Sampling and reconstruction

CS 4620 Lecture 3
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
Filtering

- Processing done on a function
  can be executed in continuous form (e.g. analog circuit)
  but can also be executed using sampled representation

- Simple example: smoothing by averaging
Roots of sampling

- Nyquist 1928; Shannon 1949  
  famous results in information theory
- 1940s: first practical uses in telecommunications
- 1960s: first digital audio systems
- 1970s: commercialization of digital audio
- 1982: introduction of the Compact Disc  
  the first high-profile consumer application
- This is why all the terminology has a communications or audio “flavor”  
  early applications are 1D; for us 2D (images) is important
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?
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
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
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
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
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
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
Preventing aliasing

- Introduce lowpass filters:
  remove high frequencies leaving only safe, low frequencies
  choose lowest frequency in reconstruction (disambiguate)
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
Convolution warm-up

- basic idea: define a new function by averaging over a sliding window
- a simple example to start off: smoothing
Convolution warm-up

- basic idea: define a new function by averaging over a sliding window
- a simple example to start off: smoothing
Convolution warm-up

- basic idea: define a new function by averaging over a sliding window
- a simple example to start off: smoothing
Convolution warm-up

- basic idea: define a new function by averaging over a sliding window
- a simple example to start off: smoothing
Convolution warm-up

- basic idea: define a new function by averaging over a sliding window
- a simple example to start off: smoothing
Convolution warm-up

- Same moving average operation, expressed mathematically:

\[ c[i] = \frac{1}{2r + 1} \sum_{j=i-r}^{i+r} a[j]. \]
Discrete convolution

- Simple averaging:
  \[ c[i] = \frac{1}{2r + 1} \sum_{j=i-r}^{i+r} a[j]. \]
  every sample gets the same weight

- Convolution: same idea but with weighted average
  \[ (a * b)[i] = \sum_{j} a[j]b[i - j]. \]
  each sample gets its own weight (normally zero far away)

- This is all convolution is: it is a **moving weighted average**
Filters

- Sequence of weights \( b \) is called a *filter*
- Filter is nonzero over its *region of support* usually centered on zero: support radius \( r \)
- Filter is *normalized* so that it sums to 1.0 this makes for a weighted average, not just any old weighted sum
- Most filters are symmetric about 0 since for images we usually want to treat left and right the same

\[
\text{Normalized box filter:} \quad \frac{1}{2r+1} \quad \text{if} \quad x = 0 \quad \text{and 0 otherwise.}
\]
Convolution and filtering

- Can express sliding average as convolution with a *box filter*
- $b_{\text{box}} = [...]/5$
Example: box and step

What is the result of convolving $a$ and $b$? At a particular index $i$, as shown in Figure 9.6, the result is the average of the step function over the range from $i - 2$ to $i + 2$. If $i < 2$, we are averaging all zeros and the result is zero. If $i > 2$, we are averaging all ones and the result is one. In between there are $i + 3$ ones, resulting in the value $i + 3 / 5$. The output is a linear ramp that goes from 0 to 1 over five samples:

$\begin{align*}
\ldots, 0, 0, 1, 2, 3, 4, 5, 5, \ldots
\end{align*}$

**Properties of Convolution**

The way we've written it so far, convolution seems like an asymmetric operation: $a$ is the sequence we're smoothing, and $b$ provides the weights. But one of the nice properties of convolution is that it actually doesn't make any difference which is which: the filter and the signal are interchangeable. To see this, just rethink the sum in Equation 9.2 with the indices counting from the origin of the filter $b$, rather than $a$. 

---

**Example: box and step**

- $a[j]$:
  - A box function, depicted as a horizontal line representing a sequence of values.
  - Index $j$.

- $b[i-j]$:
  - A step function, depicted as a sequence of values.
  - Index $i-j$.

**Diagram Details**:
- The box function $a[j]$ extends from $j = -\infty$ to $j = \infty$.
- The step function $b[i-j]$ is a sequence of values from $i-j = -\infty$ to $i-j = \infty$.

---

**Figure 9.6** Discrete convolution of a box function with a step function.
Example: box and step

\[
\begin{align*}
\text{a[j]} & \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \\
\times & \\
\text{b[i-j]} & \quad 1 \quad 1 \quad 1 \quad 1 \\
\rightarrow & \quad i-j \quad 0 \\
\end{align*}
\]

\[
a[j] \cdot b[i-j] = 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0
\]
What is the result of convolving $a$ and $b$? At a particular index $i$, as shown in Figure 9.6, the result is the average of the step function over the range from $i-2$ to $i+2$. If $i<2$, we are averaging all zeros and the result is zero. If $i>2$, we are averaging all ones and the result is one. In between there are $i+3$ ones, resulting in the value $\frac{i+3}{5}$. The output is a linear ramp that goes from 0 to 1 over five samples: $\left[\ldots, 0, 0, 0, 0, 0, 0, 0, \ldots\right]$.

Properties of Convolution

The way we've written it so far, convolution seems like an asymmetric operation: $a$ is the sequence we're smoothing, and $b$ provides the weights. But one of the nice properties of convolution is that it actually doesn't make any difference which is which: the filter and the signal are interchangeable. To see this, just rethink the sum in Equation 9.2 with the indices counting from the origin of the filter $b$.
Example: box and step

Figure 9.6. Discrete convolution of a box function with a step function.

