Announcements

Questions on Assignment 4?
Assignment 3 grades
Movie from Assignment 3

11/17/02

Image Processing

Point Processing
Filters
Dithering
Image Compression
Image Compositing

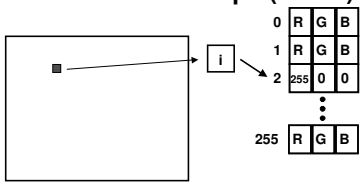
Displays and Framebuffers

- 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 (cf. Smithsonian)
  - 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
- All 2^{24} colors can be represented
- Only 2^8 = 256 at a time
- Poor approximation of full color
- Who owns the colormap?
- Colormap hacks: affect image w/o changing framebuffer (only colormap)

Color maps (LUT's)



- A single number (e.g. 8 bits) stored at each pixel.
- Used as an *index* into an array of RGB triples.
- With 8 bits per pixel, you get 256 colors of your choice
- Simple things to fill up color-maps with:
  - A grey ramp (for grey scale pictures)
  - A bunch of random colors (for color drawings.)
  - A very poor representation of full color

More Bits: Graphics Hardware

- 24 bits: RGB
- + 8 bits: A (a-channel for opacity)
- + 16 bits: Z (for hidden-surface removal)
- * 2: double buffering for smooth animation
- = 96 bits
- For 1024 * 768 screen: 9 MB

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
- Related topics (not in this lecture or this course)
  - Image enhancement and restoration
  - Complicated special effects
  - Computer vision

Outline

- Point Processing
- Filters
- Dithering
- Image Compositing

Point Processing

- **Input: a[x,y], Output b[x,y] = f(a[x,y])**
- **f transforms each pixel value separately**
- **Useful for contrast adjustment**

Suppose our picture is grayscale (a.k.a. monochrome).
Let *v* denote pixel value, suppose it's in the range [0,1].

f(v) = v **identity**; no change

f(v) = 1-v **negate** an image (black to white, white to black)

f(v) = v^p, p<1 **brighten**

f(v) = v^p, p>1 **darken**

Point Processing

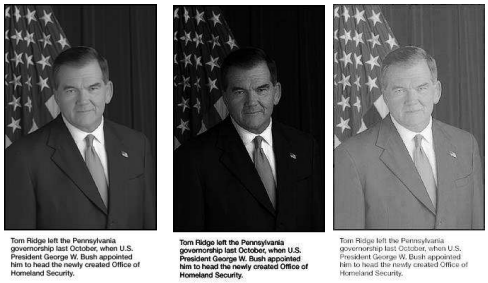
f(v) = v identity; no change

f(v) = 1-v negate an image (black to white, white to black)

f(v) = v^p, p<1 brighten

f(v) = v^p, p>1 darken

Gamma correction compensates for different monitors



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.

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.

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.

$\Gamma = 1.0$; $f(v) = v$ $\Gamma = 0.5$; $f(v) = v^{1/0.5} = v^2$ $\Gamma = 2.5$; $f(v) = v^{1/2.5} = v^{0.4}$

Outline

- Point Processing
- Filters
- Dithering
- Image Compositing

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

$$b(s) = \sum_{t=-\infty}^{+\infty} a(t)h(s-t)$$

- Convolution in 2D

$$b(x,y) = \sum_{u=-\infty}^{+\infty} \sum_{v=-\infty}^{+\infty} a(u,v)h(x-u,y-v)$$

Filters with Finite Support

- Filter h(u,v) is 0 except in given region
- Represent h in form of a matrix
- Example: 3 x 3 blurring filter

$$b(x,y) = \frac{1}{9}\begin{array}{lll} (a(x-1,y-1) & +a(x,y-1) & +a(x+1,y-1) \\ +a(x-1,y) & +a(x,y) & +a(x+1,y) \\ +a(x-1,y+1) & +a(x,y+1) & +a(x+1,y+1)) \end{array}$$

- In matrix form

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

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

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*×*n* filter:

