On the Rate-Distortion Performance and Computational Efficiency of the Karhunen–Loève Transform for Lossy Data Compression

Hanying Feng, *Student Member, IEEE,* and Michelle Effros, *Member, IEEE*

***Abstract*—We examine the rate-distortion performance and computational complexity of linear transforms for lossy data compression. The goal is to better understand the performance/complexity tradeoffs associated with using the Karhunen–Loève transform (KLT) and its fast approximations. Since the optimal transform for transform coding is unknown in general, we investigate the performance penalties associated with using the KLT by examining cases where the KLT fails, developing a new transform that corrects the KLT's failures in those examples, and then empirically testing the performance difference between this new transform and the KLT. Experiments demonstrate that while the worst KLT can yield transform coding performance at least 3 dB worse than that of alternative block transforms, the performance penalty associated with using the KLT on real data sets seems to be significantly smaller, giving at most 0.5 dB difference in our experiments. The KLT and its fast variations studied here range in complexity requirements from $O(n^2)$ to $O(n \log n)$ in coding vectors of dimension n. We empirically investigate the rate-distortion performance tradeoffs associated with traversing this range of options. For example, an algorithm with complexity $o(n^{3/2})$ and memory $o(n)$ gives 0.4 dB performance loss relative to the full KLT in our image compression experiments.**

***Index Terms*—Image compression, Karhunen–Loève transform, optimal transform coding, separable coding.**

I. Introduction

TRANSFORM coding is a popular technique for data compression, in particular for lossy compression of images and video. The popularity of transform codes arises from their combination of low computational complexity with good coding performance.

Transform codes achieve their good complexity/performance tradeoff through a combination of transformation and uniform scalar quantization. Typical transform codes operate by first transforming input data vector $x^n = (x_1, x_2, \ldots, x_n)^t$ as $y^n = T[x^n]$, then sending each transform coefficient y_i, $1 \le i \le n$, through a uniform scalar quantizer, and finally losslessly describing the quantized vector $\hat{y}^n$. The decoder reverses this procedure to form an image reproduction $\hat{x}^n = T^{-1}[\hat{y}^n]$, where $T^{-1}[\cdot]$ denotes the (possibly imperfect) reversal of transform $T[\cdot]$. By applying scalar quantization to the transformed data set rather than the original data set, transform codes approximate the rate-distortion performance of higher dimensional vector codes at lower computational cost.

Manuscript received January 18, 2000; revised October 3, 2001. This material is based upon work supported in part by the NSF under CAREER Award MIP-9501977, the Intel 2000 program, and the Powell Foundation. The material in this paper was presented in part at the International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Phoenix, AZ, in March 1999. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Joern Osterman.

The authors are with the Department of Electrical Engineering, California Institute of Technology, Pasadena, CA 91125 USA (e-mail: fhy@z.caltech.edu; effros@caltech.edu).

Publisher Item Identifier S 1057-7149(02)00806-0.

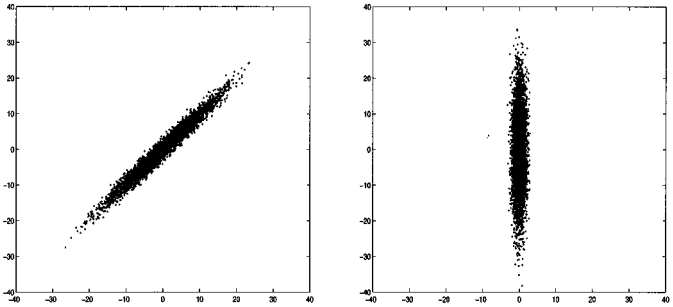


Fig. 1. The KLT rotates (left) a correlated Gaussian source distribution to (right) align with the axes.

If T is a linear block transform, then $T[x^n] = Ax^n$ for some $n \times n$ matrix A independent of x^n. Examples of popular linear block transform codes include the JPEG and MPEG image and video coding standards [1], which rely on the discrete cosine transform (DCT). The SPIHT algorithm [2] and its descendants are transform codes using wavelets rather than linear block transforms.

For years, the Karhunen–Loève transform (KLT) has been the best available transform for orthogonal block transform coding. The choice of the KLT is motivated by three arguments. First, by rotating the data so that all off-diagonal terms of the covariance matrix $E[(Y^n - EY^n)(Y^n - EY^n)^t]$ equal zero, the KLT decorrelates the data. Since decorrelation aligns a Gaussian distribution with the symbol axes (see Fig. 1), the KLT often gives a good rotation for scalar quantization. Second, the KLT achieves optimal energy compaction, minimizing the number of coefficients needed to reconstruct the data at a desired accuracy [3]. And third, when the rate-distortion performance associated with quantizing a source of variance σ^2 with a b-bit uniform scalar quantizer is approximated as $c2^{-2b}\sigma^2$ for some constant c independent of A (a combination of a high-rate approximation and an assumption about the relationship between the original distribution and its rotation), the KLT maximizes the coding gain

$$G(A) = \frac{c2^{-2b}\sigma_x^2}{c2^{-2b}\left(\prod_{k=1}^{n}\sigma_k^2(A)\right)^{1/n}} = \frac{\sigma_x^2}{\left(\prod_{k=1}^{n}\sigma_k^2(A)\right)^{1/n}}$$

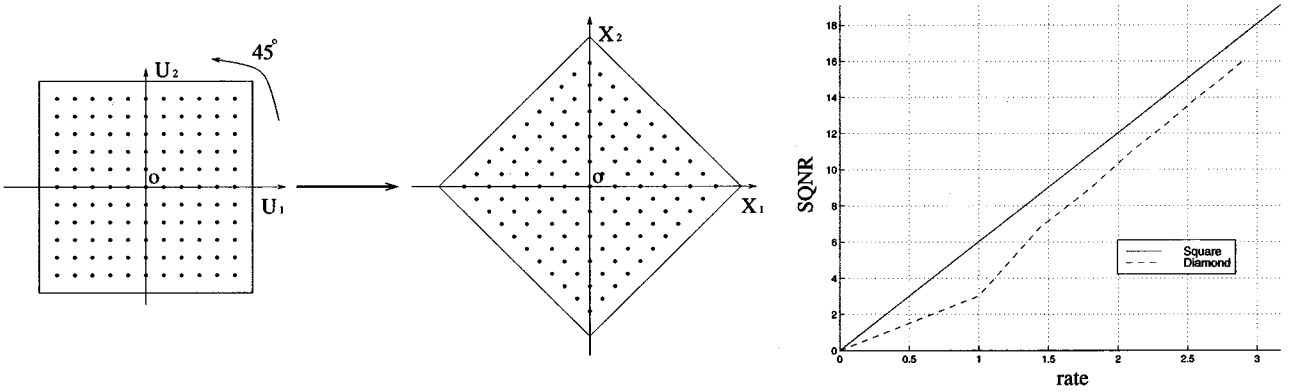


Fig. 2. (Left) the square distribution of U^2 is better suited for scalar quantization and entropy coding than (right) the diamond distribution of X^2, giving as much as 3-dB performance improvement.

where $\sigma_k^2(A)$ is the variance of the kth coefficient of $Y^n = AX^n$ (e.g., [4, Appendix C]).

Yet, the KLT is not optimal for transform coding in general. Let $U^2 = (U_1, U_2)^t$, where U_1 and U_2 are independent and identically distributed (iid) uniform random variables on $[-1, 1]$. Here, (U_1, U_2) and all possible rotations of (U_1, U_2) are uncorrelated. For example if $X^2 = (X_1, X_2)^t$ with $X_i = \left(U_1 + (-1)^i U_2\right)/\sqrt{2}$, X_1 and X_2 are uncorrelated, and thus U^2 and X^2 are both possible outcomes of the KLT. Yet U^2 and X^2 are not equally good sources for scalar quantization; without overload error, the square distribution of U^2 does up to 3 dB better than the diamond distribution of X^2 when each is coded using optimal bit allocation, uniform scalar quantization, and entropy coding (see Fig. 2, where rate is measured in bits per symbol (bps)). Details of this calculation appear in Appendix A.

