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

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

How do we represent two data points at lower resolution?

One-Dimensional Average Transform
One-Dimensional Average Transform

(3)

\[ L \rightarrow \text{Low Resolution Version} \]

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

(4)

\[ A \rightarrow B \rightarrow L \rightarrow H \]

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

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

One-Dimensional Average Inverse Transform

\[ B \rightarrow A \]

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

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

Two Dimensional Transform (1)

LL  LH  horizontal transform
HL  HH

vertical transform

Transform each row in LL

Transform each column in L and H

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

Two Dimensional Average Transform

vertical transform

negative value

vertical transform
Wavelet Transformed Image

Wavelet Transformed Image with 2 levels of wavelet transform, 1 low resolution subband, and 6 detail subbands.

Wavelet Transform Details

- Conversion to reals:
  - Convert gray scale to floating point.
  - Convert color to Y UV 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 filter coefficients:
- \[ \frac{1}{\sqrt{2}} \]
- \[ \frac{1}{\sqrt{2}} \]

High pass filter coefficients:
- \[ -\frac{1}{\sqrt{2}} \]
- \[ -\frac{1}{\sqrt{2}} \]

Want the sum of squares of the filter coefficients = 1

Daubechies 9/7 Filters

Low pass filter coefficients:
- \[ \frac{1}{\sqrt{2}} \]
- \[ \frac{1}{\sqrt{2}} \]
- \[ \frac{1}{\sqrt{2}} \]

High pass filter coefficients:
- \[ -\frac{1}{\sqrt{2}} \]
- \[ -\frac{1}{\sqrt{2}} \]
- \[ -\frac{1}{\sqrt{2}} \]

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 \( bn + bn/4 + \ldots + bn/4^k < (4/3)bn \).
- 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

Encoder

- Image (pixels) → Wavelet Transform → Transformed Image (coefficients) → Wavelet Coding → Bit Stream

Decoder

- Bit Stream → Inverse Wavelet Transform → Transformed Image (approx coefficients) → Wavelet Decoding → Distorted Image

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

- Bit Plane 1
  - Magnitude
  - Threshold
- Bit Plane 2
  - Magnitude
  - Threshold

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

<table>
<thead>
<tr>
<th>#</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.01010110</td>
</tr>
<tr>
<td>2</td>
<td>0.01011110</td>
</tr>
<tr>
<td>3</td>
<td>0.0100.0101</td>
</tr>
<tr>
<td>4</td>
<td>0.0000.0110</td>
</tr>
<tr>
<td>5</td>
<td>0.0011110.1</td>
</tr>
<tr>
<td>6</td>
<td>0.001001.01</td>
</tr>
<tr>
<td>7</td>
<td>0.1111110.1</td>
</tr>
<tr>
<td>8</td>
<td>0.01111111</td>
</tr>
<tr>
<td>9</td>
<td>0.01.001111</td>
</tr>
<tr>
<td>10</td>
<td>0.001001.01</td>
</tr>
</tbody>
</table>

Compressed size: 0.0014
PSNR: 15.3

Bit-plane 3
bit planes 1 - 2
bpp .0033
PSNR 16.8

Compressed
size

bit planes 1 - 3
bpp .0072
PSNR 18.8

Compressed
size

bit planes 1 - 4
bpp .015
ratio 533 : 1
PSNR 20.5

Compressed
size

bit planes 1 - 5
bpp .035
ratio 229 : 1
PSNR 22.2

Compressed
size

bit planes 1 - 6
bpp .118
ratio 68 : 1
PSNR 24.8

Compressed
size

bit planes 1 - 7
bpp .303
ratio 26 : 1
PSNR 28.7

Compressed
size
The Zero-Tree Method


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

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

Iteration of SPIHT Encoder

Decomposition of R
Decomposition of RC

Push each of the four trees on the stack Z.

SPIHT Coding Example: Initialization

Initial data structure:

\[ S = (0,0), (0,1), (1,0), (1,1) \]
\[ Z = (R,0,1), (R,1,0), (R,1,1) \]

Initial output:

\[ 0 \hspace{0.5cm} 1 \hspace{0.5cm} 1 \hspace{0.5cm} 1 \]

\[ \text{sign}(0,0) = - \]
\[ \text{sign}(0,1) = + \]
\[ \text{sign}(1,0) = + \]
\[ \text{sign}(1,1) = + \]

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

\[ S = (0,0), (0,1), (1,0), (1,1) \]
\[ Z = (R,0,1), (R,1,0), (R,1,1) \]
\[ (R,0,1) \text{ is significant} \]
\[ \text{output 1} \]

\[ S = (0,0), (0,1), (1,0), (1,1), (0,2), (0,3), (1,0), (1,1) \]
\[ \text{output 1101 for signs of these} \]
\[ Z = (RC,0,1), (R,1,0), (R,1,1) \]

