Rensselaer

CNGV

Wavelet and Zerotree Coding

©William A. Pearlman

Center for Next Generation Video

Rensselaer Polytechnic Institute

Troy, NY, USA

pearlman@rpi.edu

http://www.cipr.rpi.edu/staff/pearlman.html

Rensselaer

CNGV

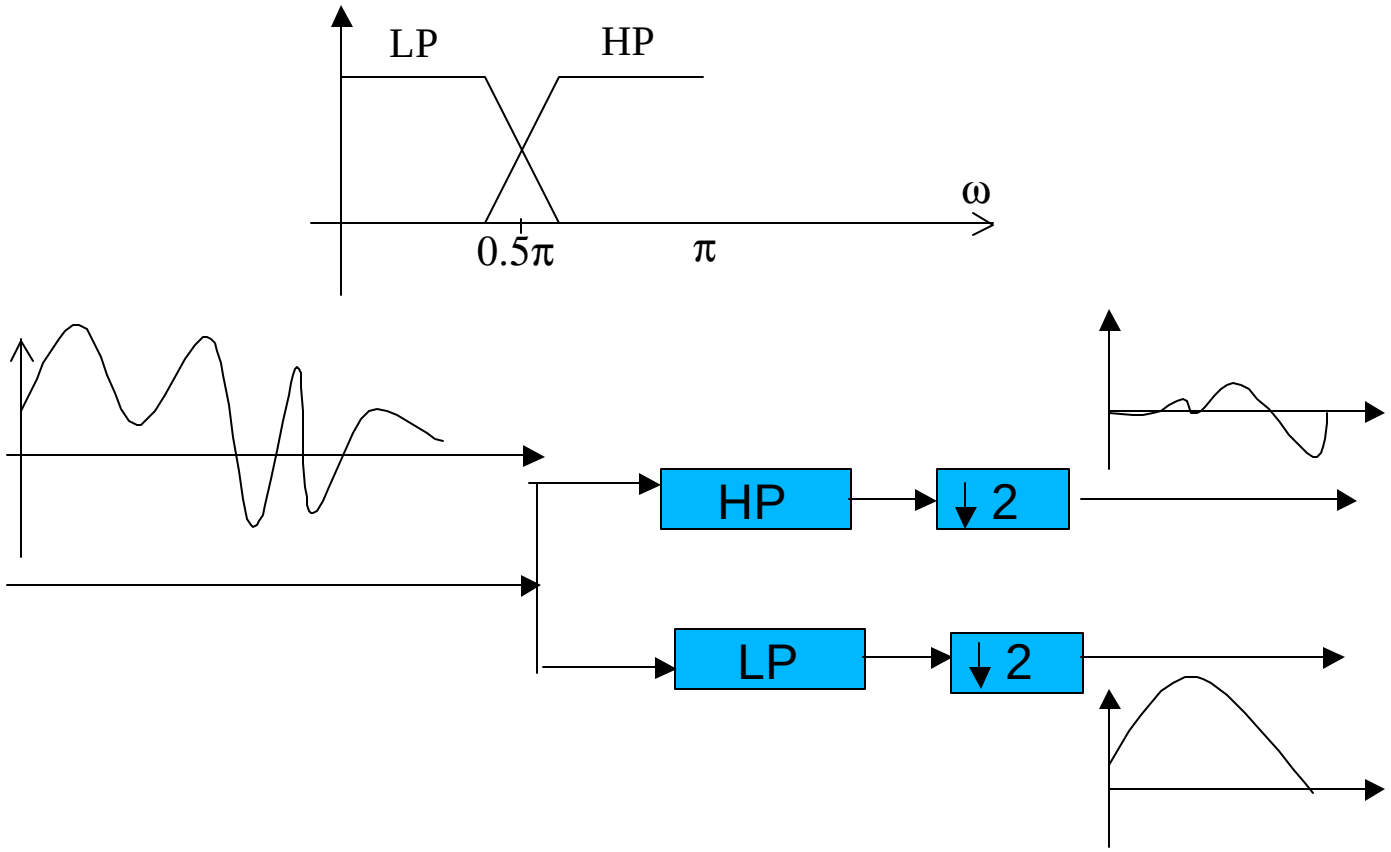
Outline

- Wavelet transforms
- Ordered bit plane transmission
- Zerotree coding principles
  - Said and Pearlman, ISCAS ’93, pp. 279-282
  - Shapiro, IEEE Trans. SP, Dec. 1993
- SPIHT coding
  - Said and Pearlman, IEEE Trans. CSVT, pp. 243-250, June 1996
- Image Reconstructions

