# 5.3 Fundamentals of Vector Quantization

Mohammad A. Khan
and Mark J. T. Smith
*Georgia Institute of Technology*

## 1 Introduction

In this age of information, we see an increasing trend toward the use of digital representations for audio, speech, images, and video. Much of this trend is being fueled by the exploding use of computers and multimedia computer applications. The high volume of data associated with digital signals, particularly digital images and video, has stimulated interest in algorithms for data compression. Many such algorithms are discussed elsewhere in this book. At the heart of all these algorithms is quantization, a field of study that has matured over the past few decades. In simplest terms, quantization is a mapping of a large set of values to a smaller set of values. The concept is illustrated in Fig. 1(a), which shows on the left a sequence of unquantized samples with amplitudes assumed to be of infinite precision, and on the right that same sequence quantized to integer values. Obviously, quantization is an irreversible process, since it involves discarding information. If it is done wisely, the error introduced by the process can be held to a minimum.

The generalization of this notion is called *vector quantization*, commonly denoted VQ. It too is a mapping from a large set to a smaller set, but it involves quantizing blocks of samples together. The conceptual notion of VQ is illustrated in Fig. 1(b). Blocks of samples, which we view as vectors, are represented by codevectors stored in a codebook — a process called encoding. The codebook is typically a table stored in a digital memory, where each table entry represents a different codevector. A block diagram of the encoder is shown in Fig. 2. The output of the encoder is a binary index that represents the compressed form of the input vector. The reconstruction process, which is called decoding, involves looking up the corresponding codevector in a duplicate copy of the codebook, assumed to be available at the decoder.

The general concept of VQ can be applied to any type of digital data. For a one-dimensional signal as illustrated in Fig. 1(b), vectors can be formed by extracting contiguous blocks from the sequence. For two-dimensional signals (i.e., digital images) vectors can be formed by taking 2-D blocks, such as rectangular blocks, and unwrapping them to form vectors. Similarly, the same idea can be applied to 3-D data (i.e., video), color and multispectral data, transform coefficients, and so on.

## 2 Theory of Vector Quantization

Although conceptually simple, there are a number of issues associated with VQ that are technically complex and relevant for an in-depth understanding of the process. To address these issues, such as design and optimality, it is useful to treat VQ in a mathematical framework.

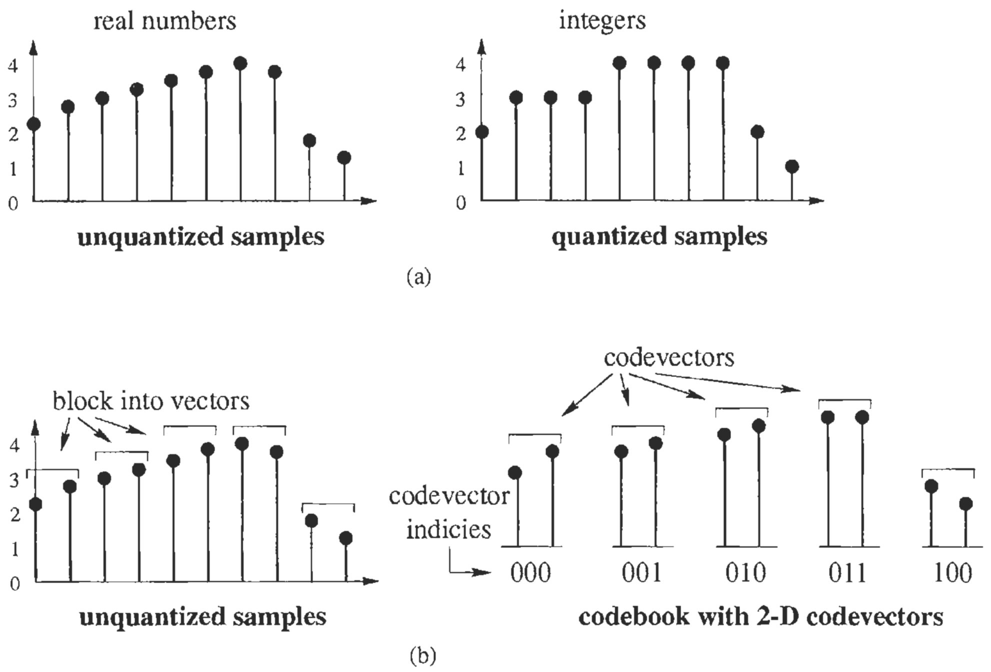

**FIGURE 1** Illustration of (a) scalar and (b) vector quantization.

Toward this end, we can view VQ as two distinct operations—encoding and decoding—shown explicitly in Fig. 2. The encoder $\mathcal{E}$ performs a mapping from $k$-dimensional space $\mathcal{R}^k$ to the index set $\mathcal{I}$, and the decoder $\mathcal{D}$ maps the index set $\mathcal{I}$ into the finite subset $C$, which is the codebook. The codebook has a positive integer number of codevectors that defines the codebook size. In this chapter, we will use $N$ to denote the codebook size and $\mathbf{y}_i$ to denote the codevectors, which are the elements of $C$. The bit rate $R$ associated with the VQ depends on $N$ (the number of codevectors in the codebook) and the vector dimension $k$. Since the bit rate is the number of bits per sample,

$$R = (\log_2 N)/k. \tag{1}$$

It is interesting to note that for VQ it is natural to have fractional bit rates such as $1/2$, $3/4$, $16/3$, etc. in contravention to basic

