SIGNAL PROCESSING USING LUT FILTERS BASED ON HIERARCHICAL VQ

Ricardo L. de Queiroz[1] *and Patrick Fleckenstein*[2]

[1] Xerox Corporation, 800 Phillips Rd., 128-27E, Webster, NY, 14580 queiroz@wrc.xerox.com
[2] Rochester Institute of Technology, Center for Imaging Science, Rochester, NY 14623 pat@csh.rit.edu

ABSTRACT

Vector quantization (VQ) is a powerful tool in signal processing. Hierarchical VQ (HVQ) is a method to implement VQ completely based on look-up tables (LUT). In HVQ, both encoders and decoders are inherently simple and fast, since there are no searches over codebooks. We introduce an overlapped HVQ (OHVQ) method, in which the number of samples is preserved after each HVQ stage. After the last stage, each OHVQ code in a particular location in the signal will map to a block (vector) which approximates that neighbourhood in the original sequence. For this reason, OHVQ is used as a basis to create a LUT-based filter, i.e. a spatial signal processor with very fast implementation. Preliminary analysis and image processing examples are shown demonstrating the efficiency of the proposed method.

1. INTRODUCTION

Vector quantization (VQ) has been widely used within the signal processing community [1]. With VQ a finite sequence of samples (typically small) can be represented through one index which can be mapped to a reconstruction sequence. VQ is fundamental to information theory since it allows us to approximate rate-distortion theorectical bounds, at the expense of extra complexity [1]. Let us assume we segment a real sequence $\{x(n)\}$ into vectors $\mathbf{x}(k)$ of N samples each. There is a codebook X_C with M real N-tuple codewords $\mathbf{w}_0$ through $\mathbf{w}_{N-1}$. Each vector is then quantized into $\hat{\mathbf{x}}(k)$ by mapping $\mathbf{x}(k)$ to one of the $\mathbf{w}_i$. VQ can be used as a compression method by conveying to a receiver the index i for the current temporal index k so that the receiver can reconstruct $\hat{\mathbf{x}}(k) = \mathbf{w}_i$ as an approximation of $\mathbf{x}(k)$. The average rate to represent the VQ indices (quantized data) is a function of the data and of the coder. Without entropy coding, one needs $\lceil \log_2(M) \rceil$ bits/vector for representing the index. Using non-contextual but efficient entropy coders this number approaches the vectors' entropy $H(\mathbf{x})$. For contextual coders, the rate will depend on conditional entropies, but we will not get into details here. In practice, it is common to see that the original signal itself is scalar quantized to b bits per sample. Hence, the rate achieved by VQ is $H(\mathbf{x})/Nb$ for perfect non-contextual entropic coders. See [1] for an excellent overview and reference on VQ.

The major drawback of VQ is the encoder complexity. The task of matching $\mathbf{x}(n)$ to one of the $\mathbf{w}_i$ can be dauntingly prohibitive in certain applications. In the straightforward case, in order to find the codeword index to quantize the n-th vector, one computes the distance $d_i = d(\mathbf{x}(n), \mathbf{w}_i)$ for every codeword. In this case $d(,)$ is some distance measure, e.g. $d(\mathbf{a}, \mathbf{b}) = ||\mathbf{a} - \mathbf{b}||_2$. The index corresponding to the smallest distance is then selected. In fact, the codebook is designed by partitioning the N-dimensional space into M regions (commonly, but not necessarily, convex) and assigning a reconstruction vector $\mathbf{w}_i$ to each region, where $\mathbf{w}_i$ is meant to be a representative element for all vectors falling within the i-th region (e.g. the centroid of the sample space density function).

Simple, in theory, but the larger your codebook (larger M) the better the approximation quality given a proper codebook design method. Hence, one might want to increase M. However, as M increases, compression ratio decreases and computation cycles increase as more distances have to be evaluated. In an image compression example, using vectors of 16 samples (4×4 pixels), 8-bit images, 18-bit codes (i.e. $M = 2^{18} = 256$K), the compression ratio is about 7:1, without any entropy coding. However, this means that for every block of 16 image pixels, 256K distance measures between 18-tuples have to be computed, along with an equal number of comparisons, etc.. This is a very slow process for most applications. Most system designers opt to spend these CPU cycles somewhere else by using other efficient compression techniques.

