Quadtree Structured Image Approximation for Denoising and Interpolation

Adam Scholefield, *Member, IEEE,* and Pier Luigi Dragotti, *Senior Member, IEEE*

***Abstract*—The success of many image restoration algorithms is often due to their ability to sparsely describe the original signal. In [3] Shukla et al. proposed a compression algorithm, based on a sparse quadtree decomposition model, which could optimally represent piecewise polynomial images. In this paper we adapt this model to image restoration by changing the rate-distortion penalty to a description-length penalty. Moreover, one of the major drawbacks of this type of approximation is the computational complexity required to find a suitable subspace for each node of the quadtree. We address this issue by searching for a suitable subspace much more efficiently using the mathematics of updating matrix factorisations. Algorithms are developed to tackle denoising and interpolation. Simulation results indicate that we beat state of the art results when the original signal is in the model (e.g. depth images) and are competitive for natural images when the degradation is high.**

***Index Terms*—Denoising, image models, interpolation, piecewise polynomial approximation, quadtree, sparse regularisation.**

I. Introduction

THE ability to accurately model images is key to many image processing tasks including image restoration. Recently the central ingredient in most modelling techniques has been sparsity. This reduces the complexity of an image to the linear combination of a few functions and the fewer functions required, the sparser the representation and the better the model. Wavelets very efficiently represent one dimensional (1-D) piecewise polynomial functions; however, due to their separability in two dimensions (2-D) they struggle to capture higher dimensional discontinuities such as edges and contours in images. This problem has been noted many times in the literature and many improved transforms have been suggested: e.g., Ridgelets [4], Curvelets [5], Contourlets [6], Wedgelets [7] and Bandlets [8]. The latter two of these use a quadtree decomposition to adaptively partition the image. This is the approach we will take in this paper. However, unlike Bandlets which uses a basis over each adaptive region, we will approximate each region using a very low dimensional model, specifically either a 2-D polynomial model or two 2-D polynomials separated by an edge discontinuity.

This model is an extension of Wedgelets, and is similar to the model used in the compression algorithm of Shukla et al. [3], which they also later extended to denoising [9]. However, they retained the rate-distortion structure which is not particularly suitable for denoising. We instead replace the bit-rate constraint with a minimum description length criterion that is more appropriate to tackle image restoration problems. This model can optimally represent piecewise polynomial images, which are of particular interest because they occur in many real world situations. For example, natural images are full of approximately piecewise polynomial regions although they also contain areas, such as texture, that cannot be efficiently represented by this model. Because of this, we expect our algorithms to perform well on large parts of natural images; however, they may struggle in highly textured areas.

A depth image has pixel values that correspond to the distance from the camera image plane to the real object; this is in contrast to natural images where the pixel value corresponds to a colour or brightness value. An example of such an image is shown in Fig. 1. It is clear that depth images are inherently piecewise smooth and will therefore be ideally suited to our image model and our restoration algorithms.

Often one of the drawbacks of this type of approximation is the computational cost required to find a suitable edge discontinuity for each region. In this paper we develop a novel technique based on modifying a QR decomposition, see Daniel et al [11], that allows us to efficiently find a suitable edge. This improved efficiency allows us to compute fast approximations and use cycle spinning [12] to further improve our restoration

Adam Scholefield and Pier Luigi Dragotti are with the Communications and Signal Processing Group, Electrical and Electronic Engineering Department, Imperial College, London, SW7 2AZ, UK, Tel: +44 (0)20 7594 6192, Fax: +44 (0)20 7594 6234 (e-mail: adam.scholefield@imperial.ac.uk; p.dragotti@imperial.ac.uk).

The work in this paper was in part presented at [1] and [2].

(a) A real complex scene (b) The resulting depth image

Fig. 1: An example depth image and its corresponding colour image (these images are taken from the stereo data set [10]).

performance.

The rest of this paper is organised as follows: Section II introduces the denoising and interpolation problems and discusses the state of the art. Section III explains our piecewise polynomial model and approximation algorithm, first in the 1-D case and then the 2-D case. Sections IV and V deal with denoising and interpolation respectively: these are both solved using slight variants of the approximation algorithm previously presented. The simulation results are given in Section VI where we compare the performance on both natural and depth images. Finally we conclude in Section VII.

In this paper we use the following notation: scalars are denoted by regular letters, vectors by lowercase bold letters and matrices by uppercase bold. Subscripts and superscripts are used to provide additional naming, e.g., $\boldsymbol{I_N}$ is the $N \times N$ identity. The only exception to this is superscripts on scalars, which are used for powers.

II. Problem set up and state of the art

The degradation process in the canonical denoising set-up can be modelled as follows:

$$\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{z}, \tag{1}$$

where the vectors $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in \mathbb{R}^N$ are the desired, measured and noise images respectively (N is the number of pixels). Often one assumes that $\boldsymbol{z}$ comes from a white Gaussian distribution, $\boldsymbol{z} \sim \mathcal{N}_N(\boldsymbol{0}, \sigma_z^2 \boldsymbol{I_N})$, and that $\boldsymbol{x}$ has a sparse representation in a proper domain. That is, $\boldsymbol{x} = \boldsymbol{D\theta}$ where $\boldsymbol{\theta} \in \mathbb{R}^m$ has only a small number of large or non zero entries and $\boldsymbol{D} \in \mathbb{R}^{N \times m}$ is the reconstruction transform (m is the maximum dimension of the approximation). Based on these assumptions, a possible solution, $\hat{\boldsymbol{x}} = \boldsymbol{D}\hat{\boldsymbol{\theta}}$, is found from

$$\hat{\boldsymbol{\theta}} = arg\min_{\boldsymbol{\theta}} \|\boldsymbol{y} - \boldsymbol{D\theta}\|_2^2 + \lambda \|\boldsymbol{\theta}\|_0, \tag{2}$$

where $\|\boldsymbol{\theta}\|_0$ is the number of non zero entries in $\boldsymbol{\theta}$. When $\boldsymbol{D}$ is an orthogonal transform ($\boldsymbol{D}^T\boldsymbol{D} = \boldsymbol{I_m}$) the problem separates and can be easily solved by projecting $\boldsymbol{y}$ onto the column space of $\boldsymbol{D}$ and hard thresholding the result. The convex relaxation of (2) is obtained by replacing $\|\boldsymbol{\theta}\|_0$ with $\|\boldsymbol{\theta}\|_1$: in this case the solution is identical except that hard thresholding is replaced with the well known soft thresholding operator.

In the more general case when $\boldsymbol{D}$ is not orthogonal the problem is more complex because the equations are coupled together: this occurs for example when $\boldsymbol{D}$ is an over complete dictionary. Daubechies et al [13] showed that iterative soft thresholding converges to the global minimum of the more general convex relaxed problem. In the non-convex case iterative hard thresholding converges to a local minimum of (2). More information on this well studied topic can be found in the literature [14]–[18].

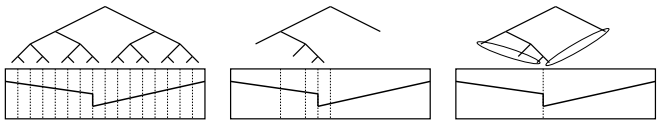
In this paper we search for a sparse solution by minimising a cost function very similar to (2), but our transform is nonlinear and we use a minimum description length prior rather than a purely sparsity promoting prior. Our model is an adaptive representation and fits into the class of models reviewed in [19]. Similar to the orthogonal linear transform case we can find a closed form solution and do not have to resort to iterative methods.

Alternatively, the image denoising problem can be solved using nonlocal and adaptive restoration algorithms [20]–[25]. These algorithms have been shown to provide excellent results over a wide range of images; however, it is possible that the added flexibility may be detrimental when the original signal follows a much simpler model. For example, depth images are inherently piecewise smooth so a prior enforcing this might be more appropriate for this class of images.

In the interpolation problem one tries to estimate missing pixels using those available. The problem can be formulated using the following degradation model:

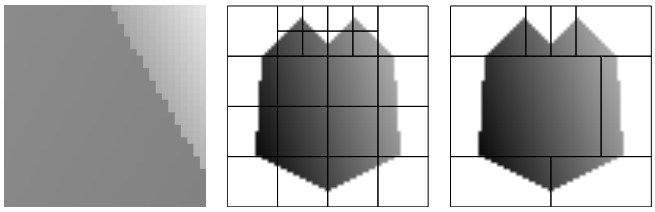
$$\boldsymbol{y} = \boldsymbol{H}\boldsymbol{x}, \tag{3}$$

where $\boldsymbol{x} \in \mathbb{R}^N$ is the complete image, $\boldsymbol{y} \in \mathbb{R}^{N_v}$ is the image of only known pixels and $\boldsymbol{H} \in \mathbb{R}^{N_v \times N}$ is the identity matrix with $N - N_v$ rows, corresponding to the unknown pixels, removed (N_v is the number of visible pixels). There are many possible interpolation setups and the algorithm that we develop will be quite general. Despite this we will present results for the problem of interpolating from an irregularly sampled grid of data points. This problem arises, for example, when trying to infer the depth map of a scene from a cloud of depth points. A very effective algorithm for this type of problem was presented in [26]. In this work Takeda et al extend classical kernel regression by adapting the shape



(a) The binary tree at its deepest depth. (b) The pruned representation. (c) The prune-joined representation.

Fig. 2: Comparison between the prune and prune-join algorithms for a piecewise linear 1-D signal.



(a) A possible tile with an edge. (b) The pruned representation. (c) The prune-joined representation.

Fig. 3: Comparison between the prune and prune-join algorithms for a piecewise linear 2-D signal.

of the kernel locally depending on the structure of the signal. Additionally, nonlocal algorithms have also been applied to interpolation. For example, Li [27] utilised state of the art nonlocal denoising, in an iterative fashion, to achieve excellent interpolation results. These two approaches, as well as bi-cubic interpolation, will be used to provide comparisons with the proposed interpolation algorithm.

III. Piecewise polynomial approximation of 1-D and 2-D functions using a binary or quadtree decomposition

Our approach is based on the assumption that images are 2-D piecewise smooth functions and can therefore be sparsely described using a 2-D piecewise polynomial model. In this section we present a fast algorithm that finds the piecewise polynomial approximation of an image using a quadtree decomposition. The accuracy of the approximation is dependent on the complexity of the decomposition. This algorithm is then used in the later sections for image restoration.

A. 1-D Piecewise polynomial approximation using a binary tree

The 2-D quadtree decomposition algorithm involves a pruning and joining step. To clarify the notion of 'prune' and 'join' we first consider the 1-D case and approximate a 1-D piecewise linear function.

We dyadically partition the signal space using a binary tree, where each leaf represents a polynomial. Therefore, to represent the 1-D piecewise linear signal shown in Fig. 2, we could use any of the trees shown in Figs. 2(a), 2(b) or 2(c). To calculate these representations we define a cost function that has two terms: the first is a two-norm data fitting term and the second is a term to penalise the description length. In 1-D we define the description length to be the sum of the degrees of the polynomials in the approximation; therefore, the cost function is

$$\|\boldsymbol{y} - \boldsymbol{x}\|_2^2 + \lambda \sum_{i \in T} d_i, \tag{4}$$

where $\boldsymbol{y}$ is the 1-D signal we are trying to approximate, T is the set of all leaves in the approximation $\boldsymbol{x}$, and d_i is the degree of the polynomial at node i. Here λ is used to provide a tradeoff between the two terms: it is set according to the quality of approximation that is required and in later sections will be set depending on the degradation of the restoration problem.

To obtain the deepest depth solution shown in Fig. 2(a) we visit each node at this deepest depth and minimise (4) locally: the approximation of a single node is a simple linear approximation problem. The pruned representation shown in Fig. 2(b) is obtained using a bottom up approach that starts from the deepest depth solution. The parent nodes of this deepest depth solution are approximated using the same approach and then two sibling leaves are pruned if the sum of their costs is greater than the cost of their parent. This process is repeated all the way up the tree to produce the final pruned tree.

