PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 :
G06T 9/40, H04N 1/64

A1

(11) International Publication Number: **WO 98/19274**

(43) International Publication Date: 7 May 1998 (07.05.98)

(21) International Application Number: PCT/AU97/00725

(22) International Filing Date: 28 October 1997 (28.10.97)

(30) Priority Data:
PO 3294 28 October 1996 (28.10.96) AU

(71) Applicants *(for all designated States except US)*: COMMONWEALTH SCIENTIFIC AND INDUSTRIAL RESEARCH ORGANISATION [AU/AU]; Limestone Avenue, Campbell, ACT 2612 (AU). AUSTRALIAN NATIONAL UNIVERSITY [AU/AU]; Acton, ACT 2600 (AU).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: BONE, Donald, James [AU/AU]; 7 Tambo Street, Kaleen, ACT 2617 (AU). MCLAUGHLIN, John, Patrick [AU/AU]; Flat 51, 14 Boolee Street, Reid, ACT 2612 (AU).

(74) Agent: PIZZEYS PATENT & TRADEMARK ATTORNEYS; P.O. Box 291, Woden, ACT 2606 (AU).

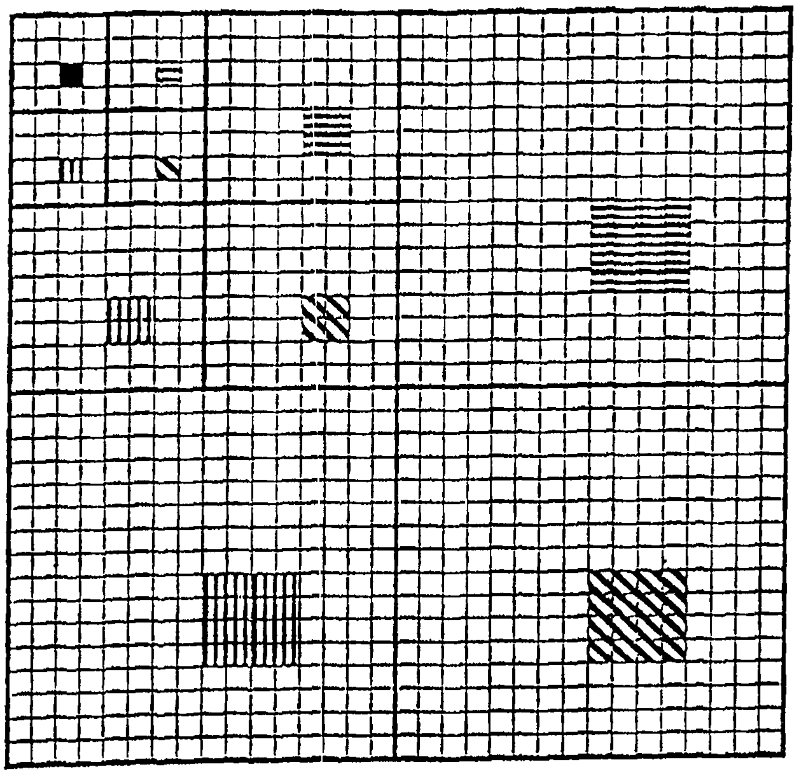
(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, TJ, TM, TR, TT, UA, UG, US, UZ, VN, ARIPO patent (GH, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published
With international search report.

(54) Title: IMAGE ENCODING

(57) Abstract

A method of embedded encoding of an image is disclosed in which image compression techniques encode spatial tilings of the image, said method including: precalculating the significance and zerotree information in a single pass; storing said significance and zerotree information in store, and interrogating said store to establish the significance status of any tree.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

"IMAGE ENCODING"

Technical Field

This invention relates to a system for and method of image encoding.

The invention has application to methods for the embedded encoding of an image in which image compression techniques encode spatial tilings of the image. The invention has particular application in progressive image transmissions.

Background of Invention

In our copending application, the specification of which is included herein by reference, there is described a method of progressively transmitting an image in which image compression techniques rely on spatial tiling of the image, the method including:-

allocating variable priority values to spatial regions within the image;

whereby a receiver of a transmitted image can interactively define the spatial focus of the image during transmission thereof.

Progressive image transmission systems are known and involve the transmission of image data in a way that the data received at the intermediate stages in the transmission can be used to reconstruct an approximation to the full image.

An embedded encoding is one in which the bits representing the image have been ordered as a single stream which can be truncated at any point so that an approximation to the image can be generated from the information to that point and such that that approximation has close to optimal distortion for the proportion of the information received. All the more compressive representations of the image are then embedded within the stream representing a less compressive representation. In a sense the bits in the stream are ordered by their importance, where the importance is determined by their magnitude (bit significance), spatial scale and spatial location.

It is desirable to produce an encoding which permits modification to the importance associated with the bits. In particular the spatial location and the scale can vary in importance depending on a number of factors. For example the importance of a particular region of an image will depend on the interest of the user viewing the image. In the situation where an image is being progressively transferred the user will want the bits associated with regions of particular interest to be delivered before the bits associated with regions of no particular interest. The decision about how interesting a region is may only be possible after some small fraction of the image has been delivered.

Scale is also something whose importance can vary dynamically during an image transmission. If an image larger than the viewing area on a monitor is initially viewed on a scale appropriate to fit the whole image on the monitor, then it does not make sense to send those bits which are associated with the information at a finer scale than can be presented on the monitor. If the user decides that a particular region is interesting and zooms the region, then those bits associated with the fine scale information are now required.

This involves a need to reprioritise the bits in the image after partial transmission of the image. Ideally this should be done without having to recode the image or retransmit any information that has already been received.

It is not clear at first that this is even possible if an embedded style of representation with good compression characteristics is to be retained. Concentration initially is on the spatial prioritisation. Briefly, the approach relies on a "tiled" representation of the image. A tile is a spatially localised subset of the image information. The term "tile" is used rather than "block" to emphasis the fact that although the tiles are spatially localised and independant of the other tiles, they are allowed to overlap in the spatial domain. In this implementation they are in fact a collection of terms in the wavelet expansion. Each spatial tile is encoded independently in an embedded representation. The handling of these tiled embedded representations to provide Interactively Spatially-prioritised Progressive Image-retrieval (ISPI) is described in greater detail in our copending application.

Among the more successful examples of embedded encodings are the Embedded Zerotree Wavelet (EZW) coding of Shapiro and the related Spatial Partitioning In Hierarchical Trees (SPIHT) encoding of Said and Pearlman. See the following:-

A.Said, W.A.Pearlman, "A New Fast and Efficient Image Codec Based on Set Partitioning in Hierarchical Trees," Transactions on Circuits and Systems for Video Technology vol. 6(3). pp 243-250 (1996);

J.M.Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," IEEE, Trans. on SP, 41 (1993), pp 3445-3462, and

J.M.Shapiro, "An Embedded Hierarchical Image Coder using Zerotrees of Wavelet Coefficients," Proc. Data Compression Conference, J.Storer, M.Cohn Eds (1992), pp 214-223.

See also US Patents 5412741, 5321776 and 5315670 to Shapiro.

This invention relates to the particulars of the embedded image encoding method and system which is used to encode the independant spatial tiles. An algorithm is developed which is an improvment on the methods of Said and Pearlman and Shapiro described above. Throughout the specification references are made to this prior art by way of explaining the advances of the present invention.

Summary of Invention

The present invention aims to provide a further alternative to known progressive image transmission systems and methods utilising embedded encoding of an image in which image compression techniques encode spatial tilings of the image.

As used herein the expression "embedded encoding of an image in which image compression techniques encode spatial tilings of the image" is to be understood to include reference to the application to an image of an algorithm which assumes that the image is divided into a set of tiles with each tile containing a set of coefficients which represent the image information.

These coefficients may be arranged or ordered in a two dimensional grid with the coefficients generally decreasing in magnitude (significance) from the top left to the bottom right corner. The ordering is not strict and occasionally a more significant coefficient can appear below or to the right of less significant.

The ordering generally results from the transformation process by which the coefficients are achieved and should not in general require information to be stored on a per tile basis to achieve the ordering, although this is possible.

To structure the coefficients within one tile they are organised in a hierarchy defined by descendant relationships with the top left coefficient being the highest antecedant. The direct descendants of any coefficient are referred to as its children. These relationships define a tree structure with a single root node at the top left coefficient.

Through the use of an appropriately chosen threshold value a first bit plane in the tile is defined by testing the coefficients for their significance relative to the threshold. The next bitplane can be defined by reducing the threshold and again testing for those points which become significant at this threshold level. Information about already significant points is sent as refinement bits which are the significance of the remainder of the coefficient after subtracting the earlier threshold value from the coefficient for any threshold levels at which the coefficient remainder was found to have been significant.

This invention in one aspect resides broadly in a method of embedded encoding of an image in which image compression techniques encode spatial tilings of the image, said method including:-

precalculating the significance and zerotree information in a single pass;

storing said significance and zerotree information in store, and

interrogating said store to establish the significance status of any tree.

In another aspect this invention resides broadly in a method of embedded encoding of an image in which image compression techniques encode spatial tilings of the image, said method including:-

ordering the coefficients in said spatial tilings whereby said tiles are defined as having the constraints (a) that all the children of a coefficient are visited before the siblings of that coefficient, and (b) that all the siblings of a coefficient are visited before any non-descendant non-siblings are visited, whereby the algorithm can be implemented without using lists in the partitioning of the tree.

In a further aspect this invention resides broadly in a method of embedded encoding of an image in which image compression techniques encode spatial tilings of the image, said method including:-

transmitting significant bits, refinement bits and partitioning bits in the order in which the corresponding coefficients are encountered during a single pass of the coefficients for each bitplane.

In another aspect this invention resides broadly in a method of embedded encoding of an image in which image compression techniques encode spatial tilings of the image, said method including:-

for a given threshold treating as insignificant all components above a given scale in the tree.

