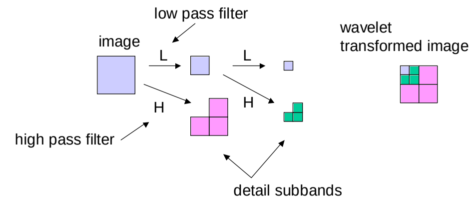
CSE 490 GZ
Introduction to Data Compression
Winter 2004

Wavelet Transform Coding

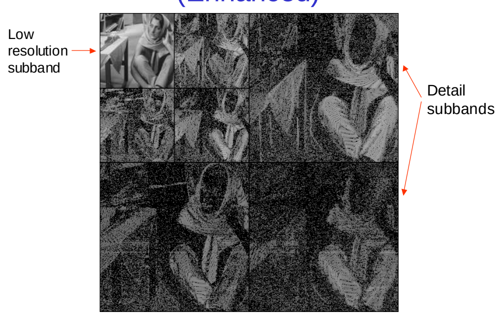
SPIHT

Wavelet Transform

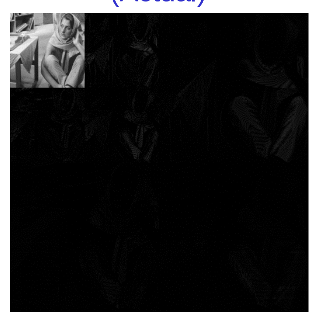
- Wavelet Transform
  - A family of transformations that filters the data into low resolution data plus detail data.



Wavelet Transformed Barbara (Enhanced)

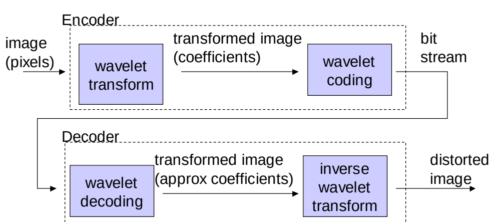


Wavelet Transformed Barbara (Actual)



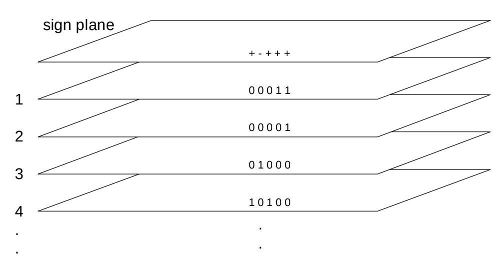
most of the details are small so they are very dark.

Wavelet Transform Compression



Wavelet coder transmits wavelet transformed image in bit plane order with the most significant bits first.

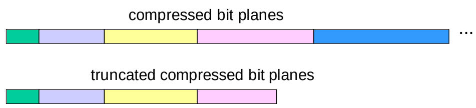
Bit Planes of Coefficients



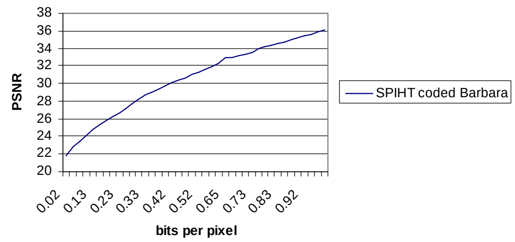
Coefficients are normalized between –1 and 1

Why Wavelet Compression Works

- Wavelet coefficients are transmitted in bit-plane order.
  - In most significant bit planes most coefficients are 0 so they can be coded efficiently.
  - Only some of the bit planes are transmitted. This is where fidelity is lost when compression is gained.
- Natural progressive transmission



Rate-Fidelity Curve

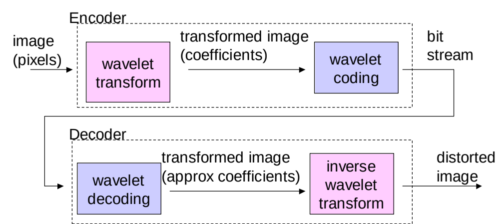


More bit planes of the wavelet transformed image that is sent the higher the fidelity.