The pruned representation is suboptimal due to the limitations of the binary tree structure: for example, because of the location of the discontinuity in Fig. 2(b), five regions are required to represent a piecewise linear signal containing only one discontinuity. By allowing neighbouring regions of the tree to jointly represent just one polynomial region we can overcome this limitation, as shown in Fig. 2(c). This prune-join representation is calculated as follows: the leaves of the pruned tree are visited, in a top-down left-right fashion, and tested to see if they can be joined to neighbouring leaves which have already been checked (i.e. nodes that are at a higher depth, or the same depth but further to the left in the tree). Two leaves are joined if their combined cost is less than the sum of their individual costs. After two leaves have been joined, the joined representation is used in place of the individual leaves for the rest of the joining algorithm.

B. 2-D Piecewise polynomial approximation using a quadtree

Now let us move to the 2-D case: a quadtree is constructed where each leaf is either a global polynomial or two polynomials separated by a continuous boundary. Figure 3(a) shows a possible node which we will also call a tile. Prune and prune-join representations are generated in almost the same way as the 1-D case and examples are shown in Figs. 3(b) and 3(c). In 2-D the penalty for the description length, $P^x(\boldsymbol{x})$, is slightly more complex: it is defined to be separable across each node so we can write

$$P^x(\boldsymbol{x}) = \sum_{i \in L} P_i^x(\boldsymbol{x}), \tag{5}$$

where $\boldsymbol{x}$ is the image vector, $P_i^x(\boldsymbol{x})$ is the penalty of tile i and L is the set of all leaves in the approximation. In 1-D the description length of a tile is the degree of the polynomial; in 2-D it is very similar: we penalise a polynomial region of degree d by $2d+1$ because this is the dimension of the space of 2-D polynomials of degree d. For example 2-D polynomials of degree one span the space which can be defined by a constant basis function and two linear basis functions, one per direction. The total description length penalty also needs to deal with the description of the edge discontinuity. This can be done by defining the penalty for the i-th node to be

$$P_i^x(\boldsymbol{x}) = \begin{cases} 2d+1 & \text{if a global tile,} \\ 2d_1 + 2d_2 + 2 + ln(N_i) & \text{if an edge tile,} \end{cases}$$

where d_1, d_2 are the degrees of the polynomials either side of the edge and N_i is the number of pixels in tile i. Finally, $ln(N_i)$ is present to penalise edges of larger tiles more harshly because there are more possible discrete edges to choose from.

Just like the 1-D case we use a two-norm data fitting term producing the final cost function

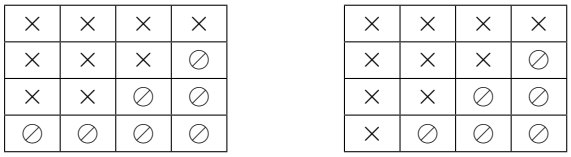
$$\|\boldsymbol{y} - \boldsymbol{x}\|_2^2 + \lambda P^x(\boldsymbol{x}), \tag{6}$$

which is separable over each tile. To find the best approximation for a particular tile we exhaustively search a dictionary of possible edges (including the global 'no edge' case) and choose the edge with the cheapest (6). This process is explained in detail in Section III-C. The prune and prune-join algorithms are identical to the 1-D case and their goal is to find

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \left\{ \|\boldsymbol{y} - \boldsymbol{x}\|_2^2 + \lambda P^x(\boldsymbol{x}) \right\}. \tag{7}$$

The quadtree piecewise polynomial model is nonlinear so we will use the notation $\boldsymbol{x} = D(\boldsymbol{\theta})$ where $\boldsymbol{\theta}$ is the parameter set which describes the tile structure, edge discontinuities and polynomial coefficients. We also use the notation that the penalty $P^x\left(D(\boldsymbol{\theta})\right) = P^\theta(\boldsymbol{\theta})$ and therefore (7) is equivalent to

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \left\{ \|\boldsymbol{y} - D(\boldsymbol{\theta})\|_2^2 + \lambda P^\theta(\boldsymbol{\theta}) \right\}.$$



(a) First edge (e). (b) Second edge ($\tilde{e}$).

Fig. 4: The two edges used for the example approximation.

C. Finding the best approximation for a node

The prune and prune-join algorithms assume we can calculate an approximation for each node of the tree. In 2-D this involves calculating a suitable edge discontinuity, which may be 'no edge', and the polynomial coefficients.

The tile approximation given a particular edge is a linear problem that is solved by projecting onto the polynomial subspace and hard thresholding. Traditionally a suitable edge is found by exhaustively searching a large number of possible edge discontinuities by approximating each one using two linear projections, one either side of the edge. The edge leading to the minimum cost is chosen. This approach is inefficient and can become unfeasible for large tiles. In this case, due to complexity, the search space is reduced so that only a small number of straight edges are tested. To overcome this limitation, we present a fast method which allows us to exhaustively search edges and also, if we wish, relax the constraint that the edge be straight. Our inspiration comes from the mathematics of updating matrix factorisations: it is often more efficient to update a factorisation than recalculate it from scratch if the original matrix has only undergone a small change. We can use this to quickly check different edges by changing just one pixel at a time. We will see that the complexity of approximating a new edge is independent of the tile size and instead depends only on the maximum degree of the polynomials.

In what follows we demonstrate the process by approximating a 4×4 tile using the two edges shown in Fig. 4. We will first approximate the tile using the edge shown in Fig. 4(a) by calculating the QR decomposition from scratch and then update this factorisation to approximate the tile using the edge shown in Fig. 4(b).

Both approximations will require a vector representation for the tile which is obtained by lexicographically stacking the tile into a vector $\boldsymbol{t} \in \mathbb{R}^{N_i}$ ($N_i = 16$ in this

example):

$$\begin{array}{|c|c|c|c|}\hline t_{11}&t_{12}&t_{13}&t_{14}\\\hline t_{21}&t_{22}&t_{23}&t_{24}\\\hline t_{31}&t_{32}&t_{33}&t_{34}\\\hline t_{41}&t_{42}&t_{43}&t_{44}\\\hline\end{array} \quad \Rightarrow \quad \boldsymbol{t} = \begin{bmatrix} t_{11}\\ t_{21}\\ \vdots\\ t_{41}\\ t_{12}\\ \vdots\\ t_{44}\end{bmatrix}.$$

1) Approximating the tile using the edge of Fig. 4(a) by computing the QR decomposition: Let the maximum degree of the polynomials be one, resulting in polynomial subspaces of dimension three. We lexicographically stack the three biorthogonal linear polynomial basis functions either side of the edge and use these vectors as the columns of two matrices $\boldsymbol{B_1^e}$ and $\boldsymbol{B_2^e}$. We then calculate the thin QR decomposition for these matrices. For the first side of the edge we have

$$\begin{array}{|c|c|c|c|}\hline 1&1&1&1\\\hline 1&1&1&\oslash\\\hline 1&1&\oslash&\oslash\\\hline \oslash&\oslash&\oslash&\oslash\\\hline\end{array}\quad\begin{array}{|c|c|c|c|}\hline 1&2&3&4\\\hline 1&2&3&\oslash\\\hline 1&2&\oslash&\oslash\\\hline \oslash&\oslash&\oslash&\oslash\\\hline\end{array}\quad\begin{array}{|c|c|c|c|}\hline 1&1&1&1\\\hline 2&2&2&\oslash\\\hline 3&3&\oslash&\oslash\\\hline \oslash&\oslash&\oslash&\oslash\\\hline\end{array} \quad\Rightarrow\quad \boldsymbol{B_1^e} = \begin{bmatrix} 1&1&1\\ 1&1&2\\ 1&1&3\\ 1&2&1\\ 1&2&2\\ 1&2&3\\ 1&3&1\\ 1&3&2\\ 1&4&1\end{bmatrix},$$

$$\boldsymbol{B_1^e} = \begin{bmatrix} \frac{1}{3}&-\frac{\sqrt{5}}{6}&-\frac{3\sqrt{3}}{10}\\ \frac{1}{3}&-\frac{\sqrt{5}}{6}&-\frac{\sqrt{3}}{30}\\ \frac{1}{3}&-\frac{\sqrt{5}}{6}&\frac{7\sqrt{3}}{30}\\ \frac{1}{3}&-\frac{\sqrt{5}}{60}&-\frac{13\sqrt{3}}{60}\\ \frac{1}{3}&-\frac{\sqrt{5}}{60}&\frac{\sqrt{3}}{20}\\ \frac{1}{3}&-\frac{\sqrt{5}}{60}&\frac{19\sqrt{3}}{60}\\ \frac{1}{3}&\frac{4\sqrt{5}}{30}&-\frac{2\sqrt{3}}{15}\\ \frac{1}{3}&\frac{4\sqrt{5}}{30}&\frac{2\sqrt{3}}{15}\\ \frac{1}{3}&\frac{17\sqrt{5}}{60}&-\frac{\sqrt{3}}{20}\end{bmatrix}\begin{bmatrix} 3&\frac{19}{3}&\frac{16}{3}\\ 0&\frac{4\sqrt{5}}{3}&-\frac{5\sqrt{5}}{12}\\ 0&0&\frac{5\sqrt{3}}{4}\end{bmatrix} = \boldsymbol{Q_1^e R_1^e}$$

and $\boldsymbol{B_2^e} = \boldsymbol{Q_2^e R_2^e}$ is constructed in the same way for the other side. Having the orthogonal Q matrices makes approximating $\boldsymbol{t}$ using this edge easy. We first split $\boldsymbol{t}$ into two corresponding vectors, $\boldsymbol{t_1^e}$ and $\boldsymbol{t_2^e}$, either side of the edge. We will use the notation $\boldsymbol{t_1^e} \cup \boldsymbol{t_2^e} = \boldsymbol{t}$ to allow us to combine vectors from different regions of the image; similarly, we use '$\cap$' for the intersection of two image vectors. Since $\boldsymbol{t_1^e} \cap \boldsymbol{t_2^e} = \oslash$, the approximation can be found independently for each side:

$$\hat{\boldsymbol{t}}_{\boldsymbol{1}}^{\boldsymbol{e}} = \boldsymbol{Q_1^e}\hat{\boldsymbol{\theta}}_{\boldsymbol{Q_1^e}} \text{ where } \hat{\boldsymbol{\theta}}_{\boldsymbol{Q_1^e}} = \boldsymbol{Q_1^{e^T} t_1^e},$$
$$\text{and } \hat{\boldsymbol{t}}_{\boldsymbol{2}}^{\boldsymbol{e}} = \boldsymbol{Q_2^e}\hat{\boldsymbol{\theta}}_{\boldsymbol{Q_2^e}} \text{ where } \hat{\boldsymbol{\theta}}_{\boldsymbol{Q_2^e}} = \boldsymbol{Q_2^{e^T} t_2^e}.$$

The error is

$$\begin{aligned}\|\boldsymbol{t} - \hat{\boldsymbol{t}}^{\boldsymbol{e}}\| &= \|\boldsymbol{t_1^e} - \hat{\boldsymbol{t}}_{\boldsymbol{1}}^{\boldsymbol{e}}\|_2^2 + \|\boldsymbol{t_2^e} - \hat{\boldsymbol{t}}_{\boldsymbol{2}}^{\boldsymbol{e}}\|_2^2\\ &= \|\boldsymbol{t}\|_2^2 - \|\hat{\boldsymbol{\theta}}_{\boldsymbol{Q_1^e}}\|_2^2 - \|\hat{\boldsymbol{\theta}}_{\boldsymbol{Q_2^e}}\|_2^2, \end{aligned} \tag{8}$$

which can be simplified by neglecting the $\|\boldsymbol{t}\|_2^2$ term when comparing the errors of different edges for the same tile.