Rensselaer

CNGV

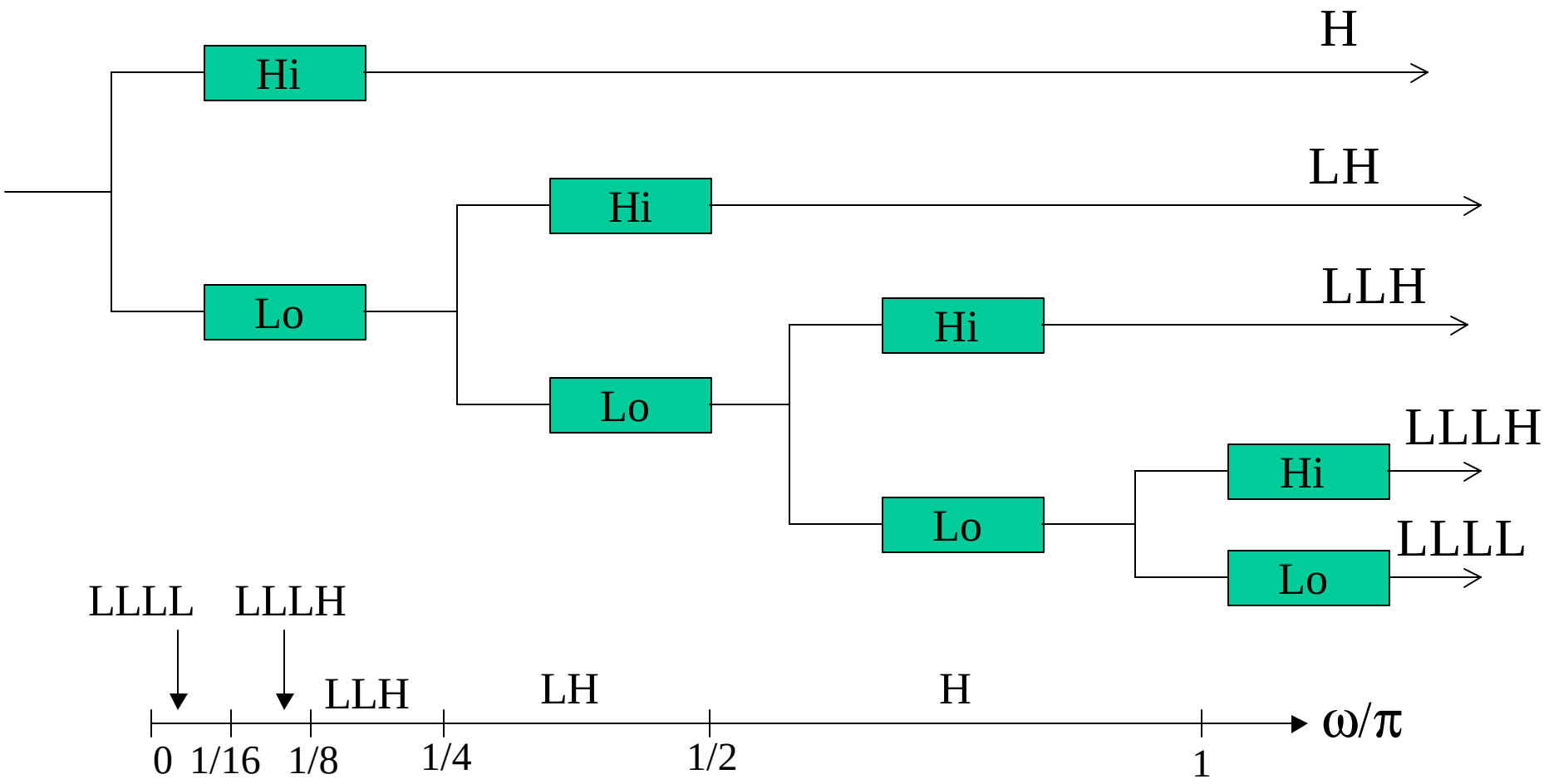
Wavelet Filtering



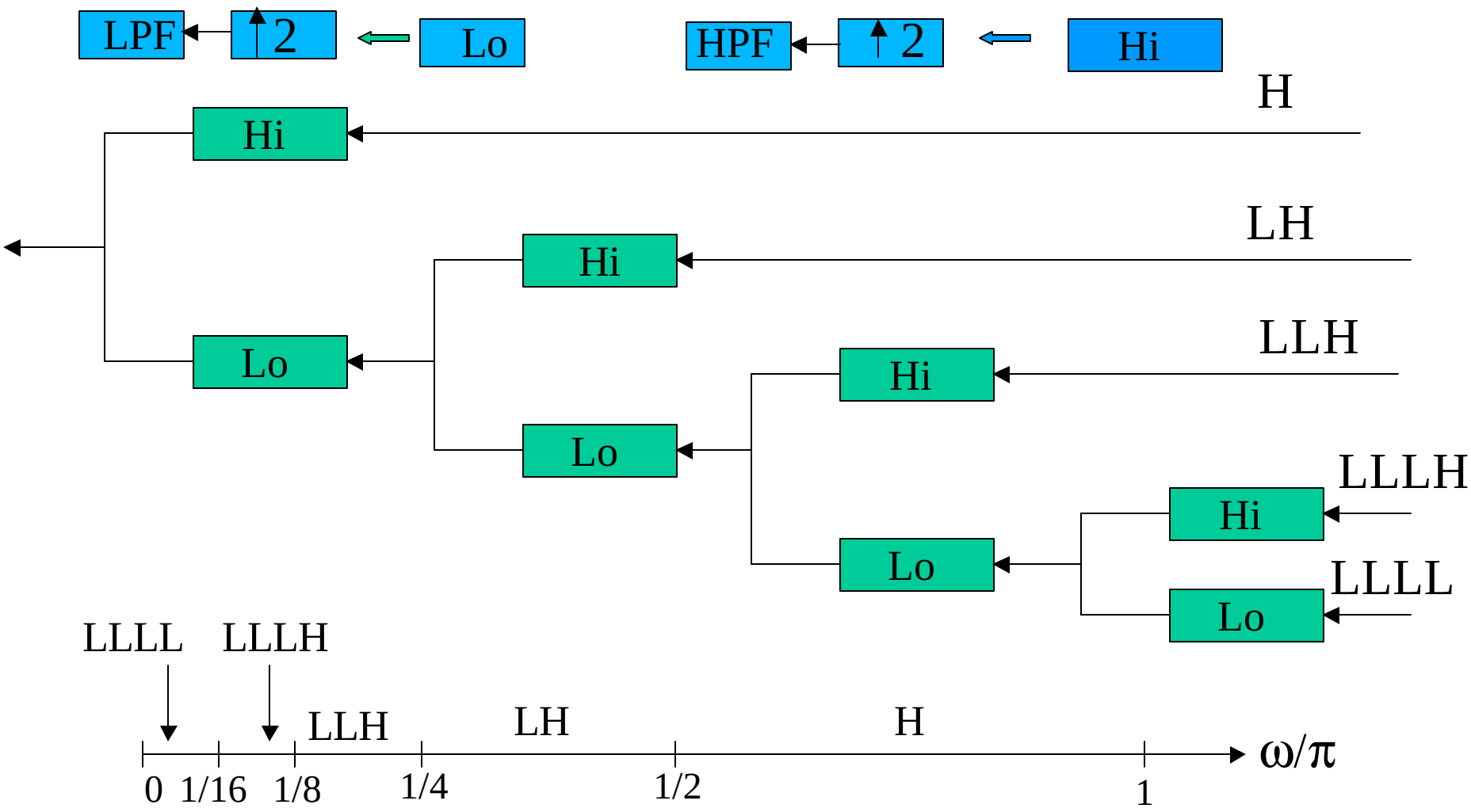
Rensselaer

CNGV

Recursive Low Pass Filtering

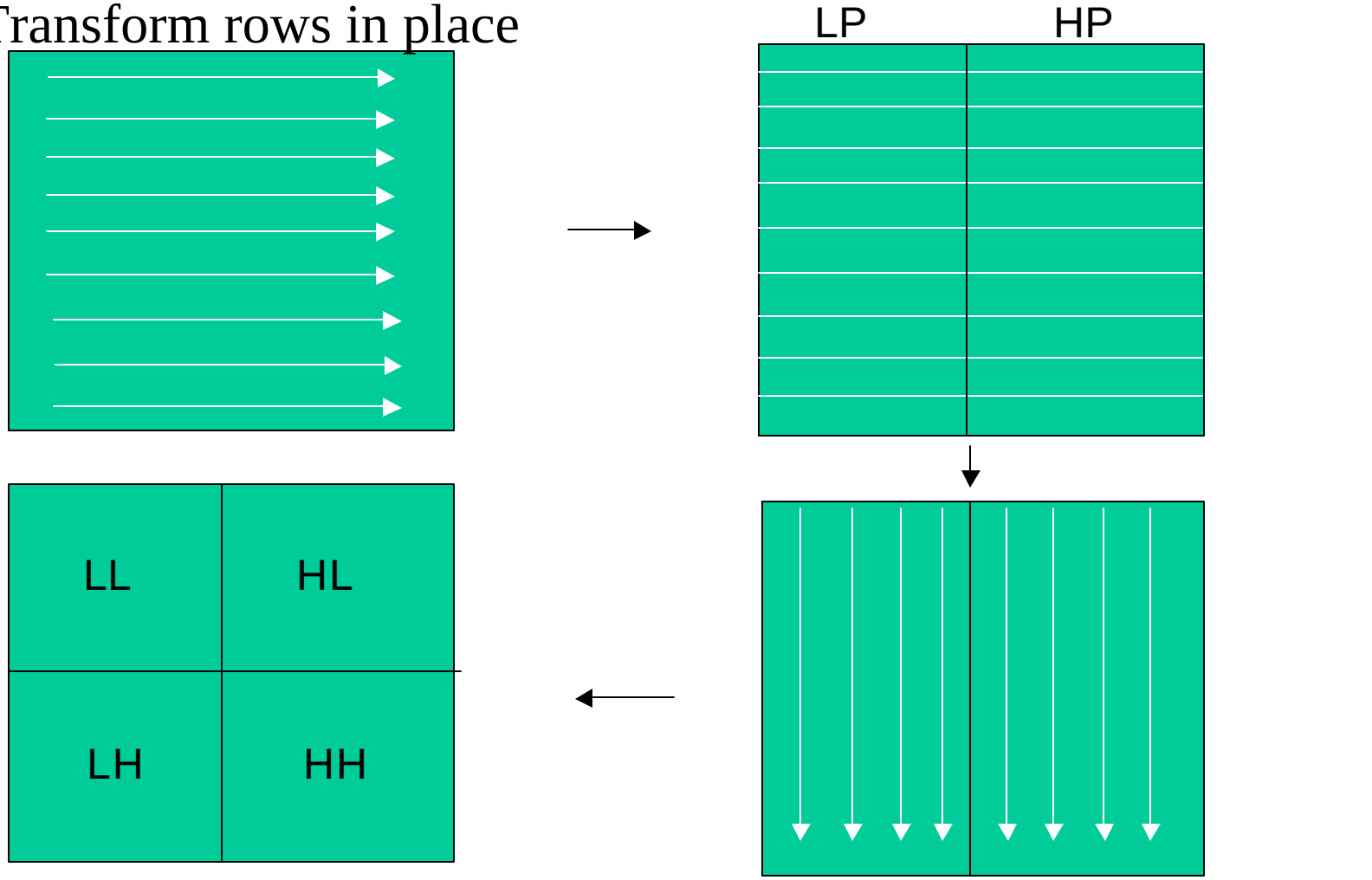


Synthesis Recursive Filtering



2D Transform

Transform rows in place

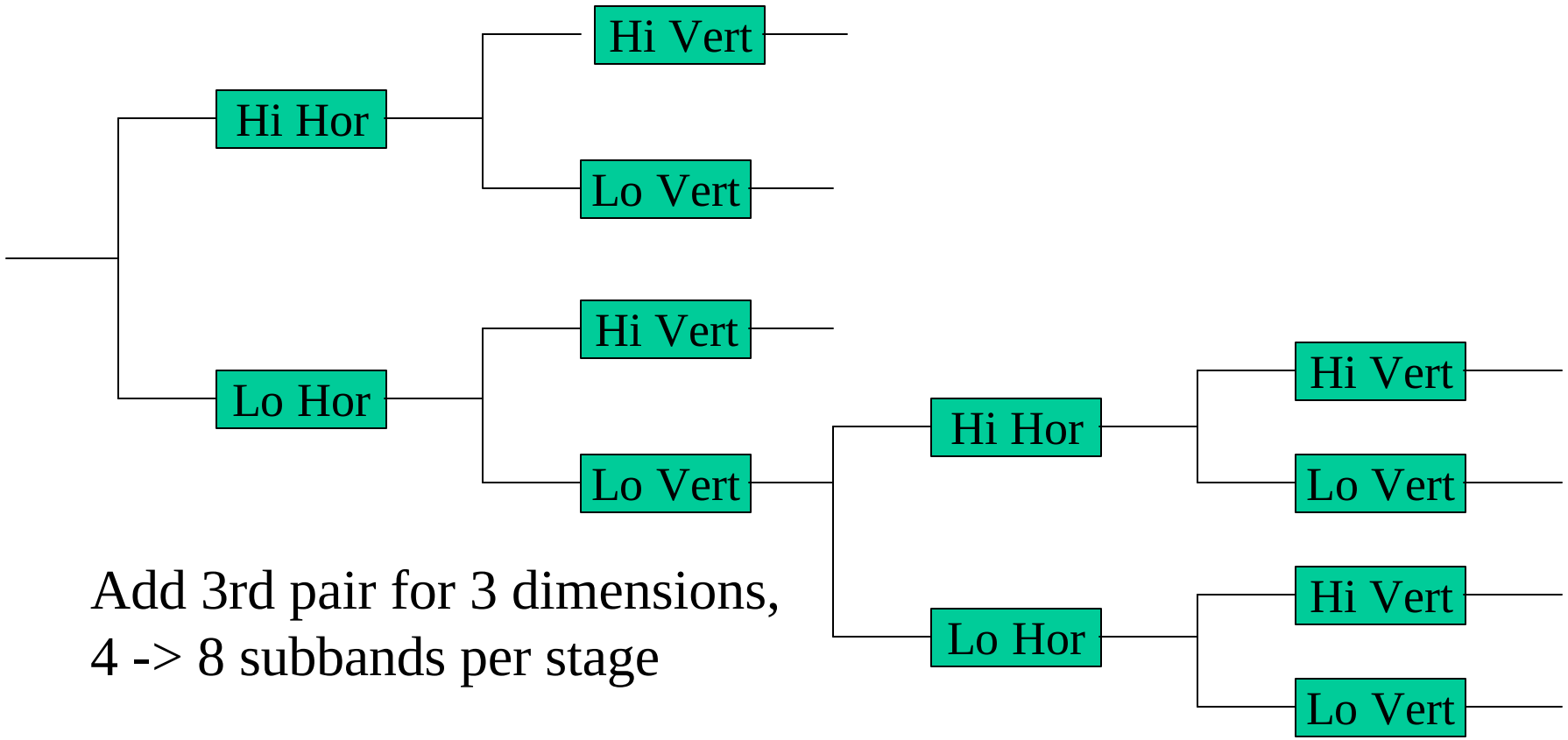


Transform columns in place

Rensselaer

CNGV

Recursive Hi/Lo 2-Stage Filtering

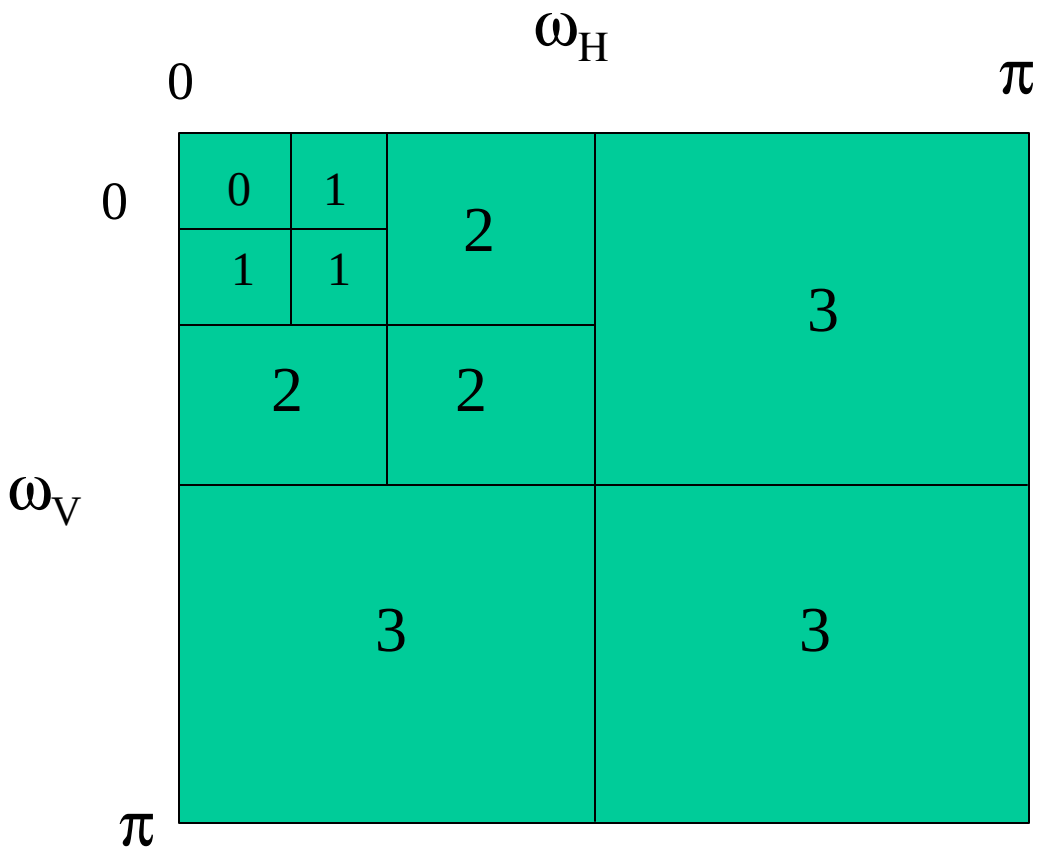


Add 3rd pair for 3 dimensions,
4 -> 8 subbands per stage

Rensselaer

CNGV

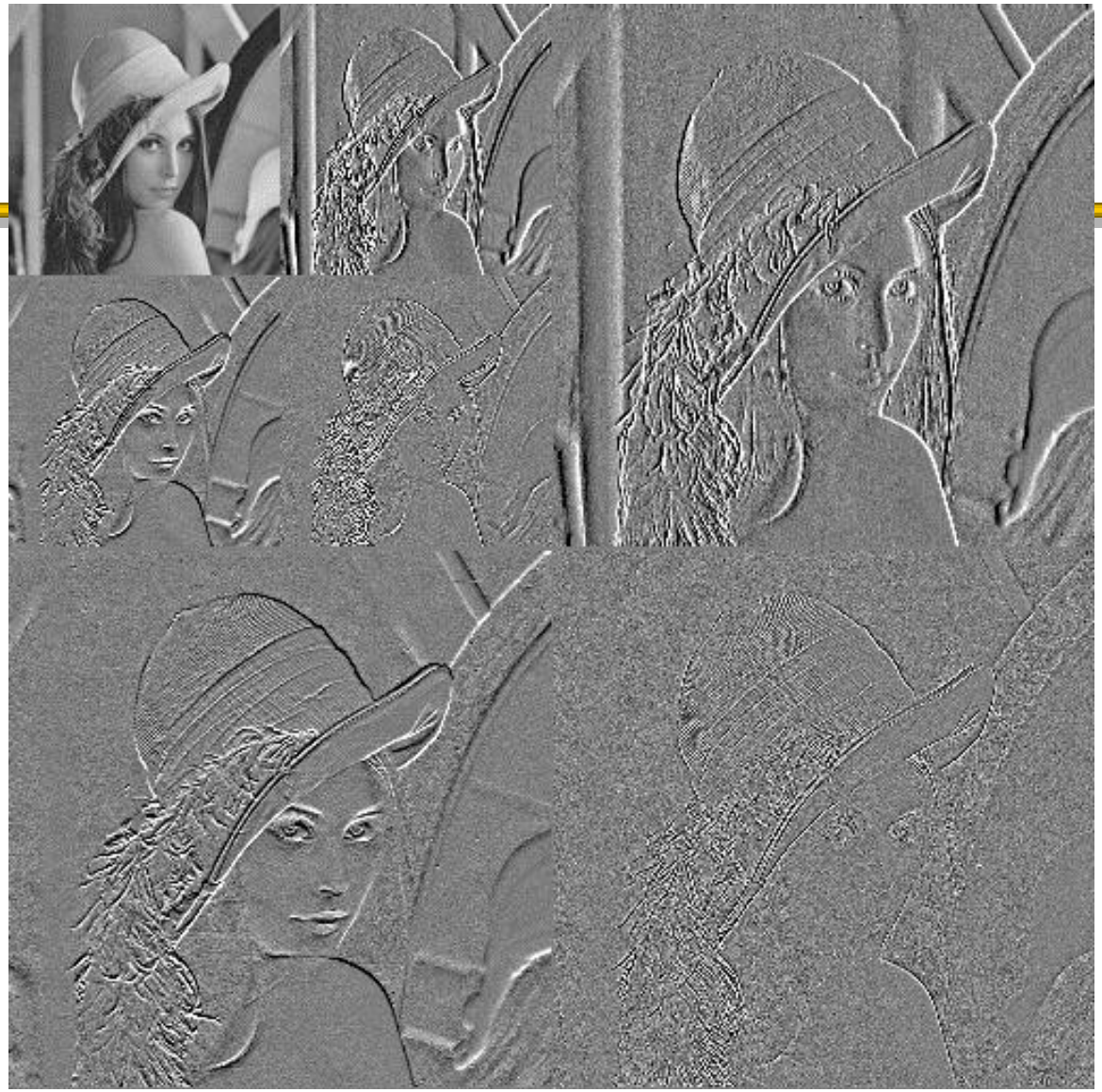
Multi-resolution (Wavelet) Transform



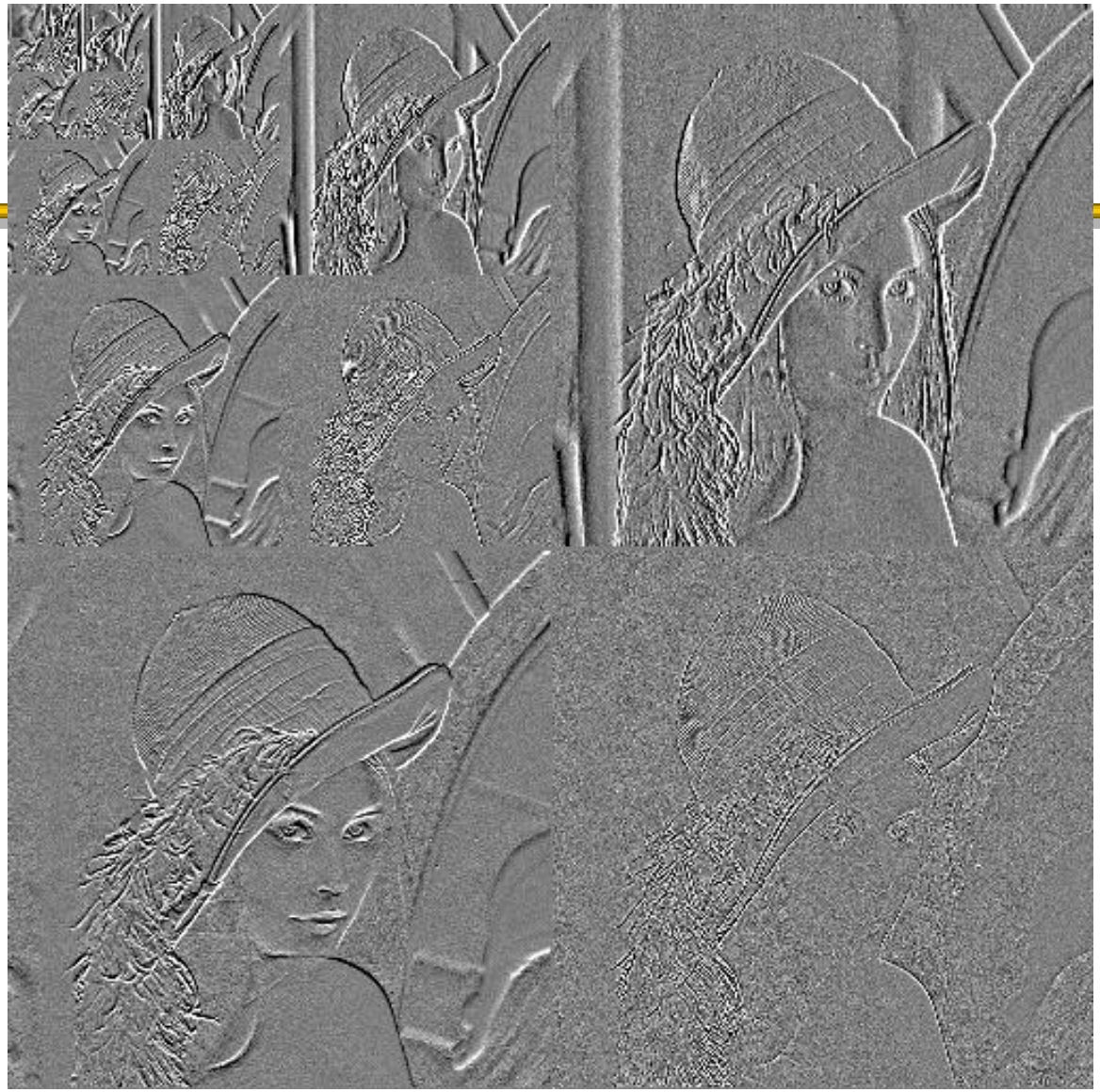
0 -> 1/8 res.
+
1,1,1 -> 1/4 res.
+
2,2,2 -> 1/2 res.
+
3,3,3 -> full res.

1st stage

2nd stage



5th stage



Haar Transform

- **Definition for one-dimensional array**

$$p_i^{k+1} = \frac{\sqrt{2}}{2}\left(p_{2i}^{k} + p_{2i+1}^{k}\right) \qquad h_i^{k+1} = \frac{\sqrt{2}}{2}\left(p_{2i}^{k} - p_{2i+1}^{k}\right)$$

- **Recursive computation**

$$\begin{array}{cccccccc}
p_0^0 & p_1^0 & p_2^0 & p_3^0 & p_4^0 & p_5^0 & p_6^0 & p_7^0 \\
p_0^1 & p_1^1 & p_2^1 & p_3^1 & h_0^1 & h_1^1 & h_2^1 & h_3^1 \\
p_0^2 & p_1^2 & h_0^2 & h_1^2 & h_0^1 & h_1^1 & h_2^1 & h_3^1 \\
p_0^3 & h_0^3 & h_0^2 & h_1^2 & h_0^1 & h_1^1 & h_2^1 & h_3^1
