding schemes. This is due to the fact that the JPEG coder is a two- dimensional coder whereas the waveform coder is a one-dimensional scheme. However, the JPEG coder is considerably more complex and in devices such as grayscale printers, the simpler decompression procedure of our waveform compression scheme might be preferable. In Fig. 6, we also show the rate-distortion bound for the test image. The rate-distortion bound is computed by imposing a two-dimensional auto-regressive model on the image and then integrating the two-dimensional power spectral density function [13].

In Fig. 7, we show the test image compressed by the waveform coding and JPEG methods. For the optimal and suboptimal waveform compression schemes, we compress the image

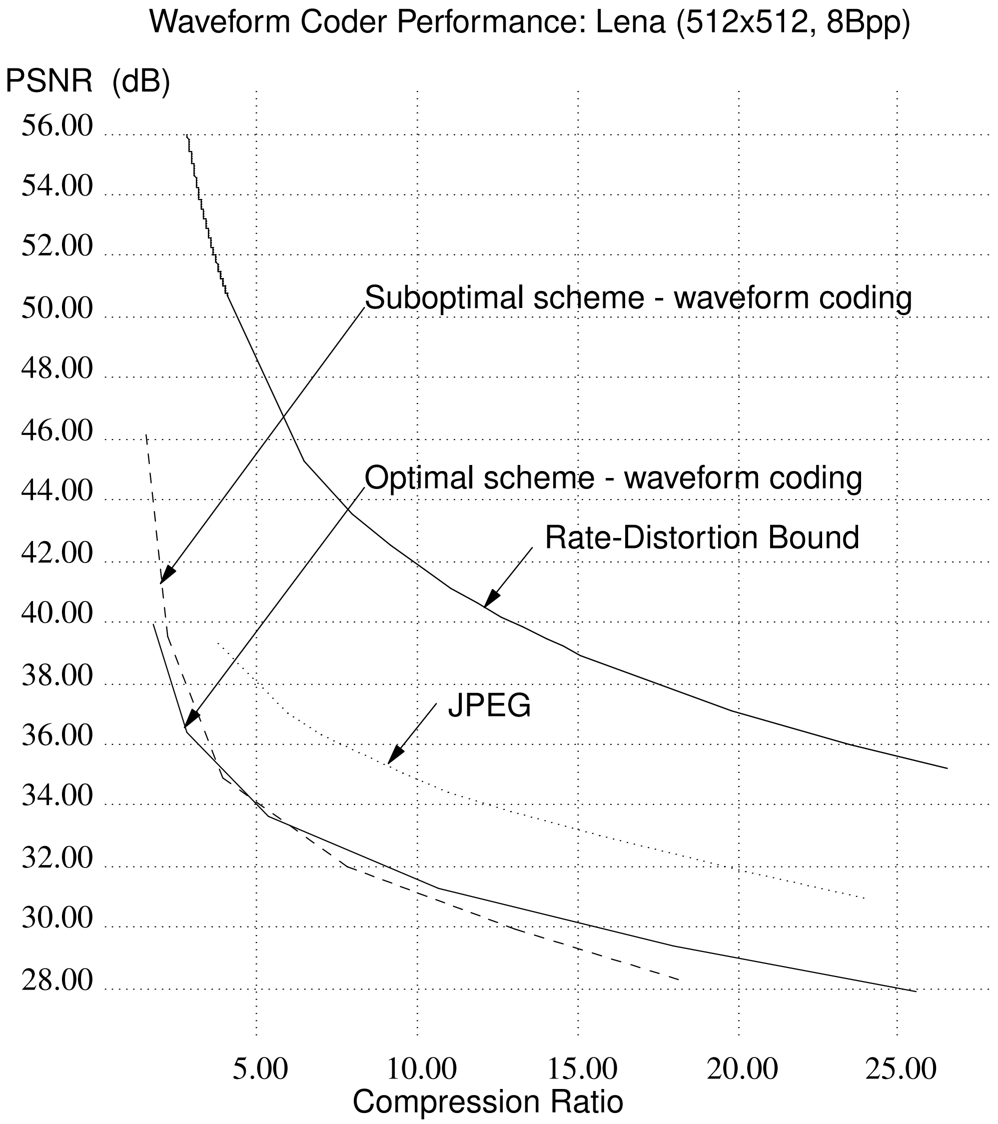

Fig. 6: Rate versus distortion for grayscale image using waveform compression schemes (optimal and suboptimal) and JPEG scheme.