2. HIERARCHICAL VECTOR QUANTIZATION

In hierarchical VQ (HVQ) [2]–[7], the codebook search is eliminated by applying a greedy divide-and-conquer approach. We first divide the input N-tuple into N_0 small sub blocks. Each of the N_0 sub blocks undergoes VQ, being mapped to a codeword. In a next stage, the N_0 codewords are broken into N_1 subblocks. Each subblock undergoes VQ again yielding N_1 codewords. The process is repeated until the K-th stage where one codeword is selected to represent the input vector. In order to be practical, we will limit ourselves to groups of 2 samples per sub-block so that N is made a power of two. The advantage of HVQ is its simplicity of implementation. Each entry (an input sample or a codeword) is represented in a reasonable number of bits B_n (e.g. 8 or 10), which are mapped to one of the $2^{B_{n+1}}$ codewords which are coded into B_{n+1} bits. It is clear that the process can be implemented using a LUT of $2^{2B_n} B_{n+1}$ bits. For example if $B_n = B_{n+1} = 8$, a 64KB LUT is sufficient. A comparison between the VQ and HVQ processes is depicted in Fig. 1, which shows a typical VQ system and its HVQ counterpart each encoding an 8-tuple of B_0-bit samples. The HVQ system uses 3 stages of LUTs to decompose the 8-tuple hierarchically.

Decoding is performed equally in both cases (VQ or HVQ). The key difference between HVQ and any other VQ method is the symmetry between encoders and decoders. While VQ

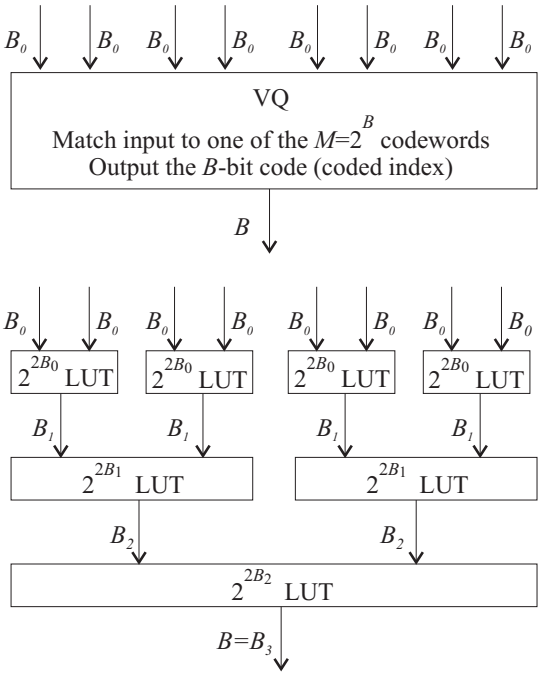


Figure 1. Top: example diagram of a VQ encoding system: 8 B_0-bit samples are encoded into one B-bit codeword. Bottom: the same encoding performed through 3 stages of HVQ. The number of bits at every HVQ stage is noted.

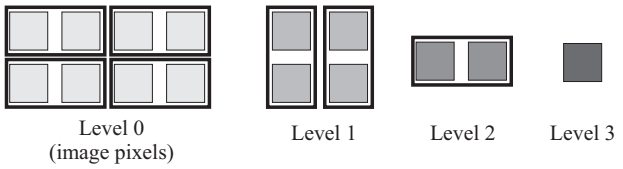


Figure 2. Illustration of 3 stages of HVQ applied to an image block of 2×4 pixels.

encoders are generally slower than transform coders, HVQ encoders are much faster. For image processing, HVQ is adapted to 2D by alternating horizontal and vertical groupings of code pairs as illustrated in Fig. 2.

