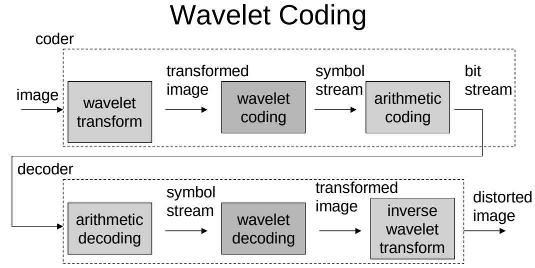
CSE 589
Applied Algorithms
Spring 1999

SPIHT Compression
Approximate String Matching

Wavelet Coding



Wavelet coder transmits wavelet transformed image in bit plane order with the most significant bits first. Compression happens when only some of the bit planes are transmitted.

Significance

- if $2^{-k} \leq |B[i,j]|$ then B[i,j] is significant in bit plane k.
- If B[i,j] is insignificant in bit plane k then $|B[i,j]| < 2^{-k}$
- The sign of B[i,j] must be output before B[i,j] becomes significant.

B

-.101	-.001
.010	.110

sign plane

-	-
+	+

bit plane 1 — B(*,*,1)

1	0
0	1

bit plane 2 — B(*,*,2)

0	0
1	1

bit plane 3 — B(*,*,3)

1	1
0	0

significant   becomes significant

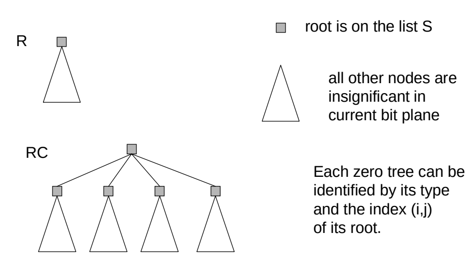
Coding Ideas

- Key coding ideas:
  - The values in first bit plane of the low resolution subband (LL...LL) are very likely significant.
  - The values in the leading bit planes of the detail subbands are likely to be insignificant.
  - Most values in the leading bit planes are insignificant.
- Transmit the wavelet transformed image in bit plane order taking advantage of the high likelihood of insignificant values.

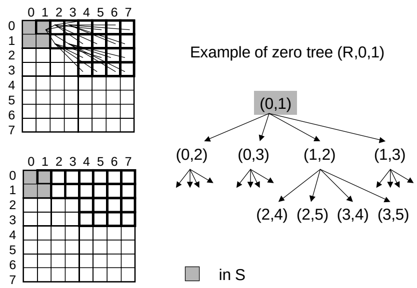
SPIHT Coding

- Runs in passes - one for each bit plane.
- Encoder maintains two data structures.
  - S, a list of indices (i,j) such that B[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)

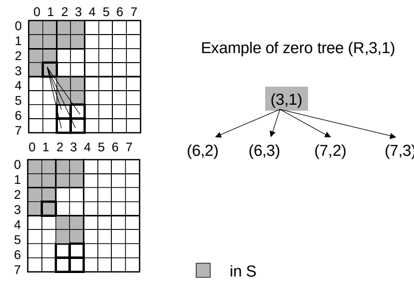
SPIHT Zero Trees



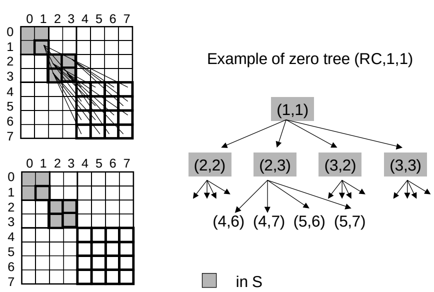
R Tree Example (1)



R Tree Example (2)

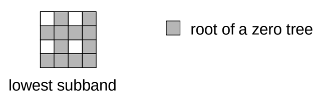


RC Tree Example



Initialization of SPIHT

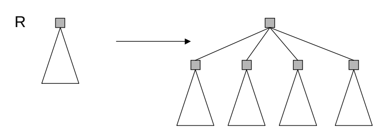
- The lowest subband indices are put into S.
  - If (i,j) in lowest subband then output sign (0 for - and 1 for +) of B[i,j] and put (i,j) into S.
- A stack Z of zero trees is formed using the lowest resolution subband indices as roots.
  - If (i,j) in the lowest subband is a root of a zero tree of type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



