A Spline-Wavelet Image Decomposition for a Difference Engine

P. C. Marais[1], E. H. Blake
Department of Computer Science,
University of Cape Town,
Rondebosch 7700, RSA

A. A. M. Kuijk
CWI, Department of Interactive Systems,
Kruislaan 413, 1098 SJ Amsterdam,
The Netherlands.

Using the concept of a Cardinal-Spline Multi-Resolution Analysis, we establish a means of generating a smoothed approximation of an input image. This approximation can be decoded and displayed in real-time at the video frame rate by the Difference Engine: CWI's systolic array display processor. This forward difference engine can rapidly compute the value of such a compact image representation across a span in constant time. Some results are given which confirm the suitability of the spline-wavelet transform as a means of producing a compact image code.

1. Introduction

The rapid growth of multi-media applications and the resultant need to process real-time video and audio streams have given renewed impetus to the search for high compression systems. These systems should (at least) satisfy the following criteria:

- they should achieve maximal, possibly lossy, compression whilst maintaining as high a degree of apparent fidelity as possible for a human viewer,
- both the encoding and decoding process should be sufficiently rapid to permit real-time processing; in particular, decompression should be possible on the fly without excessive cost.

[1]Supported by the South African Foundation for Research Development.

The nature of the specific application's requirements determine which of the above criteria may be weakened so as to achieve a workable solution.

For the compression of still images, we may accept a comparatively lengthy compression or decompression delay if the results are of sufficiently high quality. The *JPEG* still compression standard remains the method of choice for most hardware implementations. JPEG achieves compression by applying a *Discrete Cosine Transform* to the image (which is segmented into 8x8 blocks to improve performance).

Fractal based methods, which claim to achieve much higher compression ratios than JPEG, have emerged in commercially available products. Unfortunately, the artefacts introduced at high compression ratios cannot be objectively quantified: the blocking of JPEG may be considered preferable to the uniform blotches produced by fractal compression at high ratios. The issue of speed also intrudes: JPEG compresses significantly faster than fractal methods; however, the converse holds for decompression times. Fractal methods also possess an inherent scale-independence which allows them to re-size images with a minimal amount of computation and no obvious pixelation.

Another group of methods that are gaining prominence are those based on the *Wavelet Transform* (WT). This transform comes in a variety of flavours, each with their associated advantages and problems (See Table 1). However, they all possess the ability to encode texture regions efficiently — precisely what one requires when encoding "natural" scenes [11]. NACKEN [12] for a good introduction and some encouraging results. From a computational point of view, the WT is superior to the DCT [14]. In addition, even when the transform is blocked, it is able to withstand a much higher level of compression before blocking effects become apparent [13].

An issue which is seldom addressed, is the level of efficiency one can achieve when actually displaying the decompressed image. It is taken for granted that each pixel value will have to be computed by the controlling program. However, this is not necessarily the case — as illustrated by CWI's Difference Engine. In this case the display processor itself can set multiple pixels using only a small set of parameters, provided these pixel values are constrained to lie on the graph of a polynomial of some arbitrary degree. Hence, if we can describe our image as a set of polynomial primitives, we can display our image efficiently, without the need to explicitly compute the intensity value of each pixel [1]. This paper investigates the feasibility of using such an approach for the reconstruction of compressed images.

Before we can proceed, however, we must introduce the necessary mathematical tools. We have decided to do this in a tutorial fashion, over the next three sections, so that readers who are unfamiliar with the subject matter can gain insight into the theory underpinning our approach.

The next two sections contain a brief introduction to wavelet theory and multi-resolution analysis, with an emphasis on 2-D images. Section 4 discusses some of the benefits associated with the Wavelet Transform, i.e., ease of computation and compression potential. The final part of the tutorial, Section 5, introduces the cardinal spline formalism needed to understand Chui's spline wavelets. These wavelets form an essential part of our analysis.

Section 6 discusses the implementation of these ideas on the Difference Engine and the suitability of this device for the display of multi-resolution images. The results of our investigations are presented and discussed in Section 7. Concluding remarks and an overview of further work is given in Section 8.

1.1. Mathematical notation and preliminaries

The signals we deal with whether 1-D or 2-D are assumed to be *well-behaved*, that is, they are elements of the space $L^2(\mathbb{R})$. This vector space may be defined (barring a few technicalities) as the space of all functions which satisfy the criterion

$$\int |f(x)|^2\, dx < \infty$$

with a double integration substituted if our signal is 2-D. Outline characters are used as follows: $\mathbb{C}$ is the set of complex numbers, $\mathbb{R}$ the set of reals and $\mathbb{Z}$ the set of integers. Continuous functions (over the reals) are denoted as elements of $C(\mathbb{R})$. The notation $C^m(\mathbb{R})$ refers to the space of m-times continuously differentiable functions. The *norm* of u is denoted $\| u \|$ with a subscript indicating the space w.r.t which the norm is taken. The *inner product* of u, v is represented as follows: $\langle u, v\rangle$.

A circumflex is used to denote the Fourier Transform of a function f:

$$\hat{f}(x) = \int f(t)e^{-ixt}dt.$$

We use the symbols $\dot{+}$ and $\oplus$ to denote a direct sum. The former is a generic direct sum, while the latter denotes an orthogonal direct sum.

An asterisk, "*", denotes convolution; the nature of the convolution (whether discrete or continuous) will be clarified when the operator is used.

Sequences are generally indicated as follows: $\{a_k\}_{k\in\mathbb{Z}}$. The space $\ell^2(\mathbb{R})$ contains all the sequences which satisfy the criterion

$$\| \{c_k\} \|_{\ell^2}^2 = \sum_k |c_k|^2 < \infty.$$

Of course, our sequence index will usually have a finite range. To simplify formulae, we will often ignore the range subscript, it usually being the case that our index ranges from $-\infty$ to $+\infty$. Similarly, if there are no range limits on an integral, one may assume it is taken over the entire domain.

2. The integral wavelet transform (IWT)

The IWT is defined in terms of a special kernel function ψ, known as a wavelet, which is an element of the space L^2. Functions, f, in this space must satisfy $\int |f(x)|^2 dx < \infty$. The wavelet is subject to the following additional constraints (in 1-D):

1. $\int \psi(x)\, dx = 0$

2. both ψ and $\hat{\psi}$ (its Fourier Transform) must be *window functions*. A function $w(x) \in L^2$ is called a window function if $x\, w(x) \in L^2$. This window function has a well defined centre, t^* and radius, Δ_w. This implies that the function w is such that it is localized in both the time and frequency domains (within the limits imposed by the Uncertainty Principle, (see [3, 6]).

More intuitively: the wavelet must decay rapidly and also exhibit some degree of oscillation — hence the name. With our wavelet constrained in this manner, we are able to define the IWT, $(W_\psi f)(b,a)$, of an L^2 function f:

$$(W_\psi f)(b,a) = |a|^{-\frac{1}{2}} \int f(t) \overline{\psi\left(\frac{t-b}{a}\right)} dt. \tag{1}$$

where $a,\ b \in \mathbb{R}$ and $f \in L^2$ and the overbar denotes complex conjugation.

Because ψ is essentially localisable in both time and frequency (scale), the IWT is also localised and gives us information in both domains, within the bounds of the Uncertainty Principle. If we permit our variables a, b to be continuous, the *inverse transform* involves computing a $(n+1)$ dimensional integral (if the function has n variables). To ensure computational efficiency, we discretize both the scale, a and the time-localization, b, in the following manner: $a = 2^{-j},\ \ b = k2^{-j},\ \ k, j \in \mathbb{Z}$. If we then define $\psi_{j,k}(x) \equiv 2^{j/2}\psi(2^j x - k), j, k \in \mathbb{Z}$, we obtain

$$(W_\psi f)\left(\frac{k}{2^j}, \frac{1}{2^j}\right) = \langle f, \psi_{j,k} \rangle = d_k^j, \tag{2}$$

where we have used inner product notation for compactness[2]. In order that we may recover our original function from this sampling, we require that $\{\psi_{j,k}\}_{j,k\in\mathbb{Z}}$ form a *Riesz basis* [3, 6]. This is a less restrictive requirement than orthogonality of the $\psi_{j,k}$, and permits us to construct wavelets which are not orthogonal. If ψ constitutes a Riesz basis, then there is a unique Riesz basis $\{\psi^{j,k}\}$ which is *dual* to $\{\psi_{j,k}\}$ i.e.

$$\langle \psi_{j,k}, \psi^{l,p} \rangle = \delta_{jl} \cdot \delta_{kp}, \quad j, k, l, p \in \mathbb{Z}. \tag{3}$$

Every $f \in L^2$ then has the unique series expansion:

$$f(x) = \sum_{j,k} \langle f, \psi_{j,k} \rangle\, \psi^{j,k}(x). \tag{4}$$

If, in addition, there is a function $\tilde{\psi} \in L^2$ which generates the dual basis in the same fashion that ψ generates the Riesz basis $\psi_{j,k}$, then we may also expand $f(x)$ as follows:

$$f(x) = \sum_{j,k} \langle f, \psi^{j,k} \rangle\, \psi_{j,k}(x). \tag{5}$$

[2] The L^2 inner product of two functions f, g is given by $\int f(x)\overline{g(x)}\, dx$.

