Lecture 9: Image and Video Compression

Prof. YingLi Tian

Nov. 14, 2018

Department of Electrical Engineering
The City College of New York
The City University of New York (CUNY)

Thanks to G&W website, Dr. Lexing Xie, and Dr. Hao Jiang for slide materials
Outline

- image/video compression: what and why?
- Lossless compression
  - Huffman coding
  - LZW
  - Arithmetic coding
  - JPEG, PNG, GIF, etc
- Lossy compression
- Compression systems and standards
  - System standards and quality measures
  - Image coding JPEG
  - Video coding and MPEG
  - Audio coding (mp3) vs. image coding
- Summary
The need for compression

- **Image:** 6.0 million pixel camera, 3000x2000
  - 18 MB per image → 56 pictures / 1GB

- **Video:** DVD Disc 4.7 GB
  - video 720x480, RGB, 30 f/s → 31.1MB/sec
  - audio 16bits x 44.1KHz stereo → 176.4KB/s
    - → 1.5 min per DVD disc
Data Compression

- Wikipedia: “data compression, or source coding, is the process of encoding information using fewer bits (or other information-bearing units) than an unencoded representation would use through use of specific encoding schemes.”

- Applications
  - General data compression: .zip, .gz …
  - Image over network: telephone/internet/wireless/etc
  - Slow device:
    - 1xCD-ROM 150KB/s, bluetooth v1.2 up to ~0.25MB/s
  - Large multimedia databases
What can we compress?

- Goals of compression
  - Remove redundancy
  - Reduce irrelevance

- redundant: exceeding what is necessary or normal
  - symbol redundancy
    - the common and uncommon values cost the same to store
  - spatial and temporal redundancy
    - adjacent pixels are highly correlated.
Some Terms

Data Input (a sequence of symbols from an alphabet)

Encoder (compression) → Storage or networks

Decoder (decompression) → Recovered data sequence

Code (a sequence of codewords)

Information source

Lossless compression: The recovered data is exactly the same as the input.

Lossy compression: The recovered data approximates the input data.

Compression ratio = (bits used to represent the input data) / (bits of the code) = uncompressed size / compressed size
Entropy

- The number of bits needed to encode a media source is lower-bounded by its “Entropy”.
- **Self information** of an event A is defined as 
  \[-\log_b P(A)\]
  where \(P(A)\) is the probability of event A.

  If \(b\) equals 2, the unit is “bits”.
  If \(b\) equals \(e\), the unit is “nats”
  If \(b\) is 10, the unit is “hartleys”
Example

- A source outputs two symbols (the alphabet has 2 symbols) 0 or 1. P(0) = 0.25, P(1) = 0.75.

Information we get when receiving a 0 is
\[ \log_2 \left( \frac{1}{0.25} \right) = 2 \text{ bit} ; \]
when receiving a 1 is
\[ \log_2 \left( \frac{1}{0.75} \right) = 0.4150 \text{ bit} . \]
Properties of Self Information

- The letter with smaller probability has high self information.
- The information we get when receiving two independent letters are summation of each of the self information.

\[-\log_2 P(s_a, s_b) = -\log_2 P(s_a)P(s_b) = [-\log_2 P(s_a)] + [- \log_2 P(s_a)]\]
Entropy -- 1

- An source has \( n \) symbols \( \{s_1, s_2, \ldots, s_n\} \), and the symbols are independent, the average self-information is

\[
H = \sum_{i=1}^{n} P(s_i) \log_2(1/P(s_i)) \text{ bits}
\]

- \( H \) is called the *Entropy* of the source.

- The number of bits per symbol needed to encode a media source is lower-bounded by its “Entropy”.
Entropy -- 2

- Example:

A source outputs two symbols (the *alphabet* has 2 *letters*) 0 or 1. P(0) = 0.25, P(1) = 0.75.

\[
H = 0.25 \times \log_2 \left( \frac{1}{0.25} \right) + \\
0.75 \times \log_2 \left( \frac{1}{0.75} \right) \\
= 0.8113 \text{ bits}
\]

We need at least 0.8113 bits for the source in encoding.
The Entropy of an Image

- An grayscale image with 256 possible levels. $A = \{0, 1, 2, \ldots, 255\}$. Assuming the pixels are independent and the grayscales are have equal probabilities,

$$H = 256 \times \frac{1}{256} \times \log_2(1/256) = 8 \text{ bits}$$

- What about an image with only 2 levels 0 and 255? Assuming, $P(0) = 0.5$ and $P(255) = 0.5$.

