1985

On the vector quantization of images /

Larry Clifford
Lehigh University

ON THE VECTOR QUANTIZATION OF IMAGES

by

Larry Clifford

Presented to the Graduate Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Electrical Engineering

Lehigh University

1985

CERTIFICATE OF APPROVAL

This thesis is accepted and approved in partial fulfillment of the requirements for the degree of Master of Science.

July 31, 1985
(date)

Bruce D Hutchinson
Professor in Charge

Eric L. Thompson
Chairman of Department

Acknowledgment

I would like to thank Prof. B. Fritchman. my advisor. for his advice. trust. and aid throughout my work on this thesis.

Table of Contents

List of Tables

List of Figures

List of Flow Charts

ABSTRACT

Vector quantization is a vector generalization of ordinary PCM. and has wide applications in speech and image data reduction. Quantizing samples as blocks or vectors is inherently better than separate scalar quantization of the samples.

The two most important types of vector quantizer (VQ) are described in detail: the full-search vector quantizer (FSVQ) and the tree-searched vector quantizer (TSVQ). Both are designed using iterative algorithms based on Lloyd's method of designing an optimum one-dimensional quantizer. These two algorithms were implemented for varying block sizes, codebook sizes, and tree orders using FORTRAN programs. Numerical results for the MSE and l_1 norm distortion measures are presented and discussed. For very large images (relative to the size of the codebook), theory says that with increasing vector dimensionality, distortion performance should improve; however, for small images such as those used here, the effective bits per pixel decreases (with a corresponding increase in distortion) with increasing dimension. Higher-order trees are found to perform better distortionwise than binary trees, in addition to their property of requiring less memory and search time than binary trees.

ON THE VECTOR QUANTIZATION OF IMAGES

I. Introduction

Vector quantization is a vector generalization of standard pulse code modulation (PCM). In PCM, a sample value from a continuous one-dimensional source function is mapped onto one of a finite number of corresponding discrete values, and a codeword (usually binary) is associated with the discrete value for storage or transmission. A set of representative discrete values which are placed uniformly across the range of the source will not generally be the "best possible" set, in the sense that it may not minimize some error measure between the analog samples and the quantized values. Therefore, it is often desirable to calculate the "best" distribution of output levels in order to create an optimal PCM system for some particular source. This problem -- that of optimal one-dimensional PCM -- was discussed in an oft-cited 1957 paper by Lloyd. [18] He presented two algorithms, Lloyd's methods I and II, which detail how to construct an at least locally optimum quantizer given the source distribution. In 1965, Forgey [6] showed that Lloyd's Method I could be applied to a training sequence of data rather than a known source distribution.[1]

---

[1] Gray and Linde, p. 381.

In 1980, a paper was published by Linde, Buzo, and Gray [17] which described the extension of Lloyd's Methods I and II to the quantization of vectors (or blocks) of source samples. Theirs was not the first generalization of Lloyd's work (see, for example, [19]), but it relied only on a very general definition of distortion measures, was thorough, and became the basis of most subsequent work on vector quantization.

Early studies of the application of vector quantization were done almost exclusively on speech data. There are two good reasons for this. Firstly, speech waveforms can be classified into a small number of general types, such as voiced and unvoiced speech. The statistical properties of speech have been well-studied and are relatively simple. Secondly, the production mechanism of speech has also been well-studied. This approach has given rise to the well-developed field of linear predictive coding (LPC), which encodes a short sequence of speech samples into an ordered set of "LPC coefficients." This set lends itself directly to treatment as a vector in a vector quantization scheme.

On the other hand, there is no class of image which is as well behaved as speech, or which has been so intensively studied. Nor is there any standard way of classifying images, or any concept, for images, quite like that of "intelligibility" for speech. (When we go through the trouble of producing images, we usually require that they be much more than merely "recognizable.") There is no general "image production mechanism" corresponding to the acoustic tube model for speech. Images, in short, are generally less tractable than speech for the purpose of analytical description.

However, since they require such large amounts of data, it is very desirable to compress them. Also, most of the same techniques that are actually applied to

the vector quantization of speech (as opposed to the mathematical modelling of such quantization) can also be applied to images, and images can be neatly and naturally divided into frames for analysis purposes.

Actually, as will be discussed briefly in Part II, the transform coding method of image data compression is a special case of vector quantization. However, even the optimum transform coding system is a suboptimum vector quantizer. Later on, we shall look briefly at why this is so.

Studies of the performance of vector quantization on images have remained sparse since 1982, when Gersho and Ramamurthi [10] were apparently the first to produce an "implementation of vector quantization to image coding [sic] where each vector describes a spatial block of pixel intensities."[2] Their results were entirely qualitative and were restricted to the case of partitioning an image into square (p×p) blocks of pixels. For this thesis, several different versions of vector quantization were implemented on real data using FORTRAN programs, and the detailed behavior of these algorithms for several varying parameters was examined. A presentation of the mathematical theory of vector quantization and a discussion of the various algorithms and results follows.

II. Theoretical Background

A. The Context of Vector Quantization

Quantizing blocks of image data is not a new idea. Transform coding was first developed in the 1950's, whereby a block (vector) of data is first transformed into another vector, and the components of the new vector are quantized independently (i.e., using scalar quantizers). The transformation is usually a

---

[2] Gersho and Ramamurthi, p. 428.

linear operation, such as the Fourier transform, which produces a transformed vector of the same length as the original.[3] The purpose of the transformation step is, generally speaking, to decorrelate the components of the vector. Intuitively, this means that each component will contain no redundant information, so that at each scalar quantization step we are no longer re-quantizing some of the same information in nearby pixels. *The* optimum transform coding scheme, the Karhunen-Loeve transformation, is well-known [13]. This method is only optimum within the defined limitations of transform coding. Since transform coding is practical and useful despite its restrictions, and since until recently the large amounts of memory and number-crunching required for a *general* vector quantization scheme (referred to in this paper simply as "vector quantization") were unaffordable, little attention was paid over the years to the general approach. A further disincentive to research may have been the knowledge that so simple a technique as DPCM provides nearly optimal performance for asymptotically large rates.[4]

At any rate, it has long been known from Shannon that "even for memoryless sources, one can always do better by quantizing an entire vector rather than separately handling components."[5] So, even the most successful transformation technique is doomed to suboptimality by the scalar quantization of the transformed components. Recently it has become feasible, however, to design quantizers which take full advantage of the vector nature of block quantization.

---

[3] Gray and Linde, p. 382.

[4] Abut, Gray, and Rebolledo, p. 424.

[5] Tao, Abut, and Gray, p. 343.

Curiously, transform coding includes an effort to *eradicate* correlation between samples, although it is a special case of a technique which is implicitly exploiting that very correlation in an "essentially optimum manner."[6]

B. Distortion Measures

Any discussion of vector quantization must begin with a definition of a class of distortion measures, because otherwise there is no quantitative definition of what constitutes a "good" or "bad" job of image reproduction, or what is meant by a "distance" between two vectors.

Let us assume that we have a vector $x=(x_0, x_1, \ldots, x_{k-1})$ in the k-dimensional space R^k, so that $x \in R^k$. If we reproduce x by some vector $y \in R^k$, then the distortion caused by this substitution is measured by $d(x, y)$. Such a distortion measure is defined as a "difference distortion measure" if it possesses the property that[7]

$$d(x, y) = L(x-y),$$

where

$$L(x) = 0 \quad \text{if} \quad x=0,$$

and

$$L(u) \leq L(x) \quad \text{iff} \quad \|u\| \leq \|x\|.$$

Here, $\|x\|$ is any seminorm on R^k. A seminorm is defined as a function of x such that

---

[6]Gersho and Ramamurthi, p. 428.

[7]Yamada, Tazaki, and Gray, p. 7.

1. $\| x \| \geq 0$,

2. $\| ax \| = | a | \cdot \| x \|$ for $a \leq R^1$.

and

3. $\| x + y \| \leq \| x \| + \| y \|$.

In other words, a seminorm is a positive linear vector function satisfying the triangle inequality. Commonly, $L(x-y)$ is simply the seminorm itself, and the seminorm used is the special case of a power of the Holder norm l_v :

$$L_v(x-y) \equiv \| x-y \|_v = \left[\sum_{i=1}^{k} | x_i - y_i |^v \right]^{1/v}$$

For example, the ubiquitous squared error measure is the case $(\| x-y \|_2)^2$. Weighted squares distortion measures of the form

$$d(x, y) = (x-y)^T B (x-y)$$

are often discussed,[8] where B is a $k \times k$ positive definite matrix. Because of its familiarity and mathematical tractibility, authors usually restrict their attention to the mean squared error (MSE), the case where B is just $(1/k)$ times the identity matrix:

$$d(x, y) = \frac{1}{k} (\| x-y \|_2)^2 .$$

No measure is computationally simpler, of course, than the l_1 norm, which is simply $| x-y |$. Calculating this measure of distortion between two vectors involves k additions, and one multiplication must be performed if one wishes to take the average distortion over a number of vectors. The supremum or l_∞ norm, which simply involves selecting the largest component of $(x-y)$, still requires k

---

[8] [5], [12], [14], [19].

magnitude comparisons per vector pair, which is certainly no better than the k additions of the l_1 norm. However, I know of no published subjective or objective tests of the performance of the l_1 norm in a vector quantization scheme, although in digital signal processing it is sometimes considered advantageous to replace multiplications by additions.

Not all distortion measures used in vector quantization are difference distortion measures. The Itakura-Saito measure[9] has the property of being implicitly minimized by the LPC speech method, and is commonly used in speech vector quantizing schemes, but is not a difference distortion measure. It is therefore more complicated and awkward to deal with.

C. Statement of the General Vector Quantizer

Assuming that we have chosen a distortion measure (keeping in mind that there is nothing special about the l_2 norm except its obvious geometric interpretation for $k \leq 3$), we are ready to define a vector quantizer.

A quantizer of k-dimensional vectors maps all of the points in R^k onto some finite subset of the points in R^k. Two things must be specified in order to define this mapping uniquely: (a) the finite subset of R^k onto which everything is mapped, which we shall denote as $Y = \{ \boldsymbol{y}_1, \boldsymbol{y}_2 \ldots, \boldsymbol{y}_N \}$ and refer to as the "codebook," and (b) some rule for dividing up R^k so as to assign each of its points to one of the codebook vectors. This assignment rule is called the "partition."[10]

If $Q(\boldsymbol{x})$ describes the action of the quantizer, then the partition may be

---

[9] Juang, Wong, and Gray.

[10] This development is based on that of Gersho,[8].

described formally as a division of R^k into subsets $R_1, \ldots, R_N$, such that

$$R_1 \cup R_2 \ \ldots \ \cup R_N = R^k$$

and

$$R_1 \cap R_2 \ \ldots \ \cap R_N = \emptyset.$$

Furthermore, if $Q(x)$ describes the quantizer function, then

$$Q(x) = y_i \quad \text{iff} \quad x \in R_i$$

and

$$R_i = Q^{-1} = \{x \in R^k : Q(x) = y_i\}.$$

These complementary descriptions may seem redundant, but they are not. Our ability to define the partition in terms of the codebook and vice versa is essential to our ability to design good codebooks.

A partition of a subset of R^2, with its codebook, is shown in 1. This is an example of a 5-point, 2-dimensional vector quantizer.

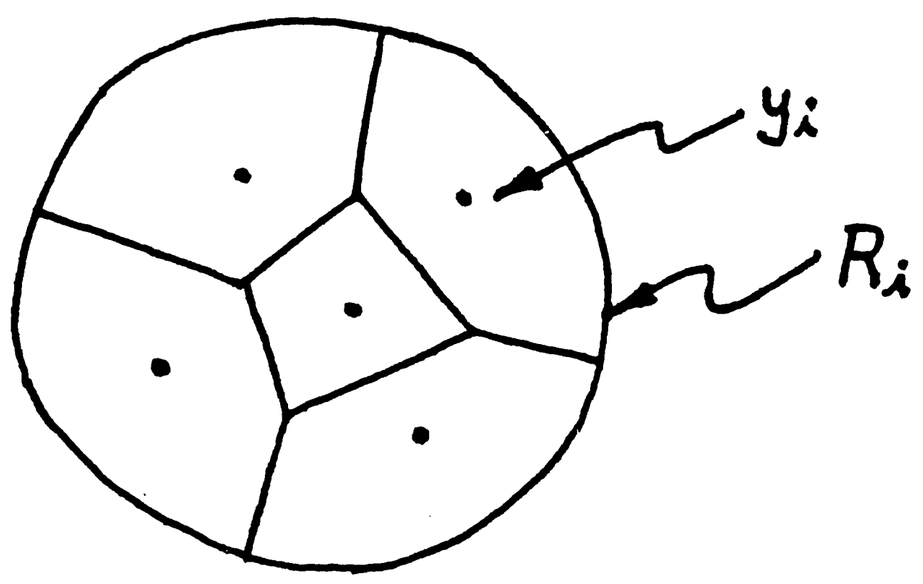


Figure 1: A Partition of a Region of R^2 with Codebook

How do we decide on a good partition and codebook? Any choice of codebook and partition, however idiotic, would meet our requirements so far. We must define what it is that we mean by a good quantizer. Assuming that x is a random vector in R^k, with either a continuous or discrete distribution, then it makes sense to define our *optimal* quantizer as that $Q(x)$ such that for all possible

quantizers $Q^*(x)$,

$$E[\,d(x, Q(x))\,] \leq E[d(x, Q^*(x))\,],$$

where E denotes expectation. "$d(x, Q(x))$" is simply the distortion resulting from quantizing x using the optimum quantizer. In other words, the average quantization error is what we would like to minimize.[11]

There are two complementary requirements which, it can be shown, an optimal vector quantizer must meet.

