Image Processing

Adrien Treuille
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 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

Invert

Lighten

Raise Contrast
Point Processing

- Original
- Darken
- Lower Contrast
- Invert
- Lighten
- Raise Contrast

Formulas:

- $255 - x$
Point Processing

Original: $x$

Darken: $x - 128$

Lower Contrast: $x / 2$

Invert: $255 - x$

Lighten: $x + 128$

Raise Contrast: $x * 2$
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.)
Shifted Image
X-Edge Detection
Y-Edge Detection
General Edge Detection

Can this be described as a convolution?
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
Noise
Blurred Noise
Median Filter

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

Image with noise

Median filter (5x5)
Example: Noise Reduction

Original image

Image with noise

Median filter (5x5)
Warp Filter
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 = $2^8$
Dithering takes advantage of the human eye's tendency to "mix" two colors in close proximity to one another.

How could we do this?
How Could We Do This?

- Deterministic Thresholding
- Random Thresholding
- Threshold Patterns
  - Dithering Matrices
- Diffusion
Deterministic Thresholding

Random Thresholding

Dithering Matrices

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

```
6 8 4
1 0 3
5 2 7
```
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

```c
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][y+1] := pixel[x][y+1] + 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

How could this be improved?

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 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, medial image processing, fax, ...
Types of Compression

- Coding Redundancy
  - Huffman Coding (lossless)

- Spatial Coherence
  - Run Length Encoding (lossless)

- Psycho visual
  - JPEG Encoding (lossy)
Types of Compression

- Coding Redundancy
  - Huffman Coding (lossless)
- Spatial Coherence
  - Run Length Encoding (lossless)
- Psycho visual
  - JPEG Encoding (lossy)
Suppose we have the following 4 colors:

| 00 | 01 | 10 | 11 |

As used in this image:

| 00 | 00 | 10 | 10 | 10 | 10 |
| 10 | 10 | 10 | 10 | 01 | 10 |
| 10 | 10 | 11 | 10 | 11 | 00 |

Switch to this encoding:

| 000 | 001 | 1 | 010 |

Which is equivalent to:

| 000 | 000 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 001 | 1 |
| 1 | 1 | 010 | 1 | 010 | 000 |

Binary String (36 bits):

00001010101010101010100110101011101100

Binary String (28 bits):

00011111111111000011110101010000
Huffman Coding

Suppose we have the following 4 colors:

00  01  10  11

As used in this image:

00  00  10  10  10  10
10  10  10  10  10  10
10  10  10  10  10  10
00  01  10  11

Switch to this encoding:

Binary String (36 bits):
0000101010101010100110101011101100

Huffman Codes
provide the optimal answer to encoding such a representation.

Binary String (28 bits):
000111111111000011110101010100

000  00  10  10  10  10
10  10  10  10  10  10
10  10  10  10  10  10
00  01  10  11
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
Types of Compression

- Coding Redundancy
  - Huffman Coding (lossless)

- Spatial Coherence
  - Run Length Encoding (lossless)

- Psycho visual
  - JPEG Encoding (lossy)
Types of Compression

- Coding Redundancy
  - Huffman Coding (lossless)

- Spatial Coherence
  - Run Length Encoding (lossless)

- Psycho visual
  - JPEG Encoding (lossy)
Run Length Encoding

Same Image As Before:

Scan Convert:

Encode:
Run Length Encoding

<table>
<thead>
<tr>
<th>00</th>
<th>00</th>
<th>10</th>
<th>10</th>
<th>10</th>
<th>10</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

Same Image As Before:

<table>
<thead>
<tr>
<th>00</th>
<th>01</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>

Scan Convert:

| 2×00 | 8×10 | 1×01 | 3×10 | 1×11 | 1×10 | 1×11 | 1×00 |

Encode:

<table>
<thead>
<tr>
<th>01</th>
</tr>
</thead>
</table>

Related Ideas:

- **Quadtrees**: Recursively subdivide until cells are constant color.
- **Region Encoding**: Represent boundary curves of constant-color regions.
Types of Compression

- Coding Redundancy
  - Huffman Coding (lossless)

- Spatial Coherence
  - Run Length Encoding (lossless)

- Psycho visual
  - JPEG Encoding (lossy)
Types of Compression

- Coding Redundancy
  - Huffman Coding (lossless)

- Spatial Coherence
  - Run Length Encoding (lossless)

- Psycho visual
  - JPEG Encoding (lossy)
JPEG Compression

Divide image into 8x8 blocks.
• Express each block as a linear combination of 8x8 basis blocks made of cosines.
• This is called the discrete cosine transform.
Key Insight!

Upper left blocks have higher values than lower right? (They are more important.)

Why?

How can we exploit this insight?
Scaled Coefficients

\[
\begin{bmatrix}
-415 & -30 & -61 & 27 & 56 & -20 & -2 & 0 \\
4 & -22 & -61 & 10 & 13 & -7 & -9 & 5 \\
-47 & 7 & 77 & -25 & -29 & 10 & 5 & -6 \\
-49 & 12 & 34 & -15 & -10 & 6 & 2 & 2 \\
12 & -7 & -13 & -4 & -2 & 2 & -3 & 3 \\
-8 & 3 & 2 & -6 & -2 & 1 & 4 & 2 \\
-1 & 0 & 0 & -2 & -1 & -3 & 4 & -1 \\
0 & 0 & -1 & -4 & -1 & 0 & 1 & 2 \\
\end{bmatrix}
\div
\begin{bmatrix}
16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\
12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\
14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\
14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\
18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\
24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\
49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\
72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \\
\end{bmatrix}
= \begin{bmatrix}
-26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\
0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\
-3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\
-1 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

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

• What can we see about the quantization matrix?
• How can we compress the scaled coefficients?

Answer:
Run Length + Huffman Coding
Types of Compression

- **Coding Redundancy**
  - Huffman Coding (lossless)

- **Spatial Coherence**
  - Run Length Encoding (lossless)

- **Psycho visual**
  - JPEG Encoding (lossy)
Overview

Image Types

Pixel Filters

Neighborhood Filters

Dithering

Compression