Image Processing

Blending
- Simple color model: R, G, B; 8 bits each
- $\alpha$-channel A, another 8 bits
- Alpha determines opacity, pixel-by-pixel
  - $\alpha = 1$: opaque
  - $\alpha = 0$: transparent

Alpha Channel

Image Compositing
- Compositing operation
  - Source: $s = [s_r, s_g, s_b, s_a]$  
  - Destination: $d = [d_r, d_g, d_b, d_a]$  
  - $b = [b_r, b_g, b_b]$ source blending factors
  - $c = [c_r, c_g, c_b, c_a]$ destination blending factors
  - $d' = [b_r s_r + c_r d_r, b_g s_g + c_g d_g, b_b s_b + c_b d_b, b_a s_a + c_a d_a + c_a d_a]$  
- Example: overlay $n$ images with equal weight
  - Set $\alpha$-value for each pixel in each image to $1/n$
  - Source blending factor is $\alpha$
  - Destination blending factor is "1"

Blending in OpenGL
- Enable blending
  - glEnable(GL_BLEND);
- Set up source and destination factors
  - glBlendFunc(source_factor, dest_factor);
- Source and destination choices
  - GL_ONE, GL_ZERO
  - GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA
  - GL_DST_ALPHA, GL_ONE_MINUS_DST_ALPHA
- Set alpha values: 4th parameter to
  - glColor4f(r, g, b, alpha)
  - glLightfv, glMaterialfv

Blending Errors
- Operations are not commutative
  - rendering order changes result
- Operations are not idempotent
  - render same object twice gives different result to rendering once
- Interaction with hidden-surface removal is tricky
  - Polygon behind opaque polygon(s) should be culled
  - Transparent in front of others should be composited
  - Solution: make z-buffer read-only for transparent polygons with
    - glEnableDepthMask(GL_FALSE);
Displays and Framebuffers

- Image stored in memory as 2D pixel array, called framebuffer
- Value of each pixel controls color
- Video hardware scans the framebuffer at 60Hz
- Depth of framebuffer 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

- Colormap is array of RGB values, k bits each (e.g., k=8)
- Each pixel not the color, but an index into colormap
- All $2^k$ colors can be represented, but only $2^k$ colors at a time
- Poor approximation of full color
- Colormap hacks: affect image without changing framebuffer (only colormap)

More Bits: Graphics Hardware

- 24 bits: RGB
- + 8 bits: A ($\alpha$-channel for opacity)
- + 16 bits: Z (for hidden-surface removal)
- * 2: double buffering for smooth animation
- = 96 bits
- For 1024 * 768 screen: 9 MB
- Easily possible on modern hardware

Image Processing

- 2D generalization of signal processing
- Image as a two-dimensional signal
- Point processing: modify pixels independently
- Filtering: modify based on neighborhood
- Compositing: combine several images
- Image compression: space-efficient formats
- Other topics
  - Image enhancement and restoration
  - Computer vision

Outline

- Blending
- Display Color Models
- Filters
- Dithering
**Point Processing**

- Process each pixel independently from others
- Input: \(a(x,y)\); Output: \(b(x,y) = f(a(x,y))\)
- Useful for contrast adjustment, false colors
- Examples for grayscale, \(0 \leq v \leq 1\)
  - \(f(v) = v\) (identity)
  - \(f(v) = 1-v\) (negate image)
  - \(f(v) = 2v\), \(p < 1\) (brighten)
  - \(f(v) = \frac{v}{p}\), \(p > 1\) (darken)

**Gamma Correction**

- Example of point processing
- Compensates monitor brightness nonlinearities (older monitors)

\[
\Gamma = 1.0; \quad f(v) = v \\
\Gamma = 0.5; \quad f(v) = \frac{1}{1/2} = v^2 \\
\Gamma = 2.5; \quad f(v) = \frac{1}{1/2.5} = v^{0.4}
\]

**Signals and Filtering**

- Audio recording is 1D signal: amplitude\((t)\)
- Image is a 2D signal: color\((x,y)\)
- Signals can be continuous or discrete
- Raster images are discrete
  - In space: sampled in \(x, y\)
  - In color: quantized in value
- Filtering: a mapping from signal to signal