In a still further aspect this invention resides broadly in a method of embedded encoding of an image in which image compression techniques encode spatial tilings of the image, said method including:-

splatting the image components corresponding to individual significant bits in the representation directly on to the image plane.

As used herein, the expression "splatting" means a process in which each incoming significant bit results in an update of the image itself through the addition of the relevant part of the basis function.

In a preferred embodiment the invention relates to a method as defined in any of the preceding statements, for the embedded encoding of an image in which image compression techniques encode spatial tilings of the image, wherein the pseudo-code description of the embedded encoding algorithm is as set out in FIG 9.

Description of Drawings

In order that this invention may be more easily understood and put into practical effect, reference will now be made to the accompanying drawings which illustrate a preferred embodiment of the invention, wherein:-

FIG 1 illustrates a hierarchical wavelet decomposition with the components which constitute a spatial tile;

FIG 2 illustrates a spatial tile in the wavelet domain;

FIG 3 illustrates the bits generated from a dyadic thresholding;

FIG 4 illustrates the spatial orientation tree partitioning of a wavelet tile;

FIG 5 illustrates the Coarse/Fine partitioning of the orientation trees in accordance with the present invention;

FIG 6 shows the root and four spatial subtrees of one of the orientation trees;

FIG 7 illustrates the Coarse-Tree tests array;

FIG 8 shows tree depth as a function of scale d(s), and

FIG 9 lists the pseudo-code description of the embedded encoding algorithm.

Description of Preferred Embodiment of Invention

A preferred embodiment of the invention will now be described with reference to the above illustrations by reference under appropriate headings to various aspects of the invention.

Wavelet Tiles

In general, it is only useful from the point of view of compression to hierarchically decompose the image on the wavelet basis to the point that the DC components are spatially decorrelated. For most images a depth of 3-5 levels in the hierarchy is sufficient to achieve this. The coefficients in the wavelet domain can then be collected into groups of components for which the centres of the corresponding basis function lie within a given spatial region of the image. This collection of coefficients can be thought of as a spatial tile, with one DC component (at the top left of FIG 1), and an hierarchy of AC components in a scale hierarchy mimicking the structure of the image subbands. By reconstructing the components in each spatial tile it is possible to arrive at a set of independent but overlapping image partitions which could be added together to reform the image.

The following description is concerned with the embedded representation of a single wavelet tile. The encoding of a full image is achieved by the encoding of all wavelet tiles.

FIG 1 illustrates a hierarchical wavelet decomposition with the components which constitute a spatial tile and FIG 2 shows The spatial tile in the wavelet domain.

Thresholding and bitplanes

The invention uses a dyadic sequence of thresholds producing a binary representation for the coefficients. The bits generated from such a dyadic sequence of thresholds for a simple although unrealistic distribution are illustrated in FIG 3.

The coefficient array corresponding to the wavelet tile is normalised using a predetermined positive threshold T such that 2T is larger than the magnitude of any of the values in the tile. For a wavelet coefficient value of w, the normalised coefficient is int(214*(w/T)). The coefficients are saved as 16 bit integer values with the first bit reserved for the sign and 15 bits used for the magnitude.

Tile partitioning (Orientation partitioning)

The spatial tiles of coefficients can be partitioned into three orientation subtrees. The coefficients in an orientation subtree are produced by the subset of wavelet basis functions with the same spatial orientation sensitivity. For this reason Said and Pearlman gave these trees the name Spatial Orientation trees. The orientation partitioning of a 4 level wavelet tile is illustrated in FIG 4. In this invention these partitions are called the H,D and V partitions.

Tile partitioning (Coarse/Fine partitioning)

Each orientation tree can be further partitioned into coarse and fine subsets. Because of the tendency of the magnitude of the coefficients to decrease from coarse scale to fine scale, components above a given threshold will tend to cluster in the coarse part of the orientation tree. The scale at which the number of significant components goes to zero will vary from one spatial tile to the next and will depend on the threshold, but by and large as the threshold is decreased, the scale at which significant components disappear will move towards the fine end.

The concept of a coarse/fine tree partition is illustrated in FIG 5 where the leftmost tree is partitioned at scale 1 and the rightmost tree is partitioned at scale 3. In this invention these are called level 1 to level 3 Coarse/Fine Partitions (CFP). A level 1 CFP corresponds to Said and Pearlman's type A LIS entry. A level 2 coarse/fine partition corresponds to a Said and Pearlman's type B LIS entry. There is no equivalent in Said and Pearlman's work to the higher level Coarse/Fine partitions of the present invention. Said and Pearlman also does not use the level 0 CFP which corresponds to the full tree.

<u>Tile partitioning</u> (Spatial partitioning)

Each orientation tree can be divided into a root node and four spatial subtrees as illustrated in FIG 6. This partitioning in conjunction with the orientation partitioning defines a tree structure for the tile. The tile has three Orientation subtrees and each Orientation subtree can be recursively divided into 4 spatial subtrees.

<u>Significance</u>

Central to the zerotree approach to coding is the concept of significance. A coefficient w(i,j) is said to be significant for a given threshold t if the coefficient is greater than or equal to the threshold. The zerotree algorithm of Shapiro actually tests a subtly different quantity, which is called the Just-Significance in the present invention.

Just-Significant points are those points which are significant at the threshold t but were not significant at the threshold 2t and are the key to the partitioning of the tree. Below the threshold at which a coefficient becomes significant, subsequent bits are equally likely to be 1 or 0 and there is no advantage to looking for zerotrees among refinement zeros. The Shapiro algorithm actually tests for trees in which no coefficients are Just-Significant. This has some advantage in the situation where an isolated significant coefficient is surrounded by insignificant coefficients. The mechanism Shapiro uses to achieve this is to replace any coefficients in the tree which is found to be significant by zero and retain the coefficient in a List of Significant Points. Lower significance bits of these coefficients are handled as "refinement" bits. Shapiro always tests the whole tree from the root node to the bottom of the tree.

Said and Pearlman use a subtly different mechanism in which trees are partitioned at each threshold so that for that threshold each tree is split into a coarse part which may be significant and a fine part which is wholly insignificant. This partitioning is the starting point for tests of the same tree at the next threshold. As the threshold is lowered the trees are only ever split, they are never rejoined. Any points within an existing wholly insignificant partition which become significant at the next threshold are therefore by implication Just-Significant. If a tree cannot be split into a coarse and fine part with the fine part insignificant, then the tree is divided into its root and four spatial subtrees. In the limit the trees will be split into individual coefficients and only refinement bits will be sent. This will then require the same number of bits as sending a binary raster representation of the bitplane.

In the algorithms described here the coefficients in the tile are traversed in strict order sending both partitioning information and sign and magnitude or refinement information associated with the root node of each subtree as we encounter them. Lists of addresses are not used as is the case in Shapiro and Said and Pearlman. Manipulating lists can be quite expensive and by avoiding the use of lists the present invention gains in efficiency.

Hierarchical Partitioning without Lists

Said and Pearlman generalised the zerotree partitioning concept used by Shapiro with the use of their Type A and Type B entries to the List of Insignificant Sets.

In the present invention this approach is more clearly understood through the concept of a Coarse-Tree, which is a tree whose fine components below some scale are all insignificant. The concept of a Coarse-Tree encompasses the Type A and Type B sets of Said and Pearlman but provides a conceptual basis within which more general Coarse/Fine Partitioning is allowed rather than just the two levels used by Said and Pearlman.

If the use of lists is to be avoided, another mechanism for tracking the partitioning process is needed. For each coefficient a Coarse/Fine partitioning level index (CF(I)) is saved which records the CF partitioning level for the orientation tree with that coefficient at its root. As the threshold is lowered this index is either left the same or incremented but never decremented. Once it has incremented above the valid range for that tree (above a preset maximum value or such that there are no components in the tree finer than the partitioning level), then no further Coarse-Tree tests are carried out on that tree for subsequent thresholds.

At a given threshold, the Coarse-Tree tests on a given tree are repeated until they either succeed in finding a level at which the fine components are all insignificant or CF(I) is incremented above the valid range.

During the traversal of the tree the present invention also carries information about the effective depth of the current tree and trees in the current scale. If the fine parts of the tree are found to be zero at some point then the effective depth of the tree is adjusted accordingly. As the tree is traversed down, these variables permit the system to keep track of what part of the tree has been found to be zero.

Coding of the Zerotree Structure

The tree structure of the wavelet tile is traversed from the root of the tree. Because of this, tests for a zerotree in the upper parts of the tree will contain tests of lower subtrees. If these tests are conducted at the time that they are required by the algorithm then this would involve some repetition.

In the present invention a scheme for compactly storing the results of all possible zero tree tests is used. The test is conducted in a single traverse of the tree, performing tests on all of the bitplanes simultaneously.

One algorithm for achieving this is described subsequently. It will be appreciated that the algorithm presented is but one possible algorithm for practising the present invention. For example, the calculation could take advantage of the zero level tree calculation in calculating the higher Coarse/Fine level partitioning.

The resulting structure is an array which is 16 bits deep and which takes the form of an irregular 3 dimensional matrix $Z^k(i,j)$. The form of the data structure is illustrated in FIG 7. The three blocks of data in that figure corresponding to the three values of k which is the Coarse/Fine partitioning level index. For k=0 the data block is of the same dimensions as the wavelet. For each increment of k the size of the data block in each of the spatial dimensions (i,j) is reduced by a factor of 2.

Some Definitions

The hierarchical trees are defined by the set of descendency and sibling relationships which are based on an addressing scheme for the wavelet tile with its origin at the root node.

