Padding Techniques for Transforms with Arbitrary Spatial Support

Timothy B. Terriberry, *Student Member, IEEE*

***Abstract*— The coding of arbitrarily shaped images and video has become an increasingly important problem, but the design of simple and efficient transforms with arbitrary spatial support remains difficult. Fixed-size transforms, such as the $N \times N$ DCT, however, are mature and well understood. We propose a simple padding framework that allows a fixed-size unitary transform such as the DCT to be used without modification over arbitrary regions. We develop several algorithms inside this framework, compare them, and identify one superior approach. Our methods are computationally efficient, can take advantage of existing hardware, and impose no additional overhead on the decoder.**

***Index Terms*— Arbitrary spatial support, shape-adaptive DCT, padding.**

I. Introduction

SHAPE ADAPTIVE transforms are required for object-based video compression, where separate objects in a scene are coded individually, along with a description defining their shape. This preserves sharp edges between objects without introducing ringing and provides additional functionality that frame-based video compression cannot. However, even in the frame-based case, encoding a rectangular image which is not a multiple of the block size in a block-based codec requires special handling of the partial blocks.

A. Shape Adaptive Transforms

One class of solutions to the arbitrary shape problem is to design a family of transforms that can be applied to every conceivable shape. These include methods developed by Gilge *et al*., Sikora and Makai, and Strasiński and Konrad [1]–[3]. They are typically characterized by equal complexity in the encoder and decoder, a complexity greater than that of the unmodified $N \times N$ DCT.

Gilge *et al*.'s method starts with an orthogonal set of basis functions over a rectangular region, restricts them to the support region of the current shape, and then uses a Gram-Schmidt procedure to re-orthogonalize them [1]. This requires both large storage and high computational complexity, but provides very good compression.

The original method uses monomials $x^i y^j$ as basis functions and requires $O(b^2 m)$ operations and $O(bm)$ storage, where b is the number of basis functions computed and m is the size of the region. At low bitrates b can be small, but at higher ones it might be larger than m, as some of the computed basis functions are linearly dependent on the previous ones and must be discarded. These computational requirements were later reduced by Philips to $O(b^{3/2} m)$ operations and $O(\sqrt{b} m)$ storage [4], still for the special case of monomials.

Sikora and Makai propose the SA-DCT algorithm [2]. This moves the samples inside the shape to the beginning of each row and applies an N_i-point DCT to each row, where N_i is the number of samples in row i. The process is then repeated on columns. The order can reversed, processing columns first and then rows, which yields different results. Further modifications ensure the transform is orthogonal and DC preserving [5], [6]. The SA-DCT requires an implementation of a DCT for every possible row size, and can distort signal statistics by mixing coefficients of different frequencies in the column transforms. This last will be discussed in more detail later.

Strasiński and Konrad developed a method for converting any orthogonal transform into a shape-adaptive transform [3]. However, their algorithm must choose between four different options at each butterfly step in the transform. When implementing this in hardware, it requires extra silicon to implement all four options, yielding a complexity the authors estimate to be about twice that of a normal DCT. A software implementation requires conditional branches, which can be very expensive on modern general purpose processors if mispredicted due to their long pipelines.

B. Shape Adaptive Padding

A more flexible class of approaches is to use a standard block transform such as the $N \times N$ DCT in the decoder, and simply discard all the pixels outside of the region of support. The encoder is then free to pad the block with any values that make the resulting transform coefficients easier to compress. This allows the encoder to make the trade-off between good compression and computational complexity according to its requirements, and allows the decoder to use a single optimized transform for all blocks, taking advantage of any standard hardware implementations.

The simplest choice of padding is to use a constant value such as 0. This is experimentally shown to produce poor results [7]. However, it may still be useful for inter frames, where the expected value of the residual is small [8]. Techniques such as mirror extension are intuitively appealing, but are hard to generalize to non-convex shapes and can yield different results depending on whether rows or columns are mirrored first. They also do not take full advantage of the signal statistics when only a few padding values need to be added, as only pixels near the edge of the shape are mirrored.

This work was undertaken for the Xiph.org Foundation.

The method adopted by MPEG-4 TMN11 initializes the padding with a constant value and then iteratively runs a small, low-pass filter over the padding region [8]. On the edge of the padding region, the support of the filter contains non-padding pixels, which helps reduce the sharp discontinuities introduced by using a constant value alone. Again, the full signal statistics are not used, as only the edge pixels—those most likely to contain outliers from a segmentation algorithm—contribute to the padding values. This approach reduces the magnitude of high-frequency coefficients, but there may be more non-zero coefficients in the transformed block than there were pixels in the original shape.

Several approaches have been proposed which attempt to force one coefficient to zero for each padding pixel added, producing a critically sampled transform. This amounts to selecting a subset of the full transform's basis functions and solving a linear system for the padding values that will force all other basis functions to zero.

Kaup and Aach propose a method that selects the basis functions by successive approximation [9]. The basis is built up by adding at each step the one function that reduces the residual error by the largest amount, stopping when the total error reaches a threshold. In order to evaluate the residual error at each iteration, the coefficients of the selected basis functions are obtained by solving a system of Gaussian normal equations. This can be done using the Cholesky decomposition, which can be constructed by adding one row at a time as the basis grows. Thus the cost of a single Cholesky decomposition is amortized across the entire process. However, this still requires time and space similar to that of the unoptimized version of Gilge *et al*.'s method. This is acceptable when encoding still images, for which the algorithm was originally designed, but is too expensive for video.

Chen *et al*. propose an iterative method based on the theory of Projection Onto Convex Sets (POCS) [10]. Initial padding values are chosen with some simple method, and a forward DCT is applied to the entire block. The most significant low-frequency coefficients of the result are kept, and the rest are set to zero. An inverse DCT is applied, and the result is used as padding values for the next iteration, after replacing the non-padding pixels with their original values. The process stops after some small number of iterations, or if the reconstruction fails to improve.

This method avoids solving a linear system by using standard transforms which may be heavily optimized or implemented in hardware instead. The difficulty with this method is that convergence is only guaranteed if the same coefficients are zeroed in every iteration, and there is no guarantee that the non-zero coefficients selected in the first iteration even correspond to linearly independent basis functions when restricted to the support region.

Shen *et al*. propose a technique of re-arranging each row so that padding pixels can be interleaved at fixed locations, chosen based on the number of non-padding pixels in the row [11]. Then a small linear system can be solved to force the highest frequency coefficients of a 1-D DCT applied to each row to zero. The choice of padding pixel locations ensures that the first N_i DCT basis functions are linearly independent when restricted to the non-padding region. Although the process can then be repeated on columns, Shen *et al*. report better energy compaction by only adding padding where an entire row is zero, even though this may result in more than m non-zero coefficients.

Like the SA-DCT, the algorithm may operate on columns first, with different results. Shen *et al*. compress each frame with both orders, and keep the one with the best rate-distortion characteristics. Also like the SA-DCT, the re-arrangements can distort signal statistics, which likely explains the improvement from limiting the shape-adaptiveness of the column transforms. Finally, this method requires modification to the decoder to read the order of row and column transforms from the bitstream and to restore the pixels to their original locations. We wish the decoder to be agnostic to the padding mechanism used, to retain the flexibility of the encoder to choose whatever method meets its requirements.

II. Padding Framework

We develop a new set of shape-adaptive padding algorithms based on the technique of selecting a subset of the available basis functions of a complete $N \times N$ unitary transform. The thesis is that, since zeros are easy to compress, we will increase compression efficiency by forcing as many coefficients as possible to zero. By choosing a subset that is linearly independent over the support region and critically sampled, the basis vectors selected uniquely determine the padding values required to force the remaining coefficients to zero. Although this goal is easy to express and design algorithms for, its optimality in the rate-distortion sense is currently unknown.