We are not interested in HVQ as a compression means but as a means to approximate the input N-tuple. For that, the design of HVQ codebooks is important. The design is done in a similar way as in the VQ case, e.g. using LBG, with a few adaptations. In order to design the first stage of Fig. 1 we have to partition the 2D sample space into 2^{B_1} Voronoi regions for a given distribution of the training data. In the second stage, the 4D space is partitioned into 2^{B_2} regions. The peculiarity in this approach is that there are only 2^{B_1} points in the 4D space and not a continuous of data. In the design of the last stage, the N-dimensional ($N = 8$) space contains at most 2^{B_2} valid 8-tuples.

HVQ is a greedy approach to VQ. It trades quality for speed. HVQ is used here because the loss in quality is small compared to the huge gains in encoding speed. In terms of signal approximation quality, the main difference between VQ and HVQ design is that for every stage the codebook design algorithm cannot simply partition a k-dimensional space

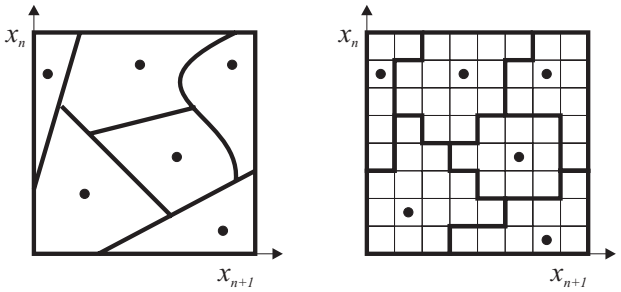


Figure 3. Illustration of quantization of a 2D space into 6 regions (left). Right: similar quantization of the 2D space, wherein the input samples have already been quantized.

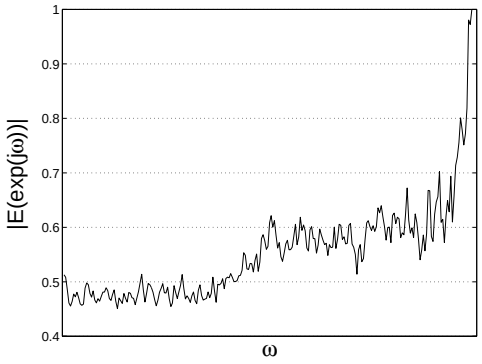


Figure 4. Error spectrum of an HVQ approximation of an image.

into 2^{B_n} regions because the input space has already been quantized. Although it is difficult to visualize the multimensional process we can trace a parallel to quantization of a simple 2D space, which is illustrated in Fig. 3. Figure 3 shows the partition of the 2D space into 6 regions. It also shows the case where the input variables have already been quantized so that the partition of the 2D space has to conform to the contours of the quantizing regions of the input data (compare the quantization regions between the two diagrams in Fig. 3). This is the main difference between HVQ and VQ: the input to one stage is already quantized into multidimensional cells or "pods". An HVQ stage can only divide the pods among the various quantization regions. The error spectrum associated to HVQ is commonly a coloured noise. A typical spectrum of the approximation error of HVQ applied to an image is shown in Fig. 4.

3. OVERLAPPED HVQ

As Fig. 1 shows, HVQ, like VQ, can make one single codeword index (code) to represent N samples. At the k-th vector, after the code c_k is found, the input is cycled by N samples and a new block of N samples is used to find a new code c_{k+1}, i.e.

$$\underbrace{x_0x_1x_2x_3}_{c_1}\underbrace{x_4x_5x_6x_7}_{c_2}\underbrace{x_8x_9x_{10}x_{11}}_{c_3}\underbrace{x_{12}x_{13}x_{14}x_{15}}_{c_4}$$

Hence, there are N times more samples than codes. We, however, introduce overlapped HVQ (OHVQ), in which the input is cycled by only one sample at a time. Hence, the vectors that are quantized actually overlap and the number of codes is the same as the number of samples.