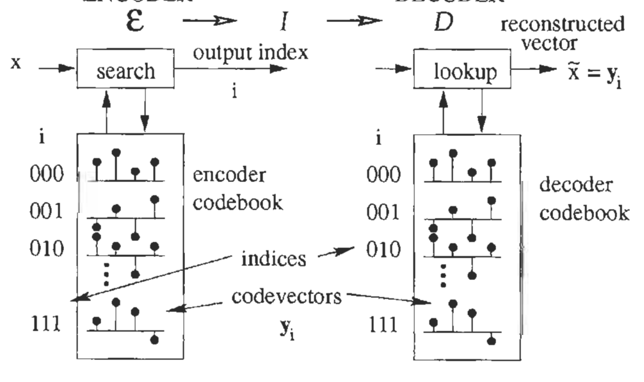

**FIGURE 2** Block diagram of a VQ encoder and decoder.

(fixed-rate) scalar quantization, in which noninteger rates do not arise naturally.

The operation associated with the decoder is extremely simple, involving no arithmetic at all. Conversely, the encoding procedure is complex, because a best matching vector decision must be made from among many candidate codevectors. To select a best matching codevector, we employ a numerically computable distortion measure $d(\mathbf{x}, \mathbf{y}_i)$, where low values of $d(\cdot, \cdot)$ imply a good match. There are many distortion measures that can be considered for quantifying the "quality of match" between two vectors $\mathbf{x}$ and $\mathbf{y}$, the most common of which is the *squared error* given by

$$d(\mathbf{x}, \mathbf{y}) = (\mathbf{x} - \mathbf{y})^t(\mathbf{x} - \mathbf{y}) = \sum_{\ell=1}^{k}(x[\ell] - y[\ell])^2,$$

where $x[\ell]$ and $y[\ell]$ are the elements of the vectors $\mathbf{x}$ and $\mathbf{y}$, respectively. For a vector $\mathbf{x}$ to be encoded, distortions are computed between it and each codevector $\mathbf{y}_i$ in the codebook. The codevector producing the smallest distortion is selected as the best match and the index associated with that codevector is used for the representation.

This process of encoding has an interesting and useful interpretation in the $k$-dimensional space. The set of codevectors defines a *partition* of $\mathcal{R}^k$ into $N$ cells $V_i$, where $i = 1, 2, \ldots, N$. If we let $\mathcal{Q}(\cdot)$ represent the encoding operator, then the $i$th cell is defined by

$$V_i = \{\mathbf{x} \in \mathcal{R}^k : \mathcal{Q}(\mathbf{x}) = \mathbf{y}_i\}. \tag{2}$$

Partitions of this type that are formed uniquely from the codebook and a nearest neighbor distortion metric such as the

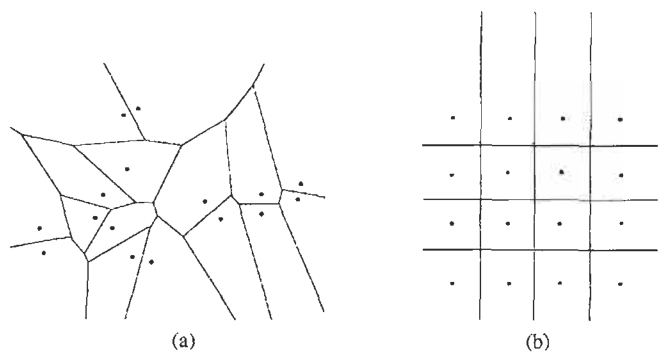

**FIGURE 3** Illustration of the partition cells associated with VQ and scalar quantization: (a) partition cells for a 2-D VQ; (b) partition cells corresponding to scalar quantization.

squared error distortion are called Voronoi partitions. The notion of partitioning can be visualized easily in two dimensions, an illustration of which is shown in Fig. 3(a). Here, each vector has two elements $(x_1, x_2)$ and consequently is a point in the two-dimensional space. That is, both the input vectors and codevectors are points in this space. The encoding procedure defines a unique partitioning of the space as shown in the figure, where the black dots denote codevectors.

The quality of performance of a VQ is typically measured by its average distortion for a given input source. In practice, sources are typically signal samples, image pixels, or some other data output associated with a signal that is being compressed. Whatever the source, average distortion measures are typically used to quantify the performance of a vector quantizer; the smaller the average distortion, the better the performance.

Vector quantizers are of interest because their performance is better than that of scalar quantization. The inherent advantage of VQ over scalar quantization can be understood through the concept of partitioning. Consider a fictitious input source for which we have designed an optimal 2-D codebook. Further, assume that the codevectors are those shown in Fig. 3(a). Observe that there are 16 2-D codevectors, implying a bit rate of 4 bits/vector, which is equivalent to 2 bits/sample. An optimal VQ design allows these codevectors to be positioned according to the statistical distribution of the input source vectors. Said another way, the codevectors are positioned to minimize the average distortion $\mathcal{D} = E\{(\mathbf{x}, \mathbf{y})\}$, where $\mathbf{x}$ and $\mathbf{y}$ are viewed as random vectors and $E$ denotes the *expected value.* Assume the codevectors shown in Fig. 3(a) as black dots represent an optimal VQ for the input source in question. The associated partitions shown in the figure illustrate the diversity of cell shapes and sizes that VQ can realize.