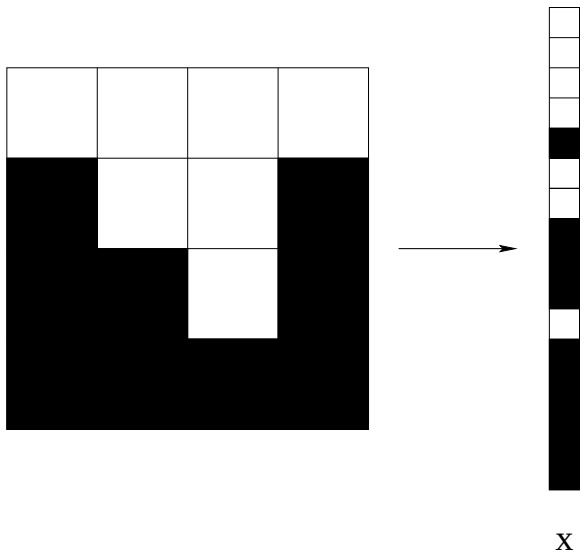

Fig. 1. Representing a 4×4 region as a column vector. Dark pixels belong to the support region, while white pixels are padding.

This section describes the formal framework in which these methods are developed. We begin with some notation. Consider a unitary matrix $\mathbf{G}$ of size $n \times n$, where $n = N^2$. That is, $\mathbf{y} = \mathbf{G}\mathbf{x}$ describes a complete 2-D transform, such as the DCT. The input values are arranged in a single column-

vector $\mathbf{x}$ in row-major order (see Figure 1), as are the output values $\mathbf{y}$.

Let $\mathbf{P}$ be a permutation matrix applied to the input vector $\mathbf{x}$ such that it can be partitioned into a non-padding part $\mathbf{u}$ of size $m \leq n$ and a padding part $\mathbf{v}$ of size $n-m$, i.e., $\mathbf{Px} = \left[\begin{smallmatrix}\mathbf{u}\\ \mathbf{v}\end{smallmatrix}\right]$. Now let $\mathbf{Q}$ be a permutation matrix applied to the output vector $\mathbf{y}$ such that it can be partitioned into m potentially non-zero coefficients $\mathbf{u}'$ and $n-m$ coefficients $\mathbf{v}'$ that will be forced to zero. That is, $\mathbf{Qy} = \left[\begin{smallmatrix}\mathbf{u}'\\ \mathbf{v}'\end{smallmatrix}\right]$. Now, let $\tilde{\mathbf{G}} = \mathbf{QGP}^{-1}$ be the permuted transform, and partition $\tilde{\mathbf{G}}$ into blocks $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, and $\mathbf{D}$ as follows:

$$\begin{bmatrix}\mathbf{u}'\\ \mathbf{v}'\end{bmatrix} = \begin{bmatrix}\mathbf{A} & \mathbf{B}\\ \mathbf{C} & \mathbf{D}\end{bmatrix}\begin{bmatrix}\mathbf{u}\\ \mathbf{v}\end{bmatrix} \tag{1}$$

The shape, which is given as input, determines $\mathbf{P}$. The effect on the transform $\mathbf{G}$ is to partition its columns, as illustrated in Figure 2 for the same example shape used in Figure 1. We use a row-transform in this particular example because its block-diagonal shape makes it easier to see how rows and columns are moved around. In general, a complete 2-D transform which is non-zero everywhere is used.

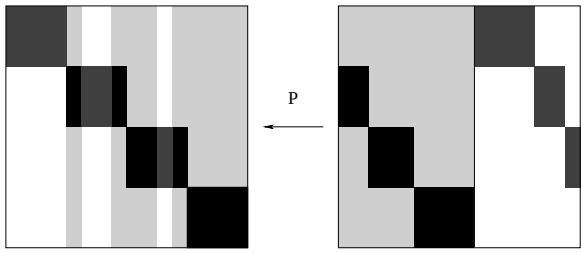

Fig. 2. The effect of $\mathbf{P}^{-1}$ on a row-transform $\mathbf{G}$. The dark blocks contain the non-zero coefficients, and the shaded columns correspond to the support region of the shape.

The basis functions we select to have non-zero coefficients determine $\mathbf{Q}$, and this is what we wish to identify. Its effect on the transform is to partition the rows, as illustrated in Figure 3. In general, the ordering within each partition is unimportant, so any two permutation matrices $\mathbf{P}$ and $\mathbf{P}'$ or $\mathbf{Q}$ and $\mathbf{Q}'$ which effect the same partitioning are equivalent.

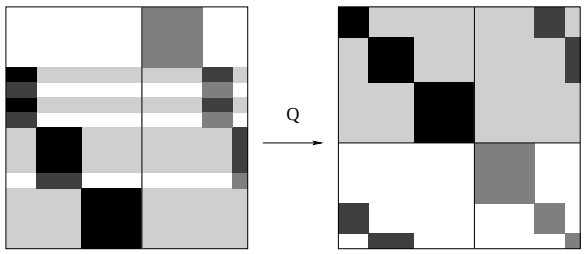

Fig. 3. The effect of $\mathbf{Q}$ on $\mathbf{GP^{-1}}$. The dark blocks contain the non-zero coefficients, and the shaded rows correspond to selected basis functions. The selection here is just one of the 24 possibilities for the given shape.

Take a moment to compare this with the strategy proposed by Shen *et al.* [11]. There the highest frequency coefficients are always forced to zero, thus fixing $\mathbf{Q}$. Instead, they vary $\mathbf{P}$ by mapping the input shape into a small set of predetermined shapes chosen only by the number of pixels in each row (or alternatively each column).

Our approach gives us the flexibility to choose a different set of basis functions for every potential shape. Thus, a shape with two pixels adjacent to each other is not treated the same as a shape with pixels on opposite ends of the block. We do not move distant, uncorrelated pixels close to each other, nor do we move neighboring, closely correlated pixels apart. Thus we can take full advantage of the underlying statistical assumptions of the image.

Our approach does allow some high frequency coefficients to be non-zero, but these are expected to remain small, as in the full transform case. Although low-frequency coefficients do not form long zero runs as often, forcing them to zero can still be very beneficial, as they are more likely to have a large magnitude which can take many bits to encode.

Following equation (1), we can express the transform coefficients as

$$\mathbf{u}' = \mathbf{Au} + \mathbf{Bv} \tag{2}$$

$$\mathbf{v}' = \mathbf{Cu} + \mathbf{Dv} \tag{3}$$

Substituting $\mathbf{0}$ for $\mathbf{v}'$ in equation (3) and solving for $\mathbf{v}$ yields:

$$\mathbf{v} = -\mathbf{D}^{-1}\mathbf{Cu} \tag{4}$$

We delay the question of whether or not $\mathbf{D}^{-1}$ exists for a brief moment. Substituting (4) into (2) gives an expression for the non-zero coefficients $\mathbf{u}'$ in terms of the non-padding pixels $\mathbf{u}$ only:

$$\mathbf{u}' = (\mathbf{A} - \mathbf{BD}^{-1}\mathbf{C})\mathbf{u} \tag{5}$$

The familiar expression in front of $\mathbf{u}$ is the Schur complement of $\mathbf{D}$, which we denote $\mathbf{S_D}$. It appears in the block matrix inversion formula:

$$\begin{bmatrix}\mathbf{A} & \mathbf{B}\\ \mathbf{C} & \mathbf{D}\end{bmatrix}^{-1} = \begin{bmatrix}\mathbf{S_D}^{-1} & -\mathbf{S_D}\mathbf{BD}^{-1}\\ -\mathbf{D}^{-1}\mathbf{CS_D}^{-1} & \mathbf{D}^{-1}+\mathbf{D}^{-1}\mathbf{CS_D}\mathbf{BD}^{-1}\end{bmatrix} \tag{6}$$

But since our transform is unitary, we know that $\tilde{\mathbf{G}}^{-1} = \tilde{\mathbf{G}}^T$, and so $\mathbf{S_D} = \mathbf{A}^{-T}$. This demonstrates that $\mathbf{A}$ is invertible so long as $\mathbf{D}$ is invertible. A similar argument with the Schur complement of $\mathbf{A}$ proves the converse.

Thus we wish to partition the truncated basis functions into two groups $\left[\begin{smallmatrix}\mathbf{A}\\ \mathbf{C}\end{smallmatrix}\right]$ in such a way that $\mathbf{A}$ is invertible, and $\mathbf{A}^{-T}$ has good energy compaction properties. Since the column vectors in $\left[\begin{smallmatrix}\mathbf{A}\\ \mathbf{C}\end{smallmatrix}\right]$ are orthogonal, it must always have full rank. Hence there must also be m linearly independent rows, ensuring at least one invertible $\mathbf{A}$ exists.