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

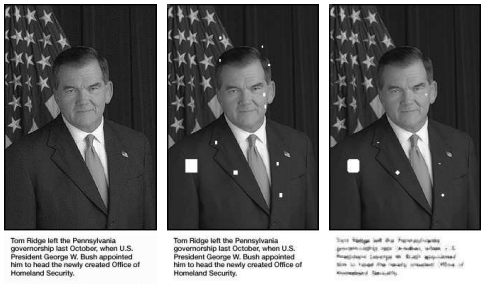
Image Filtering: Blurring

original, 64x64 pixels | 3x3 blur | 5x5 blur

Blurring Filters

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

Example: Noise Reduction



Original image | Image with noise | Median filter (5x5?)

Edge Filters

- Discover edges in image
- Characterized by large gradient
- Approximate square root

$$\nabla a = \left[\frac{\partial a}{\partial x} \quad \frac{\partial a}{\partial y}\right], \quad |\nabla a| = \sqrt{\left(\frac{\partial a}{\partial x}\right)^2 + \left(\frac{\partial a}{\partial y}\right)^2}$$

- Approximate partial derivatives, e.g.

$$|\nabla a| \approx |\frac{\partial a}{\partial x}| + |\frac{\partial a}{\partial y}|$$

$$\frac{\partial a}{\partial x} \approx a(x+1) - a(x-1)$$

Sobel Filter

- Edge detection filter, with some smoothing
- Approximate

$$\frac{\partial}{\partial x} \approx \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad \frac{\partial}{\partial y} \approx \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

- Sobel filter is non-linear
  - Square and square root (more exact computation)
  - Absolute value (faster computation)

Sample Filter Computation

- Part of Sobel filter, detects vertical edges

$\frac{1}{4}$

-1	0	1
-2	0	2
-1	0	1

h

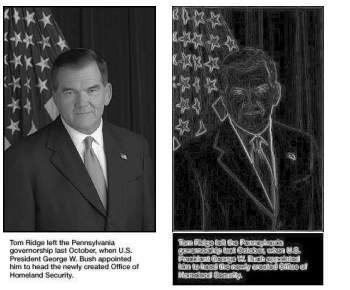
0	0	0	0	0	25	25	25	25	25
0	0	0	0	0	25	25	25	25	25
0	0	0	0	0	25	25	25	25	25
0	0	0	0	0	25	25	25	25	25
0	0	0	0	0	25	25	25	25	25
0	0	0	0	0	25	25	25	25	25
0	0	0	0	0	25	25	25	25	25
0	0	0	0	0	25	25	25	25	25
0	0	0	0	0	25	25	25	25	25
0	0	0	0	0	25	25	25	25	25

a

0	0	0	0	0	25	25	0	0	0	0
0	0	0	0	0	25	25	0	0	0	0
0	0	0	0	0	25	25	0	0	0	0
0	0	0	0	0	25	25	0	0	0	0
0	0	0	0	0	25	25	0	0	0	0
0	0	0	0	0	25	25	0	0	0	0
0	0	0	0	0	25	25	0	0	0	0
0	0	0	0	0	25	25	0	0	0	0
0	0	0	0	0	25	25	0	0	0	0
0	0	0	0	0	25	25	0	0	0	0

b

Example of Edge Filter



Original image | Edge filter, then brightened

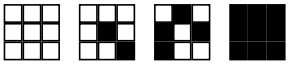
Image Filtering: Edge Detection

Outline

- Display Color Models
- Filters
- Dithering
- Image Compositing

Dithering

- Compensates for lack of color resolution
- Give up spatial resolution for 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



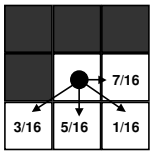
Halftone Screens

- How do we select a good set of patterns?
- Regular patterns create some artifacts
  - Avoid stripes
  - Avoid isolated pixels (e.g. on laser printer)
  - Monotonicity: keep pixels on at higher intensities
- Example of good 3x3 dithering matrix
  - For intensity n, turn on pixels 0..n–1

