LUT Filters for Quantized Processing of Signals

Ricardo L. de Queiroz, *Senior Member, IEEE,* and Patrick A. Stein

***Abstract*—We introduce a method to perform filtering on approximations (quantized versions) of the input signal, which lends itself to a practical implementation solely based on look-up tables (LUTs). The LUT filter approximates the performance of some traditional nonlinear filters at a fraction of the cost. The filter is divided into an approximation stage that is constant for all filters and a filtering stage, which is one LUT that can change in order to implement different filters. We introduce an overlapped hierarchical vector quantization (OHVQ) scheme that is used as the approximation stage. The output is produced by mapping the OHVQ codes to filtered data. Hence, all processing is done via LUTs, even though the filter size needs to be small because of typical OHVQ contraints. Switching among filters demands changing pointers to only one small LUT. Preliminary analysis and image processing examples are shown, demonstrating the efficacy of the proposed method.**

***Index Terms*—Hierarchical vector quantization, look-up table processing, nonlinear filtering.**

I. Introduction

FILTERS are the most essential building blocks of signal processing. Some nonlinear and linear filters can be expensive to implement, mainly for signals such as image and video. The design of linear filters is a very mature field and nonlinear filters of several classes have also been very well studied. Instead of trying to incrementally improve the performance of filters, we are concerned with a structure that would allow fast implementation. More particularly, we are concerned with a class of filters that would be fast and cheap to implement while being versatile enough to be able to replace several linear and nonlinear filters. In most cases, the filter output does not need to be precise but has to meet requirements or simply accomplish tasks (e.g., find edges in images or to remove noise). Hence, in these cases, we need only to produce an approximate output and use this degree of freedom to simplify implementation.

The idea in this paper is illustrated in Fig. 1. Let H be a traditional filter that maps the input sequence $\{x[n]\}$ into the output sequence $\{y[n]\}$. If we process or quantize the input before H using some operator (quantizer) G, we will generate some sequence $\{x'[n]\}$. The filtered output is the sequence $\{y'[n]\}$ that differs from $\{y[n]\}$ unless G is an identity operator. If $\{x'[n]\}$ approximates $\{x[n]\}$, then $\{y'[n]\}$ likely approximates $\{y[n]\}$, but why would anyone insert such an operator? What good can it do? Well, it turns out that we can devise a fast and versatile filter structure if we are satisfied in processing an approximation of $\{x[n]\}$, as we will explain in detail through the paper.

Let H map a neighborhood of N samples around sample $x[n]$ to produce output sample $y[n]$. If this neighborhood is $\mathbf{x}_N[n]$, then $y[n] = H(\mathbf{x}_N[n])$. In our approach, the input $\{x[n]\}$ is mapped into the sequence $\{v[n]\}$, where each $v[n]$ is a map of $\mathbf{x}_N(n)$, i.e., $v[n] = T(\mathbf{x}_N[n])$. The key to this paper is to find T that can be approximately inverted: $\mathbf{x}'_N[n] = T^{-1}(v[n]) \approx \mathbf{x}_N[n]$. Note that $\mathbf{x}_N[n]$ is derived from $\{x[n]\}$, but even though $\mathbf{x}'_N[n] \approx \mathbf{x}_N[n]$, in order to produce a signal $\{x'[n]\}$ that approximates $\{x[n]\}$, one has to sample $\mathbf{x}'_N[n]$ to retrieve $x'[n]$. Nevertheless, we assume that one can approximate $\{x[n]\}$ via $\{v[n]\}$ or $\{x'[n]\}$. Since $v[n]$ can be used to approximate the input, the output can be generated by direct mapping (1-to-1): $y'[n] = F(v[n])$. With properly designed F, $y'[n]$ is near $y[n]$ if $\mathbf{x}'_N[n]$ is near $\mathbf{x}_N[n]$. Hence, $y[n]$ can be approximated with the proper choice of T and F. Furthermore, we would like to be able to efficiently implement both T and F.

Summarizing, the objectives of this paper are the following:

- to find a mapping T for a giving training set such that $v[n] = T(\mathbf{x}_N[n])$, which can be approximately inverted, i.e., $\mathbf{x}'_N[n] = T^{-1}(v[n])$ such that $\mathbf{x}'_N[n] \approx \mathbf{x}_N[n]$;
- to find a mapping F such that $y'[n] = F(v[n]) = H(\mathbf{x}'_N[n]) = H(T^{-1}(v[n]))$;
- to implement both F and T using only a few look-up tables (LUTs).

We use vector quantization (VQ) [1] to make the mapping T. Because straight VQ is slow, we actually employ hierarchical VQ (HVQ), which is very fast and will be reviewed in Section II. Its overlapped version is introduced in Section III and will be actually used as the operator T. The operator F is derived, and the filter as a whole is analyzed in Section IV. The conclusions of this paper are presented in Section V.