\end{array}$$

Overlapping Kernels

- **One formula for overlapping multiresolution transform**

$$p_i^{k+1} = \frac{\sqrt{2}}{8}\left(-p_{2i-2}^{k} + 2p_{2i-1}^{k} + 6p_{2i}^{k} + 2p_{2i+1}^{k} - p_{2i+2}^{k}\right)$$

$$h_i^{k+1} = \frac{\sqrt{2}}{8}\left(-2p_{2i}^{k} + 4p_{2i+1}^{k} - 2p_{2i+2}^{k}\right)$$

- **Inverse transform**

$$p_{2i}^{k} = \frac{\sqrt{2}}{8}\left(-2h_{i-1}^{k+1} + 4p_i^{k+1} - 2h_i^{k+1}\right)$$

$$p_{2i+1}^{k} = \frac{\sqrt{2}}{8}\left(-h_{i-1}^{k+1} + 2p_i^{k+1} + 6h_i^{k+1} + 2p_{i+1}^{k+1} - h_{i+1}^{k+1}\right)$$

Integer Wavelet Transform Filters:

I(13, 7):

Low = (-1, 0, 18, -16, -63, 144, 348, 144, -63, -16, 18, 0, -1)/2^9

High= (1, 0, -9, 16, -9, 0, 1)/2^4

I(9, 7):

Low = (1, 0, -8, 16, 46, 16, -8, 0, 1)/2^6

High= (1, 0, -9, 16, -9, 0, 1)/2^4

I(9, 3):

Low = (3, -6, -16, 38, 90, 38, -16, -6, 3)/2^7

High= (-1, 2, -1)/2

I(5, 3):

Low = (-1, 2, 6, 2, -1)/2^3

High= (-1, 2, -1)/2

I(13, 11):

Low = (-3, 0, 22, 0, -125, 256, 724, 256, -125, 0, 22, 0, -3)/2^10

High= (-3, 0, 25, 0, -150, 256, -150, 0, 25, 0, -3)/2^8

I(5, 11):

Low = (-1, 2, 6, 2, -1)/2^3

High= (-1, 2, 7, 0, -70, 124, -70, 0, 7, 2, -1)/2^7

I(2, 6):

Low = (1, 1)/2

High= (1, 1, -8, 8, -1, -1)/2^3

I(2, 10):

