CSEP 590
Data Compression
Autumn 2007

EBCOT
JPEG 2000
History

• Embedded Block Coding with Optimized Truncation (EBCOT)
  – Taubman – journal paper 2000
  – Algorithm goes back to 1998 or maybe earlier
  – Basis of JPEG 2000

• Embedded
  – Prefixes of the encoded bit stream are legal encodings at lower fidelity, like SPIHT and GTW

• Block coding
  – Entropy coding of blocks of bit planes, not block transform coding like JPEG.
Features at a High Level

- **SNR scalability** (Signal to Noise Ratio)
  - Embedded code - The compressed bit stream can be truncated to yield a smaller compressed image at lower fidelity
  - Layered code – The bit stream can be partitioned into a base layer and enhancement layers. Each enhancement layer improves the fidelity of the image

- **Resolution scalability**
  - The lowest subband can be transmitted first yielding a smaller image at high fidelity.
  - Successive subbands can be transmitted to yield larger and larger images
Block Diagram of Encoder

- Image
- Wavelet transform
- Partition into coding blocks
- Truncate to achieve desired bit rate and maximum fidelity
- Bit-plane code each block independently
- Only is transmitted
Extreme Case is Normal

- Image
- Wavelet transform
- Wavelet transformed image
- Partition into one block
- Bit-plane code
- Truncate to achieve desired bit rate
Layering

- Image
- Wavelet transform
- Partition into coding blocks
- Blocked wavelet transformed image
- Bit-plane code each block independently
- Create layers

Legend:
- Red: Layer 1
- Green: Layer 2
- Blue: Layer 3
Resolution Ordering

Partition into coding blocks

bit-plane code each block independently

image

wavelet transformed image

Partition into coding blocks

blocked wavelet transformed image

1 2 5 6
3 4 7 8
9 10 13 14
11 12 15 16

1 2 3 4
5 6
7 8
9 10 11
12 13 14 15 16

1 2 3 4
5 6
7 8
9 10 11
12 13 14 15 16

1 lowest resolution
2 next lowest
3 next, next lowest
Block Coding

• Assume we are in block k, and \( c(i,j) \) is a coefficient in block k.

• Divide \( c(i,j) \) into its sign \( s(i,j) \) and \( m(i,j) \) its magnitude.

• Quantize to 
  \[
  v(i, j) = \left\lfloor \frac{m(i, j)}{q_k} + .5 \right\rfloor
  \]
  where \( q_k \) is the quantization step for block k.

• Example: \( c(i,j) = -10, q_k = 3 \).
  - \( s(i,j) = 0 \)
  - \( v(i,j) = \text{floor}(-10/3 + .5) = -2 \)
Bit Planes of Normalized Quantized Coefficients

Quantized coefficients are normalized between \(-1\) and \(1\)
Bit-Plane Coding of Blocks

- Sub-block significance coding (like group testing)
  - Some sub-blocks are declared insignificant
  - Significant sub-blocks must be coded
- Contexts are defined based on the previous bit-plane significance.
  - Zero coding (ZC) – 9 contexts
  - Run length coding (RLC) – 1 context
  - Sign coding (SC) – 5 contexts
  - Magnitude refinement coding (MR) – 3 contexts
- Block coded in raster order using arithmetic coding
Sub-Block Significance Coding

• Quad-tree organized group testing
• Block divided into 16x16 sub-blocks
• Identify in few bits the sub-blocks that are significant
Quad-Tree Subdivision
Quad-Tree Subdivision
Quad-Tree Subdivision Coding

Depth-first code = 1 for significant
                0 for insignificant
Quad-Tree Subdivision Coding

known significant in
last bit plane

Skip symbols that are already known:
1. nodes significant in previous bit plane
2. last child of significant parent of other children are insignificant

111000100110000100101000110001000
ZC – Zero Coding

- LH is transposed so that it can be treated the same as HL. \((LH)^T\) has similar characteristics to HL.
- Each coefficient has its neighbors in the same subband.
## ZC Contexts

- \( v \) = number of vertical neighbors significant in the previous bit-plane
- \( h \) = number of horizontal neighbors significant in the previous bit-plane
- \( d \) = number of diagonal neighbors significant in the previous bit-plane

<table>
<thead>
<tr>
<th>label</th>
<th>HL</th>
<th>(LH)(^T)</th>
<th>LL</th>
<th>HH</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>h</td>
<td>v</td>
<td>d</td>
<td>d</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>&gt;1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>*</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>2</td>
<td>*</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>&gt;0</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>&gt;0</td>
<td>*</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>*</td>
<td>*</td>
<td>&gt;2</td>
</tr>
</tbody>
</table>