$$H = 0.5 \times \log_2(1/0.5) + 0.5 \times \log_2(1/0.5) = 1 \text{ bit}$$
Estimate the Entropy

\[
\begin{align*}
\text{P(a)} &= 3/13 \\
\text{P(b)} &= 4/13 \\
\text{P(c)} &= 4/13 \\
\text{P(d)} &= 2/13
\end{align*}
\]

\[
\begin{align*}
H &= [-\text{P(a)}\log_2\text{P(a)}] + [-\text{P(b)}\log_2\text{P(b)}] + \\
&\quad [-\text{P(c)}\log_2\text{P(c)}] + [-\text{P(d)}\log_2\text{P(d)}] \\
&= 1.95\text{bits}
\end{align*}
\]
Image and Video Compression

Image Compression
Standards, Formats, and Containers

Still Image

Binary
CCITT Group 3
CCITT Group 4
JBIG (or JBIG1)
JBIG2
TIFF

Continuous Tone
JPEG
JPEG-LS
JPEG-2000
BMP
GIF
PDF
PNG
TIFF

Video
DV
H.261
H.262
H.263
H.264
MPEG-1
MPEG-2
MPEG-4
MPEG-4 AVC
AVS
HDV
M-JPEG
QuickTime
VC-1 (or WMV9)
Coding Schemes

A = \{s_1, s_2, s_3, s_4\}

P(s_1) = 0.125
P(s_2) = 0.125
P(s_3) = 0.25
P(s_4) = 0.5

Its entropy $H = 1.75$

<table>
<thead>
<tr>
<th>s_1</th>
<th>10</th>
<th>s_1</th>
<th>01</th>
</tr>
</thead>
<tbody>
<tr>
<td>s_2</td>
<td>11</td>
<td>s_2</td>
<td>11</td>
</tr>
<tr>
<td>s_3</td>
<td>0</td>
<td>s_3</td>
<td>1</td>
</tr>
<tr>
<td>s_4</td>
<td>0</td>
<td>s_4</td>
<td>0</td>
</tr>
</tbody>
</table>

Not uniquely decodeable

<table>
<thead>
<tr>
<th>s_1</th>
<th>110</th>
</tr>
</thead>
<tbody>
<tr>
<td>s_2</td>
<td>111</td>
</tr>
<tr>
<td>s_3</td>
<td>10</td>
</tr>
<tr>
<td>s_4</td>
<td>0</td>
</tr>
</tbody>
</table>

Good codewords and achieves lower bound
Huffman codes -- 1

- optimal symbol code by construction

- Binary (Huffman) tree
  - Represents Huffman code
  - Edge $\Rightarrow$ code (0 or 1)
  - Leaf $\Rightarrow$ symbol
  - Path to leaf $\Rightarrow$ encoding
  - Example
    - $A = "11", H = "10", C = "0"$

- Encoding
  1. Calculate frequency of symbols in file
  2. Create binary tree representing “best” encoding
  3. Use binary tree to encode compressed file
    - For each symbol, output path from root to leaf
    - Size of encoding = length of path
  4. Save binary tree
Huffman Coding -- 2

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Probability</th>
<th>Source reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_2$</td>
<td>0.4</td>
<td>0.4 0.4 0.4 0.4 0.6</td>
</tr>
<tr>
<td>$a_6$</td>
<td>0.3</td>
<td>0.3 0.3 0.3 0.3 0.4</td>
</tr>
<tr>
<td>$a_1$</td>
<td>0.1</td>
<td>0.1 0.2 0.3</td>
</tr>
<tr>
<td>$a_4$</td>
<td>0.1</td>
<td>0.1 0.1 0.3</td>
</tr>
<tr>
<td>$a_3$</td>
<td>0.06</td>
<td>0.1</td>
</tr>
<tr>
<td>$a_5$</td>
<td>0.04</td>
<td></td>
</tr>
</tbody>
</table>
Huffman Coding -- 3