II. Review of Hierarchical VQ

VQ has been widely used within the signal processing community [1], particularly for image compression. Its major drawback is encoder complexity because one has to search and match vectors within large codebooks [1]. 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 subblocks. Each of the N_0 subblocks undergoes VQ, where each is 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 Kth stage, where one codeword is selected to represent the input vector. In order to be practical, we will limit ourselves to groups of two samples per subblock 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

Manuscript received March 21, 2002; revised May 14, 2003. The associate editor coordinating the review of this paper and approving it for publication was Dr. Masaaki Ikehara.

R. L. de Queiroz is with Universidade de Brasilia, Brasilia, Brazil (e-mail: queiroz@ieee.org).

P. A. Stein is with the Rochester Institute of Technology, Rochester, NY 14623-5603 USA (e-mail: pat@csh.rit.edu).

Digital Object Identifier 10.1109/TSP.2003.822357

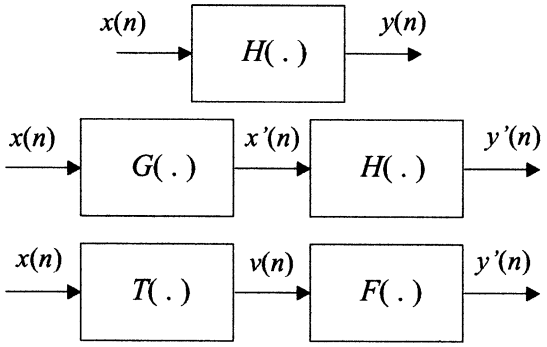

Fig. 1. (Top) Typical filtering operator H is replaced by a two-step approach. The filter $H(.)$ somehow maps each N-sample neighborhood of $x(n)$ into a sample of $y(n)$. In the proposed approach, signal $x(n)$ is approximated by the sequence $x'(n)$ via the quantizer $G(.)$. Then, the filter $H(.)$ is applied to the approximated sequence. In this paper, we demonstrate a method for finding a quantizer $T(.)$ and a filter $F(.)$ that correspond to the quantizer $G(.)$ and filter $H(.)$ with the advantage that T and F can be implemented as table lookups.

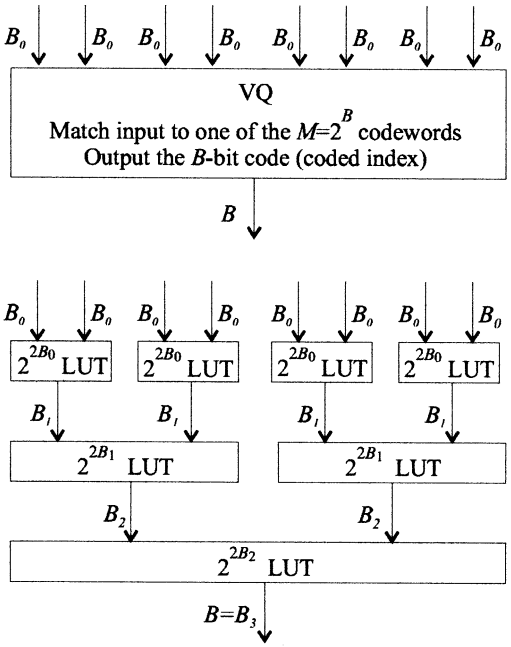

Fig. 2. (Top) Example diagram of a VQ encoding system: 8 B_0-bit samples are encoded into one B-bit codeword. (Bottom) Same encoding performed through three stages of HVQ. The number of bits at every HVQ stage is noted.

in a reasonable number of bits B_n (e.g., eight or ten), which are mapped to one of the $2^{B_{n+1}}$ codewords that are coded into B_{n+1} bits. It is clear that the process can be implemented using an LUT of $2^{2B_n}B_{n+1}$ bits. For example, if $B_n = B_{n+1} = 8$, a 64-KB LUT is sufficient. A comparison between the VQ and HVQ processes is depicted in Fig. 2, which shows a typical VQ system and its HVQ counterpart each encoding an 8-tuple of B_0-bit samples. In this example, the HVQ system uses three 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 encoders are generally slower than transform coders, HVQ encoders are much faster. For image processing, HVQ is adapted to two dimensions by alternating horizontal and vertical groupings of code pairs, as illustrated in Fig. 3.

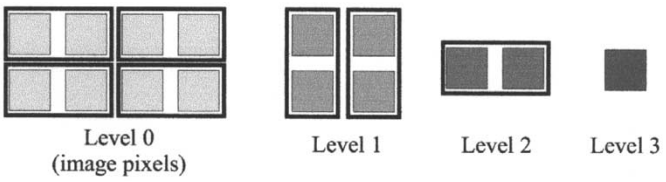

Fig. 3. Illustration of three stages of HVQ applied to an image block of 2×4 pixels.