$I = (i, j)$	A wavelet tile coefficient address
$C_1^1(I) = (2i, 2j)$	First Child of I
$C_2^1(I) = (2i+1, 2j)$	Second Child of I
$C_3^1(I) = (2i+1, 2j+1)$	Third Child of I
$C_4^1(I) = (2i, 2j+1)$	Fourth Child of I
$C_1^n(I) = C_1^1(C_1^{n-1}(I))$	First nth order Grandchild of I
$P^1(I) = (i\backslash 2, j\backslash 2)$	Parent of I
$P^n(I) = P^1(P^{n-1}(I))$	nth order Grandparent of I
$N(I)$	Next Sibling of I in sequence $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 1$

The root node of each tile is a special case where (i,j) = (0,0). In this case the first child of the root tile is the root itself and is excluded from the hierarchy as a special case when the algorithm calls for the children of this node. the root tile of the node also has no siblings.

FIG 7 illustrates the Coarse-Tree tests array $Z^k(i,j)$. The Coarse-Tree tests are calculated for all bitplanes and all Coarse-Tree Partitioning Levels in a single scan of the coefficients of the wavelet tile. The figure illustrates the Coarse-Tree array and the corresponding fine components in the wavelet tile for three different CF Partitioning levels.

Given the relationships defined above, the offspring and descendants which define the tree associated with a given coefficient are defined as follows:-

$$O^1(I) = \{C_1^1(I), C_2^1(I), C_3^1(I), C_4^1(I)\}$$
$$O^2(I) = O^1(O^1(I))$$
$$D^0(I) = I \cup D^1(I)$$
$$D^1(I) = D^0(O^1(I))$$
$$D^2(I) = D^0(O^2(I))$$

For the purposes of the encoding it is also necessary to define the following:-

- w(I), where I=(i,j) is the index into the spatial tile, is the array of wavelet components for the spatial tile encoded as sign and magnitude.

- Jn(I) is the just-significance of w(I) at threshold $t=T/2^n$, $J_n(I) = t|w(I)|<2t$. With a set of index values S= {S0,S1,...Sm} we use a shorthand Jn(S) = {Jn(S0),Jn(S1),...,Jn(Sm)}

- Sig(w(I)) the integer value with only the most significant bit of |w(I)| set. We use a look up table to perform this operation.

- $Z_{nk}(I)$ is a boolean array such that $Z_{nk}(I) = OR(J_n(D^k(I)))$, where I is an index in the spatial tile and OR(S) takes the logical OR of each member of the set S.

- Sgn(w(I)) is the sign of w(I).

<u>The coding algorithm - Generating Coarse-Tree arrays</u>

The Coarse Tree array is generated by propagating the Just Significance of each coefficient up it's inheritance tree. The logical operation are performed across all 16 bitplanes simultaneously.

- Set the number N of threshold levels for complete encoding - we use 15 which is convenient for short integer representation of the coefficient.

- Allocate memory for $Z^k(I)$ and initialise to zero

- Set the Coarse-tree testing limits k_{min} and k_{max}

```
For each I,
   for m=$k_{min}$ to $d_0 - d(I)$,
      for k = $k_{min}$ to $\min(m,k_{max})$,
         $Z^k(P^m(I)) = Z^k(P^m(I))$ | Sig(w(I))
      end
   end
end
```

$Z_n^k(I)$ is the bit of $Z^k(I)$ which gives the result of the Coarse-Tree test corresponding to threshold $t = T/2n$.

The ¦ operator is a 16 bit deep OR.

The Embedded Encoding Algorithm

Each bitplane in the wavelet tile, starting at the most significant bitplane, is coded with a single pass of the coefficients. In each pass, the algorithm iterates through the coefficients in fixed order, only missing those coefficients which are found to be insignificant fine components of a Coarse Fine partitioning a subtree of the tile.

The scan of the components starts at the root of the tree, visiting all children of a coefficient before visiting any siblings, and visiting all siblings of a coefficient before visiting non-descendent non-siblings. Each coefficient is assigned a Coarse/Fine partitioning level which is initially set to some minimum value greater than or equal to 0. For all points with valid CF levels, the algorithm performs a series of Coarse-Tree tests which check whether any of the fine components of the Coarse/Fine partitioned tree are significant. After each test the result of the test is output.

If the result is True (the tree has a Significant point), the Coarse/Fine partitioning level is incremented for the coefficient at the root of the current tree and the test is repeated until either the test returns a False result or the CF level is incremented beyond the valid range for the tree.

If the test returns a False result or the CF level is beyond the valid range for the tree then the algorithm sends either the sign and magnitude for the current coefficient (sending only that information which cannot be determined from what has been sent) and moves to the next sibling of the current coefficient. If there are no more siblings then the algorithm moves back up the tree one level at a time until it finds a level with a next sibling. If there are no more next siblings at the topmost level then the scan for the bitplane terminates and the algorithm moves to the next bitplane.

The algorithm will always send either a refinement bit or a sign and magnitude for the coefficient at the root of the current tree. If the coefficient is known to have been significant in a previous bitplane pass (which can be tested for in both the encoder and decoder), then a refinement bit is sent. If it is insignificant then a 0 is output. If it is Just-Significant then a 1 is output followed by a zero if the component is negative or a 1 if the component is positive.

There is a special case when the result of a CF level 0 Coarse Tree test is True and the subsequent result of a CF level 1 Coarse tree test is False. In this case the root is obviously the significant point and only a sign bit need be sent if the point is just significant. Also if a level 0 test is false, the root must be zero and need not be sent.

Variables

- SN(s) is the Sibling Number, which records the current sibling being tested for each scale. It is initially 1 for all scales. The valid range for the Sibling Number SN(s) for all scales other than 1 & 0 is 1 to 4. The valid range for scale s is summarised by the table

Table 1: Valid Sibling Range

Scale	Sibling Number SN
s=0	1
s=1	2...4
s>2	1...4

- CF(I), I=(i,j) indexing over the wavelet components, is a two dimensional array of the Course/Fine Partitioning Level for the tree with its root at the component I.

- $d(s), s = 0$ to d_0, is the effective tree depth in each scale of the tree. Because the orientation trees are trasversed in sequence, the same variable can be used for each orientation tree.

- $d_s(s), s = 0$ to d_0, is the effective depth of the current set of active siblings in each scale of the tree. Because the orientation trees are traversed in sequence, the same variable can be used for each orientation tree. Because we perform a depth first traversal of the tree there will be at most one set of active siblings (four coefficients) in each scale. This variable is used to get the correct effective depth of the next sibling when the we traverse within a sibling group or back up the tree.

- d_c is the depth of the current tree. This can differ from the sibling depth and is carried down the tree to form the sibling depth of scales below the current scale.

Pseudo code description of the encoding

A pseudo-code description of the embedded encoding algorithm is as set out in FIG 9.

Decoding the bitstream

This encoding is intended to be used with the Interactive Spatially-prioritised Progressive Image-retrieval ISPI technique described in our co-pending application. The decoding at the client end of the connection gets the output of the encoder for a given tile interleaved with the bitstream encoding of other tiles. The ISPI application describes how these interleaved streams can be reformed into independent stream. For the purposes of this invention it is sufficient then to consider the decoding of a single tile.

The decoder follows the same execution path as the encoder. Whenever a conditional statement is executed in the decoder the bitstream will provide the necessary information to keep the execution paths of the encoder and decoder synchronised.

In the decoder, the wavelet coefficients are not reconstructed for reasons of efficiency. Instead the incoming bit information is utilised to directly reconstruct the image. The image is only modified when a significant bit (either a refinement bit or a just-significant bit and sign) are received. A variable is kept with one bit per coefficient for the sign and one bit per coefficient to indicate that subsequent bits are refinement bits. Each incoming significant bit results in an update of the image itself through the addition of the relevant part of the basis function; a process which is referred to herein as splatting.

The term "splatting" has been adopted from the volume visualisation community. A more accurate result could be achieved by refining the image as a result of the incoming 1 and 0 coefficient bits. However this would require many more splatting operations and the final result for the full image is identical.

The splatting calculations are very efficient because the splatting functions for any signifcant bit can be precalculated and stored. This only requires one function per subband corresponding to a bit in the uppermost bitplane. For each coefficient within a subband the splatting function is a simple translation of the generic splatting function for that subband. The splatting function for each subband in each bitplane can be calculated from the equivalent function for the previous bitplane with a simple bitshift.

Some speed enhancement of splatting can be achieved by trimming the numerous small components from the function. If the bistream is saved in a buffer at the client then these trimmed components can be added in later with a second pass of the bitstreatm. In this way a perceptually accurate image can be achieved quickly while still retaining the ability to achieve a numerically accurate image with sufficient time.

To assist further in an understanding of the invention, information is now provided relating to Biorthogonal wavelet transforms and the Symmetric extension at the boundary.

Biorthogonal Wavelet transform

Orthogonal wavelet transforms have the problem that they cannot in general be symmetric. This makes the handling of image boundary difficult. With orthogonal wavelets there are two approaches to dealing with image boundaries. The first is to use a periodic boundary condition. This has the problem that any disparity between the intensity at opposite image boundaries, under a periodic boundary condition, will result in discontinuous behaviour at the image boundary in the function being coded. This make the compression less efficient. The only simple alternative to a periodic boundary condition is to modify the basis functions to incorporate the boundary for those regions where the basis functions overlap the boundary. This is relatively complex and is increasingly so as the support of the basis function is increased.

If the basis functions could be made symmetric then it would be possible to use a reflection symmetric extension to handle the boundary. As discussed above, discrete orthogonal wavelets cannot be symmetric for support greater than 2. This is easily seen as follows. Firstly the scaling function must have an even number of coefficients. If there were an odd number of coefficients say 2N+1, then for translations of +2N or -2N the scaling functions would only overlap at one point. To maintain orthogonality of the translates, one of those values would therefore have to be zero leaving an even number of coefficients. If both were zero then we could apply the same argument for N = N-1.