**Linear and Shift-Invariant Filters**

- Linear with respect to input signal
- Shift-invariant with respect to parameter
- Convolution in 1D
  - \(a(t)\) is input signal
  - \(b(s)\) is output signal
  - \(h(u)\) is filter
- Convolution in 2D

\[
b(x,y) = \sum_{t=\infty}^{\infty} \sum_{t=\infty}^{\infty} a(u,v)h(x-u,y-v)
\]

**Filters with Finite Support**

- Filter \(h(u,v)\) is 0 except in given region
- Example: 3 x 3 blurring filter

\[
b(x,y) = \frac{1}{9} \left( a(x-1,y-1) + a(x,y-1) + a(x+1,y-1) \\
+ a(x-1,y) + a(x,y) + a(x+1,y) \\
+ a(x-1,y+1) + a(x,y+1) + a(x+1,y+1) \right)
\]

- As function

\[
h(u,v) = \begin{cases} \frac{1}{9} & \text{if } -1 \leq u, v \leq 1 \\ 0 & \text{otherwise} \end{cases}
\]

- In matrix form

\[
\begin{bmatrix}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1
\end{bmatrix}
\]

**Blurring Filters**

- Average values of surrounding pixels
- Can be used for anti-aliasing
- Size of blurring filter should be odd
- What do we do at the edges and corners?
- For noise reduction, use median, not average
  - Eliminates intensity spikes
  - Non-linear filter
Examples of Blurring Filter

- Original Image
- Blur 3x3 mask
- Blur 7x7 mask

Example Noise Reduction

- Original image
- Image with noise
- Median filter (5x5)

Edge Filters

- Task: Discover edges in image
- Characterized by large gradient
  \[ \nabla a = \left[ \frac{\partial a}{\partial x}, \frac{\partial a}{\partial y} \right] \]
  \[ |\nabla a| = \sqrt{\left(\frac{\partial a}{\partial x}\right)^2 + \left(\frac{\partial a}{\partial y}\right)^2} \]
- Approximate square root
  \[ |\nabla a| \approx \frac{\partial a}{\partial x} + \frac{\partial a}{\partial y} \]
- Approximate partial derivatives, e.g.
  \[ \frac{\partial a}{\partial x} \approx a(x+1) - a(x-1) \]

Sobel Filter

- Very popular edge detection filter
- Approximate:
  \[ \frac{\partial a}{\partial x} \approx \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad \frac{\partial a}{\partial y} \approx \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \]
- Output is |\nabla a|, computed as follows:
  \[ |\nabla a| = \sqrt{\left(\frac{\partial a}{\partial x}\right)^2 + \left(\frac{\partial a}{\partial y}\right)^2} \]
- Sobel filter is non-linear
  - Square and square root (more exact computation)
  - Can also use absolute value (faster computation)

Sample Filter Computation

- One part (of the two) of the Sobel filter
- Detects vertical edges

Example of Edge Filter

- Original image
- Edge filter, then brightened
Outline

- Blending
- Display Color Models
- Filters
- Dithering

Dithering

- Compensates for lack of color resolution
- Give up spatial resolution for color resolution
- Eye does spatial averaging

Black/White Dithering

- For gray scale images
- Each pixel is black or white
- From far away, eye perceives color by fraction of white
- For 3x3 block, 10 levels of gray scale

Color Dithering

- Dither RGB separately
- Store quantized color as a k-bit value (often k=8)

Halftoning

- Create grayscale images using properly positioned sized dots
- Regular patterns create artifacts
  - Avoid stripes
  - Avoid isolated pixels (e.g. on laser printer)
  - Monotonicity: keep pixels on at higher intensities
  - Floyd-Steinberg dithering
- Example of good 3x3 dithering matrix
  - For intensity n, turn on pixels 0...n-1

Summary

- Display Color Models
  - 8 bit (colormap), 24 bit, 96 bit
- Filters
  - Blur, edge detect, sharpen, despeckle (noise removal)
- Dithering