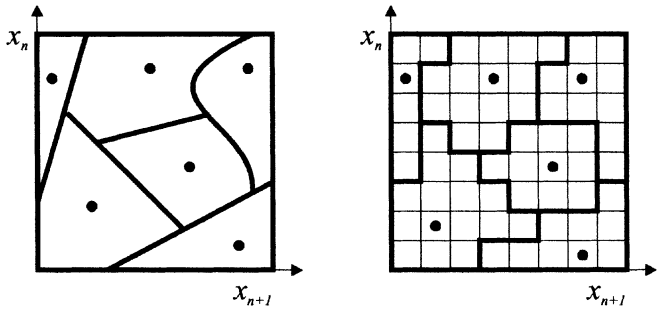

Fig. 4. (Left) Illustration of quantization of a 2-D space into six regions. (Right) Similar quantization of the 2-D space, wherein the input samples have already been quantized.

We are not interested in HVQ as a compression means but in its codebook design. For that, the design of HVQ codebooks is important. The design method is similar to the approach used in the VQ case. A popular algorithm for VQ design is the LBG [1], in which the space is partitioned into (Voronoi) quantization cells (regions). LBG is an interactive algorithm that tries to associate vectors to the closest region centroid and typically aims at reducing the reconstruction (quantization) mean-squared error. In order to design the first stage of Fig. 2, we have to partition the two-dimensional (2-D) sample space into 2^{B_1} Voronoi regions for a given distribution of the training data. In the second stage, the four-dimensional (4-D) space is partitioned into 2^{B_2} regions. The peculiarity in this approach is that there are only 2^{2B_1} valid points in the 4-D space and not continuous data. In the design of the last stage, the N-dimensional ($N = 8$) space contains at most 2^{2B_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 with 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 into 2^{B_n} regions because the input space has already been quantized. Although it is difficult to visualize the multidimensional process, we can trace a parallel to quantization of a simple 2-D space, which is illustrated in Fig. 4. Fig. 4 shows the partition of the 2-D space into six regions. It also shows the case where the input variables have already been quantized so that the partition of the 2-D 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. 4). 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 with HVQ is commonly a colored noise and is similar to the error spectrum of regular VQ. It

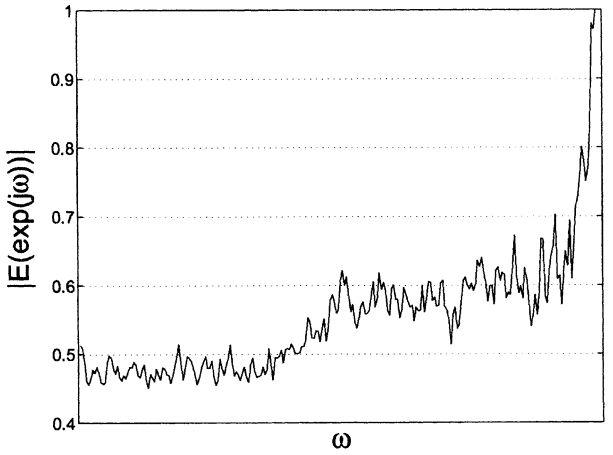

Fig. 5. Error spectrum of an HVQ approximation of an image when the training set is made of typically lowpass images.

depends on the training set used to design codebooks and quantization cells. The fact that the error spectrum is roughly complementary to the spectrum of the images in the training set is intuitive (and easily checked). If a system was trained with typical lowpass images, it will be hard for HVQ to encode high-frequency details. Conversely, if only high-frequency patterns were included for training, the codewords will likely have the same features, and low-frequency blocks will be more distorted. We trained our HVQ system using a typical image set, including several pictorial images and a few graphics so that the nature of the training set is essentially lowpass. We applied HVQ to compress the popular image Lena, and the spectrum of the approximation error is shown in Fig. 5. As expected, it is a high-pass spectrum, because of the training set. This is expected from most VQ systems.