Now consider quantizing our fictional input source with scalar quantization at an equivalent bit rate of 2 bits/sample. Two bits gives us four levels we can use to quantize the $x_1$ and $x_2$ axes. The cells implied by using a scalar quantizer for the input source are shown in Fig. 3(b). Notice that we have exactly 16 cells but that each cell is constrained to be rectangular. Moreover, scalar quantization imposes a structure that forces some cells to be placed in regions in the space where the input source may not be significantly populated. These observations lead to two immediately recognizable advantages of VQ over scalar quantization for the general $k$-dimensional case. First, VQ provides greater freedom to control the shapes of the cells to achieve more efficient tilings of the $k$-dimensional space. This property is often called *cell shape* gain. Second, VQ allows a greater number of cells to be concentrated in the $k$-dimensional regions where the source has the greatest density, which reduces the average distortion. Structural constraints associated with the scalar quantizer prevent it from capturing this property of the input. In general terms, because VQ operates on blocks of samples, it is able to exploit inherent statistical dependencies (both linear and nonlinear) within the blocks. The resulting gains in efficiency improve with higher vector dimension.

## 3 Design of Vector Quantizers

The key element in designing a VQ is determining the codebook for a given input source. In practice, the input source is represented by a large set of representative vectors called a training set. Over the years, there have been many algorithms proposed for VQ design. The most widely cited is the classical iterative method attributed to Linde, Buzo, and Gray, known as the LBG algorithm [2]. The LBG algorithm is fashioned around certain necessary conditions associated with the distinct encoder and decoder operations implicit in VQ. The first of these conditions states that for a fixed decoder codebook, an optimal encoder partition of $\mathcal{R}^k$ is the one that satisfies the *nearest neighbor rule,* which says that we map each input vector to the cell $V_i$ producing the smallest distortion. By that measure, we are selecting the codevector that is nearest to the input vector.

The second optimality condition is the *centroid condition.* It states that for a given encoder partition cell, the optimal decoder codeword is the centroid of that cell, where the centroid of cell $V_i$ is the vector $\mathbf{y}^*$ that minimizes $E\{d(\mathbf{x}, \mathbf{y})|\mathbf{x} \in V_i\}$, the average distortion in that cell. The centroid is a function of the distortion measure and is different for different distortion measures. For the popular squared error distortion, the centroid is simply the arithmetic average of the vectors in cell $V_i$, i.e.,

$$\mathbf{y}^* = \frac{1}{\|V_i\|} \sum_{\ell \in V_i} \mathbf{x}_\ell,$$

where $\|V_i\|$ denotes the number of vectors in cell $V_i$.

It can be shown that local optimality can be guaranteed by upholding these conditions, subject to some mild restrictions [1].

### 3.1 The LBG Design Algorithm

The necessary conditions for optimality provide the basis for the classical LBG VQ design algorithm. The LBG algorithm is a generalization of the scalar quantization design algorithm introduced by Lloyd, and hence it is also often called the generalized Lloyd algorithm, or GLA. Interestingly, this algorithm was known earlier in the pattern recognition community as the $k$-means algorithm.

The steps of the LBG algorithm for the design of an $N$ vector codebook are straightforward and intuitive. Starting with a large training set (much larger than $N$), one first selects $N$ initial codevectors. Initial codevectors can be selected randomly from the training set. There are two basic steps in the algorithm: encoding of the training vectors, and computation of the centroids. To begin, we first encode all the training vectors using the initial codebook. This process assigns a subset of the training vectors to each cell defined by the initial codevectors. Next, the centroid is computed for each cell. The centroids are then used to form an updated codebook. The process then repeats iteratively with a recoding of the training vectors and a new computation of the centroids to update the codebook. Ideally, at each iteration, the average distortion is reduced until convergence.

In practice, convergence is often slow near the point of convergence. Hence, in the interest of time, one often terminates the iterative algorithm when the codebook is very close to the local optimum. There are many stopping criteria that can be considered for this purpose. One approach in particular is to compute the average distortion $\mathcal{D}^{(\ell)}$ between the training vectors and the codevectors periodically during the design process, where the superscript $\ell$ denotes the $\ell$th iteration. If the normalized difference in distortion from one iteration to the next falls below a prespecified threshold, the design process can be terminated. For example, one could evaluate $\mathcal{D}$ at each iteration and compute the normalized difference,

$$\frac{(\mathcal{D}^{(\ell)} - \mathcal{D}^{(\ell-1)})}{\mathcal{D}^{(\ell)}},$$

where forced termination is imposed when this normalized difference becomes less than the stopping threshold.

Often convergence proceeds smoothly. On occasion, the encoding stage of a given iteration may result in one or more cells' not being populated by any of the training vectors. This situation, known as the "empty cell" problem, effectively reduces the codebook size against our wishes. This condition when detected can be addressed in any one of a number of ways, one in particular consisting of splitting the cell with the greatest population in two to replace the lost empty cell.

## 3.2 Other Design Methods

Many methods of VQ design have appeared in the literature in recent years. Some focus on finding a good initial set of codevectors, which are then passed on to a classical LBG algorithm. By starting with a good initial codebook, one not only converges to a good solution, but generally converges in fewer iterations. Randomly selecting the initial codevectors from the training set is the easiest approach. This approach often works well, but sometimes it does not provide sufficient diversity to achieve a good locally optimal codebook. A simple variation that can be effective for certain sources is to select the $N$ vectors (as initial codevectors) from the training set that are farthest apart in terms of the distortion measure. This tends to assure that the initial codevectors are widely distributed in the $k$-dimensional space.