Equations (4) and (5) are inverse transform formulae. These formulae relate the transform coefficients to the original function. In what follows, we assume that such a function does indeed generate the dual basis. Property (3) is called the *bi-orthogonality* property and is satisfied by all wavelets. If a wavelet is *orthogonal* it satisfies

$$\langle \psi_{j,k}, \psi_{l,p} \rangle = \delta_{jl} \cdot \delta_{kp}, \quad j, k, l, p \in \mathbb{Z}. \tag{6}$$

That is, orthogonal wavelets are *self-dual*, having $\psi = \tilde{\psi}$. Thus, when one deals with orthogonal wavelets, the added complexity of having a dual present is avoided. A wavelet which is orthogonal only between scales (frequencies) is called a *semi-orthogonal* wavelet — Chui's spline-wavelets are semi-orthogonal (cf. Table 1).

3. Multi-resolution analysis

The concept of a Multi-Resolution Analysis (MRA) is already familiar to those who have dealt with pyramidal image decompositions; it serves to formalize such a decomposition. Firstly, one must define the term "resolution". The intuitive interpretation, viz., that it serves to quantify the amount of permissable variation in a region, is formalized. Hence, a high resolution image has a large amount of detail in a region, whereas a low resolution image is much smoother over this same region. One may further quantify this concept with a statement such as: "a kth resolution image contains $k \times k$ samples per unit square". The idea here is that we can capture more detail if we are able to sample at a higher rate.

To develop the theory of such an analysis, we first consider the case of one dimensional signals.

Our signal, $f(x)$, must be an elements of the space $L^2(\mathbb{R})$, that is, it must contain finite energy. We seek a decomposition of this signal which will reveal its structure on different 'resolution' levels. Such an analysis can provide invaluable information about the relative importance of variations in the signal.

Each of these *multi-resolution approximations* resides in a space which contains all possible approximations at that resolution of every $L^2(\mathbb{R})$ function. These spaces are denoted V_j; the parameter j indicates the resolution level: the "resolution" of the jth level is given by $r = 2^j$. Thus, level 0 has $r = 1$. By convention, this is the input level.

Just as the wavelet spaces[3] W_j are spanned by the scaled translates of a single kernel function, ψ, we seek a single function, ϕ, the so-called *scaling function*, which will span the spaces V_j in the same way. If this is the case, then we may define a Multi-Resolution Analysis of $L^2(\mathbb{R})$. Since we desire that this analysis be complete, the MRA must encode the detail that is sacrificed when we go from a higher to a lower resolution. This detail is stored in the complementary *wavelet* spaces, W_j. We have the following relationship for any resolution level j

[3] $W_j \equiv \mathrm{clos}_{\mathrm{L}^2}\mathrm{span}\{\psi_{\mathrm{jk}} : \mathrm{k} \in \mathbb{Z}\}$; the operation of CLOSure essentially adds all the limit points to a space, thus 'closing' it up.

$$V_{j+1} = V_j \dot{+} W_j \tag{7}$$

This states that the higher resolution approximation may be resynthesized from the next lower approximation by adding the detail that we sacrificed to achieve that lower approximation. One can deduce the following properties:

1. $\cdots \subset V_{-1} \subset V_0 \subset V_1 \subset \cdots$;
2. $\mathrm{clos}_{\mathrm{L}^2}\left(\bigcup_{\mathrm{j}} \mathrm{V}_{\mathrm{j}}\right) = \mathrm{L}^2(\mathbb{R})$;
3. $\bigcap_j V_j = \{0\}$;
4. $V_{j+1} = V_j \dot{+} W_j, \;\; j \in \mathbb{Z}$;
5. $f(x) \in V_j \iff f(2x) \in V_{j+1}, \;\; j \in \mathbb{Z}$.

For a more detailed discussion and alternative formulation of these properties, see [8].

3.1. The Wavelet Transform and Multi-Resolution Analysis in 2-D

Since we wish to apply these techniques to images, we have to extend the previous results to 2-D. A common method of constructing the 2-D scheme, and simultaneously generating a MRA of $L^2(\mathbb{R}^2)$, is to define the space V_j as the tensor product of the space V_j with itself [6]. Then V_j induces a MRA of $L^2(\mathbb{R}^2)$: $V_j \subset V_{j+1}$ with the properties we discussed before and a scaling function $\Phi_{j;m,n}(x,y) = \phi(2^j x - m)\phi(2^j y - n), \;\; m,n \in \mathbb{Z}$.
Defining W_j to be the orthogonal[4] complement of V_j in V_{j+1} then gives us:

$$\begin{aligned} V_{j+1} &= V_{j+1} \otimes V_{j+1} \\ &= (V_j \oplus W_j) \otimes (V_j \oplus W_j) \\ &= (V_j \otimes V_j) \oplus [(W_j \otimes V_j) \oplus (V_j \otimes W_j) \oplus (W_j \otimes W_j)] \\ &= V_j \oplus W_j . \end{aligned} \tag{8}$$

So the complementary subspace W_j consists of three pieces, with Riesz Bases,

$$\psi_{j,m}(x)\phi_{j,n}(y), \quad \text{for} \quad (W_j \otimes V_j); \tag{9}$$

$$\phi_{j,m}(x)\psi_{j,n}(y), \quad \text{for} \quad (V_j \otimes W_j); \tag{10}$$

$$\psi_{j,m}(x)\psi_{j,n}(y), \quad \text{for} \quad (W_j \otimes W_j) \tag{11}$$

These three *detail* spaces contain the detail lost between two consecutive resolution approximations. In fact, each space contains the sharp variation (high frequency) information of the previous approximation in a *particular direction* Equation (9) gives the basis for the detail space which detects (represents) sharp variations in our function which are orientated in the x-direction, i.e. vertical edges. Similarly, the basis given by Equation (10) will represent edges

[4] We use $\oplus$ here since Chui's cardinal spline MRA induces such an orthogonal decomposition; one would use $\dot{+}$ for a more general setting.

in the horizontal direction. Equation (11) is the basis for the detail space which detects diagonal edges. We can now define three wavelets,

$$\Psi^{[1]}(x,y) = \phi(x)\psi(y) \tag{12}$$
$$\Psi^{[2]}(x,y) = \psi(x)\phi(y) \tag{13}$$
$$\Psi^{[3]}(x,y) = \psi(x)\psi(y). \tag{14}$$

Then $\left\{\Psi^{[i]}_{j;m,n};\ i=1,2,3\ m,n\in\mathbb{Z}\right\}$ is a Riesz basis for W_j; when we allow the scale parameter j to vary over all integers this basis is then a basis for $L^2(\mathbb{R}^2)$. As in the 1-D case, we can also find a dual, $\tilde{\Psi}^{[i]}_{j;m,n}$ which satisfies the bi-orthogonality relationship

$$\langle \Psi^{[m]}_{k;i,p}, \tilde{\Psi}^{[n]}_{l;j,q}\rangle = \delta_{mn}\delta_{kl}\delta_{ij}\delta_{pq}. \tag{15}$$

One may decompose any "well-behaved" ($L^2(\mathbb{R}^2)$) signal (image) in this manner. However, before we can proceed with our decomposition, we must ensure that we have a "valid" image at our zeroth (the input or highest) resolution level. By valid we mean that the image must be expressible on the basis, $\Phi_{0;lm}(x,y)$, which generates the approximation space of zeroth level functions, that is, V_0. For our image to be an element of the zeroth resolution space, it must satisfy the following requirement (i.e., be expressible on the zeroth level basis):

$$I^0(x,y) = \sum_{i,j} c^0_{ij}\Phi_{0;ij}(x,y) = \sum_{i,j} c^0_{ij}\Phi(x-i,y-j), \tag{16}$$

where the coefficients $\{c^0_{ij}\}$ are our zeroth level *approximation coefficients*. The means of generating these initial coefficients will be dealt with presently (Section). Assuming, for the moment, that such a relationship does hold, how do we generate subsequent lower resolution approximations? These approximations must be expressible on the appropriately scaled bases, where the scaling reflects the resolution concerned:

$$I^k(x,y) = \sum_{i,j} c^k_{ij}\Phi_{k;ij}(x,y) = \sum_{i,j} c^k_{ij}\Phi(2^k x-i, 2^k y-j),\ k=-1,-2,\ldots \tag{17}$$