The goal of this work is to better understand the KLT's strengths and limitations for block transform coding. The paper is organized as follows. In Section II we consider the question of why the KLT's properties of optimal decorrelation, energy compaction, and coding gain fail to yield transform coding optimality, and we bound the potential performance differences between the best KLT and the worst KLT for transform coding, exploring both theoretical limits and empirical results on practical data sets. Section III introduces a variety of fast variations on the KLT and demonstrates the potential performance degradations of these approximations to the KLT in transform coding.

II. Assessing the Performance of the KLT

The example of Section I demonstrates that decorrelation, energy compaction, and $G(A)$ maximization are not sufficient to guarantee optimality for transform coding. While decorrelation of a correlated Gaussian source results in the alignment of the source distribution with the symbol axes and the independence of the transform coefficients, the same is not true for all sources, as shown by the diamond distribution example. Similarly, while the KLT yields the optimal energy compaction, the diamond distribution example demonstrates that the optimal energy compaction is neither unique nor sufficient for optimal transform coding. Finally, maximizing $G(A)$ fails to yield true coding gain optimality when the approximations used to calculate $G(A)$ do not apply. For example, $\left(\prod_{i=1}^n \sigma_{y_i}^2\right)^{1/n}$ is constant while c varies significantly across rotations of (U_1, U_2); thus, in this case, finding the optimal transform for any rotation of (U_1, U_2) requires a search for the transform that minimizes c, not the transform that minimizes $\left(\prod_{i=1}^n \sigma_{y_i}^2\right)^{1/n}$.

The diamond distribution example shows that the penalty for choosing the wrong KLT in two dimensions can be high. Similar problems occur at higher dimensions.

Let $U^n = (U_1, \ldots, U_n)^t$, where $U_1, \ldots, U_n$ are iid uniform random variables on $[-1, 1]$. All possible rotations of U^n are decorrelated. Rotate the hypercube $[-1, 1]^n$ so that each long diagonal aligns with a symbol axis, and let X^n be a random vector distributed uniformly on this "generalized diamond." This rotation is not possible for all possible values of n (e.g., for $n = 3$ the long diagonals are not perpendicular) but is possible when $n = 2^p$ for some integer p. For any such n, we can achieve the desired distribution by letting $X^n = B_\diamond U^n$, where $B_\diamond = (1/\sqrt{n})B_{\diamond,p}$

$$B_{\diamond,k} = \begin{bmatrix} B_{\diamond,k-1} & -B_{\diamond,k-1} \\ B_{\diamond,k-1} & B_{\diamond,k-1} \end{bmatrix} \text{ for all } k \geq 1$$

and $B_{\diamond,0} = 1$. Since $U_1, \ldots, U_n$ are iid, when $n \gg 1$ the marginal distribution of X_i approaches normal distribution $\mathcal{N}(0, 1/3)$ for each $i = 1, 2, \ldots, n$. Fig. 3 compares the optimal uniform scalar quantization and entropy coding performance on a $\mathcal{N}(0, 1/3)$ random variable (based on numerical results from [5]) to the optimal uniform scalar quantization and entropy coding performance on U_i. In this case, the maximal loss is approximately 1.5 dB. In fact, the maximal loss of the worst KLT relative to the best KLT can be made arbitrarily large. For example given iid samples from a uniformly distributed binary random variable on $\{-a, a\}$, the output of the best KLT has a distribution that is identical to the input distribution while the output of the worst KLT has distribution that approaches the normal distribution as n grows without bound; thus as $n \to \infty$, the best KLT's output can be reproduced at distortion 0 with rate 1 while the worst KLT's output cannot be reproduced at distortion zero. The resulting difference in SQNRs approaches ∞.

In practice, the KLT is generally derived from an estimated covariance matrix matched to the training data. Estimation errors can also lead to problems with the KLT.

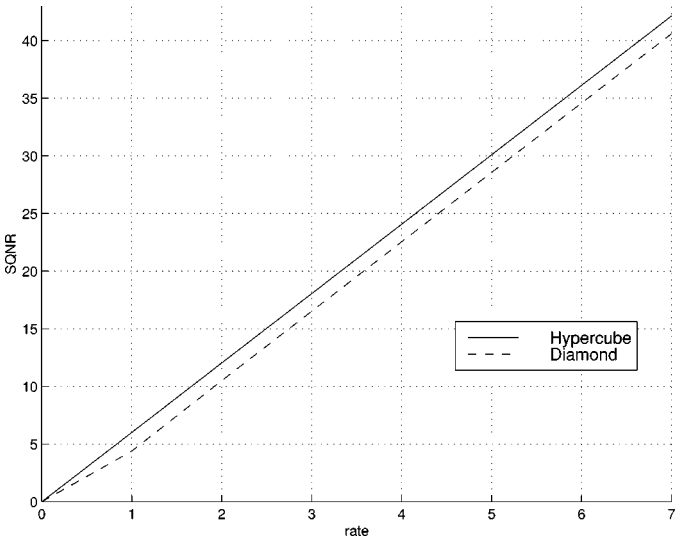


Fig. 3. The hypercube distribution of U^n is better suited for scalar quantization and entropy coding than the generalized diamond distribution of X^n, giving as much as 1.5-dB performance improvement.

Let $\hat{R}_X$ be the estimated value of the true covariance matrix R_X, where

$$R_X = \begin{bmatrix} R_1 & 0 \\ 0 & R_2 \end{bmatrix}, \quad \hat{R}_X = \begin{bmatrix} R_1 + \delta_1 & \delta_2 \\ \delta_2 & R_2 + \delta_3 \end{bmatrix}$$

and δ_1, δ_2, and δ_3 are small but nonzero. When $R_1 \approx R_2$, small variations in $(\delta_1, \delta_2, \delta_3)$ can yield large differences in the KLT. In particular, while the eigenvectors of R_X are $[0,1]^t$ and $[1,0]^t$, those of $\hat{R}_X$ are

$$c\left[1, \frac{R_2 + \delta_3 - R_1 - \delta_1 \pm \sqrt{(R_2 + \delta_3 - R_1 - \delta_1)^2 + 4\delta_2^2}}{2\delta_2}\right]^t$$

for some normalizing constant c. For example, if X_1 and X_2 are uniformly distributed on $[-1,1]$, then $R_1 = R_2 = 1/3$, and if $\delta_2 \neq 0$, $\delta_1 \approx \delta_3$ yields a 45° rotation, which turns the easily compressed square distribution into the undesirable diamond distribution. Similar difficulties arise at all dimensions.

Since the optimal transform for transform coding of general sources is unknown, assessing the potential performance penalty associated with using the KLT rather than the optimal transform on real data sets is difficult. We attempt such an assessment by designing a transform that corrects the previously observed KLT problems and comparing the performances of our "improved KLT" (iKLT) and the KLT on real data. Without loss of generality, we assume all random values have mean zero.

It can be shown that the KLT is the transform that maximizes function

$$F_2(Y^n) = \sum_{i=1}^{n} E\left[Y_i^2\right]^2$$

of $Y^n = AX^n$ over all unitary transforms A. Since the KLT's reliance on second order statistics exhibits the shortcomings illustrated previously, we turn to fourth order statistics and function

$$F_4(Y^n) = \sum_{i=1}^{n} E\left[Y_i^4\right].$$

The following example motivates the use of $F_4(Y^n)$.

Let $U^n = (U_1, \ldots, U_n)^t$ where $U_1, \ldots, U_n$ are iid random variables. Let $X^n = BU^n$, for some rotation matrix B. An intuitively good transform matrix for X^n is B^{-1}, but all possible rotation matrices are KLT's for X^n. Let $Y^n = AX^n = ABU^n$, and let $C = AB = [c_{ij}]_{n\times n}$. Then $Y_i = \sum_{j=1}^{n} c_{ij}U_j$, and since $E\left[U_i U_j^3\right] = 0$ for all $i \neq j$

$$\begin{aligned} F_4(Y^n) &= \sum_{i=1}^{n} E\left(\sum_{j=1}^{n} c_{ij}U_j\right)^4 \\ &= \sum_{i=1}^{n}\left(\sum_{j=1}^{n} c_{ij}^4 E\left[U_j^4\right] + \sum_{j=1}^{n}\sum_{k\neq j} 3c_{ij}^2 c_{ik}^2 E\left[U_j^2 U_k^2\right]\right) \\ &= \sum_{i=1}^{n}\left(\sum_{j=1}^{n} c_{ij}^4 E\left[U_1^4\right] + \sum_{j=1}^{n}\sum_{k\neq j} 3c_{ij}^2 c_{ik}^2 E\left[U_1^2\right]^2\right) \\ &= \sum_{i=1}^{n}\left[E\left[U_1^4\right]\left(\sum_{j=1}^{n} c_{ij}^2\right)^2 + \Delta\sum_{j=1}^{n}\sum_{k\neq j} c_{ij}^2 c_{ik}^2\right] \\ &= nE\left[U_1^4\right] + \Delta\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k\neq j} c_{ij}^2 c_{ik}^2 \end{aligned}$$

where $\Delta = 3E\left[U_1^2\right]^2 - E\left[U_1^4\right]$ and the final equality follows from the fact that A and B are unitary, which implies $C = AB$ is unitary, and thus $\sum_j c_{ij}^2 = 1$ for any i.