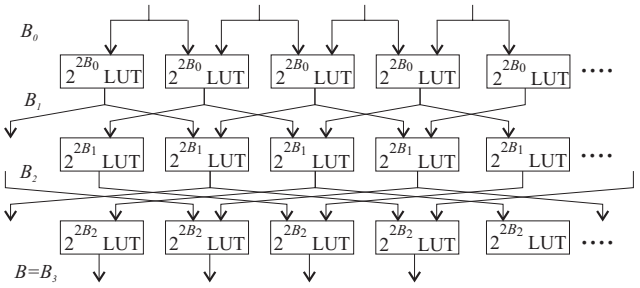
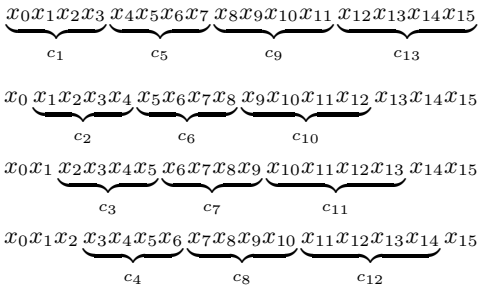


Figure 5. Implementation diagram of overlapped HVQ.



While overlapped VQ has been applied successfully to various problems, we are unaware of the use of OHVQ. One key feature of OHVQ is its simplicity of implementation. A diagram for the implementation of OHVQ is shown in Fig. 5 which should be compared to the corresponding HVQ diagram in Fig. 1. For k stages, OHVQ requires k look-ups per input sample.

4. THE LUT FILTER

In HVQ each output index represents a code vector that resembles the input N-tuple. In OHVQ each output index represents a code vector that resembles an N sample neighbourhood of a particular sample. In fact, each output OHVQ sample represents the neighbourhood of the respective input sample, furthermore one can see it as a one-to-one input-to-output system mapping each sample to its neighbourhood index. We propose to map the neighbourhood to an output sample, i.e. a spatial filter is applied to the neighbourhood. Actually, not to the real neighbourhood but to its approximation, which is accomplished by filtering the codevectors directly. Filtering is done off line and another LUT F can be used to map each code vector to the processed output. Hence, the proposed signal processing system's diagram is shown in Fig. 6. It is shown as an OHVQ-based sequence of LUTs T_n, wherein the coded index is mapped to the output through LUT F. Despite its usefulness in explaining the concept, it is obvious that F can be folded into T_{K-1}, so that the number of LUTs in the HVQ filter remains K.

The analysis of a non-linear system such as the one in Fig. 6 is a complicated task. The elements are LUTs which can be literally filled with anything. There are, however, a couple of simplifications that serve to illustrate the behaviour of the LUT filter. If the F_n were linear operators, one may hope that the system in Fig. 6(b) can implement general convolutions. However, it cannot. If $F_n(.)$ is a linear operator as $x_{out} = a_1x_{in1} + a_2x_{in2}$, then the filter's transfer function is of the type:

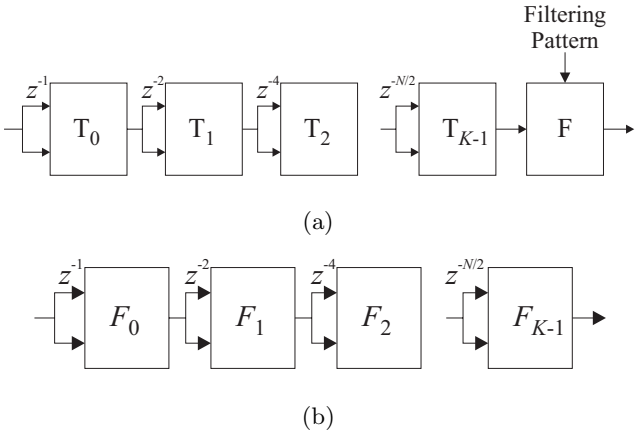


Figure 6. Implementation diagram of the proposed filter. (a) T_n are OHVQ stages and stage F is a LUT for the actual filter implementation which should be folded into T_{K-1}, so that the filter can be implemented in K LUT stages as in (b).