Thus far, we have ignored the sequence of *detail* images which makes this representation complete. Just like the approximation images, these detail images are elements of particular spaces and must thus be expressible in terms of a particular basis:

$$g^k(x,y) = \sum_{ij}\sum_{p=1}^{3} {d_p}^k_{ij}\Psi^{[p]}(2^k x-i, 2^k y-j) \tag{18}$$

The coefficients $\{{d_p}^k_{ij}\}$ are called *detail coefficients.* The functions $\Psi^{[p]}(x,y)$ are 2-D *wavelets.* Wavelets are particularly well suited to encoding detail; hence the appearance of the wavelet in the basis of the detail space is not really surprising. We may formalize the relationship between the detail and approximation images as follows (recall the corresponding 1-D relationship):

FIGURE 1. The first four approximation images in our quadratic Multi-Resolution structure. The resolution decreases clockwise from top left.

$$I^{k+1}(x,y) = I^k(x,y) + g^k(x,y). \tag{19}$$

This states that an image at the $(k+1)$th resolution level is obtained by adding the lower resolution kth level image to the kth resolution detail image, which contains the information lost when we go from level $k+1$ to level k. Using this formula, we may express our (valid) input image as follows:

$$I^0(x,y) = g^{-1}(x,y) + \cdots + g^{-M}(x,y) + I^{-M}(x,y). \tag{20}$$

That is, we may decompose our image into a sequence of successively lower resolution images; this decomposition is, in turn, exactly equivalent to a sequence of detail images plus a low resolution approximation image. To extract a particular approximation image, we merely add back the relevant number of detail images to our lowest resolution approximation.

Figure 2 shows the third level detail images for our test image. Assuming we have determined our zeroth level approximation coefficients from our input image, the remaining approximation and detail coefficients required by our decomposition may be found by costly inner product calculations. However, MALLAT [11] derived the following efficient algorithms for producing these coefficients: The *decomposition algorithm* (which produces the next lower resolution level's coefficients):

$$
\begin{aligned}
c_{kl}^{j-1} &= \sum_m \sum_n a_{m-2k} a_{n-2l} c_{mn}^j \\
d_{1\,kl}^{\;j-1} &= \sum_m \sum_n a_{m-2k} b_{n-2l} c_{mn}^j \\
d_{2\,kl}^{\;j-1} &= \sum_m \sum_n b_{m-2k} a_{n-2l} c_{mn}^j \\
d_{3\,kl}^{\;j-1} &= \sum_m \sum_n b_{m-2k} b_{n-2l} c_{mn}^j.
\end{aligned}
\tag{21}
$$

The *reconstruction algorithm* (which provides the next higher resolution level's coefficients):

$$
\begin{aligned}
c_{km}^j = \quad & \textstyle\sum_l \sum_p p_{k-2l} p_{m-2p} c_{lp}^{j-1} + \\
& \textstyle\sum_l \sum_p p_{k-2l} q_{m-2p} d_{1\,lp}^{\;j-1} + \\
& \textstyle\sum_l \sum_p q_{k-2l} p_{m-2p} d_{2\,lp}^{\;j-1} + \\
& \textstyle\sum_l \sum_p q_{k-2l} q_{m-2p} d_{3\,lp}^{\;j-1}.
\end{aligned}
\tag{22}
$$

The $\{a_k\}$, $\{b_k\}$ are called *decomposition* sequences, while the $\{p_k\}$, $\{q_k\}$ are called *reconstruction sequences.*

These summations can be interpreted as 2-D linear convolutions which are down-sampled (i.e., we keep only even index output values) in the case of the decomposition algorithm and up-sampled (i.e., the coefficient sequences have zeros inserted between their entries) in the case of the reconstruction algorithm.

Since these 2-D convolutions are *separable* (i.e., expressible as the product of two independent 1-D sequences), they can be implemented much more efficiently if one does not employ direct brute-force summation. We see that if we are given the input approximation coefficients, $\{c_{ij}^0\}$ we can decompose and reconstruct as we wish; the only time we must explicitly concern ourselves with the scaling function, $\Phi(x, y)$, is when we wish to output our pth level approximation of the input image. At this point we are forced to compute discrete samples of our pth resolution image, utilising Equation (17). If one desires to access the detail images then a similar calculation must be performed using the detail coefficients (but this is not necessary if one just wishes to quantize or threshold the detail coefficients).

4. The Wavelet Transform and Compressibility

The wavelet transform (cf. Equation (1)), provides a "sparse" mapping from the spatial domain to some transform domain. This means that many of the transform coefficients are close to zero and can be ignored. One may thus achieve a considerable reduction in data by performing such a mapping and

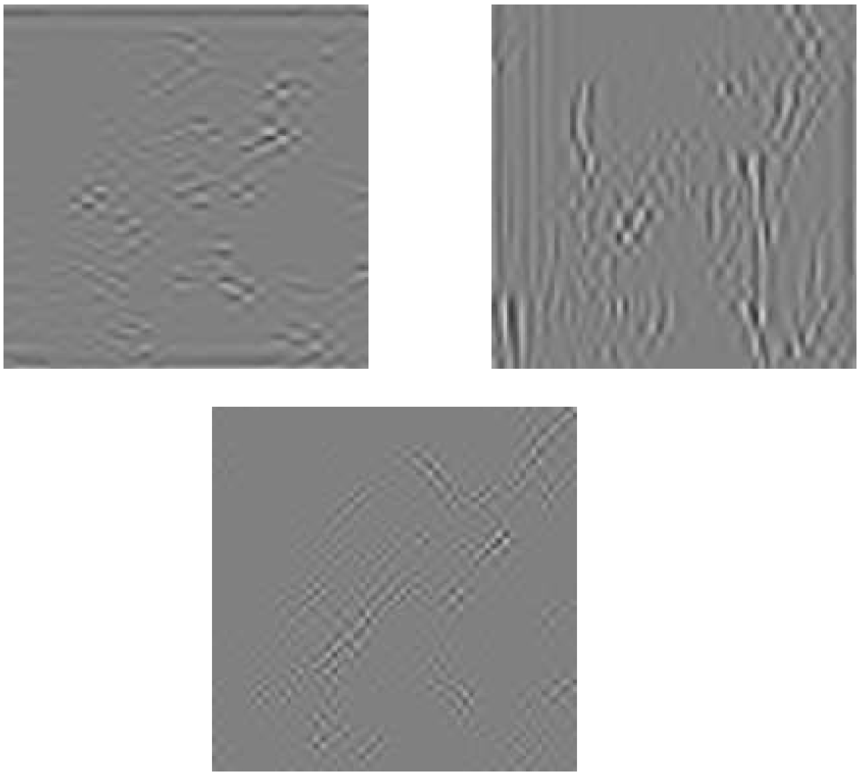

FIGURE 2. Third level detail images. These images illustrate the directional sensitivity of the Wavelet Transform. It decomposes the detail lost between consecutive levels into images which contain the detail information in the horizontal (top left), vertical (top right) and diagonal (bottom) directions. The bases underlying these images are (respectively) the wavelets $\Psi^{[1]}(x,y)$, $\Psi^{[2]}(x,y)$ and $\Psi^{[3]}(x,y)$ — cf. Equations (12, 14).

Properties	Wavelet Classes		
	Orthonormal	Semi-orthog (Chui)	Bi-orthog
Dual?	self-dual	yes	yes
Compact support?	wavelet	only wavelet	wavelet and dual
Symmetry?	no	yes	yes
Sequences?	finite	truncated	finite

TABLE 1. Some comparisons between the three major classes of wavelet. Only orthogonal wavelets do not possess a Dual Wavelet (see Section). If the wavelet or dual has compact support, we can achieve perfect reconstruction; otherwise we must truncate when we implement. Symmetry is important to reduce distortion when we reconstruct.

eliminating these small coefficeints, although naturally at some cost: it is no longer possible to achieve perfect reconstruction and there is the added overhead of performing the transform computations.

As regards the latter issue: the WT can be recast as a series of convolutions, which in turn can be very rapidly calculated using FFT based hardware. The degradation of the image with increased compression is not easily quantified, although it is often measured by means of signal-to-noise ratios (SNR). However, the WT is able to retain structure at very high compression ratios [7], unlike many of its rivals; this ability coupled with the existence of fast algorithms, make the WT an excellent choice for image coding.

