Image Compression Fundamentals

Advisor: Prof. Andy Wu

2004/12/9 Thursday
Outline

- Introduction
- Color space and human vision system
- Transform and energy compaction
- Quantization
- Information theory and entropy coding
- JPEG example
Outline

- Introduction
  - Motivation to compression
  - Compression technology
  - Image compression framework
- Color space and human vision system
- Transform and energy compaction
- Quantization
- Information theory and entropy coding
- JPEG example
Motivation to Compression

- Large original size: Multi-Dimension
  - Still image: 24-bit RGB, 1024*768, 2,359,296 byte
  - Digital video on CD: 352*240*30fps*3byte, 7,603,200 byte/sec, A CD can only store 85 seconds of video!

- Saving transmission bandwidth
  - Device transmission rate: 1xCD: 150KByte/sec
  - Network exchange

- Security
  - Encoding implicitly encapsulates content information
Compression Technology

- Remove redundancy
  - Lossy: perceptual redundancy
  - Lossless: statistical redundancy

- Technology
  - Spatial redundancy: DCT, DWT, DPCM
  - Statistical redundancy: Run-Length coding, Variable-Length coding
  - Temporal redundancy (Video): Motion estimation/ Motion compensation
Image Compression Framework

- Raw Source Image
  - Remove Spatial Redundancy
    - DCT/DWT
  - Remove Perceptual Redundancy
    - Quantizer
  - Remove Statistical Redundancy
    - RLC/VLC

- Reconstructed Image
  - Reconstruct Spatial Redundancy
    - IDCT/IDWT
  - Reconstruct Perceptual Redundancy
    - DeQuantizer
  - Reconstruct Statistical Redundancy
    - RLD/VLD

Compressed Image
Outline

- Introduction
- Color space and human vision system
  - Human Vision System characteristics
  - CIE \{X,Y,Z\} systems
  - Color transform and video formats
- Transform and energy compaction
- Quantization
- Information theory and entropy coding
- JPEG example
Human Vision System Characteristics

- More sensitive to luminance rather than chrominance
- Use cone in bright environment, rod in dark.
- More sensitive to high contrast
- Less sensitive to high frequency signal rather than low frequency signal
- Very sensitive to edge
CIE XYZ Color Space

\[ x = \frac{X}{X + Y + X} \]
\[ y = \frac{Y}{X + Y + Z} \]
\[ z = \frac{Z}{X + Y + Z} = 1 - x - y \]
Color Transform

- **RGB to YUV**
  \[
  Y = 0.299R + 0.587G + 0.114B \\
  U = -0.147R - 0.289G + 0.436B \\
  V = 0.615R - 0.515G - 0.100B
  \]

- **YUV to RGB**
  \[
  R = Y + V/0.877 \\
  G = Y - 0.299*(Y + V/0.877) - 0.114*(Y + U/0.492) / 0.587 \\
  B = Y + U/0.492
  \]

- **RGB to YIQ**
  \[
  Y = 0.299G + 0.587G + 0.114B \\
  I = 0.596R - 0.274g - 0.322B \\
  Q = 0.211R - 0.523G + 0.311B
  \]

- **YIQ to RGB**
  \[
  R = 1.0Y + 0.956I + 0.621Q \\
  G = 1.0Y - 0.272I - 0.649Q \\
  B = 1.0Y - 1.106I + 1.703Q
  \]
Down-Sampling to Reduce Size

- Down-sample chrominance to reduce data size
- **4:4:4** – Same resolution on luminance and chrominance
- **4:2:2** – Horizontal down-sample chrominance resolution
- **4:2:0** – Horizontal and Vertical down-sample chrominance resolution
Outline

- Introduction
- Color space and human vision system
- Transform and energy compaction
  - Basic 2-D transformation forms
  - Compaction efficiency of different transform
  - DCT direct implementation and fast algorithm
  - Wavelet Transform
- Quantization
- Information theory and entropy coding
- JPEG example
Basic 2-D Transform Format

- Forward transform format

\[ T(u,v) = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} g(x,y) f(x,y,u,v) \]

- Inverse transform format

\[ g(x,y) = \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} T(u,v)i(x,y,u,v) \]

- Complexity: \( O(N^4) \)
Energy Compaction Efficiency

(a) Test Image
(b) Discrete Fourier transform
(c) Discrete Cosine transform
(d) Discrete Sine transform
(e) Discrete Hadamard transform
(f) Karhunen-Loeve transform
Coefficient of 2-D Cosine Waves

- Low frequency is at left-top corner
- HVS characteristics
DCT doesn’t have best compaction efficiency, but it is easier to implement.