A complete padding algorithm in this framework is composed of two stages: selecting a set of basis functions and then padding so as to force the remaining coefficients to zero. Section III presents several solutions to the basis selection problem, under the assumption that the pixel values are given directly by solving the resulting linear system given in (5). Section IV gives a fast alternative to solving that system.

III. Basis Selection Strategies

If we began greedily adding basis functions that minimize the reconstruction error, we would obtain the method of Kaup

and Aach [9]. However, this presumes that we know the pixel values inside the shape and requires solving a large linear system for every transformed block. We present several different alternative algorithms.

A. Fixed-Shape Bases

The first approach is derived from the assumption that we can spend a long time computing a good basis to use for a particular shape, and then wish to apply the resulting transform to a large number of different input vectors, $\mathbf{u}$. For example, this can be used to extend a rectangular frame which is not a multiple of the transform block size out to an integral number of blocks. The method proposed here can be taken as a baseline against which to compare faster alternatives presented in the next section.

Shen *et al*. note that $\|\mathbf{u}'\|_2 \leq \|\mathbf{A}^{-T}\|_2\|\mathbf{u}\|_2$, where the $\|\cdot\|_2$ norm is the L_2 norm, also called the spectral norm for matrices. Since the magnitude of the output vector is bounded via the magnitude of $\mathbf{A}^{-T}$, they suggest minimizing this [11]. This is equivalent to maximizing the smallest singular value of $\mathbf{A}$. However, this only controls the worst-case performance of the transform. It cannot distinguish between two different choices which have the same smallest singular value.

We attempt instead to make the transform as close to orthogonal as possible, since non-orthogonal basis functions will carry redundant information. This is done by maximizing $\det(\mathbf{A}\mathbf{A}^T)$. Since we are considering the full 2-D transform at once, an exhaustive search could require as many as 1.8×10^{18} bases to be examined for a 32-pixel shape in an 8×8 block and is clearly impractical.

Instead we propose a greedy method. The basis is initialized with the truncated row corresponding to the DC coefficient. We then add the unselected basis vector with the largest component perpendicular to the subspace spanned by the previously selected vectors.

The above procedure can be formulated as an incremental Cholesky decomposition of $\mathbf{E} = \mathbf{A}\mathbf{A}^T$. The Cholesky decomposition factors a symmetric positive definite matrix $\mathbf{E}$ into the product $\mathbf{L}\mathbf{L}^T$, where $\mathbf{L}$ is lower-triangular. A matrix of the form $\mathbf{A}\mathbf{A}^T$ is positive definite so long as the rows of $\mathbf{A}$ are real and linearly independent. The closed-form expressions for the elements of $\mathbf{L}$ are:

$$l_{ii} = \sqrt{e_{ii} - \sum_{k=1}^{i-1} l_{ik}^2} \tag{7}$$

$$l_{ji} = \frac{1}{l_{ii}}\Big(e_{ji} - \sum_{k=1}^{i-1} l_{jk}l_{ik}\Big) \qquad j > i \tag{8}$$

Because the matrix is symmetric and positive definite, the expression under the square root is always positive.

In fact, the diagonal elements are precisely the terms we wish to maximize. l_{ii} is the magnitude of the component of row $\mathbf{a}_i$ that is perpendicular to the subspace spanned by all the rows $\mathbf{a}_j$, for $j < i$. Thus, we tentatively add each unselected row to the current $\mathbf{A}$, add the corresponding row and column to $\mathbf{E}$, and compute the new row that would be added to $\mathbf{L}$. From these, we choose the vector that maximizes the value of the next diagonal element of $\mathbf{L}$ to add to $\mathbf{A}$ permanently. When there are ties, we choose the vector of the coefficient that comes earlier in the standard zig-zag scanning order.

Note that the only divisions and square roots performed involve precisely the terms we are maximizing, and so the whole procedure is very numerically stable. It also yields intuitively appealing results for some special examples. When $\mathbf{G}$ is the $N\times N$ DCT and the support region is one quadrant of the block, all of the even DCT basis functions are selected, forming an $\frac{N}{2}\times\frac{N}{2}$ DCT. In this case the resulting transform is orthogonal, and the padding is precisely equivalent to that of the mirroring extension method.

The whole procedure requires $O(m^2n^2)$ time and $O(m^2)$ space, making it actually more computationally expensive than Kaup and Aach's method. But if it only needs to be done once up front, the actual application of the transform is less expensive. One possibility is to compute the appropriate padding values and then apply the original transform $\mathbf{G}$.

Using the fact that $\tilde{\mathbf{G}}^{-1} = \tilde{\mathbf{G}}^T$, we have

$$\begin{bmatrix}\mathbf{u}\\ \mathbf{v}\end{bmatrix} = \begin{bmatrix}\mathbf{A}^T & \mathbf{C}^T\\ \mathbf{B}^T & \mathbf{D}^T\end{bmatrix}\begin{bmatrix}\mathbf{u}'\\ \mathbf{0}\end{bmatrix} \tag{9}$$

Solving the lower half of this equation for $\mathbf{v}$ yields

$$\mathbf{v} = \mathbf{B}^T\mathbf{u}' = \mathbf{B}^T\mathbf{A}^{-T}\mathbf{u} \tag{10}$$

The Cholesky decomposition provides a convenient vehicle for computing the $\mathbf{A}^{-T}$ term:

$$\mathbf{v} = \mathbf{B}^T(\mathbf{A}\mathbf{A}^T)^{-1}\mathbf{A}\mathbf{u} = \mathbf{B}^T\mathbf{L}^{-T}\mathbf{L}^{-1}\mathbf{A}\mathbf{u} \tag{11}$$

The $\mathbf{L}^{-T}$ and $\mathbf{L}^{-1}$ terms are quickly computed using back substitution and forward substitution, due to the lower triangular nature of $\mathbf{L}$. The inverses are also very well conditioned, as the only divisions performed are by the diagonal elements of $\mathbf{L}$, which we maximized. All of the matrices may be multiplied out in advance, producing a single $(n-m)\times m$ matrix which computes the padding $\mathbf{v}$ from the input $\mathbf{u}$.

When the number of pixels in the region is small, it may be faster to compute the transform coefficients directly. Again, the Cholesky decomposition can be used to efficiently solve for the transform coefficients $\mathbf{u}'$ in terms of the input $\mathbf{u}$:

$$\mathbf{u}' = \mathbf{A}^{-T}\mathbf{u} = (\mathbf{A}\mathbf{A}^T)^{-1}\mathbf{A}\mathbf{u} = \mathbf{L}^{-T}\mathbf{L}^{-1}\mathbf{A}\mathbf{u} \tag{12}$$

The matrix in this expression is only $m\times m$, which may be better than the $(n-m)\times m$ matrix multiply required above. It also has the advantage that the original transform $\mathbf{G}$ does not need to be applied afterwards. However, the first method may still be better for a given shape if there are fast algorithms to compute $\mathbf{G}$.

Finally, another special case arises when the selected basis vectors are actually orthogonal. In this case, $\mathbf{L}$ is diagonal, and so

$$\mathbf{u}' = \mathbf{A}^{-T}\mathbf{u} = \mathbf{F}^{-2}\mathbf{A}\mathbf{u}, \tag{13}$$

where $\mathbf{F}$ is a diagonal matrix with entries equal to the magnitude of the rows of $\mathbf{A}$:

$$f_{ii} = \|\mathbf{a}_i^T\|_2 \tag{14}$$

This reduces the entire operation to padding with zeros, applying the original transform $\mathbf{G}$, and then scaling the output.

B. Fast Basis Selection

Although the computational complexity of the previous transform is manageable if the process of selecting the basis can be done once in advance, it is too expensive for a dynamically changing shape, except with very small transform sizes. This section demonstrates a more efficient method that selects a basis for each stage of a separable transform with a few table lookups. The price we pay for considering the stages separately is that we will no longer always be able to force exactly $n - m$ of the output coefficients to zero.