The wavelet transform of an image yields the set of detail coefficients over all resolutions, j. However, since we only consider a finite range of resolutions, the transform must be restricted to reflect this. The approximation coefficients encode the information contained at the lower resolutions we do not wish to consider and are thus part of the restricted transform. Also, since we do not deal with resolutions higher than 1 (level 0), we do not require the detail (wavelet) coefficients for $j \geq 0$. Thus, the WT actually provides the set of coefficients

$$\{\{d_{p_{ij}}^{\ l}\}, \{c_{ij}^{-L}\}, \ \ l = -1, -2, \ldots - L; \ \ p = 1, 2, 3\} \tag{23}$$

The transform is given by the decomposition algorithm of the previous section (Equations (21), (22)). The *inverse transform* (the reconstruction algorithm) then recombines these coefficients to arrive at the input approximation coefficients (which then provide us with the input image).

The detail coefficients provide a very compact encoding of texture in the image; during quantization, many of these coefficients will be mapped to zero. The approximation coefficients will be highly correlated, reflecting the smooth nature of the low resolution approximation. Since the dynamic range of the coefficient values will be small, we do not need many bits to represent them, that is, they may be *quantized* fairly coarsely. In addition, the supports[5] of

[5] The support of a sequence is the set of indices which have non-zero sequence values associated with them.

both these 2-D sequences shrink with lower resolution, and so we require fewer coefficients to represent lower resolution approximation and detail images.

5. The Cardinal Spline formalism

We have chosen to implement our Multi-Resolution (MR) decomposition in terms of splines, employing the formalism of Chui [3]. Our reasons for choosing this approach over the preferred orthogonal framework, in which one does not require a *dual wavelet*, was partly motivated by the architecture of the system on which we have implemented this decomposition. However, the spline approach has several other advantages which compensate for its lack of orthogonality — indeed, these properties are present precisely because orthogonality of the wavelet bases has been sacrificed. In particular, since splines are amenable to rapid and efficient computations, any scheme based upon such curves offers implementational advantages over the aforementioned orthogonal transforms. For example, simple closed-form expressions are available for many of the formulae we utilise; this is not the case with, for example, the compactly supported orthogonal wavelets of Daubechies [6], where an iterative procedure must be used to calculate the scaling function. In addition, it has been shown [3] that one must inevitably sacrifice the desirable property of (generalized) linear phase[6] if one desires both compact support and orthogonality. If the wavelet and scaling function have this property then one is assured that the reconstructed signal will be minimally distorted (this is important when one engages in intensive quantization and thresholding, which introduce distortions of their own).

In the spline formalism both the wavelet and scaling function are expressed in terms of a B-spline series. In fact, the (1-D) scaling function is precisely the mth order cardinal B-spline, denoted $N_m(x)$. This function is computed recursively as follows:

$$N_m(x) = (N_{m-1} * N_1)(x), \quad N_1(x) = \chi_{[0,1)}(x), \tag{24}$$

where $\chi_{[0,1)}$ is 1 on the interval $[0, 1)$ and zero outside this interval and $*$ is the (continuous) convolution operator. See Figure 3.

When $m = 3$, we have a *quadratic* cardinal spline with continuous first-order derivatives at the knot-points. A full characterization of the (1-D) approximation spaces V_j is given by

$$V_j = \{f \in C^{m-2} \cap L^2(\mathbb{R}) : \; f|_{(\frac{k}{2^j}, \frac{k+1}{2^j})} \in \pi_{m-1}, \;\; k \in \mathbb{Z}\}. \tag{25}$$

This states that functions which are both well behaved (in L^2) and satisfy the indicated continuity condition are elements of the jth resolution approximation space, provided that their restriction to the indicated interval shows that they are contained in π_{m-1} — the space of all polynomials of degree $\leq m-1$.

[6] One can view the wavelet and scaling functions as band-pass and low-pass filters, respectively. If one filters with a linear phase filter, distortions in the input signal are not unduly magnified.

When j is negative the intervals over which the function is required to have a uniform polynomial character become progressively larger. This explains the smoothed nature of low resolution approximations to the original function.

The spline wavelets introduced by Chui, $\psi_m(x)$, have compact support on the interval $[0, 2m-1]$. The support of the cardinal B-spline is $[0, m]$. In addition, if the wavelet has even order m, it is symmetric; otherwise it is antisymmetric (about $\frac{2m-1}{2}$). See Figure 3. The symmetry/antisymmetry of the wavelet is responsible for its distortion reduction property.

The reconstruction sequences $\{p_k\}$ and $\{q_k\}$ (cf. Equation (22)) are very short sequences; the former has $m+1$ terms and the latter $3m-1$. These sequences are given by

$$p_j^m = 2^{-m+1}\begin{pmatrix} m \\ j \end{pmatrix}, \quad j = 0, \ldots, m; \tag{26}$$

$$q_j^m = \frac{(-1)^j}{2^{m-1}} \sum_{l=0}^{m} \begin{pmatrix} m \\ l \end{pmatrix} N_{2m}(j+1-l), \quad j = 0, \ldots, 3m-2. \tag{27}$$

Although these sequences appear complicated, efficient algorithms are given in [5, 3] for their calculation (see Table 6 in Appendix). We may use the following "two-scale" equation to compute the values of the wavelet, $\psi_m(x)$:

$$\psi_m(x) = \sum_{j=0}^{3m-2} q_j^m N_m(2x - j) \tag{28}$$

In [5], details are given concerning the derivation of the sequences $\{a_k\}$ and $\{b_k\}$ (Table 7 in Appendix gives the (corrected) sequence values we used). Although these sequences are not finite they have rapid exponential decay and can be truncated after about twenty terms with little obvious effect (see Figure 6). However, this truncation should not be done arbitrarily, but with respect to the sequences centre's of symmetry (if one wishes to preserve the distortionless property of the decomposition). The symmetry of these sequences is clear from the following relationships

$$a_{m-j}^m = a_j^m \tag{29}$$

$$b_{3m-2-j}^m = (-1)^m b_j^m, \quad j \in \mathbb{Z}. \tag{30}$$

One may use this symmetry to reduce storage and computational overheads.

The 2-D scaling function and wavelets are obtained from these 1-D versions by means of tensor products. The details of this extension are given in Section .

5.1. Calculation of $\{c_{ij}^0\}$*: the level 0 approximation coefficients*

The resolution ladder stretches off to infinity in both directions; however, we are only able to measure our image at a finite resolution, denoted I^0. This is our initial approximation of the continuous image data. The superscript zero indicates that we have chosen the resolution level $j = 0$ as our reference level. In this case, the cardinal spline which constitutes our scaling function has

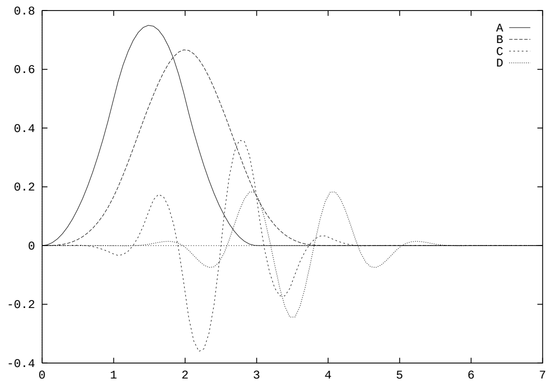

FIGURE 3. Cubic and Quadratic scaling functions and wavelets. 'A' is the quadratic scaling function (the 3rd order cardinal B-spline) and 'C' the corresponding wavelet. 'B' is the cubic scaling function and 'D' the cubic-spline wavelet. Observe that the cubic wavelet is symmetric, while the quadratic wavelet is antisymmetric.

knot points at the integers (we assume that the pixels lie on a two dimensional integer lattice); therefore, the image function expressed on this spline basis will consist of polynomial segments (patches) of degree $m-1$ translated and summed over the intervals between the knots (pixels), and will thus yield a new polynomial of degree $\leq m-1$ over each knot interval. Naturally, since our image is undefined between the discrete pixels, we will only sample our reference image at these integer knot points. Thus, from Equation (17), we have our zeroth level (continuous) approximation, with $N_m(x)$ defined by Equation (24)

$$I^0(x,y) = \sum_p \sum_k c^0_{pk} N_m(x-p) N_m(y-k). \tag{31}$$

In general, we wish to decompose our approximation from this reference level to some arbitrary lower resolution level, say j. Since j must be less than zero (lower resolution) we write

$$I^{-j}(x,y) = \sum_p \sum_k c^{-j}_{pk} N_m(2^{-j}x-p) N_m(2^{-j}y-k). \tag{32}$$

in keeping with our earlier definition, where $j = 1, 2, 3, \ldots$

As the formulae stand, they contain summations which range across $\mathbb{Z}$. However, since our image has finite spatial extent, the ranges of summation for both the detail and smoothing coefficients must be curtailed. Our input image, $I^0(i,j)$ is expressed on the zeroth level basis with the smoothing coefficients as weights. Inspection of the formula, coupled with the assumption of our image's finite extent, produce the necessary ranges of summation for c^0_{ij}, and hence reveal the number of these coefficients we are required to calculate. The limits on the decomposition and reconstruction algorithm are estimated by considering the maximum range of index values (given the finite range of the sequences $\{a_k\}$, $\{b_k\}$, $\{p_k\}$, $\{q_k\}$) which produce non-zero multiplications in the formulae. Of course, these ranges differ from level to level, since the convolutions are down- or up-sampled as required.