If $\Delta > 0$ (e.g., when the U_i's are uniformly distributed), then $F_4(Y^n) \geq nE\left[U_1^4\right]$; if $\Delta < 0$, then $F_4(Y^n) \leq nE\left[U_1^4\right]$. In both cases, $F_4(Y^n) = nE\left[U_1^4\right]$ only when $c_{ij}^2 c_{ik}^2 \equiv 0$ for any i, j, k such that $j \neq k$, which occurs only when each column of C contains exactly one nonzero element, as the following argument demonstrates. Suppose that column k of C has at least two nonzero elements, say $c_{lk}, c_{mk} \neq 0$. Then, achieving $c_{ij}^2 c_{ik}^2 \equiv 0$ for all $j \neq k$ requires c_{lj} and c_{mj} to be zero for all $j \neq k$. The resulting C cannot be a unitary matrix, giving a contradiction.

The solution $A^* = B^{-1}$ would give an iid transform output well-suited for scalar quantization. This solution requires that C be the $n \times n$ identity matrix I_n. When $\Delta > 0$, minimizing $F_4(Y^n) = F_4(CU^n)$ gives $C = I_n$; when $\Delta < 0$, maximizing $F_4(Y^n) = F_4(CU^n)$ gives $C = I_n$.

Based on this observation, we propose the iKLT

$$A^* = \begin{cases} \arg\min_A F_4(AX^n), & \text{if } \Delta > 0 \\ \arg\max_A F_4(AX^n), & \text{if } \Delta < 0 \\ \arg\max_A F_2(AX^n), & \text{if } \Delta = 0 \end{cases}$$

where $\Delta = 3E\left[U_1^2\right]^2 - E\left[U_1^4\right]$. When $X^n = BU^n$ for some iid vector U^n, the proposed transform gives the solution $A^* = B^{-1}$ if $\Delta \neq 0$ and the KLT of X^n if $\Delta = 0$.

Unfortunately, generalizing the above analysis to sources that are not iid is difficult. Even if the source is some rotation of independent random variables $U_1, \ldots, U_n$, if $U_1, \ldots, U_n$ are not identically distributed, then $E\left[U_i^2\right]$ and $E\left[U_i^4\right]$ vary with i. Further, we usually know only the statistics of X^n and not those of U^n. We therefore replace the previous equation for Δ by

$$\Delta = \sum_{j=1}^{n} \sum_{k \neq j} \left(3E\left[U_j^2\right] E\left[U_k^2\right] - E\left[U_j^4\right]\right).$$

Since $E\left[\left(\sum_j X_j^2\right)^2\right] = E\left[\left(\sum_j U_j^2\right)^2\right]$

$$\Delta = (n+2)\sum_{j=1}^{n} \sum_{k \neq j} E\left[U_j^2\right] E\left[U_k^2\right] - (n-1)E\left[\left(\sum_{j=1}^{n} X_j^2\right)^2\right]$$

where we approximate $\left\{E\left[U_j^2\right]\right\}$ by the eigenvalues of the covariance matrix of X. For $X^n = BU^n$, this approach still yields $A^* = B^{-1}$. It can be shown that the resulting A^* equals the KLT for all Δ when X^n is any joint Gaussian vector. Further, the iKLT achieves greater robustness to statistical estimation error than the KLT.

In designing an algorithm to find the iKLT when $\Delta \neq 0$ (the iKLT is identical to the KLT when $\Delta = 0$), recall that any unitary matrix A can be decomposed as the product of $n(n-1)/2$ Jacobi rotation matrices [6], where for any $1 \leq i < j \leq n$, the $n \times n$ Jacobi rotation matrix $J_{i,j}(\theta_{i,j})$ takes the form $J_{i,j}(\theta_{i,j}) = [\rho_{k,\ell}]_{n \times n}$, with

$$\rho_{k,\ell} = \begin{cases} 1, & \text{if } k = \ell \notin \{i,j\} \\ \cos\theta_{i,j}, & \text{if } k = \ell \in \{i,j\} \\ -\sin\theta_{i,j}, & \text{if } k = j, \ell = i \\ \sin\theta_{i,j}, & \text{if } k = i, \ell = j \\ 0, & \text{otherwise.} \end{cases}$$

Thus, for any unitary matrix A, there exists a collection of angles $\theta_{i,j}$ such that $A = \prod_{i=1}^{n-1} \prod_{j=i+1}^{n} J_{i,j}(\theta_{i,j})$.

Using this expansion, $Y^n = AX^n$ implies

$$Y^n = J_{n-1,n} J_{n-2,n} \cdots J_{i,n} \cdots J_{i,i+1} \cdots J_{2,3} J_{1,n} \cdots J_{1,2} X^n.$$

For any $1 \leq i < j \leq n$, define $X_{i,j}^n$ as

$$X_{i,j}^n = (X_{i,j,1}, X_{i,j,2}, \cdots, X_{i,j,n})^t = J_{i,j} \cdots J_{1,2} X^n.$$

Then, we have $n(n-1)/2$ intermediate vectors of the form $X_{i,j}^n = J_{i,j} X_{k,l}^n$ starting with $X_{1,2}^n = J_{1,2}X^n$ and ending with $Y^n = X_{n-1,n}^n = J_{n-1,n} X_{n-2,n}^n$. (Here, $(k,l) = (i, j-1)$ if $j > i+1$ and $(k,l) = (i-1, n)$ if $j = i+1$.)

We use a greedy algorithm to form progressively better and better estimates of A^*. Given a fixed $X_{k,l}^n$, let

$$\theta_{i,j}^* = \begin{cases} \arg\min_\theta F_4\left(J_{i,j}(\theta_{i,j}) X_{k,l}^n\right), & \text{if } \Delta > 0 \\ \arg\max_\theta F_4\left(J_{i,j}(\theta_{i,j}) X_{k,l}^n\right), & \text{if } \Delta < 0, \end{cases}$$

and $A_{i,j}^* = J_{i,j}\left(\theta_{i,j}^*\right) A_{k,l}^*$. Since

$$\begin{aligned} X_{i,j,m} &= X_{k,l,m} \text{ if } m \neq i \text{ and } m \neq j \\ X_{i,j,i} &= X_{k,l,i} \cos\theta_{i,j} + X_{k,l,j} \sin\theta_{i,j} \\ X_{i,j,j} &= -X_{k,l,i} \sin\theta_{i,j} + X_{k,l,j} \cos\theta_{i,j} \end{aligned}$$

finding $\theta_{i,j}^*$ is equivalent to choosing the $\theta_{i,j}$ that sends $E\left[X_{i,j,i}^4 + X_{i,j,j}^4\right]$ to its extremum. See Appendix B for a derivation $\theta_{i,j}^*$ for a fixed $X_{k,l}^n$ and a discussion of efficient methods for updating existing knowledge about the statistics of $X_{k,l}^n$ to yield the statistics of $X_{i,j}^n$ for use in the derivation of the subsequent θ values.

Since $F_4\left(X_{i,j}^n\right) = F_4\left(X_{k,l}^n\right)$ when $\theta_{i,j} = 0$, sequential optimization can only improve our estimate of A^*. Iterating the sequence of optimizations may further improve the estimate of A^*. Further, since the sequence of F_4 values is bounded and monotonic, the algorithm guarantees convergence. Since each step in the iterative procedure finds a globally optimal $J_{i,j}$, the algorithm converges to a locally optimal solution. This estimation algorithm is equivalent to Jacobi's method if the source is Gaussian.