Wavelet Coding Methods

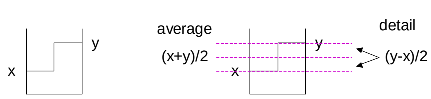
- EZW - Shapiro, 1993
  - Embedded Zerotree coding.
- SPIHT - Said and Pearlman, 1996
  - Set Partitioning in Hierarchical Trees coding. Also uses "zerotrees".
- ECECOW - Wu, 1997
  - Uses arithmetic coding with context.
- EBCOT – Taubman, 2000
  - Uses arithmetic coding with different context.
- JPEG 2000 – new standard based largely on EBCOT
- GTW – Hong, Ladner 2000
  - Uses group testing which is closely related to Golomb codes
- UWIC - Ladner, Askew, Barney 2003
  - Like GTW but uses arithmetic coding

Wavelet Transform



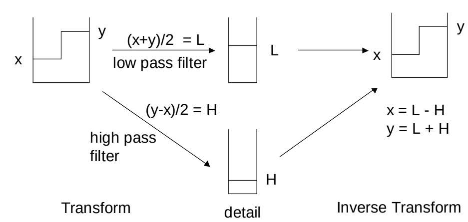
A wavelet transform decomposes the image into a low resolution version and details. The details are typically very small so they can be coded in very few bits.

One-Dimensional Average Transform (1)

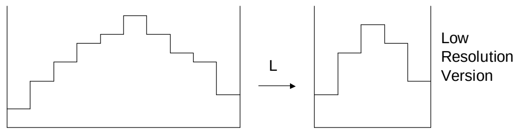


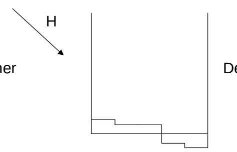
How do we represent
two data points at lower resolution?

One-Dimensional Average Transform (2)



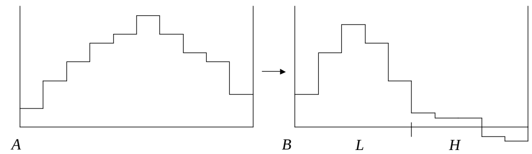
One-Dimensional Average Transform (3)





Note that the low resolution version and the detail together have the same number of values as the original.

One-Dimensional Average Transform (4)

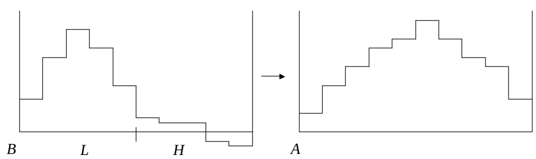


$$B[i] = \frac{1}{2}A[2i] + \frac{1}{2}A[2i+1], \quad 0 \le i < \frac{n}{2}$$

$$B[n/2+i] = -\frac{1}{2}A[2i] + \frac{1}{2}A[2i+1], \quad 0 \le i < \frac{n}{2}$$

L = B[0..n/2-1]
H = B[n/2..n-1]

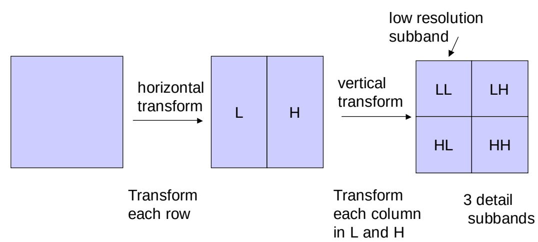
One-Dimensional Average Inverse Transform



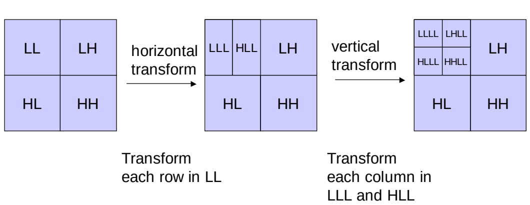
$$A[2i] = B[i] - B[n/2+i], \quad 0 \le i < \frac{n}{2}$$

$$A[2i+1] = B[i] + B[n/2+i], \quad 0 \le i < \frac{n}{2}$$

Two Dimensional Transform (1)

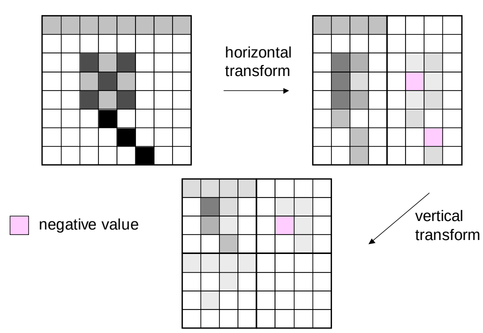


Two Dimensional Transform (1)



2 levels of transform gives 7 subbands.
k levels of transform gives 3k + 1 subbands.