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Probability</th>
<th>Code</th>
<th>Original source</th>
<th>Source reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_2$</td>
<td>0.4</td>
<td>1</td>
<td>0.4 1</td>
<td>0.4 1</td>
</tr>
<tr>
<td>$a_6$</td>
<td>0.3</td>
<td>00</td>
<td>0.3 00</td>
<td>0.3 00</td>
</tr>
<tr>
<td>$a_1$</td>
<td>0.1</td>
<td>011</td>
<td>0.1 011</td>
<td>0.2 010</td>
</tr>
<tr>
<td>$a_4$</td>
<td>0.1</td>
<td>0100</td>
<td>0.1 0100</td>
<td>0.1 011</td>
</tr>
<tr>
<td>$a_3$</td>
<td>0.06</td>
<td>01010</td>
<td>0.1</td>
<td>0.1 011</td>
</tr>
<tr>
<td>$a_5$</td>
<td>0.04</td>
<td>01011</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Code length $L = (0.4)(1) + (0.3)(2) + (0.1)(3) + (0.1)(4) + (0.06)(5) + (0.04)(5) = 2.2$ bits/pixel
Properties of Huffman Coding

- Huffman coding uses longer codewords for symbols with smaller probabilities and shorter codewords for symbols that often occur.
- The two longest codewords differ only in the last bit.
- The codewords are prefix codes and uniquely decodable.
- \[ H \cdot \text{Average Codeword Length} < H + 1 \]
Huffman coding is not effective for cases when there are small number of symbols and the probabilities are highly skewed.

Example:
A source has 2 symbols a and b. \( P(a) = 0.9 \) and \( P(b) = 0.1 \).

\[ H = 0.4690 \]

For Huffman Coding, average codeword length is 1. (far from optimal !)
We can encode a group symbols together and get better performance.

For the previous example, an extended source has symbols \{aa, ab, ba, bb\} and

\[
P(aa) = P(a)\times P(a) = 0.81 \quad \Rightarrow \quad 1
\]
\[
P(ab) = P(a)\times P(b) = 0.09 \quad \Rightarrow \quad 00
\]
\[
P(ba) = P(b)\times P(a) = 0.09 \quad \Rightarrow \quad 011
\]
\[
P(bb) = P(b)\times P(b) = 0.01 \quad \Rightarrow \quad 010
\]

Now the average codeword length per symbol is 
\[
(0.81\times(1) + 0.09 \times (2) + 0.09 \times (3) + 0.01\times(3)) /2 = 0.6450 \text{ (much better!)}
\]
Dictionary Based

- Dictionary based method is another way to capture the correlation of symbols.

- Static dictionary
  - Good when the data to be compressed is specific in some application.
  - For instance, to compress a student database, the world “Name”, “Student ID” will often appear.
  - Static dictionary method does not work well if the source characteristics change.
LZW (Lempel-Ziv-Welch)

Encoder
(Dictionary based method)

\[
s = \text{next input character}; \\
\text{While not EOF} \\
\{
    c = \text{next input character}; \\
    \text{if } s + c \text{ is in the directory} \\
    \quad s = s + c; \\
    \text{else} \\
    \quad \{
        \quad \text{output the codeword for } s; \\
        \quad \text{add } s+c \text{ to the directory; } \\
        \quad s = c; \\
    \quad \}
\}
\]
Output code for s
### LZW encoding example

The input string: `a b a b b a b c a b EOF`

<table>
<thead>
<tr>
<th>s</th>
<th>c</th>
<th>output</th>
<th>Codeword</th>
<th>Diction Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>1</td>
<td>4</td>
<td>ab</td>
</tr>
<tr>
<td>b</td>
<td>a</td>
<td>2</td>
<td>5</td>
<td>ba</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ab</td>
<td>b</td>
<td>4</td>
<td>6</td>
<td>abb</td>
</tr>
<tr>
<td>b</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ba</td>
<td>b</td>
<td>5</td>
<td>7</td>
<td>bab</td>
</tr>
<tr>
<td>b</td>
<td>c</td>
<td>2</td>
<td>8</td>
<td>bc</td>
</tr>
<tr>
<td>c</td>
<td>a</td>
<td>3</td>
<td>9</td>
<td>ca</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ab</td>
<td>EOF</td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
LZW Decoder

\[
\text{s = empty string;}
\]
\[
\text{While ( (k = next input code) ! = \text{EOF} )}
\]
\[
\{ \\
\quad \text{entry = dictionary entry for } k; \\
\quad \text{if (k is not in the dictionary)}
\]
\[
\quad \text{entry = } s + s[0];
\]
\[
\quad \text{output entry;}
\]
\[
\quad \text{if (s is not empty)}
\]
\[
\quad \quad \text{add string (s+entry[0]) to dictionary;}
\]
\[
\quad \text{s = entry;}
\]
\[
\}
\]
LZW Decoding example:

