Locally Adaptive Vector Quantization: Data Compression With Feature Preservation

K.-M. Cheung
Communications Systems Research Section

M. Sayano
California Institute of Technology

This article presents a study of a locally adaptive vector quantization (LAVQ) algorithm for data compression. This algorithm provides high-speed one-pass compression and is fully adaptable to any data source and does not require a priori knowledge of the source statistics. Therefore, LAVQ is a universal data compression algorithm. The basic algorithm and several modifications to improve performance are discussed. These modifications are nonlinear quantization, coarse quantization of the codebook, and lossless compression of the output. Performance of LAVQ on various images using irreversible (lossy) coding is comparable to that of the Linde-Buzo-Gray algorithm, but LAVQ has a much higher speed; thus this algorithm has potential for real-time video compression. Unlike most other image compression algorithms, LAVQ preserves fine detail in images. LAVQ's performance as a lossless data compression algorithm is comparable to that of Lempel-Ziv-based algorithms, but LAVQ uses far less memory during the coding process.

I. Introduction

Data compression is the art of packing data, the process of transforming a body of data to a smaller representation from which the original or an approximation to the original can be computed at a later time. Most data sources contain redundancies such as nonuniform symbol distribution, pattern repetition, and positional redundancy. A data compression algorithm encodes the data to reduce these redundancies.

Data compression has not been a standard feature in most communication/storage systems for the following reasons: Compression increases the software and/or hardware cost; compression/decompression is difficult to incorporate into high data rate (greater than 10 Mb/sec) systems; most compression algorithms are not flexible enough to process different types of data; the unpredictability of compressed data file size presents space allocation problems. These obstacles are less significant today due to recent advances in algorithm development, high-speed very large-scale integrated circuit (VLSI) technology, and packet switching communications. Data compression is now a feasible option for those communication or storage systems for which communication bandwidth and/or storage capacity are at a premium. If present trends continue, the volume of speech and image data in the near future will become prohibitively large for many communication links or storage devices.

A number of applications require data compression for efficient data storage. To facilitate fast processing, maintain accurate records, and recall old records quickly, a number of businesses are optically scanning their documents and saving them on magnetic media. This takes up large amounts of memory; efficient storage requires data compression. Documents are stored for archival purposes, so a short delay in retrieval (decompression) is not detrimental. A similar application exists in law enforcement, security, and intelligence agencies, where facial, fingerprint, and other images are kept on file for fast retrieval, analysis, and matching.

Data compression is required also in limited bandwidth communications; the two best examples of this are video telephony and high-definition television (HDTV). Video telephony requires transmission of images over a small bandwidth; this can be as small as 4 kHz in the standard voice communication channel. Sending an image, even a small one, without data compression is not feasible. For HDTV, data compression is also needed if signals are to be sent digitally over a standard television channel: HDTV signals require roughly four times the bandwidth of standard TV signals. High-speed algorithms which compress the image without substantially degrading image quality are requirements for both video telephony and HDTV. As more and more information must be transmitted over the same size bandwidth, data compression becomes imperative to maintain transmission rate and data fidelity.

Vector quantization (VQ) is an efficient data compression technique for speech and images. VQ maps a sequence of continuous or discrete vectors into a digital sequence suitable for transmission over a digital channel or storage in a digital medium. The goal is to reduce the volume of data while preserving required fidelity levels. In [1], a well-designed VQ scheme was shown capable of providing high compression ratio with good reconstructed quality.

Unlike scalar quantization where the actual coding of continuous or discrete samples into discrete quantities is done on single samples, the input data of a VQ encoder are multidimensional blocks of data (input vectors). An important technique in VQ is the training of codebooks prior to transmission [1]. Extensive preprocessing is performed on sample source data to construct the codebook to be used in the compression session. The encoder and decoder must first agree on the same codebook before data transmission. The closeness between an input vector and a codeword in the codebook is measured by an appropiate distortion function. During the compression session, distortions between an input vector and codewords in the codebook are evaluated; the codeword closest to the input vector is chosen as the quantization vector to represent the input vector. The index of this chosen codeword is then transmitted through the channel. Compression is achieved since fewer bits are used to represent the codeword index than the quantized input data. The decoder receives the codeword index and reconstructs the transmitted data using the preselected codebook.

Traditional VQ schemes have a few inherent disadvantages. (1) The generation of a good codebook requires a priori knowledge of the source data, which in practice are not often easily available. (2) Traditional VQ schemes are static schemes. They assume that the statistical properties of the source data remain the same for all compression sessions and the codebooks are optimized based on this assumption. Real-world data tend to have varying characteristics, and static algorithms may not be efficient enough to process diverse sources. (3) Both codebook generation and codeword search for input vectors involve computing the distortion between input vectors and codewords in the codebook; these are usually computationally intensive processes, especially when the codebook size is large.

A new class of data compression algorithms, locally adaptive vector quantization (LAVQ) algorithms, was suggested in [2] and [3]. Unlike traditional VQ algorithms, LAVQ algorithms do not require a priori knowledge of the source, nor do they require the tedious process of codebook generation, as the codebook is generated on the fly during encoding, and the decoder mimics the operations of the encoder to maintain an identical codebook at all times. This algorithm does not require a full codebook search: The codebook is updated after each use to maintain the most recently used codewords at the front of the codebook; a codeword which is within the error allowance is typically found in the top one-fifth to one-tenth of the book through a sequential search. This algorithm dynamically adapts to the local features of the source and is particularly good in compressing sources with varying characteristics.

In this article the basic algorithm suggested in [2] and [3] will be described. Subsequent sections will be devoted to the analogy of LAVQ to a vector differential pulse code modulation (DPCM) algorithm, improvements to the basic LAVQ algorithm, and application to lossless data compression. The algorithm has been fully implemented in software; a brief description of this implementation is included. Experimental results on both lossy image coding and lossless data compression are also presented.

II. Basic Algorithm

The basic LAVQ algorithm provides a simple yet effective one-pass data compression strategy (refer to Fig. 1). The encoder has a codebook containing codewords (vectors) where the index of the codeword corresponds to its position in the codebook. A block is taken from the image and compared to the stored codewords; if there exists a codeword sufficiently close to the image block (within the error allowance), the index itself is sent, and that codeword is moved to the top of the codebook. If no such codeword exists, a special index is sent. This index is followed by the block itself. This block becomes a new codeword and is placed at the top of the codebook. All other codewords are pushed down, and if the number of codewords exceeds the maximum allowed, the last codeword is lost. Initially, the codebook may be empty or full from the previous image encoded.

On the decoder side, the decoder expects an index. If this index is the special one denoting that a new block was sent, the decoder expects a block to be received immediately following the special index; this block is placed at the top of the codebook and all other codewords are pushed down. If the codebook is already full, the last codeword is discarded. This new block is also placed into the image being built by the decoder. If the index is not one designating a new block, then the codeword corresponding to the index is put into the image being built, and that codeword is moved to the top of the codebook. Thus, if the encoder and decoder start with the same codebook, they will have the same codebook at each step, and the image will be successfully sent [2–4].

