Data Representation: Images and Sound
Announcements

- Lab Exam?

- Reminder:
  - PS8 deadline extended to July 26, 9:00 AM
  - PS9 deadline extended to July 28, 9:00 AM
Yesterday

- Data Representation
- Data Compression
- Information and redundancy
- Huffman Codes

**ALOHA**

**Fixed Width:**
0001 0110 1001 0011 0001
20 bits

**Huffman Code:**
10 0000 010 0001 10
15 bits
Decoding

100000010000110

ALOHA

• To find the character use the bits to determine path from root
parity bits

error correction
Suppose we’re sending ASCII characters over the network.

Network communications may erroneously alter bits of a message.

Simple error detection method: the parity bit.
Reminder: ASCII table

- $2^7$ (128) characters
- 7 bits needed for binary representation
- (Not shown: control characters like tab and newline, values 0…31)
Parity

- Idea: for each character (sequence of 7 bits), count the number of bits that are 1

- Sender and receiver agree to use even parity (or odd parity); sender sends extra leftmost bit
  - Even parity: Set the leftmost bit so that the number of 1’s in the byte is even.
Parity Example

- “M” is transmitted using **even parity**.

- “M” in ASCII is $77_{10}$, or $1001101$ in binary
  - four of these bits are $1$

- Transmit $01001101$ to make the number of 1-bits **even**.

- Receiver counts the number of 1-bits in character received
  - if odd, something went wrong, request retransmission
  - if even, proceed normally
  - Two bits could have been flipped, giving the illusion of correctness. **But** the probability of 2 or more bits in error is low.
Parity Example

<table>
<thead>
<tr>
<th>H</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 bit code</td>
<td>1101000</td>
</tr>
<tr>
<td>Transmit</td>
<td>11101000</td>
</tr>
<tr>
<td>Receive</td>
<td>11101000</td>
</tr>
</tbody>
</table>

Even parity

Odd number of ones. There must be an error in transmission.
An ASCII character with a correct parity bit contains *redundant information*

...because the parity bit is *predictable* from the other bits

This idea leads into the basics of information theory
Today:

- Human sensory systems and digital representations
- Digitizing images
- Digitizing sounds
- Video
human sensory systems
Why Do We Care?

- We want to represent and reproduce sensory experiences – sights and sounds
  - typically this leads to storing a huge amount of data

- Data compression for images and sounds can exploit limits on human senses
  - throw away information not needed for good-quality experience
Human Limitations

- **Range**
  - only certain pitches and loudnesses can be heard
  - only certain kinds of light are visible, and there must be enough / not too much light

- **Discrimination**
  - pitches, loudnesses, colors, intensities can’t be distinguished unless they are different enough

- **Coding**
  - nervous systems “encode” experience, e.g. rods and cones in the eye
images
digitizing
Human Vision

- Separate receptors for gray scale/dim light and color/bright light perception
- Receptors feed nervous subsystems that respond to contrast and other factors
- Spatial and temporal limits on discrimination
  - e.g. affect frame rates in video
- Three kinds of color receptors (RGB)
There are two major ways to store images:

- Vector graphics: a series of lines or curves. Expensive to compute but smoothly rescales.
- Raster or Bit-map graphics: an array of pixels. Cheap to compute, but scales poorly.

image source: ian.umces.edu
How do digital cameras record images?

- Basic idea: array of receptors: bit-map
  - each receptor records a pixel by “counting” the number of photons that strike it during exposure

- Red, green, blue recorded separately
  - each point on image produced by group of three receptors
  - each receptor behind a color filter
“Raw” Bit-Mapped Images

- Array of pixels
  - one pixel = three numbers (RGB)

- What other information do we need to display the image?
  - look at TIFF file
  - image is just a bunch of numbers
  - we need to know how wide/high it is to make sense of it
“Raw” Bit-Mapped Image Example

255, 0, 0, 255, 0, 0, 255, 0, 0, ... 255, 0, 0

extra information in file tells how wide, high, and other things
Image Formats

- Exploit human perceptual system in quality/size tradeoff

- Exploit specialized types of images to get a lot of compression
Common Standards

- **Vector:** SVG, EPS, AI, CDR.
  - Special-purpose: commonly used for high-quality illustrations, graphics, etc.