So orthogonality requires an even number of non-zero coefficients. If symmetry is required then then, for an even number of coefficients $(c_N, c_{N-1}, \ldots, c_1, c_1, \ldots, c_{N-1}, c_N)$, for translations of the basis of 2(N-1), there would be an overlap of 2 points, with the untranslated basis. Because of symmetry the orthogonality condition would then look like $2c_{N-1}\ c_N=0$. But this can only be true if one or both of these coefficients are zero. If $c_N=0$ and $c_N-1\neq0$, then we can re-apply the argument for N = N-1. If both are zero then we can reapply the argument for N=N-2. If $c_N\neq0$ and If $c_{N-1}=0$, then at displacement of 2(N-2) we would get a condition for orthogonality of $2cN_{N-2}c_N=0$, but if $c_N\neq0$, then c_{N-2} must be zero. We can proceed with smaller and smaller displacements, requiring successive coefficients to be zero until at zero displacement we find that $c_N=0$ for orthogonality. So orthogonality and symmetry are incompatible requirements for discrete wavelets (except for the case where N = 1, for which there would never be any overlap of their translates.

Biorthogonal wavelets satisfy a much weaker condition that does not require orthogonality of the basis functions. For an orthogonal basis, the inverse transform basis is identical to the forward basis. For the discrete case this is equivalent to saying that the matrix of column vectors representing the orthogonal discrete basis has an inverse which because of the orthonormal condition is equal to the transpose of the forward transform matrix. If the basis components are not orthogonal then this no longer applies. But it is still possible to have a discrete inverse basis with compact support. This means however that we can now impose a symmetry constraint on the basis functions, which allows us to use a symmetric reflection in the boundary to handle the image boundary.

For this work we use the same biorthogonal basis used by Said and Pearlman, which was originally described by Antonini et al. The components of the discrete basis are given in Table 2. The low frequency discrete basis functions are translates of the synthesis basis , centred on the even points (0,2,4...) at a given scale, the coefficients for which are calculated by projection onto translates the analysis basis centred on even points at a given scale. Similarly, the high frequency discrete basis functions are translates of the synthesis basis, centred on the odd points (1,3,5...), the coefficients for which are calculated by projection onto translates the analysis basis centred on odd points.

Thus the transform of a 1-D sequence of values xi, into the low frequency components X and the high frequency components Y would proceed as

$$X_j = \sum_{n=-k}^{k} x_{2j+n} h_{-n}$$
$$Y_j = \sum_{n=-k}^{k} x_{2j+n+1} g_{-n} \qquad (1)$$

If we define the upsampling of X to be X' , such that

$$X'_{2n} = X_n$$
$$X'_{2n+1} = 0 \qquad (2)$$

and the upsampling of Y to be

$$Y'_{2n} = 0$$
$$Y'_{2n+1} = Y_n \qquad (3)$$

Then the resynthesis is simply expressed as

$$\hat{x}_j = \sum_{n=-l}^{l} X'_{j+n} \tilde{h}_{-n} + \sum_{n=-l}^{l} Y'_{j+n} \tilde{g}_{-n} \qquad (4)$$

However, computationally, this is not as efficient as it might be, due to the padding with zeros. A more efficient algorithm can be achieved if we define the interleaved coefficients Z such that

$$Z_j = X'_j + Y'_j \qquad (5)$$

Then if we define the interlaced filter kernels

$$\begin{aligned} \hat{h}_{2j} &= 2j \\ \hat{h}_{2j+1} &= \bar{g}_{2j+1} \\ \hat{g}_{2j} &= \bar{g}_{2j} \\ \hat{g}_{2j+1} &= \tilde{h}_{2j+1} \end{aligned} \qquad (6)$$

the resynthesis becomes

$$\begin{aligned} \hat{x}_{2j} &= \sum_{n=-l}^{l} Z_{2j+n}\hat{h}_{-n} \\ \hat{x}_{2j+1} &= \sum_{n=-l}^{l} Z_{2j+n+1}\hat{g}_{-n} \end{aligned} \qquad (7)$$

The two dimensional image data is handled by first transforming each row of the image, then transforming each column of the result. This would produce four subband images from the original image which we label LL0, LH0, HL0 and HH0, where H and L refer to the 1D filters used to produce each subimage and the number indicates the level in the hierarchy. The hierarchical decomposition of an image would take the LL0 subband and apply the same analysis to it to produce the subbands LH1, HL1 and HH1.

Table 2: Components of the discrete biorthogonal wavelet basis

n	0	±1	±2	±3	±4
$h_n/\sqrt{2}$	0.602 949	0.266 864	-0.078 223	-0.016 864	0.026 749
$g_n/\sqrt{2}$	0.557 543	-0.295 636	-0.028 772	0.045 636	0
$\tilde{h}_n/\sqrt{2}$	0.557 543	0.295 636	-0.028 772	-0.045 636	0
$\tilde{g}_n/\sqrt{2}$	0.602 949	-0.266 864	-0.078 223	0.016 864	0.026 749

The symmetric extension at the boundary.

If we have data (x0,x1,....xN-1) and a symmetric basis function with support 2K+1 such that the basis coefficients (b-K,b-K+1,....,bK-1,bK) satisfy

$$b_{-j} = b_j \qquad (1 \le j \le K) \tag{8}$$

and we symmetrically extend the boundary of the image so that

$$\begin{aligned} x_{-j} &= x_j \\ x_{N+j-1} &= x_{N-j-1} \end{aligned} \qquad (1 \le j \le K) \tag{9}$$

Then the coefficients in the transform will be related by

$$\begin{aligned} X_{-j} &= X_j \\ X_{N+j-1} &= X_{N-j} \\ Y_{-j} &= Y_{j-1} \\ Y_{N+j-1} &= Y_{N-j-1} \end{aligned} \qquad (1 \le j \le K) \tag{10}$$

If we consider the interleaved form of the coefficients then the reflection extension at the boundary becomes simply

$$\begin{aligned} Z_{-j} &= Z_j \\ Z_{N+j-1} &= Z_{N-j-1} \end{aligned} \qquad (1 \le j \le K) \tag{11}$$

This permits the boundary coefficients to be regenerated for the synthesis without having to store extra coefficients.

Implementation

To further enable a fuller understanding of one manner in which the invention may be practiced, one manner of implementing the invention is now described.

1. Introduction

The demonstrator was implemented in Java in the interests of platform independence. It is assumed that the reader is familiar with the design of this project and the Java programming language, in particular threads, the java.awt.image package and the java.net package. In particular it is important to be aware of the following Java primitive data type sizes.

TABLE 1. **Java primitive data type sizes**

Type	Size
byte	8 bits
short	16 bits
int	32 bits
float	32 bits

It is assumed that this document will be read in parallel with the code itself. The implementation consists of the following three parts:

- **Encoder** - Converts GIF or JPEG files into encoded files by applying the wavelet transform then partitioning by doing every bit-plane pass over every tile in order and writing the resulting data to a file. The encoder is implemented as a Java application.
- **Server** - Opens a network socket and listens for incoming connections from the client. Upon connection the server starts a new Thread to serve the client and reads the requested encoded image file (produced by the encoder). The server serves pass data according to the current priority which may be modified by client requests. The server is also implemented as a Java application.
- **Client** - The client is a Java applet that may be run inside a web browser window. It connects to the server (which must be running on the same machine that the client applet was loaded from). The client also maintains a current priority map that is identical to the one in the server. When coefficient information is received the image is updated through splatting of a `footprint` (see below).

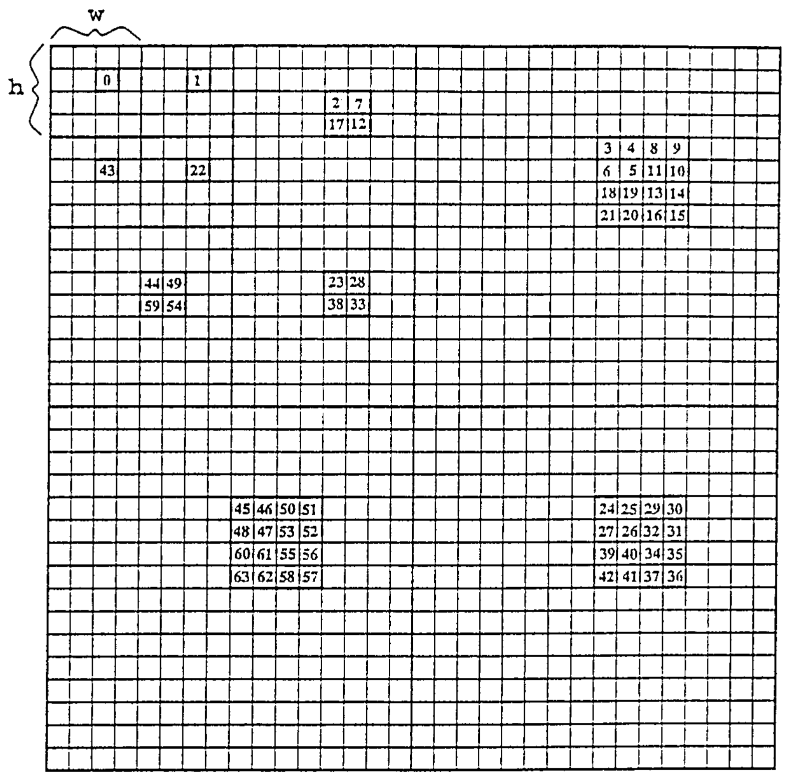
2. The coordinate system

There are two basic methods of specifying the coordinates of coefficient pixels. The one that fits most easily into the algorithmic model is one where pixels are indexed firstly by tile, then by coordinates within the tile (with the origin in the top-left corner). This model allows simple traversal of the spatial orientation tree in each tile. The operations on coordinates x, y are as follows:

- go to first child → x *= 2; y *= 2;
- go to first sibling → x++;
- go to second sibling → y++;
- go to third sibling → x--;
- go to parent → x /= 2; y /= 2;

(note that going to the first child from the 0,0 pixel is a special case, in fact we skip to the second child). However in order to simplify the data structures a different coordinate system is used. In this system pixels are specified by their global coordinates in the coefficient image rather than the coordinates within the tile. Note that in the general case the coordinate manipulations for traversing the spatial orientation tree of a tile are exactly the same. Figure 1 shows the distribution of a spatial tile on the coefficient image with a traversal of the spatial tree numbered. The tree will not always be traversed to its full depth.