The LAVQ strategy maintains the most recently used vectors in the codebook in the order of last usage; this allows the algorithm to efficiently code any image on the fly without codebook training: The algorithm needs only one pass of the image to code it entirely. In serial implementation, LAVQ has time complexity $O(nm)$ and storage complexity $O(m)$, where n is the number of pixels in the image and m is the number of codewords in the codebook. Most of the time spent on encoding is taken by finding the closest codeword in the codebook and determing if that match is close enough. Rearranging the codebook and sending the required index, and possibly a new block, can be done quickly in serial implementation using lookup tables, linked lists, and other software techniques. To minimize the amount of time spent on codebook searching and to improve performance, a partial search of the codebook can be used: Instead of searching serially for the best match, the encoder can stop searching at the first instance of a close-enough match, with the criterion dictated by the error allowance given.

The basic LAVQ algorithm, however, has poor performance compared to traditional VQ strategies such as the Linde-Buzo-Gray (LBG). Several adjustments can be made to improve the algorithm without degrading the advantage of one-pass high-speed implementation. Two approaches can be used to improve rate. First, the statistics of the coded indices can be skewed toward small values (recent indices) by (1) a partial codebook search as outlined above, (2) using tall and narrow blocks ($N \times 1$ pixels) to make each block more similar to the blocks immediately previous to it in a raster scan, and (3) coding only the differential value of each block by removing the mean value and reinserting it later. Second, the number of bits used to send new codewords can be reduced by (1) reducing the number of bits used to describe each new pixel and (2) using nonlinear quantization of these new codeword values. These two approaches, combined with lossless adaptive arithmetic coding of the output, improve the performance of LAVQ to be comparable to that of LBG. These improvements will be discussed in detail in a subsequent section.

III. LAVQ as a Vector Analogy of DPCM

Differential pulse code modulation (DPCM) data compression algorithms are efficient and have low complexity. They are particularly effective in encoding gray-scale images, which are dominantly characterized by an autoregressive (AR) stochastic model or an autoregressive moving average (ARMA) model. DPCM operates on individual samples $x(n)$ and encodes the quantized difference $e(n)$ between a predicted value $\hat{x}(n)$ and $x(n)$. The prediction is based on the pixels neighboring $x(n)$. The error $e(n)$ tends to be small rather than large, and compression is achieved by assigning fewer bits to smaller values of $e(n)$ and more bits to larger values of $e(n)$.

From an information theoretic point of view, given a data source, it is always advantageous to encode vector quantities rather than scalar quantities. A vector extension of DPCM coding involves encoding a vector of differences $E = [e_1(n),\ e_2(n),\ \ldots,\ e_N(n)]$, where N is the vector size. However, the number of combinations of E increases exponentially with N; especially for large N, this quickly becomes too large to be practically feasible for efficient encoding. Another drawback of DPCM is its inflexibility: DPCM performs well when coding sources are characterized by the AR or ARMA models (e.g., speech and image data); however, for data sources dominantly characterized by pattern repetitions (e.g., data base records and engineering data), DPCM performs poorly.

LAVQ is analogous to DPCM in a sense: Both DPCM and LAVQ can be viewed as consisting of a preprocessing stage and a compression stage. DPCM preprocesses a sample by taking the difference between the sample value and its corresponding predicted values and sends the quantized difference to be entropy coded. LAVQ, on the other hand, preprocesses a vector of samples by matching it to the codeword vectors in a dynamic codebook followed by a move-to-front codebook update, and sends either a codebook index or an uncoded vector to the decoder. The major difference is that DPCM encodes the scalar difference, whereas LAVQ encodes the vector recency, which can be considered as a different measure of difference (vector difference). Thus, LAVQ uses the locally adaptive move-to-front preprocessing unit to convert the hidden statistical and correlational redundancy to a scalar statistical redundancy. This allows representation of vectors as scalars and approaches vector entropy, which, in the information theoretic sense, is smaller than the corresponding scalar entropy. This was proven in [2] and is verified in the results given here.

IV. Improvements to LAVQ for Image Compression

A. Index Coding

1. Difference Coding. The most significant visual artifact of LAVQ is the sawtooth or staircase effect. This occurs from using discrete vectors to represent a set of vectors within a threshold set by the error allowance; thus, differences between codewords can be large enough to be noticeable. In particular, if the pixels are slowly varying across the image, as is the case with most images, the encoded blocks do not track this variation closely. That is, adjacent blocks are coded with the same block while the amount of error is within the allowance; when that allowance is exceeded, suddenly a different block is used. This difference can be noticeable and, since most images have regions of slow variations or of constant color or intensity, quite common.

This assumption of images having regions of slowly varying or constant pixels implies that adjacent blocks have similar mean values. Thus, the mean can be removed, and only the differential values of the image can be coded. Each vector now represents the difference between the actual pixel value and a reference value equal to the distorted mean of the previously coded block. This distorted mean is a valid choice of reference because both the encoder and decoder can compute it exactly from the previous block's distorted pixel values. The small difference vectors are more likely to be well approximated by the recently occurring difference vectors maintained in the current codebook. Thus, slowly varying or constant regions can be more accurately coded without extensive use of new codewords. The mean must be updated after each block is processed at both the encoder and the decoder. This ensures that both have the same mean to remove and to reinsert into the image at each step.

2. One-Pass Index Compression. Because of the move-to-front codebook rearrangement strategy and because most images have similar adjacent blocks, the smallest indices are most likely to be used more often. Therefore, they can be coded using a lossless compression code to obtain better performance. However, to maintain one-pass compression, the lossless code must also be one-pass. Furthermore, the statistics of the coded indices are unknown a priori, and no assumption can be made regarding them.

The code which yields the most promising results with minimal increase in computational complexity is the adaptive arithmetic code. This algorithm is the static arithmetic code implemented with probabilities of each symbol updated after each use. By starting with the same initial distribution at the encoder and decoder, lossless coding can be obtained. The arithmetic code approaches global symbol entropy closely; in some cases, it does even better: The average entropy of the adaptive arithmetic code is average *local* entropy based on symbol statistics from the start to the symbol being coded. Global entropy is derived from the statistics of each symbol based on the entire sequence; local entropy is derived from the statistics of each symbol based on part of the sequence in the neighborhood of the symbol being coded. The global entropy is always greater than or equal to the global average of all the local entropies. Thus, for sources which have localized characteristics which vary throughout the sequence, using localized adaptive coding methods is more advantageous than using global, nonadaptive coding methods.

B. Codeword Data Coding

1. Bit Stripping. Bit stripping of new codeword values can be used independently from or in conjunction with difference coding to obtain higher compression rates. The least-significant bits of either a block of pixels or a vector of differences tend to be uniformly random; therefore, stripping these bits before sending data to the decoder and reinserting a mean value at the decoder decreases the number of bits sent when a new codeword is generated for the codebook, with a small increase in distortion. The amount of additional error incurred is not readily noticeable in most images.

2. Nonlinear Quantization. When difference coding is used with bit stripping, only a few values typically occur, and these may be represented by a relatively small number of quantization levels. However, there may be sharp edges in the image which will have large differences in value between adjacent blocks. This can cause large errors at edges if linearly quantized difference values are too small to keep up with the rapid change in pixel values. To minimize this error, nonlinear quantization can be used. The choice of quantization step sizes is not clearly defined, as the image statistics are unknown to the encoder and cannot be easily transmitted to the decoder. Therefore, design of the quantizer cannot be made adaptive, and an arbitrary choice must be made beforehand. Quantizer design has some criteria, however. Initial step sizes (near zero) should be small, and subsequent sizes should increase. Based on this assumption, a fixed logarithmic quantizer is used.