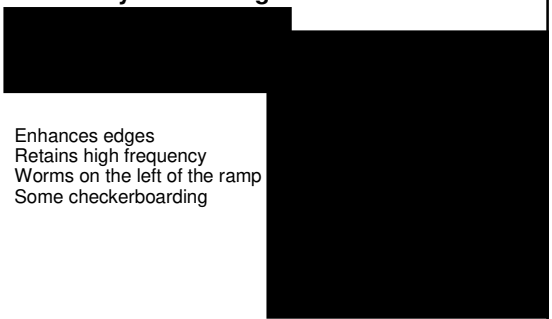
$$\begin{bmatrix} 6 & 8 & 4 \\ 1 & 0 & 3 \\ 5 & 2 & 7 \end{bmatrix}$$

Floyd-Steinberg Error Diffusion

- Approximation without fixed resolution loss
- Scan in raster order
- At each pixel, draw least error output value
- Divide error into 4 different fractions
- Add the error fractions into adjacent, unwritten pixels



Floyd-Steinberg Error Diffusion



Enhances edges
Retains high frequency
Worms on the left of the ramp
Some checkerboarding

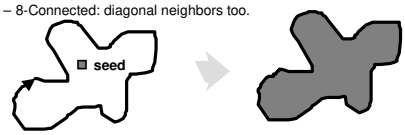
From http://www.cs.rit.edu/~pga/pics2000/node1.html

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
- Assemble results into 8 bit index into colormap
- Generally looks good

Flood Fill

- Fill in arbitrary regions by starting from a "seed" point, and propagating in all directions until all connected pixels are painted.
- Connected:
  - Flood fill: a path from seed to point all of whose pixels contain "OLDVALUE"
  - Boundary fill: a path *none* of whose pixels contain "BOUNDARYVALUE."
  - 4-Connected: path restricted to horizontal and vertical neighbors.
  - 8-Connected: diagonal neighbors too.



Recursive Flood Fill

- Start at a point, call recursively at all 4 (or 8) neighbors

```
void FloodFill4(x,y,Oldval, Newval){
   if(Pixel(x,y) == Oldval){
      SetPixel(x,y,Newval);
      FloodFill4(x,y+1,Oldval,Newval);
      FloodFill4(x,y-1,Oldval,Newval);
      FloodFill4(x+1,y,Oldval,Newval);
      FloodFill4(x-1,y,Oldval,Newval);
   }
}
```


Span-based Filling

- No recursion, more efficient.
- Span: a contiguous horizontal run of pixels, all containing OLDVALUE, bounded left and right by points that don't.
- Maintain a *stack* of spans. (can be denoted by their rightmost pixels.)

```
      Initially, Set current span to span
      containing seed.
   LOOP:
      Fill the current span.
      Go up one line.
         Find all spans connected to current span.
         Push them on the stack.
      Go down one line.  Same deal.
      Continue until stack is empty.
```

- There are many refinements and elaborations.

Color map hacks

- Reserve some of the bits in the *n*-bit index for special purposes, e.g. as "overlay planes." E.g. to scribble red lines over a grey scale image, set the color map as follows:

```
For i from 0 to 255
   if (i & 1) SetMap(i, 255, 0, 0) /* set entry i to red. */
   else SetMap(i,i,i,i) /* low bit is off, use greyscale. */
```

- To use, write the 8 bit image values into the framebuffer, but mask out the low order bit. Then draw red by inclusive-or'ing the pixel value with 1. Clearing the low bit brings back the appropriate grey.
- Similar: shadow hacks or transparent overlays.

Deeper Frame Buffers

- Some framebuffers have 96 or more bits per pixel. What are they all for? We start with 24 bits for RGB.
- Alpha channel: an extra 8 bits per pixel, to represent "transparency." Used for digital compositing. That's 32 bits.
- A Z-buffer, used to hold a "depth" value for each pixel. Used for hidden surface 3-D drawing. 16 bits/pixel of "z" brings the total to 48 bits.
- Double buffering:
  - For clean-looking flicker-free real time animation.
  - Two full framebuffers (including alpha and z).
  - Only one at a time is visible—you can toggle instantly.
  - Draw into the "back buffer" (invisible), then swap.
  - Can be faked with off-screen bitmaps (slower.)
  - 2 x 48 = 96.