FIGURE 1. Traversal of the spatial tree for an example tile.



Note the special case when moving between the highest 4 nodes of the tree. In these cases we increment/decrement by w or h. In all cases we traverse depth-first and clockwise among siblings starting at the top-left. Traversal is implemented in the `do_pass` methods of the encoder and client (see below).

3. The `priority` class

The priority class is shared by the server and the client. It is used to represent and modify the tile priority mapping. The constructor sets the initial priority to a uniform value of 1/3 for each tile.
The modification functions are of the form `do_*_priority` and are described below. Each takes an `op` argument which specifies the operator to use when modifying the priorities array. It is one of the following:

```
set_operator = 0
add_operator = 1
max_operator = 2
```

`set_operator` writes over the existing value, `add_operator` sums the existing and the new value and `max_operator` takes the maximum of the existing and new value. At present there is no user-interface support for the max operator.

3.1 The `do_uniform_priority` method

`do_uniform_priority` contributes a constant priority value of the given height to the map.

3.2 The do_hump_priority method

do_hump_priority contributes a Lorentzian function to the priority map. The equation used is:

$$\frac{ht}{1+\frac{d^2}{r^2}}$$

Where ht is the height of the hump, d is the distance from the center and r is the radius of the hump.

3.3 The do_polyline_priority method

do_polyline_priority is similar to do_hump_priority except the distance from a polyline is used. The dst_sqr_to_polyline method is used to calculate the d^2 term. See Graphics Gems II, Academic Press for a description of the algorithm.

3.4 The do_polygon_priority method

do_polygon_priority is similar to do_polyline_priority except the polyline is taken to be a closed polygon (there is an implicit edge between the last and first elements of the polyline) and points lying inside the polygon are taken to be at 0 distance from the polygon. The resulting shape is a plateau function with Lorentzian edges. The pt_in_polygon function is used to determine whether a point lies inside the polygon or not. The algorithm is a simple horizontal ray-intersection count.

3.5 The do_disc_priority method

do_disc_priority is similar to do_polygon_priority except the plateau shape is given by a circle instead of a polygon.

4. The encoder application

The encoder is implemented as a single class in a single file encoder.java. It implements ImageConsumer because it consumes the GIF or JPEG image specified on the command line.

4.1 Variables

- int test_level, init_level - specify a Said and Pearlman configuration (2,1) but may be changed. These correspond to k_{min} and k_{max} in the TWEZIR document.
- float[] basis_enc_h, basis_enc_g - store the wavelet analysis basis.
- int bitplanes - the number of bitplanes that will be saved to file for each wavelet tile.
- int depth - the number of wavelet transforms applied to the image (d_0 in the TWEZIR document).
- short[] sig_bit - a lookup table implementing a function that clears every bit except the most significant one. For example sig_bit[0010010110010110] = 0010000000000000. This is used to calculate the significance mask of a particular coefficient bit representation.
- int width, height - the dimensions of the image in pixels.
- int w, h - the dimensions of the image in tiles.
- int tile_d - the dimensions of a tile.
- float T0 - the partitioning threshold corresponding to the first bit-plane.

- `byte[][] T` - the current bitplane of each tile. We always start at 14 and count towards 0 stopping after `bitplanes` passes over the tile.
- `int[] pixels` - the pixels array. Each element is 4 bytes of the form ARGB where A is the alpha channel (always 0xff), R, G and B are the colour channels.

4.2 The `main` method

`main` analyses the command line arguments allowing them to override the default values for `bitplanes` and `depth`. It loads the input image file and specifies `this` as a consumer of that image.

4.3 The `imageComplete` method

`imageComplete` is called when the image has been completely loaded in and the `pixels` array has been defined. The encoded image file is opened and the header information is written giving the image dimensions, depth and number of bitplanes.

4.3.1 The wavelet transform

Next the wavelet transform is applied depth times to construct the wavelet coefficients. The transform is carried out using `float` values for accuracy. During the transform the maximum absolute coefficient value `max_coeff` is found. After the transform the final coefficients are stored in the temporary `working2` array.

4.3.2 The `coeffs` array

The values from the `working2` array are converted to `shorts` and placed in the `coeffs` array.
The `max_coeff` value is used so that the coefficient of maximum absolute value will be stored as `0xffff` (if it is positive) or `0x7fff` (if it is negative). `T0`, the initial threshold value corresponds to 0x4000 (or `1<<14`).
Coefficient entries in the coeffs array have the following bit layout:

$p\ s_0\ s_1\ s_2\ s_3\ s_4\ s_5\ s_6\ s_7\ s_8\ s_9\ s_{10}\ s_{11}\ s_{12}\ s_{13}\ s_{14}$

Where p is set if the coefficient is positive, s_i is used either to check whether the coefficient is just-significant at bitplane i or to check the refinement bit at bitplane i. Throughout the partitioning code expressions of the form `(c&0x8000)` are used to extract the sign bit and `(c&0x7fff)` is used to extract the absolute value. Also a value `this_sig` is used as a mask against the coefficient value for the current bitplane. For example on the first bitplane (corresponding to `T0`) `this_sig = 0100000000000000` at subsequent bitplanes it is `0010000000000000` and so on. Therefore if it is known that a coefficient has not yet been found to be significant the expression `((c&0x7fff)&this_sig) != 0` provides the just-significance. Otherwise it provides the refinement bit value.

4.3.3 The `sig` array

The `sig` array is central to the bit-wise implementation of the partitioning algorithm. It is used to lookup the significance of the descendants of a particular pixel with respect to the current significance level (`this_sig`). The synopsis of the sig array is as follows:

```
sig[ level ][ x ][ y ]
```

Stored in this element is a `short` bit array providing the significance of the `level` descendants of the coefficient at x, y for all 15 bitplanes (the sign bit is not used). For example to do a level-2 test of coefficient 234, 126 with respect to the bitplane corresponding to `this_sig`, one would use the value

```
    ( sig[ 2 ][ 234 ][ 126 ] & this_sig ) != 0
```

This corresponds to the value $S(\mathcal{D}(i,j))$ of Said and Pearlman. The pixel coordinate system used is the same as described above.

4.3.4 The partitioning algorithm

In the encoder partitioning is done by executing `bitplanes` tile-passes over each tile in turn. Each pass over a tile produces a bit-string. The length of these bit-strings are written to the output file and the data itself is added to the `image_data` array. When all the string lengths for a particular colour channel have been written, the image_data for that channel is written in one chunk. All the partitioning work is done in the `do_pass` method.

4.3.5 The `do_pass` method

The do_pass method appears both in the encoder and the client. Both versions are structurally similar except the client inputs bits where the encoder outputs bits (among other things).
Firstly the `this_sig` mask is defined for the current tile at the current bitplane (see above). The `while(d>=0)` loop constitutes the depth-first walk over the spatial orientation tree (See figure 1). The `out_bit` method is used to write bits to the buffer for the current tile. After the pass over the tile the buffered data is appended to the `image_data` array and will be written to file after the current colour channel has been done.

5. The server application

The server is the simplest of the three components. It has nothing to do with wavelets or partitioning. It reads the encoded image file and serves it to remote clients based on a dynamic prioritisation. The `server` class itself is just a connection daemon that starts off new threads to handle individual connections. This allows many simultaneous connections.

5.1 The `run` method

The `connection` class extends `java.lang.Thread` and as such is started through the `run()` method. `run()` calls `setup_file()` to establish the image data and then enters a wait -> serve loop.

5.2 The `setup_file` method

setup_file() firstly attempts to read the name of an encoded image file from the input network connection. It then reads the header information from that file and passes it on to the client. Then it reads the image data from the file and counts the total number of bits in the image which it also passes to the client. This allows the client to display the total file size in its status bar as the image is loading.
In the event of any error, a 0 is sent to the client in place of the width parameter (the first expected parameter) and the connection is stopped.
The `tile_perm` array is also defined at this point. `tile_perm` is a random permutation of the wavelet tiles that is shared by the client. It is used (rather than a pair of for-loops over the tiles) because the image can be updated at any time and it doesn't look good if tiles to one side of the image have been better defined than the other side. `tile_perm` allows tiles to be served in a random order.

5.3 The `serve_request` method

`serve_request` reads a request from the input network connection and executes it.
All requests are a string of one or more integers. The first integer is the request type. It must be one of the following:

```
pass_request  = 0
flat_request  = 1
hump_request  = 2
disc_request  = 3
pline_request = 4
pgon_request  = 5
```

5.3.1 Pass requests

`pass_request` is a request for a priority pass over the tiles of the image. It is executed by visiting each tile in the image once according to the order defined by the `tile_perm` array (see `setup_file` above). This is not to be confused with a tile pass which concerns a single tile only.
If there are no more passes for any tiles left to be sent, then a -1 is sent to the client. Otherwise the current request number is sent.
The `passes_sent` value for each tile is incremented by its `priorities` value. If the integer part of the `passes_sent` value increases as a result, the tile "fires" and data is sent for that tile. Note that the tile will also fire at the same time in the client and data will be expected for that tile. All three colour channels are sent at the same time when a tile is served. The `out_bit` method (see below) is called to send individual bits onto the output stream.

5.3.2 The priority requests

The priority requests consist of `flat_request`, `hump_request`, `disc_request`, `pline_request` and `pgon_request`. They correspond to methods of the `priority` class (see below) and are used to set or modify the priority map. The arguments to the priority class methods are input from the client as integers. Floating point arguments are converted from integers with a scaling factor of 1000.
As with the `pass_request`, the current `request_number` is sent back to the client. Upon receipt the client will execute the same priority change as the server. This way the server and client keep their priority mappings synchronised.