Property 1: The members of Y (the codebook) must satisfy

$$y_i = E(x \mid x \in R_i).$$

That is to say, the codebook vector which corresponds to one of the N partitions must be the mean of the vectors which are in that partition. If we are dealing with a finite set of vectors, this will correspond to the dimension-by-dimension sum of all the vectors which are deemed to lie in R_i (however that might be decided), divided by their number. Or, if we are dealing with a continuum of random vectors, y_i will be the "center" of some geometric region in R^k. It is not necessary, by the way, to worry about whether R_i is "really" a solid in k-dimensional space, bounded by $(k\text{-}1)$-dimensional hyperplanes (which it is), or a finite number of data vectors which have been grouped together as a set (which it also is, on occasion). For the calculation of distortion bounds, the geometrical

---

[11]This is not always the only way to define optimality. In 1977, Kasam [16] showed that for the case of one-dimensional quantization using a difference distortion measure, it was possible to optimize both $E[\,L(x - Q(x))\,]$, as discussed in this paper, and a "distance" measure Δ_g between $F(x)$, the continuous distribution of the source function, and $F_q(x)$, the distribution of the quantizer output: $\Delta_g = \int_{-\infty}^{+\infty} g[F(x)-F_q(x)]dx$. These two standards of optimality produce sets of dual equations, which are optimized when $g(\cdot) = L(\cdot)$ = the mean absolute error function, or the mean l_1 norm. If this result can be extended to arbitrary dimensions, it may be a fruitful topic to pursue in connection with vector quantization. Such work is beyond the scope of this paper.

and probabilistic version of vector quantizer structure is indispensable, but to implement an actual quantizer, it is hardly needed at all. This will be obvious as we proceed.

The second condition on the optimum quantizer is just as intuitively satisfying as the first:

Property 2 : The vectors which comprise R_i, whether they be a finite set of data points or an infinity of points filling a geometric solid, are given by

$$\{ \boldsymbol{x} \in R_i : d(\boldsymbol{x}, \boldsymbol{y}_i) \leq d(\boldsymbol{x}, \boldsymbol{y}_j) \text{ for all } i \neq j \}.$$

In other words, the vectors in R_i are simply those that are *closer* to the ith codebook vector than they are to any other. "Closeness" may not have an obvious interpretation in this context, depending on the choice of distortion measure, but even if the familiar MSE criterion is used it has no everyday geometric meaning for, say, $k = 16$. At any rate, this "nearest neighbor" partition rule makes some sense. The resulting regions are known as Dirichlet cells, and the partition in Fig. 1 is a Dirichlet partition.

If we have an optimal quantizer, then each codebook vector will be at the "center" of its partition and each partition will contain no points which are "closer" to any other partition's codebook vector than to its own. The actual encoding of an image is easy; one simply determines which partition an input vector is in (i.e., which codebook vector it is closest to) and stores a number to record that fact. In this way, an image containing M vectors will be represented by M integer numbers no larger in magnitude than N, the number of codebook vectors. However, there is no sure way of satisfying the nearest-neighbor rule without actually calculating how far (in terms of $d(\boldsymbol{x}, \boldsymbol{y}_i)$) the input vector is from all of the codebook vectors. One must *fully search* the codebook -- hence, the

optimum system is a full-search vector quantizer (FSVQ).

A useful tool for checking the performance of a vector quantizer is the scatter plot. This is produced by taking the pixels of an image two at a time. and treating the brightness of each as a coordinate on an x-y graph. If $k = 2$. then the codebook can also be plotted in this form. For any k. of course. the quantized image can be scatter-plotted just as the original was.

Fig. 2 is the scatter plot of an actual digital image containing 16.384 pixels with gray scale values ranging up to 256.

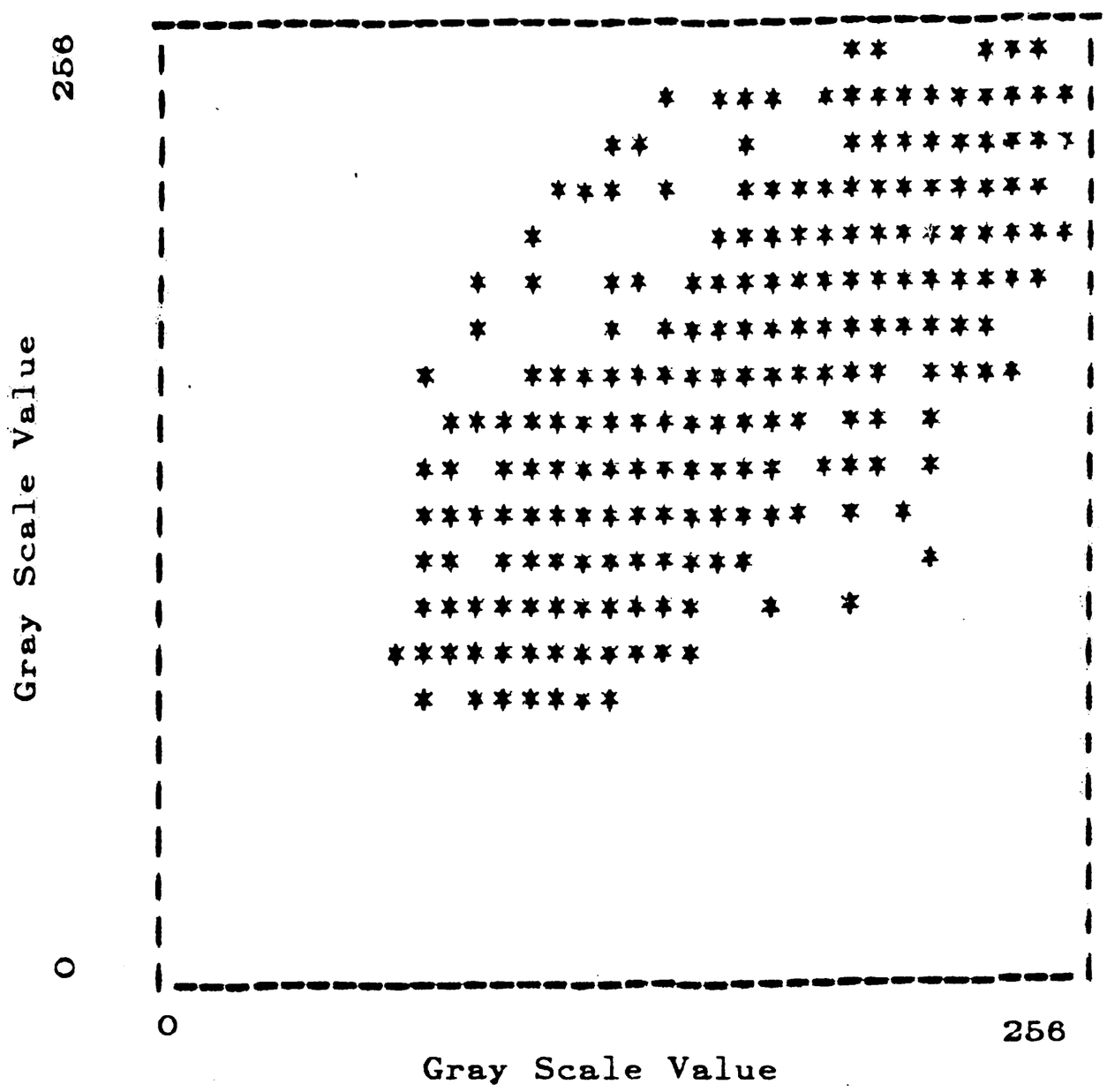


Figure 2: Scatter Plot of an Actual Image.

Fig. 3 is the scatter plot of the quantized version of the same image. A 2-dimensional FSVQ with 64 codebook vectors ($k = 2$. $N = 64$) was used. Since $k = 2$. Fig. 3 also happens to be a plot of the codebook optimized for the image of Fig. 2. It can be seen that the quantized image. as one would expect. contains a

sparser selection of vectors than the original. However. it would almost certainly not be possible to choose 64 pixel pairs which would do a better job of approximating the original image. Also. the diagonal tendency of the plots will be noticed. When two adjacent pixels have the same brightness. their vector representation will lie right on the diagonal: therefore. the farther from the diagonal. the more contrast between adjacent pixels there is in a vector and the more "edgelike" a feature it represents in the original image. The large variety of "shade" blocks and the poorer scattering of "edge" blocks are just what we would expect from a typical image.

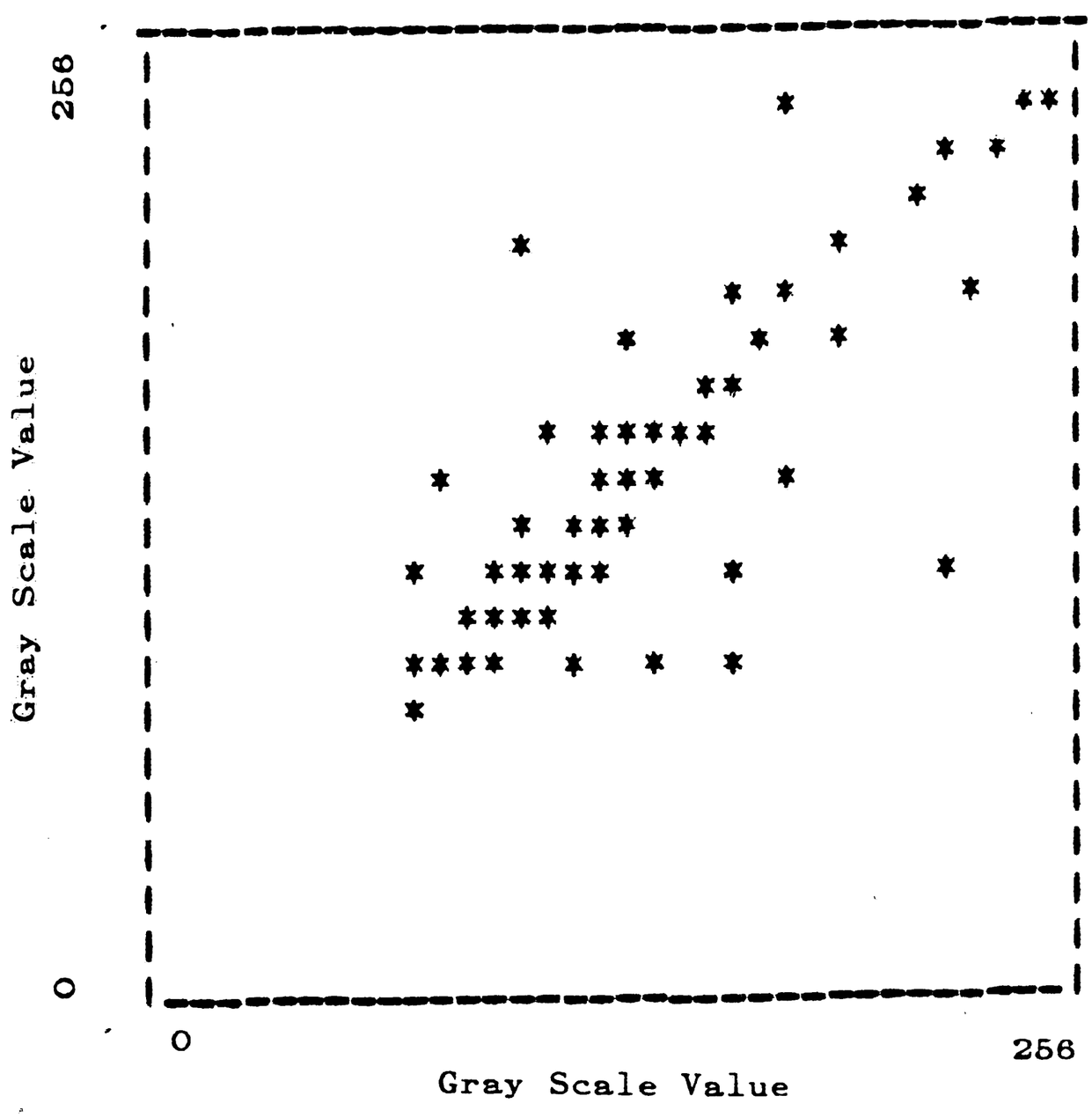


Figure 3: Scatter Plot of a Quantized Image.

In the next section. the various algorithms that can be applied to the creation of a VQ (FS or otherwise) will be discussed. Before leaving the basics. a few more matters should be mentioned.