III. Overlapped HVQ

As Fig. 2 shows, HVQ, like VQ, can make one single codeword index (code) to represent N samples. At the kth 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.

$$\begin{aligned}
&\underbrace{x_0x_1x_2x_3}_{c_1}\underbrace{x_4x_5x_6x_7}_{c_5}\underbrace{x_8x_9x_{10}x_{11}}_{c_9}\underbrace{x_{12}x_{13}x_{14}x_{15}}_{c_{13}}\\
&x_0\underbrace{x_1x_2x_3x_4}_{c_2}\underbrace{x_5x_6x_7x_8}_{c_6}\underbrace{x_9x_{10}x_{11}x_{12}}_{c_{10}}x_{13}x_{14}x_{15}\\
&x_0x_1\underbrace{x_2x_3x_4x_5}_{c_3}\underbrace{x_6x_7x_8x_9}_{c_7}\underbrace{x_{10}x_{11}x_{12}x_{13}}_{c_{11}}x_{14}x_{15}\\
&x_0x_1x_2\underbrace{x_3x_4x_5x_6}_{c_4}\underbrace{x_7x_8x_9x_{10}}_{c_8}\underbrace{x_{11}x_{12}x_{13}x_{14}}_{c_{12}}x_{15}
\end{aligned}.$$

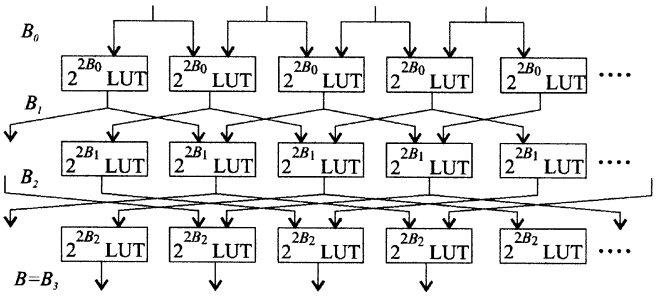

Fig. 6. Implementation diagram of overlapped HVQ.

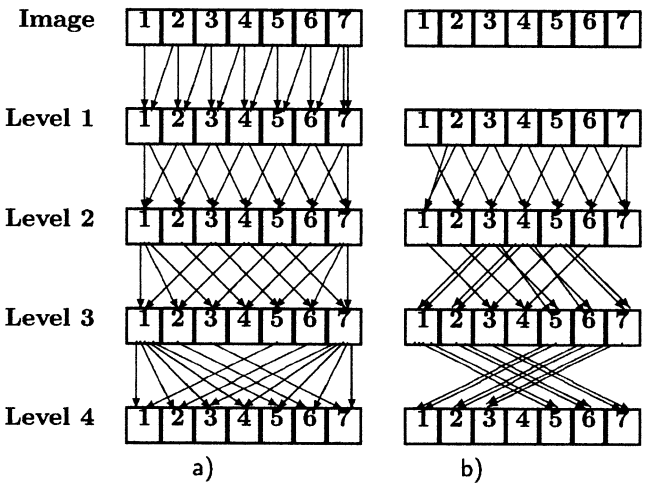

Fig. 7. Demonstration of two ways to deal with edge conditions in OHVQ, using an exagerated case where four-level OHVQ is applied to seven-sample data.

One key feature of OHVQ is its simplicity of implementation. A diagram for the implementation of OHVQ is shown in Fig. 6, which should be compared with the corresponding HVQ diagram in Fig. 2. For k stages, OHVQ requires k lookups per input sample.

In processing finite-length signals such as images, there should be a special procedure for the signal's borders. Fig. 7 shows a couple of potential methods for an intentionally stressing case of four-level OHVQ over only seven-sample data. Each HVQ sample at level i must have two parent samples at level $(i-1)$. In some cases, the intended sample at level $(i-1)$ would be outside the bounds of the image. In the method depicted in Fig. 7(a), the nearest parent sample to the intended one is used. An alternative method would be to simply repeat the parent sample that you do have, as shown in Fig. 7(b).

IV. LUT Filter

A. Filter Structure and Design

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 neighborhood ($\mathbf{x}_N[n]$) of a particular sample ($x[n]$). In other words, OHVQ is the transformation T in Fig. 1, and $v[n]$ is the output index. The mapping T can be inverted such that $x'_N[n] = T^{-1}(v[n])$. $x'_N[n]$ is an approximation of $x_N[n]$. Let $\Psi(v)$ be the set of all $x_N[n]$ that are mapped into $v[n]$. Since we use HVQ (OHVQ),

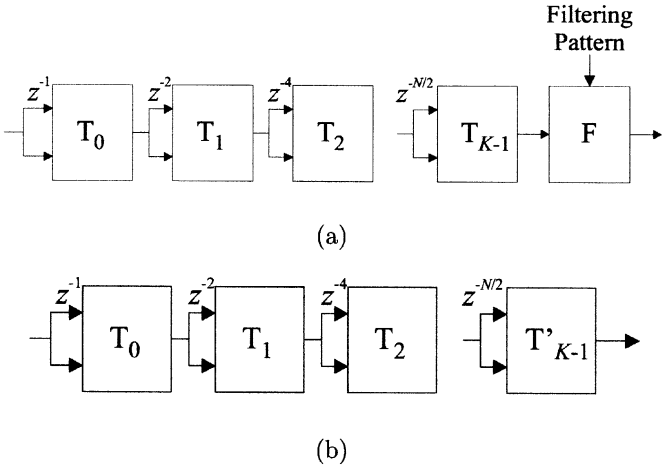

Fig. 8. Implementation diagram of the proposed filter. (a) T_n are OHVQ stages, and stage F is an LUT for the actual filter implementation, which should be folded into T_{K-1}, so that the filter can be implemented in K LUT stages, as in (b).

which uses training sets, the $\Psi(v)$ regions are commonly convex, and we typically set $x'_N[n]|v[n]$ as being the centroid of $\Psi(v)$ for some given distance measure [1].

