Image Processing

Adrien Treuille
Overview

Image Types

Pixel Filters

Neighborhood Filters

Dithering

Compression
Overview

Image Types

Pixel Filters

Neighborhood Filters

Dithering

Compression
Images

- Image stored in memory as 2D pixel array
- Value of each pixel controls color
- **Depth** of image is information per pixel
  - 1 bit: black and white display
  - 8 bit: 256 colors at any given time via colormap
  - 16 bit: 5, 6, 5 bits (R,G,B), $2^{16} = 65,536$ colors
  - 24 bit: 8, 8, 8 bits (R,G,B), $2^{24} = 16,777,216$ colors
Fewer Bits: Colormaps

- Colormaps typical for 8 bit framebuffer depth
- With screen 1024 * 768 = 786432 = 0.75 MB
- Each pixel value is index into colormap
- Colormap is an array of RGB values, 8 bits each
- Only $2^8 = 256$ at a time
- Poor approximation of full color
Overview

Image Types

Pixel Filters

Neighborhood Filters

Dithering

Compression
Overview

Image Types

Pixel Filters

Neighborhood Filters

Dithering

Compression
Pixel Operations
Point Processing

Original

Darken

Lower Contrast

Nonlinear Lower Contrast

Invert

Lighten

Raise Contrast

Nonlinear Raise Contrast
Point Processing

- **Original**: x
- **Darken**: x - 128
- **Lower Contrast**: x / 2
- **Nonlinear Lower Contrast**: \(((x / 255.0) ** 0.33) * 255.0\)
- **Invert**: 255 - x
- **Lighten**: x + 128
- **Raise Contrast**: x * 2
- **Nonlinear Raise Contrast**: \(((x / 255.0) ** 2) * 255.0\)
Gamma correction

Monitors have a intensity to voltage response curve which is roughly a 2.5 power function.

Send $v \rightarrow$ actually display a pixel which has intensity equal to $v^{2.5}$

\[ \Gamma = 1.0; \quad f(v) = v \]

\[ \Gamma = 2.5; \quad f(v) = v^{1/2.5} = v^{0.4} \]
Overview

Image Types

Pixel Filters

Neighborhood Filters

Dithering

Compression
Overview

Image Types

Pixel Filters

Neighborhood Filters

Dithering

Compression
Neighborhood Operations
Convolution

\[
F = \begin{bmatrix}
0.2 & 0.1 & -1.0 \\
0.3 & 0.0 & 0.9 \\
0.1 & 0.3 & -1.0
\end{bmatrix}
\]

\[I' = F \ast I\]
Convolutions are Linear

$$F \ast I + G \ast I = (F + G) \ast I$$

$$2F \ast I = F \ast 2I = 2(F \ast I)$$

(We will use this fact when we talk about sharpening filters.)
Original Image
Shifted Image
Original Image
X-Edge Detection
Y-Edge Detection
General Edge Detection

Can this be described as a convolution?
Blurred Image
Blurring Filters

• A simple blurring effect can be achieved with a 3x3 filter centered around a pixel,
• More blurring is achieved with a wider $n \times n$ filter:
Image Filtering: Blurring

original, 64x64 pixels

3x3 blur

5x5 blur
Blurred Image
Sharpened Image
Noise
Blurred Noise
Median Filter

Can this be described as a convolution?
Example: Noise Reduction

Image with noise  Median filter (5x5)
Example: Noise Reduction

Tom Ridge left the Pennsylvania governorship last October, when U.S. President George W. Bush appointed him to head the newly created Office of Homeland Security.

Original image

Image with noise

Median filter (5x5)
Warp Filter
Original Image
Warped Image
Warped Image

orig + vector field = warped

how?
Advection (just like a fluid)
Image Morphing
Warp + Crossfade

- forward warp
- crossfade
- backwards warp

Result
Warp Example
Overview

Image Types

Pixel Filters

Neighborhood Filters

Dithering

Compression
Overview

Image Types

Pixel Filters

Neighborhood Filters

Dithering

Compression
Dithering

• Compensates for lack of color resolution
• Eye does spatial averaging
• Black/white dithering to achieve gray scale
  – Each pixel is black or white
  – From far away, color determined by fraction of white
  – For 3x3 block, 10 levels of gray scale
Dithering takes advantage of the human eye's tendency to "mix" two colors in close proximity to one another.
Dithering takes advantage of the human eye's tendency to "mix" two colors in close proximity to one another.