with the same $\varepsilon = 0.04$. The waveform coder places samples as shown in Fig. 7c and Fig. 7f. Note that the suboptimal scheme selects more samples from the edge regions of the image. Examination of the error images indicates fewer errors in the edge regions for the suboptimal waveform compression method. In Fig. 7c and Fig. 7f, we see some horizontal dark bands. These bands correspond to scanlines in the image that were not encoded since these lines were within the error threshold of previously encoded scanlines. For comparison, we show the JPEG scheme at the same signal-to-noise ratio. Comparing decompression results with that obtained for the JPEG scheme indicates better edge reconstruction for the waveform coding method. However, the JPEG compression method is operating at nearly two to three times higher compression ratio. Due to the flexibility in modifying the error tolerance ε on a sample-by-sample basis in the waveform compression scheme, edge fidelity can be improved by adjusting ε based on further preprocessing such as using the output of an edge detector to reduce ε in edge regions.

7. Conclusions and Future Directions

We presented a compression algorithm developed from a geometric viewpoint. Specifically, the method constructs an optimum piecewise linear approximation to a given waveform within a specified error tolerance.

As discussed in this paper, the simplicity of the algorithm makes it particularly suitable for a hardware implementation. For instance, in a printer application, decompression can be trivially implemented in a print engine. Furthermore, since the method uses sample points to represent the compressed data, scaling of the image during printing or resolution conversion for printing can be accomplished by combining the scaling and decompression function in the print engine.

From an algorithm viewpoint, further investigation is necessary to examine methods of using context-sensitive features in the data to construct the error tunnel so that the error tunnel adapts to the data. For vector valued waveforms, such as color images further studies need to be performed to assess the efficacy of applying the compression scheme independently to each dimension, or use the approximation algorithm for vector valued functions described in Section 2. The problem of optimal piecewise linear tessellations to functions of several independent variables remains open. A solution to this problem would lend itself to compressing images by treating the image as a two-dimensional sequence rather than using the scanning procedure we have adopted in the present work.

8. Appendix

In this section we describe the optimum waveform compression algorithm. We first describe the algorithm for functions in one-dimension, and then extend it to univariate functions in higher dimensions. The algorithm consists of three major steps that are applied iteratively. We will describe the algorithm in terms of these steps, and then suggest how the steps may be implemented.

Let sequences $(u_1, u_2, \ldots, u_m)$ and $(l_1, l_2, \ldots, l_n)$ determine the upper and lower envelopes of the tunnel as shown in Fig. A1. A *separating* tangent of the tunnel is a straight line that touches both the upper and lower envelopes of the tunnel and lies between them. Fig. A1 shows the separating tangents for that portion of the tunnel defined by the first five vertices. Note, that there are no separating tangents for the first six vertices.

Compression Algorithm
input: Error tunnel.
begin
while the error tunnel is not empty

(1) Find the largest index i such that there exists separating tangents for the first i vertices of the error tunnel.

(2) Of these two tangents, pick the one that sees farther. Add this tangent to the piecewise linear output of the algorithm.

(3) "Cut" the tunnel with this tangent and retain the latter portion.
end

end

Of the two tangents A and B shown in Fig A1, tangent B sees farther into the tunnel in that it projects further than A. Fig. A2 shows the result of cutting the tunnel with tangent B. The algorithm is then applied iteratively to the portion of the tunnel that is to the right of B.