Unfortunately, assuming that our image has zero intensity outside some specific interval will introduce irritating boundary effects, particularly as one views lower resolution approximations. The method used to deal with such artefacts, is to extend the image by symmetry, thus ensuring a smooth transition across boundaries. However, extending our image generates additional non-zero approximation (and consequently, detail) coefficients, since these represent our image and hence mirror any increase in its extent. Fortunately, the number of additional coefficients that one need consider is small (< 10), since distant pixel values have progressively less influence the further away they are from the pixels on the periphery, and we have no wish to display pixels beyond our initial image boundary. An alternative strategy would be to allow the image to decay to zero beyond its support.

From the above it is clear that we need to calculate $\{c_{ij}^0\}$ before we are able to begin our decomposition. That is, we must obtain a representation of our image as a sequence of expansion coefficients on the basis given in Equation (32) — we wish to *project* our true image onto its zeroth level approximation. In order that we may accomplish this projection, it is necessary that our signal function be bounded and continuous. Neither of these restrictions is problematic for images; they are certainly bounded in the intensity values they may take on and, since we only sample discrete points, we can always assume that our image is continuously interpolated between these points.

We wish to determine the solution set $\{c_i^0\}, i \in \mathbb{Z}^2$, of Equation (31), where our variables x, y are constrained to be integers and the values $I^0(x,y)$ are our input intensity values. A true interpolation scheme, in which the interpolant passes though each input $(x,y,I^0(x,y))$ triple, would require the inversion of a large matrix, at considerable computational expense. *Quasi-interpolation* [4, 3] offers a cheaper alternative, since it only uses local data to determine the values of the c_{ij}^0. However, the interpolant no longer passes through each input point unless some very strong conditions are imposed (see below) or the computations are made sufficiently non-local. The scheme is based on a 2-D convolutional operator. This 2-D operator is applied to the input intensity values to produce the c_{ij}^0. See Equation (34). This operator has a sequence support that grows with the *order*, k, of the quasi-interpolation scheme. For quadratic and cubic cardinal splines, we have an operator of size $(2k+1) \times (2k+1)$.

The parameter k also determines the accuracy of the fit: as k grows larger, quasi-interpolation tends to true interpolation [3, pg. 105]. In addition, quasi-interpolation has the property that it will interpolate a polynomial (in s variables) of total degree $\leq m-1$ (that is, an element of π_{m-1}^s) perfectly, if $k > \frac{m-3}{2}$, [4, pg. 646]. For example, when $m = 3, s = 2$ (quadratic cardinal splines in 2 variables) [4], we may choose any $k > 0$ to achieve perfect reproduction of a second degree 2-D polynomial. However, this property is of little use to us, since our image can contain arbitrarily irregular data.

The method may be encapsulated (in our case) as follows [4]:

$$(Q_k I)(x,y) = \sum_{i,j} (\lambda_k I)(i,j) N_m(x + \frac{m}{2} - i) N_m(y + \frac{m}{2} - j), \tag{33}$$

where k is the order of the quasi-interpolation operator[7], Q_k. The λ coefficients are obtained as follows (by applying the convolutional operator to the input data set):

$$\{(\lambda_k I)(i)\} = (\delta - m + \cdots + (-1)^k \underbrace{m * \cdots * m}_{k \text{ times}}) * I^0, \quad i \in \mathbb{Z}^2, \tag{34}$$

where $*$ represents 2-D discrete convolution and $m = \{m_i\}$, $i \in \mathbb{Z}^2$ with

$$m_{ij} = \begin{cases} N_m(0 + \frac{m}{2}) N_m(0 + \frac{m}{2}) - 1 & \text{for } i,j = 0, \\ N_m(i + \frac{m}{2}) N_m(j + \frac{m}{2}) & \text{for } i,j \neq 0. \end{cases}$$

and $\delta = \{\delta_{i0}\}$, where $\delta_{i0} = 0$ if $i \neq 0$ and $\delta_{00} = 1$.

It can be seen that, as the order k grows, it becomes increasingly irksome to compute explicit representations for this; we have computed such explicit coefficients fore the cases $k = 1, 2$, with sequence supports of 3×3 and 5×5 (Appendix). Note that the cardinal B-splines have been centred, since the algorithm in [4] requires this (before this shift, they are symmetric with respect to $\frac{m}{2}$.) To reconcile Equation (33) with Equation (31) (and hence extract the initial smoothing) coefficients, we make the identification

$$\left\{c^0{}_{ij}\right\} = \{(\lambda_k I)(i,j)\}. \tag{35}$$

However, we must remember to introduce the appropriate shift ($\frac{3}{2}$ or 2) when we compute our approximation function Equation (32)[8].

6. Implementation

6.1. *The Difference Engine*

The Difference Engine is the final component in the rendering pipeline of a new display architecture produced at CWI. Its function is described more fully in [1], but essentially it generates the pixel stream which produces the image on the display. This processor may be described as a forward difference engine for arbitrary order polynomials — that is, given the appropriate initial differences, it will interpolate an arbitrary order polynomial (representing the intensity profile) across a span of pixels. The logic is implemented by a systolic array,

[7]We deal with boundary problems by extending the input image symmetrically about its edges before computing the quasi-interpolant.

[8]By making this identification, we are shifting our entire image by $\frac{m}{2}$ in both dimensions; thus we must remember to add this value to the x and y arguments of our jth level approximation.

allowing the (intensity) data to propagate along the scan-line in a time which depends on the order of the polynomial and not the length of the pixel span. Since the processor has an 11ns cycle time, and the systolic array elements need only perform adds as the data propagates, this leads to very rapid calculation times; indeed, the Difference Engine is able to produce pixel streams at the display refresh rate. Originally intended for the rapid production of Phong shading values along pixel spans, it was realised that the chip's design was such that it was ideally suited for the synthesis of images consisting of polynomial spline patches that is, those which have an appropriately smoothed intensity profile. Naturally, a means would have to be discovered of generating such an images. Inspired by the spline-wavelets of CHUI [3], such a connection was posited and subsequently verified (see later sections).

6.2. The Difference Engine and Multi-Resolution Approximations

In order to interpolate a span of pixels, one must first decide on the order of the polynomial to be employed, for this determines the number of initial calculations which must be performed on each span. For example, quadratic interpolation requires only the computation of first and second differences. Once these differences have been computed, the chip is able to interpolate a span of arbitrary length within the limits imposed by rounding errors [1]. Higher order polynomial interpolation achieves a better approximation to the original image, but this accuracy comes at the expense of additional difference calculations, longer instructions and a rapid decrease in the length of the spans which may be accurately interpolated.

We implement the algorithm as follows. For a particular resolution level j, the basis elements of our spline space are translations of the tensor product $N_m(2^{-j}x)N_m(2^{-j}y)$ which has support on $[0, 2^j m]^2$ and has knot-points at $2^j\mathbb{Z}^2$ on this support. If we restrict $I^{-j}(x, y)$ to the intervals $[2^j k, 2^j(k+1)]^2, k \in \mathbb{Z}$ we obtain a polynomial patch (of degree $\leq m-1$), uniquely describable in terms of a single set of coefficients[9] and hence suitable for our difference manipulations. Since the Difference Engine operates in one dimension, we fix the parameter y in our expression for $I^{-j}(x, y)$ and proceed to calculate the requisite number of differences by evaluating this expression at successive horizontal pixel locations. Once we have the differences, we compose the appropriate processor instruction and output this to the Difference Engine, which then proceeds to interpolate the span of length 2^j pixels[10].

7. RESULTS

These ideas were investigated on a simulator which emulates the action of the Difference Engine. The controlling program performs the wavelet decomposition/reconstruction (as well as several other functions) and generates the

[9] See the earlier characterization of V_j (the restriction of our images to the region between the knots points $2^j\mathbb{Z}^2$ has fixed polynomial character - remember: our 2-D V_j is just obtained by taking the tensor product of our 1-D V_j).