In our framework, in Fig. 1, we map the sample's neighborhood to one output sample, i.e., a spatial filter is applied to the neighborhood (not to the real neighborhood but to its approximation). Therefore, F is applied to the output of the OHVQ system. Since F maps $v[n]$ to $y'[n]$ and $v[n]$ only assumes a discrete number of values, it is clear that F can be implemented via one LUT. Hence, the proposed signal processing system's diagram is shown in Fig. 8. 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 overall concept, it is obvious that F is unecessary and can be folded into T_{K-1} so that the number of LUTs in the HVQ filter remains K.

The LUT filter structure allows us to give up accuracy for generality. The OHVQ system (T_n stages) can be designed using the standard HVQ design method. Since we intend to have the best approximation of $\mathbf{x}_N[n]$, the generalized Lloyd design or LBG [1] is a good choice since it aims to reduce the error $e_N[n] = ||\mathbf{x}_N[n] - \mathbf{x}'_N[n]||$. Hence, in our experiments, the T_n stages are designed using the standard LBG codebook design algorithm [1]. Let us assume that we use the largest number of bits and stages that are practically allowed and assume that $||e[n]||$ is small for the moment. How do we design the actual filtering stage F? Following the objectives described in the introduction, we actually want to find F such that $y'[n] = F(v[n]) = H(\mathbf{x}'_N[n])$. Since $v[n]$ assumes one out of at most 2^B values, F is a simple 2^B-entry LUT. Its design is performed as follows: For each value of v, find the reconstruction N-tuple, and filter it with the desired filter H. In other words, the F LUT entries are filled with the following mapping:

$$v \rightarrow H(T^{-1}(v)) \ ; \text{ for all } 2^B \text{ values of } v.$$

In HVQ language, v are the indices, and we populate the LUT with the filtering output when filters are applied to each of the reconstruction vectors.

B. Alternative Filter Design

If the HVQ training set is made available to the design of F, one can use alternative methods. Even though $x'_N[n]$ is designed to be close to all $x_N[n]$ that yield the given $v[n]$ (i.e., the centroid of $\Psi(v)$), that does not mean that $H(x'_N)$ will be close to all $H(x_N)$, $x_N \in \Psi(v)$. It is conceivable to design F such that

$$F(v) = E\{x_N | x_N \in \Psi(v)\}.$$

There are practical means to achieve this design, but we are not addressing them here. All tests in this paper were carried using the standard method $F(v) = H(T^{-1}(v))$, which needs no knowledge of the HVQ training steps for modifying F.

In both cases, we use the indirect design approach: T_n are generic stages for approximating the signal, and F is the specific stage to acomplish filtering. Another strategy is the direct design of the stages. If we apply a direct design method, we would face tremendous practical obstacles. First, codebook design is invariably slow and cumbersome and must be repeated for every filter, even for the most minute change in the filter specs. Second, it demands modification of the distortion measure when partitioning the multidimensional space at the codebook design step. Third, HVQ design is greedy, and LUT filter design will have similar suboptimality problems. Last, an HVQ-like design (no overlap) is suboptimal for an OHVQ-like design that has overlap. For all these reasons, we will not deal with the direct design here. However, it is of interest to point out that the direct case is similar to neural network-based filtering, i.e., one can also use neural networks instead of LUTs and train the stages using traditional neural network design methods in order to "mimic" the operation of H.

C. Standard Filter Approximation Analysis

Since the input is only approximated, the output only approximates that of a desired filter H. Three main factors contributing to limit the OHVQ performance can be identified as

i) the spectrum of the approximation error;
ii) the number of stages;
iii) the number of bits used at each stage.

It is rather difficult to accurately model the error ($e_N[n]$). By sampling $\mathbf{x}'_N[n]$, one can obtain $x'[n]$, which is an approximation to $x[n]$. The signal approximation error can be modeled as $e[n] = x[n] - x'[n]$, and as discussed in a previous section, it depends on the training set. Fig. 5 shows a typical approximation error spectrum $|E(e^{j\omega})|$ computed for a test image for an HVQ system trained on lowpass images. Note the higher error at high frequencies, which makes us predict poorer LUT-filter performance for high-frequency signals. Thus, even before performing a single test, we can predict that LUT filters based on (typically trained) OHVQ would better approximate linear lowpass filters instead of linear highpass ones.