The input string: 1 2 4 5 2 3 4 EOF

<table>
<thead>
<tr>
<th>Index</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>b</td>
</tr>
<tr>
<td>3</td>
<td>c</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>s</th>
<th>k</th>
<th>Entry/Output</th>
<th>Codeword</th>
<th>Diction Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>1</td>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>2</td>
<td>b</td>
<td>4</td>
<td>ab</td>
</tr>
<tr>
<td>b</td>
<td>4</td>
<td>ab</td>
<td>5</td>
<td>ba</td>
</tr>
<tr>
<td>ab</td>
<td>5</td>
<td>ba</td>
<td>6</td>
<td>abb</td>
</tr>
<tr>
<td>ba</td>
<td>2</td>
<td>b</td>
<td>7</td>
<td>bab</td>
</tr>
<tr>
<td>b</td>
<td>3</td>
<td>c</td>
<td>8</td>
<td>bc</td>
</tr>
<tr>
<td>c</td>
<td>4</td>
<td>ab</td>
<td>9</td>
<td>ca</td>
</tr>
<tr>
<td>ab</td>
<td>EOF</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Arithmetic Coding

- Arithmetic coding represents an input symbol string as a small interval in $[0, 1)$
- The size of the interval equals $P(s_{k1}) P(s_{k2}) P(s_{k3}) \ldots P(s_{kn})$
- We can then represent the interval with a binary code.

0 ----------------------- 1

An half open interval $[0.3, 0.34)$ in $[0, 1)$
An example

- A source output symbols \{A, B, C, D, E, F, $\}. $ is the termination symbol. Their probabilities are as follows.

\[
\begin{align*}
P(A) &= 0.2 \\
P(B) &= 0.1 \\
P(C) &= 0.2 \\
P(D) &= 0.05 \\
P(E) &= 0.3 \\
P(F) &= 0.05 \\
P($) &= 0.1
\end{align*}
\]
Arithmetic Coding Example

Now we have an input string C A E $

0.333 = 0.0101010101 → Code: 0101010101
### Arithmetic VS Huffman

<table>
<thead>
<tr>
<th>Compression Method</th>
<th>Arithmetic</th>
<th>Huffman</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compression Ratio</td>
<td>Very Good</td>
<td>Poor</td>
</tr>
<tr>
<td>Compression Speed</td>
<td>Slow</td>
<td>Fast</td>
</tr>
<tr>
<td>Decompression Speed</td>
<td>Slow</td>
<td>Fast</td>
</tr>
<tr>
<td>Memory Space</td>
<td>Very Low</td>
<td>Low</td>
</tr>
<tr>
<td>Compressed Pattern</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Matching</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Permits Random Access</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Compress Multimedia Data Losslessly

- Model the media (audio, image, graphics data, or video as) as a source that generates symbols.

- The naïve way of choosing symbols:
  - Pixels for images and videos
  - Samples for audios
  - Characters for text messages

- Apply the lossless compression methods to the string of symbols.
Lossless Grayscale Image Compression

In lossless image compression, a prediction method is usually applied when generating symbols.
Example

A Grayscale Image
The Differential Image
Entropy $H = 6.6483$

Original image size: 227878 bytes

GrayImage (bytes) Difference Image (bytes)

Huffman coding: 192163 129397
Arithmetic coding: 190212 127220
LZ77 coding (gzip): 151685 128252
LZW (compress): 158573 136899

Entropy $H = 4.4314$
## Lossless JPEG

- Prediction options in lossless JPEG

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

- $X \sim A$
- $X \sim B$
- $X \sim C$
- $X \sim A + B - C$
- $X \sim A + (B - C)/2$
- $X \sim B + (A - C)/2$
- $X \sim (A + B)/2$
PNG (Portable Network Graphics)

- PNG is a lossless image compressing method based on LZ77.
- PNG supports three main image types: true color, grayscale and palette-based ("8-bit").
- PNG supports alpha channel
PNG -- 2

- PNG also supports interlaced coding and decoding

```
1 6 4 6 2 6 4 6
7 7 7 7 7 7 7 7
5 6 5 6 5 6 5 6
7 7 7 7 7 7 7 7
3 6 4 6 3 6 4 6
7 7 7 7 7 7 7 7
5 6 5 6 5 6 5 6
7 7 7 7 7 7 7 7
```