2) Approximating the tile using the edge of Fig. 4(b) by updating the previous QR decomposition: For simplicity we will just consider the first side of the edge: we construct $\boldsymbol{B_1^{\tilde{e}}}$ as we did before but we do not calculate the QR decomposition. Instead we proceed as follows: we first modify the matrices $\boldsymbol{Q_1^e}$ and $\boldsymbol{R_1^e}$ so that the equality still holds after the new row is added (the new row of scalar values has been shown in bold for emphasis; this goes against the convention of the rest of the paper where lowercase bold is reserved for vectors):

$$\begin{array}{|c|c|c|c|}\hline 1&1&1&1\\\hline 1&1&1&\oslash\\\hline 1&1&\oslash&\oslash\\\hline \mathbf{1}&\oslash&\oslash&\oslash\\\hline\end{array}\quad\begin{array}{|c|c|c|c|}\hline 1&2&3&4\\\hline 1&2&3&\oslash\\\hline 1&2&\oslash&\oslash\\\hline \mathbf{1}&\oslash&\oslash&\oslash\\\hline\end{array}\quad\begin{array}{|c|c|c|c|}\hline 1&1&1&1\\\hline 2&2&2&\oslash\\\hline 3&3&\oslash&\oslash\\\hline \mathbf{4}&\oslash&\oslash&\oslash\\\hline\end{array} \quad\Rightarrow\quad \boldsymbol{B_1^{\tilde{e}}} = \begin{bmatrix} 1&1&1\\ 1&1&2\\ 1&1&3\\ \mathbf{1}&\mathbf{1}&\mathbf{4}\\ 1&2&1\\ 1&2&2\\ 1&2&3\\ 1&3&1\\ 1&3&2\\ 1&4&1\end{bmatrix},$$

$$\boldsymbol{B_1^{\tilde{e}}} = \begin{bmatrix} \frac{1}{3}&-\frac{\sqrt{5}}{6}&-\frac{3\sqrt{3}}{10}&0\\ \frac{1}{3}&-\frac{\sqrt{5}}{6}&-\frac{\sqrt{3}}{30}&0\\ \frac{1}{3}&-\frac{\sqrt{5}}{6}&\frac{7\sqrt{3}}{30}&0\\ \mathbf{0}&\mathbf{0}&\mathbf{0}&\mathbf{1}\\ \frac{1}{3}&-\frac{\sqrt{5}}{60}&-\frac{13\sqrt{3}}{60}&0\\ \frac{1}{3}&-\frac{\sqrt{5}}{60}&\frac{\sqrt{3}}{20}&0\\ \frac{1}{3}&-\frac{\sqrt{5}}{60}&\frac{19\sqrt{3}}{60}&0\\ \frac{1}{3}&\frac{4\sqrt{5}}{30}&-\frac{2\sqrt{3}}{15}&0\\ \frac{1}{3}&\frac{4\sqrt{5}}{30}&\frac{2\sqrt{3}}{15}&0\\ \frac{1}{3}&\frac{17\sqrt{5}}{60}&-\frac{\sqrt{3}}{20}&0\end{bmatrix}\begin{bmatrix} 3&\frac{19}{3}&\frac{16}{3}\\ 0&\frac{4\sqrt{5}}{3}&-\frac{5\sqrt{5}}{12}\\ 0&0&\frac{5\sqrt{3}}{4}\\ \mathbf{1}&\mathbf{1}&\mathbf{4}\end{bmatrix} = \tilde{\boldsymbol{Q}}_{\boldsymbol{1}}^{\tilde{\boldsymbol{e}}}\tilde{\boldsymbol{R}}_{\boldsymbol{1}}^{\tilde{\boldsymbol{e}}}.$$

Currently, $\boldsymbol{B_1^{\tilde{e}}}$ is factored into an orthogonal matrix, $\tilde{\boldsymbol{Q}}_{\boldsymbol{1}}^{\tilde{\boldsymbol{e}}}$, multiplied by a matrix, $\tilde{\boldsymbol{R}}_{\boldsymbol{1}}^{\tilde{\boldsymbol{e}}}$, which is not upper

triangular. To get the factorisation into the desired thin QR form we introduce an orthogonal matrix, $\boldsymbol{G} \in \mathbb{R}^{4\times4}$, which is the product of three Givens rotation matrices: $\boldsymbol{G} = \boldsymbol{G_3 G_2 G_1}$. Since $\boldsymbol{G}^T\boldsymbol{G} = \boldsymbol{I_4}$, we can write

$$\boldsymbol{B_1^{\tilde{e}}} = \boldsymbol{\tilde{Q}_1^{\tilde{e}} G}^T \boldsymbol{G \tilde{R}_1^{\tilde{e}}}.$$

The aim of $\boldsymbol{G}$ is to make $\boldsymbol{G\tilde{R}_1^{\tilde{e}}}$ upper triangular. To do this the three Givens rotation matrices are constructed to reflect $\begin{bmatrix} 1 & 1 & 4 \end{bmatrix}$ into the diagonal entries of $\boldsymbol{R_1^e}$. The first rotation matrix, $\boldsymbol{G_1}$, reflects the first element of $\begin{bmatrix} 1 & 1 & 4 \end{bmatrix}$ into the top left element of $\boldsymbol{R_1^e}$:

$$\boldsymbol{G_1} = \begin{bmatrix} \frac{3\sqrt{10}}{10} & 0 & 0 & \frac{\sqrt{10}}{10} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \frac{\sqrt{10}}{10} & 0 & 0 & -\frac{3\sqrt{10}}{10} \end{bmatrix}$$

so that

$$\boldsymbol{G_1} \begin{bmatrix} 3 & \frac{19}{3} & \frac{16}{3} \\ 0 & \frac{4\sqrt{5}}{3} & -\frac{5\sqrt{5}}{12} \\ 0 & 0 & \frac{5\sqrt{3}}{4} \\ 1 & 1 & 4 \end{bmatrix} = \begin{bmatrix} \sqrt{10} & 2\sqrt{10} & 2\sqrt{10} \\ 0 & \frac{4\sqrt{5}}{3} & -\frac{5\sqrt{5}}{12} \\ 0 & 0 & \frac{5\sqrt{3}}{4} \\ 0 & \frac{\sqrt{10}}{3} & -\frac{2\sqrt{10}}{3} \end{bmatrix}.$$

Similarly $\boldsymbol{G_2}$ and $\boldsymbol{G_3}$ are constructed to reflect the second and third elements into the other two diagonal elements of $\boldsymbol{R_1^e}$ leading to

$$\boldsymbol{G} \begin{bmatrix} 3 & \frac{19}{3} & \frac{16}{3} \\ 0 & \frac{4\sqrt{5}}{3} & -\frac{5\sqrt{5}}{12} \\ 0 & 0 & \frac{5\sqrt{3}}{4} \\ 1 & 1 & 4 \end{bmatrix} = \begin{bmatrix} \sqrt{10} & 2\sqrt{10} & 2\sqrt{10} \\ 0 & \sqrt{10} & -\frac{\sqrt{10}}{2} \\ 0 & 0 & \frac{\sqrt{30}}{2} \\ 0 & 0 & 0 \end{bmatrix}.$$

We see that the Givens matrices have 'zeroed' the bottom row so that $\boldsymbol{G\tilde{R}_1^{\tilde{e}}}$ is upper triangular:

$$\boldsymbol{G\tilde{R}_1^{\tilde{e}}} = \begin{bmatrix} \boldsymbol{R_1^{\tilde{e}}} \\ \boldsymbol{0}^T \end{bmatrix},$$

where $\boldsymbol{R_1^{\tilde{e}}} \in \mathbb{R}^{3\times3}$ is, as we will see, the desired upper triangular matrix.

The product $\boldsymbol{\tilde{Q}_1^{\tilde{e}} G}^T = \begin{bmatrix} \boldsymbol{Q_1^{\tilde{e}}} & \boldsymbol{q} \end{bmatrix}$ is orthogonal (since both $\boldsymbol{G}$ and $\boldsymbol{\tilde{Q}_1^{\tilde{e}}}$ are orthogonal) so $\boldsymbol{B_1^{\tilde{e}}}$ is now factored into an orthogonal matrix times an upper triangular matrix:

$$\boldsymbol{B_1^{\tilde{e}}} = \begin{bmatrix} \boldsymbol{Q_1^{\tilde{e}}} & \boldsymbol{q} \end{bmatrix} \begin{bmatrix} \boldsymbol{R_1^{\tilde{e}}} \\ \boldsymbol{0}^T \end{bmatrix};$$

the full expansion of this equation, for our example, is given in (9).

Since the bottom row of the upper triangular matrix is zero we can simplify to $\boldsymbol{B_1^{\tilde{e}}} = \boldsymbol{Q_1^{\tilde{e}} R_1^{\tilde{e}}}$, which is the required thin QR decomposition.

One may be tempted to calculate the polynomial coefficients by using the updated $\boldsymbol{Q_1^{\tilde{e}}}$ to directly calculate $\boldsymbol{Q_1^{\tilde{e}}}^T \boldsymbol{t}$; there is, however, a much more efficient way. Since

$$\begin{bmatrix} \boldsymbol{Q_1^{\tilde{e}}} & \boldsymbol{q} \end{bmatrix}^T \boldsymbol{t_1^{\tilde{e}}} = \left(\begin{bmatrix} \boldsymbol{Q_1^e} & \boldsymbol{0} \\ \boldsymbol{0}^T & 1 \end{bmatrix} \boldsymbol{G}^T \right)^T \begin{bmatrix} \boldsymbol{t_1^e} \\ t_{41} \end{bmatrix},$$

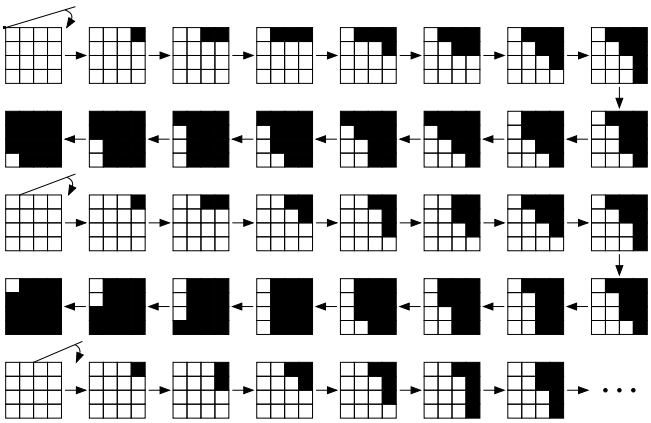
we can calculate the coefficients from

$$\boldsymbol{\hat{\theta}_{Q_1^{\tilde{e}}}} = \boldsymbol{Q_1^{\tilde{e}^T} t_1^{\tilde{e}}} = \underline{\boldsymbol{G}} \begin{bmatrix} \boldsymbol{\hat{\theta}_{Q_1^e}} \\ t_{41} \end{bmatrix}, \tag{10}$$

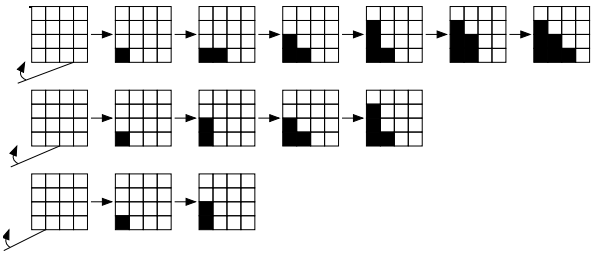
where $\underline{\boldsymbol{G}}$ is $\boldsymbol{G}$ with the last row removed. This is the key result because it allows us to update the coefficients without calculating $\boldsymbol{Q_1^{\tilde{e}}}$. All that is needed is to construct the Givens matrices using $\boldsymbol{R_1^e}$ and then the updated coefficients and upper triangular matrix are easily found. Once the coefficients have been calculated the error can efficiently be calculated from (8). The computational cost of the update is independent of the tile size and instead dependent on the degree of the polynomials used, which in our case is very small. This is because we require d Givens matrices, constructed to reflect the new row into the diagonal elements of $\boldsymbol{R_1^e} \in \mathbb{R}^{d\times d}$, to update the coefficients.