A monochrome digital image typically consists of an array of integer numbers between 0 and some power of 2, with the magnitude of each number corresponding to the brightness of an image pixel. Therefore, a k-dimensional vector drawn from such data must reside within a k-dimensional hypercube with one of its corners on the origin. This will be a useful basic fact when we wish to generate a uniform k-dimensional quantizer, as will be discussed later.[12]

Much labor has been devoted to studying the theoretical bounds on the performance of vector quantizers.[13] The fundamental result which makes vector quantization so attractive is that the SNR of a VQ system approaches the "ultimate rate-distortion limit for a given source"[14] for long block length. However, block length need not be long for good performance,[15] and very long lengths are impractical anyway because of the computational burden they impose. The largest dimension that has been used in published work that I know of is sixteen.[16]

Also, the results of such analysis have not proven particularly applicable to real VQ systems except in the case of some speech work, where a reasonable

---

[12]It is not always the case. For instance, if varying bit allocations are used for the vector components in a transform coding scheme, the region of R^k in which such vectors might be found is not only not cubical but is not even constant. It is hard to see how one would define a "uniform quantizer" in such a situation.

[13][2], [5], [7], [23].

[14]Gersho and Cuperman, p. 15.

[15]Gray and Linde, p. 381.

[16]Gersho and Ramamurthi.

statistical model of the source is available.[17] As one paper puts it,

> since theory can characterize only optimal performance values ... the best available means of finding the performance limits of a class of codes of. nonasymptotic rate [bits/pixel] and block length remains the actual design and test of good codes incorporating optimization techniques...[18]

Because of the difficulty of calculating useful performance predictions for a general image VQ, this paper confines itself to analyzing the actual behavior of several such algorithms.

III. Vector Quantization Algorithms

A. FSVQ Design

As mentioned above, our ability to define the codebook in terms of the partition, and vice versa, is the key to designing optimum or near-optimum quantizers.

An algorithm published by Linde, Buzo, and Gray in 1980, often referred to as the LBG algorithm, exploits the complementary nature of codebooks and partitions to iteratively approach the optimum codebook starting from a "reasonable guess".

There are at least three distinct ways to make a "reasonable guess" and improve upon it; the splitting method (used by the LBG algorithm and most of its descendants), the product code method, and the uniform quantizer initial guess. All of the algorithms outlined here are for use on a source with an unknown distribution, namely, a training sequence of real data.

The iterative splitting method is as follows (see Flow Chart 1):

---

[17] Abut, Grey, and Rebolledo.

[18] *Ibid.*

(1) **Initialization.** Set NN = 1. This is the number of vectors in the current codebook. (The algorithm goes through a number of intermediate codebooks before finishing.) To begin, we must also specify N, the desired size of the final codebook, and ϵ, the convergence decision threshold. The meaning of ϵ will be explained at stage (2). Let Y_n be the codebook during the current iteration; it contains NN vectors. To finish initialization, we set the single vector which Y_1 contains equal to the centroid of the entire training sequence S:

$$Y_1 = y_1 = \sum_{x \in S} x. \text{ /(\# of vectors in S)}$$

(2) **Splitting.** Given the NN codebook vectors of Y_k from Step 1, "split" each one into B nearby vectors by adding B arbitrary, small vectors δ_j, $j = 1, 2, \ldots, B$. Set total stage distortion $D_1 = \infty$. Set NN = 2×NN.

(3) **Optimization.** Optimization is an iterative procedure for a fixed codebook size. It must be completed satisfactorily before returning to step (2).

(a) *Partitioning.* Find the best partition for the current codebook of size NN. This is done by evaluating $d(x, y_i)$ for all $x \in S$, and partitioning each x to that y_i for which the distortion is minimum. In other words, find out which codebook vector each training sequence vector is closest to.

(b) *Distortion.* Calculate the average distortion of the quantizer using the partition from part (a) by adding up all of the minimum $d(x, y_i)$ and dividing by the number of vectors in the training sequence. This is, of course, exactly the average distortion that we would have between the training sequence and a version of it quantized using the current codebook, since the procedure for "partitioning" is the same as that for "encoding."

Set D = the average distortion.

(c) *Test for convergence.* This is where ϵ comes in. ϵ is some number smaller than 1; it should be quite small. It is called the "convergence decision threshold" because if

$$(\,|\,D_1 - D\,|\,)/\,D \leq \epsilon.$$

then we assume that only negligible improvements will be realized by further iterations, and we end the optimization loop by jumping. (If NN = N, the size of the final codebook, then we jump to (4), **End**. If NN < N, we jump back to (2).) Since D_1 was set to ∞ before beginning optimization, the test for convergence is sure to fail the first time through. If the test is not satisfied, then we proceed. Set $D_1 = D$.

(d) *Find codebook.* As promised much earlier, the dual definitions of the y_i and the R_i are the essence of optimization. Having determined in (c) that the improvements being gained by iteration have not bottomed out yet, we now find a codebook to satisfy the partition that has been inherited from Step (b). This is done by replacing every vector in Y_n (the current codebook) by the centroid of the corresponding partition. This will be significantly different from the codebook already in place if the optimization stage has not yet iterated a sufficient number of times. Unconditionally, go to (a).

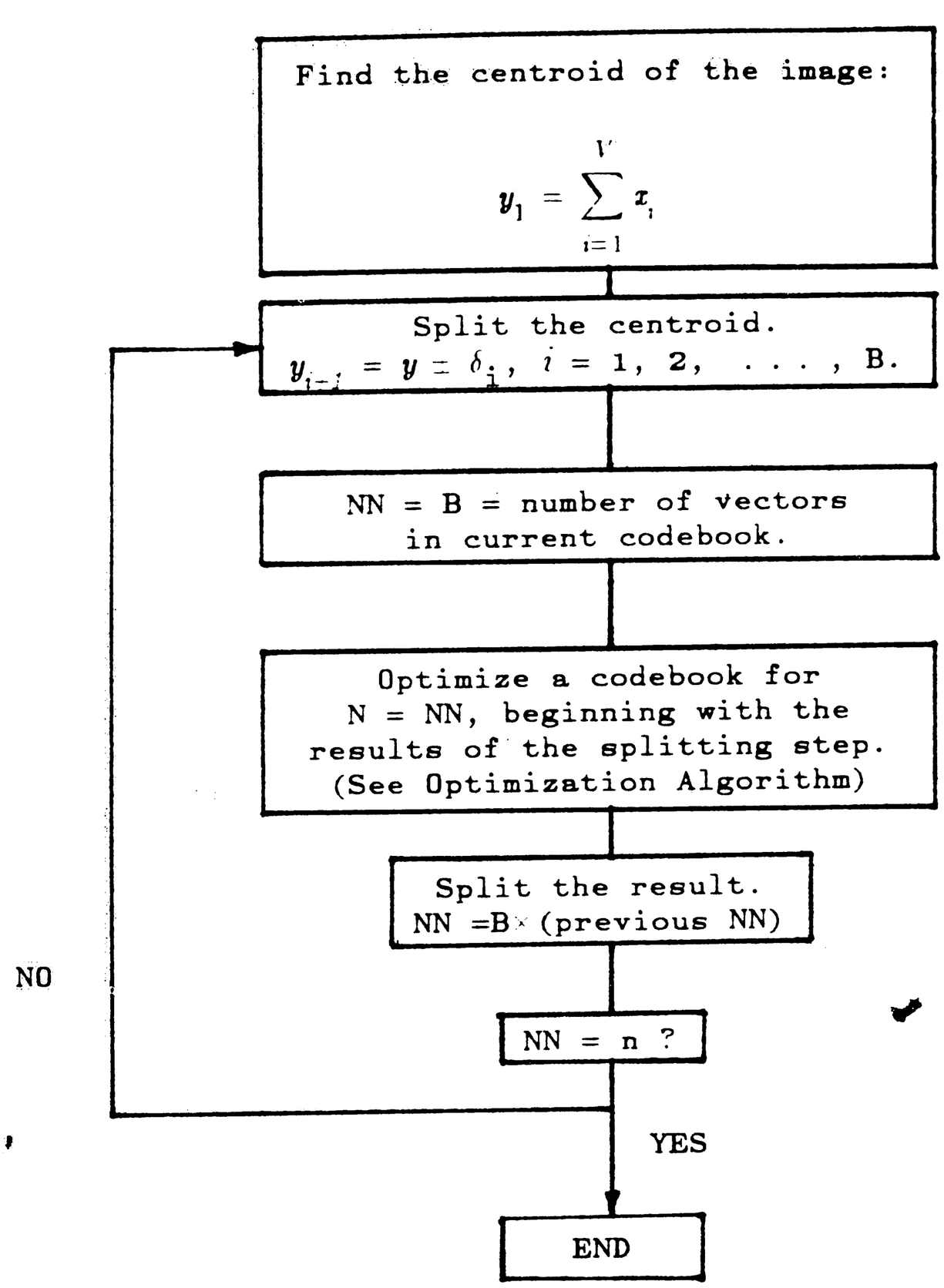
(4) **End.** Reached only from (2b). The codebook has been split a sufficient number of times, and the final codebook containing N vectors has been found from an initial guess based on a codebook containing N/B vectors, which was the product of the last *Splitting* operation. Stop.

Note that, unless we wish to perform the splitting operation differently

Flow Chart 1

FSVQ DESIGN ALGORITHM

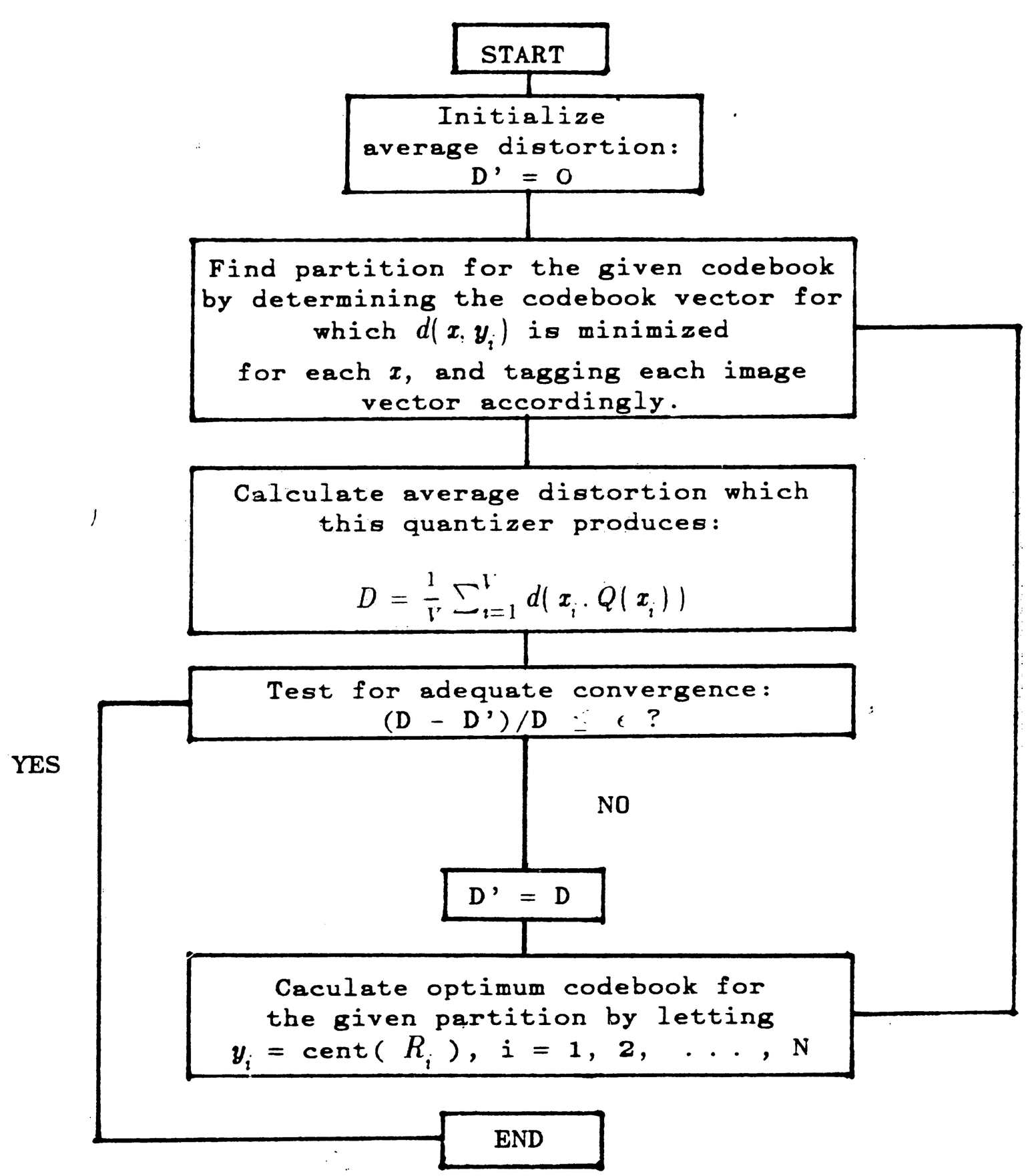
- Arbitrary dimension
- V vectors in training sequence
- N is desired codebook size



Flow Chart 2

OPTIMIZATION ALGORITHM

- ϵ = decision threshcld
- Fixed codebook size NN
- V image vectors



somewhere along the line, the final codebook size N will be a power of B. If B = 2, then the tags identifying the N codebook vectors will completely use up all the binary words of length $\log_2(N)$. Therefore, if the quantized image is being stored or transmitted in the form of codebook tags without any added redundancy, binary splitting (or splitting by some power of 2) will lead directly to a codebook of the most efficient size.