0 ≤ h, v ≤ 2
0 ≤ d ≤ 4

Higher labels mean more likely to be significant
Examples

<table>
<thead>
<tr>
<th>Context 4</th>
<th>Context 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HL</strong></td>
<td><strong>HH</strong></td>
</tr>
<tr>
<td><img src="image1.png" alt="Diagram" /></td>
<td><img src="image2.png" alt="Diagram" /></td>
</tr>
<tr>
<td>$h = 0$</td>
<td>$h = 0$</td>
</tr>
<tr>
<td>$v = 2$</td>
<td>$v = 2$</td>
</tr>
<tr>
<td>$d = 0$</td>
<td>$d = 0$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Context 8</th>
<th>Context 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HL</strong></td>
<td><strong>HH</strong></td>
</tr>
<tr>
<td><img src="image3.png" alt="Diagram" /></td>
<td><img src="image4.png" alt="Diagram" /></td>
</tr>
<tr>
<td>$h = 2$</td>
<td>$h = 2$</td>
</tr>
<tr>
<td>$v = 0$</td>
<td>$v = 0$</td>
</tr>
<tr>
<td>$d = 0$</td>
<td>$d = 0$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Context 2</th>
<th>Context 6</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HL</strong></td>
<td><strong>HH</strong></td>
</tr>
<tr>
<td><img src="image5.png" alt="Diagram" /></td>
<td><img src="image6.png" alt="Diagram" /></td>
</tr>
<tr>
<td>$h = 0$</td>
<td>$h = 0$</td>
</tr>
<tr>
<td>$v = 0$</td>
<td>$v = 0$</td>
</tr>
<tr>
<td>$d = 2$</td>
<td>$d = 2$</td>
</tr>
</tbody>
</table>
RLC – Run Length Coding

• Looks for runs of 4 that are likely to be insignificant

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

• If all insignificant then code as a single symbol
• Main purpose – to lighten the load on the arithmetic coder.
SC – Sign Coding

$$hs = \begin{cases} 
0 & \text{if horizontal neighbors are both insignificant or of opposite sign} \\
1 & \text{if at least one horizontal neighbor is positive} \\
-1 & \text{if at least one horizontal neighbor is negative} 
\end{cases}$$

$$vs = \begin{cases} 
0 & \text{if vertical neighbors are both insignificant or of opposite sign} \\
1 & \text{if at least one vertical neighbor is positive} \\
-1 & \text{if at least one vertical neighbor is negative} 
\end{cases}$$

### SC Example

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>-1</th>
<th>-1</th>
<th>-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>hs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>0</th>
<th>-1</th>
<th>1</th>
<th>0</th>
<th>-1</th>
<th>1</th>
<th>0</th>
<th>-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>vs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>-1</th>
<th>1</th>
<th>1</th>
<th>-1</th>
<th>-1</th>
<th>-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>sign prediction</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>label</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
MR – Magnitude Refinement

• This is the refinement pass.
• Define $t = 0$ if first refinement bit, $t = 1$ otherwise.

<table>
<thead>
<tr>
<th>label</th>
<th>t</th>
<th>$h + v$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>&gt;0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>*</td>
</tr>
</tbody>
</table>
Bit Allocation

• How do we truncate the encoded blocks to achieve a desired bit rate and get maximum fidelity
Basic Set Up

- Encoded block $k$ can be truncated to $n_k$ bits.
- Total Bit Rate
  \[ \sum_k n_k \]
- Distortion attributable to block $k$ is
  \[ D_k^{n_k} = w_k^2 \sum_{(i,j) \in B_k} (c_k^{n_k}(i,j) - c(i,j))^2 \]
  where $w_k$ is the “weight” of the basis vectors for block $k$ and $c_k^{n_k}(i,j)$ is the recovered coefficients from $n_k$ bits of block $k$. 
Bit Allocation as an Optimization Problem

- Input: Given $m$ embedded codes and a bit rate target $R$
- Output: Find truncation values $n_k$, $1 \leq k \leq m$, such that

$$D = \sum_k D_k^{n_k}$$

is minimized and

$$\sum_k n_k \leq R$$
Facts about Bit Allocation

• It is an NP-hard problem generally
• There are fast approximate algorithms that work well in practice
  – GBFOS
  – Lagrange multiplier method
  – Multiple choice knapsack method
Rate-Distortion Curve

distortion
rate
encoded block
rate-distortion curve
truncation points
Picture of Bit Allocation

Pick one point from each curve so that the sum of the x values is bounded by R and the sum of the y values is minimized.

Good approximate algorithms exist because the curves are almost convex.
Notes on EBCOT

• EBCOT is quite complicated with many features.
• JPEG 2000 based on EBCOT but differs to improve compression and decompression time.
• EBCOT has
  – resolution scalability
  – SNR scalability
  – quantization
  – bit allocation
  – arithmetic coding with context and adaptivity
  – group testing (quad trees)
  – sign and refinement bit contexts
  – lots of engineering
Notes on Wavelet Compression

• Wavelets appear to be excellent for image compression
  – No blocking artifacts
  – Wavelet coding techniques abound and are very effective

• Some of the wavelet coding techniques can apply to block transforms.

• Newest generation of image compressor use wavelets, JPEG 2000.