We now describe how step (1) may be achieved. Suppose that for a given value of i, we can determine in $O(m+n)$ time whether or not separating tangents exist for the first i vertices of the tunnel. Then, using binary search, the maximum such i can be determined in $O((m+n)log(m+n))$. In particular, we know that separating tangents always exist for the first two vertices ($i = 2$). We then check for $i = 4, 8, 16$, etc, until we find an $i = k$ for which no separating tangents exist. We then know that the maximum value of i lies between $k/2$ and k. By repeatedly halving this interval, we can isolate the value of i.

The following algorithm describes how to determine whether or not separating tangents exist for a particular value of i. As one candidate for the tangent, we join u_1 and l_n. As the other candidate, we join l_1 and u_m. We now adjust these candidates to check whether they are feasible. Since the two tangents are treated in the same fashion, we limit the rest of the discussion to just one of them, say the one joining u_1 and l_n. The following procedure adjusts the candidate to obtain a separating tangent if one exists.

Begin

$i1 = i = 1$;

$j1 = j = n$;

done = *FALSE*;

while *done* is FALSE **do**

done = *TRUE*;

if u_i is to the right of l_j **then**

no tangent exists; halt;

end

find least k in $i1, i1 + 1, \ldots, m$ such that

u_k is below the line (u_i, l_j);

if such k exists **then**

$i1 = i = k$;

done = *FALSE*;

end

else $i1 = m$;

find greatest k in $j1, j1 - 1, \ldots, 1$ such that

l_j is above the line (u_i, l_j).

if such k exists **then**

$j1 = j = k$;

done = *FALSE*;

end

else $j1 = 1$;

end

report (u_i, l_j) as a separating tangent.

end

Step (1) of the the compression algorithm can also be implemented in $O(n)$ time, using the methods of Imai and Iri [9] and Suri [10]. However, we will not describe such an implementation here, since it is considerably more complex, and involves the maintenance of the convex hull of the upper and lower envelopes of the tunnel.

For functions in higher dimensions, we build a tunnel in each of the dimensions, and find the greatest number i of vertices for which a separating tangent exists in each of the tunnels. The first segment in the piecewise linear approximation is obtained by picking one separating tangent from each tunnel and composing them to form a single line in the higher dimensional space. We then cut off the first i vertices of each of the tunnels and repeat the procedure.

REFERENCES

1. C. Rosenberg, *A lossy image compression algorithm based on non-uniform sampling and interpolation of image intensity surfaces.* M.S. Thesis, Dept. of Electrical Engineering and Computer Science, Mass. Inst. of Tech., 1990.
2. E. Walach, E. Karnin, "A fractal based approach to image compression," Proc. of Int. Conf. on Acoustics, Speech and Signal Processing, pp. 529-533, 1986.
3. J. Sklansky, and V. Gonzalez, "Fast polygonal approximation of digital curves," Pattern Recognition, Vol. 12, pp. 327-331, 1980.
4. J. Roberge , "A data reduction algorithm for plain curves," Computer Vision, Graphics and Image Processing, Vol. 29, pp. 168-195, 1985.
5. M. Ishijama , S. B. Shin, G. H. Hostetter, and J. Sklansky, "Scan-along polygonal approximation for data compression of electrocardiograms," IEEE Trans. on Biomedical Eng., Vol. BME-30, No. 11, pp. 723-729, 1983.
6. I. Ihm, and B. Naylor, "Piecewise linear approximations of digitized curves with applications," *Scientific Visualization of Phyical Phenomena*, pp. 545-568, N.M Patrilakis, editor, Springer-Verlag, New York, 1991.
7. G. T. Toussaint, "On the complexity of approximating polygonal curves in the plane," In Proc. of the IASTED International Symposium on Robotics and Automation, pp. 59-62, 1985.
8. H. Imai, and M. Iri, "An optimal algorithm for approximating a piecewise linear function," Journal of Information Processing, Vol. 9, No. 3, pp. 159-162, 1986.
9. H. Imai, and M. Iri, "Computational geometric methods for polygonal approximations of a curve," Computer Vision, Graphics and Image Processing, Vol. 36, pp.31-41, 1986.
10. S. Suri, "On some link distance problems in a simple polygon," IEEE Trans. on Robotics and Automation, vol 6, No.1, pp. 108-113, 1988.
11. B. K. Natarajan , "On piece-wise linear approximations to curves," SIAM Conference on Geometric Design, 1991.
12. K. Konstantinides and B. K. Natarajan, "An architecture for lossy signal compression," *VLSI Signal Processing V*, K. Yao et al. Editors, pp. 237-246, IEEE Press, 1992.
13. D. G. Daut, R. W. Fries, J. W. Modestino, "Two-dimensional DPCM image coding based on a assumed stochastic image model," IEEE Trans. Commun., Vol. COM-29, pp. 1365-1374, 1981.