**Original**
Colors = $2^{24}$

**No Dithering**
Colors = $2^{8}$

**With Dithering**
Colors = $2^{8}$
Ordered Dithering

• How do we select a good set of patterns?
• Regular patterns create some artifacts
• Example of good 3x3 dithering matrix

\[
\begin{bmatrix}
6 & 8 & 4 \\
1 & 0 & 3 \\
5 & 2 & 7 \\
\end{bmatrix}
\]
Floyd-Steinberg Error Diffusion

- Diffuse the quantization error of a pixel to its neighboring pixels
- Scan in raster order
- At each pixel, draw least error output value
- Add the error fractions into adjacent, unwritten pixels

- If a number of pixels have been rounded downwards, it becomes more likely that the next pixel is rounded upwards

```
for each y
    for each x
        oldpixel := pixel[x][y]
        newpixel := find_closest_palette_color(oldpixel)
        pixel[x][y] := newpixel
        quant_error := oldpixel - newpixel
        pixel[x+1][y] := pixel[x+1][y] + 7/16 * quant_error
        pixel[x-1][y+1] := pixel[x-1][y+1] + 3/16 * quant_error
        pixel[x][y+1] := pixel[x][y+1] + 5/16 * quant_error
        pixel[x+1][y+1] := pixel[x+1][y+1] + 1/16 * quant_error
```
Floyd-Steinberg Error Diffusion
Floyd-Steinberg Error Diffusion

Enhances edges
Retains high frequency
Some checkerboarding

From http://www.cs.rit.edu/~pga/pics2000/node1.html
Color Dithering

• Example: 8 bit framebuffer
  – Set color map by dividing 8 bits into 3,3,2 for RGB
  – Blue is deemphasized because we see it less well

• Dither RGB separately
  – Works well with Floyd-Steinberg

• Generally looks good
Overview

Image Types

Pixel Filters

Neighborhood Filters

Dithering

Compression
Overview

Image Types

Pixel Filters

Neighborhood Filters

Dithering

Compression
Image Compression

• Exploit redundancy
  – Coding: some pixel values more common
  – Interpixel: adjacent pixels often similar
  – Psychovisual: some color differences imperceptible

• Distinguish lossy and lossless methods
Image Sizes

• 1024*1024 at 24 bits uses 3 MB

• Encyclopedia Britannica at 300 pixels/inch and 1 bit/pixel requires 25 gigabytes (25K pages)

• 90 minute movie at 640x480, 24 bits per pixel, 24 frames per second requires 120 gigabytes

• Applications: HDTV, DVD, satellite image transmission, medical image processing, fax, ...
Exploiting Coding Redundancy

• Not limited to images (text, other digital info)
• Exploit nonuniform probabilities of symbols
• Entropy as measure of information content
  - $H = -\sum_i \text{Prob}(s_i) \log_2 (\text{Prob}(s_i))$
  - Low entropy $\rightarrow$ non uniform probability
  - High entropy $\rightarrow$ uniform probability
  - If source is independent random variable need $H$ bits
Exploiting Coding Redundancy

• Idea:
  – More frequent symbols get shorter code strings
  – Best with high redundancy (= low entropy)

• Common algorithms
  – Huffman coding
  – LZW coding (gzip)

<table>
<thead>
<tr>
<th>Symbol (a&lt;sub&gt;i&lt;/sub&gt;)</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weights (w&lt;sub&gt;i&lt;/sub&gt;)</td>
<td>0.10</td>
<td>0.15</td>
<td>0.30</td>
<td>0.16</td>
<td>0.29</td>
<td>= 1</td>
</tr>
<tr>
<td>Codewords (c&lt;sub&gt;i&lt;/sub&gt;)</td>
<td>000</td>
<td>001</td>
<td>10</td>
<td>01</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>Codeword length (in bits) (l&lt;sub&gt;i&lt;/sub&gt;)</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
Huffman Coding