Of course, now that we have a quantizer, we can use it on any input that we please. The results are only guaranteed to be optimum or locally optimum if the training sequence was long enough to accurately represent the statistics of a class of inputs, and all future inputs are also members of that class. Linde, Buzo and Gray distinguish between the case where "*only* the training sequence is to be classified" and that where we wish to produce a good quantizer for use on data outside the training sequence which has the same statistical properties as that within.[19] However, this distinction has no bearing at all on the algorithm itself and it shall be disregarded until the discussion of results.

Throughout the LBG algorithm as described above, all vectors dealt with were of the same dimension, *k*. An alternative method of growing codebooks exists, although it is rarely mentioned.[20] The "product code" method, as it is called, is briefly outlined below.

(1) *Initialization.* Begin with two vectors of dimension 1 as an initial guess. Set NN = 1 and $d = 1$. "*d*" is the dimension of the stage in progress.

(2) *Optimization.* Use the same optimization routine as in the splitting

---

[19] [17], p.89.

[20] The description of this method by Juang and Wong [13] is the only one that I know of.

method to find an optimum codebook of size NN and dimension d. If $d = k$, the desired final dimensionality, stop.

(3) *Forming a product code.* Form the initial guess of the next pass through (2) by appending a new dimension to the optimum codebook of dimension d and forming all permutations of the original codewords and two fixed possible values for the new dimension. (These values. or "multiplication factors," play the same role as the δ's of the splitting method.) Go to (2).

Juang and Wong report that the product code method does not provide startlingly different results from the splitting method, and certainly not superior results. In one case they found that the product code method converged to a truly distinct quantizer, apart from that found by the splitting method, which proves that the restriction "locally" optimum is not just a mathematical fiction. The product code method was not implemented for this project.

B. Tree-Searched Vector Quantizers

The LBG algorithm, though wordy, is simple in concept and realization, considering the task it has to perform. It produces a codebook that has no particular structure and is appropriate for use in an FSVQ. However, it is computationally very burdensome. N vector distortion computations are required to quantize a single input vector, since all of the N codebook vectors must be searched. There is an alternative method, the tree-searched vector quantizer (TSVQ), which is not only far easier to search through but is quicker to design. There is, naturally, a price to pay; a TSVQ will generally be suboptimal in comparison to an FSVQ, and it requires up to twice as much memory. However, the computational savings are so great as to make it well worthwhile to look at TSVQ's.

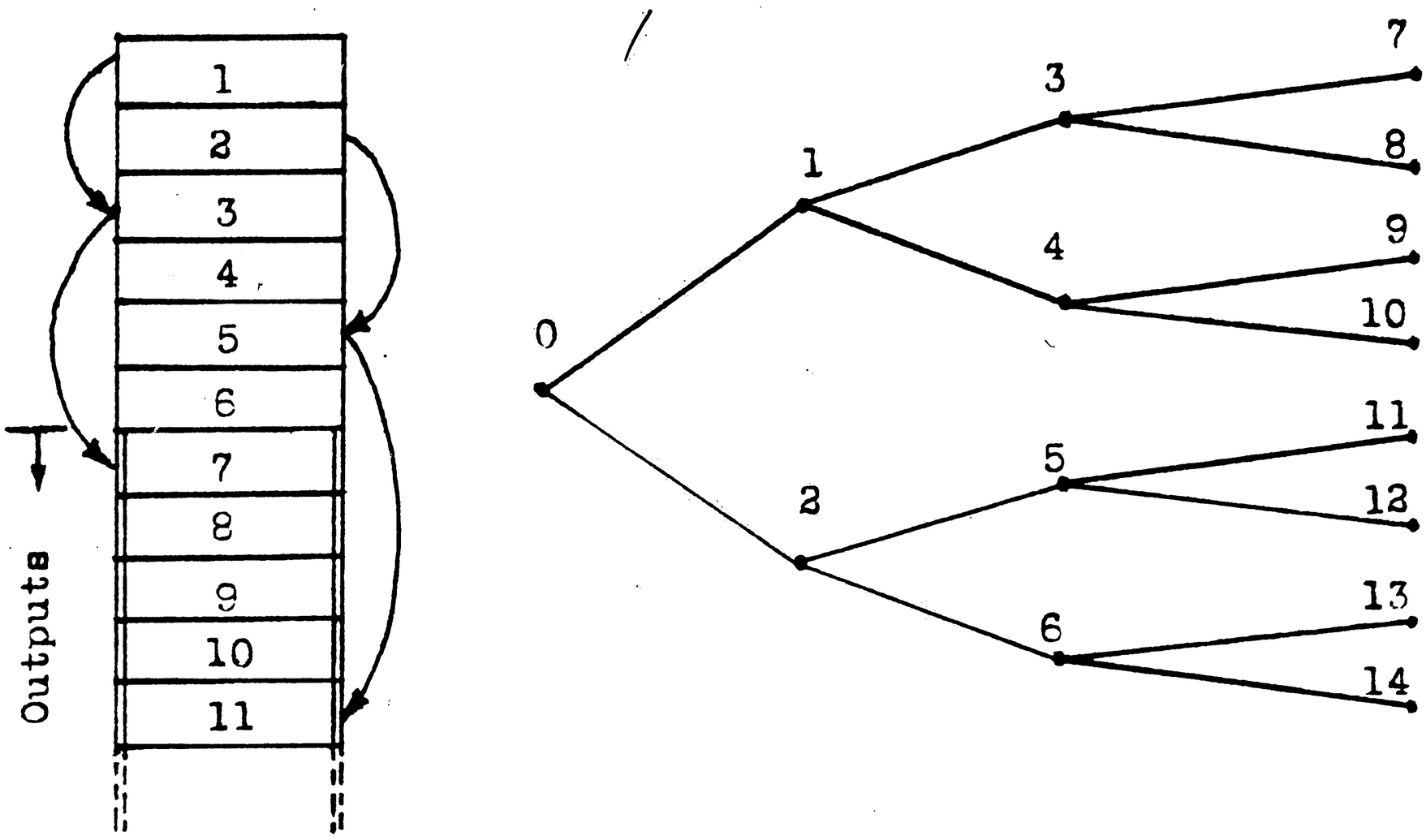


Figure 4: TSVQ Structure

A typical tree is drawn in Fig. 4. A TSVQ is designed and searched from left to right -- at each level, a new (and larger) set of vectors (corresponding to the nodes in Fig. 4) must be computed and retained permanently. The fact that the whole tree must be stored is the source of the extra memory requirement for TSVQ's.

The nodes at the Lth level, on the far right of Fig. 4, are the output vectors of the quantizer. Most of the other nodes serve to facilitate the search procedure, which is as follows:

(1) Ignore the single node at the base of the tree. This is only needed during the design stage and need not be stored.

(2) Treat the 2 children of the base node as an FSVQ with N = 2, and quantize the input vector accordingly -- that is, find which of the two is closer to the input.

(3) Repeat step (2) for the children of the node which was chosen as the "output" of the first-level quantizer, and then for the children of *that* node, and

so on until the output level of the tree is reached.

As can be readily seen, a binary TSVQ requires at most 2×L vector distance computations, in contrast to the FSVQ. Furthermore, there is no reason why we should restrict ourselves to binary trees, i.e., those with 2 children per node. If a tree has B children per node then the quantizer will have B^L output levels and require B×L distance computations for the quantization of a single input.

The creation of the tree is a more elaborate process than its searching. I present the algorithm below in a form similar to that used by Gray and Linde.[21]

As the encoding procedure works its way through the tree, it must have some way of keeping track of where it is. This is done by assigning an integer label b_i to every node vector in the tree, and retaining these integers in the form of a "path map" $\{b_1, \ldots, b_m\}$ to record which nodes the quantizer has passed through up till the mth level. "m" will equal L, the number of levels, when the search procedure is complete. Gray and Linde point out that since the b_i must all be distinct, each can be uniquely represented by a binary number B_i, and the binary channel vector pointing to the desired output vector at the Lth level may be simply $\sum_{l=1}^{L} B_i$. The assignment of b_i's to the tree nodes can be made independently of the actual contents of the nodes, which is determined as follows:

(1) **Initialization.** As during the design of the FSVQ, find the centroid of the training sequence. Place this at the base node of the tree. After this step is over, it need not be retained.

(2) **Level *l*.** There will be B^l nodes on the lth level, beginning with 1 node on the 0th level. Repeat the next stage B^l times, independently for each node on

---

[21] [12]

the *l*th level.

(a) *Node i of level l.* A certain subset of the training sequence will be partitioned to this node. If it is the root node, then *all* of the training vectors will be partitioned to it. Split the node vector into B nearby vectors, just as was done during the FSVQ splitting step. Then, optimize a B-vector FSVQ utilizing only those training vectors which are partitioned to this particular node. When the optimization is finished, set the B children of this node equal to the optimized B-vector FSVQ codebook. To each of the children, some of the training vectors will be partitioned. Retain this information for use when the time comes to form the children of *these* nodes.

(b) If i = B+1, go to (3). Otherwise, increment i and go back to (a).

(3) If l = L, go to (4). Otherwise, increment l and go back to (2).

(4) **End.** All the nodes on all the levels have been dealt with.

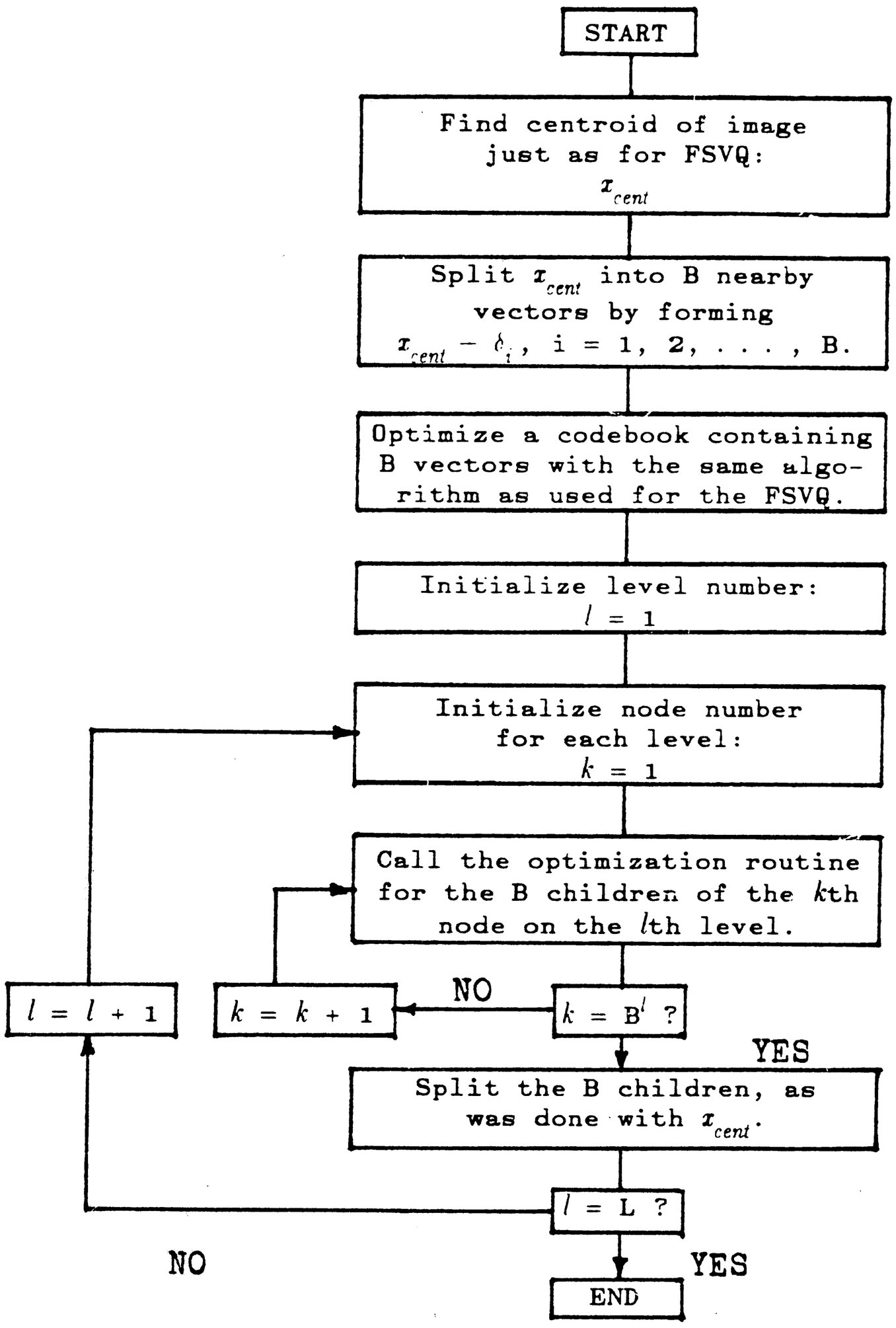
A flow chart for the TSVQ design algorithm is included on the following page.