[10] The span is actually of length 2^j+1. However, the last pixel is set by the next instruction.

Difference Engine instruction stream, which is then piped to the simulator.

The theoretical analysis of the previous sections was used to produce a viable image encoding scheme. The primary purpose was to evaluate the suitability of the Difference Engine as a reconstruction engine.

7.1. Difference Engine performance

We may quantify the reduction in processing required when displaying a multi-resolution approximation image, in terms of *function evaluations gained per span.* That is, the number of intensity function evaluations along a span which we are no longer required to perform because of our interpolation scheme. We only need to evaluate Equation (32) when we compute our differences; the Difference Engine does the rest.

The maximum number of operation occurs when we wish to display our zeroth level approximation: in this case we are forced to transmit an instruction to set each pixel — this is our *baseline count.* If we proceed to level one, we have spans of length three (the last pixel being taken as the first pixel of the next span); we are thus able to compute the necessary difference information. However, it would be more economical to just set each pixel, since this means we no longer have to compute difference information. For the both the above cases, then, we need to transmit $N \times M$ instructions for a display of size $N \times M$.

If we employ the quadratic spline approach, we realise gains from the second resolution level downwards. The calculation of the (quadratic) differences involves the evaluation of our intensity function at three consecutive points on our span, via Equation (32). Since the spans overlap, and we are interpolating a span of 5 pixels, we gain one function evaluation per span. For level three we gain a reduction in computation and transmission costs equivalent to five function evaluations. On the jth resolution level we gain $2^j - 3$ function evaluations per span (when utilising quadratic interpolation). In the case of a cubic, we may quantify the number of function evaluations gained per span as $2^j - 4$. For an image of size $2^x \times 2^y$ pixels, utilising an mth order cardinal spline scheme, we require approximately $\frac{2^x}{m2^j}2^y = \frac{2^{x+y-j}}{m}$ Difference Engine instructions to produce a jth resolution approximation of the input image, since our spans overlap and each scan-line must be interpolated separately. We assume here that $j \leq x$ i.e. our spans are no wider than the image. This formula holds for any order of polynomial interpolation[11]. However, one must bear in mind that at least $n + 1$ pixels must be available to allow calculation of the initial differences in an nth order scheme. Clearly, as the resolution becomes coarser these operations become more economical, eventually permitting one to interpolate the entire scan-line with one instruction. The compression relative to the baseline case is given by $\frac{2^j}{m} : 1$; thus for $m = 3, j = 2$ (second level approximation based on quadratics) we achieve a $\frac{4}{3} : 1$ compression gain over the baseline case. We have implemented both quadratic and cubic schemes; the difference in quality is scarcely discernable (numerically the quadratic scheme

[11] Recall that an mth order scheme is based on polynomials of degree $m - 1$.

Quadratic Case	Mean	Standard Deviation	|Max Error|
$k = 1$	0.00	1.69	21
$k = 2$	0.01	0.90	10
Cubic Case			
$k = 1$	0.01	2.74	34
$k = 2$	0.00	1.79	21

TABLE 2. The error induced by quasi-interpolation of our test image. The quadratic scheme ensures both a lower projection error and a lower maximum error. The benefit of using a higher order quasi-interpolation is clear: even $k = 2$ provides a considerable gain over $k = 1$.

wins out because it, requires fewer difference computations and, as we shall see, has lower interpolation error and is able to reproduce good images even when the filters are severely truncated).

7.2. Reconstruction errors

The following sections deal with the three sources of error we have identified in our scheme: interpolation error, the error induced by sequence truncation and the error caused by neglecting small detail coefficients.

7.2.1. Interpolation error

The prime source of error in our approximation is a consequence of our employing a quasi-interpolation scheme (Equation (33)), and not interpolating the data precisely, (Figure 4). Also, since we are projecting our function into the space of cardinal splines (which are forced to obey certain smoothness constraints at their knot-points) we must expect a measure of smoothing. However, this is minimal at the input resolution level and can be eliminated entirely if one employs true interpolation. To quantify these results, we have the following estimate for an upper bound on the projection error (adapted from [3]):

$$\max_{\ell} |(Q_k f - J_m f)(\ell)| \leq (\max_{\ell} f(\ell) + \min_{\ell} f(\ell))\frac{1}{2}\beta_m^{k+1} \tag{36}$$

where the function f represents our (finitely supported) image values, m is either 3 or 4, depending on whether the scheme used is based on quadratics or cubics and $\beta_3 = \frac{1}{2}, \beta_4 = \frac{2}{3}$. The function $(J_m f)(l)$ is a true interpolant based on the appropriate spline. It is clear from this that the quadratic scheme produces a *better* approximation than the cubic scheme; this rather counter-intuitive result is borne out by Table 2.

Table 2 also illustrates the kind of errors which arise from such an approximation; in particular, the mean error is very acceptable even for such low order k's, although the maximum error can be large. Fortunately, the regions where such error would occur (i.e. sharp spikes) can be less accurately interpolated without noticeable degradation of the image. Smoothing is an integral part of picture capture, since any device has a finite spatial-frequency bandwidth

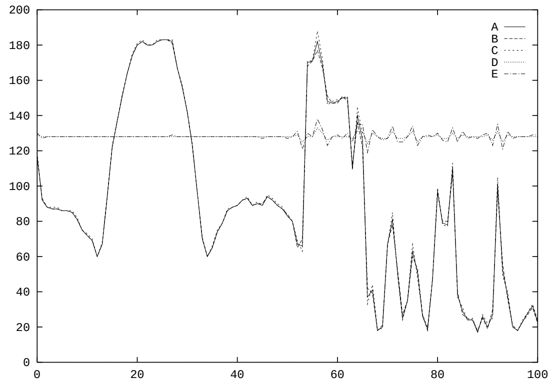

FIGURE 4. Quasi-interpolation Error Effects. Cross-section at scan-line 133 of the lenna image; the interpolation error is biased by 128. 'A' gives the quadratic quasi-interpolation ($k = 2$) of the scan-line, 'B' the cubic interpolation ($k = 2$). The graph 'C' is the input data for scan-line 133. Graph's 'D' and 'E' give the interpolation error for the quadratic and cubic cases, respectively. Observe that the interpolation error for the cubic scheme is greater than that of the quadratic scheme for the same k.

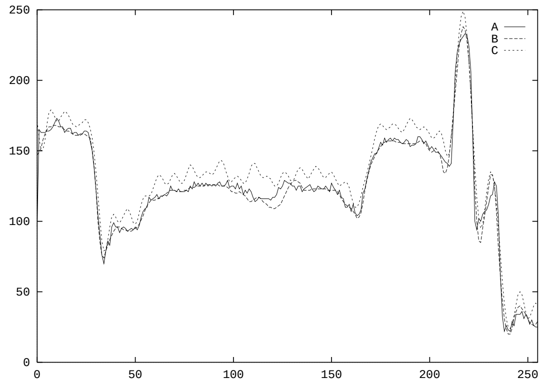

FIGURE 5. Failure of the cubic filters at low truncations. 'A' gives the data on scan-line 20, 'B' the 2nd resolution level cubic decomposition approximating the image and 'C' the reconstruction to level 2 after decomposing with the over truncated cubic decomposition sequences. When the cubic filters are not over truncated, they result in a reconstruction which is very similar to the quadratic case. Note: the reconstruction sequences are *never* truncated.

Quadratic Case	Mean	Standard Deviation	|Max Error|
$k = 1$	0.50	1.71	21
$k = 2$	0.50	0.93	11

TABLE 3. Effect of reconstruction after projecting with different order quasi-interpolation schemes. With $k = 2$ the reconstruction is, on average, within one grey-scale value of the input image.

	$\#a$	$\#b$	Mean	Std Dvtn	|Max Error|	$\mathrm{O}(\sum a)$	$\mathrm{O}(\sum b)$
Quad	40	36	0.14	0.91	10	10^{-4}	10^{-4}
Cubic	39	33	1.48	2.07	23	10^{-3}	10^{-2}
Quad	34	30	-0.2	0.91	10	10^{-4}	10^{-4}
Cubic	33	27	1.31	2.09	25	10^{-3}	10^{-2}
Quad	30	26	0.5	0.93	11	10^{-4}	10^{-4}
Cubic	29	23	-2.57	2.71	22	10^{-3}	10^{-2}
Quad	14	10	15.51	6.36	44	10^{-2}	10^{-4}
Cubic	13	7	-40.78	32.42	244	10^{-2}	10^{-1}

TABLE 4. Reconstruction error after truncating the decomposition sequences (order 2 quasi-interpolation). The left-most two columns indicate the number of a, b coefficients we maintain after truncation. The statistical data gives an indication of the effects of our truncation on the error image. The final two columns indicate the order of magnitude of the error to within which the sequences approach their filter conditions, Equations (37).

and thus performs a low-pass filtering on the original image; a little additional smoothing is more than acceptable when one considers the local nature of the quasi-interpolation operator.