Forward 2D DCT

\[
F(u, v) = \frac{2}{N} c(u)c(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x, y) \cos\left(\frac{u\pi(2x + 1)}{2N}\right) \cos\left(\frac{v\pi(2y + 1)}{2N}\right)
\]

Inverse 2D DCT

\[
f(x, y) = \frac{2}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} c(u)c(v)F(u, v) \cos\left(\frac{u\pi(2x + 1)}{2N}\right) \cos\left(\frac{v\pi(2y + 1)}{2N}\right)
\]

\[
c(i) = \begin{cases} 
1/\sqrt{2}, & i = 0 \\
1, & \text{otherwise}
\end{cases}
\]
How to Increase DCT/IDCT Speed?

- Use fast algorithm: Row/Column decomposition

Forward 1D DCT

\[ F(u) = c(u) \sum_{x=0}^{N-1} f(x) \cos\left(\frac{u\pi(2x+1)}{2N}\right) \]

Inverse 1D DCT

\[ f(x) = \sum_{u=0}^{N-1} c(u)F(u) \cos\left(\frac{u\pi(2x+1)}{2N}\right) \]

- Reduce complexity to \(O(2*N^3)\)
- Use Block-based fast algorithm, e.g.: 8x8 block
- Compute cosine value at first and put them in faster memory
Wavelet Transform

- Partition signal BW into several filter banks

- Can be cascade to do multilevel wavelet transform
Outline

- Introduction
- Color space and human vision system
- Transform and energy compaction
- Quantization
  - Decision level and reconstruction level
  - Minimize quantization error
- Information theory and entropy coding
- JPEG example
Review: Signal Characteristic

- Any analog quantity that is to be processed by a digital system must be converted to a discrete-valued number proportional to its amplitude.
- The conversion process between analog samples and discrete-valued samples is called quantization.
- Quantization will lose some information from the original signal, and this information is unrecoverable. It’s a many-to-one mapping.
If $D=Q$, the quantizer is uniform quantizer.

Larger dead-zone size generate more zero while doing quantization, but reconstruction suffer more non-linear effect.
Minimize Quantization Error

$E : \text{mean-square quantization error}$

$f : \text{amplitude of real scalar signal sample}$

$\hat{f} : \text{quantized value}$

$J : J \text{ quantization levels}$

$r_j : j_{th} \text{ reconstruction level}$

$d_j : j_{th} \text{ decision level}$

\[
E = E \{(f - \hat{f})^2\} = \int_{a_L}^{a_U} (f - \hat{f})^2 p(f)df = \sum_{j=0}^{J-1} \int_{d_j}^{d_{j+1}} (f - r_j)^2 p(f)df
\]

The optimum placing of the reconstruction level $r_j$ within the range $d_{j-1}$ to $d_j$ can be determined by minimization of $E$ with respect to $r_j$. If $J$ is large and the distribution is uniform, the probability density $p(f)$ between $d_{j+1}$ and $d_j$ may be represented as $p(r_j)$ and simply yielding:

\[
\frac{dE}{dr_j} = 0 \quad \Rightarrow \quad r_j = \frac{d_{j+1} + d_j}{2}
\]
Max Quantizer Example