Some comments are in order. One can decompose the idea of a quantizer into an encoding step and a decoding step: encoding means deciding which of the codebook vectors an input vector should be mapped to, and decoding consists of forming an image using this information and the codebook. In a storage or transmission system, this separation of functions would be quite real. The *decoder* for a TSVQ is the same as that for an FSVQ: one need only store the N reproduction vectors and their identifying tags (the binary channel vectors), and create a new image based on this codebook and a string of tags. It is *encoding* that makes tree- and full-searched quantization different.

Flow Chart 3

TSVQ DESIGN ALGORITHM

- V image vectors
- N codebook vectors desired at highest level of tree
- L levels
- B children per node



The amount of vector storage needed for a TSVQ of order B with L levels can be easily calculated as

$$N_{B,L} = \sum_{i=1}^{L} B^i = \frac{B^{L+1} - B}{B - 1}$$

This can be close to 2×B. but not greater. so using a TSVQ can at most double the memory needed during the encoding stage of the quantizer.

In practice. there is no need to go through an elaborate procedure of accumulating a "path map" and translating it into the address of the appropriate output vector. Trees can be numbered in a fashion which makes the addressing of nodes throughout the search and design procedures simple and automatic.[22] If one numbers the nodes of the tree one level at a time. starting with 0 for the base node (see Fig. 4). every node will have a distinct number n and the addresses of its B children (for a Bth order tree) will be given by

$$n \cdot B - i, \quad i = 1, 2, \ldots, B$$

For a computer implementation of a TSVQ, this makes for very convenient travelling through a single stack containing all of the nodes of the tree in an orderly arrangement, as is also indicated in Fig. 4.

This method dispenses with the need for keeping track of some kind of path map, even if not all the nodes on all the levels are of the same order. In that case, one would merely need to number the whole tree according to the order of the highest-order node, and use up some of the numbers on non-existent children for the lower-order nodes. For this paper, a program was written which could

---

[22] I chanced upon this technique for the binary case, and am indebted to Mr. Peter Floriani for pointing out the general case.

design TSVQ's for any fixed B and any number of levels, but no provision was made for varying numbers of children per node.

C. Uniform Vector Quantizers.

The most obvious possible arrangement of output vectors for any kind of VQ is the uniform quantizer; that is, an arrangement of codebook vectors in R^k which causes an equal volume of space to be assigned to each when R^k is partitioned according to the nearest-neighbor rule. This is not always as simple as it sounds. There is no fixed way of scattering an arbitrary number of points through R^k so as to assign equal volume to each -- and of course, we would like to avoid iterative methods of doing so, since the other two ways of designing an optimum quantizer are iterative anyway, and an iterative method of generating a uniform quantizer would not necessarily have any particular advantage. Since the region of the "image space" we are dealing with in the case of digital gray scale images is a simple hypercube (i.e., a region where the value of any coordinate can vary between two fixed values, which are the same for all coordinates), it is rather easy to produce certain numbers of uniformly distributed output vectors -- namely, anything of the size $2^{k \cdot n}$. A k-dimensional hypercube has 2^k "corners", and since the space around each corner can be considered as another, smaller hypercube, this division of R^k is easy to generate. For $k = 2$ and $n = 1$, for instance, the $2^{2 \cdot 1}$ "output vectors" of the uniform quantizer, where either coordinate of the region to be quantized can range between 0 and R, are given by

$$y_1 = (R/4,\ 3R/4)$$
$$y_2 = (R/4,\ R/4)$$
$$y_3 = (3R/4,\ R/4)$$
$$y_4 = (3R/4,\ 3R/4)$$

A uniform quantizer initial guess enables one to design an FSVQ (or the children of one node at a time during the design of a TSVQ) without going

through an elaborate process of codebook-growing. However, there are practical difficulties with this method which shall be discussed in the next section, in addition to the difficulty of generating a truly uniform quantizer for an arbitrary codebook size.

IV. Results and Discussion

A. FSVQ Results

Image data was available in the form of two 32×512 blocks of pixels whose gray scale values were integers ranging up to 256. Because of run time limitations, data was collected thoroughly for the performance of the FSVQ and TSVQ only on one of these blocks. Quantization was performed on the other block for selected dimensions and codebook sizes to verify that the results were similar.

For this relatively small set of training data, the algorithms described in Section III are not so much designing a quantizer for a class of inputs as they are "grouping a long sequence of vectors in a low distortion manner."[23] The outstanding result of this fact is that the average distortion per pixel resulting from quantization *increases* with increasing dimension for constant codebook size, contrary to what is usually said about vector quantization. The reason for this is as follows:

The original image is represented by 8 bits/pixel (bpp). Therefore, if there are P pixels in the image, 8·P bits are needed for its complete representation. In a vector quantization scheme, $\log_2 N$ bits are needed for each of the codebook labels or channel vectors. Furthermore, if the dimension used is k, then there are

---

[23] Linde, Buzo, and Gray.

P/k such binary numbers needed to represent the image, given the codebook. The codebook itself requires $8 \cdot N \cdot k$ bits, since it is an $N \times k$ array of pixels with the same gray-scale value range as the original image. Therefore, the *compression factor* of the quantizer, or ratio of the bits needed for the quantized image to the bits needed for the original, is given by:

$$c(k, N, P) = \frac{8 \cdot P}{8 \cdot N \cdot k + (P/k) \cdot (\log_2 N)}$$

For $P \gg N \cdot k$, this will approximately equal $8 \cdot k / \log_2 N$, which is the standard that is always used by authors when the amount of available data is great enough. Fig. 5 shows the compression factor for both cases. It is reasonable to expect that the distortion performance of the quantizer should follow the bpp of the quantized representation, and this is true for relatively small dimensions. Fig. 6 shows the performance of three different-sized quantizers ($N = 2$, 16, and 64) for increasing dimension.

It can be seen that $c(k, N, P)$ is a decreasing function of N and therefore one would expect the distortion to also decrease with N. This is in fact true, as can be seen from Fig. 7. However, $c(k, N, P)$ is not a monotonic function of k; for small k it increases, and for large k it decreases, with a maximum at

$$k = \left[\frac{P \cdot \log_2 N}{8 \cdot N} \right]^{1/2}.$$

From the above, one would expect the distortion to start decreasing with increasing k at some point for any given N. Several trials showed that this is not true. Probably the cause is that there are only so many *distinct* vectors in any image. As the number of codebook vectors becomes a reasonable fraction of the