Alternatively, one can apply the splitting algorithm, which is a data dependent approach that systematically grows the initial codebook. The method, introduced in the original paper by Linde *et al.* [2], starts with a codebook consisting of the entire training set. First the centroid of the training set is computed. This centroid is then split into two codewords by perturbing the elements of the centroid. For instance, this could be done by adding some small value epsilon to each element. The original centroid and the perturbed centroid are used to encode the training set, after which centroids are computed to form a new initial codebook. These new centroids can then be perturbed and used to encode the training set. After centroids are computed, we have four codevectors in the codebook. The process can be repeated until $N$ codevectors are obtained. At this point, the LBG algorithm can be applied as described earlier.

These approaches are intended primarily as a way to obtain initial codebooks for the LBG algorithms. Other methods have been proposed that attempt to find good codebooks directly, which may be optimized further by the LBG algorithm if so desired. One such algorithm in particular is the pairwise nearest neighbor, or PNN, algorithm [3]. In the PNN algorithm, we start with the training set and systematically merge vectors together until we arrive at a codebook of size $N$. The idea is to identify pairs of vectors that are closest together in terms of the distortion measure, and replace these two vectors with their mean, which reduces the codebook size at each stage. The PNN algorithm effectively merges those partitions that would result in the smallest increase in distortion.

The task of finding partitions to be merged is computationally demanding. In order to avoid this, a fast PNN method was developed that does not attempt to find the absolute smallest cost at each step. The interested reader is referred to the original paper by Equitz [3] for details. Codebooks designed by the PNN algorithm can be used directly for VQ or as initial codebooks for the LBG algorithm. It has been observed that using the PNN algorithm as a front end to the LBG algorithm (i.e., in lieu of the random selection or splitting methods) can lead to better locally optimal solutions. It is impossible to discuss all the design algorithms that have been proposed. However, it is appropriate to mention a few others in closing this section. There are a number of modifications to the LBG algorithm that can lead to an order of magnitude speedup in design time. One approach involves transforming all the training vectors into the discrete cosine transform domain and performing the VQ design in that domain. Because many of the transform coefficients are close to zero and hence can be neglected, codebook design can be performed effectively with lower dimensional vectors. Although there is overhead associated with performing the transform, it is offset by the efficiency concomitant with the design in a reduced dimensional space.

Neural nets have also been considered for VQ design. A number of researchers have successfully used neural nets to generate

VQ codebooks [5, 6]. Neural net algorithms can have advantages over the classical LBG algorithm, such as less sensitivity to the initialization of the codebook, better rate-distortion performance, and faster convergence.

The ultimate design algorithm is one that finds the global optimal. Several attempts at this have been reported, such as design by simulated annealing, by stochastic relaxation, and by genetic algorithms [7–10]. Algorithms of this type are perhaps the best in terms of performance, but they tend to have a very high computational complexity. Interestingly, amid all of these choices, the LBG algorithm still remains one of the most popular.

## 4 VQ Implementations

VQ is attractive because it has a performance advantage over scalar quantization. However, like all things in life, quality comes with a price. For VQ, that price comes in the form of increased encoder complexity and codebook memory. The number of codevectors that must be stored in a codebook grows exponentially with increasing bit rate. For example, a 16-dimensional VQ at a rate of 0.25 bits/sample requires a codebook of size 16, while the same VQ at a rate of 1 bit/sample requires 65,536 codevectors. Codebook memory also grows exponentially with vector dimension. For example, an eight-dimensional VQ at a rate of 1 bit/sample (with 1 byte codevector elements) would occupy 2048 bytes. Increasing the dimension to 32 causes the memory storage requirement to jump to over 34 Gbytes. Similarly, the same kind of exponential dependence exists for encoder complexity. Unlike scalar quantization, careful attention should be given to dimension and rate, because memory and complexity requirements can easily become prohibitively large. As a general rule, VQs that are employed in practice have a dimension of 16 or less, because complexity, memory, and performance tradeoffs are generally most attractive in this range.

A host of fast search methods have been reported for VQ that can be grouped into two general types. The first can be called *fast optimal search* methods, which are optimal in the sense that they guarantee that the encoder will find the best matching codevector for each input vector [1, 11, 12].

One of the simplest methods of this type is known as the *partial distortion* method. Consider the VQ encoder in which in the conventional paradigm the input vector $\mathbf{x}$ is compared to each of the codevectors by explicit computation of $d(\mathbf{x}, \mathbf{y}_i)$ for $i = 1, 2, \ldots, N$. The partial distortion method involves keeping track of the lowest distortion calculation to date as the codebook is being searched. To understand how complexity is reduced, assume that we have searched $N/4$ of the codevectors in the codebook and that the minimum distortion found thus far is $D[\min]$. For the next distortion calculation, we compute

$$D[i] = \sum_{\ell=1}^{k} (x[\ell] - y_i[\ell])^2,$$

where $x[\ell]$ and $y_i[\ell]$ are the elements of $\mathbf{x}$ and $\mathbf{y}_i$, respectively. If during the process of evaluating the summation above, the value of $D[i]$ exceeds $D[\min]$, then we can terminate the calculation since we know that this vector is no longer a candidate. The net result of applying this procedure for encoding is that many of the vectors will be eliminated from further consideration prior to the full evaluation of the distortion calculation.

In addition, the triangle inequality can be used to reduce complexity, the idea being to use some reference points from which the distance to each code vector is precomputed and stored. The encoder then computes only the distance between the input vector and each reference point. Using these less complex comparisons in conjunction with precomputed data, one can achieve a reduction in complexity. The speed improvement realized by techniques of this type are clearly dependent on the codebook and input source; however, in general, one can expect a modest speedup.