The scanning pattern in a 8x8 block. The whole image is Partitioned into 8x8 blocks and scanned based on the pattern In each block.
GIF (Graphics Interchange Format)

- GIF was devised by UNISYS and CompuServe.
- GIF is based on LZW lossless compression.
- GIF supports 8bit (256) color images only. Each image can have its own color table.
- It supports transparency layer and simple animation functions.
- It also supports interlaced coding and decoding.
Binary Image Compression

- Run length Coding

We can encode the image as: 2b 8w 2b 4b 8w

In fact, we do not have to save the black or white information. New code is like:

\[0 \ 2 \ 8 \ 6 \ 8\]

Tag for start from black
Run-Length Coding

- Encode the number of consecutive ‘0’ s or ‘1’ s
- Used in FAX transmission standard
- Why is run-length coding with $P(X=0) \gg P(X=1)$ actually beneficial?

$g(l) = \begin{cases} 
  p^l(1-p), & 0 \leq l \leq M - 1 \\
  p^M, & l = M 
\end{cases}$

average run-length

$$\mu_l = \frac{1 - p^M}{1 - p}$$

compression ratio

$$C = \frac{\mu_l}{m} = \frac{1 - p^M}{m(1 - p)}$$
Binary Image Compression

- More scanning patterns?
- We can compress the run-length symbols using Huffman coding, arithmetic coding, or dictionary based methods.
- Binary image compression is widely used in applications such as Facsimile.
- JBIG is a standard for binary image compression.
Audio Compression

- Audio can also be compressed in a similar way to image data.
- For lossless audio compression, prediction is usually applied first.
  - Simple prediction.
  - Adaptive prediction.
- Stereo decorrelation.
- Entropy coding.
## Audio coding versus image coding

<table>
<thead>
<tr>
<th></th>
<th>MP3 (wideband audio coding)</th>
<th>JPEG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Unit</td>
<td>Frame</td>
<td>Block</td>
</tr>
<tr>
<td>Transform</td>
<td>MDCT</td>
<td>DCT</td>
</tr>
<tr>
<td>Quantization</td>
<td>Fixed Quantization matrix base on psychoacoustic masking</td>
<td>Baseline quantization matrix + adaptive rate control</td>
</tr>
<tr>
<td>Entropy coding</td>
<td>Huffman code</td>
<td>Huffman code, run-length, differential</td>
</tr>
</tbody>
</table>
Examples

- FLAC (free lossless audio codec)
  - Polynomial fitting prediction or linear prediction.
  - Rice coding.
- MPEG4-ALS
  - Lossless audio coding standard in MPEG4.
  - Adaptive Linear Prediction.
  - Supports up to 65535 channels.
  - Fast random access.
  - Rice coding.
Lossy Compression

- Apart from lossless compression, we can further reduce the bits to represent media data by discarding “unnecessary” information.

- Media such as image, audio and video can be “modified” without seriously affecting the perceived quality.

- Lossy multimedia data compression standards include JPEG, MPEG, etc.
Methods of Discarding Information

- Reducing resolution

Original image 1/2 resolution and zoom in
- Reduce pixel color levels

Original image  

½ color levels
Video Compression and Standards
# Video Compression Standards

**TABLE 8.11**

Predictive coding in video compression standards.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>H.261</th>
<th>MPEG-1</th>
<th>H.262</th>
<th>H.263</th>
<th>MPEG-4</th>
<th>VC-1 WMV-9</th>
<th>H.264 MPEG-4 AVC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motion vector precision</td>
<td>1</td>
<td>$\frac{1}{2}$</td>
<td>$\frac{1}{2}$</td>
<td>$\frac{1}{2}$</td>
<td>$\frac{1}{4}$</td>
<td>$\frac{1}{4}$</td>
<td>$\frac{1}{4}$</td>
</tr>
<tr>
<td>Macroblock sizes</td>
<td>$16 \times 16$</td>
<td>$16 \times 16$</td>
<td>$16 \times 16$</td>
<td>$16 \times 16$</td>
<td>$16 \times 16$</td>
<td>$16 \times 16$</td>
<td>$16 \times 16$</td>
</tr>
<tr>
<td></td>
<td>$16 \times 8$</td>
<td>$16 \times 8$</td>
<td>$8 \times 8$</td>
<td>$8 \times 8$</td>
<td>$8 \times 8$</td>
<td>$8 \times 8$</td>
<td>$8 \times 8$</td>
</tr>
<tr>
<td>Transform</td>
<td>$8 \times 8$</td>
<td>$8 \times 8$</td>
<td>$8 \times 8$</td>
<td>$8 \times 8$</td>
<td>$8 \times 8$</td>
<td>$8 \times 8$</td>
<td>$4 \times 4$</td>
</tr>
<tr>
<td></td>
<td>DCT</td>
<td>DCT</td>
<td>DCT</td>
<td>DCT</td>
<td>DCT</td>
<td>4 integer DCT</td>
<td>Integer</td>
</tr>
<tr>
<td>I-frame intra-predictions</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
H261