It is impractical to pre-compute a set of basis functions for every possible shape in a 2-D transform of any moderate size. If our transform is separable, we can reduce the number of possible shapes to something more tractable by considering each stage independently. The assumption of a separable transform is reasonable, since such transforms afford fast implementations, making them desirable on their own merits. Ideally, we would like to be able to select a basis for each stage of the transform so that we could force $n - m$ coefficients to zero after the second stage. We first demonstrate why this cannot be done and then propose an alternative.

Consider now a two-stage transform, $\mathbf{y} = \mathbf{G}\mathbf{x}$, $\mathbf{z} = \mathbf{H}\mathbf{y}$, where both $\mathbf{G}$ and $\mathbf{H}$ are unitary transforms. We also assume $\mathbf{G}$ is a *row transform*, e.g., $\mathbf{G}$ is block diagonal, with N equal $N \times N$ blocks, like the one shown in Figure 2. Similarly, it is assumed that there is a suitable permutation matrix $\mathbf{R}$ so that $\mathbf{R}\mathbf{H}\mathbf{R}^{-1}$ is also a row transform. For the DCT, $\mathbf{R}$ rearranges $\mathbf{y}$ and $\mathbf{z}$ so that they are indexed in column-major order instead of row-major order, e.g., $\mathbf{R}$ maps $Ni + j$ to $Nj + i$, for $i, j \in \{0 \dots N-1\}$.

Extending our previous notation, we now use four permutations $\mathbf{P}$, $\mathbf{Q}$, $\mathbf{S}$, and $\mathbf{T}$, to partition the input and output of each transform into padding and non-padding values. The permutation $\mathbf{R}$ is injected after the latter two so that we can operate on the block diagonal $\mathbf{R}\mathbf{H}\mathbf{R}^{-1}$. Let

$$\mathbf{P}\mathbf{x} = \begin{bmatrix}\mathbf{u}\\ \mathbf{v}\end{bmatrix} \qquad \mathbf{Q}\mathbf{y} = \begin{bmatrix}\mathbf{u}'\\ \mathbf{v}'\end{bmatrix} \tag{15a}$$

$$\mathbf{S}\mathbf{R}\mathbf{y} = \begin{bmatrix}\mathbf{u}'\\ \mathbf{v}'\end{bmatrix} \qquad \mathbf{T}\mathbf{R}\mathbf{z} = \begin{bmatrix}\mathbf{u}''\\ \mathbf{v}''\end{bmatrix} \tag{15b}$$

The total transform is thus:

$$\mathbf{y} = \mathbf{Q}^{-1}\tilde{\mathbf{G}}\mathbf{P}\mathbf{x} \tag{16}$$

$$\mathbf{z} = \mathbf{R}^{-1}\mathbf{T}^{-1}\tilde{\mathbf{H}}\mathbf{S}\mathbf{R}\mathbf{y}, \tag{17}$$

where $\tilde{\mathbf{G}} = \mathbf{Q}\mathbf{G}\mathbf{P}^{-1}$ and $\tilde{\mathbf{H}} = \mathbf{T}\mathbf{R}\mathbf{H}\mathbf{R}^{-1}\mathbf{S}^{-1}$.

Once more, the transforms $\tilde{\mathbf{G}}$ and $\tilde{\mathbf{H}}$ can be partitioned into blocks as follows:

$$\begin{bmatrix}\mathbf{u}'\\ \mathbf{v}'\end{bmatrix} = \begin{bmatrix}\mathbf{A}_\mathbf{G} & \mathbf{B}_\mathbf{G}\\ \mathbf{C}_\mathbf{G} & \mathbf{D}_\mathbf{G}\end{bmatrix}\begin{bmatrix}\mathbf{u}\\ \mathbf{v}\end{bmatrix} \tag{18}$$

$$\begin{bmatrix}\mathbf{u}''\\ \mathbf{v}''\end{bmatrix} = \begin{bmatrix}\mathbf{A}_\mathbf{H} & \mathbf{B}_\mathbf{H}\\ \mathbf{C}_\mathbf{H} & \mathbf{D}_\mathbf{H}\end{bmatrix}\begin{bmatrix}\mathbf{u}'\\ \mathbf{v}'\end{bmatrix} \tag{19}$$

Again, $\mathbf{P}$ is fixed, determined by the shape, and similarly $\mathbf{S}$ is also fixed as $\mathbf{S} = \mathbf{Q}\mathbf{R}^{-1}$. The permutations we are free to choose are $\mathbf{Q}$ and $\mathbf{T}$. However, observe what happens when we try to force $\mathbf{v}''$ to zero as before.

In this case, $\mathbf{v}'$ will not in general be zero, which requires us to revise our derivation of the output of the first stage, $\mathbf{u}'$ in terms of $\mathbf{u}$, given in equation (5). We must add a term to equation (4), giving

$$\mathbf{v} = \mathbf{D}_\mathbf{G}^{-1}\mathbf{v}' - \mathbf{D}_\mathbf{G}^{-1}\mathbf{C}_\mathbf{G}\mathbf{u} \tag{20}$$

Substituting (20) into (2) now produces

$$\mathbf{u}' = \mathbf{A}_\mathbf{G}^{-T}\mathbf{u} + \mathbf{B}_\mathbf{G}\mathbf{D}_\mathbf{G}^{-1}\mathbf{v}' \tag{21}$$

Replacing $\mathbf{u}$ and $\mathbf{v}$ with $\mathbf{u}'$ and $\mathbf{v}'$ in equation (10) and substituting that into (21) yields

$$\begin{aligned}\mathbf{u}' &= \mathbf{A}_\mathbf{G}^{-T}\mathbf{u} + \mathbf{B}_\mathbf{G}\mathbf{D}_\mathbf{G}^{-1}\mathbf{B}_\mathbf{H}^{T}\mathbf{A}_\mathbf{H}^{-T}\mathbf{u}'\\ &= (\mathbf{I} - \mathbf{B}_\mathbf{G}\mathbf{D}_\mathbf{G}^{-1}\mathbf{B}_\mathbf{H}^{T}\mathbf{A}_\mathbf{H}^{-T})^{-1}\mathbf{A}_\mathbf{G}^{-T}\mathbf{u}\end{aligned} \tag{22}$$

Finally, replacing $\mathbf{u}'$ with $\mathbf{u}''$ and $\mathbf{u}$ with $\mathbf{u}'$ in equation (5) and substituting the expression in equation (22) for $\mathbf{u}'$ gives us an equation for the entire transform:

$$\mathbf{u}'' = (\mathbf{A}_\mathbf{H}^{T} - \mathbf{B}_\mathbf{G}\mathbf{D}_\mathbf{G}^{-1}\mathbf{B}_\mathbf{H}^{T})^{-1}\mathbf{A}_\mathbf{G}^{-T}\mathbf{u} \tag{23}$$

Note that the first stage contains an inverse involving terms from both stages. In general, we cannot ensure that this inverse exists without considering both stages jointly. In fact, an initial implementation which ignored this fact frequently ran into cases where this matrix was singular, despite $\mathbf{A}_\mathbf{G}$, $\mathbf{D}_\mathbf{G}$, and $\mathbf{A}_\mathbf{H}$ all being well-conditioned.

One solution to this is to use the transform $\mathbf{u}'' = \mathbf{A}_\mathbf{H}^{-T}\mathbf{A}_\mathbf{G}^{-T}\mathbf{u}$ instead. This can be accomplished by forcing $\mathbf{v}'$ to zero in the first stage, and then holding $\mathbf{u}'$ fixed while solving for a new $\mathbf{v}'$ that forces $\mathbf{v}''$ to zero in the second stage. However, this would require a modification to the decoder to replace $\mathbf{v}'$ with zero after the first stage of the inverse transform. Such a modification is against our design principles.

Instead, we apply our algorithm to the first stage of the transform only, to force $\mathbf{v}'$ to zero. Then, we apply it again to the second stage, but this time we only consider a coefficient to be padding if the entire row it belongs to is padding in the initial shape. This is the only case where we can ensure that the decoder will discard any changes we make to the coefficients.

Shen *et al.* consider two similar approaches in their padding scheme, and also reject the first one [11]. However, their reasoning was that the coding performance of the second one was better, even though it was no longer a non-expansionist transform.