C. Interpolation and Smoothing

As mentioned earlier, since LAVQ uses discrete vectors to represent a set of vectors within a threshold set by the error allowance, there is noticeable blockiness in the output. Difference coding helps remove this effect, but it is insufficient: Since zeroth order estimation of the next block is used—the mean is assumed to be identical across adjacent blocks—regions with gradual pixel value changes can cause a sawtooth or staircase visual artifact at high compression rates. In addition, difference coding does not remove the block boundaries visible in the vertical direction. In cases where difference coding is used and where it is not, horizontal interpolation can remove the horizontal artifacts, and vertical smoothing can remove the vertical artifacts; if done carefully, both techniques do not excessively destroy detail or cause a blurry appearance.

Horizontal interpolation essentially interpolates across those blocks represented by different codewords. Each block is classified as either a repeat of the previous block or not; the first occurrence of a block that is not a repeat of the previous block is recorded; the rest are initially left blank. These blank blocks are filled with a pixelwise linear interpolation of the edges of the two closest nonblank blocks. However, this can smear edges which occur in the image; therefore, a threshold is used: If the difference between two differing blocks is larger than this threshold, no interpolation is done. This threshold must be adjusted externally.

Vertical smoothing averages the pixels on the vertical block borders. If both blocks on the border are not new codewords, that is, both already exist in the codebook, then the two border pixels are averaged; this average value is substituted for the border pixels. If both blocks are new codewords, nothing is done. If only one is new and the other is an existing codeword, then the new block is unaltered; the existing block's boundary pixel is substituted with the average of the boundary pixel of the new block and the two boundary pixels (the boundary and the pixel vertically adjacent to it) of the existing block. In this way, new codewords, which usually describe detailed areas, are unaltered, while existing codewords, which describe areas of low detail, are smoothed.

V. Lossless Data Compression

As mentioned earlier, the LAVQ algorithm is also suitable for lossless compression of database records and other data dominantly characterized by pattern repetitions. Examples of these data sources include textual data, accounting and payroll database data, telemetry data, and engineering data. Lossless compression using LAVQ can be achieved easily by setting the error allowance to zero. In the lossless compression mode, the basic LAVQ encoder becomes a locally adaptive move-to-front (MTF) algorithm.

Lossless LAVQ works best on data with fixed record sizes. Data represented in fixed-size packets and with patterns confined to the packets or fixed subfields within them are the best candidates for LAVQ. However, arbitrary fixed-length blocking of data can also be used on other sources without defined block sizes without significant detriment. Universal data compressors such as Lempel-Ziv (LZ) -based algorithms assume no structure of the source except for intersymbol correlation; however, many of these algorithms require large amounts of memory during coding, use complex tree data structures such as the "Patricia tree" to maintain their dictionaries, and have sophisticated pruning techniques to update the code tree. LAVQ requires relatively less memory and less complex data structures; it uses a simple codebook updating algorithm (MTF in this case) with a much smaller data buffer. Several high-performance LZ-based algorithms use back-end entropy coders to further improve performance; LAVQ can be likewise equipped. In short, lossless LAVQ can achieve rates comparable to the best LZ-based algorithms.

VI. Software Implementation

Software implementation of LAVQ, complete with all improvements, has been completed. To minimize source code complexity, the arithmetic coder has been implemented separately from the encoder. Parameters variable in the encoder include block dimensions, codebook size, error tolerance, number of bits stripped, first or best occurrence of an acceptable codeword, and difference coding. Furthermore, to allow use of the encoder for image sequences, as in video, codebook preservation is also provided. This allows the codebook from the previous frame to be used in the subsequent one.

A. LAVQ Encoder and Decoder

The encoder program first reads in all the data, then converts the image into a linked list of blocks. The codebook is specified as a doubly linked list to facilitate speed in rearrangement: Only ten pointers at most need be changed to do a complete codebook rearrangement. The codebook is initially assumed empty. At each step, a block is compared with the codebook entries and the best or first occurrence of an acceptable codeword is found. The codebook is updated as needed, and the requisite values are output. The program can also calculate global entropy of the encoded file to give an estimate of the highest compression possible with these parameters. The decoder program reverses this arrangement, with each input being an index or a new codebook value, and the codebook is rearranged in a manner identical to the encoder codebook. The image is rebuilt as a linked list and is then converted to image format and output.

B. Adaptive Arithmetic Coding

The arithmetic coder implemented is one described in [5]; it is used here with only minor modifications; the most notable of these is the ability to input symbols of alphabet size less than 256. The assumption is made at first that the symbols are all equiprobable; after each symbol is encoded, its statistics are modified to reflect this. Two encoders and decoders are used: One pair encodes and decodes only the indices from the output of the LAVQ encoder and ignores the new codeword information; the second pair codes and decodes these new codeword values.

The basic arithmetic code operates in the following manner: The symbols are arranged in some order and are assigned regions of size corresponding to their probabilities. These regions span the space from 0 to 1. The coding region is initially defined to be [0,1). When a symbol is coded, the symbol space [0,1) is scaled to fit the coding region, and the new coding region is defined by the region specified by the symbol coded. Therefore, at each step, the coding region becomes more and more narrow. In practice, the coding region is scaled in size as each unambiguous most significant bit is transmitted. The decoder reverses this by scaling up the coding region as symbols are decoded. A detailed discussion of arithmetic codes is available in [5].

VII. Experimental Results

A. Image Compression

The LAVQ algorithm was tested on a number of monochrome 8-bit 512 × 512 pixel images. Global pixel entropies, which are entropies estimated over the whole images, are listed in Table 1. These images were selected to provide a diverse cross section of images to examine the flexibility of LAVQ. A portrait ("lena"), a wildlife/natural scene of a seal on a rocky seashore ("seal"), a high detail overhead view of Los Angeles International Airport ("lax"), the cratered surface of Mercury ("mercury"), the rings of Saturn ("saturn"), and a medical CAT scan image ("cat01") were used. The images are shown in Fig. 2.

Performance parameters are measured in mean squared error (MSE) for distortion and required bits per pixel for rate. MSE is defined as

$$\mathrm{MSE} = \frac{1}{512^2} \sum_{i=1}^{512^2} \left[p_{\text{original},i} - p_{\text{processed},i}\right]^2$$

for images of size 512 × 512 pixels.

The LAVQ parameters used were 8 × 1 blocks with 255 codewords in the codebook. Difference coding was used, and 4 bits with logarithmic quantization levels were used to represent each new codeword value. There is not much difference in performance between finding the best match in the codebook (complete codeword search) and stopping after the first instance of an acceptable (within error allowance) codeword in the book (partial codebook search, typically one-fifth to one-tenth of the entire book); therefore, the latter strategy is used to maximize speed and to favorably skew the index statistics. Codebook and block sizes were selected to obtain good results; block sizes larger or smaller than 8 × 1 yielded worse results, and increasing codebook size beyond 255 had only marginal rate improvement with substantial increase in computational complexity. Block interpolation and smoothing are used with a threshold of 32. The LBG algorithm using blocks of size 4 × 4 pixels (found to yield good results in general and better results than LBG with 8 × 1, 4 × 2, or 16 × 1 pixel blocks) is presented for comparison.