A summary of the iKLT estimation algorithm follows.

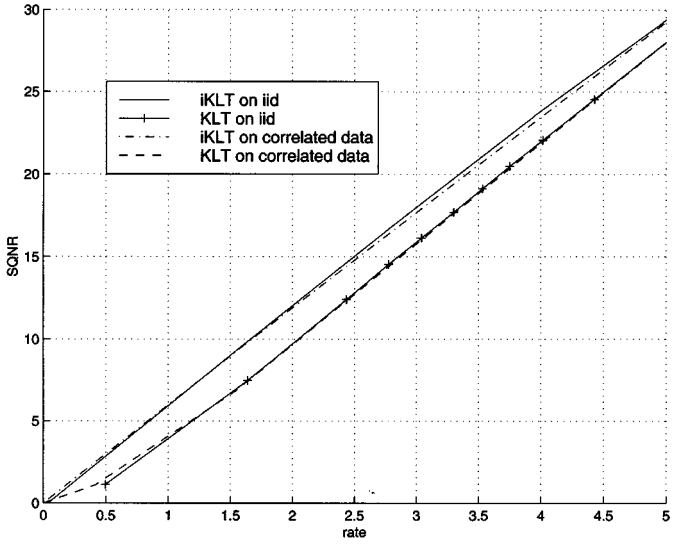
iKLT Estimation Algorithm

```
Initialize:
 Calculate fourth-order statistics:
  {E'_{p,q,s,t} = E[X_p X_q X_s X_t]}_{p,q,s,t∈{1,...,n}}.
 Set A = I_n.
 Calculate Δ.
If Δ = 0
 then find the KLT.
Else Repeat until convergence:
  For i = 1 to n
   For j = i + 1 to n
    Find θ*_{i,j}.
    Update A to J_{i,j}(θ*_{i,j}) A.
    Update the estimates {E'_{p,q,s,t}}.
   End.
  End.
End.
Output A.
```

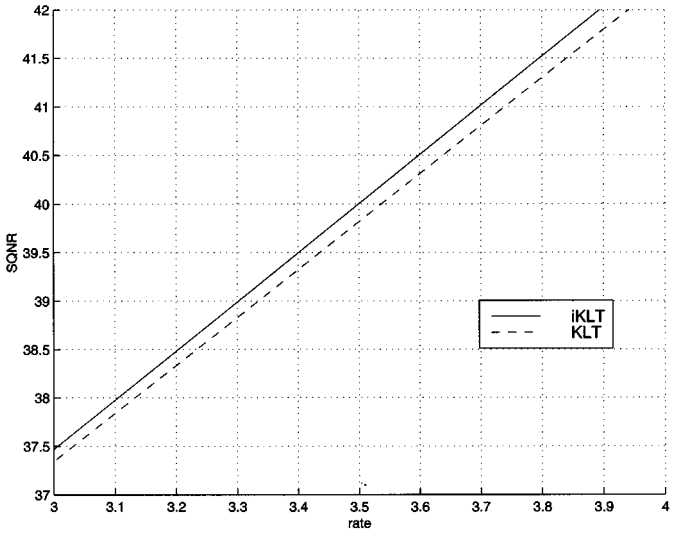
A similar approach was proposed in [7] after the submission of this paper.

Since the expectation is independent of the order of the coefficients, $E_{pqst} = E_{pqts} = \cdots = E_{tsqp}$, we only need E_{pqst} for which $p \leq q \leq s \leq t$, and the algorithm requires calculation and storage of approximately $n^4/24$ expectations.

Since the KLT seems to run into problems primarily when the eigenvalues of multiple components are close, we can improve iKLT estimation efficiency by performing the design only on those components. In practice, the number of such components is typically small, and this approach requires less computation and memory than full iKLT design and gives performance between that of the iKLT and KLT. We do not use this approach in the reported experimental results. (The complexity considered here is design complexity. The algorithms are identical to



(a)



(b)

Fig. 4. Comparison of KLT and iKLT on (a) iid and correlated data sets and (b) image data set.

both each other and the KLT in their run-time complexity and memory.)

In order to assess the performance penalty associated with using the KLT rather than the optimal transform for transform coding, we compare the transform coding performance of the iKLT to that of the KLT achieved by Householder reduction followed by the QL algorithm with implicit shifts [6]. Results included here show the code performance on two synthetic data sets and an image data set. The first synthetic data set contains iid uniform samples on [0,255]. The second contains correlated vectors created by blocking the first data set into 64-dimensional vectors and rotating each vector as $X^n = B_\diamond U^n$, where $B_\diamond$ is chosen to maximize symbol correlation. The final data set consists of images scanned from the *IEEE Spectrum* that contain similar proportions of text and photographic material.

Fig. 4 compares the performance of transform codes based on the KLT and iKLT on (a) the iid and correlated data sets and (b) the image data set. All results show signal to quantization noise ratio (SQNR) as a function of rate. Each data set contains training and test sets that do not overlap. For the synthetic data sets, the iKLT gives gains of approximately 1.5 dB over the KLT. Since, given precise knowledge of the source distribution, the iKLT and KLT would achieve identical performance for the iid source, the 1.5 dB performance improvement can be attributed to the iKLT's greater robustness to estimation errors. For the image data set, the iKLT gives gains up to approximately 0.5 dB over the KLT (at rate 5). The smaller gains on the image data set suggest that fewer equivalent eigenvalues are encountered in designing transform codes for this data set. This empirical evidence suggests that while use of the KLT rather than the optimal transform for transform coding can result in severe performance penalties, the penalty for real data sets—even data sets that clearly do not meet any assumptions of Gaussianity—may be quite small. While the results of Fig. 4(a) compare performance using a traditional transform code, the results of Fig. 4(b) compare performance using a weighted universal transform code (WUTC) [8]–[10]. The WUTC replaces the single transform and bit allocation of a traditional transform code with a collection of transforms and bit allocations in order to allow good performance on inhomogeneous data sets.

TABLE I
GAINS OF iKLT OVER KLT IN 3-D-SPIHT CODING OF MULTISPECTRAL DATA

IMAGE	1	2	3	4	5	6	7	8	9
WIDTH	512	256	256	256	256	256	256	256	256
BANDS	16	32	32	32	32	32	32	32	32
GAIN (dB)	.229	.041	.158	.278	-.00004	.067	.303	.191	.060

We also compare the performance of iKLT and KLT in the 3-D-SPIHT algorithm for compression of multispectral images [11]. Table I shows the maximum gains achieved by using iKLT instead of KLT on nine multispectral images. Here "width" and "bands" describe the size and number of spectral bands of each image. In most cases, there are small performance improvements. In one case, we observe a tiny performance degradation. This degradation may be due to local optimality, floating point error, or mismatch between the iKLT and the wavelet used in this code.

III. COMPUTATIONAL COMPLEXITY

The discussion of Section II investigates the potential performance penalties associated with using the KLT rather than the optimal transform for transform coding. In this section, we consider the transform coding performance penalties associated with using low-complexity approximations to the KLT rather than the KLT itself.

Fast transforms are often achieved through transform decomposition, as examples like the fast Fourier transform (FFT), DCT, and separable KLT [12] (SKLT) illustrate. We pursue a similar approach here, but since the transform matrices of the KLT are not separable in general, the best that we can hope for are fast *approximations* to the KLT. We here consider the performance/complexity tradeoffs of such approximations for transform coding.