So far we have only told half the story: we also need to be able to calculate the coefficients $\boldsymbol{\hat{\theta}_{Q_2^{\tilde{e}}}}$ for the other side

$$\boldsymbol{B_1^{\tilde{e}}} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 2 \\ 1 & 1 & 3 \\ 1 & 1 & 4 \\ 1 & 2 & 1 \\ 1 & 2 & 2 \\ 1 & 2 & 3 \\ 1 & 3 & 1 \\ 1 & 3 & 2 \\ 1 & 4 & 1 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{10}}{10} & -\frac{\sqrt{10}}{10} & -\frac{\sqrt{30}}{10} & -\frac{\sqrt{2}}{10} \\ \frac{\sqrt{10}}{10} & -\frac{\sqrt{10}}{10} & -\frac{\sqrt{30}}{30} & \frac{\sqrt{2}}{10} \\ \frac{\sqrt{10}}{10} & -\frac{\sqrt{10}}{10} & \frac{\sqrt{30}}{30} & \frac{3\sqrt{2}}{10} \\ \frac{\sqrt{10}}{10} & -\frac{\sqrt{10}}{10} & \frac{\sqrt{30}}{10} & -\frac{5\sqrt{2}}{10} \\ \frac{\sqrt{10}}{10} & 0 & -\frac{\sqrt{30}}{15} & -\frac{\sqrt{2}}{10} \\ \frac{\sqrt{10}}{10} & 0 & 0 & \frac{\sqrt{2}}{10} \\ \frac{\sqrt{10}}{10} & 0 & \frac{\sqrt{30}}{15} & \frac{3\sqrt{2}}{10} \\ \frac{\sqrt{10}}{10} & \frac{\sqrt{10}}{10} & -\frac{\sqrt{30}}{30} & -\frac{\sqrt{2}}{10} \\ \frac{\sqrt{10}}{10} & \frac{\sqrt{10}}{10} & \frac{\sqrt{30}}{30} & \frac{\sqrt{2}}{10} \\ \frac{\sqrt{10}}{10} & \frac{\sqrt{10}}{5} & 0 & -\frac{\sqrt{2}}{10} \end{bmatrix} \begin{bmatrix} \sqrt{10} & 2\sqrt{10} & 2\sqrt{10} \\ 0 & \sqrt{10} & -\frac{\sqrt{10}}{2} \\ 0 & 0 & \frac{\sqrt{30}}{2} \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} \boldsymbol{Q_1^{\tilde{e}}} & \boldsymbol{q} \end{bmatrix} \begin{bmatrix} \boldsymbol{R_1^{\tilde{e}}} \\ \boldsymbol{0}^T \end{bmatrix} \tag{9}$$



(a) The start of a possible 4×4 dictionary of edges.



(b) The end of a possible 4×4 dictionary of edges.

Fig. 5: Part of a possible dictionary of 4×4 edge tiles that can be searched with the proposed strategy.

of the edge. This is done by updating the factorisation after a row has been removed from $\boldsymbol{B}_{\boldsymbol{2}}^{e}$. The process is almost the same as adding a row although we have the additional complication that the matrix may become rank deficient. The details of removing a row and maintaining full rank are given in Appendix A.

We search for the best edge using the above derivation as follows: we construct a dictionary of straight edges such that each edge differs from the previous entry by only one pixel. Constructing the dictionary in this way allows it to be exhaustively searched efficiently using the updating algorithm just described. Once all edges have been searched, the edge leading to the smallest (8) is selected.

There are many possible ways to construct a dictionary of edges that meet the above requirement. Figure 5 shows part of a possible dictionary of 4×4 tiles. This dictionary is created by initialising the tile so that all pixels are on the same side of the edge (the global 'no edge' case). Then, a straight line is rotated clockwise around a particular point on the boundary and when the line crosses the centre point of a pixel it is moved to the other side of the edge. When all pixels have been moved to the other side of the edge, we are back to the global 'no edge' case. The point of rotation is then moved one discrete step clockwise around the tile boundary and the process repeated. For example, the first two rows of Fig. 5(a) correspond to rotating the line around the top left corner, $(0, 0)$, and the next two rows correspond to rotating the edge around the point, $(0, 1)$, one step to the right.

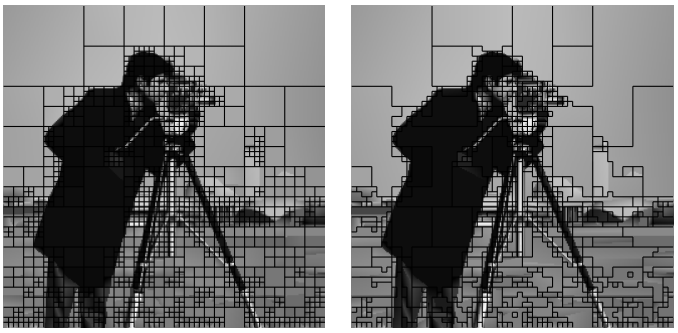
Note that it is possible to check all these edges in one continuous chain; however, resetting to the 'no edge' case, when possible, reduces rounding errors. Additionally this strategy allows the dictionary size to be reduced. As previously proposed, the dictionary has $4n^3$ edges for an $n \times n$ tile, since there are $4n$ rotation points and n^2 edges per point. This can be reduced to $2n^3 + n^2/2$ if we stop rotating the edge when we reach a boundary point we have already used as a rotation point. The intuition is that, since we have already checked edges starting and ending from approximately these locations, the vast majority of future edges will already have been checked. Figure 5(b) shows the last few edges of the same dictionary, when this strategy is employed. Despite the reduced size, the number of edges still has cubic growth, so we only exhaustively search tiles up to a size of 32×32. Larger tiles are down sampled and then exhaustively searched. This produces a rough approximation of the edge discontinuity which is refined at the larger, original tile size.

As can be seen in Fig. 5, constructing a dictionary of edges such that each edge differs from the previous entry in only one pixel results in checking some edges more than once. The reduced dictionary, just described, reduces this replication but it is still present. However, since the computational cost of moving a pixel from one side of the edge to the other is so cheap, we can tolerate this replication.

In many cases, we can further reduce the computation by pre-computing and storing the Givens matrices, for the whole dictionary of possible edges, offline. This makes the computation required to update the coefficients from one edge to the next very small. In practise we precompute and store these Givens matrices for square tiles up to 32×32 used in the pruning algorithm, but not for the more flexible regions that can be obtained when joining.

D. Example speed and sparsity of our approximation algorithm for a natural image

To conclude this section we show an example of the speed improvement that is obtained by updating the QR decomposition in the way that we have presented. We also use the example to demonstrate the sparsity of our model. Figure 6 shows two approximations of the cameraman image that have a peak signal-to-noise



(a) Reconstruction using the prune model only.

(b) Reconstruction using the prune-join model.

Fig. 6: Approximation of the cameraman image to a PSNR of 30dB using the prune and prune-join models.

ratio (PSNR[1]) of 30dB. The first approximation was calculated in 1.0 seconds using the prune only model and the second was calculated in 8.3 seconds using the more complex prune-join model[2]; for comparison these approximations would take around 100 and 10000 seconds if we calculated the QR decomposition from scratch each time.

For a rough comparison of sparsity, the prune and prune-join models use 3602 and 2753 polynomial coefficients respectively, whereas a Daubechies 4 tap wavelet decomposition would require 4712 coefficients to achieve the same approximation error. This greater sparsity should aid us in restoration, particularly in cases of high degradation where a strong prior is required.

IV. Denoising

In this section we will adapt the previously presented approximation algorithm to tackle the well studied denoising problem. Over the years there has been such a vast amount of research in this area that a comparison to the state of the art will provide a very stringent test of our modelling technique.

We define the denoising problem as approximating $\boldsymbol{x}$ from $\boldsymbol{y}$ where

$$\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{z}$$

and $\boldsymbol{z}$ is white Gaussian noise.

We approximate $\boldsymbol{x}$ from $\boldsymbol{y}$ by solving (7) using our prune-join algorithm, which can be interpreted as a maximum a posteriori (MAP) estimator using a probabilistic framework. If we define the probability of a particular image $\boldsymbol{x}$ occurring to be

$$p(\boldsymbol{x}) = A exp\left[-\frac{\zeta P^x(\boldsymbol{x})}{2}\right],$$

where $P^x(\boldsymbol{x})$ is given in (5) and A is a constant so that

$$\sum_{\boldsymbol{x}\in\mathbb{R}^N} A exp\left[-\frac{\zeta P^x(\boldsymbol{x})}{2}\right] = 1,$$

then the MAP estimator is

$$\hat{\boldsymbol{x}} = arg\min_{\boldsymbol{x}} \left\{ \|\boldsymbol{y}-\boldsymbol{x}\|_2^2 + \zeta\sigma_z^2 P^x(\boldsymbol{x}) \right\}. \qquad (11)$$

Equation (11) is equivalent to (7) with $\lambda = \zeta\sigma_z^2$. We have replaced the unknown constant λ with another unknown constant ζ. However, ζ is independent of σ_z so can be tuned experimentally just once (in all our simulations $\zeta = 3.3$).

A. Cycle spinning

Our quadtree decomposition approximation algorithm is shift variant allowing us to construct multiple approximations from different shifts of the image. Cycle spinning [12] is the process of averaging these shifts to construct a better approximation and improved results can be obtained by performing weighted averaging [28].

We can reduce the complexity of computing an approximation for each new shift by noticing that $N_i \times N_i$ tiles only have N_i^2 unique shifts. For example 2×2 tiles only have four unique shifts. This means that only the first four shifts have to calculate 2×2 tiles and all future shifts can simply look up these results from previous trees.

B. Example tilings

Figure 7 shows examples of the tilings generated by four different denoising experiments, calculated using 256 shifts of cycle spinning and the prune-only model. Note that there is a slightly different tiling for each shift but the figure just shows one of these tilings. One can see that, even in the presence of noise, the adaptive regions sensibly fit the data and, since λ is proportional to the noise variance, a coarser model is used in higher degradation cases.

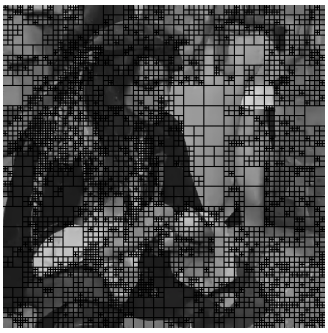
V. Interpolation

With very minor modifications, the previously described approximation algorithm can be used for interpolation. We model the problem by setting

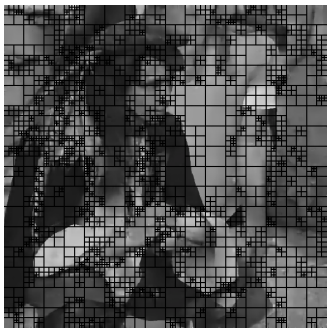
$$\boldsymbol{y} = \boldsymbol{H}\boldsymbol{x},$$

[1] All PSNRs in this paper are calculated as $10\log_{10}\left(\frac{255^2}{\frac{1}{N}\|\boldsymbol{x_1}-\boldsymbol{x_2}\|_2^2}\right)$.

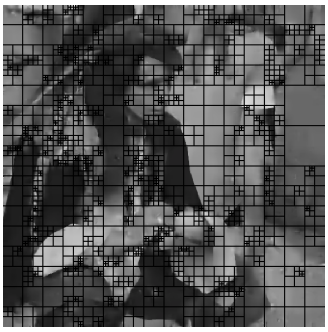
[2] All calculations were made in MATLAB on a 2.2GHz Intel Core i7 Macbook Pro with 4GB of RAM (no multi-threading).



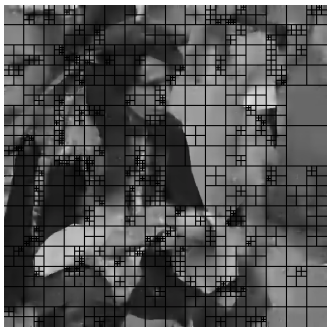
(a) Tiling with $\sigma_z = 25$.



(b) Tiling with $\sigma_z = 50$.

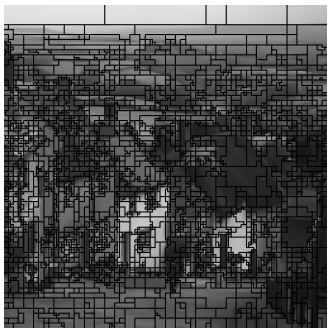


(c) Tiling with $\sigma_z = 75$.

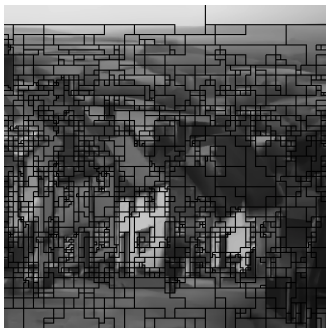


(d) Tiling with $\sigma_z = 100$.

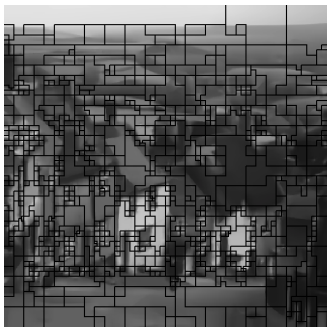
Fig. 7: Example tilings for the denoising algorithm with different noise levels.