Low = (1, 1)/2

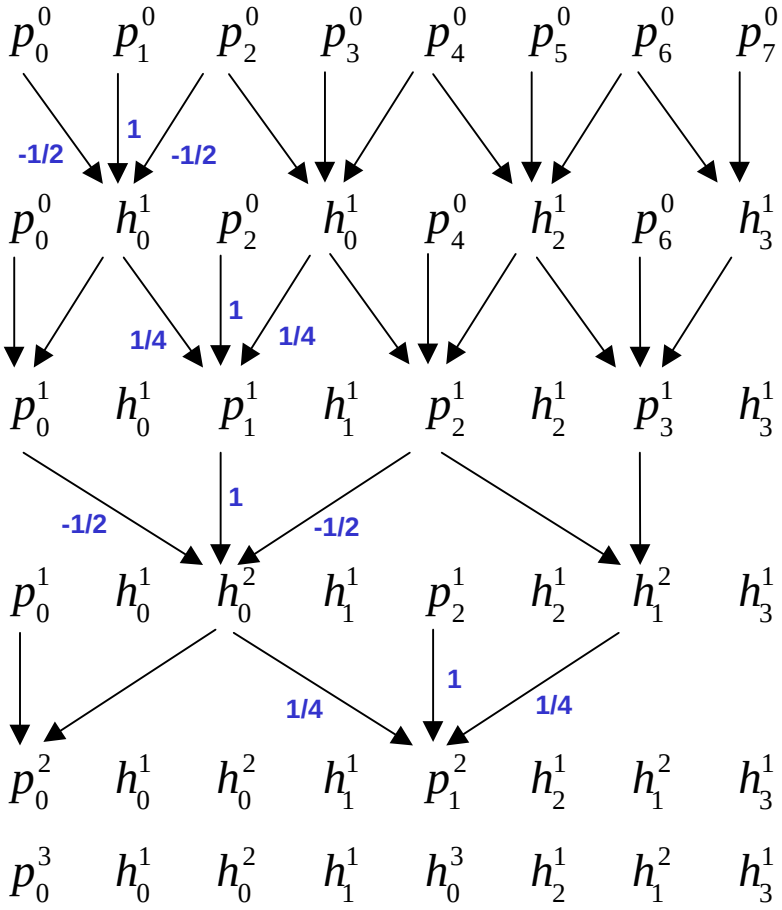
High= (-3, -3, 22, 22, -128, 128, -22, -22, 3, 3)/2^7

I(2, 2):

Low = (1, 1)/2

High= (-1, 1)

The “Lifting” Technique



add neighbors, divide by 2, subtract

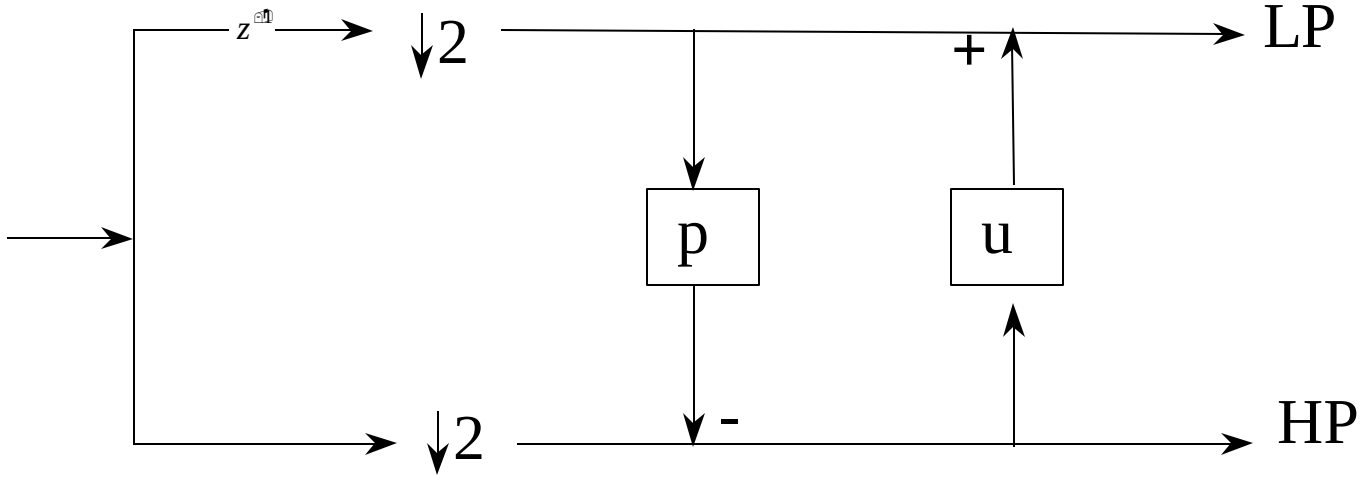
add neighbors, divide by 4, add

add neighbors, divide by 2, subtract

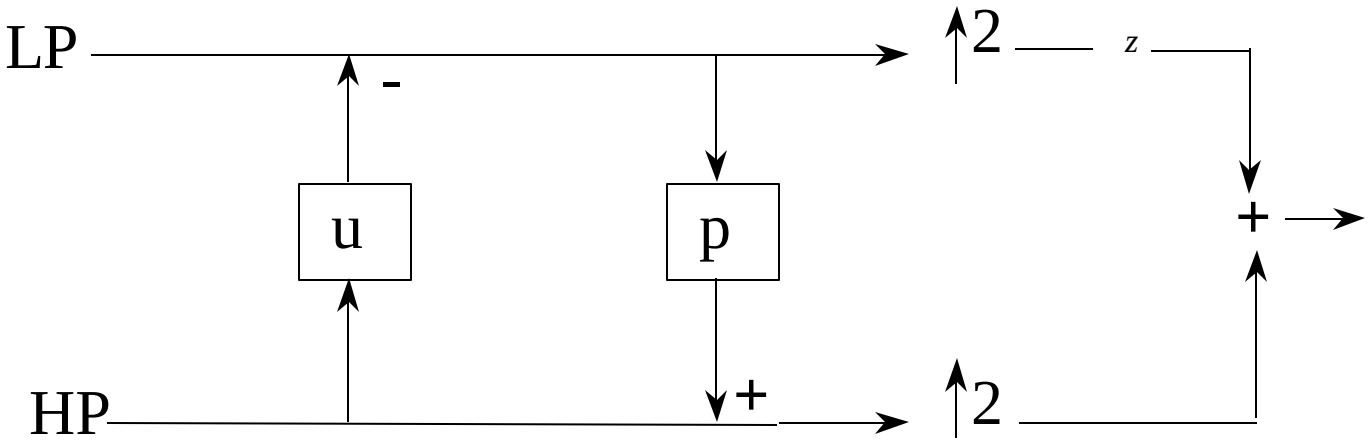
add neighbors, divide by 4, add

use bit reversal to reorder

Lifting Scheme 1



DWT



Inverse DWT

Filter Coefficients:

I(2, 2):
p = 1,
u = 1/2
I(5, 3):
p = (1, 1)/2,
u = (1, 1)/4

I(9, 7):
p = (-1, 9, 9, -1)/16,
u = (1, 1)/4

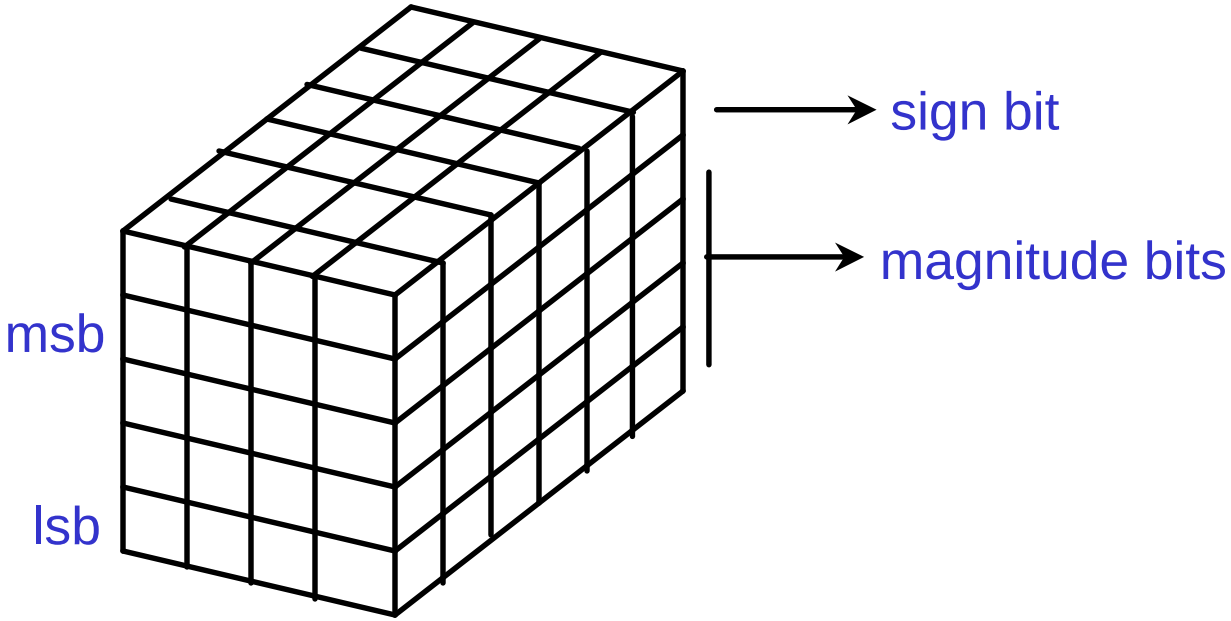
I(9, 3):
p = (1, 1)/2,
u = (-3, 19, 19, -3)/64

I(13, 11):
p = (3, -25, 75, 75, -25, 3)/256,
u = (1, 1)/4

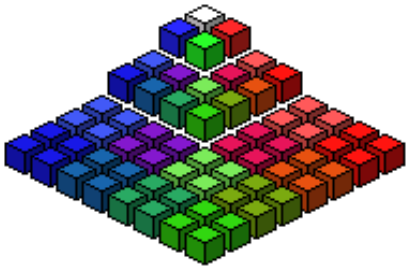
I(13, 7):
p = (-1, 9, 9, -1)/16,
u = (-1, 9, 9, -1)/32

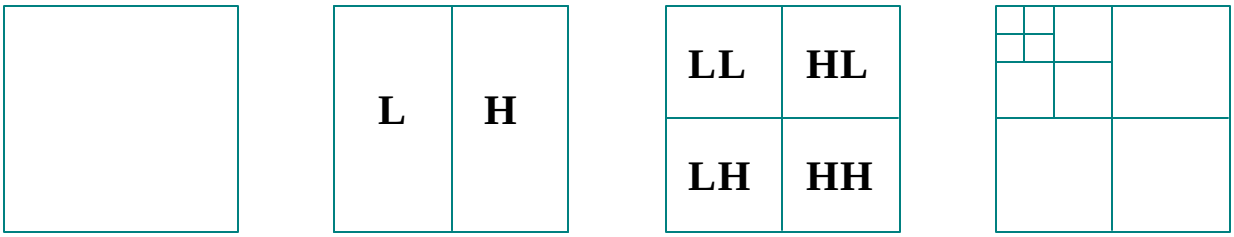
Bit-plane Coding

- **Progressive coding of wavelet coefficients**



Multiresolution Pyramid



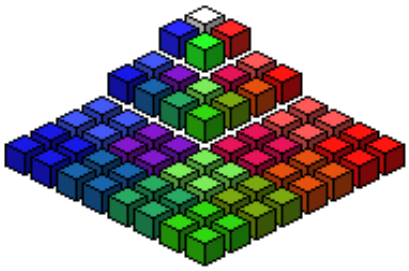


original image | rows transformed | columns transformed | pyramid structure

- Original image: $p_{i,j}$ where (i,j) is the pixel coordinate
- Unitary image transformation: $\mathbf{c} = \Omega(\mathbf{p})$
- Distortion measure: mean squared-error (MSE)