Two Dimensional Average Transform



Wavelet Transformed Image

2 levels of wavelet transform

1 low resolution subband

6 detail subbands

Wavelet Transform Details

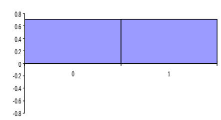
- Conversion to reals.
  - Convert gray scale to floating point.
  - Convert color to Y U V and then convert each to band to floating point. Compress separately.
- After several levels (3-8) of transform we have a matrix of floating point numbers called the wavelet transformed image (coefficients).

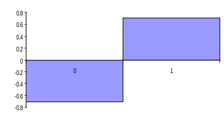
Wavelet Transforms

- Technically wavelet transforms are special kinds of linear transformations. Easiest to think of them as filters.
  - The filters depend only on a constant number of values. (bounded support)
  - Preserve energy (norm of the pixels = norm of the coefficients)
  - Inverse filters also have bounded support.
- Well-known wavelet transforms
  - Haar – like the average but orthogonal to preserve energy. Not used in practice.
  - Daubechies 9/7 – biorthogonal (inverse is not the transpose). Most commonly used in practice.

Haar Filters

low pass = $\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}$ high pass = $-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}$





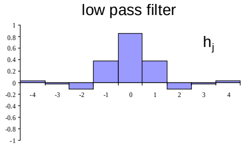
low pass $B[i] = \frac{1}{\sqrt{2}}A[2i] + \frac{1}{\sqrt{2}}A[2i+1], \quad 0 \le i < \frac{n}{2}$

high pass $B[n/2+i] = -\frac{1}{\sqrt{2}}A[2i] + \frac{1}{\sqrt{2}}A[2i+1], \quad 0 \le i < \frac{n}{2}$

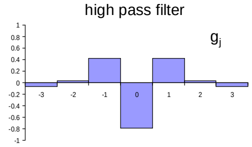
Want the sum of squares of the filter coefficients = 1

Daubechies 9/7 Filters

low pass filter



high pass filter



low pass $B[i] = \sum_{j=-4}^{4} h_j A[2i+j], \quad 0 \le i < \frac{n}{2}$

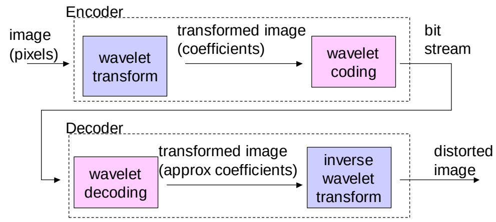
high pass $B[n/2+i] = \sum_{j=-3}^{3} g_j A[2i+j], \quad 0 \le i < \frac{n}{2}$

reflection used near boundaries

Linear Time Complexity of 2D Wavelet Transform

- Let n = number of pixels and let b be the number of coefficients in the filters.
- One level of transform takes time
  - O(bn)
- k levels of transform takes time proportional to
  - $bn + bn/4 + ... + bn/4^{k-1} < (4/3)bn$.
- The wavelet transform is linear time when the filters have constant size.
  - The point of wavelets is to use constant size filters unlike many other transforms.

Wavelet Transform

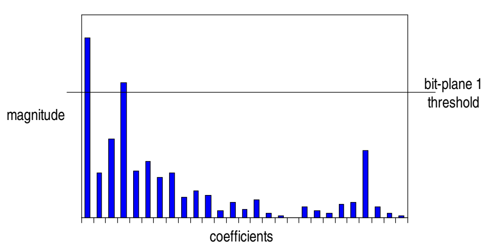


Wavelet coder transmits wavelet transformed image in bit plane order with the most significant bits first.