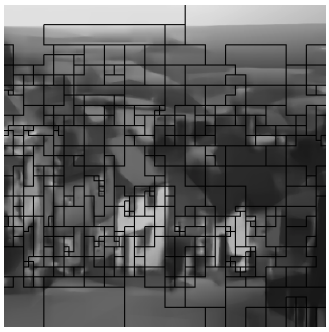
(a) Tiling with 75% missing.



(b) Tiling with 85% missing.



(c) Tiling with 90% missing.



(d) Tiling with 95% missing.

Fig. 8: Example tilings for the interpolation algorithm with different percentages of missing pixels.

where $\boldsymbol{H} \in \mathbb{R}^{N_v \times N}$ is the identity matrix but with the rows corresponding to the unknown pixels removed. Here $\boldsymbol{x} \in \mathbb{R}^N$ is the desired image and $\boldsymbol{y} \in \mathbb{R}^{N_v}$ is a truncated vector over just the, N_v, known or visible pixels. We will assume that we know $\boldsymbol{H}$ which is equivalent to saying we know the locations of the available samples. The approximation problem in this framework can be posed as follows:

$$\begin{aligned}\hat{\boldsymbol{\theta}} &= arg\min_{\boldsymbol{\theta}} \left[\|\boldsymbol{y} - HD(\boldsymbol{\theta})\|_2^2 + \lambda \tilde{P}^{\theta}(\boldsymbol{\theta}) \right] \\ &= arg\min_{\boldsymbol{\theta}} \left[\|\boldsymbol{y} - D_v(\boldsymbol{\theta})\|_2^2 + \lambda \tilde{P}^{\theta}(\boldsymbol{\theta}) \right], \end{aligned} \tag{12}$$

where D_v is the corresponding truncated quadtree representation over just the visible pixels given the parameters $\boldsymbol{\theta}$. This truncated quadtree representation can be calculated by putting holes in the polynomial subspace basis functions where there is a missing pixel. Interpolation is achieved by reconstructing with the corresponding functions with no holes.

Equation (12) uses a modified penalty, $\tilde{P}^{\theta}(\boldsymbol{\theta})$, that is almost identical to the previous described penalty, $P^{\theta}(\boldsymbol{\theta})$. The only difference is that the cost of a polynomial region is increased by a factor of $\frac{N_i}{N_i^v}$, where N_i and N_i^v are the number of pixels and the number of visible pixels in the i-th region respectively. This modification increases the penalty on regions with fewer known pixels resulting in a sparser model, and obviously when $N_i = N_i^v$ the penalty is as previously defined.

Modifying the penalty in this way allows successful interpolation over a wide range of images and sampling rates with a fixed λ, chosen once experimentally. In the following simulations $\lambda = 50$; however, in some cases we could have obtained more accurate results by optimising λ for the particular experiment.

A. Example tilings

Figure 8 shows examples of the tilings generated by four different interpolation experiments, calculated using 256 shifts of cycle spinning and the full prune-join model. Like the denoising case, a coarser model is used in high degradation cases, i.e. when less samples are available, and the regions fit the data, even with missing pixels.

VI. Simulation results

In this section we test the performance of the proposed algorithms on artificially degraded natural and depth images and provide comparisons to the state of the art[3].

[3]The depth images were obtained from the stereo data set [10] and then truncated and downsampled to the given dimensions. In the aid of reproducible research, exact copies of the images and the executables of our code are available from http://www.commsp.ee.ic.ac.uk/%7Eajs03/.

	Degradation		PSNR Denoising Results				SSIM Index Denoising Results			
Image	σ_z	Degraded	BM3DSAPCA	Proposed	PLOW	KSVD	Proposed	BM3DSAPCA	KSVD	PLOW
Boat	10	28.14	**34.10**	33.34	33.02	33.69	0.8763	**0.8923**	0.8834	0.8661
(512×512)	25	20.18	**30.03**	29.35	29.53	29.34	0.7832	**0.8039**	0.7723	0.7909
	50	14.16	**26.89**	26.55	26.60	25.92	0.6905	**0.7082**	0.6571	0.6910
	75	10.63	24.92	**25.01**	24.69	24.02	**0.6341**	0.6276	0.5812	0.6024
	100	8.14	23.69	**23.97**	23.33	22.85	**0.5948**	0.5789	0.5286	0.5308
Cameraman	10	28.10	**34.59**	33.24	33.17	33.74	0.9261	**0.9352**	0.9270	0.9187
(256×256)	25	20.14	**29.81**	29.00	28.59	29.00	0.8484	**0.8642**	0.8402	0.8400
	50	14.12	**26.58**	26.05	25.61	25.76	0.7834	**0.7868**	0.7497	0.7453
	75	10.60	**24.41**	24.35	23.61	23.41	**0.7442**	0.7051	0.6636	0.6224
	100	8.10	22.88	**23.08**	22.22	21.56	**0.7102**	0.6445	0.5775	0.5294
Hill	10	28.14	**33.83**	33.05	32.62	33.38	0.8683	**0.8896**	0.8778	0.8565
(512×512)	25	20.18	**29.96**	29.36	29.56	29.22	0.7522	**0.7788**	0.7392	0.7633
	50	14.16	**27.20**	26.94	26.95	26.30	0.6559	**0.6756**	0.6222	0.6567
	75	10.63	25.42	**25.63**	25.24	24.89	**0.6034**	0.5983	0.5644	0.5764
	100	8.14	24.27	**24.66**	24.06	24.01	**0.5654**	0.5492	0.5279	0.5171
Lena	10	28.14	**36.07**	35.24	35.32	35.57	0.9075	**0.9183**	0.9118	0.9077
(512×512)	25	20.18	**32.22**	31.40	31.87	31.37	0.8517	**0.8650**	0.8436	0.8571
	50	14.16	**29.07**	28.65	28.66	27.82	0.7962	**0.8014**	0.7612	0.7754
	75	10.63	26.83	**27.11**	26.55	25.82	**0.7597**	0.7247	0.6973	0.6943
	100	8.14	25.37	**26.00**	25.08	24.54	**0.7317**	0.6747	0.6451	0.6312
Man	10	28.14	**34.25**	33.57	32.98	33.64	0.9008	**0.9125**	0.9020	0.8879
(512×512)	25	20.18	**29.81**	29.39	29.33	29.12	0.7961	**0.8111**	0.7798	0.7967
	50	14.16	**26.94**	26.75	26.56	26.08	0.6995	**0.7107**	0.6652	0.6858
	75	10.63	25.13	**25.28**	24.86	24.44	**0.6421**	0.6308	0.5967	0.6002
	100	8.14	23.96	**24.31**	23.68	23.42	**0.6049**	0.5793	0.5502	0.5397
Peppers	10	28.10	**34.94**	34.45	33.60	34.29	0.9272	**0.9287**	0.9241	0.9186
(256×256)	25	20.14	**30.43**	30.11	29.63	29.71	**0.8710**	0.8690	0.8563	0.8571
	50	14.12	**27.00**	26.83	26.38	26.08	**0.8039**	0.7942	0.7715	0.7570
	75	10.60	24.74	**24.94**	24.26	23.64	**0.7575**	0.7220	0.6906	0.6595
	100	8.10	23.24	**23.61**	22.76	21.96	**0.7207**	0.6726	0.6237	0.5901
Average			**28.29**	28.04	27.68	27.49	**0.7602**	0.7551	0.7244	0.7222

TABLE I: PSNR and SSIM index comparisons of the proposed denoising algorithm for natural images. The denoising algorithms used for comparison are BM3DSAPCA [22], PLOW [29] and KSVD [30] . The best result is shown in bold and σ_z denotes the standard deviation of the noise.

(a) Original. (b) Noisy, $\sigma_z = 75$. (c) Denoised as proposed. (d) Denoised with [22].

Fig. 9: Visual comparison of the proposed denoising algorithm with the BM3D-SAPCA algorithm [22] for the Lena natural image.

The PSNR and SSIM index [31] objective measures are quoted for all simulations.

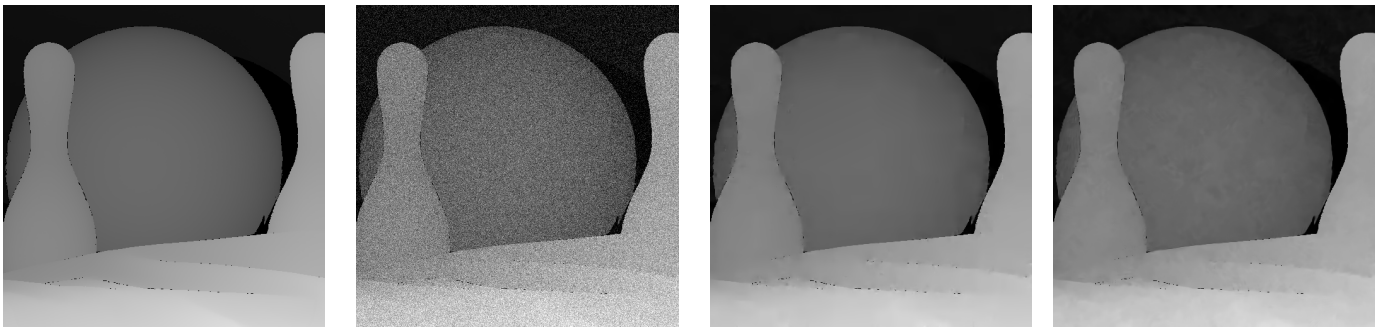
A. Denoising

We tested the denoising algorithm on a range of natural and depth images with additive white Gaussian noise as given by the degradation model (1). The results for natural and depth images are presented in Tables I and II respectively. Visual comparisons can be made from Figs. 9 and 10. All these results were calculated without joining and using 256 shifts of cycle spinning.

We see that, although natural images have many complex structures, such as texture, which are not well approximated by our model, we still perform competitively. In particular when the noise level is high we often

	Degradation		PSNR Denoising Results				SSIM Index Denoising Results			
Image	σ_z	Degraded	Proposed	BM3DSAPCA	KSVD	PLOW	Proposed	BM3DSAPCA	KSVD	PLOW
Aloe	10	28.14	41.83	**42.35**	41.16	36.17	**0.9908**	0.9901	0.9832	0.9699
(512 × 512)	25	20.18	**35.51**	34.24	32.91	31.84	**0.9652**	0.9549	0.9308	0.9263
	50	14.16	**30.91**	30.00	28.77	28.91	**0.9202**	0.8949	0.8438	0.8386
	75	10.63	**28.80**	27.50	26.64	27.07	**0.8866**	0.7910	0.7683	0.7454
	100	8.14	**27.57**	25.95	25.27	25.76	**0.8636**	0.7298	0.7039	0.6790
Art	10	28.14	41.80	**42.82**	41.08	36.58	**0.9930**	0.9917	0.9854	0.9771
(512 × 512)	25	20.18	**35.58**	34.77	33.65	31.17	**0.9725**	0.9634	0.9465	0.9293
	50	14.16	**30.33**	29.35	28.11	28.23	**0.9277**	0.8959	0.8465	0.8397
	75	10.63	**28.17**	27.03	26.06	26.50	**0.8947**	0.7985	0.7721	0.7474
	100	8.14	**26.81**	25.59	24.84	25.28	**0.8674**	0.7421	0.7106	0.6838
Baby	10	28.14	**45.40**	44.65	42.73	39.57	**0.9958**	0.9941	0.9878	0.9851
(512 × 512)	25	20.18	**38.44**	37.08	35.71	35.88	**0.9825**	0.9733	0.9542	0.9572
	50	14.16	**34.44**	32.62	31.53	32.27	**0.9634**	0.9301	0.8894	0.8846
	75	10.63	**32.89**	29.67	29.20	30.05	**0.9518**	0.8309	0.8233	0.8066
	100	8.14	**31.77**	27.72	27.49	28.58	**0.9421**	0.7635	0.7562	0.7535
Bowling ball	10	28.14	45.91	**46.23**	44.12	39.80	**0.9953**	0.9947	0.9873	0.9862
(512 × 512)	25	20.18	**39.75**	38.07	36.24	34.93	**0.9858**	0.9777	0.9558	0.9540
	50	14.16	**34.72**	32.71	31.25	31.62	**0.9673**	0.9342	0.8853	0.8829
	75	10.63	**32.69**	29.81	28.82	29.44	**0.9550**	0.8362	0.8136	0.8007
	100	8.14	**31.33**	27.99	27.04	27.88	**0.9438**	0.7820	0.7422	0.7339
Average			**34.91**	33.59	32.40	31.56	**0.9482**	0.8885	0.8643	0.8541

TABLE II: PSNR and SSIM index comparisons of the proposed denoising algorithm for depth images. The denoising algorithms used for comparison are BM3DSAPCA [22], PLOW [29] and KSVD [30] . The best result is shown in bold and σ_z denotes the standard deviation of the noise.