In the rate-distortion curves generated for all six images (see Figs. 3 through 8), the LAVQ blocks are 8 × 1 pixels;

255 codewords are in the codebook, the difference coding has 4 bits per codeword value, and there are logarithmic quantization, block interpolation, and smoothing, except for Fig. 5, which lacks block interpolation and smoothing. LBG was done with 4 × 4 pixel blocks. LAVQ sends the codebook simultaneously with the indices while compressing the image; therefore, a true comparison should include the cost of sending the codebook for LBG as well. The LBG curves were generated using 4 × 4 pixel blocks and varying codebook sizes (from 16 to 8192). LAVQ curves were generated using the parameters outlined above. The curves have differing scales because each image has different characteristics which alter the algorithms' ability to compress them.

In most cases, LAVQ does better (defined as having a lower distortion for a given rate or vice versa) than LBG in the low-distortion regions (high-rate regions); in this region, LAVQ sends more new codewords, and these new codewords are more accurate renditions of the original image than the codewords used by LBG, which are the centroids of many blocks. Because LAVQ requires that more codewords be sent than LBG for low-distortion cases, this factor by itself would seem to make the rate for LAVQ worse than for LBG. However, LAVQ can take good advantage of lossless entropy coding of indices and new codewords. In contrast, because LBG's codebook search algorithm distributes its codewords to span the space in which the image's vectors exist, in general LBG does not profit as much as LAVQ does from lossless compression. Only about a 5-percent improvement using arithmetic coding was noted, compared to over 50-percent improvement obtained for LAVQ using arithmetic coding of both indices and codewords. As a result, LAVQ often has the potential for better performance than LBG after entropy coding is applied to LAVQ.

At lower rates (and higher distortions), LBG takes time to select a good codebook and therefore can do much better than LAVQ, which does not train a codebook at all. This is the dominant factor which can make LAVQ performance worse than LBG: Since LAVQ does not train a codebook, it relies on recently occurring vectors for a source of new codewords. These do not necessarily reflect a good choice of codewords, so for a given rate, the distortion for LAVQ can be higher than for LBG.

Such generalizations, however, have many exceptions. For example, two images in the set used here, "cat01" and "saturn," have large regions of uniformity compared to the other images. In the high-rate (low-distortion) region, these low-detail regions are better coded by LBG than by LAVQ because the LBG codewords can code regions of low detail with less distortion; this effect is enough in these two images to defeat the advantage that LAVQ gains by lossless compression of the indices and codewords.

In the low-rate (high-distortion) region, "cat01" and "saturn" are better coded with LAVQ than with LBG. With other images, the time spent by LBG in developing a good codebook made the performance of LBG better than what LAVQ could achieve without training. With these two images, however, LAVQ can code the low-detail regions quite easily with very few new codewords while the details are better preserved in the high-detail regions. This can be enough to offset the disadvantages mentioned earlier that LAVQ encounters in the low-rate (high-distortion) region.

Each LBG rate-distortion curve of 10 data points took approximately 100 hours of computation on a Sun Sparc 2, while one LAVQ curve with 24 data points, complete with lossless compression, was done in about 1 hour: LAVQ is much faster than LBG and can achieve performance comparable to LBG if the cost of the codebook is included. Furthermore, LAVQ preserves detail better than LBG, but does so with increased blockiness in low-detail regions. This detail-preserving feature of LAVQ is discussed in the next section.

B. Detail Preservation

As mentioned in the previous sections, LAVQ operates by matching a vector to the codewords in the codebook using a predetermined fidelity criterion, and new codewords are entered verbatim from the examined vector if no match occurs within this error allowance. Therefore, those vectors which are significantly different from previous vectors are coded without distortion; these occur at edges or areas of high detail. The cost of preserving these details, however, is increased distortion in low-detail ("smooth") areas. Codewords are not optimized to best represent these regions, so they exhibit more blockiness. Thus, LAVQ preserves details and is potentially very attractive to military intelligence and space applications: These applications require close examination of details to identify and differentiate among various objects.

These effects are illustrated in Figs. 9 and 10. In Fig. 9, the upper right edge of the hat brim of "lena" is shown. The LBG codebook size, to be comparable to LAVQ, is fixed at 256. The LAVQ error allowance was adjusted to yield a distortion comparable to LBG. LBG achieves 43.83 MSE at 0.50 bits/pixel (compression ratio of 16:1). LAVQ achieves 43.68 MSE at 0.56 bits/pixel (compression ratio of 14:1). The edge of the hat brim is rendered with much less blockiness with LAVQ; the drawback is that the low-detail areas exhibit noticeable blockiness.

In Fig. 10, the lower right edge of the upper terminal of "lax" is shown. The LBG codebook size is again fixed at 256. LBG achieves 166.8 MSE at 0.50 bits/pixel (compression ratio of 16:1). LAVQ achieves 166.0 MSE at 0.77 bits/pixel (compression ratio of 10:1). Here, LBG does poorly in preserving the details of the aircraft, terminal, and service vehicles; details of the terminal and several service vehicles have disappeared. LAVQ does much better on these details, but exhibits more blockiness than LBG when representing the tarmac, which has less detail. With LAVQ, it is possible to identify aircraft type, while it is more difficult with LBG. For the two aircraft in Fig. 10, the fuselage and wing shapes, engine locations, and other details are preserved more clearly by LAVQ than by LBG.

C. Lossless Data Compression

Demonstration of lossless compression of structured data was conducted on Magellan space-probe engineering data. The Magellan spacecraft engineering data file "mdata" consists of 1692 records of 100 bytes each; each record consists of 15 fields of various sizes ranging from 4 bytes to 16 bytes. Many good existing LZ variants can achieve compression ratios for "mdata" of 2.2:1 to 5:1 of the original file size, depending on the amount of memory consumed. The UNIX program "compress" achieves a compression of 3:1 at a cost of 544 kbytes of memory used.

The Magellan engineering data have both natural parsing (a frame marker at the beginning of each record) and artificial parsing (the record size is fixed, 100 bytes) properties. Most real-world data sources like texts, images, and engineering data records possess either or both of these two parsing properties. To apply LAVQ to these data, the file was first blocked in sequential order into 5-byte blocks. LAVQ was then applied to each block position; thus, all of the first 5-byte blocks (bytes 1 to 5 in the 100-byte record) were coded using one encoder, all of the second 5-byte blocks (bytes 6 to 10 in the 100-byte record) were coded using a second encoder, and so on.

The resulting LAVQ output, both indices and new codeword values, was then coded with the adaptive arithmetic coder used earlier. Results using a Q-coder [6,7] and the theoretical limit reached by nonadaptive means (computed from the global entropy) are also included for comparison. The Q-coder is essentially an adaptive arithmetic coder; it differs from the arithmetic coder used above: The arithmetic coder maintains a running total of frequency of use of each symbol and uses this to determine probabilities for arithmetic coding. The Q-coder uses a fixed probability table accessed by an adaptive finite state machine. This finite state machine adapts with the previous symbols; its state is determined by the bits in each byte.