- H.261 is an ITU video compression standard finalized in 1990.
- The basic scheme of H.261 has been retained in the newer video standards.

### Video Formats Supported by H261

<table>
<thead>
<tr>
<th>Video format</th>
<th>Luminance image resolution</th>
<th>Chrominance image resolution</th>
<th>Bitrate (Mbps) (if 30 fps and uncompressed)</th>
<th>H.261 support</th>
</tr>
</thead>
<tbody>
<tr>
<td>QCIF</td>
<td>176 × 144</td>
<td>88 × 72</td>
<td>9.1</td>
<td>Required</td>
</tr>
<tr>
<td>CIF</td>
<td>352 × 288</td>
<td>176 × 144</td>
<td>36.5</td>
<td>Optional</td>
</tr>
</tbody>
</table>
I frames and P frames

- In H.261, there are two types of compressed video frames.
- The first type of compressed frames are like JPEG compressed images. Such frames are denoted as I-frames (Intra-frames).
- The second type of frames are compressed using motion compensation schemes. These frames are denoted as P-frames (Predictive-frames).
Compression of I-frames

For each macroblock

For each 8 × 8 block

DCT
Quantization
Entropy coding

1010010...
Motion Compensation

In H.261, motion vectors are in the range [-15,15]x[-15,15], e.g., p = 15.
P-frame Compression
Quantization

- H.261 uses a constant step-size for different DCT coefficients.
- For DC coefficients

\[ Q_{DCT} = \text{round} \left( \frac{DCT}{\text{step} \_ \text{size}} \right) = \text{round} \left( \frac{DCT}{8} \right) \]

- For AC coefficients

\[ Q_{DCT} = \left\lfloor \frac{DCT}{\text{step} \_ \text{size}} \right\rfloor = \left\lfloor \frac{DCT}{2 \times \text{scale}} \right\rfloor \]

Where scale = 1 .. 31
The Encoder Diagram

Local Decoder

6 : Decoded video
The Decoder
Group of macroBlocks (GOB)

- To reduce the error propagation problem, H.261 makes sure that a "group" of Macro-Blocks can be decoded independently.
H.261 Bit Stream Syntax

- **PSC**: Picture Start Code
- **PType**: Picture Type
- **GBSC**: GOB Start Code
- **GQuant**: GOB Quantizer
- **MQuant**: MB Quantizer
- **CBP**: Coded Block Pattern
- **TR**: Temporal Reference
- **GOB**: Group of Blocks
- **GN**: Group Number
- **MB**: Macroblock
- **MVD**: Motion Vector Data
- **EOB**: End of Block
H.263

- H.263 is an improved video coding standard for video conferencing through PSTN (public switching telecommunication network).
- Apart from QCIF and CIF, it supports SubQCIF, 4CIF and 16CIF.
- H.263 has a different GOB scheme.
H.263 Motion Compensation

- The difference of MV with the median of surrounding MVs is encoded.

- Supports sub-pixel motion estimation.
MPEG-1 Video

- MPEG-1 was approved by ISO and IEC in 1991 for "Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to about 1.5Mbps".

- MPEG-1 standard is composed of
  - System
  - Video
  - Audio
  - Conformance
  - And Software

- MPEG-1's video format is called SIF(Source Input Format)
  - 352x240 for NTSC at 30f/s
  - 352x288 for PAL at 25f/s
MPEG-1 Motion Compensation

MPEG-1 introduces a new type of compressed frame: the B-frame.
Why do we need B-frames?

- Bi-directional prediction works better than only using previous frames when occlusion occurs.