(a) Original. (b) Noisy, $\sigma_z = 25$. (c) Denoised as proposed. (d) Denoised with [22].

Fig. 10: Visual comparison of the proposed denoising algorithm with the BM3D-SAPCA algorithm [22] for the bowling ball depth image.

outperform the current state of the art because, in these situations, it is only possible to restore a coarse version of the image, which is very often piecewise smooth.

For depth images, which are of course much closer to our model, we outperform the current state of the art in almost all cases.

B. Interpolation from irregularly sampled data

We tested our interpolation algorithm by randomly removing between 75% and 95% of the image pixels. Tables III and IV show the interpolation results for natural and depth images respectively and Figs. 11 and 12 provide visual comparisons. For natural images we used the full prune-join algorithm with 256 shifts of cycle spinning and for depth images we, again, used the full prune-join algorithm, but with just 64 shifts. The proposed method and non local interpolation algorithm are unsupervised, since we use a fixed λ. In order for a fair comparison we used the fixed parameters, given in the kernel regression software, whenever possible. However, in order to produce competitive results in high degradation cases, we tuned the kernel size when 95% of the pixels were removed.

Like denoising we are behind non local methods on natural images in low degradation cases; however, when the degradation is high, we often produce state of the art results. Finally, for depth images we produce the best performance in all cases.

	Degradation		PSNR Interpolation Results				SSIM Index Interpolation Results			
Image	%	Degraded	NL	Proposed	KR	Bi-Cubic	NL	Proposed	KR	Bi-Cubic
Boat	75	6.60	**29.38**	28.67	27.84	27.41	**0.8573**	0.8323	0.8153	0.8065
(512 × 512)	80	6.32	**28.28**	27.69	27.19	26.57	**0.8286**	0.7991	0.7910	0.7762
	85	6.05	**26.79**	26.40	26.04	25.57	**0.7878**	0.7534	0.7554	0.7382
	90	5.80	25.14	**25.14**	23.96	24.34	**0.7329**	0.6970	0.6908	0.6863
	95	5.56	23.10	**23.39**	21.40	22.52	**0.6406**	0.6055	0.5901	0.6067
Cameraman	75	6.83	**25.33**	24.95	24.73	23.88	**0.8677**	0.8459	0.8379	0.8253
(256 × 256)	80	6.56	**24.19**	24.16	23.90	23.11	**0.8399**	0.8210	0.8146	0.7996
	85	6.29	22.94	**23.15**	22.63	22.05	**0.8032**	0.7856	0.7812	0.7639
	90	6.04	21.86	**22.42**	20.95	21.14	**0.7599**	0.7476	0.7302	0.7221
	95	5.81	20.13	**20.81**	17.68	19.36	**0.6915**	0.6744	0.6430	0.6551
Hill	75	7.62	**30.44**	29.91	29.44	28.99	**0.8487**	0.8181	0.8113	0.8117
(512 × 512)	80	7.34	**29.42**	29.08	28.76	28.19	**0.8188**	0.7865	0.7869	0.7813
	85	7.08	**28.19**	28.06	27.73	27.31	**0.7781**	0.7448	0.7518	0.7424
	90	6.83	26.89	**26.96**	25.82	26.25	**0.7253**	0.6886	0.6878	0.6920
	95	6.59	24.99	**25.19**	23.84	24.22	**0.6406**	0.6030	0.5987	0.6167
Lena	75	6.93	**33.18**	32.21	32.10	31.31	**0.9102**	0.8945	0.8967	0.8885
(512 × 512)	80	6.65	**31.80**	31.07	31.37	30.20	**0.8924**	0.8754	0.8834	0.8702
	85	6.38	**30.24**	29.74	30.09	29.00	**0.8711**	0.8499	0.8626	0.8472
	90	6.14	**28.36**	28.11	27.13	27.48	**0.8373**	0.8117	0.8127	0.8124
	95	5.90	25.67	**26.14**	24.91	24.94	**0.7793**	0.7551	0.7587	0.7548
Man	75	7.70	**29.49**	29.12	28.99	28.49	**0.8700**	0.8407	0.8415	0.8445
(512 × 512)	80	7.42	**28.47**	28.26	28.24	27.58	**0.8424**	0.8103	0.8197	0.8153
	85	7.15	**27.37**	27.23	27.19	26.66	**0.8078**	0.7692	0.7886	0.7799
	90	6.91	26.02	**26.03**	25.08	25.46	**0.7562**	0.7144	0.7284	0.7288
	95	6.67	24.18	**24.30**	22.50	23.55	**0.6699**	0.6305	0.6301	0.6476
Peppers	75	6.83	**29.37**	27.22	25.78	26.71	**0.9091**	0.8926	0.8922	0.8783
(256 × 256)	80	6.55	**27.80**	26.07	25.10	25.56	**0.8838**	0.8701	0.8751	0.8547
	85	6.29	**26.67**	24.84	24.14	24.50	**0.8657**	0.8427	0.8491	0.8278
	90	6.04	**25.18**	23.40	21.93	23.44	**0.8312**	0.7980	0.7897	0.7896
	95	5.81	**22.65**	21.60	20.15	21.35	**0.7597**	0.7170	0.7081	0.7150
Average			**26.78**	26.38	25.55	25.57	**0.8036**	0.7758	0.7741	0.7693

TABLE III: PSNR and SSIM index comparisons of the proposed interpolation algorithm for natural images. The interpolation algorithms used for comparison are NL, the non-local algorithm presented in [27], KR, adaptive kernel regression [26], and bi-cubic interpolation. The best result is shown in bold and % denotes the percentage of pixels that have been randomly removed.

(a) Original. (b) 90% Missing Pixels. (c) Interpolated as preposed. (d) Interpolated with [27].

Fig. 11: Visual comparison of the proposed interpolation algorithm with the non-local algorithm presented in [27] for the hill natural image.

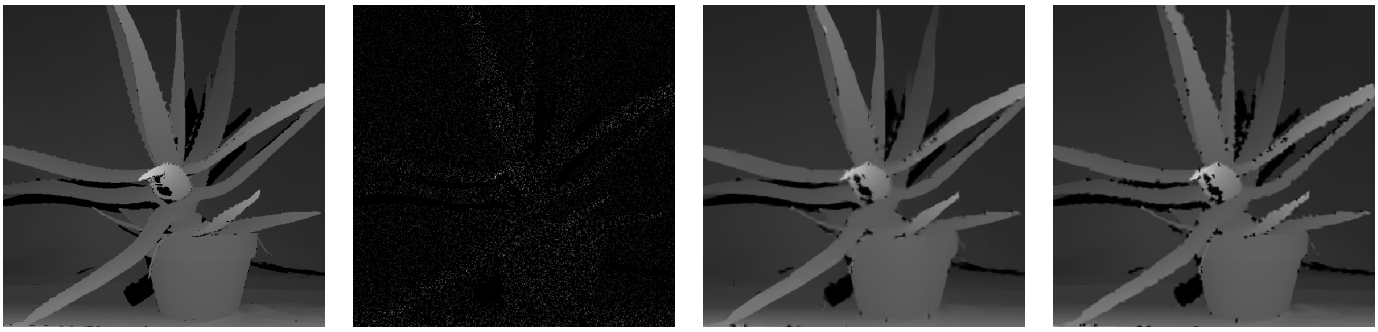
VII. Conclusion

We have presented a novel quadtree structured image approximation algorithm, which can be used for image denoising and interpolation. The algorithm achieves state of the art performance when the original image is very close to our piecewise polynomial model and is very competitive for natural images in high degradation cases. The efficiency of our algorithm has been vastly improved by exploiting the mathematics of updating matrix factorisations.

Simulation results suggest that this framework would be ideal for depth image applications; however, we note that in many practical depth sensing setups a colour image is obtained as well as a depth image. Exploiting the additional information provided by this colour image

	Degradation		PSNR Interpolation Results				SSIM Index Interpolation Results			
Image	%	Degraded	Proposed	NL	KR	Bi-Cubic	Proposed	NL	KR	Bi-Cubic
Aloe	75	11.17	**31.05**	30.60	30.73	29.32	**0.9532**	0.9516	0.9491	0.9303
(512 × 512)	80	10.89	**30.42**	29.87	29.89	28.77	**0.9454**	0.9426	0.9390	0.9202
	85	10.62	**29.64**	28.82	28.54	27.97	**0.9344**	0.9283	0.9243	0.9062
	90	10.37	**28.72**	27.83	26.79	27.15	**0.9193**	0.9099	0.8952	0.8883
	95	10.14	**27.21**	26.23	25.61	25.60	**0.8871**	0.8767	0.8700	0.8584
Art	75	6.67	**29.76**	29.03	29.33	28.13	**0.9469**	0.9433	0.9411	0.9257
(512 × 512)	80	6.39	**29.11**	28.19	28.37	27.42	**0.9392**	0.9337	0.9319	0.9153
	85	6.12	**28.42**	27.65	27.48	27.02	**0.9286**	0.9241	0.9200	0.9054
	90	5.88	**27.52**	26.76	25.47	26.21	**0.9122**	0.9080	0.8951	0.8886
	95	5.64	**26.03**	25.24	24.67	24.37	**0.8784**	0.8768	0.8701	0.8600
Baby	75	11.80	**35.97**	34.82	35.58	34.19	**0.9774**	0.9738	0.9759	0.9691
(512 × 512)	80	11.52	**35.45**	34.15	34.91	33.63	**0.9745**	0.9699	0.9723	0.9651
	85	11.25	**34.50**	33.15	33.62	32.83	**0.9700**	0.9633	0.9656	0.9594
	90	11.01	**33.81**	32.29	32.04	32.16	**0.9649**	0.9563	0.9557	0.9530
	95	10.77	**32.53**	30.65	31.20	30.97	**0.9540**	0.9456	0.9465	0.9422
Bowling ball	75	7.50	**34.59**	32.94	33.73	32.14	**0.9789**	0.9758	0.9767	0.9691
(512 × 512)	80	7.22	**33.88**	32.07	32.88	31.35	**0.9757**	0.9710	0.9725	0.9643
	85	6.95	**33.36**	31.41	31.54	30.82	**0.9720**	0.9665	0.9658	0.9592
	90	6.70	**32.21**	30.43	29.31	29.93	**0.9656**	0.9586	0.9542	0.9507
	95	6.46	**30.34**	29.11	29.39	27.89	**0.9497**	0.9480	0.9482	0.9397
Average			**31.23**	30.06	30.05	29.39	**0.9464**	0.9412	0.9385	0.9285

TABLE IV: PSNR and SSIM index comparisons of the proposed interpolation algorithm for depth images. The interpolation algorithms used for comparison are NL, the non-local algorithm presented in [27], KR, adaptive kernel regression [26], and bi-cubic interpolation. The best result is shown in bold and % denotes the percentage of pixels that have been randomly removed.



(a) Original. (b) 90% Missing Pixels. (c) Interpolated as preposed. (d) Interpolated with [27].

Fig. 12: Visual comparison of the proposed interpolation algorithm with the non-local algorithm presented in [27] for the aloe depth image.

is an area of potential future work. Further research into improved regularisation parameter selection may also produce improved results, particularly for interpolation.