7.2.2. Errors induced by sequence truncation
The length of the decomposition sequences has a profound effect on the processing required to calculate the detail and smoothing coefficients and on the accuracy of these coefficients. Longer sequences require more work but result in a more accurate image representation.

How then, does intensive truncation of the decomposition sequences $\{a_k\}$ and $\{b_k\}$ affect the quality of the image? We truncated the sequences simultaneously. Table 4 provides some data to quantify our experiments. It is clear that for low truncation limits the reconstruction is badly distorted; as the number of terms increases the error quickly falls to acceptable limits. There is, however, a very noticeable asymmetry in the performance of the quadratic and cubic schemes, which is manifest at low truncations. The cubic representation suffers noticeable high-frequency distortion when we truncate to below a critical threshold (23 terms in $\{b_k\}$). This noise is realised as a tartan-like pattern which distorts the image (see the cross-section scan Figure 5) and is a

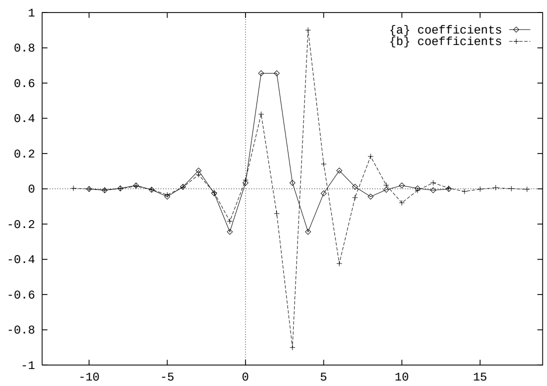

FIGURE 6. The decomposition sequences $\{a\}$ and $\{b\}$ for the quadratic case. Both these sequences have infinite extent but decay exponentially. $\{a\}$ is symmetric about 1.5 while $\{b\}$ is anti-symmetric about 3.5.

consequence of the filter's full-integer symmetry (that is, it is symmetric about a particular coefficient index in the sequence (index 5 for the cubic case). Both the filters $\{a\}$ and $\{b\}$ are required to satisfy the following conditions:

$$\sum_k \{a_k\} = 1, \quad \sum_k \{b_k\} = 0. \tag{37}$$

If these conditions are not met, then the filters are dysfunctional and the output signal is polluted by unwanted frequency components.

Referring to Table 4, we see that the cubic $\{b\}$ sequence is very sensitive to truncation when we take few terms. This same sensitivity is *not* present in the quadratic case, since the sequence $\{b\}$ is perfectly symmetric with respect to a half-integer point and hence tends to zero regardless of our truncation level (that is, its form is ...d,e,f,-f,-e,-d... about its centre of symmetry, whereas the cubic case is ...d,e,f,g,f,e,d... about its centre of symmetry (g) and is thus not guaranteed to sum near zero unless the coefficients surrounding the centre of symmetry are appropriately defined, which no longer happens below 23 terms for $\{b\}$). Note that, in all cases, we truncate so as to preserve the sequences' symmetry (which is responsible for the linear phase property that eliminates/reduces distortion). A comparative test of the impact of this formalism's linear phase aspect was not done, since we did not implement a non-linear-phase scheme. However, one can see from Figure 7 that even with very severe truncation of detail, the main structures persist and strong edges are essentially undistorted.

7.2.3. Errors induced by detail coefficient elimination

To determine the suitability of the spline-wavelet transform for compression

Threshold	%zeroed	Mean	max $|\text{Error}|$	Std Devn
0.1	86%	0.55	35	6.44
0.16	91%	0.67	55	10.36
0.2	93%	0.72	65	13.00
0.3	96%	0.85	100	19.81

TABLE 5. The effect of zeroing detail (wavelet) coefficients — quadratic case. The threshold determines the percentage of detail coefficients which are neglected in the reconstruction. The other three columns give statistical information about the nature of the reconstruction error.

FIGURE 7. Reconstruction after zeroing detail coefficients. Truncation thresholds 0.1, 0.3

purposes, we zeroed all the detail coefficients below a specified threshold and produced the data in Table 5; the corresponding reconstructed images are in Figure 7. From this data we can see that the Multi-Resolution structure can be used to encode an image efficiently; one merely decomposes until the support of the smoothing coefficients is acceptably small and then applies a suitably chosen limit which eradicates a large number of detail coefficients. The position of the coefficients can be encoded using some kind of run-length encoding while the magnitude of the coefficients has to be quantized (the data may be further reduced by an entropy coding). From the images one can see that as we zero more detail coefficients we begin to lose texture and eventually larger scale high-frequency data, such as edges. Examination of our first level approximation image reveals very little difference from the input image; hence we can zero *all* first level detail coefficients (cf. Figure 1). One can also see the effects of our assumption of finite image extent (the support of our input sequence is essentially the same as the unexpanded image support) in the slight low-frequency ripples which emanate from the image edges. Taking a larger input coefficient support will reduce these effects (which are not noticeable when we do not engage in intensive thresholding).

The above provides some indication of the Wavelet Transform's suitability for image compression. Of course, to obtain high quality reconstructions with maximal compression, we would have to threshold in a more intelligent way and/or utilise a scheme such as Vector Quantization. This is an area we are investigating further.

8. Conclusion

In this paper we have shown how one might exploit the architecture of CWI's Difference Engine to achieve more efficient output of an image, provided one is willing to accept some measure of "blurrings". Since such a scheme produces fewer processor instructions, we can produce images at a higher rate.

Another advantage of such a scheme is the ease with which one can achieve progressive transmission — we merely transmit the next tier of detail coefficients, which are then combined to produce our new approximation image. One could, conceivably, use this ability to rapidly scan through a video database in order to get a feel for the material it contained.

We performed some elementary tests which confirmed the choice of the semi-orthogonal wavelet transform as one which will enable us to achieve our dual goals of rapid compression and efficient display. To achieve higher compression, we must utilise a Vector Quantization scheme; preferably one which can exploit the multi-resolution structure of the WT, as was done in [10]. An effective quantization scheme can ensure high compression ratios while maintaining image quality, particularly when followed by an entropy coding scheme such as Huffman coding.

8.1. 2-D Area Interpolation

Our 2-D multi-resolution approximations are required to have a fixed polynomial character over squares with support $[2^j k, 2^j(k+1)]^2$. This coherence is not exploited in our decoding, since the Difference Engine is inherently one dimensional. This state of affairs could be rectified if *two-dimensional interpolation* were used. That is, instead of interpolating along spans only, we could also interpolate across *scan-lines*. Naturally, the final instruction stream would have to be a one dimensional pixel stream — we could thus maintain the Difference Engine and precede it by a "Y-processor" which would accept (square) area primitives, each supporting a spline patch, and then perform a difference interpolation scheme in the $y-$direction, outputting a scan-line's worth of Difference Engine instructions after each new scan-line. We would be required to produce *eight* differences per 2-D instruction. In addition, we would need corresponding instruction fields for the y starting position, initial intensity and the span length (the same for both dimensions). Thus, to interpolate a block of size $n \times n$, we would have to produce 12 pieces of information, compared with the $5n$ (5 fields per span over n scanlines) required for a straight Difference Engine interpolation. With larger block sizes, the gain would become more significant. The fact that we are now interpolating in 2-D would cause a reduction in the size of the squares we could accurately interpolate — in the region of 64x64 with 24 precision bits for quadratic interpolation. This is not really a limitation since spans of 64x64 pixels correspond to an *eighth* level approximation — something we would be unlikely to require.

8.2. Adaptive multi-resolution encoding

Another possibility, which might reduce blurring, is to use an adaptive synthesis procedure: rather than using a fixed level of approximation, we generate instructions to produce detail where necessary. We can perform such a reconstruction because our image is the sum of a sequence of detail images and a low-resolution approximation image — see Equation (20).

Such a scheme would produce Difference Engine instructions to reproduce i) the low level approximation image and ii) the important regions of the detail images. These important detail regions will correspond to large detail coefficients; hence, our level of thresholding would determine the number of detail instructions that are generated and, consequently, the total number of processor instructions. There are a number of issues that would have to be addressed before such a scheme could be successfully implemented.

This scheme will be most appropriate if our detail coefficients are clustered around major texture features, with sparse regions where these coefficients are zero or may be approximated by zero. From the support of the processed detail coefficients we can determine the important non-zero detail areas in our detail images and hence the spans across a scanline with which these regions intersect.