For this example, the prediction from next frame is used and the prediction from previous frame is not considered.
Compression of B-frames

Previous reference frame  Target frame  Future reference frame

Difference macroblock

For each $8 \times 8$ block

DCT
Quantization
Entropy coding

Motion vectors

0011101...
Difference of MPEG-1 with H.261

- Picture formats (SIF vs. CIF)
- GOB structure

Slices in MPEG-1
**Difference of MPEG-1 with H.261**

MPEG-1 uses different quantization tables for I and P or B frames.

<table>
<thead>
<tr>
<th>8</th>
<th>16</th>
<th>19</th>
<th>22</th>
<th>26</th>
<th>27</th>
<th>29</th>
<th>34</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>16</td>
<td>22</td>
<td>24</td>
<td>27</td>
<td>29</td>
<td>34</td>
<td>37</td>
</tr>
<tr>
<td>19</td>
<td>22</td>
<td>26</td>
<td>27</td>
<td>29</td>
<td>34</td>
<td>34</td>
<td>38</td>
</tr>
<tr>
<td>22</td>
<td>22</td>
<td>26</td>
<td>27</td>
<td>29</td>
<td>34</td>
<td>37</td>
<td>40</td>
</tr>
<tr>
<td>22</td>
<td>26</td>
<td>27</td>
<td>29</td>
<td>32</td>
<td>35</td>
<td>40</td>
<td>48</td>
</tr>
<tr>
<td>26</td>
<td>27</td>
<td>29</td>
<td>32</td>
<td>35</td>
<td>40</td>
<td>48</td>
<td>58</td>
</tr>
<tr>
<td>26</td>
<td>27</td>
<td>29</td>
<td>34</td>
<td>38</td>
<td>46</td>
<td>56</td>
<td>69</td>
</tr>
<tr>
<td>27</td>
<td>29</td>
<td>35</td>
<td>38</td>
<td>46</td>
<td>56</td>
<td>69</td>
<td>83</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>16</th>
<th>16</th>
<th>16</th>
<th>16</th>
<th>16</th>
<th>16</th>
<th>16</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
</tbody>
</table>

**Intra-coding quantization table**

**Inter-coding quantization table**

**Intra mode:**

\[
QDCT[i, j] = \text{round} \left( \frac{8 \times DCT[i, j]}{\text{step\_size}[i, j]} \right) = \text{round} \left( \frac{8 \times DCT[i, j]}{Q_1[i, j] \times \text{scale}} \right)
\]

**Inter mode:**

\[
QDCT[i, j] = \left\lfloor \frac{8 \times DCT[i, j]}{\text{step\_size}[i, j]} \right\rfloor = \left\lfloor \frac{8 \times DCT[i, j]}{Q_2[i, j] \times \text{scale}} \right\rfloor
\]

(the prediction error is like noise and their DCT coefficients are quite “flat”. We can use a uniform quantization table.)
Difference of MPEG-1 with H.261

- Sub pixel motion estimation in MPEG-1.
- Motion range up to 512 pixels.
- MPEG adds another layer called “Group Of Pictures” (GOP) to allow random video access.
MPEG-1 Video Stream

- Video sequence
  - Sequence header
    - GOP
      - Picture
        - Slice
          - Macroblock
            - Block header
              - Block 0
                - Differential DC coefficient
                  - VLC run
            - Block 1
            - Block 2
            - Block 3
            - Block 4
            - Block 5
            - end_of_block
  - Sequence end code

- Group of picture layer
- Picture layer
- Slice layer
- Macroblock layer
- Block layer
### MPEG-2

- **MPEG-2 profiles and levels:**

#### Profiles and Levels in MPEG-2

<table>
<thead>
<tr>
<th>Level</th>
<th>Simple profile</th>
<th>Main profile</th>
<th>SNR scalable profile</th>
<th>Spatially scalable profile</th>
<th>High profile</th>
<th>4:2:2 profile</th>
<th>Multiview profile</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>High 1440</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Main</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Low</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

### Four levels in the main profile of MPEG-2.

<table>
<thead>
<tr>
<th>Level</th>
<th>Maximum resolution</th>
<th>Maximum fps</th>
<th>Maximum pixels/sec</th>
<th>Maximum coded data rate (Mbps)</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>1,920 × 1,152</td>
<td>60</td>
<td>62.7 × 10^6</td>
<td>80</td>
<td>Film production</td>
</tr>
<tr>
<td>High 1440</td>
<td>1,440 × 1,152</td>
<td>60</td>
<td>47.0 × 10^6</td>
<td>60</td>
<td>Consumer HDTV</td>
</tr>
<tr>
<td>Main</td>
<td>720 × 576</td>
<td>30</td>
<td>10.4 × 10^6</td>
<td>15</td>
<td>Studio TV</td>
</tr>
<tr>
<td>Low</td>
<td>352 × 288</td>
<td>30</td>
<td>3.0 × 10^6</td>
<td>4</td>
<td>Consumer tape equivalent</td>
</tr>
</tbody>
</table>
Interlace Video Compression

(a)
Scalability

- SNR scalability
  - Base layer uses rough quantization, while enhancement layers encode the residue errors.

