15-462 Computer Graphics I

Lecture 15

Image Processing

Blending
Display Color Models
Filters
Dithering
Image Compression

March 18, 2003
Frank Pfenning
Carnegie Mellon University

http://www.cs.cmu.edu/~fp/courses/graphics/

Blending

- Frame buffer
  - Simple color model: R, G, B; 8 bits each
  - α-channel A, another 8 bits
- Alpha determines opacity, pixel-by-pixel
  - $\alpha = 1$: opaque
  - $\alpha = 0$: transparent
- Blend translucent objects during rendering
- Achieve other effects (e.g., shadows)

Image Compositing

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

Blending in OpenGL

- Enable blending

```
glEnable(GL_BLEND);
```

- Set up source and destination factors

```
glBlendFund(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`

Blending Errors

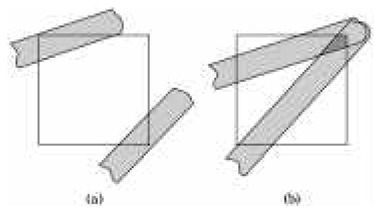
- Operations are not commutative
- Operations are not idempotent
- Interaction with hidden-surface removal
  - Polygon behind opaque one should be culled
  - Translucent in front of others should be composited
  - Solution: make z-buffer read-only for translucent polygons with `glDepthMask(GL_FALSE);`

Antialiasing Revisited

- Single-polygon case first
- Set α-value of each pixel to covered fraction
- Use destination factor of "$1 - \alpha$"
- Use source factor of "α"
- This will blend background with foreground
- Overlaps can lead to blending errors

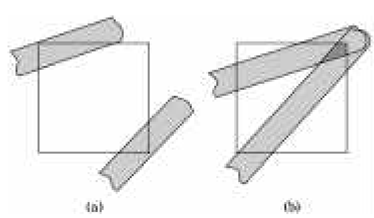
Antialiasing with Multiple Polygons

- Initially, background color $\mathbf{C}_0$, $\alpha_0 = 0$
- Render first polygon; color C_1 fraction α_1
  - $\mathbf{C}_d = (1 - \alpha_1)\mathbf{C}_0 + \alpha_1\mathbf{C}_1$
  - $\alpha_d = \alpha_1$
- Render second polygon; assume fraction α_2
- If no overlap (a), then
  - $\mathbf{C'}_d = (1 - \alpha_2)\mathbf{C}_d + \alpha_2\mathbf{C}_2$
  - $\alpha'_d = \alpha_1 + \alpha_2$



Antialiasing with Overlap

- Now assume overlap (b)
- Average overlap is $\alpha_1\alpha_2$
- So $\alpha_d = \alpha_1 + \alpha_2 - \alpha_1\alpha_2$
- Make front/back decision for color as usual



Antialiasing in OpenGL

- Avoid explicit α-calculation in program
- Enable both smoothing and blending

```
glEnable(GL_POINT_SMOOTH);
glEnable(GL_LINE_SMOOTH);
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```


Outline

- Blending
- Display Color Models
- Filters
- Dithering
- Image Compression

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)

More Bits: Graphics Hardware

- 24 bits: RGB
- + 8 bits: A (α-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

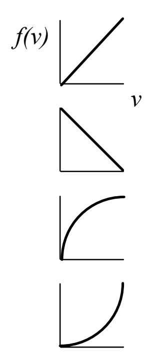
- 2D generalization of signal processing
- Image as a two-dimensional signal
- Point processing: modify pixels independently
- Filtering: modify based on neighborhood
- Compositing: combine several images
- Image compression: space-efficient formats
- Other topics (not in this course)
  – Image enhancement and restoration
  – Computer vision

Outline

- Blending
- Display Color Models
- Filters
- Dithering
- Image Compression

Point Processing

- 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) = v^p, p < 1 (brighten)
  - f(v) = v^p, p > 1 (darken)
- Gamma correction compensates monitor brightness loss



Gamma Correction Example

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

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
  - Shorthand: b = a   h (= h   a, as an aside)

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

- 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 × 3 blurring filter

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

- As function

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

- In matrix form

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

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

Pictures have been removed for printing purposes due to a PowerPoint bug

Original Image Blur 3x3 mask Blur 7x7 mask

Example Noise Reduction

Pictures have been removed for printing
due to a PowerPoint bug

Original image

Image with noise

Median filter (5x5?)

Edge Filters

- Discover edges in image
- Characterized by large gradient

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

- Approximate 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

- 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

-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	25	25	0	0	0	0
0	0	0	0	25	25	0	0	0	0
0	0	0	0	25	25	0	0	0	0
0	0	0	0	25	25	0	0	0	0
0	0	0	0	25	25	0	0	0	0
0	0	0	0	25	25	0	0	0	0
0	0	0	0	25	25	0	0	0	0
0	0	0	0	25	25	0	0	0	0
0	0	0	0	25	25	0	0	0	0
0	0	0	0	25	25	0	0	0	0

b

Example of Edge Filter

Images have been removed due to a PowerPoint bug

Original image

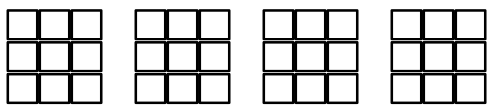
Edge filter, then brightened

Outline

- Blending
- Display Color Models
- Filters
- Dithering
- Image Compression

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



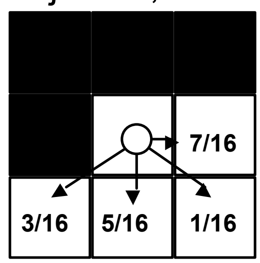
Halftone Screens

- Regular patterns create some artefacts
  - Avoid stripes
  - Avoid isolated pixels (e.g. on laser printer)
  - Monotonicity: keep pixels on at higher intensities
- Example of good 3×3 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 Example

Images have been removed due to a PowerPoint bug

Gray Scale Ramp

•Some worms

•Some checkerboards

•Enhance edges

Peter Anderson

Color Dithering

- Example: 8 bit framebuffer
  - Set color map by dividing 8 bits into 3,3,2 for RGB
  - Blue is deemphasized since we see it less well
- Dither RGB separately
  - Works well with Floyd-Steinberg
- Assemble results into 8 bit index into colormap
- Generally looks good

Outline

- Blending
- Display Color Models
- Filters
- Dithering
- Image 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

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, ...

Exploiting Coding Redundancy

- Not limited to images (text, other digital info)
- Exploit nonuniform probabilities of symbols
- Entropy as measure of information content
  - $H = -\Sigma_i \text{Prob}(s_i) \log_2 (\text{Prob}(s_i))$
  - 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 × 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

Summary

- Display Color Models
  - 8 bit (colormap), 24 bit, 96 bit
- Filters
  - Blur, edge detect, sharpen, despeckle
- Dithering
  - Floyd-Steinberg error diffusion
- Image Compression
  - Coding, interpixel, psychovisual redundancy
  - Lossless vs. lossy compression

Preview

- Assignment 5 due Thursday
- Assignment 6 out Thursday
- Thursday: Ray Tracing