In order to implement a single padding stage, we take advantage of the block structure of $\mathbf{G}$ and $\mathbf{R}\mathbf{H}\mathbf{R}^{T}$. Due to this, we need only consider the input and output of a single block at a time. We construct a lookup table indicating which basis vectors are to be used for every possible shape. Since this is done in advance, we can use any metric we want to weigh the different possible bases. Our method from Section III-A, Shen *et al.*'s suggestion of maximizing the smallest singular value of $\mathbf{A}$, or even the coding gain under some statistical image model such as an $AR(1)$ process [12], all can be used to evaluate the different choices.

If all N blocks in each transform are the same, only one lookup table is needed for each stage. For a transform such as the DCT, where $\mathbf{G} = \mathbf{R}\mathbf{H}\mathbf{R}^{-1}$, we can even share the same

lookup table between stages. The total cost is thus 256 bytes and 128K of ROM for $N = 8$ and $N = 16$, respectively.

The entire process can also be reformulated to operate on columns first, instead of rows. Since the decoder operates identically regardless of the transform order in the encoder, we do not need to send any side information and so can make this choice on a block-by-block basis. It is possible to encode a block both ways and use the one that provides better rate-distortion performance. However, this is computationally expensive, so we propose a simpler scheme.

We choose the order that allows us to use more padding in the second stage. In the event of a tie, we select the order that yields a better transform according to our optimization criteria. We store the logarithm of the determinant, minimum singular value, or coding gain for each shape so that they may simply be added. The log of the minimum singular value is first scaled by the number of pixels in the shape to preserve their relative importance. Keeping these to 16 bits of precision requires an extra 512 bytes and 128K for $N = 8$ and $N = 16$ respectively. In our implementation with $N = 8$, we store them to twelve bits of precision, and use the special value `0x8000` for the empty shape. This lets us decide on the transform order simply by making 16 table lookups and adding the results for each direction.

This method also eliminates the need to solve one large linear system, and instead allows us to solve one small linear system for each row (resp. column), and then one additional small linear system for all of the columns (resp. rows). However, solving these linear systems is still very expensive compared to the normal cost of applying a linear transform in the non-shape-adaptive case. Even though the number of shapes we have to consider is now limited, for large transforms the cost of pre-computing and storing all the solutions in advance would be prohibitive.

The next section addresses the problem of finding a fast alternative to solving these systems. The method we adopt is an improvement of Chen *et al*.'s iterative POCS approximation method [10]. Since we will now consider each stage independently, we drop the more complicated notation of this section and revert to the simpler notation of Section III-A.

IV. POCS-Based Transform

First we present a review of the POCS-based method of Chen *et al*. [10]. The idea is to find a point in the intersection of two convex sets. The first, $C_1 = \left\{ \left[\begin{smallmatrix} \mathbf{u} \\ \mathbf{v} \end{smallmatrix} \right] : \mathbf{u} = \mathbf{u}^* \right\}$, contains some fixed pixel values $\mathbf{u}^*$ padded with some arbitrary values $\mathbf{v}$. The second set, $C_2 = \left\{ \tilde{\mathbf{G}}^{-1} \left[\begin{smallmatrix} \mathbf{u}' \\ \mathbf{v}' \end{smallmatrix} \right] : \mathbf{v}' = \mathbf{0} \right\}$, contains all possible pixel values that force the coefficients of the unselected basis functions to be zero. We also define $C_1' = C_1 - \left[\begin{smallmatrix} \mathbf{u}^* \\ \mathbf{0} \end{smallmatrix} \right]$, which unlike C_1 is a proper linear subspace, since it contains the zero vector.

The algorithm proceeds by projecting back and forth between the two sets iteratively. Because the sets are convex, the distance between the point and its projection cannot increase from iteration to iteration. Convergence is reached when that distance fails to decrease, in which case the algorithm has either reached a point in the intersection, or the closest pair of points in two non-intersecting sets. Unlike Chen *et al.*'s original approach, our basis selection strategy only needs m potentially non-zero coefficients to guarantee that $\mathbf{A}$ has a left inverse and thus that the intersection will always be non-empty.

The basic steps of the algorithm are as follows:

1) Compute initial padding values $\mathbf{v}$ with some method.
2) Compute $\left[\begin{smallmatrix} \mathbf{u}' \\ \mathbf{v}' \end{smallmatrix} \right] = \tilde{\mathbf{G}} \left[\begin{smallmatrix} \mathbf{u}^* \\ \mathbf{v} \end{smallmatrix} \right]$.
3) If $\mathbf{v}'$ is sufficiently small, or a fixed number of iterations have passed, stop.
4) Compute $\left[\begin{smallmatrix} \mathbf{u} \\ \mathbf{v} \end{smallmatrix} \right] = \tilde{\mathbf{G}}^{-1} \left[\begin{smallmatrix} \mathbf{u}' \\ \mathbf{0} \end{smallmatrix} \right]$.
5) Go to Step 2.

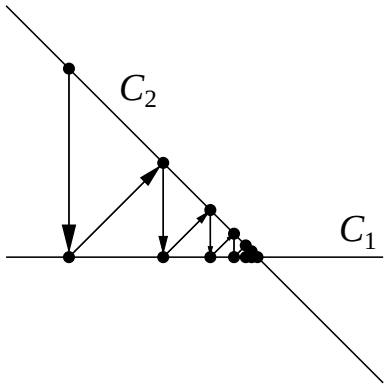

Fig. 4. Iterative projection between two convex sets.

For a single 1-D 8-point DCT, the cost of applying the adaptive transform using the Cholesky factorization methods given in Section III-A averaged over all shapes is about twelve times the cost of a single iteration of Chen *et al.*'s algorithm. Unfortunately, this algorithm can take nearly that many iterations to converge, even when the rows of $\mathbf{A}$ are orthogonal. Examine the illustration in Figure 4. In this example, each set is a 1-D line in a 2-D plane. The second set, C_2, has been rotated by the transform $\tilde{\mathbf{G}}^{-1}$. Yet every full iteration, the distance to the intersection point is only cut in half.

In general, the algorithm converges linearly by a factor of $\cos\psi(C_1', C_2)$ each projection, where $\psi(C_1', C_2)$ is the angle between the two subspaces, defined by [13]

$$\cos\psi(C_1', C_2) = \sup_{\mathbf{x}^{(1)} \in C_1', \mathbf{x}^{(2)} \in C_2} \frac{|\mathbf{x}^{(1)} \cdot \mathbf{x}^{(2)}|}{\|\mathbf{x}^{(1)}\|_2 \|\mathbf{x}^{(2)}\|_2} \qquad (24)$$

As the angle between the subspaces gets smaller, the algorithm takes even longer to converge.

From this illustration, it is clear that we can do a lot better. It is too expensive to solve the $m \times m$ linear system that would give us the intersection point in one step. However, it is possible that by solving a very small linear system, we can improve the convergence speed without excessive computation.

Given two points in each set, we can extrapolate a pair of 1-D linear subspaces that pass through each point pair. In the illustration above, these lines meet at the intersection point of C_1 and C_2, but in higher dimensions they might not intersect. We can still find the point on the line in C_1 that is closest to the line in C_2, however. This point must be at least as close

as the point produced by the original POCS algorithm and is often much closer.

More formally, consider the four points $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(4)}$, the first and third in C_2 and the second and fourth their respective projections into C_1. Then we can express the points on each line that are closest to the opposing line as

$$\mathbf{x}^a = \mathbf{x}^{(1)} + \mu_a(\mathbf{x}^{(3)} - \mathbf{x}^{(1)}) \in C_2 \tag{25}$$

$$\mathbf{x}^b = \mathbf{x}^{(2)} + \mu_b(\mathbf{x}^{(4)} - \mathbf{x}^{(2)}) \in C_1 \tag{26}$$

for some values of μ_a and μ_b. Noting that the line joining these two points is perpendicular to both lines, we have:

$$(\mathbf{x}^b - \mathbf{x}^a) \cdot (\mathbf{x}^{(3)} - \mathbf{x}^{(1)}) = 0 \tag{27}$$

$$(\mathbf{x}^b - \mathbf{x}^a) \cdot (\mathbf{x}^{(4)} - \mathbf{x}^{(2)}) = 0 \tag{28}$$

Substituting equations (25) and (26) into equations (27) and (28) reduces this to a 2×2 linear system which we can solve for μ_b, yielding

$$\mu_b = \frac{d_{3142}d_{2131} - d_{3131}d_{2142}}{d_{4242}d_{3131} - d_{3142}^2}, \tag{29}$$

where $d_{pqrs} = (\mathbf{x}^{(p)} - \mathbf{x}^{(q)}) \cdot (\mathbf{x}^{(r)} - \mathbf{x}^{(s)})$.