Although every little bit helps, the complexity gains realized by optimal fast search algorithms fall short of addressing the exponential complexity growth associated with VQ. In this regard, efficient structured VQ encoding algorithms are attractive.

## 5 Structured VQ

A class of time-efficient methods has been studied extensively that sacrifice performance for substantial improvement in speed. The approach taken is to impose efficient structural constraints on the VQ codebook. These constraints are often formulated to make encoding complexity and/or memory *linearly* or *quadratically* dependent on the rate and dimension rather than *exponentially* dependent. The price paid, however, is usually inferior performance for the same rate and dimension. Nonetheless, the substantial reduction in complexity usually more than offsets the degradation in performance. To begin, we consider the most popular structured VQ of this class, tree-structured vector quantization (TSVQ).

### 5.1 Tree-Structured VQ

TSVQ consists of a hierarchical arrangement of codevectors, which allows the codebook to be searched efficiently. It has the property that search time grows linearly with rate instead of exponentially. Binary trees are often used for TSVQ because they are among the most efficient in terms of complexity. The concept of TSVQ can be illustrated by examining the binary tree shown in Fig. 4. As shown, the TSVQ has a root node at the top of the tree with many paths leading from it to the bottom. The codevectors of the tree,

$$\mathbf{y}_{000}, \mathbf{y}_{001}, \ldots, \mathbf{y}_{111},$$

are represented by the nodes at the bottom. The search path to reach any node (i.e., to find a codevector) is shown explicitly in the tree. In our particular example there are $N = 8$ codebook vectors and $N = 8$ paths in the tree, each leading to a different

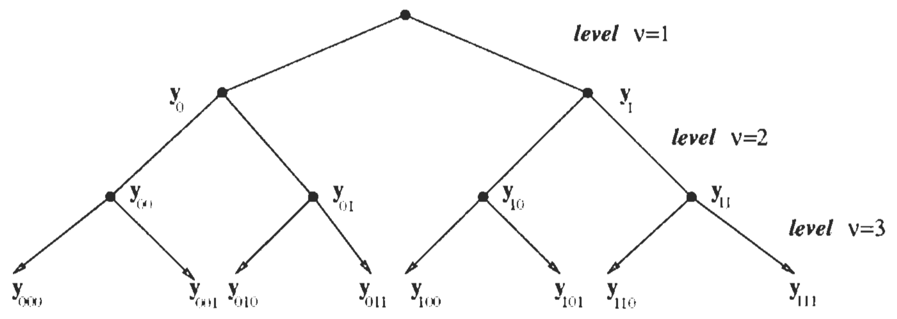

**FIGURE 4** TSVQ diagram showing a three-level balance binary tree.

codevector. To encode an input vector $\mathbf{x}$, we start at the top and move to the bottom of the tree. During that process, we encounter $\nu = 3$ (or $\log_2 N$) decision points (one at each level). The first decision (at level $\nu = 1$) is to determine whether $\mathbf{x}$ is closer to vector $\mathbf{y}_0$ or $\mathbf{y}_1$ by performing a distortion calculation. After a decision is made at the first level, the same procedure is repeated for the next levels until we have identified the codeword at the bottom of the tree. For a binary tree, it is apparent that $N = 2^\nu$, which means that for a codebook of size $N$, only $\log_2 N$ decisions have to be made. As presented, this implies the computation of two vector distortion calculations, $d(\cdot, \cdot)$, for each level, which results in only $2 \log_2 N$ distortion calculations per input vector.

Alternatively, one can perform the decision calculation explicitly in terms of hyperplane partitioning between the intermediate codevectors. The form of this calculation is the inner product between the hyperplane vector and input vector, where the sign of the output ($+$ or $-$) determines selection of either the right or left branch in the tree at that node. Implementated this way, only $\log_2 N$ distortion calculations are needed.

For the eight-vector TSVQ example above, this results in three instead of eight vector distortion calculations. For a larger (more realistic) codebook of size $N = 256$, the disparity is eight versus 256, which is quite significant.

TSVQ is a popular example of a constrained quantizer that allows implementation speed to be traded for increased memory and a small loss in performance. In many coding applications, such tradeoffs are often attractive.

## 5.2 Mean-Removed VQ

Mean-removed VQ is another popular example of a structured quantizer that leads to memory-complexity-performance tradeoffs that are often attractive in practice. It is a method for effectively reducing the codebook size by extracting the variation among vectors due specifically to the variation in the mean and coding that extracted component separately as a scalar. The motivation for this approach can be seen by recognizing that a codebook may have many similar vectors differing only in their mean values.

A functional block diagram of mean-removed VQ is shown in Fig. 5. First the mean of the input vector is computed and quantized with conventional scalar quantization. Then the mean-removed input vector is vector quantized in the conventional way by using a VQ that was designed with mean-removed training vectors. The outputs of the overall system are the VQ codewords and the mean values.

At the decoder, the mean-removed vectors are obtained by table loopup. These vectors are then added to a unit amplitude vector scaled by the mean, which in turn restores the mean to the mean-removed vector. This approach is really a hybrid of scalar quantization and VQ. The mean values, which are scalar quantized, effectively reduce the size of the VQ, making the overall system less memory and computation intensive.

One can represent the system as being a conventional VQ with codebook vectors consisting of all possible codewords obtainable by inserting the means in the mean-removed vectors. This representation is generally called a super codebook. The size of such a super codebook is potentially very large, but clearly it is also very constrained. Thus, better performance can always be achieved, in general, by using a conventional unconstrained codebook of the same size instead. However, since memory and complexity demands are often costly, mean-removed VQ is attractive.