Pass of SPIHT

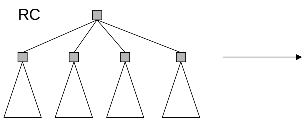
<u>k-th pass</u>
We have list S of significant values and a stack Z of zero trees from the previous pass or the initialization.
<u>Sorting step</u>.
```
  while Z is not empty do
     T := pop(Z);
     if T has an index that becomes significant in bit plane k then
        output 1;
        decompose(T);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

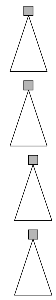
Decomposition of R



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.

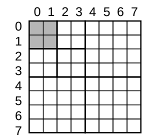
Decomposition of RC





Push each of the four trees on the stack Z.

SPIHT Coding Example: Initialization



in S

Initial data structure:

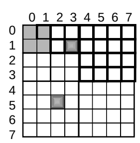
S = (0,0), (0,1), (1,0), (1,1)

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, Sorting Step (1)



became significant
in S

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

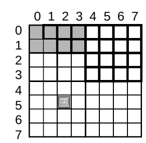
Z = (R,0,1), (R,1,0), (R,1,1)

(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, Sorting Step (2)



became significant
in S

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

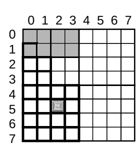
Z = (RC,0,1), (R,1,0), (R,1,1)

(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, Sorting Step (3)



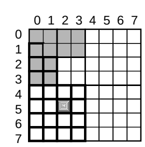
became significant
in S

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, Sorting Step (4)



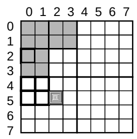
became significant
in S

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, Sorting Step (5)

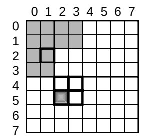


became significant
in S

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, Sorting Step (6)

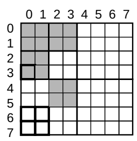


became significant
in S

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, Sorting Step (7)

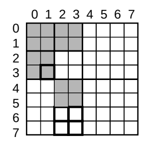


became significant
in S

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, Sorting Step (8)

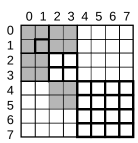


became significant
in S

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)

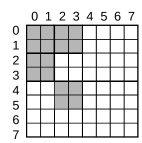
SPIHT Coding Example: Pass 1, Sorting Step (9)



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)

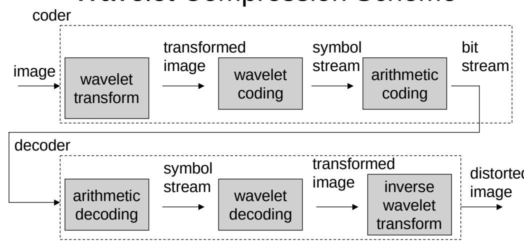
output 1011000100000010
one bit for each member of S.

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.

Wavelet Compression Scheme



9/7 Daubechies wavelet filters for 3 to 8 levels.
SPIHT encoding does well for wavelet coding.
Arithmetic coding adds a small improvement.

Notes on Wavelet Compression

- Currently the best compression available for natural images.
  - Excellent rate-fidelity curve.
  - Encoder and decoder well matched in speed.
  - No training required.
  - SPIHT has good time complexity.
  - Wavelet compressed image do not have the blockiness found in VQ and JPEG coded images.
  - Arithmetic code doesn't add much.
- Wavelet compression is very practical
  - JPEG 2000
  - FBI fingerprint database

DNA

- DNA is a large molecule that can be abstractly defined as a sequence of symbols from the set, A, C, G, T, called nucleotides.
- The human genome has about 3 billion nucleotides.
  - A huge percentage of the genome is shared by all humans.
  - Some of the variation makes us different.
  - Some of the variation is inconsequential.
  - The human genome is still being discovered.

Approximate Matching

- Two DNA sequences approximately match if one can be transformed into the other by a short sequence of replacements and insertions of gaps.
- Example:
  - `S = AGCATG`
  - `T = AGATCGT`
- Approximate matching (- is a gap)
  - `S′ = A G - - C A T G`
  - `T′ = A G A T C G T -`

Applications of Approximate Matching

- DNA string alignment.
  - Given two similar DNA sequences find the best way to align them to the same length.
- DNA database searching.
  - Find DNA sequences that are similar to the query.
- Approximate text matching for searching.
  - agrep in unix
- Spell checking
  - Find the words that most closely match the misspelled word.

Scoring an Approximate Matching

- We need a way of scoring the quality of an approximate matching.
- A scoring function is a mapping σ from {A, C, G, T, -}^{2} to integers.
  - The quantity $\sigma(x,y)$ is the score of a pair of symbols, x and y.
- Example:
  - $\sigma(x,y)$ = +2 if x=y and x in {A,C,G,T}
  - $\sigma(x,y)$ = -1 otherwise

Scoring Example

- Example:
  - S = A G - - C A T G
  - T = A G A T C G T -
- Score = 4 x 2 + 4 x(-1) = 4
- Is this the best match between the two strings with this scoring function?
  - S′ = AGCATG
  - T′ = AGATCGT

Approximate String Matching Problem

- Input: Two strings S and T in an alphabet Σ and a scoring function σ.
- Output: Two strings S' and T' in the alphabet $\Sigma' = \Sigma$ union {-} with the properties:
  - S = S' with the -'s removed.
  - T = T' with the -'s removed.
  - |S'| = |T'|
  - The score $\sum_{i=1}^{|S'|} \sigma(S'[i], T'[i])$ is maximized.