<table>
<thead>
<tr>
<th>Bits</th>
<th>Uniform</th>
<th>Gaussian</th>
<th>Laplacian</th>
<th>Rayleigh</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$d_i$</td>
<td>$r_i$</td>
<td>$d_i$</td>
<td>$r_i$</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-1.0000</td>
<td>-0.5000</td>
<td></td>
<td>-∞</td>
<td>-0.7979</td>
</tr>
<tr>
<td>0.0000</td>
<td>0.5000</td>
<td></td>
<td>0.0000</td>
<td>0.7979</td>
</tr>
<tr>
<td>1.0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-1.0000</td>
<td>-0.7500</td>
<td></td>
<td>-∞</td>
<td>-1.5104</td>
</tr>
<tr>
<td>-0.5000</td>
<td>-0.2500</td>
<td></td>
<td>-0.9816</td>
<td>-0.4528</td>
</tr>
<tr>
<td>-0.0000</td>
<td>0.2500</td>
<td></td>
<td>0.0000</td>
<td>0.4528</td>
</tr>
<tr>
<td>0.5000</td>
<td>0.7500</td>
<td></td>
<td>0.9816</td>
<td>1.5104</td>
</tr>
<tr>
<td>1.0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-1.0000</td>
<td>-0.8750</td>
<td></td>
<td>-∞</td>
<td>-2.1519</td>
</tr>
<tr>
<td>-0.7500</td>
<td>-0.6250</td>
<td></td>
<td>-1.7479</td>
<td>-1.3439</td>
</tr>
<tr>
<td>-0.5000</td>
<td>-0.3750</td>
<td></td>
<td>-1.0500</td>
<td>-0.7560</td>
</tr>
<tr>
<td>-0.2500</td>
<td>-0.1250</td>
<td></td>
<td>-0.5005</td>
<td>-0.2451</td>
</tr>
<tr>
<td>0.0000</td>
<td>0.1250</td>
<td></td>
<td>0.0000</td>
<td>0.2451</td>
</tr>
<tr>
<td>0.2500</td>
<td>0.3750</td>
<td></td>
<td>0.5005</td>
<td>0.7560</td>
</tr>
<tr>
<td>0.5000</td>
<td>0.6250</td>
<td></td>
<td>1.0500</td>
<td>1.3439</td>
</tr>
<tr>
<td>0.7500</td>
<td>0.8750</td>
<td></td>
<td>1.7479</td>
<td>2.1519</td>
</tr>
<tr>
<td>1.0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Outline

- Introduction
- Color space and human vision system
- Transform and energy compaction
- Quantization
- Information theory and entropy coding
  - Self-information and entropy
  - Run-Length coding
  - Static Huffman coding
  - Static Arithmetic coding
- JPEG example
Self-Information and Entropy

- Self information $i()$ of event $A$:
  \[ i(A) = \log_x \frac{1}{P(A)} = -\log_x P(A) \]

- Entropy is, the weighted averaged self-information:
  \[ H = \sum P(A_i) i(A_i) = -\sum P(A_i) \log_x P(A_i) \]

- E.g.:
  \[ P(A) = 0.1638, P(B) = 0.55, P(C) = 0.2862 \]
  \[ i(A) = 2.6100, i(B) = 0.8625, i(C) = 1.8049 \quad (x=2) \]
  \[ H = 1.4185 \]

- Best lossless compression : equal to Entropy
Entropy Coding

- Usually include statistical models of input data:
  - The model can be fixed or adaptive
  - Fixed case
  - Adaptive case

- Entropy coding lossless compress input data, such as;
  - Run-Length Coding
  - Lempel-Ziv-Welch (LZW) Coding
  - Huffman Coding
  - Quad-tree Coding
  - Arithmetic Coding
Run-Length Coding

- Zigzag scanned sequence:
  - 61, -3, 4, -1, -4, 2, 0, 2, 0, 0, -1, -3, 4, -1, 2, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

- Symbol format (Run, Size)

- Run-Length encoded sequence:
  - (0, 6)61, (0, 2)-3, (0, 3)4, (0, 1)-1, (0, 3)-4, (0, 2)2, (1, 2)2, (0, 2)-2, (0, 2)-2, (5, 2)2, (3, 1)1, (6, 1)-1, (2, 1)-1, (4, 1)-1, (7, 1)-1, (0, 0)

- Use (0, 0) to present End of Block
Huffman Coding

- Invented by D. A. Huffman in 1952

**Algorithm:**
- Build binary tree correspond to symbol probability
- Encode symbol by traversing path from root node to leaf

**Properties of Huffman code**
- Symbols occur more frequently have shorter codeword than symbols occur less frequently. (approximate self-information)
- Unique decodability: Prefix code, tree traversing from root
- Approximate entropy closer if symbol occurrence probability is closer to quadratic distribution distributed on binary tree.
Build Static Huffman Tree/Code

Algorithm
- Sort $N$ symbols according to their probability, and map them to disjoint tree root node.
- Merge smallest two tree to one tree and update the new tree’s probability to the sum of its two children.
- Resort $N-1$ disjoint trees according to their probability.
- Merge..
- Stop when there is only one tree left.