Two differing approaches are taken. First, the codebook size is fixed at 256, and only one coder is available. Thus, a large buffer is needed to store the data to allow sequential coding of each of the 20 blocks in each record. Results tabulated in Table 2 show that the best performance is obtained from using large buffers: Larger files have more opportunities for repetitions and patterns of codeword usage. In Table 2, compression ratios of Magellan engineering data are of 169,200-byte size. Tabulated are compression ratios achieved by the adaptive arithmetic coder used for LAVQ, the Q-coder, and the theoretical maximum using a global entropy coder. This last value is derived from the global per-symbol entropy. Note that larger buffer sizes provide greater compression. Arithmetic coder compression approaches the theoretical value (global entropy) closely. Even better results are sometimes obtained. The Q-coder and arithmetic coder sometimes appear to have results better than entropy; this occurs because these adaptive coders code on the basis of local entropy (entropy over a localized window of characters) as opposed to the global entropy listed here. The Q-coder does better than arithmetic coding, which does about as well as theoretically possible with a global entropy coder.

In the second approach, the codebook size is allowed to vary, and all 20 coders are available for parallel coding. No additional buffer is needed in this case, as the 20 blocks are coded simultaneously. Again, the indices and new codeword values are coded with the adaptive arithmetic coder and compared with the Q-coder and the global entropies; results are listed in Table 3. In this table, as in Table 2, compression ratios of Magellan engineering data are of 169,200-byte size. No data buffer is allowed here, but 20 coders are operating in parallel. In this case, the arithmetic coder does slightly better than the Q-coder. Again, the Q-coder and arithmetic coder sometimes appear to have results better than entropy for the same reason given for Table 2. Best performance is obtained with larger codebooks; large codebooks can record codewords farther into the past and therefore have more opportunities for codeword matching. Here, the arithmetic coder does better than entropy and the Q-coder. In both cases, LAVQ performance is comparable to LZ-based algorithms. The advantage of LAVQ is that far less memory is required than for LZ algorithms; this is of importance in systems with size, weight, or complexity constraints, as is the case with deep-space probes.

VIII. Conclusion

The LAVQ algorithm provides a fast, one-pass data compression algorithm. Improvements to the basic algorithm maintain this one-pass high-speed property while increasing performance measurably. Experimental results in image compression yield performance not significantly inferior to LBG, but at a fraction of the complexity. Distortion in images occurs as blockiness in low-detail areas, while high-activity areas maintain sharp details. LAVQ also performs well in lossless compression, again with low complexity as compared with other algorithms.

References

[1] Y. Linde, A. Buzo, and R. M. Gray, "An Algorithm for Vector Quantization," *IEEE Transactions on Communications*, vol. C-28, pp. 84–89, 1980.

[2] J. L. Bentley, D. D. Sleator, R. E. Tarjan, and V. K. Wei, "A Locally Adaptive Data Compression Scheme," *Communications of the ACM*, vol. 29, pp. 320–330, 1986.

[3] K. M. Cheung and V. K. Wei, "A Locally Adaptive Source Coding Scheme," *Proceedings of the Bilkent Conference on New Trends in Communications, Control, and Signal Processing*, Ankara, Turkey, pp. 1473–1482, July 2–5, 1990.

[4] K. M. Cheung and V. K. Wei, "A Locally Adaptive Source Coding Scheme," submitted to *IEEE Transactions on Communications*, 1990.

[5] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic Coding for Data Compression," *Communications of the ACM*, vol. 30, pp. 520–540, 1987.

[6] Joint Photographic Experts Group, *JPEG Draft Technical Specification*, Rev. 5, Washington, DC: International Organization for Standardization/International Telegraph and Telephone Consultative Committee (ISO/CCITT), Section 12, January 15, 1990.

[7] R. B. Arps, T. K. Truong, D. J. Lu, R. C. Pasco, and T. D. Friedman, "A Multi-Purpose VLSI Chip for Adaptive Data Compression of Bilevel Images," *IBM Journal of Research and Development*, vol. 32, pp. 775–795, 1988.

Table 1. Pixel entropies. Global pixel entropies of images.

Images	Entropy, bits/pixel
"cat01"	5.503
"lax"	6.827
"lena"	7.445
"mercury"	6.416
"saturn"	6.885
"seal"	7.356

Table 2. Magellan data compression: sequential compression.

Buffer size, bytes	Codebook size	Compression ratio achieved by		
		Arithmetic coder	Q-coder	Global entropy
169200	256	3.957	5.053	3.887
84600	256	3.752	4.764	3.748
56400	256	3.635	4.556	3.640
42300	256	3.429	4.210	3.440
28200	256	3.285	3.929	3.296
14100	256	2.913	3.320	2.921

Table 3. Magellan data compression: simultaneous compression.

Codebook size	Compression ratio achieved by		
	Arithmetic coder	Q-coder	Global entropy
256	3.957	3.912	3.887
128	3.873	3.815	3.798
64	3.411	3.290	3.314
32	2.973	2.814	2.949
16	2.747	2.545	2.6076