The number of OHVQ stages determines the block size and filter spatial support. The larger the block size, the worse the VQ performance, of course. The worse the approximation, the lower LUT filter performance. However, small filter support is not very useful, and therefore, we need a compromise. The number of bits used in each stage defines the LUT size influencing complexity and accuracy. The number of bits is bound, since a b-bit codewords stage forces the next stage to have 2^{2b} entries.

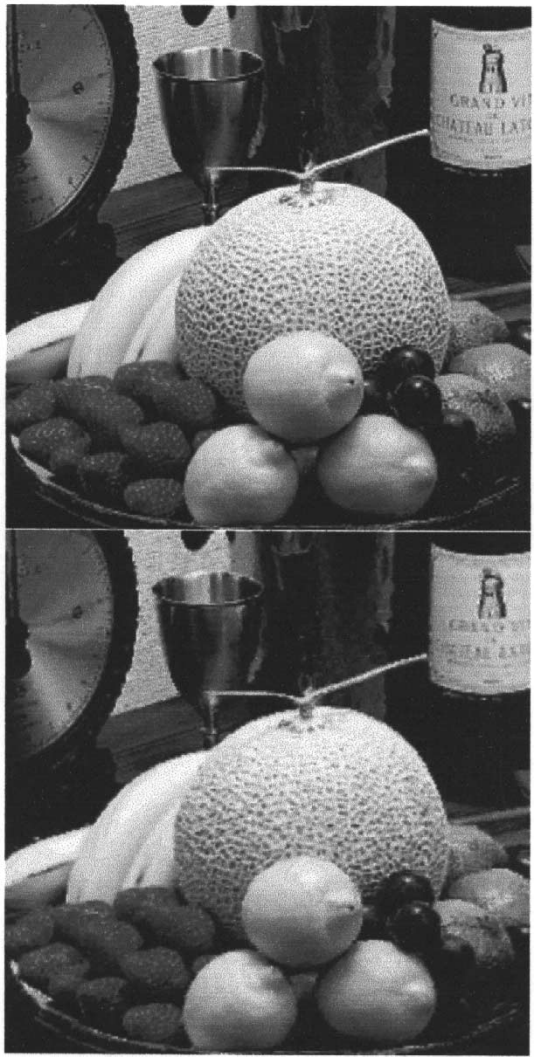

Fig. 9. (Top) Original image and (bottom) its reconstruction through OHVQ.

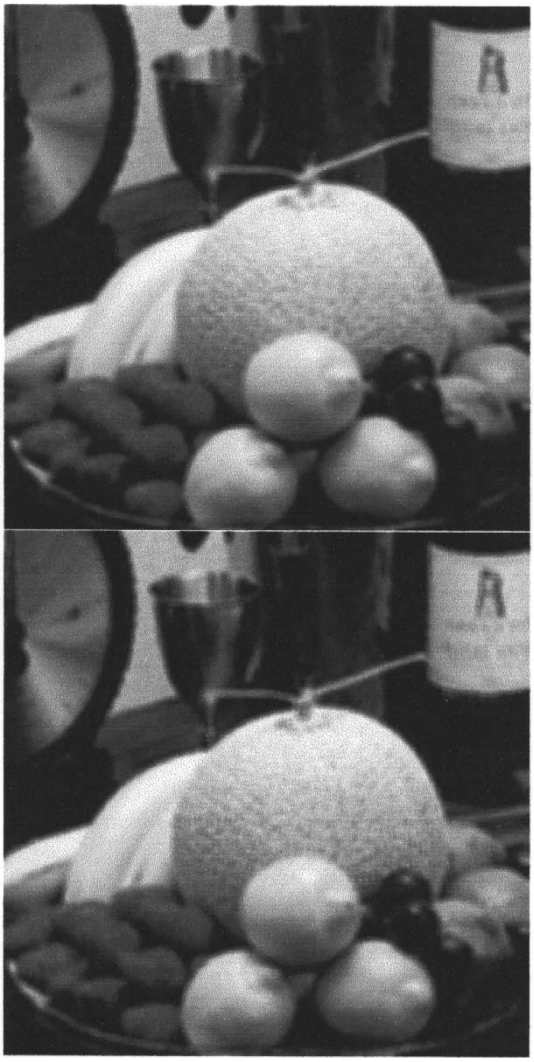

Fig. 10. (Top) Smoothed versions of the original image using regular smoothing filter and (bottom) the proposed LUT filter.

These three limitations (derived from trade-off compromises) narrow the scope of the LUT filter so that it should be used under specific conditions where K and the T_n can provide sufficiently good approximation in the frequency range of interest.