- **Raster:** JPEG (compression), GIF (compression, transparency), PNG (web portability), TIFF (printing, huge), BMP (huge)
  - Commonly used for photos and pretty much everything
Bit mapped images

a closer look at
A bit-mapped image is stored in a computer as a sequence of pixels, picture elements.
The resolution of an image is the number of pixels used to represent the image (e.g. 1024 X 768).

Each pixel represents the average color in that region.

The more pixels per area, the higher the resolution, and the more accurate the image will appear.
In bitmapped images, each pixel is represented in computer memory in binary, just like other data types.

If pixels of an image are black or white only, then we only need 1 bit per pixel to store the image, e.g. 00100 might be top row of “A”.

1 bit per pixel
Grayscale Images

- Grayscale images contain pixels that are various shades of gray, from black (maximum gray) to white (minimum gray).

- If there are 256 levels of gray for pixels, we can represent each pixel using 8 bits.
  
  11111111 = white
  ...
  (shades of gray)
  00000000 = black
Each pixel is represented with a 8-bit value that is an index into a palette of 256 colors.
RGB color systems

24 bits per pixel

Colors are represented as mixtures of red (R), green (G), and blue (B).

Each pixel is represented using three 8-bit values, one for each color component.

This representation allows for $2^{24} = 16,777,216$ different colors.

This representation is also called “true color”.

Explore with DigitalColor Meter

(image from Wikipedia)
Comparing Representations

- For a 640 X 480 image (307,200 pixels), how many bytes needed?
  - B&W: 38,400 bytes (307200/8)
  - 8-bit grayscale: 307,200 bytes
  - 256-color (8-bit color): 307,200 bytes
  - 24-bit color: 921,600 bytes (307200*24/8)

- A single RGB image of size 1600 X 1200 requires over 5.76 million bytes!

so we need compression
Compressing Raster Data

- Run-length encoding (lossless, limited)
- Color maps (GIF, good for graphics with solid areas of color)
- JPEG (lossy - a suite of techniques exploiting human visual perception)
Run-Length Encoding is a lossless compression technique used in early image files.

Instead of storing the 8-bit value for every pixel, we store an 8-bit value along with how many of these occur in a row (run).

This saves a lot when there are large runs of the same color.

(Colors: 0=Black, 255=White)
# RLE Comparison

<table>
<thead>
<tr>
<th>RLE</th>
<th>Bitmap</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 bytes</td>
<td>16 bytes</td>
</tr>
<tr>
<td>2 bytes</td>
<td>16 bytes</td>
</tr>
<tr>
<td>6 bytes</td>
<td>16 bytes</td>
</tr>
<tr>
<td>6 bytes</td>
<td>16 bytes</td>
</tr>
<tr>
<td>6 bytes</td>
<td>16 bytes</td>
</tr>
<tr>
<td>10 bytes</td>
<td>16 bytes</td>
</tr>
<tr>
<td>10 bytes</td>
<td>16 bytes</td>
</tr>
<tr>
<td>6 bytes</td>
<td>16 bytes</td>
</tr>
<tr>
<td>6 bytes</td>
<td>16 bytes</td>
</tr>
<tr>
<td>6 bytes</td>
<td>16 bytes</td>
</tr>
<tr>
<td>2 bytes</td>
<td>16 bytes</td>
</tr>
<tr>
<td>2 bytes</td>
<td>16 bytes</td>
</tr>
<tr>
<td>64 bytes</td>
<td>192 bytes</td>
</tr>
</tbody>
</table>
GIF: Graphic Interchange Format

- 8-bit pixels, mapping to a table of 256 24-bit RGB colors.
- A codebook stores recurring sequences.
- Useful for representing images with fewer colors or large areas of color like company logos.
GIF and photos

Only 256 colors leads to strange effects
JPEG (JPG): Joint Photographic Experts Group

- A **lossy** compression technique for photographic images.
- Perceptual Coding: based on what we can/cannot see.