• Codebook is precomputed and static
  – Use probability of each symbol to assign code
  – Map symbol to code
  – Store codebook and code sequence

• Precomputation is expensive

lossless

• What is “symbol” for image compression?
Exploiting Interpixel Redundancy

• Neighboring pixels are correlated
• Spatial methods for low-noise image
  – Run-length coding:
    » Alternate values and run-length
    » Good if horizontal neighbors are same
    » Can be 1D or 2D (e.g. used in fax standard)
    » WWWWBWWWBBWWW
    » 12W 1B 12W 3B 24W 1B 14W
  – Quadtrees:
    » Recursively subdivide until cells are constant color
  – Region encoding:
    » Represent boundary curves of color-constant regions

lossless
Improving Noise Tolerance

• Predictive coding:
  – Predict next pixel based on prior ones
  – Output difference to actual

• Transform coding
  – Exploit frequency domain
  – Example: discrete cosine transform (DCT)
  – Used in JPEG

lossy compression
Discrete Cosine Transform

- Used for lossy compression (as in JPEG)

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

where \( c(u) = 1/\sqrt{n} \) if \( u = 0 \), \( c(u) = \sqrt{2/n} \) otherwise

- Subdivide image into \( n \times n \) blocks (\( n = 8 \))
- Apply discrete cosine transform for each block
- Each tile is converted to frequency space

\[
\begin{bmatrix}
52 & 55 & 61 & 66 & 70 & 61 & 64 & 73 \\
63 & 59 & 55 & 90 & 109 & 85 & 69 & 72 \\
62 & 59 & 68 & 113 & 144 & 104 & 66 & 73 \\
63 & 58 & 71 & 122 & 154 & 106 & 70 & 69 \\
67 & 61 & 68 & 104 & 126 & 88 & 68 & 70 \\
79 & 65 & 60 & 70 & 77 & 68 & 58 & 75 \\
85 & 71 & 64 & 59 & 55 & 61 & 65 & 83 \\
87 & 79 & 69 & 68 & 65 & 76 & 78 & 94
\end{bmatrix}
\]

\[
\begin{bmatrix}
-415 & -30 & -61 & 27 & 56 & -20 & -2 & 0 \\
4 & -22 & -61 & 10 & 13 & -7 & -9 & 5 \\
-47 & 7 & 77 & -25 & -29 & 10 & 5 & -6 \\
-49 & 12 & 34 & -15 & -10 & 6 & 2 & 2 \\
12 & -7 & -13 & -4 & -2 & 2 & -3 & 3 \\
-8 & 3 & 2 & -6 & -2 & 1 & 4 & 2 \\
-1 & 0 & 0 & -2 & -1 & -3 & 4 & -1 \\
0 & 0 & -1 & -4 & -1 & 0 & 1 & 2
\end{bmatrix}
\]
Discrete Cosine Transform

• Quantize
  – Human eye good at seeing variations over large area
  – Not good at seeing the exact strength of a high frequency
  – Greatly reducing the amount of information in the high frequency components

\[
\text{round}\left(\frac{-415}{16}\right) = \text{round}(-25.9375) = -26
\]

\[
\begin{bmatrix}
16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\
12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\
14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\
14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\
18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\
24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\
49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\
72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
-415 & -30 & -61 & 27 & 56 & -20 & -2 & 0 \\
4 & -22 & -61 & 10 & 13 & -7 & -9 & 5 \\
-47 & 7 & 77 & -25 & -29 & 10 & 5 & -6 \\
-49 & 12 & 34 & -15 & -10 & 6 & 2 & 2 \\
12 & -7 & -13 & -4 & -2 & 2 & -3 & 3 \\
-8 & 3 & 2 & -6 & -2 & 1 & 4 & 2 \\
-1 & 0 & 0 & -2 & -1 & -3 & 4 & -1 \\
0 & 0 & -1 & -4 & -1 & 0 & 1 & 2 \\
\end{bmatrix}
\]

• Use variable length coding (e.g. Huffman)
Overview

Image Types

Pixel Filters

Neighborhood Filters

Dithering

Compression
Overview

Image Types

Pixel Filters

Neighborhood Filters

Dithering

Compression