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

\[ S = (0,0), (0,1), (1,0), (1,1), (0,2), (0,3), (1,0), (1,1) \]
\[ Z = (RC,0,1), (R,1,0), (R,1,1) \]
\[ (RC,0,1) \text{ is not significant} \]
\[ \text{output 0} \]

\[ S = (0,0), (0,1), (1,0), (1,1), (0,2), (0,3), (1,2), (1,3) \]
\[ Z = (RC,0,1) \]

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

\[ S = (0,0), (0,1), (1,0), (1,1), (0,2), (0,3), (1,0), (1,1) \]
\[ Z = (RC,0,1) \]
\[ (R,1,0) \text{ is significant} \]
\[ \text{output 1} \]

\[ S = (0,0), (0,1), (1,0), (1,1), (0,2), (0,3), (1,2), (1,3), (2,0), (2,1), (3,0), (3,1) \]
\[ \text{output 1100 for signs of these} \]
\[ Z = (RC,1,0), (R,1,1) \]
\[ Z' = (RC,0,1) \]

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

\[ S = (0,0), (0,1), (1,0), (1,1), (0,2), (0,3), (1,2), (1,3), (2,0), (2,1), (3,0), (3,1) \]
\[ Z = (RC,1,0), (R,1,1) \]
\[ Z = (RC,0,1) \]

\[ (RC,1,0) \text{ is significant} \]
\[ \text{output 1} \]

\[ S = (0,0), (0,1), (1,0), (1,1), (0,2), (0,3), (1,2), (1,3), (2,0), (2,1), (3,0), (3,1) \]
\[ Z = (RC,1,0), (R,1,1) \]
\[ Z = (RC,0,1) \]

\[ \text{became significant} \]
\[ \text{in S} \]
SPIHT Coding Example: Pass 1, Significance Pass (5)

\[ S = (0.0), (0.1), (1.0), (1.1), \\
(0.2), (0.3), (1.2), (1.3), \\
(2.0), (2.1), (3.0), (3.1) \]

\[ Z = (R,2,0), (R,1,1), (R,3,0), (R,3,1), (R,1,1) \]

(R,2,0) is not significant
output 0

\[ S = (0.0), (0.1), (1.0), (1.1), \\
(0.2), (0.3), (1.2), (1.3), \\
(2.0), (2.1), (3.0), (3.1) \]

\[ Z = (R,3,0), (R,3,1), (R,1,1), (R,2,0)(RC,0,1) \]

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

\[ S = (0.0), (0.1), (1.0), (1.1), \\
(0.2), (0.3), (1.2), (1.3), \\
(2.0), (2.1), (3.0), (3.1) \]

\[ Z = (R,2,1), (R,3,0), (R,3,1), (R,1,1) \]

(R,2,1) is significant
output 1

\[ S = (0.0), (0.1), (1.0), (1.1), \\
(0.2), (0.3), (1.2), (1.3), \\
(2.0), (2.1), (3.0), (3.1) \]

\[ Z = (R,2,0)(RC,0,1) \]

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

\[ S = (0.0), (0.1), (1.0), (1.1), \\
(0.2), (0.3), (1.2), (1.3), \\
(2.0), (2.1), (3.0), (3.1), \\
(2.1), (3.0), (1.1) \]

\[ Z = (R,3,0), (R,3,1), (R,1,1), (R,2,0)(RC,0,1) \]

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

\[ S = (0.0), (0.1), (1.0), (1.1), \\
(0.2), (0.3), (1.2), (1.3), \\
(2.0), (2.1), (0.0), (0.1), \\
(2.1), (0.0), (0.1) \]

\[ Z = (R,3,0), (R,2,0)(RC,0,1) \]

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

\[ S = (0.0), (0.1), (1.0), (1.1), \\
(0.2), (0.3), (1.2), (1.3), \\
(2.0), (2.1), (3.0), (3.1), \\
(2.1), (3.0), (1.1) \]

\[ Z = (R,3,1), (R,3,0), (R,2,0)(RC,0,1) \]

(R,1,1) is significant
output 1

\[ S = (0.0), (0.1), (1.0), (1.1), \\
(0.2), (0.3), (1.2), (1.3), \\
(2.0), (2.1), (3.0), (3.1), \\
(2.1), (3.0), (1.1) \]

\[ Z = (R,1,1), (R,3,1), (R,3,0), (R,2,0)(RC,0,1) \]

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

\[ \text{output 10110001000000010} \]

one bit for each member of S.

10
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

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

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