$$H(z) = A\prod_{i=0}^{K-1}\left(1+\beta_i z^{-2^i}\right),$$

which is obviously not general. For example for an order-3 filter, if we use 2 stages ($K = 2$) the resulting filter will be

$$H(z) = A\left(1+\beta_0 z^{-1}+\beta_1 z^{-2}+\beta_0\beta_1 z^{-3}\right)$$

which imposes a constraint on the third power. This is a restriction imposed by the delay chain and overal lattice format. In order to implement a broad number of systems, the use of HVQ tables in the LUT filter stages allows us to give up accuracy for generality. In effect, when HVQ is used, what is processed is a new sequence $z[n] = x[n] + e[n]$ in place of $x[n]$. In other words what is processed is a quantized input. If, for the sake of simplifying the analysis, we linearize the processing by assuming a linear filter $F(z)$, then the output signal is:

$$Y(z) = F(z)X(z) + F(z)E(z) = Y_d(z) + D(z),$$

wherein $Y_d(z)$ is the desired output and $D(z)$ is the disparity of a LUT filter as compared to a regular filter. One interesting fact to note is that $e[n]$ is made of features which were not captured in the codebook. For example, in images, $e[n]$ is expected to be a high-pass signal due to HVQ's inability to represent small details. For example, Fig. 4 shows a typical approximation error spectrum $|E(e^{j\omega})|$ computed for a test image. Thus, for this "blue-ish" noise error spectrum, if $F(z)$ represents a low-pass filter, $D(z)$ can become small, while if it represents a high-pass filter the processed image can present artifacts caused by the high frequencies of $D(z)$.

As a signal processing example, we processed images using the LUT filter. Figure 7 shows several image processing examples obtained using the approach in Fig. 6. In these tests we used standard HVQ tables (LBG design on completely independent test set) as the OHVQ stages. Four stages of 10-bit codes (1024 codewords) were used, so that each resulting code vector represents a 4×4 input neighbourhood. The original image is shown along with examples of blurring,