5.4 The `out_bit` method

`out_bit` is called to send a single bit of data to the client. Data is buffered into 32 bit integers and transmitted as integers. The `flush_bits` method flushes the integer bit buffer.

6. The client applet

The client is by far the largest component of the system. It consists of the `client.java`, `footprint.java` and `priority.java` files.
The `client` class is the applet itself and it simply creates an instance of the `client_decoder Thread`, starts it and forwards the appropriate user events to it.

6.1 Variables

Variables by the same name as those used in the encoder or server may be assumed to have the same meaning.

- `boolean do_rms` - determines whether an RMS error calculation will be performed on the image each time it is updated. This option requires the original image file to be hardwired into the code.
- `int default_port` - the port at which the server will attempt to connect to the server on. This value may be overidden by an applet parameter tag.

- int default_image_update_interval - the default interval in milliseconds at which the displayed image will be updated. This value may be overidden by an applet parameter tag.
- int default_splat_threshold - the default value of the threshold which is used to trim all footprints (see the footprint class below). This value may be overriden by an applet parameter tag.
- int requests_size - the size of the recorded requests array. After requests_size requests have been used, the counter wraps around to 0 and continues.
- int requests[][], requests_made - a circular buffer of the requests made by the client. When a request is made (one of pass_, flat_, hump_, disc_, pline_ or pgon_request) the exact data sent to the server is stored in the requests array at the position specified by requests_made. The request is not acted upon until the server replies with the request number which is the same as the position in the requests array. At that point the request is executed in the client via the execute_request method. Pass requests are executed by receiving the image data. Priority requests are executed by calling a do_*_priority method of the priority class according to the arguments stored in the requests array.
- int[] chan_lookup - a lookup table that performs the following function:

  ```
  if( x < 1000 ) x = 1000;
  if( x > 1255 ) x = 1255;
  return x;
  ```

  It is used to trim floating point values near the range [0,255] to the integer range [0,255]. The 1000 buffer at each end is necessary because the floating point intensity values can exceed the range [0,255] slightly.
- footprint[][][] feet - the footprints used to splat the image (see the footprint class below).
- int[] pixels - the image as it will appear on the screen. Entries are in the defaultRGB ColorModel (ie in ARGB form).
- float[][] coeffs - the image in floating point intensity form for each colour channel. coeffs[0], coeffs[1] and coeffs[2] correspond to the Y, I and Q channels respectively. chan_coeffs is used to point to one of these three sub-arrays during a tile pass. The floating point values are necessary to retain prescision over many small splat contributions. The pixels array is derived from the coeffs array in the draw_image method.
- int current_request - is defined as the user enters a priority request. When the request has been finished, it is transmitted to the server and added to the requests array.
- update_image ui - the instance of the update_image Thread used to periodically update the image on the screen (see below).

6.2 The run method

The run method is called to begin execution of the client_decoder thread. Firstly we send the name of the encoded file we wish to be served then we read in the image parameters. If a 0 is received for the width (the first parameter) then we know there has been a problem with that file and the thread stops.
Next the feet array is defined (see the footprint class below) and we enter the request -> reply loop. The client begins by making a request for image data (a pass_request) then we wait for the server to reply with the request number being served. If it is -1 then the server is finished and we stop. Otherwise we execute the request according to the associated entry in the requests array (see above).

6.3 The `footprint` class

Footprints are splats that are applied to the image when we receive information about the value of a particular wavelet coefficient. All the feet are derived from a single delta function footprint through wavelet derivation, copying and halving as follows.

6.3.1 The `footprint( float v )` constructor

This constructor produces a trivial delta function footprint 1x1 in size and having a single image value of v. This footprint represents the original value in the coefficients image that will be decoded to derive the other footprints.

6.3.2 The `footprint( parent, x_low, y_low )` constructor

This constructor is used to derive a larger footprint from an existing `parent` (possibly the delta footprint) using the wavelet synthesis basis. If `x_low` is `true` (`false`) then we treat the parent footprint as X (Y) values for the horizontal transform. Similarly `y_low` is for the vertical transform.

6.3.3 The `halve` method

The halve method halves the intensity of the splat image. This is used to derive a footprint for a certain bitplane from the associated footprint in the previous bitplane.

6.3.4 The `trim` method

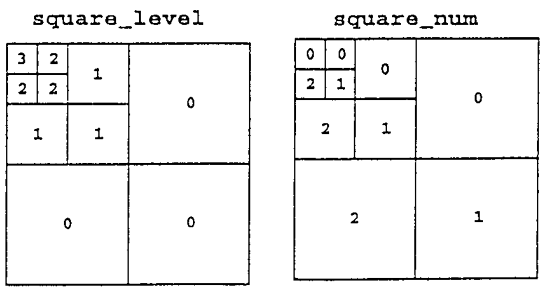
Trimming is used to reduce the size of footprints in order to reduce rendering time. The floating point threshold (which may be specified as an applet parameter tag) is applied to coefficients on the boundary of the `footprint`. The resulting `footprint` is effectively obtained by shrinking a rectangle the height of the threshold around the `footprint` until it encounters splat elements that exceed the threshold on all four sides.

6.4 The do_pass method

The do_pass method in the client differs from that of the encoder in the following ways,

- There is an extra chan argument that specifies the colour channel we are working in.
- A second set of coordinates are maintained. They are `square_level`, `square_num`, `square_x` and `square_y`. Figure 2 shows these four values in the various regions of the spatial tile. `square_level` and `square_num` are used to specify which footprint splat to use for a particular coefficient and `square_x` and `square_y` are used to specify the location of the splat.

FIGURE 2. The values of the `square_` coordinates within a spacial tile. `square_level` and `square_num` identify the footprint to splat with. `square_x` and `square_y` determine the location of the splat.



square_x

0	0	0	1	0	1	2	3
0	0	0	1	0	1	2	3
0	1	0	1	0	1	2	3
0	1	0	1	0	1	2	3
0	1	2	3	0	1	2	3
0	1	2	3	0	1	2	3
0	1	2	3	0	1	2	3
0	1	2	3	0	1	2	3

square_y

0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	1
0	0	0	0	2	2	2	2
1	1	1	1	3	3	3	3
0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3

- Since actual coefficient values are not stored in the client (only the final image intensity values are stored using the `coeffs` array) we need to keep track of the sign and significance status of all the coefficients. To do this we have two flags for each pixel, a sign flag and a refinement flag. The refinement flag gets set when a coefficient is found to be just-significant. These two flags reside in bit positions 5 and 6 in the `level` array (since the level value itself will never be large enough to encroach on these positions). To query the sign of a coefficient (`level[x][y] & 0x40`) is used. To check whether a refinement bit is expected for a coefficient,
(`level[x][y] & 0x20`) is used. Previously two boolean arrays were used to store these values but doing it this way typically saves around 500k of memory.
- When information about a coefficient is received the following action is taken. Firstly if we discover that the coefficient has just become significant we set the refinement flag in the corresponding `level` array element (see above). Similarly if we have just discovered the coefficient is positive, we set the sign flag. Finally if the implicitly stored value of the coefficient has changed, we call `update_coefficient` to reflect that change in the `coeffs` array representation of the actual image (see the `update_coefficient` method below).
- Data is input with the `in_bit` method rather than output. `in_bit` reads data from the network 32 bits at a time into an integer buffer.

6.5 The `update_coefficient` method

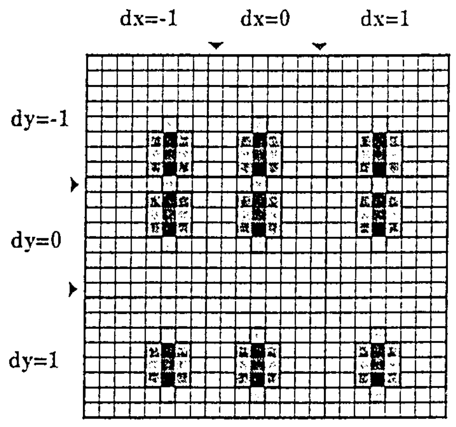
The `update_coefficient` method is responsible for applying an individual splat to the floating point representation of the final image. The footprint to use is identified by the `level` and `index` arguments which correspond to the `square_level` and `square_num` coordinates of the `do_pass` method (See figure 2). The position of the splat is identified by the `x` and `y` arguments which correpond to the `square_x` and `square_y` coordinates of the `do_pass` method. The `sign` argument specifies whether the splat will add or subtract from the image. If the coefficient causing the splat is positive we add to the image, otherwise we subtract from it. Recall that in the client the sign of each coefficient is stored in bit position 6 of the `level` array.
Inside `update_coefficient` the `chan_coeffs` array points to either `coeffs[0]`, `coeffs{1]` or

`coeffs[2]` (i.e. one of the colour channels).
The bulk of the `update_coefficient` method is broken into 9 similar parts. Each one corresponds to one of the possible reflections that the splat can undergo (See figure 3). Originally this code was contained in the following loop.

```
for( int dx=-1; dx<2; dx++ ) for( int dy=-1; dy<2; dy++ )
```

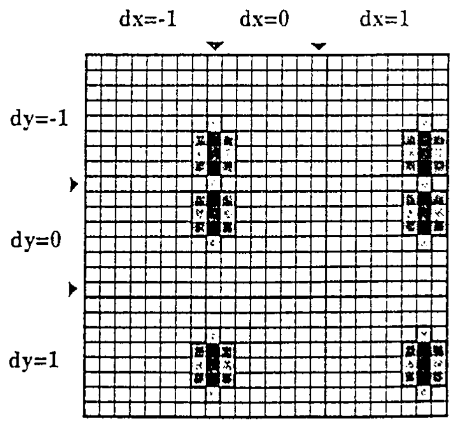
Since `update_coefficient` is critical to the performance of the client, the loop was unrolled as an optimisation..