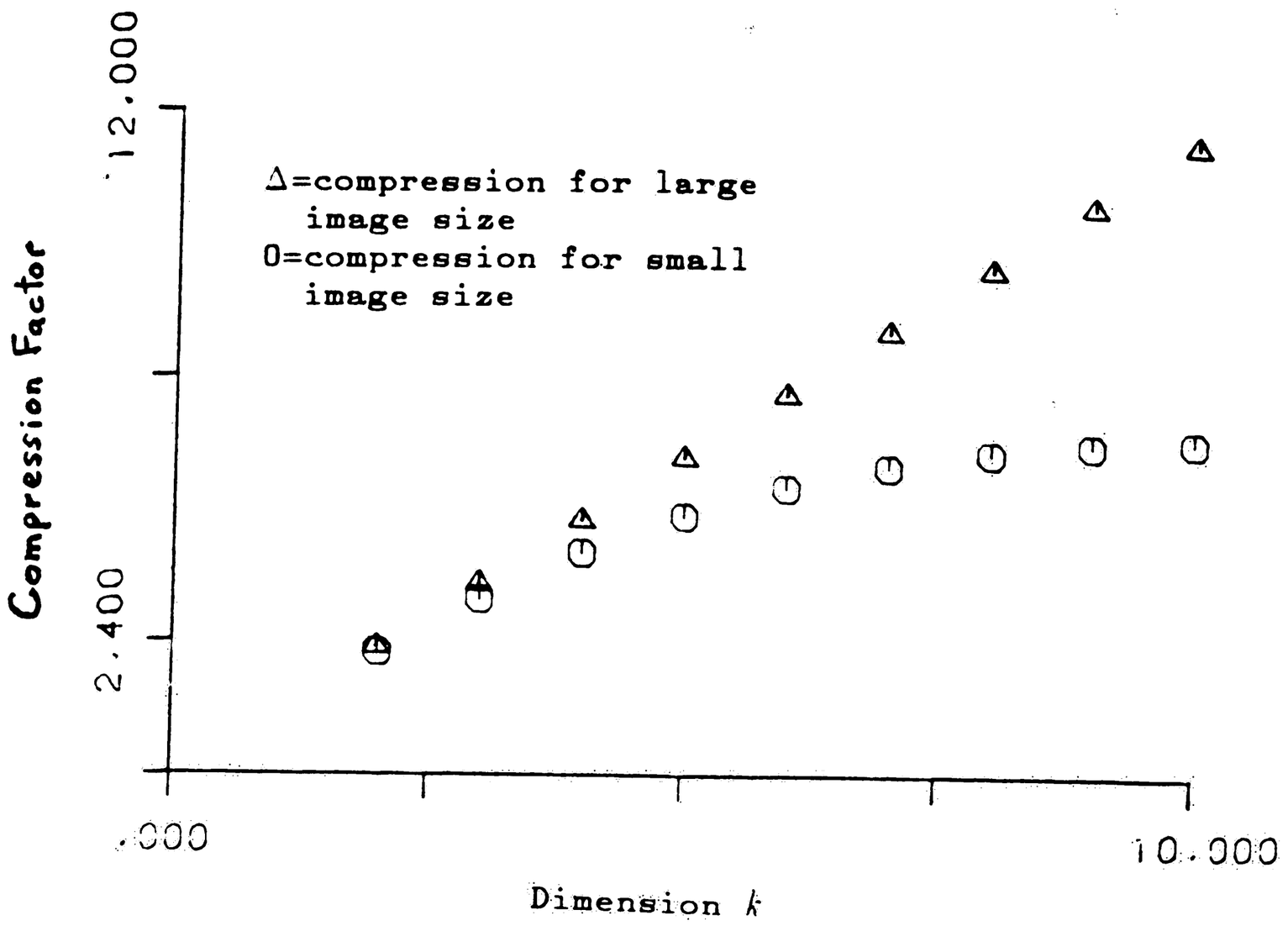


Figure 5: Compression vs. k for N = 128

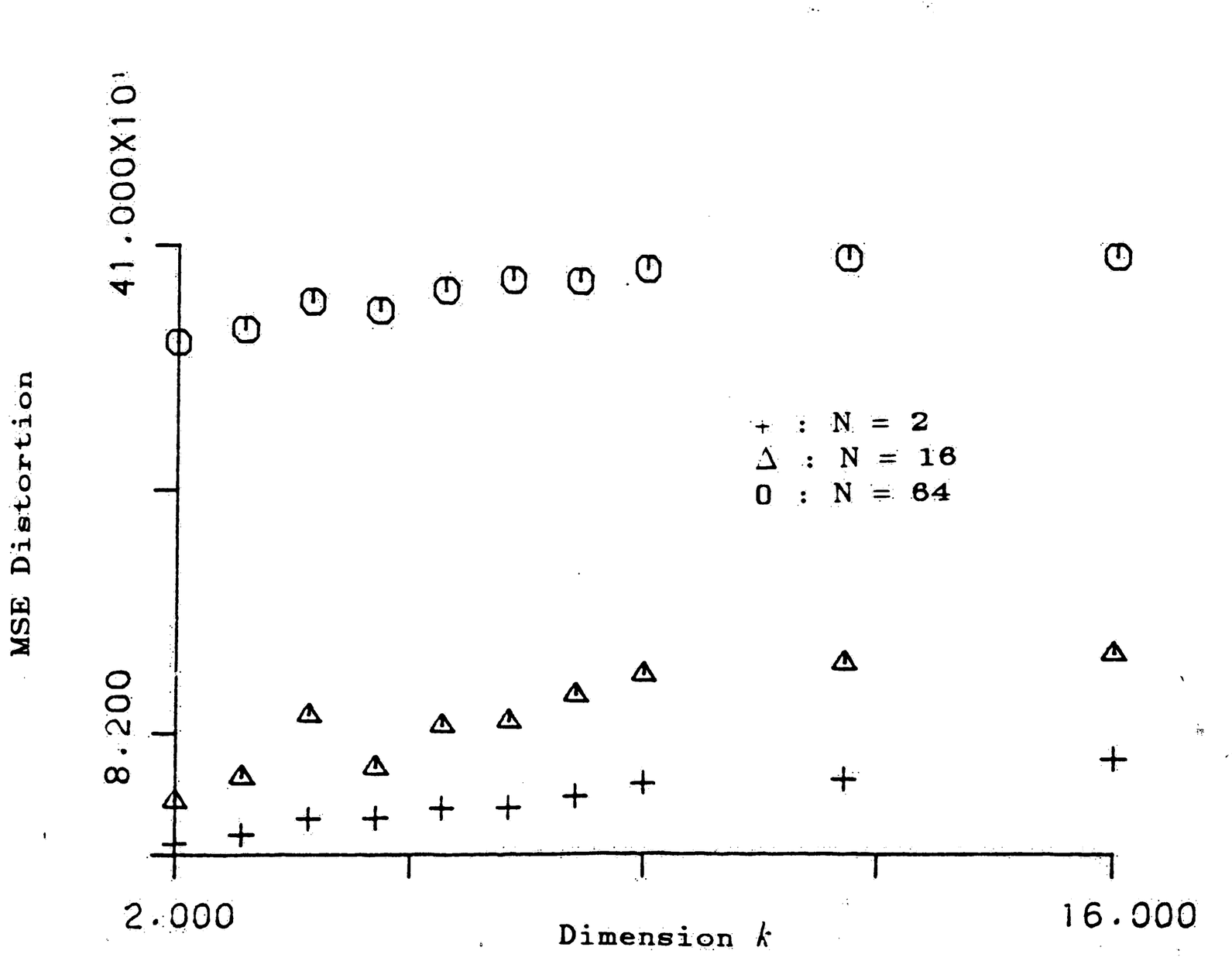


Figure 6: Distortion vs. k for N = 2. 16. 64

number of image vectors. there will be a codebook vector to represent almost

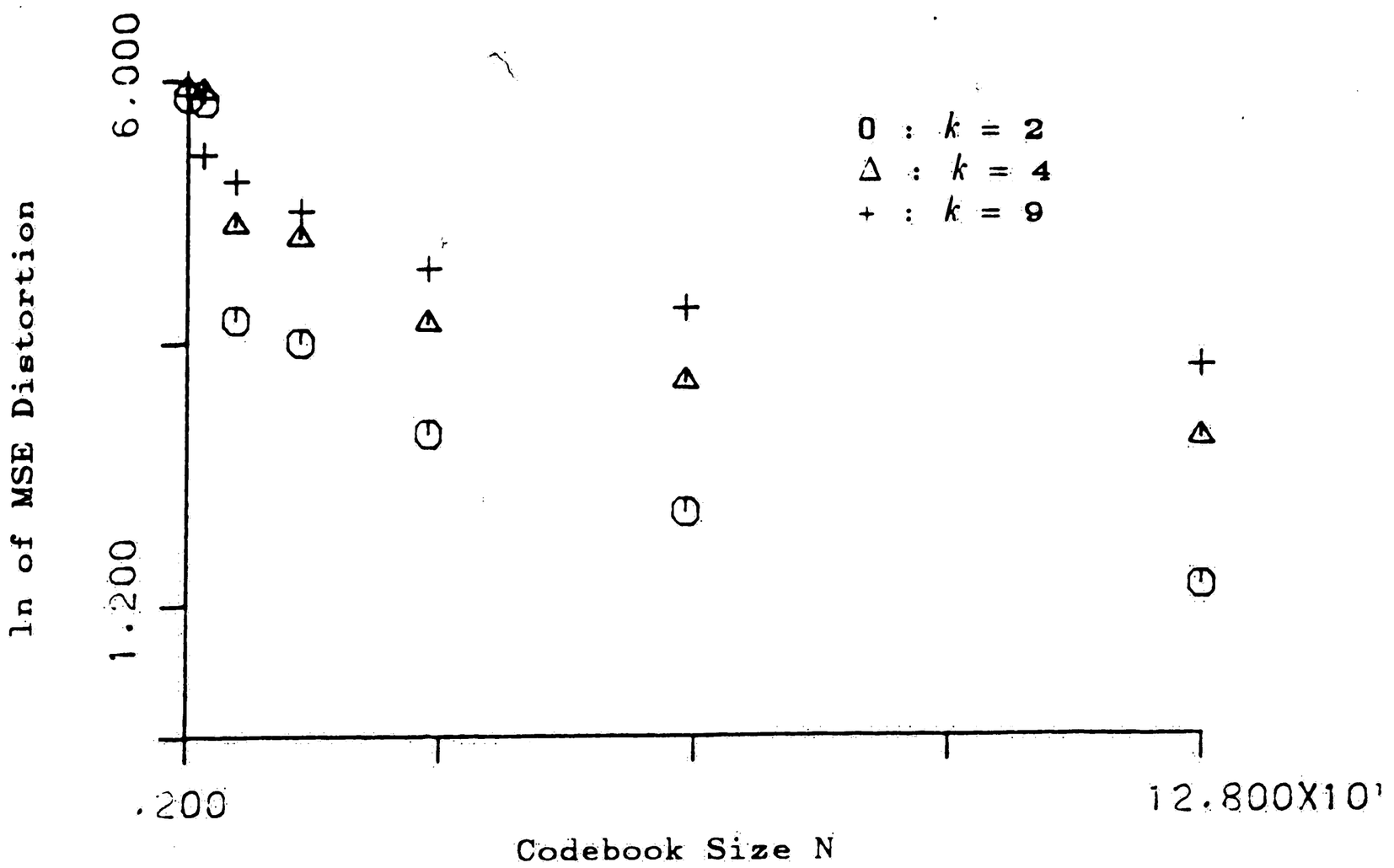


Figure 7: Distortion vs. Increasing Codebook Size for $k = 2, 4, 9$

every distinct image vector, with the result that N will effectively not increase beyond a certain point regardless of what we do. This effect might be called "partition emptying," because when a partition becomes empty during the LBG algorithm -- which occurs when there are no more distinct vectors to assign to it -- its centroid is 0 and it no longer significantly participates in the process. Several tests showed that for N on the order of 256 or 512, all codebooks had such empty partitions in percentages which increased steadily for increasing k.

Despite the above considerations, it was true even for the limited image sizes which were available for testing that for fixed bpp (as measured by the large P standard), distortion performance generally improved for increasing N and k. This was as expected. Fig. 8 shows the improvement for the case of 1 bpp (a compression factor of 8).

Another consideration in VQ implementation is the block shape. Any integer factorization of k corresponds to a different block shape. Usually, square

blocks are chosen. limiting the choice of dimensions to k's which are perfect squares. In Table 2. FSVQ performance for varying block shape and dimension is tabulated. and in Fig. 8 the performance for $k = 4$ and the 3 different possible block shapes is graphed. As can be seen. for this image the square block performance is almost always intermediate between that of the horizontal linear and vertical linear blocks. Presumably this is caused by the characteristics of the individual image. In general. it is difficult to say that any shape is better than any other. A linear block will take advantage of correlation between samples that are farther apart. but will be treating the image as if it were only a 1-dimensional signal. thereby losing some advantage.

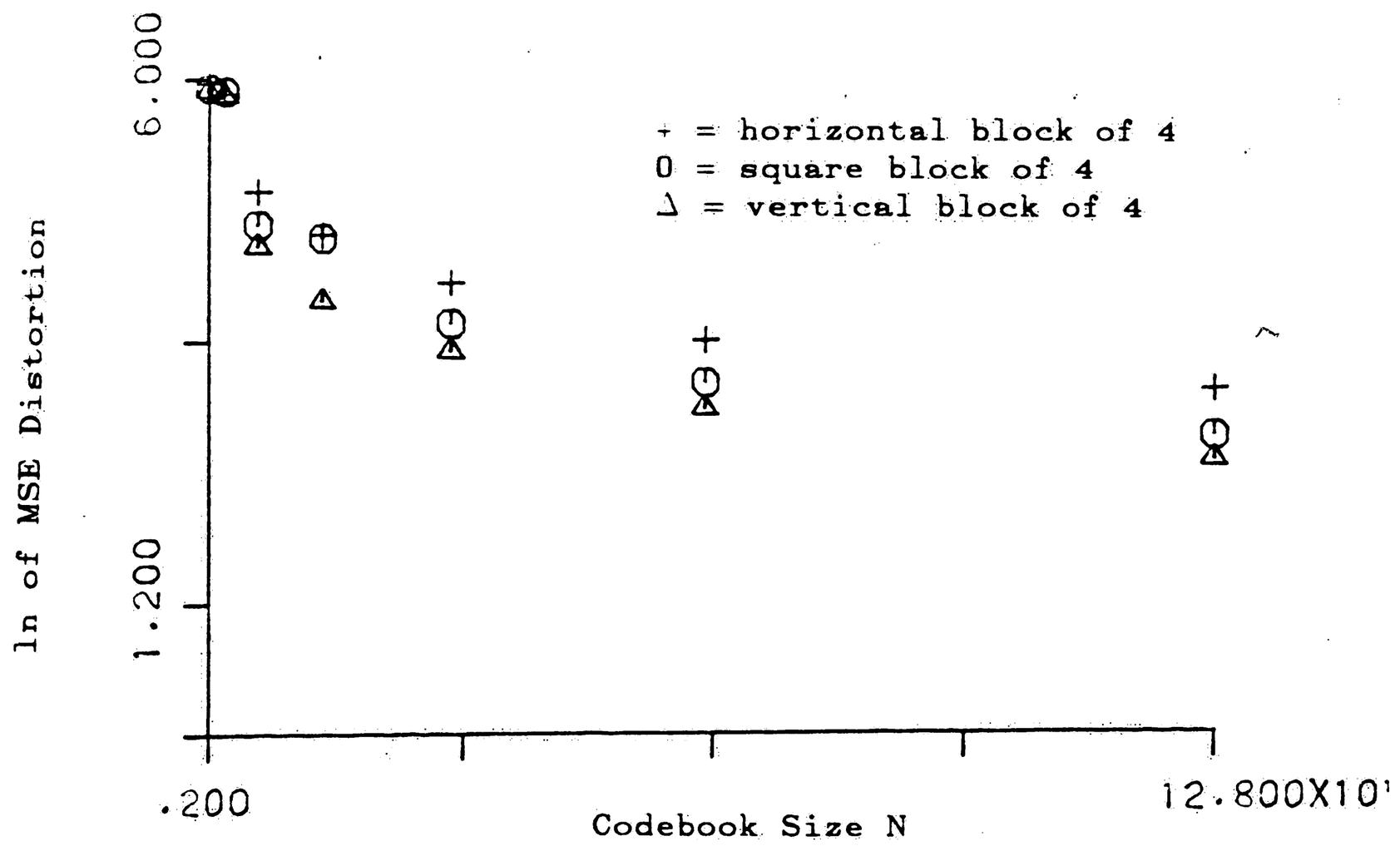


Figure 8: Distortion vs. Codebook Size for $k = 4$

There are numerous "tricks" that can be played with the FSVQ. One of them -- the distinction between edge and shade blocks -- corresponds to the distinction between voiced and unvoiced speech. although it is not so useful. An

FSVQ which gives minimum distortion may yet generate unpleasant "stepping" effects when quantizing diagonal lines. One possible way to attack this problem is to give more attention to edge blocks due to their perceptual significance than they would otherwise receive. This is done by designing a separate codebook for the edge blocks of an image, designing another for the shade blocks, and then combining them. This was done with results that are shown in Figs. 9 and 10.

Fig. 9 shows an image quantized with a $k = 2$, $N = 64$ codebook.

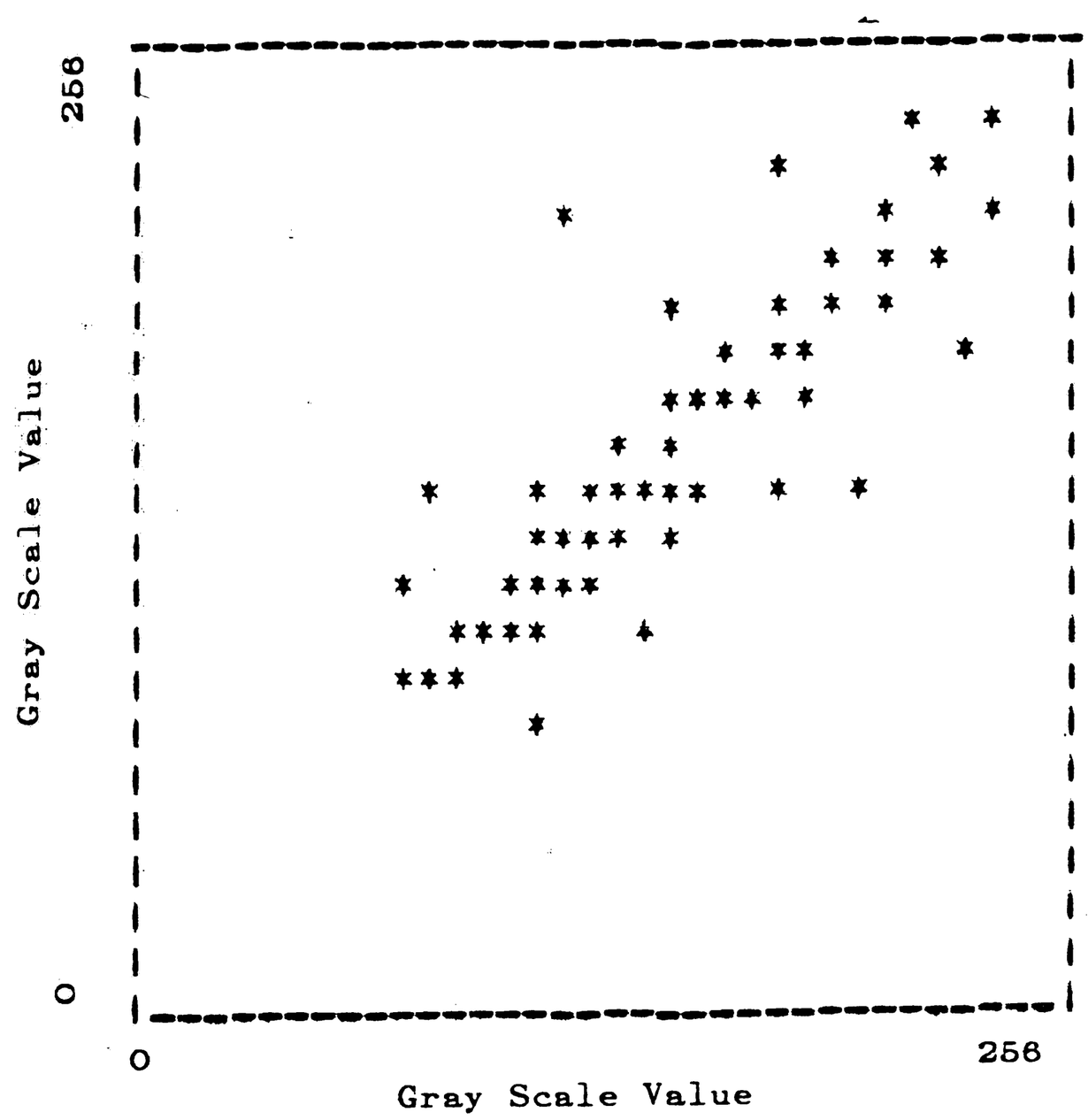


Figure 9: FSVQ Coded Image Scatter Plot

Fig. 10 shows the same image quantized with a codebook of the same size, but with half of the codebook specialized to each class of image vector. Plainly, there is a better representation of edge blocks. (An adjacent-pixel difference decision level of 20% of the brightness range was used for this example.) Without actually displaying these images, it is not possible to say for sure that this