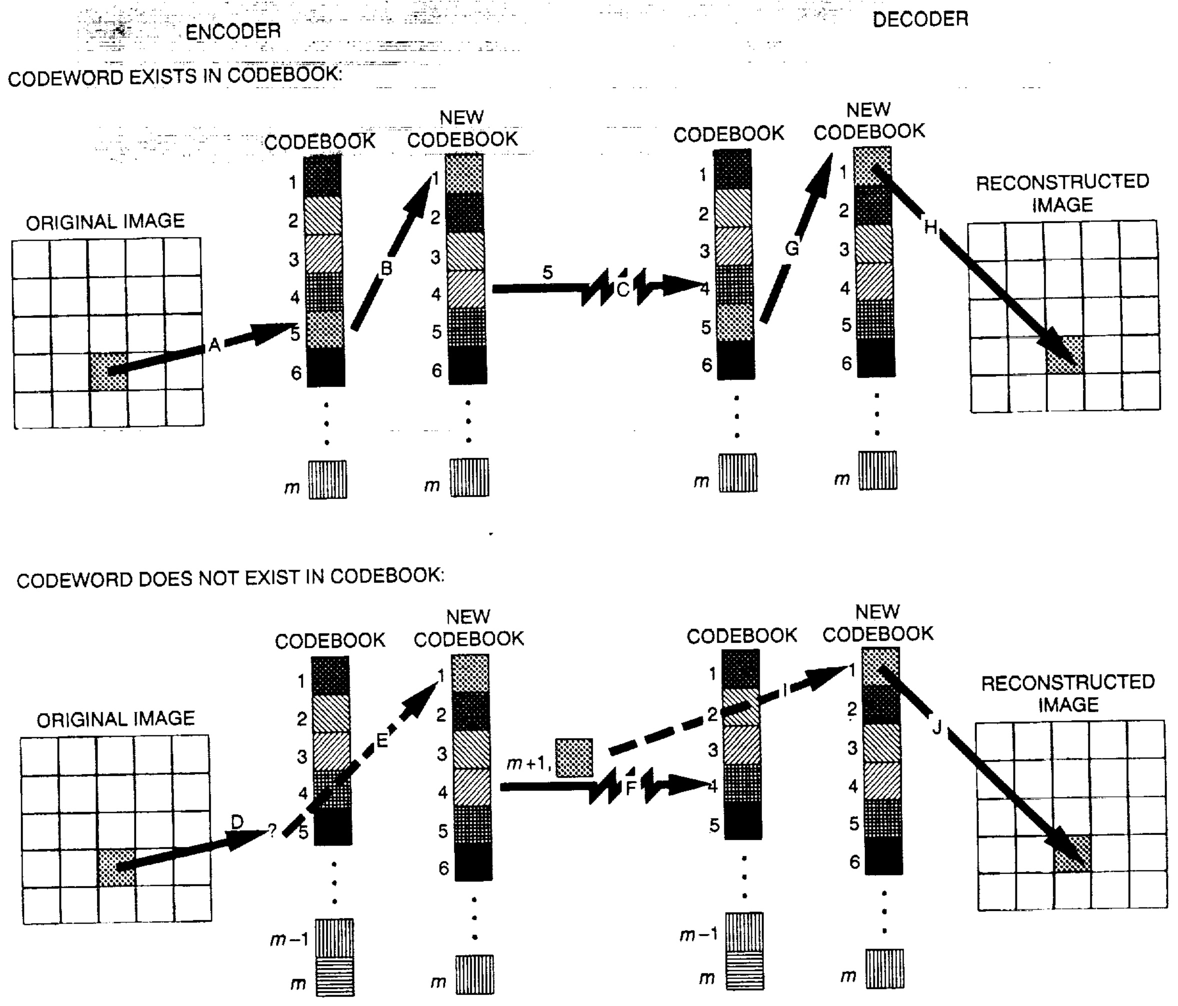

Fig. 1. Encoder and decoder: An image block is compared to the codebook (A); if a codeword close enough exists, that codeword is moved to the top of the codebook (B) and the index is transmitted (C). If it does not exist (D), then the block is inserted at the top of the codebook (E) and the index $m + 1$ and the block are transmitted (F). On the receiver side, if an index is received, the corresponding codeword (G) is moved to the top of the codebook and the block is inserted into the reconstructed image (H). If the special index $m + 1$ is received (I), a raw block is anticipated immediately following; this block is placed in the codebook and also in the reconstructed image (J).

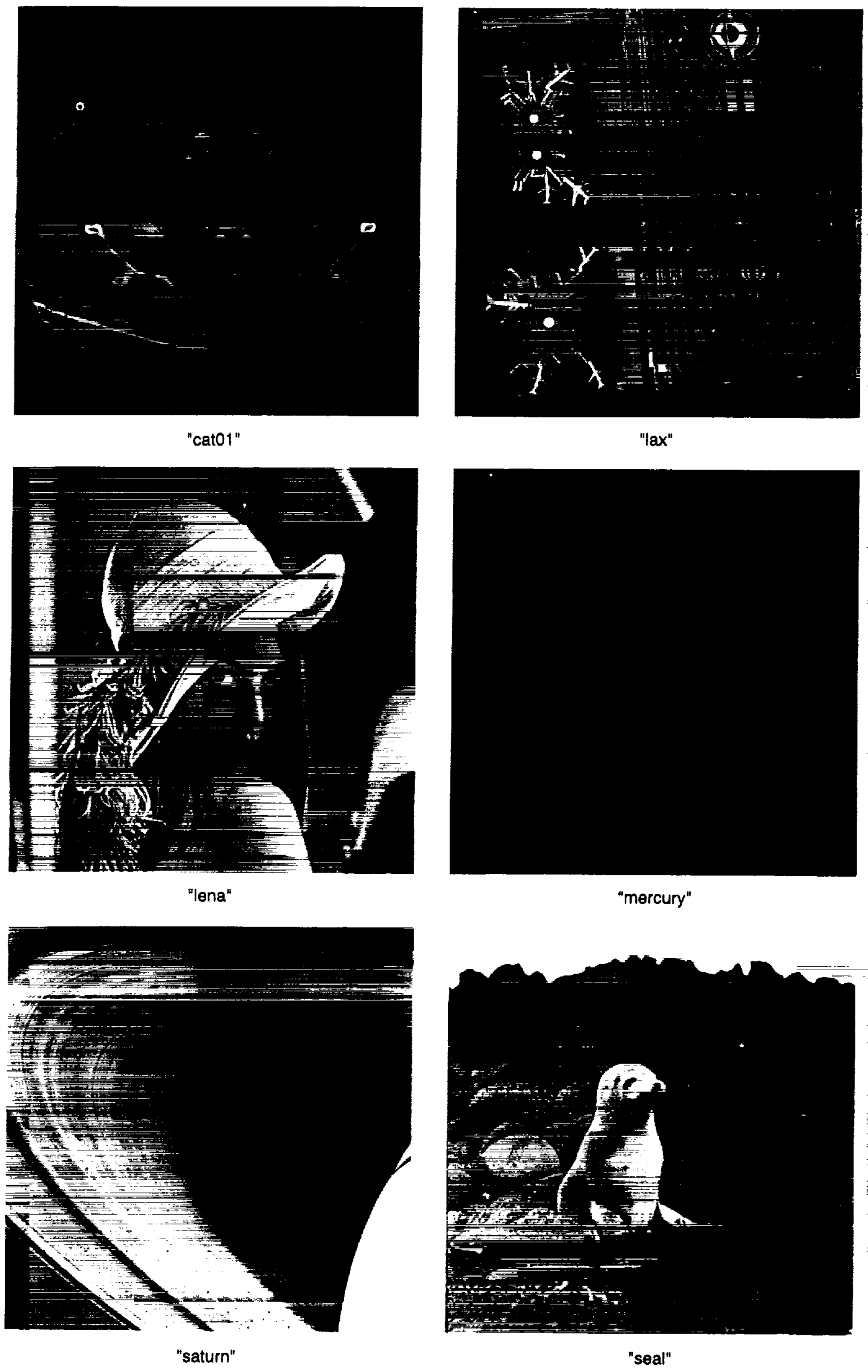

Fig. 2. Original Images. All images are 512 × 512 pixel, 8-bit monochrome.

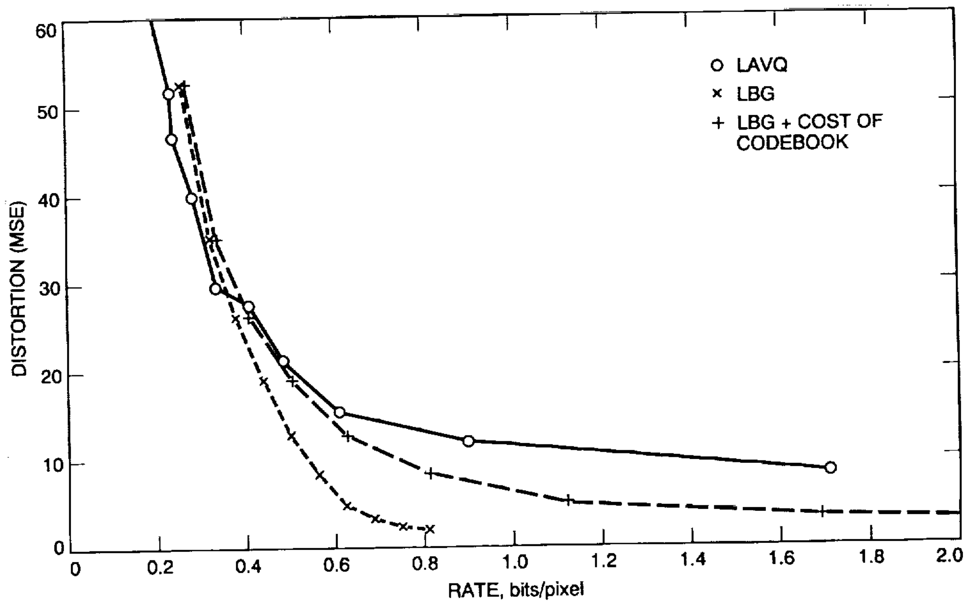

Fig. 3. Rate-distortion curve for "cat01."

