Recap from Wednesday

Spectra and Color
Light capture in cameras and humans
Ted Adelson’s checkerboard illusion
Motion illusion, rotating snakes
Sampling and Reconstruction
Sampled representations

• How to store and compute with continuous functions?
• Common scheme for representation: samples
  – write down the function’s values at many points
Reconstruction

- Making samples back into a continuous function
  - for output (need realizable method)
  - for analysis or processing (need mathematical method)
  - amounts to “guessing” what the function did in between
1D Example: Audio

low frequencies

high frequencies
Sampling in digital audio

- Recording: sound to analog to samples to disc
- Playback: disc to samples to analog to sound again
  - how can we be sure we are filling in the gaps correctly?
Sampling and Reconstruction

- Simple example: a sine wave
Undersampling

• What if we “missed” things between the samples?
• Simple example: undersampling a sine wave
  – unsurprising result: information is lost
Undersampling

• What if we “missed” things between the samples?
• Simple example: undersampling a sine wave
  – unsurprising result: information is lost
  – surprising result: indistinguishable from lower frequency
Undersampling

• What if we “missed” things between the samples?

• Simple example: undersampling a sine wave
  – unsurprising result: information is lost
  – surprising result: indistinguishable from lower frequency
  – also was always indistinguishable from higher frequencies
  – aliasing: signals “traveling in disguise” as other frequencies
Aliasing in video

Imagine a spoked wheel moving to the right (rotating clockwise). Mark wheel with dot so we can see what’s happening.

If camera shutter is only open for a fraction of a frame time (frame time = 1/30 sec. for video, 1/24 sec. for film):

Without dot, wheel appears to be rotating slowly backwards! (counterclockwise)
Aliasing in images
What’s happening?

Input signal:

x = 0:.05:5; imagesc(sin((2.^x).*x))

Plot as image:

x = 0:.05:5; imagesc(sin((2.^x).*x))

Alias!
Not enough samples
Antialiasing

What can we do about aliasing?

Sample more often
- Join the Megapixel craze of the photo industry
- But that only shifts the problem to higher frequencies

Make the signal less “wiggly”
- Get rid of some high frequencies
- Will lose information
- But it’s better than aliasing
Preventing aliasing

- Introduce lowpass filters:
  - remove high frequencies leaving only safe, low frequencies
  - choose lowest frequency in reconstruction (disambiguate)
Linear filtering: a key idea

- Transformations on signals; e.g.:
  - bass/treble controls on stereo
  - blurring/sharpening operations in image editing
  - smoothing/noise reduction in tracking

- Key properties
  - linearity: $\text{filter}(f + g) = \text{filter}(f) + \text{filter}(g)$
  - shift invariance: behavior invariant to shifting the input
    • delaying an audio signal
    • sliding an image around

- Can be modeled mathematically by *convolution*
Moving Average

• basic idea: define a new function by averaging over a sliding window
• a simple example to start off: smoothing
Weighted Moving Average

- Can add weights to our moving average
- \textit{Weights} \space [..., 0, 1, 1, 1, 1, 1, 0, ...] / 5
Weighted Moving Average

- bell curve (gaussian-like) weights [..., 1, 4, 6, 4, 1, ...]
## Moving Average In 2D

### What are the weights $H$?

- $H[u, v]$

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>90</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>90</td>
<td>0</td>
<td>90</td>
<td>0</td>
<td>90</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

$F[x, y]$
Cross-correlation filtering

- Let’s write this down as an equation. Assume the averaging window is $(2k+1) \times (2k+1)$:

$$
G[i, j] = \frac{1}{(2k + 1)^2} \sum_{u=-k}^{k} \sum_{v=-k}^{k} F[i + u, j + v]
$$

- We can generalize this idea by allowing different weights for different neighboring pixels:

$$
G[i, j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u, v] F[i + u, j + v]
$$

- This is called a **cross-correlation** operation and written:

$$
G = H \otimes F
$$

- H is called the “filter,” “kernel,” or “mask.”
Gaussian filtering

A Gaussian kernel gives less weight to pixels further from the center of the window

\[
F[x, y] = \begin{array}{cccccccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 90 & 90 & 90 & 90 & 90 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 90 & 90 & 90 & 90 & 90 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 90 & 90 & 90 & 90 & 90 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 90 & 90 & 90 & 90 & 90 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 90 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

\[
H[u, v] = \frac{1}{16}
\]

\[
h(u, v) = \frac{1}{2\pi\sigma^2}e^{-\frac{u^2+v^2}{\sigma^2}}
\]

Slide by Steve Seitz
Mean vs. Gaussian filtering
Convolution

cross-correlation: \[ G = H \otimes F \]

\[ G[i, j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u, v] F[i + u, j + v] \]

A convolution operation is a cross-correlation where the filter is flipped both horizontally and vertically before being applied to the image:

\[ G[i, j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u, v] F[i - u, j - v] \]

It is written:

\[ G = H \ast F \]

Suppose H is a Gaussian or mean kernel. How does convolution differ from cross-correlation?
Convolution is nice!

• Notation: $b = c \ast a$

• Convolution is a multiplication-like operation
  – commutative $a \ast b = b \ast a$
  – associative $a \ast (b \ast c) = (a \ast b) \ast c$
  – distributes over addition $a \ast (b + c) = a \ast b + a \ast c$
  – scalars factor out $\alpha a \ast b = a \ast \alpha b = \alpha(a \ast b)$
  – identity: unit impulse $e = [\ldots, 0, 0, 1, 0, 0, \ldots]$
    
    $$a \ast e = a$$

• Conceptually no distinction between filter and signal

• Usefulness of associativity
  – often apply several filters one after another: $(((a \ast b_1) \ast b_2) \ast b_3)$
  – this is equivalent to applying one filter: $a \ast (b_1 \ast b_2 \ast b_3)$