FIGURE 3. The 9 reflected splats. The central 1/9th of the grid represents the actual image. The 9 splats correspond to reflections of the splat about the center of the first and last row and column of pixels in the central image (marked with triangles). Note that in this case the only *reflected* splat that contributes to the image is the dx,dy = (0,-1) one. The other reflections are trimmed down to 0 dimensions and are not used.



When the center of a splat lies on one of the reflecting rows or columns of pixels, it is not reflected on that row or column (See figure 4).

FIGURE 4. Omission of the dx=-1 reflections. Here the central splat lies on a reflecting column of pixels (marked with a gray triangle) and is therefore not reflected along that column.



It will be appreciated that the present invention has a number of advantages over known methods and systems of progressively transmitting an image wherein compression techniques rely on spatial tiling of the image. These include:-

- A wider range of partitioning elements for the tree of the Said and Pearlman algorithm.

- The present invention implements the code as a single pass for each bitplane rather than requiring partitioning and refinement passes as with prior art algorithms.

- Each pass of the partitioning tree iterates through each node which for the current pass has not already been found to be part of a zero tree. At each node the algorithm generates either a refinement bit or significance information. This has advantages in terms of the ordering of the coded bits.

- Furthermore, the implementation is achieved without the necessity of using lists as necessary with the Said and Pearlman algorithm and precalculates all of the significance information required by the coding passes in a single pass of the tree.

- After transmission of as little as 1-2% of an image, the user has enough information to identify regions of potential interest. The user can then click on that area and define a smooth priority map which can be communicated to the server such that the image will appear to resolve smoothly and progressively around the selected region.

- The user can redefine the priority without the server having to reformat the image representation at the transmission end or without having to resend any information.

- The refinement bits are sent in the order in which the corresponding coefficients are encountered during a single pass of the coefficients for each bitplane. This is to be contrasted with the method of Said and Pearlman where a partitioning pass and a refinement pass are used.

- When decoding each image tile the wavelet representation is not regenerated. Rather, the image components corresponding to individual significant bits in the representation are splatted directly on to the image plane. Because only a small proportion of the bits are significant this is computationally efficient and on some platforms it could take advantage of hardware to achieve even greater efficiency.

- To achieve control of spatial priority, the coefficients in the biorthogonal wavelet transform used in implementing the embedded encoding are rearranged. This explicit rearranging and separating the encoding streams allows greater flexibility than is possible with the Said and Pearlman approach.

It will of course be realised that whilst the above has been given by way of an illustrative example of this invention, all such and other modifications and variations hereto, as would be apparent to persons skilled in the art, are deemed to fall within the broad scope and ambit of this invention as is herein set forth.

The Claims defining the Invention are as follows:-

1. A method of embedded encoding of an image in which image compression techniques encode spatial tilings of the image, said method including:-
precalculating the significance and zerotree information in a single pass;
storing said significance and zerotree information in store, and
interrogating said store to establish the significance status of any tree.

2. A method of embedded encoding of an image in which image compression techniques encode spatial tilings of the image, said method including:-
ordering the coefficients in said spatial tilings whereby said tiles are defined as having the constraints (a) that all the children of a coefficient are visited before the siblings of that coefficient, and (b) that all the siblings of a coefficient are visited before any non-descendant non-siblings are visited, whereby the algorithm can be implemented without using lists in the partitioning of the tree.

3. A method of embedded encoding of an image in which image compression techniques encode spatial tilings of the image, said method including:-
transmitting significant bits, refinement bits and partitioning bits in the order in which the corresponding coefficients are encountered during a single pass of the coefficients for each bitplane.

4. A method of embedded encoding of an image in which image compression techniques encode spatial tilings of the image, said method including:-
for a given threshold treating as insignificant all components above a given scale in the tree.

5. A method of embedded encoding of an image in which image compression techniques encode spatial tilings of the image, said method including:-

splatting the image components corresponding to individual significant bits in the representation directly on to the image plane.

6. A method as claimed in any one of the preceding claims of embedded encoding of an image in which image compression techniques encode spatial tilings of the image, wherein the pseudo-code description of the embedded encoding algorithm is as set out in FIG 9.

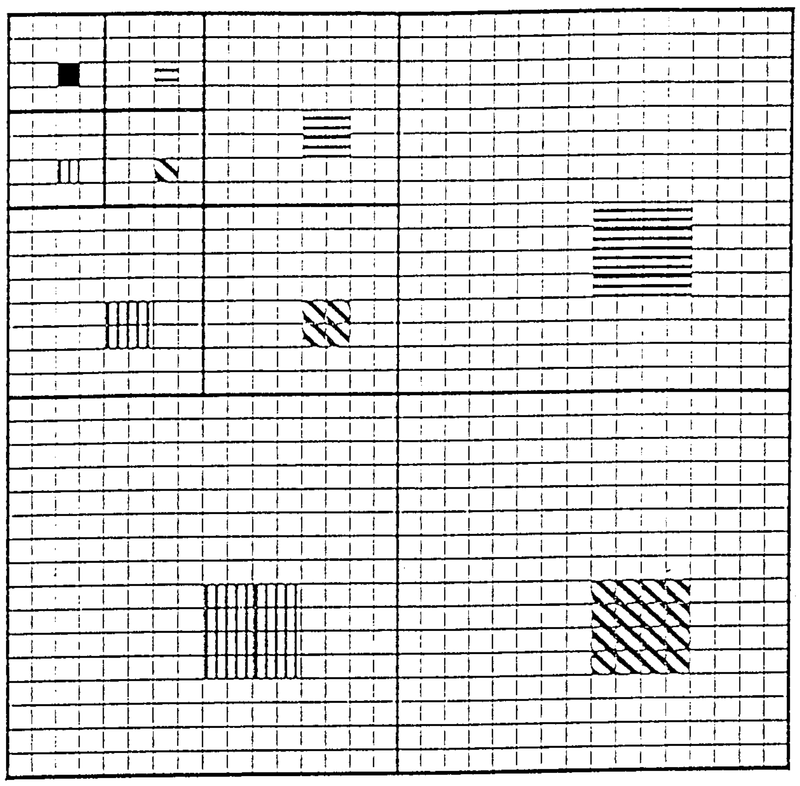


Figure 1.

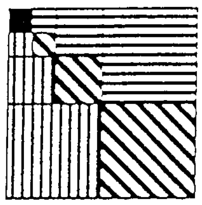


Figure 2.

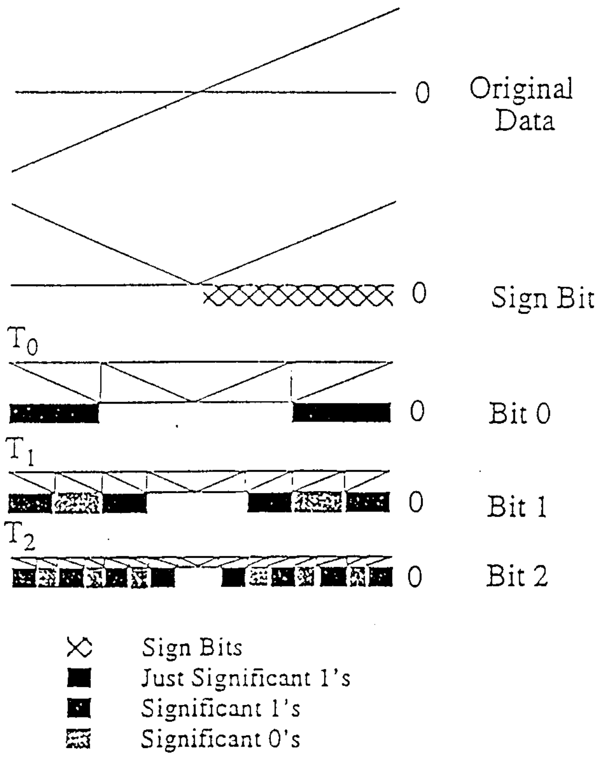


Figure 3.

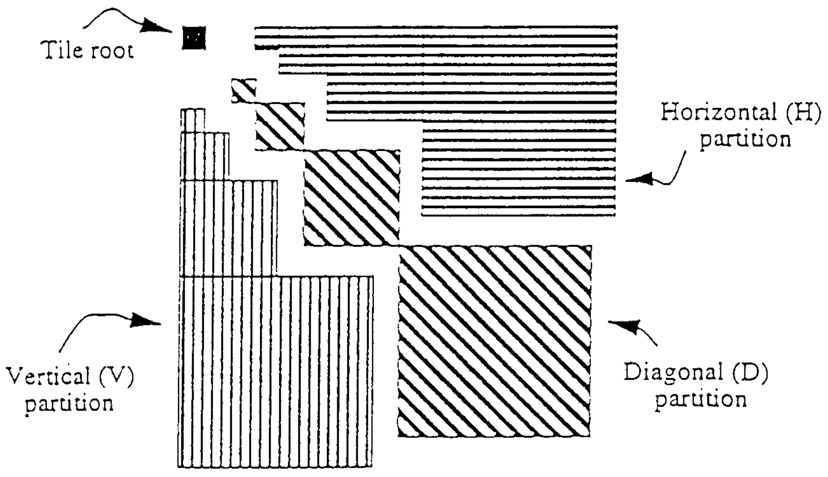


Figure 4.

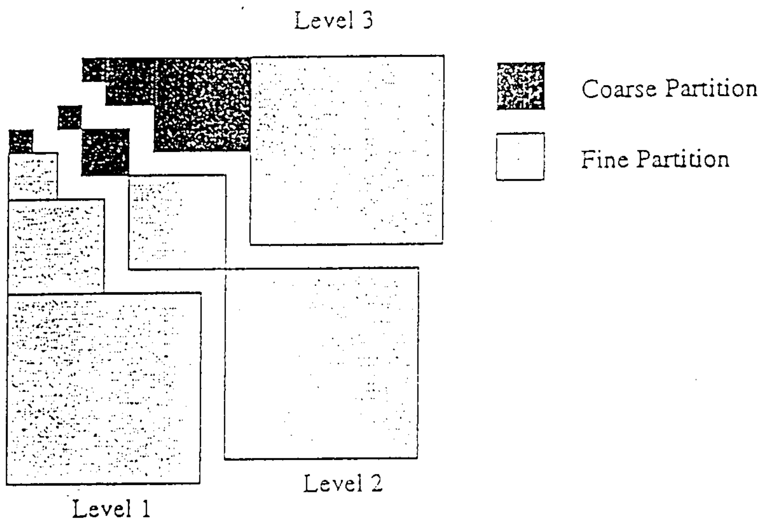


Figure 5.