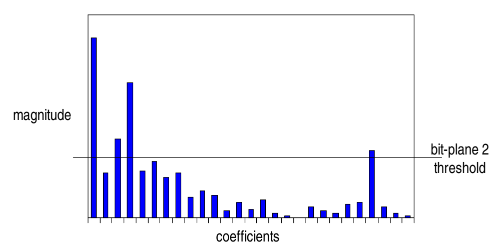
Wavelet Coding

- Normalize the coefficients to be between –1 and 1
- Transmit one bit-plane at a time
- For each bit-plane
  - Significance pass: Find the newly significant coefficients, transmit their signs.
  - Refinement pass: transmit the bits of the known significant coefficients.

Significant Coefficients



Significant Coefficients



Significance & Refinement Passes

- Code a bit-plane in two passes
  - Significance pass
    - codes previously insignificant coefficients
    - also codes sign bit
  - Refinement pass
    - refines values for previously significant coefficients
- Main idea:
  - Significance-pass bits likely to be 0;
  - Refinement-pass bit are not

Coefficient List

#	value
1	010010010110
2	001011011110
3	000001001001
4	000000010110
5	000100111101
6	000000100101
7	101101110101
8	010010011111
9	001011101101
10	000010100101

refinement bits

Bit-plane 3

bit plane 1

bpp .0014

PSNR 15.3

Compressed size

bit planes
1 – 2

bpp
.0033

PSNR
16.8

Compressed size

CSE 490gz - Lecture 15 - Winter 2004 31

bit planes
1 – 3

bpp
.0072

PSNR
18.8

Compressed size

CSE 490gz - Lecture 15 - Winter 2004 32

bit planes
1 – 4

bpp
.015

533 : 1

PSNR
20.5

Compressed size

CSE 490gz - Lecture 15 - Winter 2004 33

bit planes
1 – 5

bpp
.035

ratio
229 : 1

PSNR
22.2

Compressed size

CSE 490gz - Lecture 15 - Winter 2004 34

bit planes
1 – 6

bpp
.118

ratio
68 : 1

PSNR
24.8

Compressed size

CSE 490gz - Lecture 15 - Winter 2004 35

bit planes
1 – 7

bpp
.303

ratio
26 : 1

PSNR
28.7

Compressed size

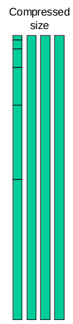
CSE 490gz - Lecture 15 - Winter 2004 36

bit planes
1 – 8

bpp
.619

ratio
13 : 1

PSNR
32.9



bit planes
1 – 9

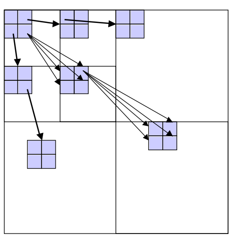
bpp
1.116

ratio
7 : 1

PSNR
37.5

The Zero-Tree Method

- Invented by Shapiro, 1993, and refined by Said and Pearlman, 1996.



If a bit plane value in a low resolution subband is insignificant then it is likely that the corresponding values in higher subbands are also insignificant in the same bit plane.

Such groups of insignificant values are called zero-trees.

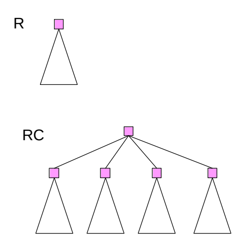
Zero-Tree Example

Values in a zero-tree are correlated.

Simplified SPIHT Coding

- Runs in passes - one for each bit plane.
- C[i,j] is the coefficient at index (i,j) and C[i,j,k] is the k-th bit of C[i,j].
- Encoder maintains two data structures.
  - S, a list of indices (i,j) such that C[i,j] is declared significant in the current bit plane.
  - Z, a stack of zero trees of two types.
    - rootless (R)
    - root-and-childless (RC)
  - The nodes in a zero tree are insignificant in the current bit plane. (ignore root in R and root and children in RC)

SPIHT Zero-Trees



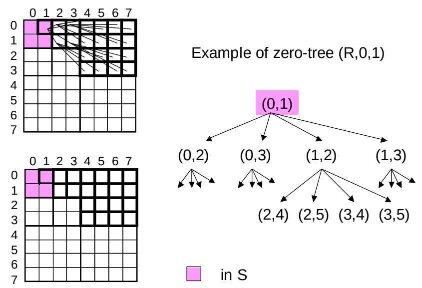
root is on the list S

all other nodes are insignificant in current bit plane

Each zero tree can be identified by its type and the index (i,j) of its root.