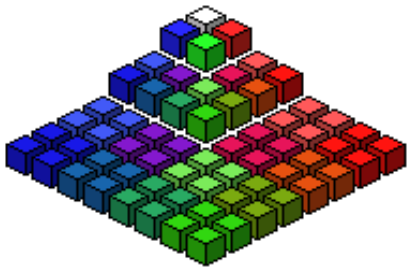
$$D_{mse}(\mathbf{p}-\hat{\mathbf{p}}) = \frac{1}{N}\|\mathbf{p}-\hat{\mathbf{p}}\|^2 = D_{mse}(\mathbf{c}-\hat{\mathbf{c}})$$

Progressive Image Transmission



- Basic scheme & objective
  1. Decoder sets reconstruction $\hat{\mathbf{c}} = \mathbf{0}$
  2. After updating reconstruction with the first bits we want $\hat{\mathbf{p}} = \Omega^{-1}(\hat{\mathbf{c}})$ to have the smallest *achievable* distortion
- Ranking: most important data go first
- Simple scheme for MSE distortion
  1. elements of $\mathbf{c}$ with largest *magnitude* first
  2. most significant *bits* of the magnitude first

Bit-Plane Transmission

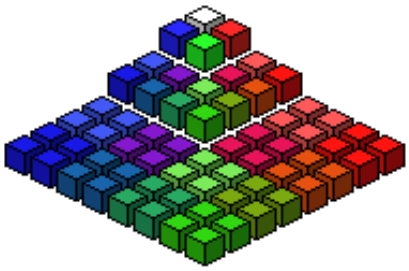


- Transmission the magnitude-sorted coefficients

	sign	s	s	s	s	s	s	s	s	s	s	s	s	s
msb	5	1	1	0	0	0	0	0	0	0	0	0	0	0
	4	®	®	1	1	0	0	0	0	0	0	0	0	0
	3	®	®	®	®	1	1	1	1	0	0	0	0	0
	2	®	®	®	®	®	®	®	®	1	1	1	1	1
	1	®	®	®	®	®	®	®	®	®	®	®	®	®
lsb	0	®	®	®	®	®	®	®	®	®	®	®	®	®

- Quantization performance
  - ready for embedded coding
  - ordering makes uniform & memoryless quantization *very* effective
  - efficient coding of the ordering data is needed

Algorithm I



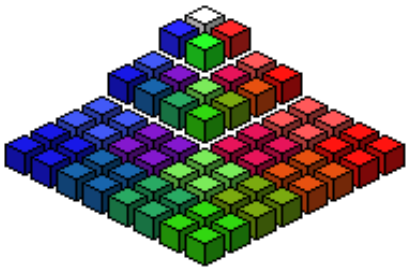
1. *Output* $n = \left\lfloor \log_2 \left(\max_{i,j} \left\{ \left| c_{i,j} \right| \right\} \right) \right\rfloor$

2. *Sorting pass: output* v_n*, followed by the pixel coordinates and signs of all* v_n *coefficients such that* $2^n \leq |c_{i,j}| < 2^{n+1}$

3. *Refinement pass: output the n-th most significant bits of all coefficients with* $|c_{i,j}| \geq 2^{n+1}$ *(i.e., those that had their coordinates transmitted in previous sorting passes), in the same order used to send the coordinates.*

4. *Decrement* n *by 1 and go to step 2.*

Coding the Ordering Data



- Sorting Objective:

  progressive selection of the coefficients such that

  $$|c_{i,j}| \geq 2^n, \quad n = n_0, n_0 - 1, n_0 - 2, \ldots$$

- Implicit Transmission
  - encoder and decoder use the same sorting algorithm
  - the decoder receives all comparison results (branching) and duplicates the encoder´s execution path
  - ordering is recovered from execution path

Rensselaer

CNGV

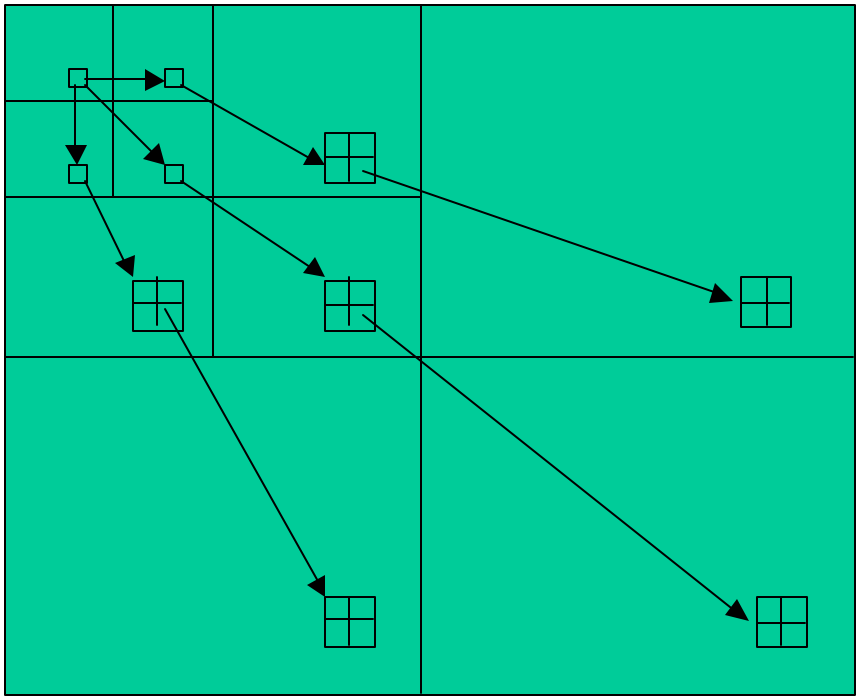
Tree Structure

3 level dyadic subband transform shown
Arrows depict parent-child relationships in the tree structure
Coefficients are grouped to exploit magnitude dependence
Each subband coefficient has four children (except leaves)

Each coefficient is denoted by it coordinates (i,j)

Set Types

(i,j): Single coefficient
O(i,j): Children of (i,j)
D(i,j): Descendants of (i,j)
T(i,j): {(i,j)} U *D*(i,j)
H : indices of all tree roots



Rensselaer

CNGV

Partitioning Rules

1. Initial partition *T*(i,j) for all (i,j) in *H*.
2. If *T*(i,j) significant, it is partitioned into sets (i,j) and *D*(i,j)
3. If *D*(i,j) is significant, it is partitioned into sets *T*(k,l), for all (k,l) in *O*(i,j).

- Map (i,j) to n = 0,1,2, ..., N-1 (notational convenience)

Rensselaer

CNGV

Ordered Lists

- Dominant List:
  - Node index *n*,
  - State of node *n* relative to threshold T, S(*n*,T)
    - State used to convey significance of node *{n}* and *D(n)* and conveys ordering information
- Subordinate list
  - Contains refinement bits of significant coefficients

Rensselaer

CNGV

Significance States

Operate through significance decisions for trees of wavelet transform coefficients. Define significance function

$$j(n) = \begin{cases} \left(\max_{m \in D(n)} |c_n|\right), & D(n) \neq f \\ \infty, & \text{otherwise.} \end{cases}$$

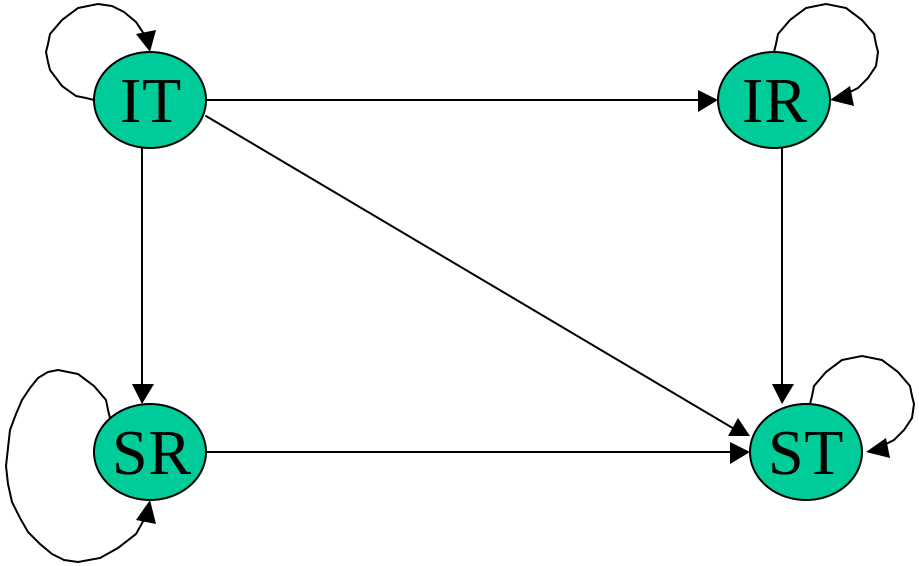
Given a threshold T >0, the state S(n,T) may have four different values:

IT (Insignificant Tree) : $|c_n| < T$, $j(n) < T$
IR (Insignificant Root): $|c_n| < T$, $j(n) > T$
SR (Significant Root) : $|c_n| > T$, $j(n) < T$
ST (Significant Tree) : $|c_n| > T$, $j(n) > T$

Rensselaer

CNGV

State Transitions



1 bit for transitions from IR and SR, 2 bits from IT

Rensselaer

CNGV

EZW States

EZW States	Description	IEZW States
ZTR	zerotree root	IT
ISZ	isolated zero	IR
POS	significant, positive root	
NEG	significant, negative root	

States, not state transitions, coded in EZW

--2 bits per state

Rensselaer

CNGV

Algorithm II (IEZW)

1. Define initial threshold: set $k_0 = \lfloor \log_2(\max_n |c_n|) \rfloor$ $T = 2^{k_0}$, and output k_0 to the decoder
2. Initialize index lists: the subordinate list is set as empty; dominant list contains n in H
3. Dominant Pass: for each entry n in the dominant list do:

Rensselaer

CNGV

Algorithm II, Step 3.

a) save old state S_{old}= S(*n*,2T) and find the new state S_{new} = S(*n*,T) ;

b) Output the code of the state transition S_{old}-> S_{new};

c) if $S_{old} \neq$ SR and $S_{new} \neq$ IR, then add index *n* to the subordinate list and output sign of c_n;

d) If $S_{old} \neq$ IR and $S_{new} \neq$ SR, add indices of *O(n)* to the end of the dominant list;

e) If S_{new}= ST, remove index *n* from dominant list;

Rensselaer

CNGV

Algorithm II (cont.)

4. Subordinate Pass: for each entry *n* in the subordinate list output the *k*-th most significant bit of $|c_n|$;
5. Threshold update: decrement *k*, set T = T/2, and go to Step 3.