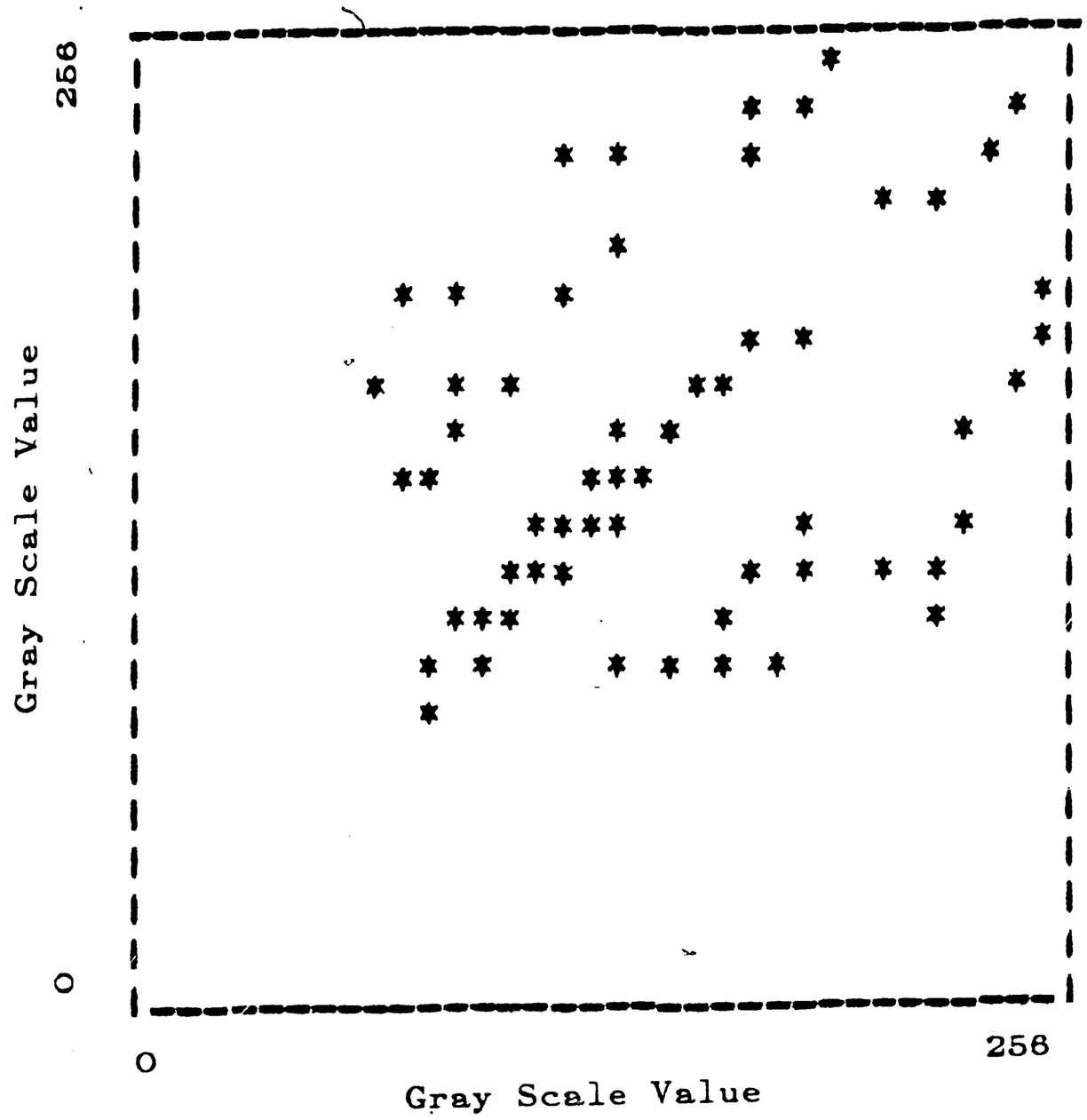


Figure 10: Edge-Block Adaptive FSVQ Coded Image Scatter Plot

process has realized any qualitative image improvement.

B. TSVQ Results

It should first be pointed out that TSVQ's only make sense when data outside the training sequence is to be quantized. The purpose of the tree structure is to speed the quantization of vectors. something which has already in effect been done for the training sequence during the calculation of the tree.

Binary trees are the most widely used type. However. if the desired output codebook size N happens to be a power of 4. quaternary (fourth order) trees are superior in every way. The quaternary tree always requires less memory than the binary tree, because the memory requirements for identical N are given by

$$N_{2,2L} = 2^{2L+1} - 1 = 4^{L} \cdot 2 - 1,$$

$$N_{4,L} = (4^{L+1} - 1)/3 = (4^{L} \cdot 4 - 1)/3,$$

and $\quad (4^{L} \cdot 4 - 1)/3 < 4^{L} \cdot 2 - 1 \quad$ for all $\quad L > 0.$

-- where the quaternary tree has L levels. By virtue of having half as many levels, the quaternary tree will also be twice as fast to search. Furthermore, a wide variety of tests (Tables 3 - 6) showed that for the available image data, the quaternary TSVQ usually performed better than the binary. (Presumably this is because the quaternary is less constrained by the tree structure, having more choices of route at every node.) In fact, for small N it sometimes actually performed better than an FSVQ. Fig. 11 graphs the relative performance of the FSVQ and quaternary TSVQ for $k = 4$ and N equal to powers of 4. This is further proof of the *local* optimality of the codebooks found using any of these techniques. However, over a wide variety of images the FSVQ would almost certainly be found to perform better for all N.

The TSVQ's suffer from the same decay in performance with increasing k for limited image size as the FSVQ's. Although a TSVQ stores much more information, up to half of it is only used only to speed the search process and does not increase the actual bpp of the quantized representation.

The problem of "partition emptying" applies to TSVQ's with a vengeance. An empty node at the mth level will result in $\sum_{i=1}^{L-m}(B^{L-m+1} - 1)/(B - 1)$ empty nodes further down the tree (including B^{L-m} zero output vectors) unless steps are taken to assign a vector to it according to some rule.

C. UVQ and Other Results

A uniform quantizer was implemented for several codebook sizes and dimensions. The performance was very poor, as can be seen from Table 7. The

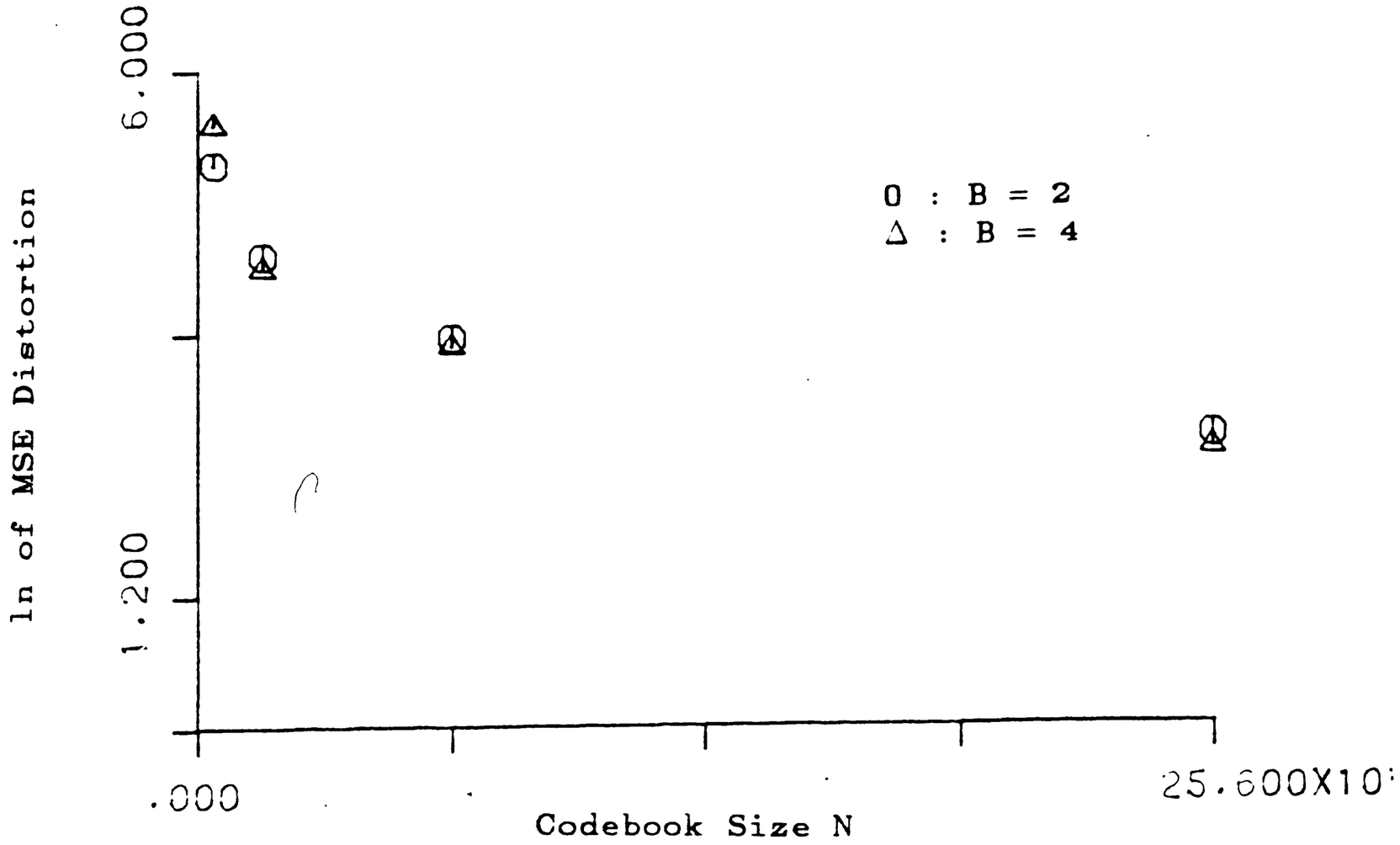


Figure 11: Relative Performance of Binary and Quaternary Trees

reason for this is probably that many of the initial guess points were scattered through regions which contained no image vectors: so. during the first part of the optimization routine. no image vectors were partitioned to them, and during the second part the codebook vectors corresponding to them were set to zero. Some kind of reassignment procedure is clearly called for. but adding such elaborations destroys the simplicity which is the whole attraction of the UVQ initial guess.

Some results were obtained for the l_1 norm distortion measure in an FSVQ (Table 8). In general. its behavior was similar to that of the MSE measure. although for small N increasing k seems to have very little effect on the distortion performance. In Fig. 12 the performance for $k = 2$ is plotted. It is impossible to say how subjectively acceptable the results of using the l_1 norm are without displaying the resulting images.