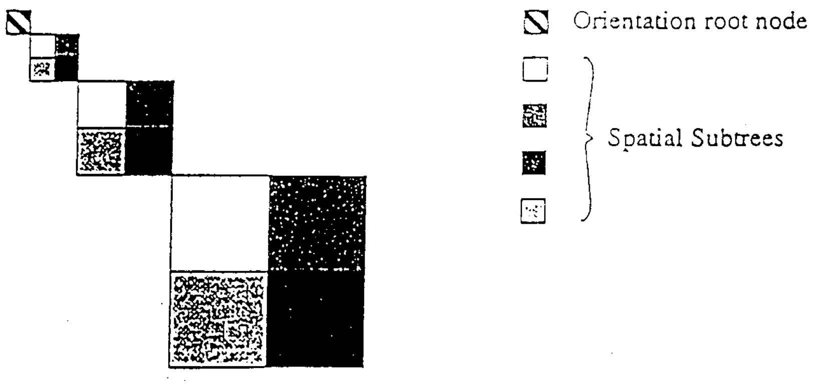


Figure 6.

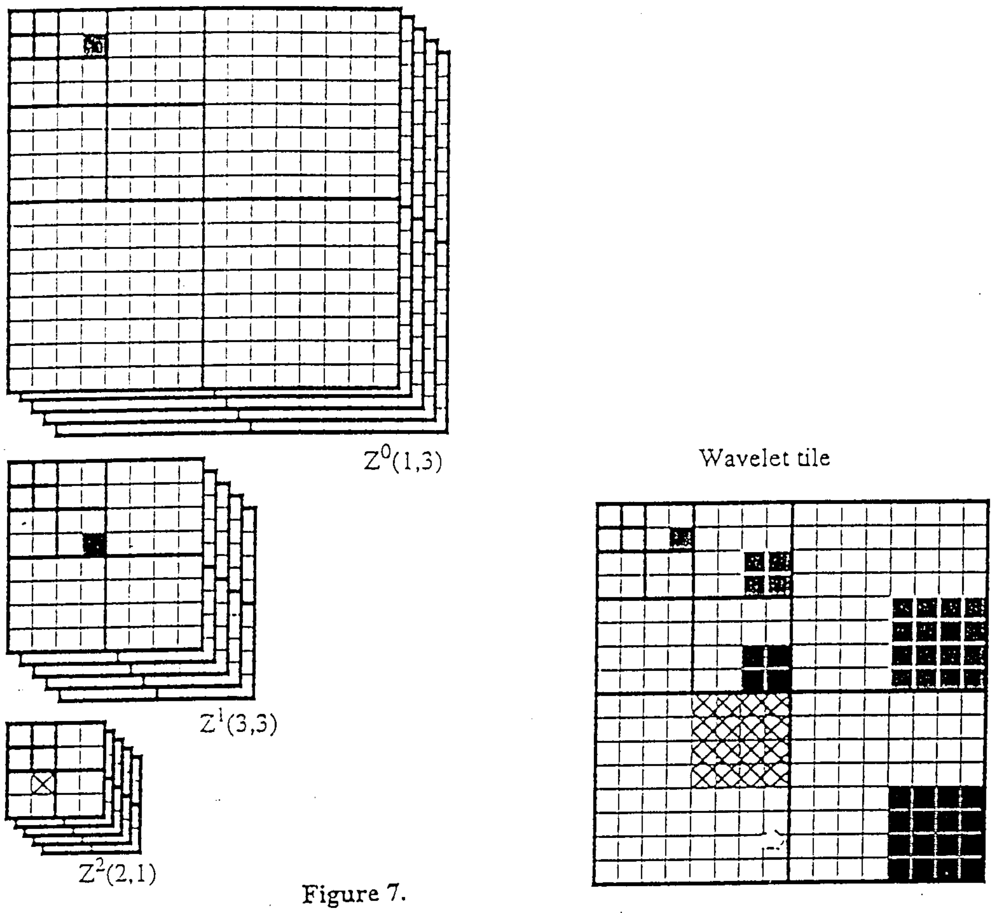


Figure 7.

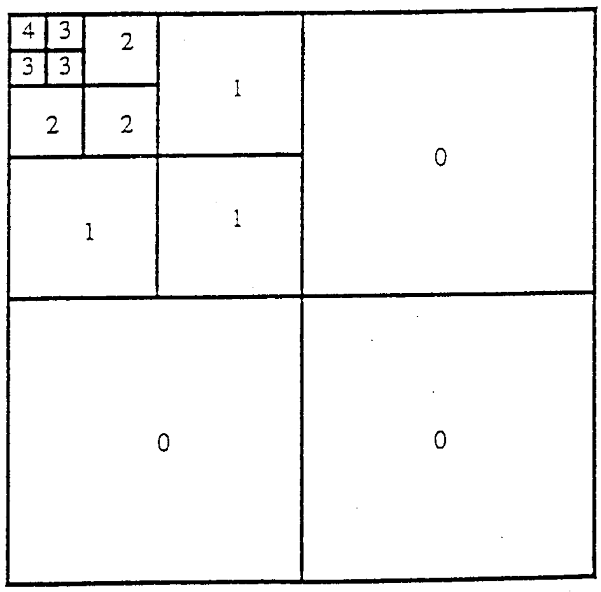


Figure 8.

Pseudo-code description of the encoding.

```
For all I, set the Coarse/Fine partition level CF(I) = $k_{min}$

Set the depth of the current tree to $d_c = d_0$

For each bitplane n = 0 to 16
   Initialise the coefficient address I = (i,j) = (0,0)
   Initialise the scale s = 0.
   Set the sibling number SN(s)=1, for all scales s = 0 to $d_0$
   While s >= 0,
      While (SN(s) is in the valid range for scale s),
         While(CF(I) <= $k_e$ and CF(I)<= $k_{max}$ )
            %Perform CF Partition of the tree
            output ($Z_n^{CF(I)}$(I))
            if($Z_n^{CF(I)}$(I))
               CF(I)=CF(I)+1
            else
               $d_c$ = CF(I)-1;
            endif
         end

         %Handle the root coefficient
         if(w(I) previously significant)
            output bit n of the magnitude of w(I)
         else
            output $J_n$(I)
            if($J_n$(I))
               output sgn(I)
            endif
         end

         if($k_e$ > 1)
            %Go down the tree to first child
            s = s+1
            I = $C^1_1$(I)
            $d_c = d_c-1$
            $d_s(s) = d_c$
         else
            %Go to next sibling
            SN(s) = SN(s)+1
            I = N(I)
            $d_c = d_s(s)$
         end
      end
      %go up the tree to next sibling of parent
      s = s-1
      SN(s) = SN(s)+1
      $d_c = d_s(s)$
      I = N(P(I))
   end
end
```

FIG 9

INTERNATIONAL SEARCH REPORT

International Application No.
PCT/AU 97/00725

A. CLASSIFICATION OF SUBJECT MATTER

Int Cl^{6}: G06T 9/40; H04N 1/64

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC^{6} as above plus (G06F and H03M)

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
AU: IPC as above

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
WPAT, INSPEC (imag:, encod:, decod:, cod:, tree)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 5321776 (SHAPIRO) 14 June 1994 column 1-4	4
A	"Supra-Threshold Perceptual Image Coding" (Pappas et al) Proc Int Conf on Image Proc volume 1 pages 237-240, 16-19 September 1996 whole document	1
A	US 5412741 (SHAPIRO) 2 May 1995 whole document	1

[X] Further documents are listed in the continuation of Box C

[X] See patent family annex

*	Special categories of cited documents:
"A"	document defining the general state of the art which is not considered to be of particular relevance
"E"	earlier document but published on or after the international filing date
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O"	document referring to an oral disclosure, use, exhibition or other means
"P"	document published prior to the international filing date but later than the priority date claimed
"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"&"	document member of the same patent family

Date of the actual completion of the international search 19 December 1997	Date of mailing of the international search report 13 JAN 1998
Name and mailing address of the ISA/AU AUSTRALIAN INDUSTRIAL PROPERTY ORGANISATION PO BOX 200 WODEN ACT 2606 AUSTRALIA Facsimile No.: (02) 6285 3929	Authorized officer **DALE SIVER** Telephone No.: (02) 6283 2196

INTERNATIONAL SEARCH REPORT	.ıternational Application No. **PCT/AU 97/00725**

C (Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	"Image Coding using the Embedded Zerotree Wavelet" (SHAPIRO) Proceedings of SPIE, Volume 2034 Mathematical Imaging pages 180-193 whole document	1
A	WO 95/15530 (POLAROID CORP) 8 June 1995 whole document	4
A	EP 466475 A2 (FUJITSU) 15 January 1992	4

INTERNATIONAL SEARCH REPORT
Information on patent family members

International Application No.
PCT/AU 97/00725

This Annex lists the known "A" publication level patent family members relating to the patent documents cited in the above-mentioned international search report. The Australian Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

Patent Document Cited in Search Report		Patent Family Member					
US	5321776	JP	7504306	WO	93/17524		
US	5412741	EP	680643	JP	8506226	WO	94/17492
WO	95/15530	NONE					
EP	466475	CA	2046544	JP	4070059	KR	9411607
		JP	4070061				

END OF ANNEX