Note ordering information implicitly recovered in Step 3.c

Rensselaer

CNGV

SPIHT Tree Structure

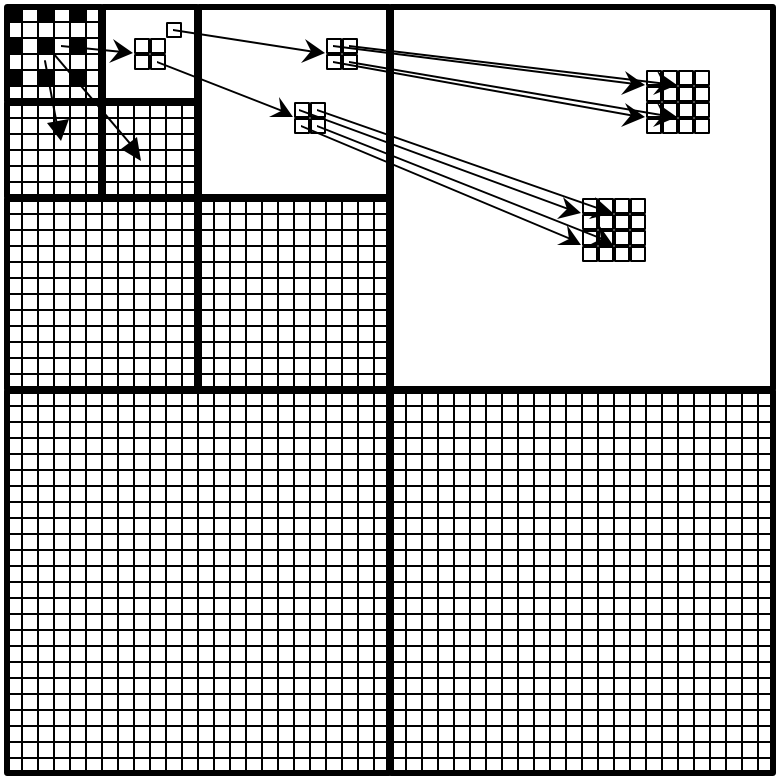
3 level dyadic subband transform shown
Arrows depict parent-child relationships in the SPIHT tree structure
Coefficients are grouped to exploit magnitude dependence
Each subband coefficient has four children
Some coefficients in the DC subband have no children

Each coefficient is denoted by it coordinates (i,j)

Set Types

(i,j): Single coefficient
C(i,j): Children of (i,j)
D(i,j): Descendants of (i,j)
L(i,j): D(i,j) - C(i,j)

Significant sets D(i,j) partitioned C(i,j) and L(i,j)



Rensselaer

Coding Sequence

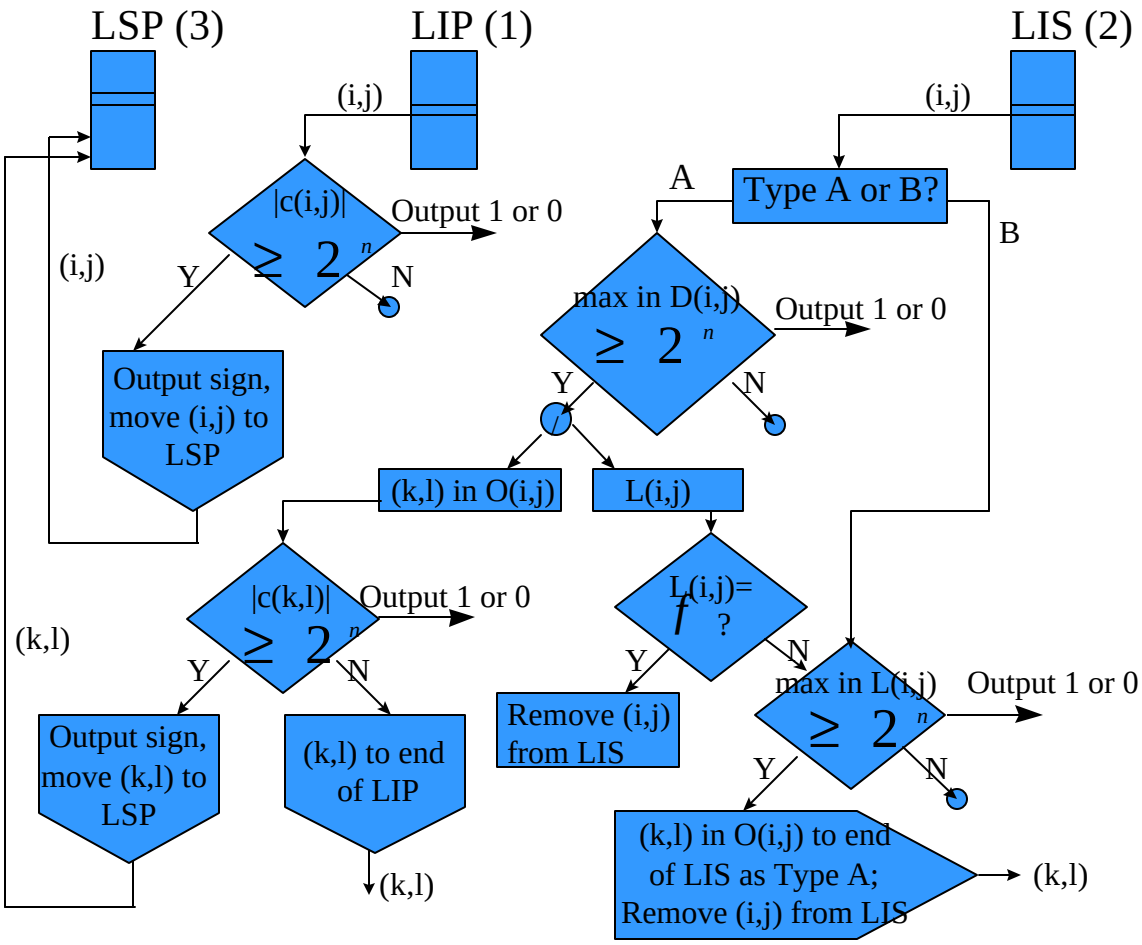
CNGV

- SPIHT uses three lists LIP, LIS, LSP visited for significance testing in that order, for a given bit-plane (*n*).
  - LIP: list of coordinates of insignificant pixels
    - Initialized by highest level low-pass (DC) subband
  - LIS: list of coordinates of insignificant sets and their type (D or L)
    - Initialized by coordinates in DC subband with descendants as D type
  - LSP: list of coordinates of significant pixels
    - Send n-th bit of all pixels found to be significant in previous passes.

Rensselaer

CNGV

SPIHT Algorithm



Rensselaer

CNGV

SPIHT ALGORITHM

1. **Initialization:** output $n = \left\lfloor \log_2 \left(\max_{(i,j)} \{ |c_{i,j}| \} \right) \right\rfloor$; set the LSP as an empty list, and add the coordinates $(i,j) \in \mathcal{H}$ to the LIP, and only those with descendants also to the LIS, as type A entries.

2. **Sorting pass:**

   2.1. for each entry (i,j) in the LIP do:

      2.1.1. output $S_n(i,j)$;

      2.1.2. if $S_n(i,j) = 1$ then move (i,j) to the LSP and output the sign of $c_{i,j}$;

   2.2. for each entry (i,j) in the LIS do:

      2.2.1. if the entry is of type A then

      - output $S_n(\mathcal{D}(i,j))$;
      - if $S_n(\mathcal{D}(i,j)) = 1$ then
        * for each $(k,l) \in \mathcal{O}(i,j)$ do:
          · output $S_n(k,l)$;
          · if $S_n(k,l) = 1$ then add (k,l) to the LSP and output the sign of $c_{k,l}$;
          · if $S_n(k,l) = 0$ then add (k,l) to the end of the LIP;
        * if $\mathcal{L}(i,j) \neq \emptyset$ then move (i,j) to the end of the LIS, as an entry of type B, and go to Step **2.2.2**; otherwise, remove entry (i,j) from the LIS;

      2.2.2. if the entry is of type B then

      - output $S_n(\mathcal{L}(i,j))$;
      - if $S_n(\mathcal{L}(i,j)) = 1$ then
        * add each $(k,l) \in \mathcal{O}(i,j)$ to the end of the LIS as an entry of type A;
        * remove (i,j) from the LIS.

3. **Refinement pass:** for each entry (i,j) in the LSP, except those included in the last sorting pass (i.e., with same n), output the n-th most significant bit of $|c_{i,j}|$;

4. **Quantization-step update:** decrement n by 1 and go to Step **2**.

Rensselaer

CNGV

LSP Sorting by Magnitude and Progressive Bit-Plane Transmission

Transmission of magnitude-sorted coefficients

	sign	s	s	s	s	s	s	s	s	s	s	s	s	s
msb	5	1	1	0	0	0	0	0	0	0	0	0	0	0
	4	®	®	1	1	0	0	0	0	0	0	0	0	0
	3	®	®	®	®	1	1	1	1	0	0	0	0	0
	2	®	®	®	®	®	®	®	®	1	1	1	1	1
	1	®	®	®	®	®	®	®	®	®	®	®	®	®
lsb	0	®	®	®	®	®	®	®	®	®	®	®	®	®

Send *n*-th bit of all coefficients found to be significant in previous passes.

Rensselaer

CNGV

	0	1	2	3	4	5	6	7
0	63	-34	49	10	7	13	-12	7
1	-31	23	14	-13	3	4	6	-1
2	15	14	3	-12	5	-7	3	9
3	-9	-7	14	8	4	-2	3	2
4	-5	9	-1	47	4	6	-2	2
5	3	0	-3	2	3	-2	0	4
6	2	-3	6	-4	3	6	3	6
7	5	11	5	6	0	3	-4	4

Wavelet Transform Example

Example of Application for Image Compression

1 Example with SPIHT algorithm

Figure 1 shows the example of data in a small pyramid structure, of the type resulting from an image wavelet decomposition, that was used by J.M. Shapiro in his paper "Embedded Image Coding Using Zerotrees of Wavelet Coefficients," *IEEE Transactions on Signal Processing,*, vol. 41, Dec. 1993, to describe his EZW image coding algorithm.

We applied the SPIHT algorithm to the same set of data, for one pass. The results are shown in Table 1, indicating the data coded and the updating on the control lists (to save space only the modifications are shown). The notation is defined in the patent description of the algorithm. For a quick reference, here are some of the important definitions.

LIS List of insignificant sets: contains sets of wavelet coefficients which are defined by tree structures, and which had been found to have magnitude smaller than a threshold (are insignificant). The sets exclude the coefficient corresponding to the tree or all subtree roots, and have at least four elements.

LIP List of insignificant pixels: contains individual coefficients that have magnitude smaller than the threshold.

LSP List of significant pixels: pixels found to have magnitude larger that the threshold (are significant).

$\mathcal{O}(i,j)$ in the tree structures, the set of offspring (direct descendants) of a tree node defined by pixel location (i,j).

$\mathcal{D}(i,j)$ set of descendants of node defined by pixel location (i,j).

$\mathcal{L}(i,j)$ set defined by $\mathcal{L}(i,j) = \mathcal{D}(i,j) - \mathcal{O}(i,j)$.

The following refer to the respective numbered entries in Table 1:

(1) These are the initial SPIHT settings. The initial threshold is set to 32. The notation (i,j)A or (i,j)B, indicates that an LIS entry is of type 'A' or 'B', respectively. Note the duplication of co-ordinates in the lists, as the sets in the LIS are trees without the roots. The coefficient (0,0) is not considered a root.

	0	1	2	3	4	5	6	7
0	63	–34	49	10	7	13	–12	7
1	–31	23	14	–13	3	4	6	–1
2	15	14	3	–12	5	–7	3	9
3	–9	–7	–14	8	4	–2	3	2
4	–5	9	–1	47	4	6	–2	2
5	3	0	–3	2	3	–2	0	4
6	2	–3	6	–4	3	6	3	6
7	5	11	5	6	0	3	–4	4

Figure 1: Set of image wavelet coefficients used by example. The numbers outside the box indicate the set of co-ordinates used.

(2) SPIHT begins coding the significance of the individual pixels in the LIP. When a coefficient is found to be significant it is moved to the LSP, and its sign is also coded. We used the notation 1+ and 1– to indicate when a bit 1 is immediately followed by a sign bit.

(3) After testing pixels it begins to test sets, following the entries in the LIS (active entry indicated by bold letters). In this example $\mathcal{D}(0,1)$ is the set of 20 coefficients {(0,2), (0,3), (1,2), (1,3), (0,4), (0,5), (0,6), (0,7), (1,4), (1,5), (1,6), (1,7), (2,4), (2,5), (2,6), (2,7), (3,4), (3,5), (3,6), (3,7)}. Because $\mathcal{D}(0,1)$ is significant SPIHT next tests the significance of the four offspring {(0,2), (0,3), (1,2), (1,3)}.