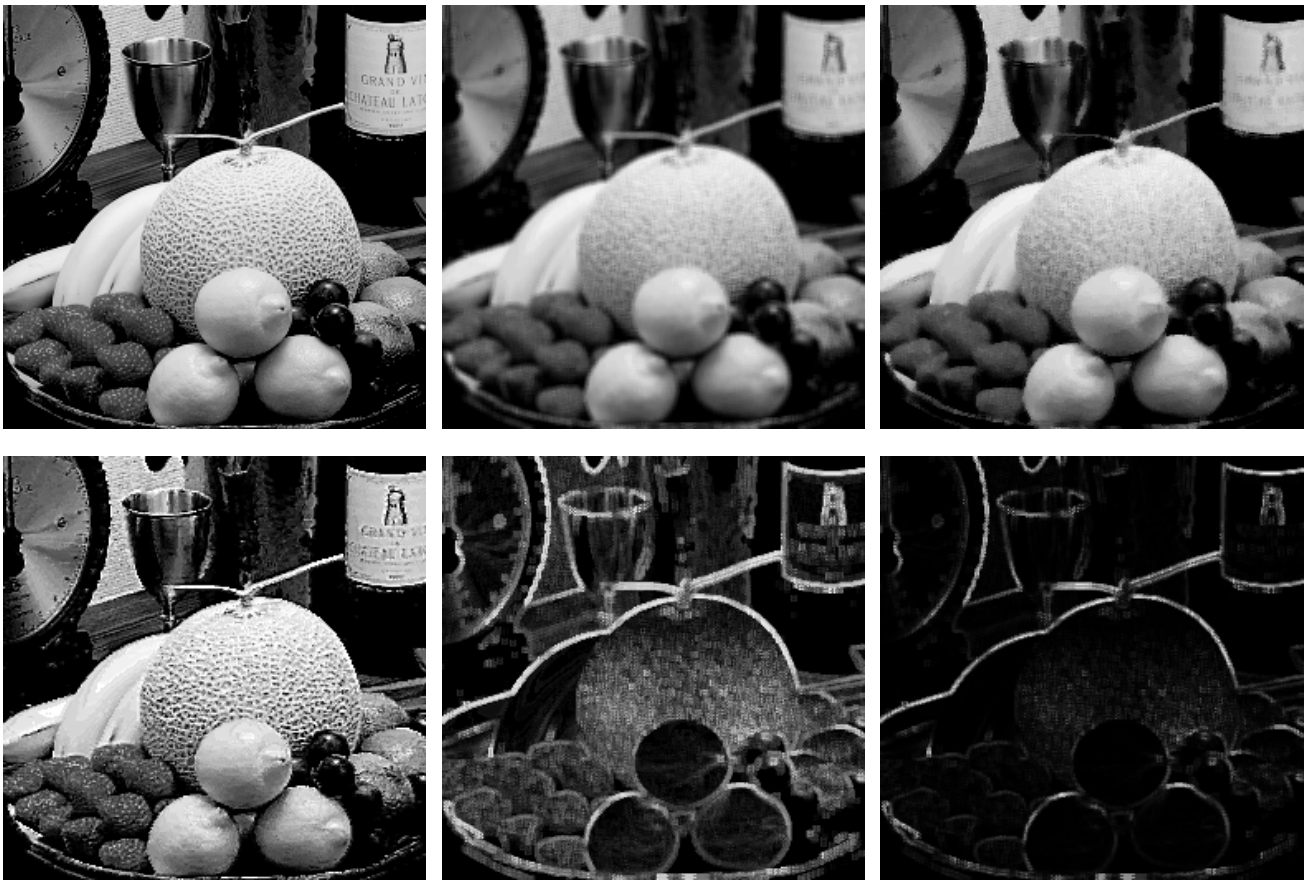


Figure 7. A four-stage HVQ-LUT filter applied to the (a) original image for implementing (b) mean, (c) median, and (d) sharpening filters. Within the same approach one can also compute local (e) dynamic range and (f) variance. (a)–(f) in top-down left-right order.

sharpening and median filtering the image, as well as computing the local dynamic range and the local variance. In line with our discussion on the spectrum of $D(z)$, one can see that the sharpening LUT filter performs worse than the the blurring one, as compared to the expected output of their traditional convolution-based counterparts.

5. CONCLUSION

The proposed HVQ-LUT filter is very fast and flexible. It is a generic process which can be applied to most spatial processing operations. Quality is limited to the approximation power of OHVQ. Complexity, however, is K look-ups per pixel regardless of the operation. This is faster than any comparable linear spatial filter and much faster than more complex operations such as computing local statistics. Other possible operations could be image classification and segmentation, resolved on a pixel by pixel basis.

This is an ongoing work. Because of the difficulty in the analysis of such a non-linear process, future work will concentrate on better analysis tools, on improving the codebook design and on redesigning the operator F. In the more immediate future, a better examimation of HVQ-LUT filtering of images will be published soon.

REFERENCES

[1] A. Gersho and R. Gray, *Vector Quantization and Signal Compression,* Kluwer Academic, Norwell, MA, 1992.

[2] P. C. Chang, J. May and R. Gray, "Hierarchical vector quantization with table look-up encoders," *Proc. Intl. Conf. Communications,* Chicago, IL, pp. 1452–1455, 1985.

[3] M. Vishwanath and P. Chou, "An efficient algorithm for hierarchical compression of video," *Proc. Intl. Conf. Image Processing,* Austin, TX, Vol. 3, pp. 275–279, 1994.

[4] N. Chadha, M. Vishwanath, and P. Chou, "Hierarchical vector quantization of perceptually weighted block transforms," *Proc. Data Compression Conference*, Snowbird, UT, March 1995.

[5] N. Chadha, M. Vishwanath, and P. Chou, "Constrained and recursive hierarchical table look-up vector quantization," *Proc. Data Compression Conference*, Snowbird, UT, March 1996.

[6] A.Aiyer and R. M. Gray, "A fast table look-up algorithm for classifying document images," *Proc. Intl. Conf. Image Processing,* Kobe, Japan, 1999.

[7] R. de Queiroz and P. Fleckenstein, " Very fast JPEG compression using hierarchical vector quantization ," *IEEE Signal Processing Letters,* Vol. 7, pp. 97-99, May 2000.