The SKLT decomposes an $n \times n$ KLT into a pair of $\sqrt{n} \times \sqrt{n}$ KLT's, as shown in Fig. 5. The SKLT treats an n-dimensional

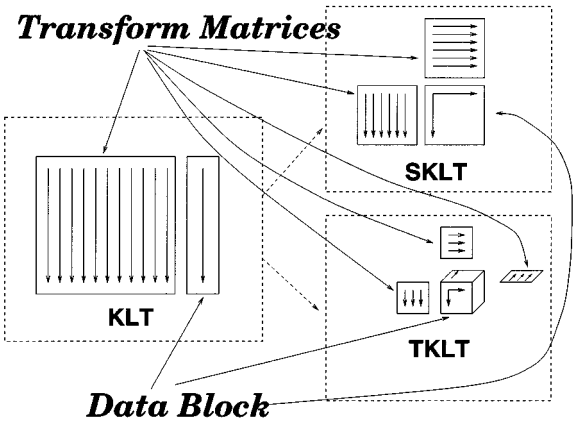


Fig. 5. Decomposition of KLT to SKLT and TKLT.

data vector as a $\sqrt{n} \times \sqrt{n}$ data matrix. (This blocking is natural in images, where the data vector may be a square block of pixels.) The SKLT then uses KLT L to decorrelate the data column by column and KLT R to decorrelate the data row by row. The transform is implemented as $Y = LXR$, where X is the $\sqrt{n} \times \sqrt{n}$ data matrix and Y the transform output. Together, the two $\sqrt{n} \times \sqrt{n}$ transforms approximate the action of a single $n \times n$ KLT while requiring only $2/\sqrt{n}$ times as many multiplications. More precisely, operation of the SKLT (including the pair of matrix multiplications) requires $2n^{3/2}$ multiplications while straight forward implementations of the KLT require n^2 multiplications. Furthermore, each SKLT has only $2n$ coefficients total, while the KLT requires n^2 coefficients. Thus, for $n = 64$, the SKLT decreases the computational complexity and memory requirements of the KLT by factors of four and 32, respectively.

Since the statistics of the rows and columns of an image are typically similar, we can further reduce a KLT's storage requirements by using the same transformation on the rows and the columns (i.e., setting $L' = R$). The resulting code, called a single SKLT (SSKLT), requires the same computational complexity as the SKLT, but reduces the memory requirements by another factor of two.

The separation approach may also be applied in higher dimensions. For example, arranging a 64-dimensional data vector into a $4 \times 4 \times 4$ data cube and decorrelating each dimension independently results in a triple-separable KLT (TKLT). Fig. 5 illustrates the structure of the TKLT. For $n = 64$, the TKLT requires 3/16 as many multiplications as the corresponding KLT and uses 3/256 as much memory.

More generally, consider a vector of dimension $n = \prod_{i=1}^{p} n_i$, which is arranged as a p-dimensional hypercube. In the kth dimension ($k = 1, 2, \ldots, p$), an $n_k \times n_k$ transform matrix is needed to process $\left(\prod_{i=1}^{k-1} n_i\right)\left(\prod_{i=k+1}^{p} n_i\right)$ data vectors of dimension n_k, using $n_k n$ multiplications. In total, transformation of an n-dimensional vector using a p-separable KLT requires $n\sum_{i=1}^{p} n_i$ multiplications and storage of $\sum_{i=1}^{p} n_i^2$ transform coefficients. The memory and the number of multiplications achieve their minima when $n_i = \sqrt[p]{n}$ for all $1 \le i \le p$.

The balanced decomposition into a hypercube of size $\sqrt[p]{n}$ in each dimension is feasible when $\sqrt[p]{n}$ is an integer. In this case, the number of multiplications required equals $n\sum_{i=1}^{p} n_i = pn^{(1+p)/p}$ while the total number of coefficients required equals $\sum_{i=1}^{p} n^{2/p} = pn^{2/p}$. The storage requirements may be further reduced by using the same transformation for more than one dimension.

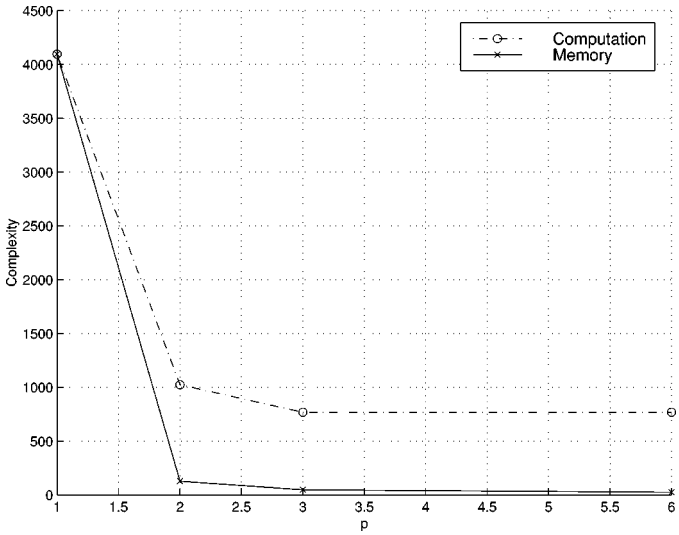


Fig. 6. Complexity and memory of a p-separable KLT for $n = 64$.

We bound the computational complexity by noting that $pn^{(1+p)/p} \ge en \ln n$ with equality if and only if $p = \ln n$. Thus the fast KLT can be made to approach the FFT's $O(n \log n)$ complexity. For the given choice of p, the memory consumption is $pn^{2/p} = e^2 \ln n$. Thus the memory consumption of this fast KLT is $O(\ln n)$, in contrast to the KLT's $O(n^2)$ memory requirements. Fig. 6 shows how the complexity and memory of a p-separable balanced KLT vary with p for the case where $n = 64$. The optimal decomposition for this example is either $p = 3$ or $p = 6$.

While the previous lower bound, which requires a noninteger p-value ($p = \ln n$), cannot be achieved, it can be approached very closely for carefully chosen values of n. This is achieved by fixing some constant ℓ and considering p-separable balanced KLT's when $n = \ell^p$ and p is allowed to grow without bound. In this case, operation of the KLT requires $\ell pn = \ell n \ln n / \ln \ell = O(n \ln n)$ complexity. This complexity is minimized for a given n when $\ell = 3$.

The savings in design complexity for balanced p-separable KLT's are even greater than the savings in their operation. The covariance matrix decomposition used in KLT design requires an average computational complexity of $o(n^3)$ [6] while p-separable balanced KLT design complexity for $n = \ell^p$ is $o\left(p\ell^3\right) = o(pn^{3/p})$.

While SKLT and TKLT are natural matches for two- and three-dimensional (3-D) data sets like images and video, we can find a p-separable approximation to the KLT of any dimension $n = \prod_{i=1}^{p} n_i$. Separable KLTs allow decorrelation only of those coefficients along a single dimension. For example, the SKLT yields decorrelation of values within each row and column of a data block but does not allow explicit decorrelation of coefficients along each diagonal. The more one separates a KLT, the more correlations are being neglected, and, thus, very high dimensional decompositions may give inaccurate estimates to the n-dimensional KLT.

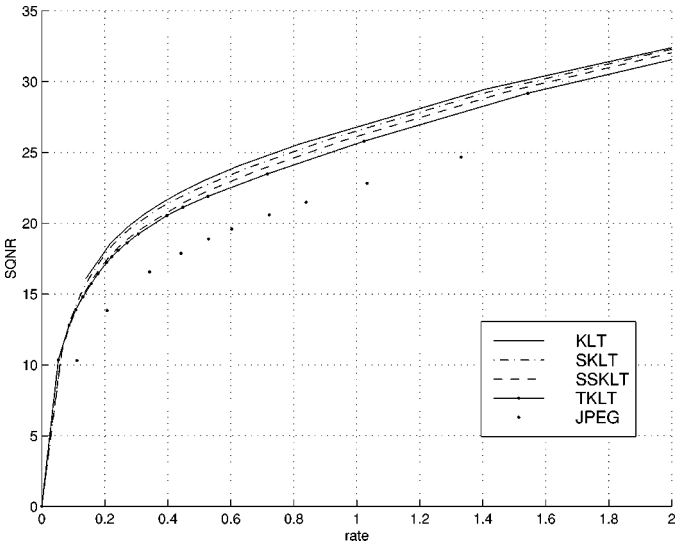
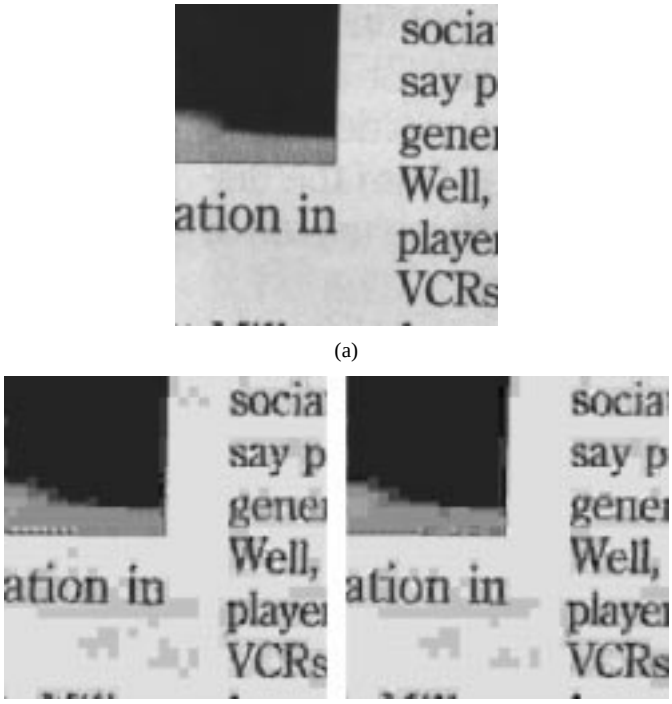


Fig. 7. SKLT, SSKLT, and TKLT transform coding performance compared to that of the KLT and DCT. The SKLT requires 25% of the complexity and 3% of the memory of the KLT. The TKLT requires 18.75% of the complexity and 1% of the memory of the KLT.