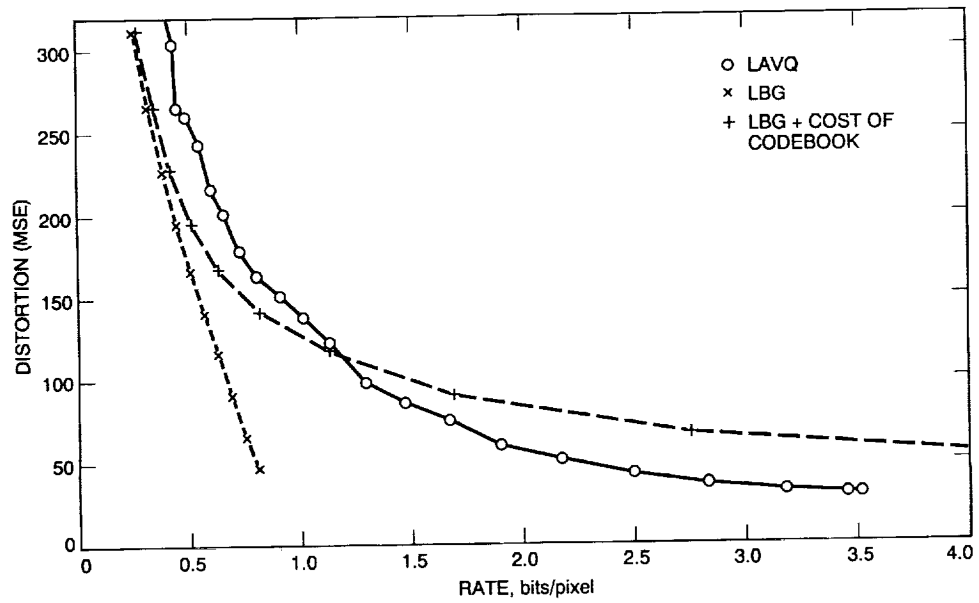

Fig. 4. Rate-distortion curve for "lax."

C-3

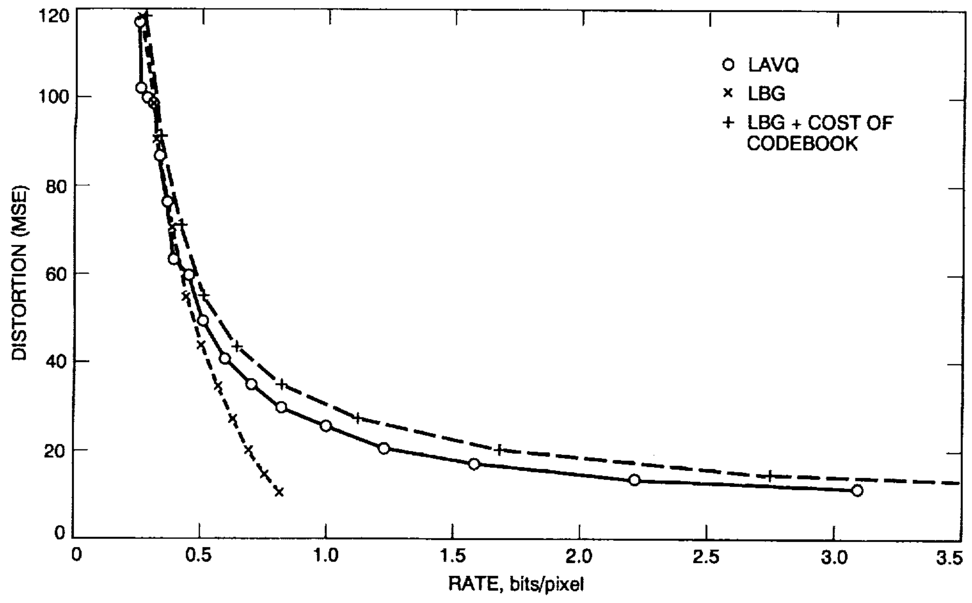

Fig. 5. Rate-distortion curve for "lena."

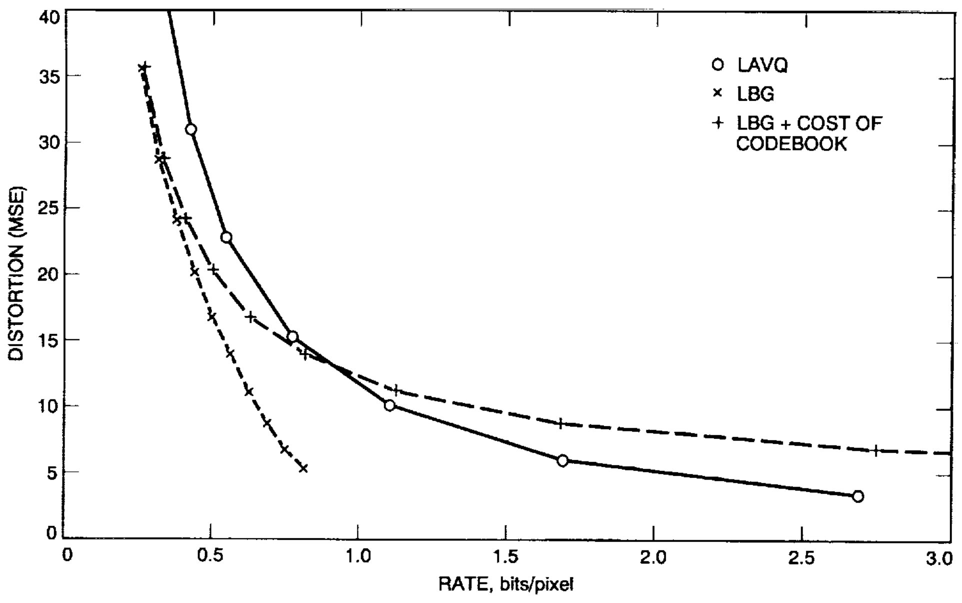

Fig. 6. Rate-distortion curve for "mercury."