APPENDIX A
UPDATING A QR DECOMPOSITION

In Section III-C we gave an example of approximating a tile using two similar edges: the first approximation was calculated using the orthogonal matrix of the thin QR decomposition to directly calculate the inner products; the second approximation was calculated, at much reduced computational cost, by applying rank one updates to the first QR decomposition. We only gave an example of adding a row and neglected removing a row and the additional complications of maintaining full rank, which we will now address. In what follows we assume we have the thin QR decomposition of a full rank matrix $\boldsymbol{B} \in \mathbb{R}^{N\times d}$:

$$\boldsymbol{B} = \boldsymbol{Q}\boldsymbol{R},$$

where $\boldsymbol{Q} \in \mathbb{R}^{N\times d}$ is orthogonal and $\boldsymbol{R} \in \mathbb{R}^{d\times d}$ is upper triangular. In the next four subsections we summarise how to update the QR decomposition when we add a row to $\boldsymbol{B}$, remove a row from $\boldsymbol{B}$, add a column to $\boldsymbol{B}$ and remove a column from $\boldsymbol{B}$. This is covered in detail in [11] as well as many text books (e.g. [32] and [33]). In each of these subsections we will also derive the formulas to update the coefficients $\boldsymbol{Q}^T\boldsymbol{t}$: this material is a simple extension of the updating formulas however we have never seen it in print. In the fifth and final subsection of this appendix we will explain how we can

add and remove rows to $\boldsymbol{B}$, whilst preventing $\boldsymbol{B}$ from becoming rank deficient. This is done by adding and removing columns at the correct moment.

A. Adding a row to $\boldsymbol{B}$

The row $\boldsymbol{\beta}^T = \begin{bmatrix} \beta_1 & \beta_2 & \dots & \beta_d \end{bmatrix}$ can be added to $\boldsymbol{B}$ using

$$\boldsymbol{B}_+ = \begin{bmatrix} \boldsymbol{B} \\ \boldsymbol{\beta}^T \end{bmatrix} = \boldsymbol{Q}_+ \boldsymbol{R}_+,$$

where

$$\boldsymbol{Q}_+ = \begin{bmatrix} \boldsymbol{Q} & \boldsymbol{0} \\ \boldsymbol{0}^T & 1 \end{bmatrix} \underline{\boldsymbol{G}}^T \quad \text{and} \quad \boldsymbol{R}_+ = \underline{\boldsymbol{G}} \begin{bmatrix} \boldsymbol{R} \\ \boldsymbol{\beta}^T \end{bmatrix}.$$

$\boldsymbol{G} \in \mathbb{R}^{(d+1)\times(d+1)}$ is the product of d Givens matrices $\boldsymbol{G} = \boldsymbol{G_d G_{d-1}} \dots \boldsymbol{G_1}$ which reflect $\boldsymbol{\beta}^T$ into the diagonal elements of $\boldsymbol{R}$. $\boldsymbol{G_1}$ reflects β_1 into r_{11}, $\boldsymbol{G_2}$ reflects β_2 into r_{22} and so on:

$$\boldsymbol{G_d G_{d-1}} \dots \boldsymbol{G_1} \begin{bmatrix} \boldsymbol{R} \\ \boldsymbol{\beta}^T \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}_+ \\ \boldsymbol{0}^T \end{bmatrix}.$$

The coefficients ${\boldsymbol{Q}_+}^T \boldsymbol{t}_+$ for a tile vector $\boldsymbol{t}_+ \in \mathbb{R}^{N+1}$ can be calculated from

$$\begin{aligned} {\boldsymbol{Q}_+}^T \boldsymbol{t}_+ &= \left(\begin{bmatrix} \boldsymbol{Q} & \boldsymbol{0} \\ \boldsymbol{0}^T & 1 \end{bmatrix} \underline{\boldsymbol{G}}^T \right)^T \boldsymbol{t}_+ \\ &= \underline{\boldsymbol{G}} \begin{bmatrix} \boldsymbol{Q}^T & \boldsymbol{0} \\ \boldsymbol{0}^T & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{t} \\ \boldsymbol{t}_+[N+1] \end{bmatrix} \\ &= \underline{\boldsymbol{G}} \begin{bmatrix} \boldsymbol{Q}^T \boldsymbol{t} \\ \boldsymbol{t}_+[N+1] \end{bmatrix}. \end{aligned}$$

B. Removing a row from $\boldsymbol{B}$

Let $\boldsymbol{\beta}^T$ be the last row of the matrix $\boldsymbol{B}$ and $\boldsymbol{B}_- \in \mathbb{R}^{(N-1)\times d}$ be the the remainder of the matrix; i.e.,

$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{B}_- \\ \boldsymbol{\beta}^T \end{bmatrix}.$$

We can remove this row using

$$\boldsymbol{B}_- = \boldsymbol{Q}_- \boldsymbol{R}_-$$

where

$$\boldsymbol{Q}_- = \begin{bmatrix} \tilde{\boldsymbol{Q}}_- & -\frac{\tilde{\boldsymbol{Q}}_- \boldsymbol{\rho}}{r_{dd}} \end{bmatrix} \underline{\boldsymbol{G}}^T \quad \text{and} \quad \boldsymbol{R}_- = \underline{\boldsymbol{G}} \begin{bmatrix} \boldsymbol{R} \\ \boldsymbol{0}^T \end{bmatrix}.$$

Here, $\boldsymbol{\rho}^T = \begin{bmatrix} \rho_1 & \rho_2 & \dots & \rho_d \end{bmatrix}$ is the last row of $\boldsymbol{Q}$ and $\tilde{\boldsymbol{Q}}_-$ is the remainder of the matrix; i.e.,

$$\boldsymbol{Q} = \begin{bmatrix} \tilde{\boldsymbol{Q}}_- \\ \boldsymbol{\rho}^T \end{bmatrix}$$

and $r_{dd} = \sqrt{1 - \boldsymbol{\rho}^T \boldsymbol{\rho}}$.

$\boldsymbol{G} \in \mathbb{R}^{(d+1)\times(d+1)}$ is the product of d Givens matrices $\boldsymbol{G} = \boldsymbol{G_d G_{d-1}} \dots \boldsymbol{G_1}$ which reflect the elements of $\boldsymbol{\rho}$ into r_{dd}. $\boldsymbol{G_1}$ reflects ρ_d into r_{dd}, $\boldsymbol{G_2}$ reflects ρ_{d-1} into the modified r_{dd} and so on. This has the effect that

$$\begin{bmatrix} \tilde{\boldsymbol{Q}}_- & -\frac{\tilde{\boldsymbol{Q}}_- \boldsymbol{\rho}}{r_{dd}} \\ \boldsymbol{\rho}^T & r_{dd} \end{bmatrix} \boldsymbol{G}^T = \begin{bmatrix} \boldsymbol{Q}_- & \boldsymbol{0} \\ \boldsymbol{0}^T & \pm 1 \end{bmatrix}.$$

If $\boldsymbol{t}_- \in \mathbb{R}^{N-1}$ is the tile vector over the reduced region, i.e.

$$\boldsymbol{t} = \begin{bmatrix} \boldsymbol{t}_- \\ \boldsymbol{t}[N] \end{bmatrix},$$

then the expression for the coefficients, ${\boldsymbol{Q}_-}^T \boldsymbol{t}_-$, can be derived as

$$\begin{aligned} \begin{bmatrix} \boldsymbol{Q}_- & \boldsymbol{0} \\ \boldsymbol{0}^T & \pm 1 \end{bmatrix}^T \boldsymbol{t} &= \left(\begin{bmatrix} \tilde{\boldsymbol{Q}}_- & -\frac{\tilde{\boldsymbol{Q}}_- \boldsymbol{\rho}}{r_{dd}} \\ \boldsymbol{\rho}^T & r_{dd} \end{bmatrix} \boldsymbol{G}^T \right)^T \boldsymbol{t} \\ &= \boldsymbol{G} \begin{bmatrix} {\tilde{\boldsymbol{Q}}_-}^T & \boldsymbol{\rho} \\ -\frac{\boldsymbol{\rho}^T {\tilde{\boldsymbol{Q}}_-}^T}{r_{dd}} & r_{dd} \end{bmatrix} \boldsymbol{t} \\ &= \boldsymbol{G} \begin{bmatrix} \boldsymbol{Q}^T \boldsymbol{t} \\ \frac{r_{dd}^2 \boldsymbol{t}[N] - \boldsymbol{\rho}^T (\boldsymbol{Q}^T \boldsymbol{t} - \boldsymbol{\rho} \boldsymbol{t}[N])}{r_{dd}} \end{bmatrix} \\ {\boldsymbol{Q}_-}^T \boldsymbol{t}_- &= \underline{\boldsymbol{G}} \begin{bmatrix} \boldsymbol{Q}^T \boldsymbol{t} \\ \frac{\boldsymbol{t}[N] - \boldsymbol{\rho}^T \boldsymbol{Q}^T \boldsymbol{t}}{\sqrt{1 - \boldsymbol{\rho}^T \boldsymbol{\rho}}} \end{bmatrix}. \end{aligned} \tag{13}$$

C. Adding a column to $\boldsymbol{B}$

To add a column $\boldsymbol{b_i}$ into the i-th position of $\boldsymbol{B}$ we first need to split $\boldsymbol{B}$ into two matrices $\boldsymbol{B_L}$ and $\boldsymbol{B_R}$ where $\boldsymbol{B_L}$ contains the first $i-1$ columns of $\boldsymbol{B}$ and $\boldsymbol{B_R}$ the last $d-i+1$ columns. We use similar notation to split $\boldsymbol{R}$:

$$\boldsymbol{B} \equiv \begin{bmatrix} \boldsymbol{B_L} & \boldsymbol{B_R} \end{bmatrix} = \boldsymbol{Q} \begin{bmatrix} \boldsymbol{R_L} & \boldsymbol{R_R} \end{bmatrix} \equiv \boldsymbol{QR}. \tag{14}$$

The new column b_i is inserted using

$$\boldsymbol{B}_+ \equiv \begin{bmatrix} \boldsymbol{B_L} & \boldsymbol{b_i} & \boldsymbol{B_R} \end{bmatrix} = \boldsymbol{Q}_+ \boldsymbol{R}_+,$$

where

$$\begin{aligned} \boldsymbol{Q}_+ &= \begin{bmatrix} \boldsymbol{Q} & \frac{\boldsymbol{b_i} - \boldsymbol{Q}\boldsymbol{Q}^T \boldsymbol{b_i}}{r_{dd}} \end{bmatrix} \boldsymbol{G}^T, \\ \boldsymbol{R}_+ &= \boldsymbol{G} \begin{bmatrix} \boldsymbol{R_L} & \boldsymbol{Q}^T \boldsymbol{b_i} & \boldsymbol{R_R} \\ \boldsymbol{0}^T & r_{dd} & \boldsymbol{0}^T \end{bmatrix} \text{ and} \\ r_{dd} &= \sqrt{\|\boldsymbol{b_i}\|^2 - \|\boldsymbol{Q}^T \boldsymbol{b_i}\|^2}. \end{aligned}$$

$\boldsymbol{G} \in \mathbb{R}^{(d+1)\times(d+1)}$ is the product of $d-i$ Givens matrices, $\boldsymbol{G} = \boldsymbol{G_{d-i} G_{d-i-1}} \dots \boldsymbol{G_1}$, constructed to introduce zeros in the bottom $d-i$ entries of $\begin{bmatrix} \boldsymbol{Q}^T \boldsymbol{b_i} \\ r_{dd} \end{bmatrix}$. $\boldsymbol{G_1}$ reflects r_{dd} into $\boldsymbol{Q}^T \boldsymbol{b_i}[d-1]$, $\boldsymbol{G}_2$ reflects the updated $\boldsymbol{Q}^T \boldsymbol{b_i}[d-1]$ into $\boldsymbol{Q}^T \boldsymbol{b_i}[d-2]$ and so on.