Fig. 7 compares the transform coding performance of the SKLT, SSKLT, and TKLT algorithms to that of the KLT. Also included in that graph are the results from the JPEG algorithm. The KLT-based codes differ from JPEG both in the transform applied and in the fact that the KLT-based algorithms incorporate multiple transforms and bit allocations. All use 8×8 blocks of pixels as their $\sqrt{n} \times \sqrt{n}$ data blocks. To arrange an 8×8 image block into a $4 \times 4 \times 4$ data cube, we break the image block into four 4×4 subblocks and then "stack" those blocks to form a cube, as shown in Fig. 8. All experiments use the image data set described in Section II. The SKLT, which employs the closest approximation of the KLT, achieves performance less than 0.4 dB worse than that of the KLT. Thus, the move from the true KLT to its SKLT approximation costs less than 0.4 dB and yields a $4 \times$ and $32 \times$ improvement in complexity and memory respectively. The SSKLT suffers a 0.4 dB performance degradation when compared to SKLT but requires half as much memory. The TKLT achieves performance less than 0.9 dB worse than that of the SKLT and 1.3 dB worse than that of the KLT. Thus the move from true KLT to its approximation TKLT costs more than 1 dB in rate-distortion performance. While this loss in performance is significant, the accompanying 81.25% reduction in complexity and 98.83% reduction in memory requirements make the TKLT an appropriate choice for some applications. All of the KLT-based codes give significant performance improvements over the JPEG algorithm.

Fig. 9 shows a portion of the original image and images compressed using KLT and SKLT.

Fig. 10 compares the performance of the KLT, SKLT, and TKLT on a video sequence, a good match for TKLT given its natural 3-D data structure. (For SKLT, we code the video as a sequence of images.) The video data is a "polyhedral scene" from [13]. We use the first 16 images as our training set and the last 15 images as the test set. In this plot, the performances of TKLT and SKLT are very close, with TKLT achieving better performance than SKLT (up to 1.5 dB gains) at low rates. This



(a) (b) (c)

Fig. 9. (a) Test image with its rate-0.0885 bps reproductions using (b) KLT and (c) SKLT.

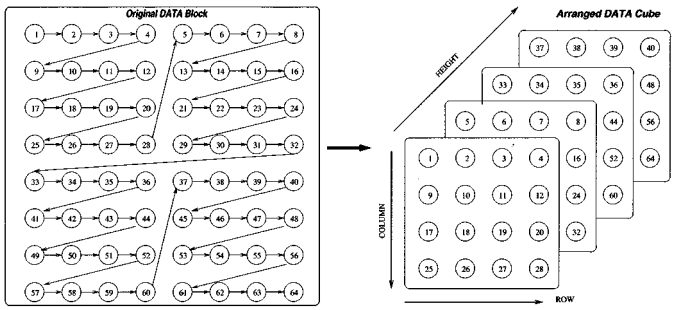


Fig. 8. Blocking technique used for TKLT on image data.

performance improvement is achieved using only 75% of the multiplications and 37.5% of the memory required by SKLT. The improvement likely results from the TKLT's ability to take advantage of the high correlation between frames of the video sequence. The performance improvements over KLT are even greater, giving up to 2.3 dB gains using only 18.75% of the multiplications and 1.17% of the memory of KLT.

IV. Summary and Conclusion

This paper investigates the rate-distortion performance penalties and complexity tradeoffs associated with using the KLT and its fast approximations in transform coding. Since the optimal transform for transform coding is unknown in general, performance penalties relative to the optimal transform are estimated by first examining examples where the KLT can fail, then developing a transform that corrects those shortcomings but agrees with the KLT for examples where the KLT gives good performance, and finally comparing the performance of that new transform to that of the KLT in compressing natural data sets. While

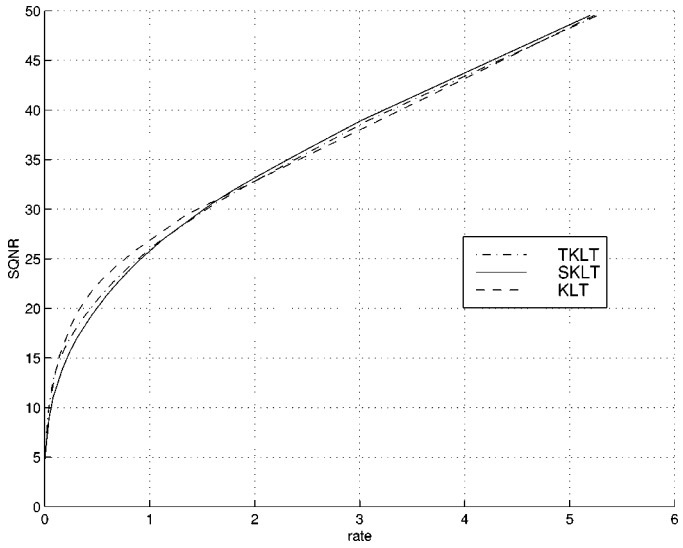


Fig. 10. TKLT performance compared with KLT and SKLT performance on the video data set.

the worst KLT yields 3 dB loss relative to the optimal transform for some sources, the penalty for image data sets seems to be smaller, with observed examples of up to 0.5 dB performance loss relative to the improved transform. The improved transforms require computational complexity and memory identical to those of the KLT.

This paper also investigates the transform coding performance penalties associated with replacing the KLT with a variety of fast variations. The SKLT reduces the complexity of the KLT by a factor of four and the memory requirements by a factor of 32. Application of the SKLT in place of the KLT within the KLT framework yields experimental image coding performance within 0.4 dB of the performance achieved by the KLT.

While the questions of KLT performance and speed are treated separately in this work, the improved transforms of Section II and the fast transforms of Section III can be combined to yield better fast transforms. By such a combination, we achieve the performance benefits of the improved transforms with the savings in computational complexity and memory of the fast algorithms. This joint optimization can be accomplished by performing the optimization of Section II independently on each dimension of a data block or cube like those used in Section III.

Appendix A
Analysis for the Square and Diamond Distributions

Using a P-step uniform scalar quantizer on each dimension of the square distribution gives total distortion $D_U = 2P\int_{-1/P}^{1/P}(1/2)u^2du = 2/(3P^2)$. The entropies for U_1 and U_2 are both $\log P$.