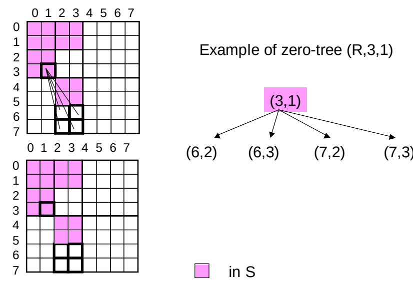
R-Tree Example (1)

Example of zero-tree (R,0,1)



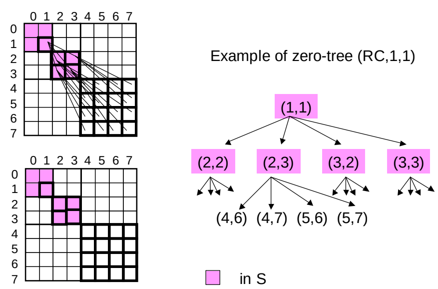
R-Tree Example (2)

Example of zero-tree (R,3,1)



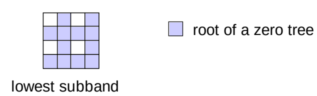
RC-Tree Example

Example of zero-tree (RC,1,1)



Initialization of SPIHT

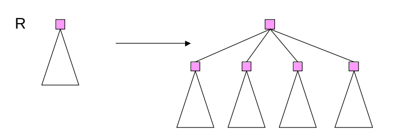
- The lowest subband indices are put into S.
  - If (i,j) in lowest subband then output sign (0 for - and 1 for +) of C[i,j] and put (i,j) into S.
- A stack Z of zero trees is formed using the lowest resolution subband indices as roots.
  - If (i,j) in the lowest subband is a root of a zero tree of type R if i is odd or (i is even and j is odd).



Iteration of SPIHT Encoder

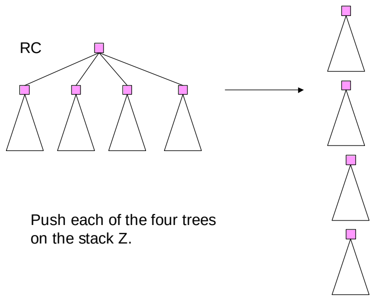
```
k-th iteration
We have list S of significant values and a stack Z of
zero trees from the previous pass or the initialization.
Significance Pass.
   while Z is not empty do
      T := pop(Z);
      if T has an index that becomes significant in bit plane k then
         output 1;
         decompose(T);
      else
         output 0;
         push T on Z'
   Z := Z'; {At this point all indices in zero trees in Z are insignificant}
Refinement Pass.
   for each (i,j) in S output the k-th significant bit, C[i,j,k].
```


Decomposition of R



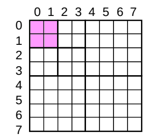
Output the sign (0 for - and 1 for +) of each of the children of the root and put them in S. Push the RC tree on the stack Z. Exception is when tree has no grandchildren. In this case, the tree dies.

Decomposition of RC



Push each of the four trees
on the stack Z.

SPIHT Coding Example: Initialization



in S

Initial data structure:

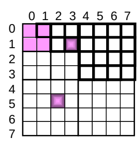
S = (0,0), (0,1), (1,0), (1,1)

Z = (R,0,1), (R,1,0), (R,1,1)

Initial output:
0 1 1 1

sign(0,0) = -
sign(0,1) = +
sign(1,0) = +
sign(1,1) = +

SPIHT Coding Example: Pass 1, Significance Pass (1)



became significant
in S

S = (0,0), (0,1), (1,0), (1,1)

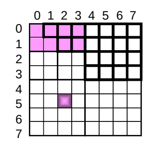
Z = (R,0,1), (R,1,0), (R,1,1)