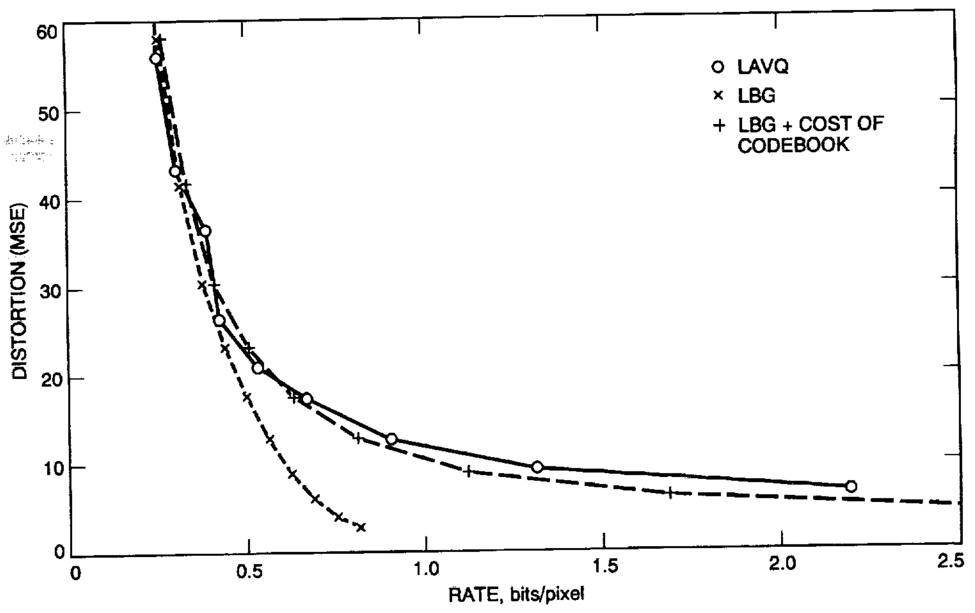

Fig. 7. Rate-distortion curve for "saturn."

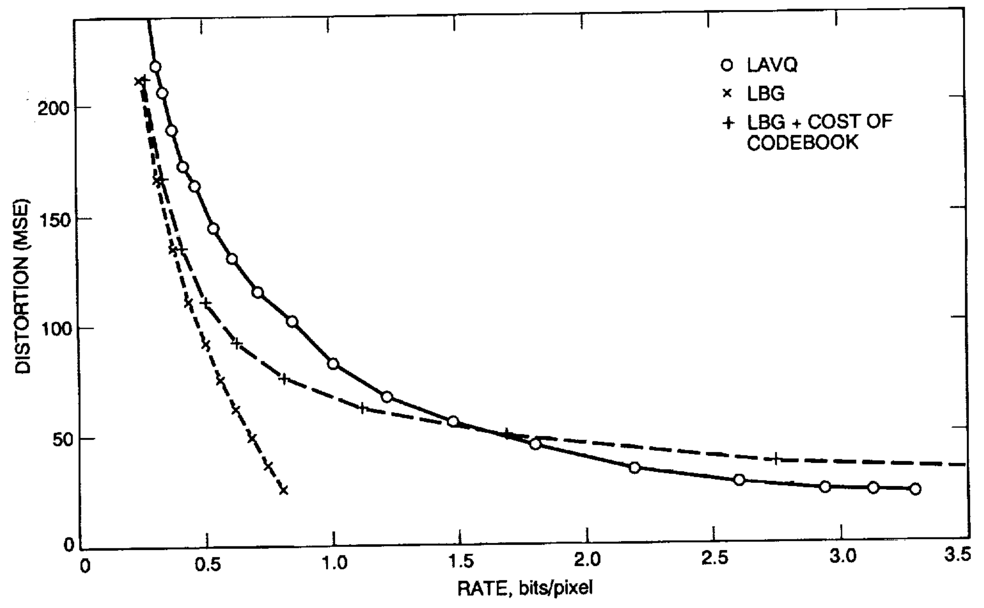

Fig. 8. Rate-distortion curve for "seal."

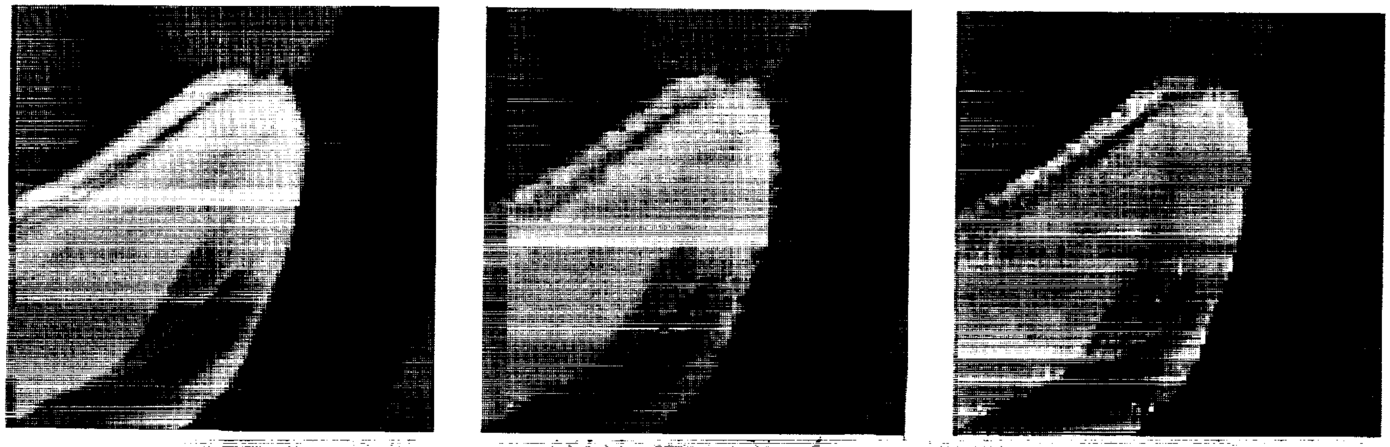

Fig. 9. Detail of "lena." Note that LBG has more blockiness at the edge, but represents low-detail ("smooth") areas without as much blockiness as LAVQ.

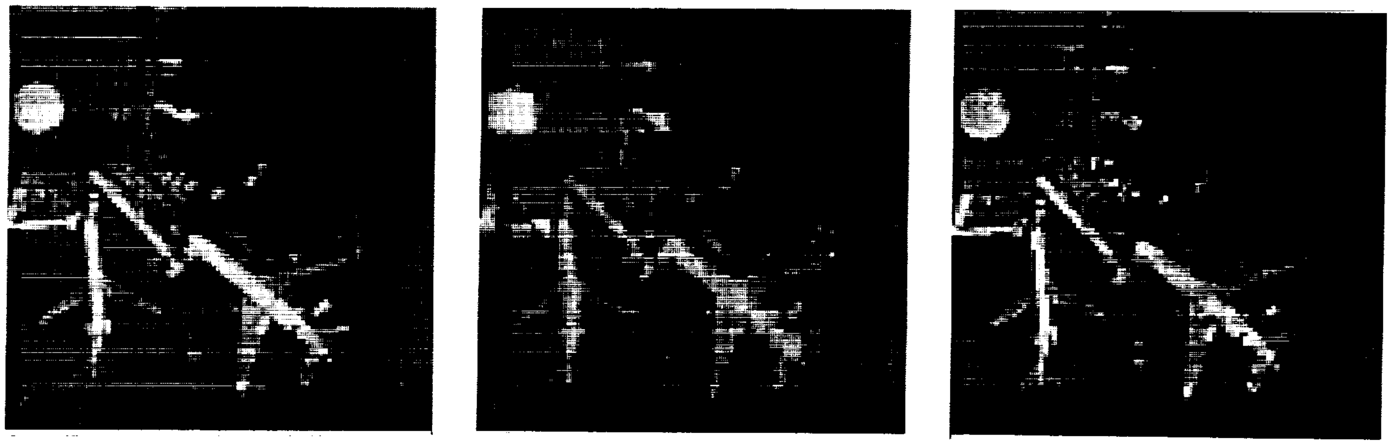

Fig. 10. Detail of "lax."