The diamond distribution's marginals are $g_X(x) = (1/2)\left(\sqrt{2}-|x|\right)$ for $x \in \left[-\sqrt{2},\sqrt{2}\right]$. Using a $(P=2M)$-step uniform scalar quantizer gives total distortion

$$D_X = 2\sum_{i=1}^{M}\int_{(i-1)\sqrt{2}/M}^{i(\sqrt{2}/M)}\left[x-\left(i-\frac{1}{2}\right)\frac{\sqrt{2}}{M}\right]^2\left(\sqrt{2}-x\right)dx = \frac{1}{3M^2} = \frac{4}{3P^2}.$$

For any $M+1 \le i \le 2M$, the probability P_i for the ith bin $\left[(i-1)\sqrt{2}/M-\sqrt{2}, i\sqrt{2}/M-\sqrt{2}\right]$ is

$$P_i = \int_{-\sqrt{2}+(i-1)\sqrt{2}/M}^{-\sqrt{2}+i\sqrt{2}/M}\frac{1}{2}\left(\sqrt{2}-|x|\right)dx = \frac{4M+1-2i}{2M^2}.$$

Thus, the rate of each dimension, measured as entropy, equals

$$R_X = 1+2\log M - \frac{\sum_{i=2}^{M}(2i-1)\log(2i-1)}{M^2}.$$

Similarly, a $(P=2M+1)$-step uniform quantizer gives total distortion

$$D_X = 2\frac{8M^2+8M+1}{3(2M+1)^4} = \frac{4P^2-2}{3P^4}.$$

The bins $\left[(i-1)2\sqrt{2}/(2M+1)-\sqrt{2}, i2\sqrt{2}/(2M+1)-\sqrt{2}\right]$, $i=1,\ldots,2M+1$, have probabilities

$$P_{M+1} = \int_{-\sqrt{2}/(2M+1)}^{\sqrt{2}/(2M+1)}\frac{1}{2}(\sqrt{2}-|x|)dx = \frac{4M+1}{(2M+1)^2}$$

and, for $i=1,2,\ldots,M$,

$$P_i = \int_{2(i-1)\sqrt{2}/(2M+1)-\sqrt{2}}^{2i\sqrt{2}/(2M+1)-\sqrt{2}}\frac{1}{2}\left(\sqrt{2}+x\right)dx = \frac{2(2i-1)}{(2M+1)^2}.$$

The resulting rate of each dimension is

$$R_X = 2\log(2M+1) - \frac{4M^2+(4M+1)\log(4M+1)}{(2M+1)^2} + \frac{4\sum_{i=2}^{M}(2i-1)\log(2i-1)}{(2M+1)^2}.$$

Appendix B
Rotation Matrix Optimization

The algorithm of Section II gives an iterative technique for approximating iKLT A^* through the sequential optimization of rotation angles $\{\theta_{i,j}\}$. The procedure for finding the $\theta_{i,j}^*$ works as follows.

Recall from Section II that finding the extremum of $F_4\left(X_{i,j}^n\right) = F_4\left(J_{i,j}X_{k,l}^n\right)$ for fixed statistics on $X_{k,l}^n$ is equivalent to finding the extremum of

$$\begin{aligned}
&E\left[(X_{k,l,i}\cos\theta_{i,j}+X_{k,l,j}\sin\theta_{i,j})^4\right.\\
&\quad\left.+(-X_{k,l,i}\sin\theta_{i,j}+X_{k,l,j}\cos\theta_{i,j})^4\right]\\
&= E\left[(X_{k,l,i})^4+(X_{k,l,j})^4\right]\cos^4\theta_{i,j}\\
&\quad+4E\left[(X_{k,l,i})^3X_{k,l,j}-X_{k,l,i}(X_{k,l,j})^3\right]\cos^3\theta_{i,j}\sin\theta_{i,j}\\
&\quad+6E\left[2(X_{k,l,i})^2(X_{k,l,j})^2\right]\cos^2\theta_{i,j}\sin^2\theta_{i,j}\\
&\quad+4E\left[X_{k,l,i}(X_{k,l,j})^3-(X_{k,l,i})^3X_{k,l,j}\right]\cos\theta_{i,j}\sin^3\theta_{i,j}\\
&\quad+E\left[(X_{k,l,j})^4+(X_{k,l,i})^4\right]\sin^4\theta_{i,j}\\
&= E_1\cos^4\theta_{i,j}+E_1\sin^4\theta_{i,j}+4E_2\cos^3\theta_{i,j}\sin\theta_{i,j}\\
&\quad+6E_3\cos^2\theta_{i,j}\sin^2\theta_{i,j}-4E_2\cos\theta_{i,j}\sin^3\theta_{i,j}\\
&= E_1+E_2\sin(4\theta_{i,j})+\frac{(3E_3-E_1)(1-\cos(4\theta_{i,j}))}{4}\\
&= \frac{3(E_1+E_3)}{4}+\frac{\sqrt{16E_2^2+(E_1-3E_3)^2}}{4}\sin(4\theta_{i,j}+\phi).
\end{aligned}$$

Here

$$\begin{aligned}E_1 &= E\left[(X_{k,l,i})^4 + (X_{k,l,j})^4\right]\\ E_2 &= E\left[(X_{k,l,i})^3 X_{k,l,j} - X_{k,l,i}(X_{k,l,j})^3\right]\\ E_3 &= 2E\left[(X_{k,l,i})^2 (X_{k,l,j})^2\right]\\ \phi &= \arctan\left(\frac{E_1 - 3E_3}{4E_2}\right) + \pi 1(E_2 < 0).\end{aligned}$$

Since E_1, E_2, E_3, and ϕ do not vary with $\theta_{i,j}$,

$$\theta^*_{i,j} = \begin{cases} \arg\min_\theta \sin(4\theta + \phi), & \text{if } \Delta > 0\\ \arg\max_\theta \sin(4\theta + \phi), & \text{if } \Delta < 0.\end{cases}$$

Finding $\theta^*_{i,j}$ requires knowledge of the fourth order statistics of $X^n_{i,j}$. The following equations relate the statistics of $X^n_{i,j}$ to those of $X^n_{k,l}$ to allow for easy updating of the fourth order statistics. For notational simplicity, let

$$\begin{aligned}E_{p,q,r,s} &= E[X_{i,j,p}X_{i,j,q}X_{i,j,r}X_{i,j,s}]\\ E'_{p,q,r,s} &= E[X_{k,l,p}X_{k,l,q}X_{k,l,r}X_{k,l,s}]\end{aligned}$$

for $p, q, r, s \in \{1, 2, \ldots, n\}$. If the values of E' are known from the previous step, then we calculate E as follows. Suppose $t, v, w, z \in \{1, 2, \ldots, n\} - \{i, j\}$, then