## 5.3 Gain-Shape Vector Quantization

Gain-shape VQ is very similar to mean-removed VQ, but it involves extracting a gain term as the scalar component instead of a mean term. Specifically, the input vectors are decomposed

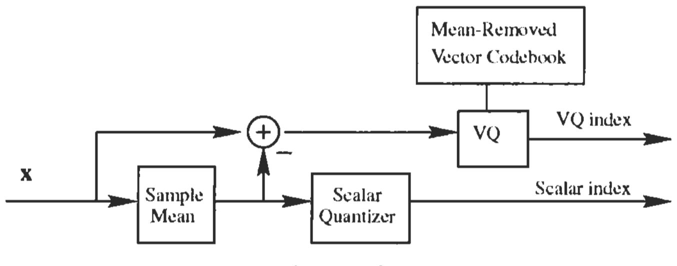

**FIGURE 5** Block diagram of mean-removed VQ.

into a scalar *gain* term and a gain normalized vector term, which is commonly called the *shape*. The gain value is the Euclidean norm given by

$$g = \|\mathbf{x}\| = \sqrt{\sum_{i=1}^{k} x^2[i]}, \tag{3}$$

and the shape vector **S** is given by

$$\mathbf{S} = \frac{\mathbf{x}}{g}. \tag{4}$$

The gain term is quantized with a scalar quantizer, whereas the shape vectors are represented by a shape codebook designed specifically for the gain normalized shape vectors.

Perhaps not surprisingly, gain-shape VQ and mean-removed VQ can be combined effectively together to capture the complexity and memory reduction gains of both. Similarly, the implicit VQ could be designed as a TSVQ to achieve further complexity reduction if so desired.

To illustrate the performance of VQ in a printed medium such as a book, we find it convenient to use image coding as our application. Comparative examples are shown in Fig. 6. The image in Fig. 6(a) is an original eight bit/pixel 256 × 256 monochrome image. The image next to it is the same image coded with convention unstructured 4 × 4 VQ at a rate of 0.25 bits/pixel. The images on the bottom are results of the same coding using mean-removed and gain-shape VQ. From the example, one can observe distortion in all cases at this bit rate. The quality, however, for the unconstrained VQ case is better than that of the structured

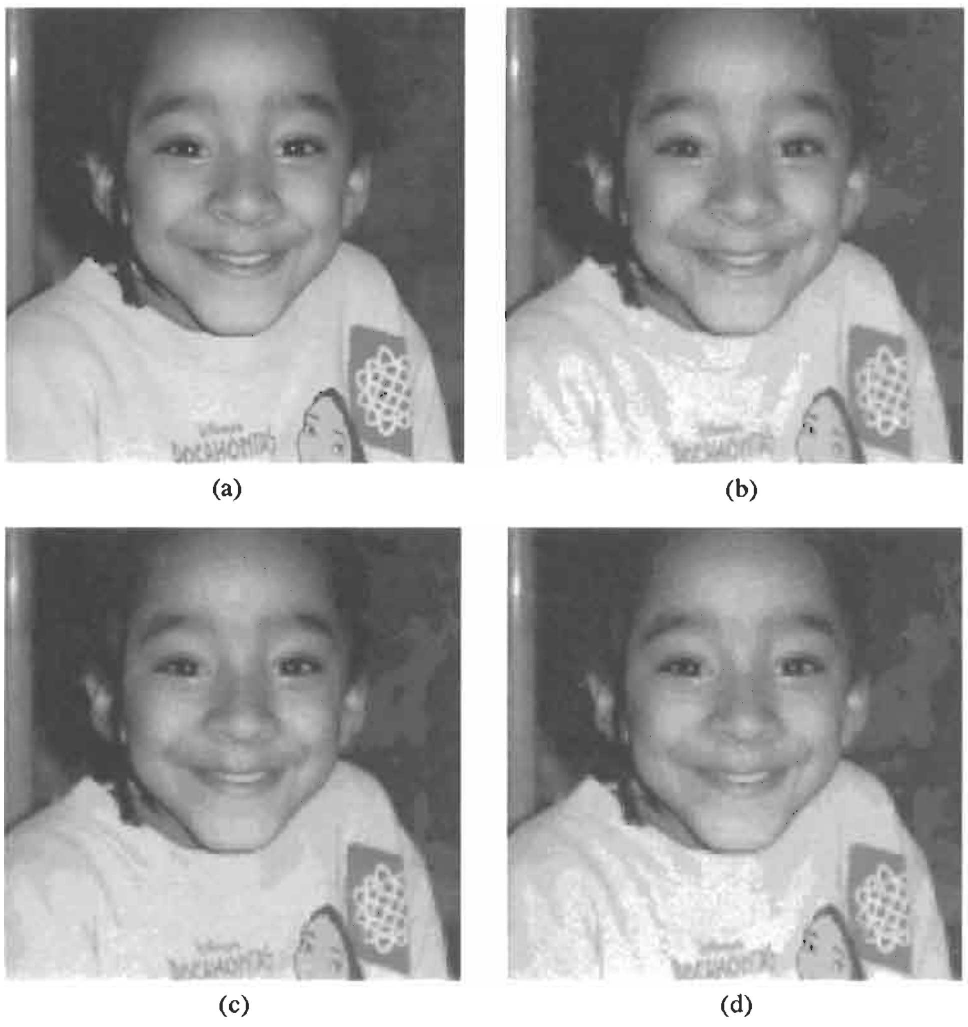

**FIGURE 6** Comparative illustration of images coded using conventional VQ, mean-extraction VQ, and gain-shape VQ: (a) original image 256 × 256, Jennifer; (b) coded with VQ at 0.25 bpp (PSNR, 31.4 dB); (c) coded with mean-extraction VQ at 0.25 dB (PSNR = 30.85 dB); (d) coded with gain-shape VQ at 0.25 dB (PSNR = 30.56 dB). All coded images were coded at 0.25 bits/pixel using 4 × 4 vector blocks.