Calculation of the detail image, $g^k(l,m)$, values requires the evaluation of the functions $\Psi^{[p]}(i,j)$ which is significantly more expensive than evaluating $\Phi(i,j)$ (we have *three* wavelets). However, if the detail regions are sparse enough this overhead should be less telling. One could also attempt to accelerate these computations by means of LPTA (Linear Pascal Triangular Algorithms, [3, pages 189–194]).

From our point of view the central issue is the number of processor instructions that we can save when compared to the high-resolution baseline case, in which we must individually set each pixel. Unfortunately, this problem is highly dependent on the image — images with little texture will require few 'detail-filling' instructions, while those with a high level of important texture information will require many such instructions. The level of thresholding on our detail coefficients will directly control the number of these instructions. The automation of such thresholding is a non-trivial problem, since there is little agreement on the properties that an objective image fidelity metric should satisfy. Without an extensive analysis, there is little one can say aside from the fact that our gain over the baseline will be bounded below by $\frac{2^j}{m}:1$. Thus, for sufficiently large j, the level of detail we wish to reproduce will be the primary factor determining the number of instructions we require. Care would have to be taken, however, to ensure that we do not permit excessive detail-filling instructions to be generated: under no circumstances should we produce more instruction than the baseline count.

8.3. More efficient compression

Using Vector Quantization with a wavelet-based compression scheme can provide compression ratios of around 40:1 with very good reproduction [10]. The possibility exists to improve the compression potential of the wavelet coding markedly by utilising a so-called "second generation" scheme, which exploits features inherent in the human visual system. One approach is to extract and code the visually relevant edge information (which produces an extremely compact encoding) and then to code the residual error using wavelets. This approach, a modification of the one proposed by CARLSSON [2], forms the the basis of a compression scheme employed by FROMENT and MALLAT [7].

Such a coding should achieve better compression because the edge image we extract contains most of the high-frequency information — it is this information that gives us large WT coefficients. We are currently investigating a coding scheme which combines all the above elements.

9. ACKNOWLEDGMENT

We would like to thank the referees for their suggestions and comments.

REFERENCES

1. E. H. BLAKE AND A.A.M. KUIJK (1993). A difference engine for images with applications to wavelet decomposition. *Proceedings of the Second International Conference on Image Communications (IMAGE'COM)*, pp. 309–314.
2. S. CARLSSON (1988). Sketch based coding of grey level images. *Signal Processing* **15**, pp. 57–83.
3. C.K. CHUI (1992). *An Introduction to Wavelets: Wavelet Analysis and its Applications*, volume one. Academic Press, Boston.
4. C.K. CHUI and H. DIAMOND (1987). A natural formulation of quasi-interpolation by multi-variate splines. *Proceedings of the American Mathematical Society* **99** (4).
5. C.K. CHUI and J.Z. WANG (1990). Computational and algorithmic aspects of cardinal-spline wavelets. *Technical Report 235, TAMU, Centre for Approximation Theory* (CAT). Note: There are some transcription errors in the (quadratic) decomposition sequences given here.
6. I. DAUBECHIES (1992). Ten lectures on wavelets. *CBMS-NSF series in applied mathematics* **61**. SIAM.
7. J. FROMENT and S. MALLAT (1992). Second generation compact image coding with wavelets. In C.K. CHUI, editor, *Wavelets: A Tutorial in Theory and Applications*, volume two, pp. 655–678. Academic Press, Boston.
8. H.J.A.M. HEIJMANS (1992). Discrete wavelets and multiresolution analysis. In KOORNWINDER [9], pp. 49–79. This article originally appeared in CWI Quarterly, Vol. 5, No. 1.
9. T. H. KOORNWINDER, editor (1993). *Wavelets: An Elementary Treatment of Theory and Applications*, volume 1 of *Approximations and Decompositions.* World Scientific.

$m=3$	$k=2$	$k=1$
Q	-0.0117187	-1/64
I	-0.1699218	-3/32
#	1.665	23/16
x	0.00293	-
X	0.0092773	-
O	0.0002441	-

FIGURE 8. The entries for the convolution operator

10. P. MATHIEU, M. ANTONINI, M. BARLAUD and I. DAUBECHIES (1992). Image coding using wavelet transforms. *IEEE trans. on image processing* **1** (2), pp. 205–220.
11. S. MALLAT (1989). A theory of multiresolution signal decomposition: The wavelet representation. *IEEE Trans. on Patt. Ana. and Mach Intell.* **11**, pp. 674–693.
12. P. NACKEN (1992). Image compression using wavelets. In KOORNWINDER [9], pp. 81–91. This article originally appeared in CWI Quarterly, Vol. 5, No. 1.
13. B. MACQ P. DESARTE and D. SLOCK (1992). Signal-adapted multiresolution transform for image coding. *IEEE Trans. on Info. Theory* **38** (2), pp. 719–746.
14. O. RIOUL and P. DUHAMEL (1992). Fast algorithms for discrete and continuous wavelet transforms. *IEEE Trans. on Info. Theory* **38** (2), pp. 569–585.
15. J. WOODS and S.D. O'NEIL (1986). Subband coding of images. *IEEE trans. ASSP-34*, (5).

APPENDICES

A. IMPLEMENTATIONAL DATA

This appendix provides the data one needs to implement the quadratic ($m=3$) cardinal spline MR scheme. If a higher order quasi-interpolation operator or more terms in the $\{a\}$, $\{b\}$ sequences are desired, then one must consult [5, 3].

The coefficients λ_{ij} (for the case $k=1,2$) are produced when applying the convolutional operators specified below to the image data. In Figure 9, the matrix represents the support (i.e. grid-points) over which the coefficients of the intensity samples I_{ij} are non-zero. The centre of the matrix represents the coefficient of I_{ij}.

Cardinal splines satisfy the following recursive identity:

$$N_m(x) = \frac{x}{m-1} N_{m-1}(x) + \frac{m-x}{m-1} N_{m-1}(x-1) \tag{38}$$

where $N_1(x) = \chi_{[0,1)}(x)$. These formulae can easily be expanded to explicit (non-recursive) definitions. Through the use of LPTA's (Linear Pascal Triangular Algorithms) [3], one can derive formulae to calculate the values of both

$$\begin{bmatrix} O & x & X & x & O \\ x & Q & I & Q & x \\ X & I & \# & I & X \\ x & Q & I & Q & x \\ O & x & X & x & O \end{bmatrix}$$

FIGURE 9. The arrangement of coefficients of the quasi-interpolation operator

	0	1	2	3	4	5	6	7
$\{p_k\}$	$\frac{1}{4}$	$\frac{3}{4}$	$\frac{3}{4}$	$\frac{1}{4}$				
$\{q_k\}$	$\frac{1}{480}$	$-\frac{29}{480}$	$\frac{147}{480}$	$-\frac{303}{480}$	$\frac{303}{480}$	$-\frac{147}{480}$	$\frac{29}{480}$	$-\frac{1}{480}$

TABLE 6. The reconstruction sequences for case $m = 3$

	$\{a_k\}$	$\{b_k\}$		$\{a_k\}$	$\{b_k\}$
0	0.033978977	0.049781017	12	-0.008232310	0.034166241
1	0.655340376	0.423982818	13	-0.000934671	0.003879280
2	0.655340376	-0.140377187	14	0.003544624	-0.014711266
3	0.033978977	-0.900597911	15	0.000402447	-0.001670285
4	-0.243780520	0.900597911	16	-0.001526227	0.006334313
5	-0.025936016	0.140377187	17	-0.000173284	0.000719182
6	0.103311291	-0.423982818	18	0.000657155	-0.002727399
7	0.011654634	-0.049781017	19	0.000074611	-0.000309662
8	-0.044411988	0.184116960	20	-0.000282955	0.001174351
9	-0.005039196	0.020974988	21	-0.000032126	0.000133332
10	0.019119634	-0.079343472	22	0.000121833	-0.000505646
11	0.002170658	-0.009011510			

TABLE 7. The decomposition sequences for the case $m = 3$

our splines and scaling functions efficiently. We did not pursue this approach. The decomposition sequences are derived from the roots of an Euler-Frobenius polynomial [5]. This complex polynomial (of order $2m - 1$) is defined as

$$E_{2m-1}(z) = (2m-1)! \sum_{j=-m+1}^{m-1} N_{2m}(m+j) z^{j+m-1}$$

and clearly has an intimate relationship with the cardinal splines. We do not have enough space to develop this approach further here. Interested readers are referred to [5, 3]; it should be noted that [5] contains some transcription errors in the quadratic decomposition sequences. By utilising the formulae presented there, one can check the sequences and produce additional terms (Table 7) contains the sequences we used).