As a curiosity, if the T_n were linear operators of the type $x_{out} = a_1 x_{in1} + a_2 x_{in2}$, would the system in Fig. 8(b) implement general convolutions? The filter's transfer function would be $H(z) = A\prod_{i=0}^{K-1}\left(1+\beta_i z^{-2^i}\right)$, which is obviously not general. For example for an order-3 filter, if we use two 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 overall lattice format.

D. Implementation Discussion

For images, HVQ blocks of 8×8-pixels (six stages) are already too big for typical HVQ design (simple MSE-driven LBG). A support of 4×4 seems to be adequate, unless better design is applied. For typical 8-bit images, we use a minimum 10 bits per stage, which would generate 2-MB LUTs (2 MB if one decides to spend 2 bytes to encode a 10-bit word). For the tests in the next section, we employed 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 neighborhood. Hence, the filter used around 16 MB of memory space and four look-ups per pixel.

The choice to use an LUT filter instead of a regular one is clear: Whenever the approximation quality is good enough, and implementation is much faster. A question beyond the scope of this paper is when is "much faster" to implement LUTs than regular spatial filters. A simple averaging filter is probably faster than a few look-ups, but as filters become more complicated, involving floating point, comparisons, ranking, etc., the more advantageous it is for LUT filters. In addition, one has to chose an architecture with very large caches that is friendly to LUTs. Note that in some low bit-depth applications, the tables can be kept small to fit into fast memory banks.

E. Image Processing Examples

As a signal processing example, we processed images using the LUT filter. The original image is shown in Fig. 9, along with