Because of the close relationships between our points, this last equation is particularly simple to compute. We denote the four points:

$$\mathbf{x}^{(1)} = \tilde{\mathbf{G}}^{-1}\begin{bmatrix}\mathbf{u}'^{(1)}\\ \mathbf{0}\end{bmatrix} = \begin{bmatrix}\mathbf{u}^{(1)}\\ \mathbf{v}^{(1)}\end{bmatrix} \qquad \mathbf{x}^{(2)} = \begin{bmatrix}\mathbf{u}^{*}\\ \mathbf{v}^{(1)}\end{bmatrix} \tag{30a}$$

$$\mathbf{x}^{(3)} = \tilde{\mathbf{G}}^{-1}\begin{bmatrix}\mathbf{u}'^{(2)}\\ \mathbf{0}\end{bmatrix} = \begin{bmatrix}\mathbf{u}^{(2)}\\ \mathbf{v}^{(2)}\end{bmatrix} \qquad \mathbf{x}^{(4)} = \begin{bmatrix}\mathbf{u}^{*}\\ \mathbf{v}^{(2)}\end{bmatrix} \tag{30b}$$

Substituting these into each of the constants that appear in equation (29) produces:

$$d_{3142} = \|\mathbf{v}^{(2)} - \mathbf{v}^{(1)}\|_2^2 \tag{31a}$$

$$d_{2131} = (\mathbf{u}^{*} - \mathbf{u}^{(1)}) \cdot (\mathbf{u}^{(2)} - \mathbf{u}^{(1)}) \tag{31b}$$

$$d_{3131} = \|\mathbf{x}^{(3)} - \mathbf{x}^{(1)}\|_2^2 \tag{31c}$$

$$d_{2142} = 0 \tag{31d}$$

$$d_{4242} = d_{3142} \tag{31e}$$

The equations in (31) can be substituted into the expression for μ_b from equation (29), simplifying it to:

$$\begin{aligned}\mu_b &= \frac{d_{2131}}{d_{3131} - d_{3142}} \\ &= \frac{(\mathbf{u}^{*} - \mathbf{u}^{(1)}) \cdot (\mathbf{u}^{(2)} - \mathbf{u}^{(1)})}{\|\mathbf{u}^{(2)} - \mathbf{u}^{(1)}\|_2^2}\end{aligned} \tag{32}$$

The divisor in equation (32) is zero only if $\mathbf{u}^{(1)}$ and $\mathbf{u}^{(2)}$ are equal. In this case, the algorithm has converged, and iteration can stop. The new point in C_1 is now given by computing a new set of padding values, which we will denote $\mathbf{v}^{*}$.

$$\mathbf{v}^{*} = \mathbf{v}^{(1)} + \mu_b(\mathbf{v}^{(2)} - \mathbf{v}^{(1)}) \tag{33}$$

The complete algorithm is thus:

1) Assign $\mathbf{u}^{(1)}$ and $\mathbf{v}^{(1)}$ the value $\mathbf{0}$.
2) Compute $\begin{bmatrix}\mathbf{u}'\\ \mathbf{v}'\end{bmatrix} = \tilde{\mathbf{G}}\begin{bmatrix}\mathbf{u}^{*}\\ \mathbf{0}\end{bmatrix}$.
3) If $\mathbf{v}'$ is sufficiently small, or a fixed number of iterations have passed, stop.
4) Compute $\begin{bmatrix}\mathbf{u}^{(2)}\\ \mathbf{v}^{(2)}\end{bmatrix} = \tilde{\mathbf{G}}^{-1}\begin{bmatrix}\mathbf{u}'\\ \mathbf{0}\end{bmatrix}$.
5) Compute μ_b via equation (32).
6) Compute $\mathbf{v}^{*} = \mathbf{v}^{(1)} + \mu_b(\mathbf{v}^{(2)} - \mathbf{v}^{(1)})$.
7) Compute $\begin{bmatrix}\mathbf{u}'\\ \mathbf{v}'\end{bmatrix} = \tilde{\mathbf{G}}\begin{bmatrix}\mathbf{u}^{*}\\ \mathbf{v}^{*}\end{bmatrix}$.
8) Replace $\mathbf{u}^{(1)}$ with $\mathbf{u}^{(2)}$ and $\mathbf{v}^{(1)}$ with $\mathbf{v}^{(2)}$.
9) Go to Step 3.

Note that instead of allowing an arbitrary method of initializing the padding values, which can improve convergence in the original algorithm, we always initialize with zeros. This lets us avoid an extra application of $\tilde{\mathbf{G}}$ and $\tilde{\mathbf{G}}^{-1}$ in order to get the recurrence started. The padding values computed after a single iteration of the modified algorithm are already very good.

The total cost of computing μ_b is just $2m$ multiplies. Another $(n - m)$ multiplies and one division are needed to compute the new padding values. Thus the total cost of the extra steps is only $n+m$ multiplies and one division. When m is large, the entire formulation can be moved to the dual spaces with $\mathbf{y}^{(1)}, \mathbf{y}^{(3)} \in \tilde{\mathbf{G}}C_1$ and $\mathbf{y}^{(2)}, \mathbf{y}^{(4)} \in \tilde{\mathbf{G}}C_2$. The algebra is virtually identical, and the location, μ_b', of the closest point in C_2 becomes

$$\mu_b' = \frac{-\mathbf{v}'^{(1)} \cdot (\mathbf{v}'^{(2)} - \mathbf{v}'^{(1)})}{\|\mathbf{v}'^{(2)} - \mathbf{v}'^{(1)}\|_2^2} \tag{34}$$

This gives us the solution directly in the transformed domain, avoiding a transform after padding, but requires one more up front to start the recurrence. The advantage is that $2n - m$ multiplies are used, which might be smaller than $n + m$.

On average, the total cost of the best of the two methods is $1.25n$ multiplies, or 10 multiplies for an 8-point transform, and one division. The best algorithm for the 8-point DCT, by comparison, requires 11 multiplies [14]. In this case, the additional computation required is about half the cost of one iteration. In practice the relative cost could be larger, if $\mathbf{G}$ and $\mathbf{G}^{-1}$ are available in dedicated hardware.

V. Results

We have implemented the padding algorithms described in an encoder for the Theora video codec [15]. Theora is a block based codec that utilizes an 8×8 DCT and motion compensation.

Four different basis selection algorithms were implemented: greedy maximization of the transform's determinant over the entire 2-D transform as described in Section III-A (*2-D Det*) and independent maximization over row and column transforms as described in Section III-B with respect to three different criteria—the determinant (*Det*), the minimum singular value (*Min* σ), and the coding gain under an $AR(1)$ model with correlation coefficient $\rho = 0.95$ (*Gain*). For the last three, an exhaustive search was performed over all possible bases and a table of the optimal choice for each of the 256 different input shapes was created. A fifth selection strategy, Kaup and Aach's original basis selection method (*KA*), was also included for comparison [9]. Where there were ties, the first basis in the lexicographic order on the coefficient numbers was used.

Three different algorithms were implemented to generate the padding. The first was the direct solution of the linear system, using the matrix multiply given in equation (11)

(*MMult*), the second was the original POCS method (*POCS*) [10], and the third was our accelerated POCS method given in Section IV (*POCS-A*). For *MMult*, we did not implement the alternative formulations given in equations (12) and (13). Similarly, for *POCS-A*, we only implemented the version that operates in the image domain, not the dual space formulation that operates in the transform domain. Implementing these alternative versions should increase computational efficiency, but should not otherwise affect the results.

Any combination of basis selection and padding algorithm can be used together, though *KA* does not require one, as the transform coefficients are computed directly as part of the selection strategy. Finally, as a baseline for comparisons, we implemented the MPEG-4 padding algorithm (*MPEG-4*) [8]. This gives a total of fourteen available algorithms, all of which produced bitstreams which could correctly be displayed by the existing Theora decoder without modification.