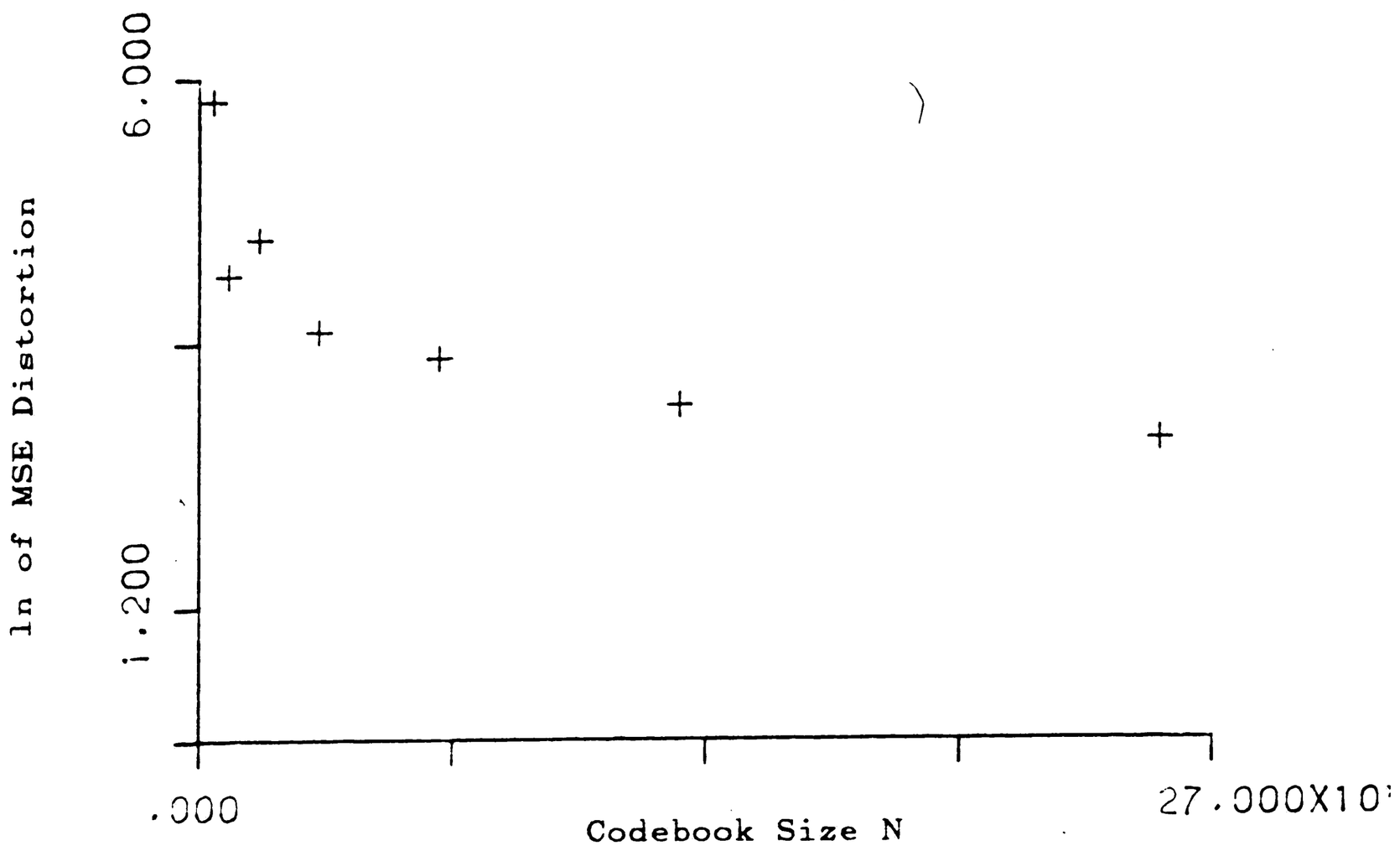


Figure 12: l_1 Norm Performance

V. Conclusion

Vector quantization is an extremely versatile technique. Many choices have to be made during its implementation which may affect performance. only a few of which have been discussed here. In the case of TSVQ's. it seems that higher-order trees perform better distortionwise in addition to using less memory and requiring less search time. Applying vector quantization to individual images as a data reduction technique introduces a tradeoff between dimensionality and bpp which is not usually considered in discussions of vector quantization. Some method of image display to provide qualitative results would be necessary to a more thorough evaluation of the performance of vector quantization on images.

Tables

TABLE 1
Mean Distortion Per Pixel (FSVQ)

N:	2	4	8	16	32	64	128	256
k:								
2	345.37	330.62	44.98	36.59	15.79	7.70	3.88	1.90
3	354.24	108.55	67.54	52.47	23.09	13.75	8.31	4.69
4	372.59	362.39	106.41	94.26	42.91	24.91	14.87	9.02
5	366.90	139.10	97.92	58.80	40.74	25.57	15.99	7.87
6	379.90	179.60	127.22	86.78	64.05	32.12	21.98	12.5
7	386.93	181.01	138.76	89.74	51.73	32.59	20.82	
8	386.17	374.49	148.27	107.09	59.42	39.88	25.88	
9	394.57	202.65	159.30	121.18	71.02	49.16	28.71	
12	400.90	381.12	168.97	128.81	76.58	51.39	33.91	
16	402.02	376.03	196.45	135.28	103.43	64.67		

TABLE 2
Distortion Performance for Varying Block Shapes
(m is block height, n is block width)

N:		2	4	8	16	32	64	128	256
m	n								
2	2	372.59	362.39	106.41	94.26	42.91	24.91	14.87	9.02
1	4	362.32	347.54	87.24	52.81	33.25	19.47	12.03	6.74
4	1	387.86	351.33	143.91	97.23	62.52	36.58	22.79	12.53
2	3	379.90	179.60	127.22	86.78	64.05	32.11	21.98	
1	6	375.15	357.30	113.96	65.28	43.53	28.31	18.33	
6	1	400.34	375.43	182.31	114.7	75.60	50.30	30.10	
2	4	386.17	374.49	148.27	107.0	59.42	39.88	25.88	
1	8	392.65	389.51	129.26	84.15	51.29	36.10	21.88	
8	1	410.39	377.59	197.89	133.6	100.6	60.30	35.05	
3	3	394.57	202.65	159.30	121.1	71.02	49.16	28.71	
1	9	396.88	180.11	139.20	91.05	58.61	37.12	21.70	
9	1	395.93	242.95	191.36	133.1	94.23	53.26	32.50	
4	4	402.02	376.03	196.45	135.2	103.4	64.67		
1	16	460.06	380.66	165.61	95.32	67.71	44.14		

TABLE 3
Mean Distortion Per Pixel vs. Codebook
(Size N and TSVQ Order B -- $k = 2$.)

	N:	2	4	8	16	32	64	128	256
B:	2	346.7	87.89	45.33	29.46	20.75	10.37	5.41	2.97

	3	9	27	81	243
3	237.2	49.38	26.64	10.92	4.12

	4	16	64	256
4	214.6	38.73	15.08	3.92

TABLE 4
Mean Distortion Per Pixel vs. Codebook
(Size N and TSVQ Order B -- $k = 3$.)

	N:	2	4	8	16	32	64	128	256
B:	2	355.4	113.3	68.94	54.87	42.39	27.14	17.05	8.35

	3	27	81	243
3	218.9	73.22	16.29	6.71

	4	16	64	256
4	109.5	54.99	17.95	7.29

	5	25	125
5	95.34	40.76	11.95

	6	36	216
6	101.2	28.97	8.80

TABLE 5
Mean Distortion Per Pixel vs. Codebook
(Size N and TSVQ Order B -- $k = 4$.)

	N:	2	4	8	16	32	64	128	256
B:	2	376.6	171.6	123.8	73.89	53.49	34.74	23.47	13.85

	3	9	27	81	243
3	361.8	98.24	54.42	29.93	13.50

	4	16	64	256
4	242.9	65.01	31.79	12.18

	5	25	125
5	142.6	51.86	21.88

	6	36	216
6	124.7	48.59	15.70

TABLE 6
Mean Distortion Per Pixel Vs. Codebook
(Size N and TSVQ Order B -- $k = 9$.)

	N:	2	4	8	16	32	64	128	256
B:	2	397.8	208.6	169.4	129.0	91.81	69.51	39.95	25.18

	3	9	27	81	243
3	294.5	160.3	105.7	52.99	23.66

	4	16	64	256
4	286.2	112.9	57.91	23.56

	5	25	125
5	283.6	92.08	38.57

	6	36	216
6	266.6	79.22	31.94

TABLE 7

Mean Distortion Per Pixel of FSVQ's With UVQ Initial Guess

k: \ N:	64	128	256
2			78.13
3	70.31		
4			87.89
6	40.63		
7		64.84	
8			73.44

TABLE 8

Performance of the l_1 Norm For Various Dimensions k and Codebook Sizes N

(Average Distortion Per Pixel)

k: \ N:	2	4	8	16	32	64	128	256
2	14.62	14.46	4.47	4.02	2.43	1.81	1.26	.90
4	14.80	14.30	6.11	5.63	3.82	2.98	2.33	1.89
9	14.81	8.59	6.84	5.82	5.05	3.84	3.15	
16	14.94	14.24	8.12	6.85	5.76	4.67		

Vita

Larry Clifford was born in East Orange, NJ on May 3, 1962 to Eugene and Lillian Clifford and lives in Blairstown, NJ. He received a BSEE from the Rutgers College of Engineering in 1984 with Highest Honors and attended graduate school at Lehigh University in 1984 and 1985 to obtain an MSEE.

References

[1] Abut, H., et. al.
Vector Quantization of Speech and Speech-Like Waveforms.
IEEE Trans. on ASSP ASSP-30, June, 1982.

[2] Bucklew, J.
Upper Bounds to the Asymptotic Performance of Block Quantizers.
IEEE Transactions on Information Theory IT-27, Sep., 1981.

[3] Buzo, A., Gray, A., Gray, R., and Markel, J.
Speech Coding Based Upon Vector Quantization.
IEEE Transactions on Acoustics, Speech, and Signal Processing ASSP-28, Oct., 1980.

[4] Buzo de la Pena, Luis.
Optimal Vector Quantization for Linear Predicted Coded Speech.
PhD thesis, Stanford University, Aug.. 1978.
PhD thesis for Dept.of Electrical Engineering.

[5] D. Chen.
On Two or More Dimensional Optimum Quantizers,
Proc. 1977 IEEE International Conf. on Acoustics, Speech and Signal Processing :640 - 643, 1977.

[6] Forgey, E.
Cluster Analysis of Multivariate Data: Efficiency vs. Interpretability of Classification (Abstract).
Biometrics 21:768, 1965.

[7] Gersho, A.
Asymptotically Optimal Block Quantization.
IEEE Trans. on Information Theory IT-25, July, 1979.

[8] Gersho, A.
On the Structure of Vector Quantizers.
IEEE Trans. on Information Theory IT-28, March, 1982.

[9] Gersho, A., and Cuperman, V.
Vector Quantization: A Pattern-Matching Technique for Speech Coding.
IEEE Communications Magazine , Dec., 1983.

[10] Gersho, A. and Ramamurthi, B.
Image Coding Using Vector Quantization.
Proc. of the IEEE International Conf. on Acoustics, Speech, and Signal Processing , May, 1982.

[11] Gray, R., and Abut. H.
Full Search and Tree Searched Vector Quantization of Speech Waveforms.
Proc. of the IEEE International Conf. on Acoustics, Speech, and Signal Processing , May, 1982.

[12] Gray, R., and Linde, Y.
Vector Quantizers and Predictive Quantizers for Gauss-Markov Sources.
IEEE Trans. on Communications COM-30, Feb., 1982.

[13] Habibi, A., and Wintz, P.
Image Coding by Linear Transformation and Block Quantization.
IEEE Trans. on Communication Technology COM-19, Feb., 1971.
The term "Block Quantization" refers specifically to transform coding in this context.

[14] Juang, B.
Multiple Stage Vector Quantization for Speech Coding.
Proc. of the IEEE International Conf. on Acoustics, Speech, and Signal Processing , May, 1982.

[15] Juang, B.
Distortion Performance of Vector Quantization for LPC Voice Coding.
IEEE Trans. on ASSP ASSP-30, April, 1982.

[16] Kasam.
The Mean-Absolute-Error Criterion for Quantization.
Proc. 1977 IEEE International Conf. on Acoustics, Speech, and Signal Processing :632 - 635, 1977.

[17] Linde, L., Buzo, A., and Gray, R.
An Algorithm For Vector Quantizer Design.
IEEE Trans. on Communications COM-28. Jan., 1980.

[18] Lloyd, S. P.
Least Squares Quantization in PCM's.
Bell Telephone Laboratories Paper, Murray Hill, NJ , 1957.

[19] Menez, J., et. al.
Optimum Quantizer Algorithm for Real-Time Block Quantizing.
Proc. International Conf. on Acoustics, Speech, and Signal Processing :980 - 984, 1979.

[20] Rebolledo, G.
A Multirate Voice Digitizer Based Upon Vector Quantization.
IEEE Trans. on Communications COM-30, April, 1982.

[21] Tao, B., et. al.
Hardware Realization of Waveform Vector Quantizers.
IEEE Journal on Selected Areas in Communications SAC-2, March, 1984.

[22] Wong, D., and Juang, B. H.
Voice Coding at 800 BPS and Lower Data Rates with LPC Vector Quantization.
Proc. of IEEE International Conference on Acoustics, Speech, and Signal Processing , May, 1982.

[23] Yamada, Y., Tazaki, S., and Gray, R.
Asymptotic Performance of Block Quantizers with Difference Distortion Measures.
IEEE Trans. on Information Theory IT-26, Jan., 1980.

[24] Yamaguchi, H.
Efficient Coding of Colored Pictures in R, G, B Components.
IEEE Trans. on Communications COM-32, Nov., 1984.