(4) After all offspring are tested, (0,1) is moved to the end of the LIS, and its type changes from 'A' to 'B', meaning that the new LIS entry meaning changed from $\mathcal{D}(0,1)$ to $\mathcal{L}(0,1)$ (i.e., from set of all descendants to set of all descendants minus offspring).

(5) Same procedure as in comments (3) and (4) applies to set $\mathcal{D}(1,0)$. Note that even though no offspring of (1,0) is significant, $\mathcal{D}(1,0)$ is significant because $\mathcal{L}(1,0)$ is significant.

(6) Since $\mathcal{D}(1,1)$ is insignificant, no action need to be taken. The algorithm moves to the next element in the LIS.

(7) The next LIS element, (0,1), is of type 'B', and thus $\mathcal{L}(0,1)$ is tested. Note that the co-ordinate (0,1) was moved from the beginning of the LIS in this pass. It is now tested again, but with another interpretation by the algorithm.

(8) Same as above, but $\mathcal{L}(1,0)$ is sigificant, so the set is partitioned in $\mathcal{D}(2,0)$, $\mathcal{D}(2,1)$, $\mathcal{D}(3,0)$, and $\mathcal{D}(3,1)$, and the corresponding entries are added to the LIS. At the same time, the entry (1,0)B

is removed from the LIS.

(9) The algorithm keeps evaluating the set entries as they are appended to the LIS.

(10) Each new entry is treated as in the previous cases. In this case the offspring of (2,1) are tested.

(11) In this case, because $\mathcal{L}(2,1) = \emptyset$ (no descendant other than offspring), the entry (2,1)A is removed from the LIS (instead of having its type changed to 'B').

(12) Finally, the last two entries of the LIS correspond to insignificant sets, and no action is taken. The sorting pass ends after the last entry of the LIS is tested.

(13) The final list entries in this sorting pass form the initial lists in the next sorting pass, when the threshold value is 16.

Without using any other form of entropy coding, the SPIHT algorithm used 29 bits in this first pass.

2 Example with EZW algorithm

Table 2 shows the results obtained with the EZW algorithm. The explantions, and original definition, can be found in the paper by J.M. Shapiro mentioned above. We use the abbreviation DL and SL for Shapiro's dominant and subordinate lists, respectively. The notation of **F** following a co-ordinate on the dominant list means that that an internal flag is set to indicate "significant" on that pass and its magnitude on the dominant list is set to 0 for subsequent passes.

Assuming the EZW uses (at least initially) two bits to code symbols in the alphabet {POS, NEG, ZTR, IZ}, and one bit to code the symbol Z, the EZW algorithm used 26+7=33 bits in the first pass. Since both methods are coding the same bit-plane defined by the threshold 32, both find the same set of significant coefficients, yielding images with the same mean squared error. However, SPIHT used about 10% less bits to obtain the same results because it coded different data.

The final lists may have some equal co-ordinates, but as shown in the examples, the interpretation and use of those co-ordinates by the two methods are quite different. Also, in the following passes they grow and change differently.

Comm.	Pixel or Set Tested	Output Bit	Action	Control Lists
(1)				LIS = {(0,1)A,(1,0)A,(1,1)A} LIP = {(0,0),(0,1),(1,0),(1,1)} LSP = $\emptyset$
(2)	(0,0)	1+	(0,0) to LSP	LIP = {(0,1),(1,0),(1,1)} LSP = {(0,0)}
	(0,1)	1–	(0,1) to LSP	LIP = {(1,0),(1,1)} LSP = {(0,0),(0,1)}
	(1,0)	0	none	
	(1,1)	0	none	
(3)	$\mathcal{D}(0,1)$	1	test offspring	LIS = {**(0,1)A**,(1,0)A,(1,1)A}
	(0,2)	1+	(0,2) to LSP	LSP = {(0,0),(0,1),(0,2)}
	(0,3)	0	(0,3) to LIP	LIP = {(1,0),(1,1),(0,3)}
	(1,2)	0	(1,2) to LIP	LIP = {(1,0),(1,1),(0,3),(1,2)}
	(1,3)	0	(1,3) to LIP	LIP = {(1,0),(1,1),(0,3),(1,2),(1,3)}
(4)			type changes	LIS = {(1,0)A,(1,1)A,(0,1)B}
(5)	$\mathcal{D}(1,0)$	1	test offspring	LIS = {**(1,0)A**,(1,1)A,(0,1)B}
	(2,0)	0	(2,0) to LIP	LIP = {(1,0),(1,1),(0,3),(1,2),(1,3),(2,0)}
	(2,1)	0	(2,1) to LIP	LIP = {(1,0),(1,1),(0,3),(1,2),(1,3),(2,0),(2,1)}
	(3,0)	0	(3,0) to LIP	LIP = {(1,0),(1,1),(0,3),(1,2),(1,3),(2,0),(2,1),(3,0)}
	(3,1)	0	(3,1) to LIP	LIP = {(1,0),(1,1),(0,3),(1,2),(1,3),(2,0),(2,1),(3,0),(3,1)}
			type changes	LIS = {(1,1)A,(0,1)B,(1,0)B}
(6)	$\mathcal{D}(1,1)$	0	none	LIS = {**(1,1)A**,(0,1)B,(1,0)B}
(7)	$\mathcal{L}(0,1)$	0	none	LIS = {(1,1)A,**(0,1)B**,(1,0)B}
(8)	$\mathcal{L}(1,0)$	1	add new sets	LIS = {(1,1)A,(0,1)B,(2,0)A,(2,1)A,(3,0)A,(3,1)A}
(9)	$\mathcal{D}(2,0)$	0	none	LIS = {(1,1)A,(0,1)B,**(2,0)A**,(2,1)A,(3,0)A,(3,1)A}
(10)	$\mathcal{D}(2,1)$	1	test offspring	LIS = {(1,1)A,(0,1)B,(2,0)A,**(2,1)A**,(3,0)A,(3,1)A}
	(4,2)	0	(4,2) to LIP	LIP = {(1,0),(1,1),(0,3),(1,2),(1,3),(2,0),(2,1),(3,0),(3,1),(4,2)}
	(4,3)	1+	(4,3) to LSP	LSP = {(0,0),(0,1),(0,2),(4,3)}
	(5,2)	0	(5,2) to LIP	LIP = {(1,0),(1,1),(0,3),(1,2),(1,3),(2,0),(2,1),(3,0),(3,1),(4,2),(5,2)}
	(5,3)	0	(5,3) to LIP	LIP = {(1,0),(1,1),(0,3),(1,2),(1,3),(2,0),(2,1),(3,0),(3,1),(4,2),(5,2),(5,3)}
(11)			(2,1) removed	LIS = {(1,1)A,(0,1)B,(2,0)A,(3,0)A,(3,1)A}
(12)	$\mathcal{D}(3,0)$	0	none	LIS = {(1,1)A,(0,1)B,(2,0)A,**(3,0)A**,(3,1)A}
	$\mathcal{D}(3,1)$	0	none	LIS = {(1,1)A,(0,1)B,(2,0)A,(3,0)A,**(3,1)A**}
(13)				LIS = {(1,1)A,(0,1)B,(2,0)A,(3,0)A,(3,1)A} LIP = {(1,0),(1,1),(0,3),(1,2),(1,3),(2,0),(2,1),(3,0),(3,1),(4,2),(5,2),(5,3)} LSP = {(0,0),(0,1),(0,2),(4,3)}

Table 1: Example of image coding using the SPIHT method.

Tree Root	Output Symbol	DL: dominant list SL: subordinate list
		DL = {(0,0)} SL = $\emptyset$
(0,0)	POS	DL = {(0,0)**F**,(0,1),(1,0),(1,1)} SL = {63}
(0,1)	NEG	DL = {(0,0)**F**,(1,0),(1,1),(0,2),(0,3),(1,2),(1,3),(0,1)**F**} SL = {63,34}
(1,0)	IZ	DL = {(0,0)**F**,(1,0),(1,1),(0,2),(0,3),(1,2),(1,3),(0,1)**F**,(2,0),(2,1),(3,0),(3,1)}
(1,1)	ZTR	
(0,2)	POS	DL = {(0,0)**F**,(1,0),(1,1),(0,3),(1,2),(1,3),(0,1)**F**,(2,0),(2,1),(3,0),(3,1),(0,4),(0,5),(1,4),(1,5),(0,2)**F**} SL = {63,34,49)}
(0,3)	ZTR	
(1,2)	ZTR	
(1,3)	ZTR	
(2,0)	ZTR	
(2,1)	IZ	DL = {(0,0)**F**,(1,0),(1,1),(0,3),(1,2),(1,3),(0,1)**F**, (2,0),(2,1),(3,0),(3,1),(0,4),(0,5),(1,4),(1,5),(0,2)**F**, (4,2),(4,3),(5,2),(5,3)}
(3,0)	ZTR	
(3,1)	ZTR	
(0,4)	Z	
(0,5)	Z	
(1,4)	Z	
(1,5)	Z	
(4,2)	Z	
(4,3)	POS	DL = {(0,0)**F**,(1,0),(1,1),(0,3),(1,2),(1,3),(0,1)**F**, (2,0),(2,1),(3,0), (3,1),(0,4),(0,5),(1,4),(1,5),(0,2)**F**, (4,2),(5,2),(5,3),(4,3)**F**} SL = {63,34,49,47}
(5,2)	Z	
(5,3)	Z	
		DL = {(0,0)**F**,(1,0),(1,1),(0,3),(1,2),(1,3),(0,1)**F**,(2,0),(2,1),(3,0), (3,1),(0,4),(0,5),(1,4),(1,5),(0,2)**F**, (4,2),(5,2),(5,3),(4,3)**F**} SL = {63,34,49,47}

Table 2: Example of image coding using Shapiro's EZW method.

Example of Application for Image Compression

1 Example with SPIHT algorithm

Figure 1 shows the example of data in a small pyramid structure, of the type resulting from an image wavelet decomposition, that was used by J.M. Shapiro in his paper "Embedded Image Coding Using Zerotrees of Wavelet Coefficients," *IEEE Transactions on Signal Processing,*, vol. 41, Dec. 1993, to describe his EZW image coding algorithm.

We applied the SPIHT algorithm to the same set of data, for one pass. The results are shown in Table 1, indicating the data coded and the updating on the control lists (to save space only the modifications are shown). The notation is defined in the patent description of the algorithm. For a quick reference, here are some of the important definitions.

LIS List of insignificant sets: contains sets of wavelet coefficients which are defined by tree structures, and which had been found to have magnitude smaller than a threshold (are insignificant). The sets exclude the coefficient corresponding to the tree or all subtree roots, and have at least four elements.

LIP List of insignificant pixels: contains individual coefficients that have magnitude smaller than the threshold.

LSP List of significant pixels: pixels found to have magnitude larger that the threshold (are significant).

$\mathcal{O}(i,j)$ in the tree structures, the set of offspring (direct descendants) of a tree node defined by pixel location (i,j).

$\mathcal{D}(i,j)$ set of descendants of node defined by pixel location (i,j).

$\mathcal{L}(i,j)$ set defined by $\mathcal{L}(i,j) = \mathcal{D}(i,j) - \mathcal{O}(i,j)$.

The following refer to the respective numbered entries in Table 1:

(1) These are the initial SPIHT settings. The initial threshold is set to 32. The notation (i,j)A or (i,j)B, indicates that an LIS entry is of type 'A' or 'B', respectively. Note the duplication of co-ordinates in the lists, as the sets in the LIS are trees without the roots. The coefficient (0,0) is not considered a root.