Complexity:
- merge $O(1)$, resort $O(\log_2 n)$, total $O(N \log_2 n)$

Need to build tree:
- Before encoding
- Before decoding
Build Static Huffman Tree/Code (Cont.)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Probability</th>
<th>Codeword</th>
</tr>
</thead>
<tbody>
<tr>
<td>k</td>
<td>0.05</td>
<td>10101</td>
</tr>
<tr>
<td>l</td>
<td>0.2</td>
<td>01</td>
</tr>
<tr>
<td>u</td>
<td>0.1</td>
<td>100</td>
</tr>
<tr>
<td>w</td>
<td>0.05</td>
<td>10100</td>
</tr>
<tr>
<td>e</td>
<td>0.3</td>
<td>11</td>
</tr>
<tr>
<td>r</td>
<td>0.2</td>
<td>00</td>
</tr>
<tr>
<td>?</td>
<td>0.1</td>
<td>1011</td>
</tr>
</tbody>
</table>

---

Step 1: Merge Symbols

Step 2: Generate Codewords

Step 3: Build Static Huffman Tree

Step 4: Build Static Huffman Code

Step 5: Build Static Huffman Code (Cont.)

Step 6: Build Static Huffman Code (Cont.)

---

Example:

- $k$: 1/16
- $l$: 0.2
- $u$: 0.1
- $w$: 1/16
- $e$: 0.3
- $r$: 0.2
- $?$: 0.1
- $w$: 1/16

---

Example (Cont.):

- $k$: 1/16
- $l$: 0.2
- $u$: 0.1
- $w$: 1/16
- $e$: 0.3
- $r$: 0.2
- $?$: 0.1
- $w$: 1/16
Encoding/Decoding Flow

- **Encoding:**
  - Build tree model based on probability of each symbol, i.e. occurrence frequency of each symbol.
  - Base on built table/tree of codeword, encode each symbol to variable length bitstream
  - Concatenate these bitstreams of symbol

- **Decoding:**
  - Based on probability information, build binary tree.
  - For each input bit, traverse tree until reach leaf node
  - Extract symbol on leaf node, and restart.
# Entropy Approximation

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Code</th>
<th>Bits(actual)</th>
<th>Bits(ideal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>1</td>
<td>1</td>
<td>0.63</td>
</tr>
<tr>
<td>C</td>
<td>00</td>
<td>2</td>
<td>3.1</td>
</tr>
<tr>
<td>E</td>
<td>011</td>
<td>3</td>
<td>3.31</td>
</tr>
<tr>
<td>A</td>
<td>01000</td>
<td>5</td>
<td>6.16</td>
</tr>
<tr>
<td>G</td>
<td>01001</td>
<td>5</td>
<td>5.97</td>
</tr>
<tr>
<td>B</td>
<td>01010</td>
<td>5</td>
<td>5.38</td>
</tr>
<tr>
<td>F</td>
<td>01011</td>
<td>5</td>
<td>5.21</td>
</tr>
</tbody>
</table>

Ex: transmit ABCDEFG requires 26 bits 16.56
Arithmetic Coding

### TABLE I. Example Fixed Model for Alphabet \{a, e, i, o, u, !\}

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Probability</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>.2</td>
<td>[0, 0.2)</td>
</tr>
<tr>
<td>e</td>
<td>.3</td>
<td>[0.2, 0.5)</td>
</tr>
<tr>
<td>i</td>
<td>.1</td>
<td>[0.5, 0.6)</td>
</tr>
<tr>
<td>o</td>
<td>.2</td>
<td>[0.6, 0.8)</td>
</tr>
<tr>
<td>u</td>
<td>.1</td>
<td>[0.8, 0.9)</td>
</tr>
<tr>
<td>!</td>
<td>.1</td>
<td>[0.9, 1.0)</td>
</tr>
</tbody>
</table>

### Diagram

```
After seeing

Nothing    e     a

!    u     o     i

0.26!    0.236!

0.2!  0.233!

0.23354!
```
Encoding Flow of AC