In the interests of creating reproducible research [16], all the code and data gathered in these experiments can be accessed from the Xiph.org Subversion repository at `http://svn.xiph.org/experimental/derf/theora-exp/doc/theory/padding/`. Test sequences can be downloaded from `http://media.xiph.org/`.

A. Boundary Extension

The first experiment we performed was to use our algorithms to pad an input source with non-standard dimensions to a multiple of 16. Theora handles non-multiple-of-16 image sizes by expanding the width and height to the next multiple of 16 at the encoder, and then storing a cropping rectangle in the header so the decoder can recover the original image dimensions.

This form of signal extension is a special case of the fixed-shape padding case postulated in Section III-A, where basis selection can be done once up front. In this particular case, the independent two-stage process we use in Section III-B is exactly equivalent to the coupled two-stage process given in equation (23). This is due to the fact that, for any rectangular shape, the zeros from the row transforms will always be in the same column, thus forcing the entire column to zero. Also, in this case the number of possible shapes is small enough to solve the linear system for each in advance.

For this experiment we used the standard QCIF *Coast Guard* sequence. This sequence continually pans, causing a good deal of activity around the borders—some that is easily predicted by motion compensation, and some that is not. Sixty-three test sequences were generated by cropping between 0 and 7 pixels from the top, bottom, left, and right. The left and right side always used the same cropping value, as did the top and bottom. All of the test sequences were then encoded with a constant quantizer, and the total bitrate and squared error added up. This was repeated for all available quantizers and for each of the four basis selection algorithms, using the *MMult* algorithm to perform the padding for each chosen basis. The same sequences were also padded with the *KA* and *MPEG-4* algorithms for comparison. Bitrate was recorded only for the residual DCT coefficients, and only for image blocks which required some padding; interior blocks were excluded. Similarly, distortion was only measured over the support region of padded blocks.

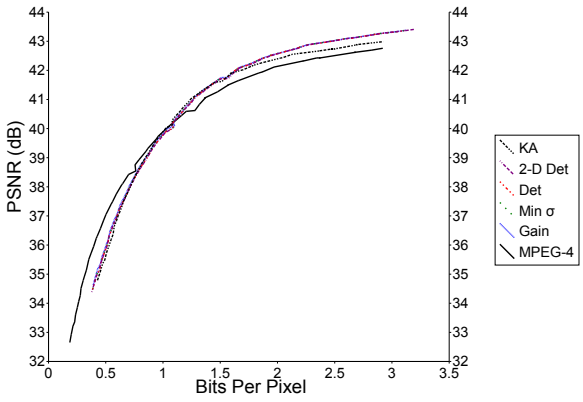

Fig. 5. Bitrate vs. distortion for the cropped *Coast Guard* sequences with various basis selection strategies.

The results are shown in Figure 5. The performance of all four of our basis selection algorithms are virtually indistinguishable, which is unsurprising as much of the time they select the same basis for a given shape. In most cases, 1-D optimization for coding gain gives an extremely slight advantage. At low rates *KA* method gives a slight increase in PSNR for a given quantizer, but nearly identical rate-distortion performance. At high rates, performance drops off below that of the other basis selection algorithms.

This suggests that full 2-D optimization of the selected basis is in fact unnecessary, at least for rectangular shapes. The greedy algorithm presented may still be useful for 1-D optimization with larger transform sizes, where an exhaustive search could prove intractable. For the small DCT used by most video compression algorithms, however, there is no reason not to use one of the fast basis selection algorithms.

It can also be seen that at low bitrates, the *MPEG-4* padding algorithm outperforms our padding algorithms. For each quantizer, our algorithms yield about a 1.5 dB improvement over *MPEG-4* with the same quantizer setting, but also requires many more bits. This indicates that our initial assumption, that forcing zeros into the DCT coefficients will increase coding efficiency, does not hold at low bitrates.

Part of the reason for this is that the size of the non-zero coefficients generally increases, which both makes it more likely a coefficient will not be quantized to zero and increases the number of bits required to code the coefficients that are not zero. At medium to high bitrates, the story is reversed, and our padding algorithms are superior. Part of the advantage of forbidding decoder modifications in our design is that we can select the appropriate algorithm for the target bitrate.

B. Dynamic Basis Selection

The next experiment we performed was to compare the various basis selection algorithms on dynamic shapes. For this purpose, we ran the algorithms on a segmented version of the

standard *Claire* sequence. The framerate was decimated by 5, and the remaining frames segmented into ten to fourteen segments.

Since Theora does not yet support arbitrary shape coding, we encode a separate sequence for each segment with a constant quantizer. Only the blocks that belong to that shape are coded, and for blocks on the boundary of the shape, our shape adaptive padding is used, even though no shape information is transmitted. Because the various foreground objects do not keep the same segment number throughout the video, we encode only INTRA frames. This was repeated for all available quantizers and for each of our four basis selection algorithms, using the *MMult* algorithm to perform the padding for each chosen basis. The experiment was also run using the *KA* and *MPEG-4* algorithms. Bitrate and squared error error were measured in the padding blocks as before, and summed over all segments and all frames in the sequence.

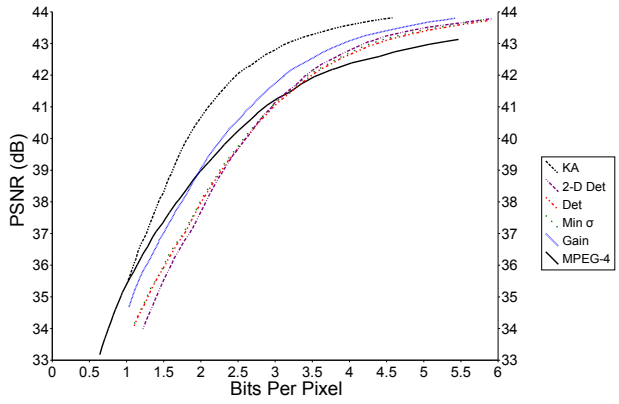

Fig. 6. Bitrate vs. distortion for the segmented *Claire* sequence with various basis selection strategies.

The results are shown in Figure 6. Here the picture is quite different. This time the *KA* algorithm peforms the best, though it is the second most computationally expensive. The *Gain* algorithm outperforms all of our other basis selection strategies at all bitrates by as much as 1 dB or more. This further increases the range of bitrates at which this padding algorithm is superior to *MPEG-4*.

Interestingly, *2-D Det* is actually worse than 1-D *Det* and *Min* σ at low bitrates, and gives only a very minor improvement at high bitrates, despite being the slowest algorithm. Initially, we chose to make a 2-D version of the *Det* strategy because the formulation is simple. It might also be possible to make a 2-D version of the *Gain* strategy, which is clearly superior to *Det* and *Min* σ, but these results suggest that it is not likely to be worth the increased computation and storage requirements.

C. POCS Acceleration

Our final experiment used the same setup as the previous experiment. However, this time we only tested the *Gain* basis selection method, and varied the method of computing the padding pixels. For both the *POCS* and *POCS-A* methods, the maximum number of iterations allowed was varied, and the rate-distortion curve for each limit is plotted against that of the *MMult* method in Figures 7 and 8.

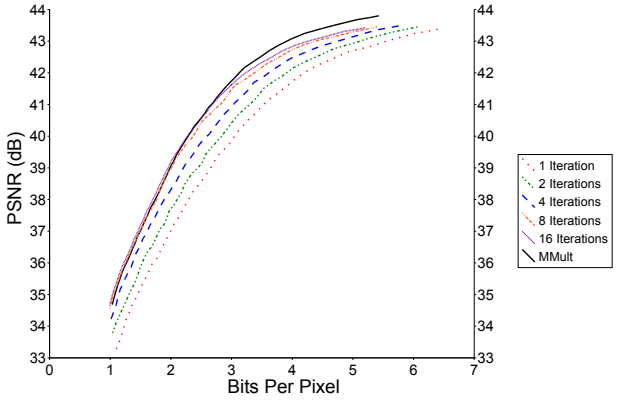

Fig. 7. Bitrate vs. distortion for the segmented *Claire* sequence padded with the original POCS algorithm.