VQs in Fig. 6(c) and Fig. 6(d), both subjectively and in terms of the signal-to-noise ratio (SNR). For quantitative assessment of the quality, we can consider the peak SNR (PSNR) defined as

$$\text{PSNR} = 10\log_{10}\left(\frac{255^2}{1/(N_1N_2)\sum_{n_1=1}^{N_1}\sum_{n_2=1}^{N_2}(x[n_1,n_2]-\hat{x}[n_1,n_2])^2}\right). \tag{5}$$

Although the PSNR can be faulted easily as a good objective measure of quality, it can be useful if used with care. PSNRs are quoted in the examples shown, and they confirm the quality advantage of unconstrained VQ over the structured methods. However, the structured VQs have significantly reduced complexity.

## 5.4 Multistage Vector Quantization

A technique that has proven to be valuable for storage and complexity reduction is multistage VQ. This technique is also referred to as *residual* VQ, or RVQ. Multistage VQ divides the encoding task into a sequence of cascaded stages. The first stage performs a first-level approximation of the input vector. The approximation is refined by the second-level approximation that occurs in the second stage, and then is refined again in the third stage, and so on. The series of approximations or successive refinements is achieved by taking stage vector input and subtracting the coded vector from it, producing a residual vector. Thus, multistage VQ is simply a cascade of stage VQs that operate on stage residual vectors. At each stage, additional bits are needed to specify the new stage vector. At the same time, the quality of the representation is improved. A block diagram of a residual VQ is shown in Fig. 7.

Codebook design for multistage VQ can be performed in stages. First the original training set can be used to design the first-stage codebook. Residual vectors can then be computed for the training set using that codebook. The next stage codebook can then be designed using the first-stage residual training vectors, and so on until all stage codebooks are designed. This design approach is simple conceptually, but suboptimal. Improvement in performance can be achieved by designing the residual codebooks jointly as described in [13, 14].

The most dramatic advantage of residual VQ comes from its savings in memory and complexity, which for large VQs can be orders of magnitude less than that of the unconstrained VQ counterpart. In addition, residual VQ has the property that it allows the bit rate to be controlled simply by specifying the number of VQ stage indices to be transmitted.

## 6 Variable-Rate Vector Quantization

The basic form of VQ alluded to thus far is more precisely called fixed rate VQ. That is, codevectors are represented by binary indices all with the same length. For practical data compression applications, we often desire variable rate coding, which allows statistical properties of the input to be exploited to further enhance the compression efficiency. Variable rate coding schemes of this type (entropy coders, an example of which is a Huffman coder) are based on the notion that codevectors that are selected infrequently on average are assigned longer indices, while codevectors that are used frequently are assigned short-length indices. Making the index assignments in this way (which is called entropy coding) results in a lower average bit rate in general and thus makes coding more efficient. Entropy coding the codebook indices can be done in a straightforward way. One only needs estimates of the codevector probabilities $P(i)$. With these estimates, methods such as Huffman coding will assign to the $i$th index a codeword whose length $L_i$ is approximately $-\log_2 P(i)$ bits.

We can improve upon this approach by designing the VQ and the entropy coder together. This approach is called entropy-constrained VQ, or ECVQ. ECVQs can be designed by a modified LBG algorithm. Instead of finding the minimum distortion $d(\mathbf{x}, \mathbf{y}_i)$ in the LBG iteration, one finds the minimum modified distortion

$$J_i = d(\mathbf{x}, \mathbf{y}_i) + \lambda L_i,$$

where $L_i = -\log_2 P(i)$. Employing this modified distortion $J_i$, which is a Lagrangian cost function, effectively enacts a Lagrangian minimization that seeks the minimum weighted cost of quantization error and bits.

To achieve rate control flexibility, one can design a set of codebooks corresponding to a discrete set of $\lambda$s, which gives a set of VQs with a multiplicity of bit rates. The concept of ECVQ is powerful and can lead to performance gains in data compression systems. It can also be applied in conjunction with other structured VQs such as mean-removed VQ, gain-shape VQ, and residual VQ; the last of these is particularly interesting. Entropy constrained residual VQ, or EC-RVQ, and variations of it have proven to be among the most effective VQ methods for direct application to image compression. Schemes of this type involve the use of conditional probabilities in the entropy coding block

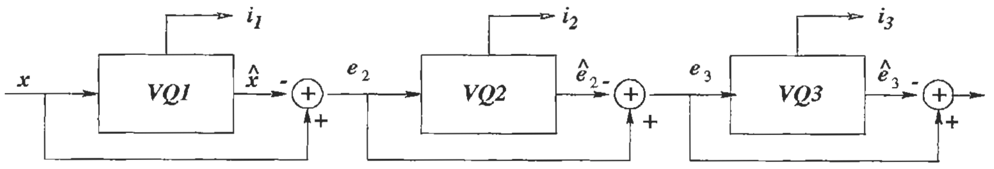

FIGURE 7 Block diagram of a residual VQ, also called multistage VQ.

where conditioning is performed on the previous stages and/or on adjacent stage vector blocks. Like ECVQ, the design is based on a Lagrangian cost function, but it is integrated into the RVQ design procedure. In the design algorithm reported in [14, 15], both the VQ stage codebooks and entropy coders are jointly optimized iteratively.