Outline

- Display Color Models
- Filters
- Dithering
- Image Compositing

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

Some Image File Formats

	Depth	File Size	Comments
JPEG	24	Small	Lossy compression
TIFF	8, 24	Medium	Good general purpose
GIF	1, 4, 8	Medium	Popular, but 8 bit
PPM	24	Big	Easy to read/write
EPS	1,2,4,8,24	Huge	Good for printing

Image Sizes

- 1024*1024 at 24 bits uses 3 MB
- Encyclopedia Britannica at 300 pixels/inch and 1 bit/pixes requires 25 gigabytes (25K pages)
- 90 minute movie at 640x480, 24 bits per pixels, 24 frames per second requires 120 gigabytes
- Applications: HDTV, DVD, satellite image transmission, medial image processing, fax, ...
- Compression is still a real problem despite rate of growth of disk sizes

Exploiting Coding Redundancy

- Not limited to images (text, other digital info)
- Exploit nonuniform probabilities of symbols
- Entropy as measure of information content
  - $H = -S_i \text{ Prob}(s_i) \log_2 (\text{Prob}(s_i))$
  - If source is independent random variable need H bits
- Idea:
  - More frequent symbols get shorter code strings
  - Best with high redundancy (= low entropy)
- Common algorithms
  - Huffman coding
  - LZW coding (gzip)

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
- What is “symbol” for image compression?

Lempel-Ziv-Welch (LZW) Coding

- Compute codebook on the fly
- Fast compression and decompression
- Can tune various parameters
- Both Huffman and LZW are lossless

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)
  - Quadtrees:
    - » Recursively subdivide until cells are constant color
  - Region encoding:
    - » Represent boundary curves of color-constant regions
- Combine methods
- Not good on natural images directly

Improving Noise Tolerance

- Predictive coding:
  - Predict next pixel based on prior ones
  - Output difference to actual
- Fractal image compression
  - Describe image via recursive affine transformation
- Transform coding
  - Exploit frequency domain
  - Example: discrete cosine transform (DCT)
  - Used in JPEG
- Transform coding for lossy compression

Discrete Cosine Transform

- Used for lossy compression (as in JPEG)

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

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

- JPEG (Joint Photographic Expert Group)
  - Subdivide image into n x n blocks (n = 8)
  - Apply discrete cosine transform for each block
  - Quantize, zig-zag order, run-length code coefficients
  - Use variable length coding (e.g. Huffman)
- Many natural images can be compressed to 4 bits/pixels with little visible error

Outline

- Display Color Models
- Filters
- Dithering
- Image Compositing

Image Compositing

- Represent an image as layers that are composited (matted) together
- To support this, give image an extra *alpha* channel in addition to R, G, B
- Alpha is opacity: 0 if totally transparent, 1 if totally opaque
- Alpha is often stored as an 8 bit quantity; usually not displayed.
- Mathematically, to composite a_2 over a_1 according to matte α

  $b(x,y) = (1-\alpha(x,y))\cdot a_1(x,y) + \alpha(x,y)\cdot a_2(x,y)$

  α = 0 or 1 -- a hard matte, α = between 0 and 1 -- a soft matte
- Compositing is useful for photo retouching and special effects.

Special Effects: Compositing

- Lighting match
- Proper layering
- Contact with the real world
- Realism (perhaps)

Roger Rabbit

http://members.tripod.com/~Willy_Wonka/Theatr.jpg

Special Effects: Green Screen

Green screen
Match move of camera using markers
Relighting of original footage
Second green screen shot
Compositing of everything

Digital Domain (from http://www.vfxhq.com/1997/titanic-picssink.html)

Special Effects: Green Screen

Green screen
Match move of camera to match move with model of ship
Compositing of people with ship model, sky and digital water

Digital Domain (from http://www.vfxhq.com/1997/titanic-picssink.html)

Announcements

Questions on Assignment 4?
Assignment 3 grades
Movie from Assignment 3

11/17/02