- Spatial scalability
  - Base layer encodes a small resolution video; enhancement layers encode the difference of bigger resolution video with the “un-sampled” lower resolution one.

- Temporal scalability
  - Base layer down-samples the video in time; enhancement layers include the rest of the frames.

- Hybrid scalability
- Data partitioning
MPEG-4

- Initial goal of MPEG-4
  - Very low bit rate coding of audio visual data.

- MPEG-4 (at the end)
  - Officially up to 10 Mbits/sec.
  - Improved encoding efficiency.
  - Content-based interactivity.
  - Content-based and temporal random access.
  - Integration of both natural and synthetic objects.
  - Temporal, spatial, quality and object-based scalability.
  - Improved error resilience.
Audio-Video Object

- MPEG4 is based on the concept of media objects.
Audio Video Objects

- A media object in MPEG4 could be
  - A video of an object with “shape”.
  - The speech of a person.
  - A piece of music.
  - A static picture.
  - A synthetic 3D cartoon figure.

- In MPEG4, a scene is composed of media objects based on a scene graph:
MPEG-4 Standard

- Defines the scheme of encoding audio and video objects
  - Encoding of shaped video objects.
  - Sprite encoding.
  - Encoding of synthesized 2D and 3D objects.
- Defines the scheme of decoding media objects.
- Defines the composition and synchronization scheme.
- Defines how media objects interact with users.
Video Coding in MPEG4

- Support for 4 types of video coding:
  - Video Object Coding
    - For coding of natural and/or synthetic originated, rectangular or arbitrary shaped video objects.
  - Mesh Object Coding
    - For visual objects represented with a mesh structure.
  - Model-based Coding
    - For coding of a synthetic representation and animation of a human face and body.
  - Still Texture Coding
    - For wavelet coding of still textures.
Video Object Coding

- Video Object (VO)
  - Arbitrarily shaped video segment that has a semantic meaning.

- Video Object Plane (VOP)
  - 2D snapshot of a VO at a particular time instance.

- Coding of VOs: 3 “elements”
  - Shape
    - Rectangularly shaped VO.
    - Arbitrarily shaped VO.
  - Motion
  - Texture
Shape Coding

**Shape coding:**
- Bitmap image of a shape – alpha plane
  - Binary alpha plane.
  - Grayscale alpha plane.
- Binary alpha plane – shape information only.
- Grayscale alpha plane – shape and transparency information.
- Inter and Intra coding for the binary shapes.
Motion Compensation

- We have to deal with shaped objects.
- Motion estimation for internal blocks uses similar schemes as MPEG-1 and 2.
- For the boundary blocks, we first do “padding”, and then do motion estimation and compensation.

![Diagram showing shape boundaries and padding examples](image.png)
Shape Adaptive DCT in Texture Coding

(a) $f(x,y)$

(b) $f'(x,y)$

(c) $F'(x,v)$

(d) $F''(x,v)$

(e) $G(u,v)$
Sprite Coding

- Sprite coding is used for encoding a scene with large static background with small foreground objects.
- Background is coded only once at the beginning of the sequence as an Intra-VOP.
- It uses global motion parameters to manipulate the background.
Mesh Coding

- **Mesh**
  - Partitioning of an image into polygonal patches.

- **MPEG-4 supports 2D meshes**
  - with triangular patches.

- **Benefits of using mesh coding**
  - Easy to manipulate an object.
  - Easy to track the motion of a video object after it has been encoded.

- **Superior compression**
Model Based Coding

- MPEG-4 supports 2 types of models
  - Face object model
    - Synthetic representation of the human face with 3D polygon meshes that can be animated.
  - Body object model
    - Synthetic representation of a human body with 3D polygon meshes that can be rendered to simulate body movement.
i love compression

compression sacks that is. definitely one of my best travel/backpacking purchases

http://www.flickr.com/photos/jmhouse/2250089958/