	0	1	2	3	4	5	6	7
0	63	–34	49	10	7	13	–12	7
1	–31	23	14	–13	3	4	6	–1
2	15	14	3	–12	5	–7	3	9
3	–9	–7	–14	8	4	–2	3	2
4	–5	9	–1	47	4	6	–2	2
5	3	0	–3	2	3	–2	0	4
6	2	–3	6	–4	3	6	3	6
7	5	11	5	6	0	3	–4	4

Figure 1: Set of image wavelet coefficients used by example. The numbers outside the box indicate the set of co-ordinates used.

(2) SPIHT begins coding the significance of the individual pixels in the LIP. When a coefficient is found to be significant it is moved to the LSP, and its sign is also coded. We used the notation 1+ and 1– to indicate when a bit 1 is immediately followed by a sign bit.

(3) After testing pixels it begins to test sets, following the entries in the LIS (active entry indicated by bold letters). In this example $\mathcal{D}(0,1)$ is the set of 20 coefficients {(0,2), (0,3), (1,2), (1,3), (0,4), (0,5), (0,6), (0,7), (1,4), (1,5), (1,6), (1,7), (2,4), (2,5), (2,6), (2,7), (3,4), (3,5), (3,6), (3,7)}. Because $\mathcal{D}(0,1)$ is significant SPIHT next tests the significance of the four offspring {(0,2), (0,3), (1,2), (1,3)}.

(4) After all offspring are tested, (0,1) is moved to the end of the LIS, and its type changes from 'A' to 'B', meaning that the new LIS entry meaning changed from $\mathcal{D}(0,1)$ to $\mathcal{L}(0,1)$ (i.e., from set of all descendants to set of all descendants minus offspring).

(5) Same procedure as in comments (3) and (4) applies to set $\mathcal{D}(1,0)$. Note that even though no offspring of (1,0) is significant, $\mathcal{D}(1,0)$ is significant because $\mathcal{L}(1,0)$ is significant.

(6) Since $\mathcal{D}(1,1)$ is insignificant, no action need to be taken. The algorithm moves to the next element in the LIS.

(7) The next LIS element, (0,1), is of type 'B', and thus $\mathcal{L}(0,1)$ is tested. Note that the co-ordinate (0,1) was moved from the beginning of the LIS in this pass. It is now tested again, but with another interpretation by the algorithm.

(8) Same as above, but $\mathcal{L}(1,0)$ is sigificant, so the set is partitioned in $\mathcal{D}(2,0)$, $\mathcal{D}(2,1)$, $\mathcal{D}(3,0)$, and $\mathcal{D}(3,1)$, and the corresponding entries are added to the LIS. At the same time, the entry (1,0)B

is removed from the LIS.

(9) The algorithm keeps evaluating the set entries as they are appended to the LIS.

(10) Each new entry is treated as in the previous cases. In this case the offspring of (2,1) are tested.

(11) In this case, because $\mathcal{L}(2,1) = \emptyset$ (no descendant other than offspring), the entry (2,1)A is removed from the LIS (instead of having its type changed to 'B').

(12) Finally, the last two entries of the LIS correspond to insignificant sets, and no action is taken. The sorting pass ends after the last entry of the LIS is tested.

(13) The final list entries in this sorting pass form the initial lists in the next sorting pass, when the threshold value is 16.

Without using any other form of entropy coding, the SPIHT algorithm used 29 bits in this first pass.

2 Example with EZW algorithm

Table 2 shows the results obtained with the EZW algorithm. The explantions, and original definition, can be found in the paper by J.M. Shapiro mentioned above. We use the abbreviation DL and SL for Shapiro's dominant and subordinate lists, respectively. The notation of **F** following a co-ordinate on the dominant list means that that an internal flag is set to indicate "significant" on that pass and its magnitude on the dominant list is set to 0 for subsequent passes.

Assuming the EZW uses (at least initially) two bits to code symbols in the alphabet {POS, NEG, ZTR, IZ}, and one bit to code the symbol Z, the EZW algorithm used 26+7=33 bits in the first pass. Since both methods are coding the same bit-plane defined by the threshold 32, both find the same set of significant coefficients, yielding images with the same mean squared error. However, SPIHT used about 10% less bits to obtain the same results because it coded different data.

The final lists may have some equal co-ordinates, but as shown in the examples, the interpretation and use of those co-ordinates by the two methods are quite different. Also, in the following passes they grow and change differently.

Comm.	Pixel or Set Tested	Output Bit	Action	Control Lists
(1)				LIS = {(0,1)A,(1,0)A,(1,1)A} LIP = {(0,0),(0,1),(1,0),(1,1)} LSP = $\emptyset$
(2)	(0,0)	1+	(0,0) to LSP	LIP = {(0,1),(1,0),(1,1)} LSP = {(0,0)}
	(0,1)	1–	(0,1) to LSP	LIP = {(1,0),(1,1)} LSP = {(0,0),(0,1)}
	(1,0)	0	none	
	(1,1)	0	none	
(3)	$\mathcal{D}(0,1)$	1	test offspring	LIS = {**(0,1)A**,(1,0)A,(1,1)A}
	(0,2)	1+	(0,2) to LSP	LSP = {(0,0),(0,1),(0,2)}
	(0,3)	0	(0,3) to LIP	LIP = {(1,0),(1,1),(0,3)}
	(1,2)	0	(1,2) to LIP	LIP = {(1,0),(1,1),(0,3),(1,2)}
	(1,3)	0	(1,3) to LIP	LIP = {(1,0),(1,1),(0,3),(1,2),(1,3)}
(4)			type changes	LIS = {(1,0)A,(1,1)A,(0,1)B}
(5)	$\mathcal{D}(1,0)$	1	test offspring	LIS = {**(1,0)A**,(1,1)A,(0,1)B}
	(2,0)	0	(2,0) to LIP	LIP = {(1,0),(1,1),(0,3),(1,2),(1,3),(2,0)}
	(2,1)	0	(2,1) to LIP	LIP = {(1,0),(1,1),(0,3),(1,2),(1,3),(2,0),(2,1)}
	(3,0)	0	(3,0) to LIP	LIP = {(1,0),(1,1),(0,3),(1,2),(1,3),(2,0),(2,1),(3,0)}
	(3,1)	0	(3,1) to LIP	LIP = {(1,0),(1,1),(0,3),(1,2),(1,3),(2,0),(2,1),(3,0),(3,1)}
			type changes	LIS = {(1,1)A,(0,1)B,(1,0)B}
(6)	$\mathcal{D}(1,1)$	0	none	LIS = {**(1,1)A**,(0,1)B,(1,0)B}
(7)	$\mathcal{L}(0,1)$	0	none	LIS = {(1,1)A,**(0,1)B**,(1,0)B}
(8)	$\mathcal{L}(1,0)$	1	add new sets	LIS = {(1,1)A,(0,1)B,(2,0)A,(2,1)A,(3,0)A,(3,1)A}
(9)	$\mathcal{D}(2,0)$	0	none	LIS = {(1,1)A,(0,1)B,**(2,0)A**,(2,1)A,(3,0)A,(3,1)A}
(10)	$\mathcal{D}(2,1)$	1	test offspring	LIS = {(1,1)A,(0,1)B,(2,0)A,**(2,1)A**,(3,0)A,(3,1)A}
	(4,2)	0	(4,2) to LIP	LIP = {(1,0),(1,1),(0,3),(1,2),(1,3),(2,0),(2,1),(3,0),(3,1),(4,2)}
	(4,3)	1+	(4,3) to LSP	LSP = {(0,0),(0,1),(0,2),(4,3)}
	(5,2)	0	(5,2) to LIP	LIP = {(1,0),(1,1),(0,3),(1,2),(1,3),(2,0),(2,1),(3,0),(3,1),(4,2),(5,2)}
	(5,3)	0	(5,3) to LIP	LIP = {(1,0),(1,1),(0,3),(1,2),(1,3),(2,0),(2,1),(3,0),(3,1),(4,2),(5,2),(5,3)}
(11)			(2,1) removed	LIS = {(1,1)A,(0,1)B,(2,0)A,(3,0)A,(3,1)A}
(12)	$\mathcal{D}(3,0)$	0	none	LIS = {(1,1)A,(0,1)B,(2,0)A,**(3,0)A**,(3,1)A}
	$\mathcal{D}(3,1)$	0	none	LIS = {(1,1)A,(0,1)B,(2,0)A,(3,0)A,**(3,1)A**}
(13)				LIS = {(1,1)A,(0,1)B,(2,0)A,(3,0)A,(3,1)A} LIP = {(1,0),(1,1),(0,3),(1,2),(1,3),(2,0),(2,1),(3,0),(3,1),(4,2),(5,2),(5,3)} LSP = {(0,0),(0,1),(0,2),(4,3)}

Table 1: Example of image coding using the SPIHT method.

Tree Root	Output Symbol	DL: dominant list SL: subordinate list
		DL = {(0,0)} SL = $\emptyset$
(0,0)	POS	DL = {(0,0)**F**,(0,1),(1,0),(1,1)} SL = {63}
(0,1)	NEG	DL = {(0,0)**F**,(1,0),(1,1),(0,2),(0,3),(1,2),(1,3),(0,1)**F**} SL = {63,34}
(1,0)	IZ	DL = {(0,0)**F**,(1,0),(1,1),(0,2),(0,3),(1,2),(1,3),(0,1)**F**,(2,0),(2,1),(3,0),(3,1)}
(1,1)	ZTR	
(0,2)	POS	DL = {(0,0)**F**,(1,0),(1,1),(0,3),(1,2),(1,3),(0,1)**F**,(2,0),(2,1),(3,0),(3,1),(0,4),(0,5),(1,4),(1,5),(0,2)**F**} SL = {63,34,49)}
(0,3)	ZTR	
(1,2)	ZTR	
(1,3)	ZTR	
(2,0)	ZTR	
(2,1)	IZ	DL = {(0,0)**F**,(1,0),(1,1),(0,3),(1,2),(1,3),(0,1)**F**, (2,0),(2,1),(3,0),(3,1),(0,4),(0,5),(1,4),(1,5),(0,2)**F**, (4,2),(4,3),(5,2),(5,3)}
(3,0)	ZTR	
(3,1)	ZTR	
(0,4)	Z	
(0,5)	Z	
(1,4)	Z	
(1,5)	Z	
(4,2)	Z	
(4,3)	POS	DL = {(0,0)**F**,(1,0),(1,1),(0,3),(1,2),(1,3),(0,1)**F**, (2,0),(2,1),(3,0), (3,1),(0,4),(0,5),(1,4),(1,5),(0,2)**F**, (4,2),(5,2),(5,3),(4,3)**F**} SL = {63,34,49,47}
(5,2)	Z	
(5,3)	Z	
		DL = {(0,0)**F**,(1,0),(1,1),(0,3),(1,2),(1,3),(0,1)**F**,(2,0),(2,1),(3,0), (3,1),(0,4),(0,5),(1,4),(1,5),(0,2)**F**, (4,2),(5,2),(5,3),(4,3)**F**} SL = {63,34,49,47}

Table 2: Example of image coding using Shapiro's EZW method.

Baseline JPEG: compressed 45:1

Wavelets & SPIHT: compressed 50:1

original

Wavelet & SPIHT: compressed 100:1

original

Wavelet & SPIHT: compressed 50:1

Rensselaer

CNGV

Conclusions

- Wavelet transform and zerotree coding bring many desirable features:
  - Progressive fidelity and resolution transmission
  - Efficient lossless coding
  - Low complexity