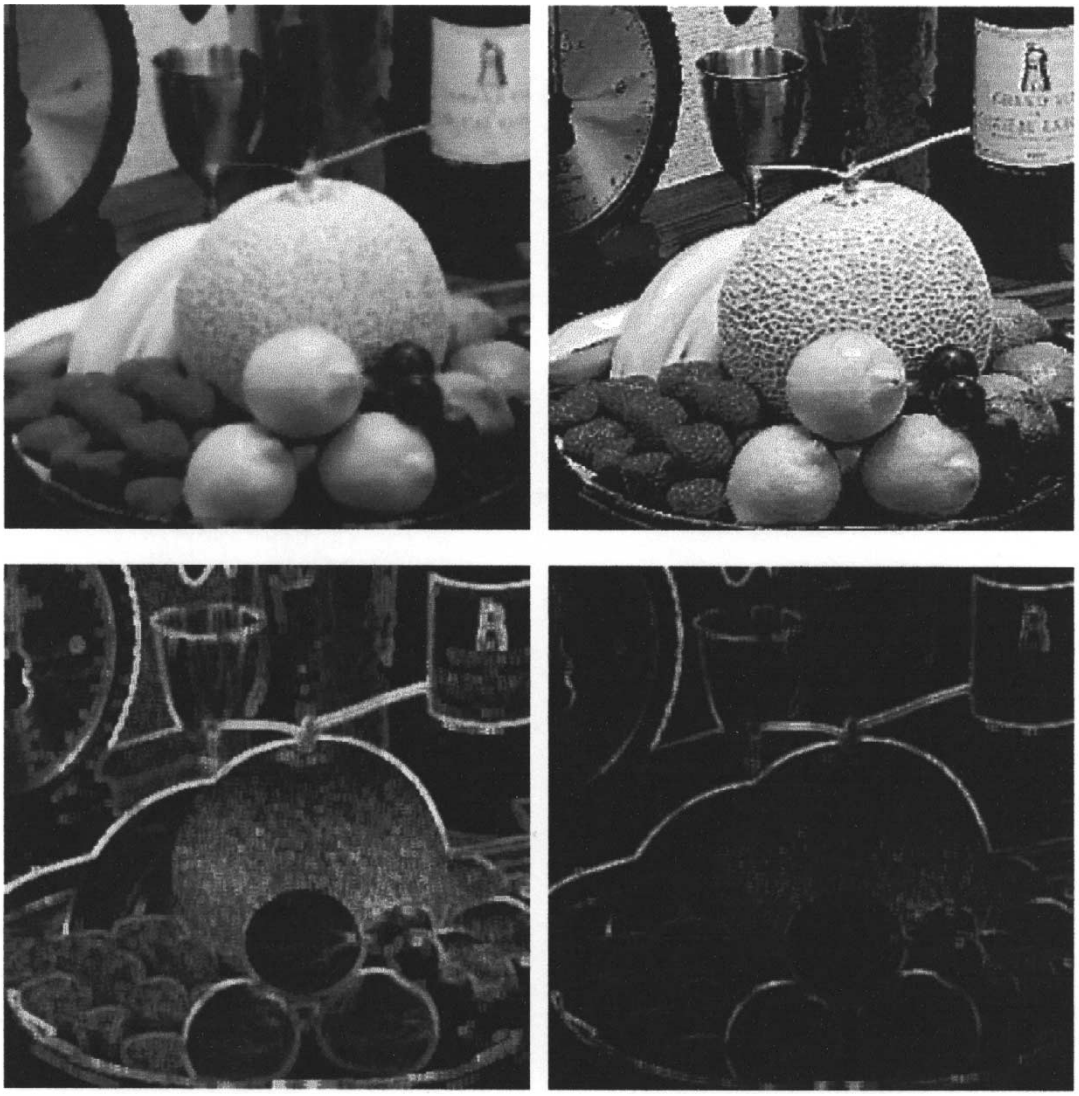

Fig. 11. Four-stage HVQ-LUT filter applied to the original image in 9 for implementing (top left) median and (top right) sharpening filters. Within the same approach, one can also compute local (bottom left) dynamic range and (bottom right) variance.

the reconstructed image using OHVQ (LUT filter where F is designed so that H is identity). For this reconstruction, we selected one central pixel of each 4×4 code vector (e.g., the one in the second column and row). Because HVQ and OHVQ introduce several errors, the distinction between the two "unprocessed" images is clear. The spectrum of the error between these two images is similar to the one in Fig. 5. Noise is spread through the spectrum with high concentration in the high frequencies. As we discussed, we expect better performance from lowpass-type operations. Fig. 10 shows images that were smoothed using a regular spatial filter (H is a mask with all 1's) and the proposed LUT filter (i.e., the $\mathbf{x}'_N$ in the codebook were averaged to produce the output pixel). One can see that the LUT filter is quite effective in mimicking the linear filter behavior. Fig. 11 shows a few other image processing examples obtained using the approach in Fig. 8. The original image is shown along with examples of 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 error spectrum, one can see that the sharpening LUT filter performs worse than the smoothing one, as compared with the expected output of their traditional convolution-based counterparts. The reader should note that all these examples were produced using exactly the same system, at a complexity of four LUTs per pixel. Filters are changed with a simple change of the pointer to the memory space where the entries of the fourth (last) stage reside.

V. Conclusion

The proposed HVQ-LUT filter is very fast and flexible. It is a generic process that can be applied to most spatial processing operations. Quality is limited to the approximation power of OHVQ. Complexity, however, is K lookups per pixel, regardless of the operation. This is typically faster than most linear or nonlinear spatial filters. Apart from filters, other possible operations could be image classification and segmentation, resolved on a pixel-by-pixel basis. LUT filter is an ongoing work. Because of the difficulty in the analysis of such a nonlinear process, future work will concentrate on better analysis tools, on improving the codebook design, and on redesigning the operator F.

REFERENCES

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

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

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

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

[5] ——, "Constrained and recursive hierarchical table look-up vector quantization," in *Proc. Data Compression Conf.*, Snowbird, UT, Mar. 1996.

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

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

Ricardo L. de Queiroz (SM'99) received the B.S. degree from Universidade de Brasilia, Brasilia, Brazil, in 1987, the M.S. degree from Universidade Estadual de Campinas, Campinas, Brazil, in 1990, and the Ph.D. degree from University of Texas at Arlington, in 1994, all in electrical engineering.

From 1990 to 1991, he was with the DSP research group at Universidade de Brasilia, as a research associate. He joined Xerox Corp., Webster, NY, in 1994, where he was a member of the research staff until 2002. From 2000 to 2001, he was also an Adjunct Faculty at the Rochester Institute of Technology, Rochester, NY. He is now with the Electrical Engineering Department at Universidade de Brasilia. He has published extensively in Journals and conferences and contributed chapters to books as well. He also holds 25 issued patents, while many others are are still pending. He received grants over all his graduate school years, including several scholarships and grants from the Brazilian government and awards and assistantships from universities. His research interests include multirate signal processing, image and signal compression, and color imaging.

Dr. De Quiroz is an associate editor for the IEEE SIGNAL PROCESSING LETTERS. He has been actively involved with the Rochester chapter of the IEEE Signal Processing Society, where he served as Chair and organized the Western New York Image Processing Workshop since its inception. He was also part of the organizing committee of ICIP'2002. He is a member of IS&T.

Patrick A. Stein received the B.S. degree in computational mathematics from the Rochester Institute of Technology, Rochester, NY.

He spent seven years doing digital image processing work at the Xerox Corporation, Webster, NY. He is currently with the Rochester Institute of Technology, working on NASA's Stratospheric Observatory for Infrared Astronomy.

Mr. Stein is a member of the American Mathematical Society and the sole-proprietor of Nklein Software.