## 7 Closing Remarks

In the context of data compression, the concepts of optimality, partitioning, and distortion that we discussed are insightful and continue to inspire new contributions in the technical literature, particularly with respect to achieving useful tradeoffs among memory, complexity, and performance. Equally important are design methodologies and the use of variable length encoding for efficient compression. Although we have attempted to touch on the basics, the reader should be aware that the VQ topic area embodies much more than can be covered in a concise tutorial chapter. Thus, in closing, it is appropriate to at least mention several other classes of VQ that have received attention in recent years. First is the class of lattice VQs. Lattice VQs can be viewed as vector extensions of uniform quantizers, in the sense that the cells of a $k$- dimensional lattice VQ form a uniform tiling of the $k$-dimensional space. Searching such a codebook is highly efficient. The advantage achieved over scalar quantization is the ability of the lattice VQ to capture cell shape gain. The disadvantage, of course, is that cells are constrained to be uniform. Nonetheless, lattice VQ can be attractive in many practical systems.

Second is the general class of predictive VQs, which may include finite-state VQ (FSVQ), predictive VQ, vector predictive VQ, and several others. Some of these predictive approaches involve using neighboring vectors to define a state unambiguously at the encoder and decoder and then employing a specially designed codebook for that state. VQs of this type can exploit statistical dependencies (both linear and nonlinear) among adjacent vectors, but they have the disadvantage of being memory intensive.

Finally, it should be evident that VQ can be applied to virtually any lossy compression scheme. Prominent examples of this are transform VQ, in which the output of a linear block transform such as the DCT is quantized with VQ; and subband VQ, in which VQ is applied to the output of an analysis filter bank. The latter of these cases has proven to yield some of the best data compression algorithms currently known.

Interestingly, application of the principles of VQ extend far beyond our discussion. In addition to enabling construction of a wide variety of VQ data compression algorithms, VQ provides a useful framework in which one can explore fractal compression, motion estimation and motion compensated prediction, and automated classification. The inquisitive reader is challenged to explore this rich area of information theory in the literature [16].

## References

[1] A. Gersho and R. Gray, *Vector Quantization and Signal Compression* (Kluwer, Boston, 1992).

[2] Y. Linde, A. Buzo, and R. Gray, "An algorithm for vector quantizer design," *IEEE Trans. Commun.* **C-28**, 84–95 (1980).

[3] W. Equitz, "A new vector quantization clustering algorithm," *IEEE Trans. Acoust. Speech Signal Process.* **37**, 1568–1575 (1989).

[4] R. King and N. Nasrabadi, "Image coding using vector quantization in the transform domain," *Pattern Recog. Lett.* **1**, 323–329 (1983).

[5] N. Nasrabadi and Y. Feng, "Vector quantization of images based upon the kohonen self-organizing feature map," in *IEEE International Conference on Neural Networks* (San Diego, CA, 1988), Vol. 1, pp. 101–105.

[6] J. McAuliffe, L. Atlas, and C. Rivera, "A comparison of the LBG algorithm and kohonen neural network paradigm for image vector quantization," in *IEEE International Conference on Acoustics, Speech, and Signal Processing* (Albuquerque, NM, 1990), pp. 2293–2296.

[7] J. Vaisey and A. Gersho, "Simulated annealing and codebook design," in *IEEE International Conference on Acoustics, Speech and Signal Processing* (New York, 1988), pp. 1176–1179.

[8] K. Zeger and A. Gersho, "A stochastic relaxation algorithm for improved vector quantizer design," *Electron. Lett.* **25**, 896–898 (1989).

[9] K. Zeger, J. Vaisey, and A. Gersho, "Globally optimal vector quantizer design by stochastic relaxation," *IEEE Trans. Signal Process.* **40**, 310–322 (1992).

[10] K. Krishna, K. Ramakrishna, and M. Thathachar, "Vector quantization using genetic $k$-means algorithm for image compression," in *Proceedings of the 1997 International Conference on Information, Communication, and Signal Processing, Part 3* (1997), Vol. 3, pp. 1585–1587.

[11] M. Soleymani and S. Morgera, "An efficient nearest neighbor search method," *IEEE Trans. Commun.* **COM-35**, 677–679 (1987).

[12] D. Cheng, A. Gersho, B. Ramamurthi, and Y. Shoham, "Fast search algorithms for vector quantization and pattern matching," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing* (San Diego, CA, 1984), pp. 911.1–911.4

[13] C. Barnes, "Residual quantizers," Ph.D. thesis (Brigham Young University, Provo, UT, 1989).

[14] F. Kossentini, M. Smith, and C. Barnes, "Necessary conditions for the optimality of variable rate residual vector quantizers," *IEEE Trans. Inf. Theory*, November, **41**, 1903–1915 (1995).

[15] F. Kossentini, W. Chung, and M. Smith, "Conditional entropy-constrained residual VQ with application to image coding," *Trans. Image Process. Spec. Issue VQ*, February, **5**, 311–321 (1996).

[16] M. Barlaud, P. A. Chou, N. M. Nasrabadi, D. Neuhoff, M. Smith, and J. Woods, eds., *IEEE Transactions on Image Processing, Special Issue on Vector Quantization*, February 1996.