As can be seen in Figure 7, the original POCS algorithm fails to coverge to the same quality *MMult* provides at high bitrates, even after 16 iterations. This is likely due to the fact that our implementation uses only integer operations, and the step size becomes smaller than a single pixel value. Increasing the precision to which pixel values are stored could resolve this, but would also require a more accurate implementation of the DCT transform. Even at low bitrates, it can take 8 iterations to achieve *MMult* quality, while medium bitrates can require as many as all 16.

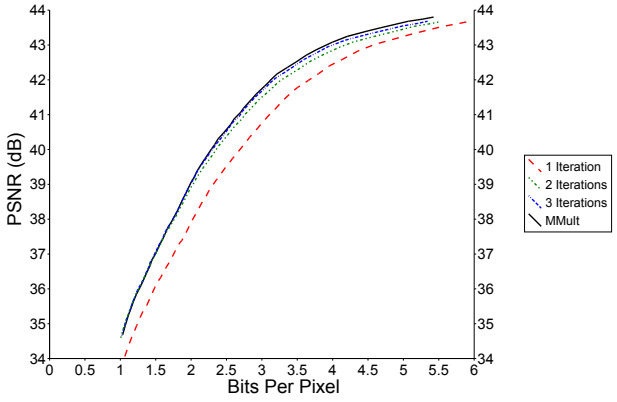

Fig. 8. Bitrate vs. distortion for the segmented *Claire* sequence padded with the accelerated POCS algorithm.

In Figure 8, our accelerated POCS algorithm tells a very different story. Two iterations are sufficient to do better than all 16 iterations of the original POCS algorithm, and 3 is sufficient to get within 0.1 dB of *MMult*. Because the steps are larger, integer precision is sufficient to reach *MMult* quality at all bitrates. It should also be noted that although the same basis is used for all 8 columns or rows for the second direction of the transform, it still requires a total of 16 to 24 iterations. In this case, it should be faster to only use *POCS-A* on the first

direction, and use *MMult* for the second direction. Solving the linear system generally requires floating point, however, which may not be available on some hardware.

VI. Conclusion

We have designed a new family of padding algorithms that can all be decoded with a standard inverse transform in the decoder. Our experimental results demonstrate that joint optimization of the full 2-D transform yields little or no benefit over optimization of each stage independently. Among the 1-D basis selection strategies, we have shown that maximizing the coding gain gives superior rate-distortion performance at all bitrates.

Our padding algorithms are also computationally efficient. The decoder uses a standard non-adaptive transform, and so can take advantage of existing fast implementations. The total cost of the 1-D foward transform, if all of the linear systems can be solved in advance and stored in a table, is just $\min(\mathrm{c}(\mathbf{G}) + (m-n)m, m^2)$ multiplies, where $\mathrm{c}(\mathbf{G})$ is the cost of a normal forward transform. This table is more practical than it might at first seem. By merging duplicate rows of coefficients and duplicate columns of row pointers, the entire set of solutions for an 8-point DCT requires about 6.3k of ROM, and the average transform cost is only 15.4 multiplies, or a 40% increase over the non-shape-adaptive cost.

If the storage space for the linear system solutions is not available, our algorithms can take advantage of customized hardware or hand-coded assembly routines for the ordinary DCT transform. We have presented an accelerated POCS algorithm that demonstrates greatly improved convergence speed and superior numerical accuracy without requiring floating point operations, with only a minor additional computational cost per iteration.

The cost of this computational efficiency is 1-1.5 dB when compared to the much more expensive *KA* algorithm for the dynamic shape case. For the boundary extension case, there is no penalty. Our algorithms give rate-distortion performance identical to *KA*. When compared against the MPEG-4 TMN 11 padding algorithm in both cases, we have demonstrated superior rate-distortion performance at medium and high bitrates, and superior distortion performance for all quantizer settings.

Our thesis that introducing zeros into the transformed coefficients would increase compression efficiency was *not* validated at low bitrates, where MPEG-4 exhibited superior rate-distortion performance. Thus, our framework does not yield optimal padding results at all bitrates, because it does not take into account the effects of quantization. Designing a framework which can take these effects into account is a subject of future work. Because our decoder uses a standard non-adaptive inverse transform, we are free to choose the algorithm that performs best for the target bitrate or quality and meets our complexity requirements, and we can take advantage of future algorithmic improvements as they become available.

Acknowledgment

The author would like to thank the Xiph.org Foundation for developing the patent-unencumbered, open source Theora codec and On2 Technologies, for donating the VP3 codec, on which Theora is based.

References

[1] M. Gilge, T. Engelhardt, and R. Mehlan, "Coding of arbitrarily shaped image segments based on a generalized orthogonal transform," *Signal Processing: Image Communication*, vol. 1, no. 2, pp. 153–180, Oct. 1989.

[2] T. Sikora and B. Makai, "Shape-adaptive DCT for generic coding of video," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 5, no. 1, pp. 59–62, Feb. 1995.

[3] R. Stasiński and J. Konrad, "A new class of fast shape-adaptive orthogonal transforms and their application to region-based image compression," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 9, no. 1, pp. 16–34, Feb. 1999.

[4] W. Philips, "A fast algorithm for orthogonalizing polynomials on an arbitrarily shaped region," *Multidimensional Systems and Signal Processing*, vol. 8, no. 4, pp. 409–421, Oct. 1997.

[5] A. Kaup and S. Panis, "On the performance of the shape adaptive DCT in object-based coding of motion compensated difference images," in *Proc. 1997 Picture Coding Symposium*, Berlin, Sept. 1997, pp. 653–657.

[6] P. Kauff and K. Schüür, "An extension of shape-adaptive DCT (SA-DCT) towards DC separation and ΔDC correction," in *Proc. 1997 Picture Coding Symposium*, Berlin, Sept. 1997, pp. 647–652.

[7] S.-F. Chang and D. G. Messerschmitt, "Transform coding of arbitrarily-shaped image segments," in *Proc. First ACM International Conference on Multimedia*, Anaheim, California, Aug. 1993, pp. 83–90.

[8] A. Kaup, "Object-based texture coding of moving video in MPEG-4," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 9, no. 1, pp. 5–15, Feb. 1999.

[9] A. Kaup and T. Aach, "Segment-oriented coding of textured images based on successive approximation," in *Proc. 1994 IEEE International Symposium on Speech, Image Processing, Neural Networks*, Hong Kong, Apr. 1994, pp. 197–200.

[10] H. H. Chen, M. R. Civanlar, and B. G. Haskell, "A block transform coder for arbitrarily shaped image segments," in *Proc. IEEE International Conference on Image Processing*, vol. 1, Austin, Texas, Nov. 1994, pp. 85–89.

[11] G. Shen, B. Zeng, and M. L. Liou, "Arbitrarily shaped transform coding based on a new padding technique," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 11, no. 1, pp. 67–79, Jan. 2001.

[12] J. Katto and Y. Yasuda, "Performance evaluation of subband coding and optimization of its filter coefficients," in *Proc. SPIE Visual Communication and Image Processing '91*, Boston, Nov. 1991, pp. 95–106.

[13] D. C. Youla, "Generalized image restoration by the method of alternating orthogonal projections," *IEEE Trans. Circuits Syst.*, vol. 25, no. 9, pp. 694–702, Sept. 1978.

[14] C. Loeffler, A. Ligtenberg, and G. S. Moschytz, "Practical fast 1-D DCT algorithms with 11 multiplications," in *Proc. 1989 IEEE International Conference on Acoust., Speech, Signal Processing*, vol. 2, Glasgow, May 1989, pp. 988–991.

[15] *Theora I Specification*, Xiph.org Foundation, 2004, http://www.theora.org/doc/Theora_I_spec.pdf.

[16] J. B. Buckheit and D. L. Donoho, *Wavelets and Statistics*, ser. Lecture Notes on Statistics. Berlin, New York: Springer-Verlag, 1995, vol. 103, ch. WaveLab and Reproducible Research, pp. 53–81.

Timothy B. Terriberry (S'05) received dual B.S. and M.S. degrees in both Mathematics and Computer Science from Virginia Tech in 1999 and 2001, respectively. He is currently pursuing a Ph.D. in Computer Science at the University of North Carolina at Chapel Hill. He also volunteers for the Xiph.org Foundation, a non-profit organization that develops free, open multimedia protocols and software.