To calculate the coefficients $\boldsymbol{Q_+}^T\boldsymbol{t}$ for a tile vector $\boldsymbol{t} \in \mathbb{R}^N$ we can use

$$\begin{aligned}\boldsymbol{Q_+}^T\boldsymbol{t} &= \left(\left[\begin{array}{cc}\boldsymbol{Q} & \frac{\boldsymbol{b_i}-\boldsymbol{Q}\boldsymbol{Q}^T\boldsymbol{b_i}}{r_{dd}}\end{array}\right]\boldsymbol{G}^T\right)^T t \\ &= \boldsymbol{G}\left[\begin{array}{c}\boldsymbol{Q}^T\boldsymbol{t} \\ \left(\frac{\boldsymbol{b_i}-\boldsymbol{Q}\boldsymbol{Q}^T\boldsymbol{b_i}}{r_{dd}}\right)^T\boldsymbol{t}\end{array}\right] \\ &= \boldsymbol{G}\left[\begin{array}{c}\boldsymbol{Q}^T\boldsymbol{t} \\ \frac{\boldsymbol{b_i}^T\boldsymbol{t}-(\boldsymbol{Q}^T\boldsymbol{b_i})^T\boldsymbol{Q}^T\boldsymbol{t}}{\sqrt{\|\boldsymbol{b_i}\|^2-\|\boldsymbol{Q}^T\boldsymbol{b_i}\|^2}}\end{array}\right].\end{aligned}$$

D. Removing a column from $\boldsymbol{B}$

To remove the i-th column, $\boldsymbol{b_i}$, from $\boldsymbol{B}$ we need to split $\boldsymbol{B}$ and $\boldsymbol{R}$:

$$\boldsymbol{B} \equiv \left[\begin{array}{ccc}\boldsymbol{B_L} & \boldsymbol{b_i} & \boldsymbol{B_R}\end{array}\right] = \boldsymbol{Q}\left[\begin{array}{ccc}\boldsymbol{R_L} & \boldsymbol{r_i} & \boldsymbol{R_R}\end{array}\right] \equiv \boldsymbol{QR} \tag{15}$$

where $\boldsymbol{B_L}$ and $\boldsymbol{B_R}$ are the first $i-1$ and last $d-i$ columns of $\boldsymbol{B}$ respectively and similarly for $\boldsymbol{R_L}$, $\boldsymbol{R_R}$ and $\boldsymbol{R}$.

We can remove $\boldsymbol{b_i}$ from the system using

$$\boldsymbol{B_-} \equiv \left[\begin{array}{cc}\boldsymbol{B_L} & \boldsymbol{B_R}\end{array}\right] = \boldsymbol{Q_-}\boldsymbol{R_-},$$

where

$$\boldsymbol{Q_-} = \boldsymbol{Q}\underline{\boldsymbol{G}}^T \quad \text{and} \quad \boldsymbol{R_-} = \underline{\boldsymbol{G}}\left[\begin{array}{cc}\boldsymbol{R_L} & \boldsymbol{R_R}\end{array}\right].$$

$\boldsymbol{G} \in \mathbb{R}^{d\times d}$ is the product of $d-i$ Givens matrices, $\boldsymbol{G} = \boldsymbol{G_{d-i}}\boldsymbol{G_{d-i-1}}\ldots\boldsymbol{G_1}$, constructed to reflect the $d-i$ entries of $\left[\begin{array}{cc}\boldsymbol{R_L} & \boldsymbol{R_R}\end{array}\right]$ that are below the diagonal into the diagonal. $\boldsymbol{G_1}$ reflects r_{dd} into r_{d-1d}, $\boldsymbol{G_2}$ reflects r_{d-1d-1} into r_{d-2d-1} and so on.

The calculation of the coefficients, $\boldsymbol{Q_-}^T\boldsymbol{t}$, for a tile vector $\boldsymbol{t} \in \mathbb{R}^N$ is very simple in this case:

$$\boldsymbol{Q_-}^T\boldsymbol{t} = (\boldsymbol{Q}\underline{\boldsymbol{G}}^T)^T\boldsymbol{t} = \underline{\boldsymbol{G}}\boldsymbol{Q}^T\boldsymbol{t}.$$

E. Maintaining full rank when adding and removing rows

When moving a pixel from one side of the edge to the other, we need to add a row to one QR decomposition and remove a row from another. This process can produce a rank deficient system, if we do not take preventative measures.

We can see from (13) that removing a row will be problematic if $\boldsymbol{\rho}^T\boldsymbol{\rho} = 1$. We must therefore check, before removing a row, that $\boldsymbol{\rho}^T\boldsymbol{\rho} < 1$. If this is not the case we remove a column from the system to maintain full rank.

When adding a row the system cannot become rank deficient; however, it is possible that adding a row will allow a previously removed column to be added back to the system whilst maintaining full rank. A column $\boldsymbol{b}$ can be added to a QR decomposition without causing rank deficiencies if $\|\boldsymbol{b}\|^2 > \|\boldsymbol{Q}^T\boldsymbol{b}\|^2$.

REFERENCES

[1] A. Scholefield and P. L. Dragotti, "Quadtree structured restoration algorithms for piecewise polynomial images," in *Acoustics, Speech and Signal Processing, IEEE International Conference on*, 2009, pp. 705–708.

[2] ——, "Image restoration using a sparse quadtree decomposition representation," in *Image Processing, IEEE International Conference on*, 2009, pp. 1473–1476.

[3] R. Shukla, P. L. Dragotti, M. Do, and M. Vetterli, "Rate-distortion optimized tree-structured compression algorithms for piecewise polynomial images," *Image Processing, IEEE Transactions on*, vol. 14, no. 3, pp. 343–359, 2005.

[4] E. Candes and D. Donoho, "Ridgelets: a key to higher-dimensional intermittency?" *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, vol. 357, no. 1760, pp. 2495–2509, 1999.

[5] ——, "Curvelets: a surprisingly effective nonadaptive representation of objects with edges," in *Curve and Surface Fitting: Saint-Malo*. University Press, 2000, pp. 0–82 651 357.

[6] M. Do and M. Vetterli, "The contourlet transform: an efficient directional multiresolution image representation," *Image Processing, IEEE Transactions on*, vol. 14, no. 12, pp. 2091–2106, 2005.

[7] D. Donoho, "Wedgelets: nearly minimax estimation of edges," *Ann. Statist.*, vol. 27, no. 3, pp. 859–897, 1999.

[8] E. Le Pennec and S. Mallat, "Sparse geometric image representations with bandelets," *Image Processing, IEEE Transactions on*, vol. 14, no. 4, pp. 423–438, 2005.

[9] R. Shukla and M. Vetterli, "Geometrical image denoising using quadtree segmentation," in *Image Processing, IEEE International Conference on*, 2004, pp. 1213–1216 Vol.2.

[10] D. Scharstein and C. Pal, "Learning Conditional Random Fields for Stereo," in *Computer Vision and Pattern Recognition, IEEE Conference on*, 2007, pp. 1–8.

[11] J. Daniel, W. Gragg, L. Kaufman, and G. Stewart, "Reorthogonalization and stable algorithms for updating the Gram-Schmidt QR factorization," *Mathematics of Computation*, vol. 30, no. 136, pp. 772–795, 1976.

[12] R. Coifman and D. Donoho, "Translation-invariant de-noising," Department of Statistics, Tech. Rep., 1995.

[13] I. Daubechies, M. Defrise, and C. De Mol, "An iterative thresholding algorithm for linear inverse problems with a sparsity constraint," *Comm. Pure Appl. Math.*, vol. 57, no. 11, pp. 1413–1457, Jan. 2004.

[14] M. Elad, B. Matalon, J. Shtok, and M. Zibulevsky, "A wide-angle view at iterated shrinkage algorithms," *Proc. SPIE 6701, Wavelets XII*, p. 670102, Sep. 2007.

[15] P. Combettes and V. Wajs, "Signal recovery by proximal forward-backward splitting," *Multiscale Modeling & Simulation*, vol. 4, no. 4, pp. 1168–1200, 2006.

[16] T. Blumensath and M. Davies, "Iterative thresholding for sparse approximations," *Journal of Fourier Analysis and Applications*, vol. 14, no. 5-6, pp. 629–654–654, 2008.

[17] M. Figueiredo, R. Nowak, and S. Wright, "Gradient projection for sparse reconstruction: Application to compressed sensing and other inverse problems," *Selected Topics in Signal Processing, IEEE Journal of*, vol. 1, no. 4, pp. 586–597, 2007.

[18] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM Journal on Imaging Sciences*, vol. 2, no. 1, pp. 183–202, 2009.

[19] G. Peyré, "A review of adaptive image representations," *Selected Topics in Signal Processing, IEEE Journal of*, vol. 5, no. 5, pp. 896–911, 2011.

[20] A. Buades, B. Coll, and J. Morel, "A non-local algorithm for image denoising," in *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 2005, pp. 60–65 vol. 2.
[21] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, "Image denoising by sparse 3-D transform-domain collaborative filtering," *Image Processing, IEEE Transactions on*, vol. 16, no. 8, pp. 2080–2095, 2007.
[22] ——, "BM3D image denoising with shape-adaptive principal component analysis," *Proc. Workshop on Signal Processing with Adaptive Sparse Structured Representations*, 2009.
[23] V. Katkovnik, A. Foi, K. Egiazarian, and J. Astola, "From local kernel to nonlocal multiple-model image denoising," *International journal of computer vision*, vol. 86, no. 1, pp. 1–32, 2010.
[24] M. Aharon, M. Elad, and A. Bruckstein, "K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation," *Signal Processing, IEEE Transactions on*, vol. 54, no. 11, pp. 4311–4322, 2006.
[25] J. L. Starck, D. Donoho, and E. Candes, "Very high quality image restoration by combining wavelets and curvelets," pp. 9–19, 2001.
[26] H. Takeda, S. Farsiu, and P. Milanfar, "Kernel regression for image processing and reconstruction," *Image Processing, IEEE Transactions on*, vol. 16, no. 2, pp. 349–366, 2007.
[27] X. Li, "Patch-based image interpolation: algorithms and applications," *International Workshop on Local and Non-Local Approximation in Image Processing*, 2008.
[28] O. Guleryuz, "Weighted averaging for denoising with overcomplete dictionaries," *Image Processing, IEEE Transactions on*, vol. 16, no. 12, pp. 3020–3034, 2007.
[29] P. Chatterjee and P. Milanfar, "Patch-based near-optimal image denoising," *Image Processing, IEEE Transactions on*, vol. 21, no. 4, pp. 1635–1649, 2012.
[30] M. Elad and M. Aharon, "Image denoising via sparse and redundant representations over learned dictionaries," *Image Processing, IEEE Transactions on*, vol. 15, no. 12, pp. 3736–3745, 2006.
[31] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *Image Processing, IEEE Transactions on*, vol. 13, no. 4, pp. 600–612, 2004.
[32] G. Golub and C. Van Loan, *Matrix Computations*, 3rd ed. The Johns Hopkins University Press, 1996.
[33] Å. Björck, *Numerical Methods For Least Squares Problems*. SIAM, 1996.

Adam Scholefield is currently a post-doctoral researcher in the Communications and Signal Processing Group at Imperial College London, UK. He received, from the same university, his MEng. degree in Electrical and Electronic Engineering, in 2007; and, after a brief interruption of studies to represent Team GB in the 2012 Olympics, his Ph.D. degree in 2013. His research interests include image restoration, enhancement and compression; computer vision; and sampling theory, particularly its application to image super-resolution.

Pier Luigi Dragotti is currently a Reader (Associate Professor) with the Electrical and Electronic Engineering Department at Imperial College London, U.K. He received the Laurea degree (summa cum laude) in electrical and electronic engineering from the University of Naples Federico II, Naples, Italy, in 1997; the masters degree in communications systems from the Swiss Federal Institute of Technology of Lausanne (EPFL), Lausanne, Switzerland, in 1998, and the Ph.D. degree from EPFL in April 2002. He has held several visiting positions at different universities and research centers including: Visiting Student with Stanford University, CA, USA, in 1996, visiting Researcher with the Mathematics of Communications Department, Bell Labs, Lucent Technologies, Murray Hill, NJ, USA, in 2000, Visiting Professor with the Politecnico of Milan, Italy, in 2010, and Visiting Scientist with the Massachusetts Institute of Technology (MIT), Cambridge, MA, USA, in 2011. He was Technical Co-Chair for the European Signal Processing Conference in 2012, Associate Editor of the IEEE TRANSACTIONS ON IMAGE PROCESSING from 2006 to 2009 and an Elected Member of the IEEE Image, Video and Multidimensional Signal Processing Technical Committee. He is currently a member of the IEEE Signal Processing Theory and Methods Technical Committee and a recipient of an ERC Starting investigator Award for the project RecoSamp. His research interests include sampling theory, wavelet theory and its applications, image processing and image super-resolution, image-based rendering.