Fig. 7: Image compression results at 32 dB PSNR.

Left to right, top to bottom: a) Decompressed image using optimal waveform compression. Compression ratio = 10.65, ε=0.04. b) Error image. c) Optimal compressor's output samples, shown as white pixels. d) Decompressed image using suboptimal waveform compression. Compression ratio = 7.78, ε=0.04. e) Error image. f) Suboptimal compressor's output samples. g) JPEG decompressed image. Compression ratio = 20.23. h) Error image. i) Original image.

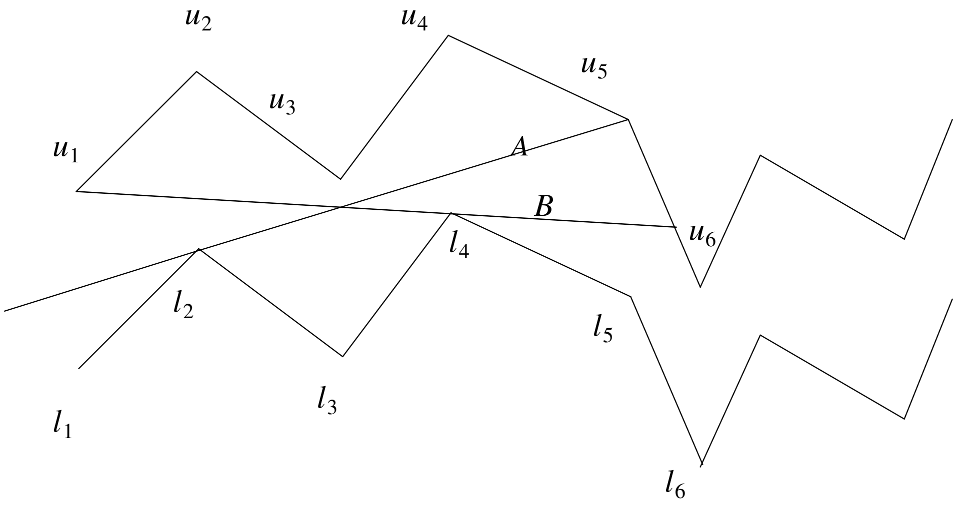

Fig. A1: The separating tangents for the first four vertices of the error tunnel.

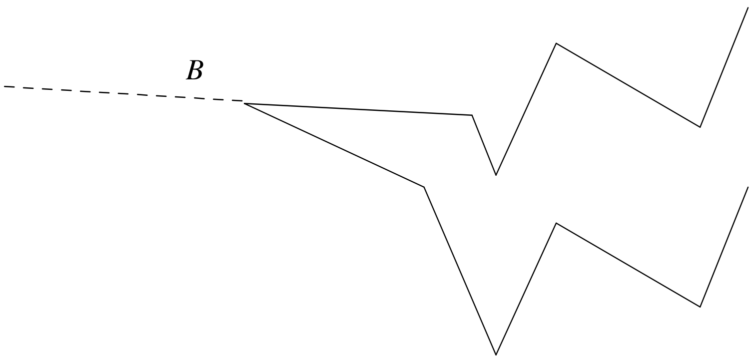

Fig. A2: The error tunnel after cutting by tangent B.