$$\begin{aligned}E_{t,v,w,z} &= E'_{t,v,w,z}\\ E_{i,v,w,z} &= E'_{i,v,w,z}\cos\theta_{i,j} + E'_{j,v,w,z}\sin\theta_{i,j}\\ E_{j,v,w,z} &= -E'_{i,v,w,z}\sin\theta_{i,j} + E'_{j,v,w,z}\cos\theta_{i,j}\\ E_{i,i,w,z} &= E'_{i,i,w,z}\cos^2\theta_{i,j} + E'_{i,j,w,z}\sin 2\theta_{i,j}\\ &\quad + E'_{j,j,w,z}\sin^2\theta_{i,j}\\ E_{j,j,w,z} &= E'_{i,i,w,z}\sin^2\theta_{i,j} - E'_{i,j,w,z}\sin 2\theta_{i,j}\\ &\quad + E'_{j,j,w,z}\cos^2\theta_{i,j}\\ E_{i,j,w,z} &= -E'_{i,i,w,z}\cos\theta_{i,j}\sin\theta_{i,j} + E'_{i,j,w,z}\cos 2\theta_{i,j}\\ &\quad + E'_{j,j,w,z}\cos\theta_{i,j}\sin\theta_{i,j}\\ E_{i,i,i,z} &= E'_{i,i,i,z}\cos^3\theta_{i,j} + 3E'_{i,i,j,z}\cos^2\theta_{i,j}\sin\theta_{i,j}\\ &\quad + 3E'_{i,j,j,z}\cos\theta_{i,j}\sin^2\theta_{i,j} + E'_{j,j,j,z}\sin^3\theta_{i,j}\\ E_{j,j,j,z} &= -E'_{i,i,i,z}\sin^3\theta_{i,j} + 3E'_{i,i,j,z}\cos\theta_{i,j}\sin^2\theta_{i,j}\\ &\quad - 3E'_{i,j,j,z}\cos^2\theta_{i,j}\sin\theta_{i,j} + E'_{j,j,j,z}\cos^3\theta_{i,j}\\ E_{i,i,j,z} &= -E'_{i,i,i,z}\cos^2\theta_{i,j}\sin\theta_{i,j}\\ &\quad + E'_{i,i,j,z}\left(\cos^3\theta_{i,j} - 2\cos\theta_{i,j}\sin^2\theta_{i,j}\right)\\ &\quad + E'_{i,j,j,z}\left(2\cos^2\theta_{i,j}\sin\theta_{i,j} - \sin^3\theta_{i,j}\right)\\ &\quad + E'_{j,j,j,z}\cos\theta_{i,j}\sin^2\theta_{i,j}\\ E_{i,j,j,z} &= E'_{i,i,i,z}\cos\theta_{i,j}\sin^2\theta_{i,j}\\ &\quad + E'_{i,i,j,z}\left(\sin^3\theta_{i,j} - 2\cos^2\theta_{i,j}\sin\theta_{i,j}\right)\\ &\quad + E'_{i,j,j,z}\left(\cos^3\theta_{i,j} - 2\cos\theta_{i,j}\sin^2\theta_{i,j}\right)\\ &\quad + E'_{j,j,j,z}\cos^2\theta_{i,j}\sin\theta_{i,j}\\ E_{i,i,i,i} &= E'_{i,i,i,i}\cos^4\theta_{i,j} + 4E'_{i,i,i,j}\cos^3\theta_{i,j}\sin\theta_{i,j}\\ &\quad + 6E'_{i,i,j,j}\cos^2\theta_{i,j}\sin^2\theta_{i,j}\\ &\quad + 4E'_{i,j,j,j}\cos\theta_{i,j}\sin^3\theta_{i,j} + E'_{j,j,j,j}\sin^4\theta_{i,j}\\ E_{j,j,j,j} &= E'_{i,i,i,i}\sin^4\theta_{i,j} - 4E'_{i,i,i,j}\cos\theta_{i,j}\sin^3\theta_{i,j}\\ &\quad + 6E'_{i,i,j,j}\cos^2\theta_{i,j}\sin^2\theta_{i,j}\\ &\quad - 4E'_{i,j,j,j}\cos^3\theta_{i,j}\sin\theta_{i,j} + E'_{j,j,j,j}\cos^4\theta_{i,j}\\ E_{i,i,i,j} &= -E'_{i,i,i,i}\cos^3\theta_{i,j}\sin\theta_{i,j}\\ &\quad + E'_{i,i,i,j}\left(\cos^4\theta_{i,j} - 3\cos^2\theta_{i,j}\sin^2\theta_{i,j}\right)\\ &\quad + 3E'_{i,i,j,j}\left(\cos^3\theta_{i,j}\sin\theta_{i,j} - \cos\theta_{i,j}\sin^3\theta_{i,j}\right)\\ &\quad + E'_{i,j,j,j}\left(3\cos^2\theta_{i,j}\sin^2\theta_{i,j} - \sin^4\theta_{i,j}\right)\\ &\quad + E'_{j,j,j,j}\cos\theta_{i,j}\sin^3\theta_{i,j}\\ E_{i,j,j,j} &= -E'_{i,i,i,i}\cos\theta_{i,j}\sin^3\theta_{i,j}\\ &\quad + E'_{i,i,i,j}\left(3\cos^2\theta_{i,j}\sin^2\theta_{i,j} - \sin^4\theta_{i,j}\right)\\ &\quad + 3E'_{i,i,j,j}\left(\cos\theta_{i,j}\sin^3\theta_{i,j} - \cos^3\theta_{i,j}\sin\theta_{i,j}\right)\\ &\quad + E'_{i,j,j,j}\left(\cos^4\theta_{i,j} - 3\cos^2\theta_{i,j}\sin^2\theta_{i,j}\right)\\ &\quad + E'_{j,j,j,j}\cos^3\theta_{i,j}\sin\theta_{i,j}\\ E_{i,i,j,j} &= \left(E'_{i,i,i,i} + E'_{j,j,j,j}\right)\cos^2\theta_{i,j}\sin^2\theta_{i,j}\\ &\quad + E'_{i,i,j,j}\left(\cos^4\theta_{i,j} - 4\cos^2\theta_{i,j}\sin^2\theta_{i,j}\right.\\ &\qquad \left. + \sin^4\theta_{i,j}\right) + 2\left(E'_{i,i,i,j} - E'_{i,j,j,j}\right)\\ &\quad \cdot\left(\cos\theta_{i,j}\sin^3\theta_{i,j} - \cos^3\theta_{i,j}\sin\theta_{i,j}\right).\end{aligned}$$

ACKNOWLEDGMENT

The authors are grateful to the anonymous reviewers for their detailed suggestions and comments.

REFERENCES

[1] W. B. Pennebaker and J. L. Mitchell, *JPEG Still Image Data Compression Standard*. New York: Van Nostrand Reinhold, 1993.
[2] A. Said and W. A. Pearlman, "A new, fast, and efficient image codec based on set partitioning in hierarchical trees," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 6, pp. 243–250, June 1996.
[3] A. Koschman, "On the filtering of nonstationary time series," in *Proc. 1954 Electron. Conf.*, 1954, p. 126.
[4] P. P. Vaidyanathan, *Multirate Systems and Filter Banks*. Englewood Cliffs, NJ: Prentice-Hall, 1993.
[5] P. Noll and R. Zelinski, "Bounds on quantizer performance in the low bit-rate region," *IEEE Trans. Commun.*, vol. 26, pp. 300–304, Feb. 1978.
[6] W. H. Press, *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge, U.K.: Cambridge Univ. Press, 1992.
[7] C. Archer and T. K. Leen, "The coding-optimal transform," in *Proc. Data Compression Conf.*, Snowbird, UT, Mar. 2001, pp. 381–390.
[8] M. Effros, "Universal and Adaptive Source Coding: Theory and Practice," Ph.D. dissertation, Stanford Univ., Stanford, CA, 1994.
[9] M. Effros and P. A. Chou, "Weighted universal transform coding: Universal image compression with the Karhunen–Loève transform," in *Proc. IEEE Int. Conf. Image Processing*, Washington, DC, Oct. 1995.
[10] M. Effros, P. A. Chou, and R. M. Gray, "Weighted universal image compression," *IEEE Trans. Image Processing*, vol. 8, pp. 1317–1329, Oct. 1999.
[11] P. L. Dragotti, G. Poggi, and A. R. P. Ragozini, "Compression of multispectral images by three-dimensional SPIHT algorithm," *IEEE Trans. Geosci. Remote Sensing*, vol. 38, pp. 416–428, Jan. 2000.
[12] A. K. Jain, *Fundamentals of Digital Image Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1989.
[13] M. Otte. Polyhedral scene with a moving marbled block and moving camera. [Online]. Available: http://i21www.ira.uka.de/cgi-bin/download?tr_bsb

Hanying Feng (S'99) received the B.S. degree in electronics engineering from Tsinghua University, Beijing, China, in 1997 and the M.S. degree in electrical engineering from the California Institute of Technology, Pasadena, in 1998, where he is currently pursuing the Ph.D. degree in electrical engineering.

His research interests include linear transforms, data compression, and information theory.

Michelle Effros (S'93–M'95) received the B.S. degree (with distinction) in 1989, the M.S. degree in 1990, and the Ph.D. degree in 1994, all in electrical engineering, from Stanford University, Stanford, CA.

During the summers of 1988 and 1989, she was with Hughes Aircraft Company. She joined the faculty at the California Institute of Technology, Pasadena, in 1994 and is currently an Associate Professor of electrical engineering. Her research interests include information theory, data compression, communications, pattern recognition, and image processing.

Dr. Effros received Stanford's Frederick Emmons Terman Engineering Scholastic Award (for excellence in engineering) in 1989, the Hughes Masters Full-Study Fellowship in 1989, the National Science Foundation Graduate Fellowship in 1990, the AT&T Ph.D. Scholarship in 1993, the NSF CAREER Award in 1995, the Charles Lee Powell Foundation Award in 1997, the Richard Feynman-Hughes Fellowship in 1997, and an Okawa Research Grant in 2000. She served as advisor on the Capocelli Prize winning paper in 2001. She is a member of Tau Beta Pi, Phi Beta Kappa, Sigma Xi, and the IEEE Information Theory, Signal Processing, and Communications societies. She served as the Editor of the *IEEE Information Theory Society Newsletter* from 1995 to 1998 and has been a member of the Board of Governors of the IEEE Information Theory Society since 1998 and the IEEE Signal Processing Society Image and Multidimensional Signal Processing (IMDSP) Technical Committee since 2001.