(R,0,1) is significant
output 1

S = (0,0), (0,1), (1,0), (1,1),
(0,2), (0,3), (1,2), (1,3)
output 1101 for signs of these

Z = (RC,0,1), (R,1,0), (R,1,1)

SPIHT Coding Example: Pass 1, Significance Pass (2)



became significant
in S

S = (0,0), (0,1), (1,0), (1,1),
(0,2), (0,3), (1,2), (1,3)

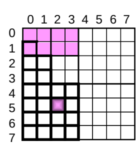
Z = (RC,0,1), (R,1,0), (R,1,1)

(RC,0,1) is not significant
output 0

S = (0,0), (0,1), (1,0), (1,1),
(0,2), (0,3), (1,2), (1,3)

Z = (R,1,0), (R,1,1)
Z' = (RC,0,1)

SPIHT Coding Example: Pass 1, Significance Pass (3)



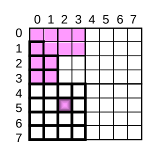
became significant
in S

S = (0,0), (0,1), (1,0), (1,1),
(0,2), (0,3), (1,2), (1,3)
Z = (R,1,0), (R,1,1)
Z' = (RC,0,1)

(R,1,0) is significant
output 1

S = (0,0), (0,1), (1,0), (1,1),
(0,2), (0,3), (1,2), (1,3),
(2,0), (2,1), (3,0), (3,1)
output 1100 for signs of these
Z = (RC,1,0), (R,1,1)
Z' = (RC,0,1)

SPIHT Coding Example: Pass 1, Significance Pass (4)



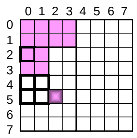
became significant
in S

S = (0,0), (0,1), (1,0), (1,1),
(0,2), (0,3), (1,2), (1,3),
(2,0), (2,1), (3,0), (3,1)
Z = (RC,1,0), (R,1,1)
Z' = (RC,0,1)

(RC,1,0) is significant
output 1

S = (0,0), (0,1), (1,0), (1,1),
(0,2), (0,3), (1,2), (1,3),
(2,0), (2,1), (3,0), (3,1)
Z = (R,2,0), (R,2,1), (R,3,0),
(R,3,1), (R,1,1)
Z' = (RC,0,1)

SPIHT Coding Example: Pass 1, Significance Pass (5)

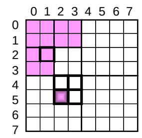


became significant
in S

S = (0,0), (0,1), (1,0), (1,1),
(0,2), (0,3), (1,2), (1,3),
(2,0), (2,1), (3,0), (3,1)
Z = (R,2,0), (R,2,1), (R,3,0),
(R,3,1) (R,1,1)
Z' = (RC,0,1)

(R,2,0) is not significant
output 0
S = (0,0), (0,1), (1,0), (1,1),
(0,2), (0,3), (1,2), (1,3),
(2,0), (2,1), (3,0), (3,1)
Z = (R,2,1), (R,3,0), (R,3,1), (R,1,1)
Z' = (R,2,0),(RC,0,1)

SPIHT Coding Example: Pass 1, Significance Pass (6)

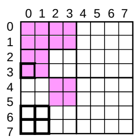


became significant
in S

S = (0,0), (0,1), (1,0), (1,1),
(0,2), (0,3), (1,2), (1,3),
(2,0), (2,1), (3,0), (3,1)
Z = (R,2,1), (R,3,0), (R,3,1), (R,1,1)
Z' = (R,2,0),(RC,0,1)

(R,2,1) is significant
output 1
S = (0,0), (0,1), (1,0), (1,1),
(0,2), (0,3), (1,2), (1,3),
(2,0), (2,1), (3,0), (3,1),
(4,2), (4,3), (5,2), (5,3)
output 1010 for signs of these
Z = (R,3,0), (R,3,1), (R,1,1)
Z' = (R,2,0),(RC,0,1)

SPIHT Coding Example: Pass 1, Significance Pass (7)