/* ARITHMETIC ENCODING ALGORITHM. */

/* Call encode_symbol repeatedly for each symbol in the message. */
/* Ensure that a distinguished "terminator" symbol is encoded last, then */
/* transmit any value in the range [low, high). */

encode_symbol(symbol, cum_freq)
    range = high - low
    high = low + range*cum_freq[symbol-1]
    low = low + range*cum_freq[symbol]

❖ Need to determine lower bound and upper bound for currently encoding symbol
❖ Need to prevent underflow (Insufficient precision)
❖ Encode End of sequence after all symbol encoded
Decoding Flow of AC

/* ARITHMETIC DECODING ALGORITHM. */

/* "Value" is the number that has been received. */
/* Continue calling decode_symbol until the terminator symbol is returned. */

decode_symbol(cum_freq)
    find symbol such that
    cum_freq[symbol] <= (value-low)/(high-low) < cum_freq[symbol-1]
    /* This ensures that value lies within the new */
    /* [low, high) range that will be calculated by */
    /* the following lines of code. */
    range = high - low
    high  = low + range*cum_freq[symbol-1]
    low   = low + range*cum_freq[symbol]
    return symbol

- Adjust upper bound and lower bound to find first symbol in bitstream
- Need to prevent underflow and overflow while re-scaling
Implement AC by Fixed-Point Arithmetic

- Remember, most DSP processors for multimedia are fixed-point.
- Also, there is no data type has infinite resolution (AC doesn’t constrain input symbol number)
- Rescale upper bound and lower bound can solve insufficient precision problem.
Outline

- Introduction
- Color space and human vision system
- Transform and energy compaction
- Quantization
- Information theory and entropy coding
- JPEG codec example
JPEG Encoder Framework

- Table specification
  - 8x8 quantize matrix
  - Huffman table
- JPEG File Interchange Format (JFIF)
JPEG Decoder Framework

- Inverse the encoding procedures
- Table selection
  - Default quantize matrix, default Huffman table
  - User-defined tables
Differential DC and ZigZag Scan

- Energy compaction
  - Run-Length coding efficiency
  - Block size
- Do entropy coding on difference of DC
  - Huffman table for Differential DC term
| 139 144 149 153 155 155 155 155 | 235.6 -1.0 -12.1 -5.2 2.1 -1.7 -2.7 1.3 | 16 11 10 16 24 40 51 61 |
| 144 151 153 156 159 156 156 156 | -22.6 -17.5 -6.2 -3.2 -2.9 -0.1 0.4 -1.2 | 12 12 14 19 26 58 60 55 |
| 150 155 160 163 158 156 156 156 | -10.9 -9.3 -1.6 1.5 0.2 -0.9 -0.6 -0.1 | 14 13 16 24 40 57 69 56 |
| 159 161 162 160 160 159 159 159 | -7.1 -1.9 0.2 1.5 0.9 -0.1 0.0 0.3 | 14 17 22 29 51 87 80 62 |
| 159 160 161 162 162 155 155 155 | -0.6 -0.8 1.5 1.6 -0.1 -0.7 0.6 1.3 | 18 22 37 56 68 109 103 77 |
| 161 161 161 161 160 157 157 157 | 1.8 -0.2 1.6 -0.3 -0.8 1.5 1.0 -1.0 | 24 35 55 64 81 104 113 92 |
| 162 162 161 163 162 157 157 157 | -1.3 -0.4 -0.3 -1.5 -0.5 1.7 1.1 -0.8 | 49 64 78 87 103 121 120 101 |
| 162 162 161 161 163 158 158 158 | -2.6 1.6 -3.8 -1.8 1.9 1.2 -0.6 -0.4 | 72 92 95 98 112 100 103 99 |

(a) source image samples  
(b) forward DCT coefficients  
(c) quantization table

| 15 0 -1 0 0 0 0 0 | 240 0 -10 0 0 0 0 0 |
| 0 0 0 0 0 0 0 0 | -24 -12 0 0 0 0 0 0 |
| 0 0 0 0 0 0 0 0 | -14 -13 0 0 0 0 0 0 |
| 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 |
| 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 |
| 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 |
| 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 |
| 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 |

(d) normalized quantized coefficients  
(e) denormalized quantized coefficients  
(f) reconstructed image samples