Higher quality
Compression 2.6:1
(images from Wikipedia)

Medium quality
Compression 23:1

Lowest quality
Compression 144:1
Digitizing sound
Sound Is a Pressure Wave

- When an instrument is played or a voice speaks, periodic (many times per second) changes occur in air pressure, which we interpret as sound.
Human Sound Perception

- Frequency range: about 20 Hz* to 20,000 Hz
- Frequency discrimination drops off at high part of range
- Amplitude (roughly, volume) range: about $10^9$ (huge!)
- Sensitivity to volume (amplitude) drops off at ends of range

* Hz stands for Hertz, meaning cycles per second
Pressure varies continuously—sampling measures how much pressure at fixed intervals.

Accuracy determined by:
- Sampling rate
- Sample size

**Sampling rate**: how many times per second do we measure?

**Sample size**: how many bits do we store per sample?
Sampling
When Sampling Is Too Slow

Figure 5.7: Sampling a sinusoid at too slow of a rate.

Samples Must Have Enough Bits

Figure 5.11: Quantized versions of an analog signal.

High-Quality Sampling

- Rate: 44,100 samples per second (Hertz – Hz).
  - sampling theorem: the sampling rate must be at least twice the highest frequency in the sound (humans can hear up to approx. 20,000 Hz.)

- Sample size: 16-bits per sample (so there are 65,536 amplitude levels that can be measured).
  - Quantization (rounding to integer sample values) introduces noise. Adding one bit cuts the noise in half.
sound file formats
Compressing Sound Files

- **codecs** (compression/decompression) implement various compression/decompression techniques

- **Lossless**: WMA Lossless, ALAC, MPEG-4 ALS, ...

- **Lossy**: MPEG, like JPEG, a family of perceptually-based techniques
MP3

- MP3 (MPEG3) is a **lossy** compression technique.
- Takes advantage of human perception (**psychoacoustics**)
  - Our hearing is better in mid range frequencies than on the low and high ends.
  - If a loud and soft sound play at about the same time or about the same frequencies, we can’t hear the soft sound: this is called **masking**
  - *Masking can hide noise introduced by compression.*
MP3 Demo

Let Me Call You Sweetheart

**MP3 Compression**

- Like JPEG, MP3 has various levels of compression:

<table>
<thead>
<tr>
<th>Bit Rate</th>
<th>Compression Ratio</th>
<th>Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>256Kbps</td>
<td>5:1</td>
<td>Supreme (near best)</td>
</tr>
<tr>
<td>192Kbps</td>
<td>7:1</td>
<td>Excellent (better)</td>
</tr>
<tr>
<td>128Kbps</td>
<td>11:1</td>
<td>(good)</td>
</tr>
<tr>
<td>96Kbps</td>
<td>19:1</td>
<td>(fair)</td>
</tr>
<tr>
<td>64Kbps</td>
<td>22:1</td>
<td>FM quality (poor)</td>
</tr>
</tbody>
</table>

- MP3 also has Variable Bit Rate (VBR) since compression ability can vary at different segments of the digital recording.
image + sound = video
Problem: a torrent of data

- Imagine if we used “raw” images and sound for video
  - about 5MB of image data per frame, times 30 frames/sec = about 150 MB image data per second
  - about 1400 kbps, or 175 KB sound data per second
  - 10 minutes of this: about 90.1 Gigabytes
MP4

- MP4 (MPEG4): compression technique for video
- Sophisticated engineering exploits
  - *redundancy* (next frame is likely to resemble this frame)
  - *perception* (what the eye and ear can do)
- Applications: streaming, HDTV broadcast, Digital Cinema, cameras (e.g. GoPro), phones
YouTube, Vimeo, etc.

- YouTube, Vimeo, etc. support many formats, including MP4, AVI (Microsoft), QuickTime (Apple), and Flash (Adobe).
- You can download videos from these sites in your preferred format using tools such as KeepVid.
- Uploading and then downloading a video may reduce the quality due to lossy compression.
Summary

- **Samples**
  - **Pixels** are samples of the image in space; *resolution* and *number of bits* determine quality
  - **Audio samples** measure the signal in time; *sampling rate* and *number of bits* determine quality

- **Tradeoff** between quality and size

- **Compression** methods exploit
  - Coding redundancy (e.g. Huffman codes)
  - Data redundancy (e.g. run-length coding)
  - Perceptual redundancy (e.g. MP3, JPEG)