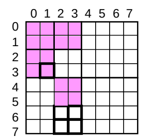


became significant
in S

S = (0,0), (0,1), (1,0), (1,1),
(0,2), (0,3), (1,2), (1,3),
(2,0), (2,1), (3,0), (3,1),
(4,2), (4,3), (5,2), (5,3)
Z = (R,3,0), (R,3,1), (R,1,1)
Z' = (R,2,0),(RC,0,1)

(R,3,0) is insignificant
output 0
S = (0,0), (0,1), (1,0), (1,1),
(0,2), (0,3), (1,2), (1,3),
(2,0), (2,1), (3,0), (3,1),
(4,2), (4,3), (5,2), (5,3)
Z = (R,3,1), (R,1,1)
Z' = (R,3,0), (R,2,0),(RC,0,1)

SPIHT Coding Example: Pass 1, Significance Pass (8)

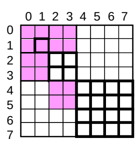


became significant
in S

S = (0,0), (0,1), (1,0), (1,1),
(0,2), (0,3), (1,2), (1,3),
(2,0), (2,1), (3,0), (3,1),
(4,2), (4,3), (5,2), (5,3)
Z = (R,3,1), (R,1,1)
Z' = (R,3,0), (R,2,0),(RC,0,1)

(R,3,1) is insignificant
output 0
S = (0,0), (0,1), (1,0), (1,1),
(0,2), (0,3), (1,2), (1,3),
(2,0), (2,1), (3,0), (3,1),
(4,2), (4,3), (5,2), (5,3)
Z = (R,1,1)
Z' = (R,3,1), (R,3,0), (R,2,0),(RC,0,1)

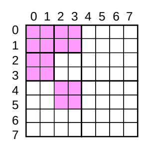
SPIHT Coding Example: Pass 1, Significance Pass (9)



became significant
in S

S = (0,0), (0,1), (1,0), (1,1),
(0,2), (0,3), (1,2), (1,3),
(2,0), (2,1), (3,0), (3,1),
(4,2), (4,3), (5,2), (5,3)
Z = (R,1,1)
Z' = (R,3,1), (R,3,0), (R,2,0),(RC,0,1)
(R,1,1) is insignificant
output 0
S = (0,0), (0,1), (1,0), (1,1),
(0,2), (0,3), (1,2), (1,3),
(2,0), (2,1), (3,0), (3,1),
(4,2), (4,3), (5,2), (5,3)
Z =
Z' = (R,1,1), (R,3,1), (R,3,0),
(R,2,0),(RC,0,1)

SPIHT Coding Example: Pass 1, Refinement Step



in S

S = (0,0), (0,1), (1,0), (1,1),
(0,2), (0,3), (1,2), (1,3),
(2,0), (2,1), (3,0), (3,1),
(4,2), (4,3), (5,2), (5,3)
Z = (R,1,1), (R,3,1), (R,3,0),
(R,2,0),(RC,0,1)

output 1011000100000010
one bit for each member of S.

37 total bits in pass 1 were output.
Initialization was 4 bits.
Total of 41 bits to send 64 bits plus
16 sign bits.

SPIHT Decoding

- The decoder emulates the encoder.
  - The decoder maintains exactly the same data structures as the encoder.
  - When the decoder has popped the Z stack to examine a zero tree it receives a bit telling it whether the tree is significant. The decoder can then do the right thing.
    - If it is significant then it does the decomposition.
    - If it is not significant then it deduces a number of zeros in the current bit plane.

SPIHT Decoder

```
k-th iteration
We have list S of significant values and a stack Z of
zero trees from the previous pass or the initialization.
Significance Pass.
   while Z is not empty do
      T := pop(Z);
      input := read;
      if input = 1 then decompose(T);
      else push T on Z'
   Z := Z'; {At this point all indices in zero trees in Z are insignificant}
Refinement Pass.
   for each (i,j) in S do C[i,j,k] := read.
```

In decompose the signs of coefficients are input

Notes on SPIHT

- SPIHT was very influential
  - People really came to believe that wavelet compression can really be practical (fast and effective).
- To yield the best compression an arithmetic coding step is added to SPIHT
  - The improvement is about .5 DB

SPIHT-AC