What is the result of convolving $a$ and $b$? At a particular index $i$, as shown in Figure ??, the result is the average of the step function over the range from $i - 2$ to $i + 2$. If $i < 2$, we are averaging all zeros and the result is zero. If $i > 2$, we are averaging all ones and the result is one. In between there are $i + 3$ ones, resulting in the value $i + 3 / 5$. The output is a linear ramp that goes from 0 to 1 over five samples: \[ \ldots, 0, 0, 1, 2, 3, 4, 5, \ldots \]

Properties of Convolution

The way we've written it so far, convolution seems like an asymmetric operation: $a$ is the sequence we're smoothing, and $b$ provides the weights. But one of the nice properties of convolution is that it actually doesn't make any difference which is which: the filter and the signal are interchangeable. To see this, just rethink the sum in Equation 9.2 with the indices counting from the origin of the filter $b$, rather than from $i$. 

Example: box and step
Example: box and step

\[ a[j] \times b[i-j] \]

\[
\sum b[i-j] \rightarrow \frac{1}{5} \rightarrow \sum a[j]b[i-j] \rightarrow \frac{1}{5} \rightarrow \sum (a \ast b)[i]
\]

Properties of Convolution

The way we've written it so far, convolution seems like an asymmetric operation: \( a \) is the sequence we're smoothing, and \( b \) provides the weights. But one of the nice properties of convolution is that it actually doesn't make any difference which is which: the filter and the signal are interchangeable. To see this, just rethink the sum in Equation 9.2 with the indices counting from the origin of the filter \( b \), rather than \( a \).
Convolution and filtering

- Convolution applies with any sequence of weights
- Example: bell curve (gaussian-like) \(..., 1, 4, 6, 4, 1, ...\)/16
Convolution and filtering

- Convolution applies with any sequence of weights
- Example: bell curve (gaussian-like) \([..., 1, 4, 6, 4, 1, ...]/16\)
Discrete convolution

- 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
Discrete filtering in 2D

- Same equation, one more index
  \[(a * b)[i, j] = \sum_{i', j'} a[i', j'] b[i - i', j - j']\]
  now the filter is a rectangle you slide around over a grid of numbers

- Commonly applied to images
  blurring (using box, using gaussian, …)
  sharpening (impulse minus blur)

- Usefulness of associativity
  often apply several filters one after another: 
  \[((a * b_1) * b_2) * b_3\)
  this is equivalent to applying one filter: 
  \[a * (b_1 * b_2 * b_3)\]
And in pseudocode...

```plaintext
function convolve2d(filter2d a, filter2d b, int i, int j)
    s = 0
    r = a.radius
    for i' = -r to r do
        for j' = -r to r do
            s = s + a[i'][j'] * b[i - i'][j - j']
    return s
```
original 🛄 | 🅽 box blur

sharpened 🛄 | 🅽 gaussian blur
Optimization: separable filters

- basic alg. is $O(r^2)$: large filters get expensive fast!
- definition: $a_2(x,y)$ is separable if it can be written as:
  \[ a_2[i, j] = a_1[i]a_1[j] \]
  this is a useful property for filters because it allows factoring:
  \[
  (a_2 \ast b)[i, j] = \sum_{i'} \sum_{j'} a_2[i', j']b[i - i', j - j'] \\
  = \sum_{i'} \sum_{j'} a_1[i']a_1[j']b[i - i', j - j'] \\
  = \sum_{i'} a_1[i'] \left( \sum_{j'} a_1[j']b[i - i', j - j'] \right)
  \]
Separable filtering

\[ a_2[i, j] = a_1[i] a_1[j] \]

\[
\begin{array}{cccc}
1 & 4 & 6 & 4 & 1 \\
4 & 16 & 24 & 16 & 4 \\
6 & 24 & 36 & 24 & 6 \\
4 & 16 & 24 & 16 & 4 \\
1 & 4 & 6 & 4 & 1 \\
\end{array}
\begin{array}{cccc}
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
\end{array}
\begin{array}{cccc}
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 4 & 0 & 0 \\
0 & 0 & 6 & 0 & 0 \\
0 & 0 & 4 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
\end{array}
\]

first, convolve with this

\[
\sum_{i'} a_1[i'] \left( \sum_{j'} a_1[j'] b[i - i', j - j'] \right)
\]
Separable filtering

\[ a_2[i, j] = a_1[i] a_1[j] \]

\[
\begin{array}{cccc}
1 & 4 & 6 & 4 & 1 \\
4 & 16 & 24 & 16 & 4 \\
6 & 24 & 36 & 24 & 6 \\
4 & 16 & 24 & 16 & 4 \\
1 & 4 & 6 & 4 & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
1 & 4 & 6 & 4 & 1 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{cccc}
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 4 & 0 & 0 \\
0 & 0 & 6 & 0 & 0 \\
0 & 0 & 4 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
\end{array}
\]

First, convolve with this

Second, convolve with this

\[
\sum_{i'} a_1[i'] \left( \sum_{j'} a_1[j'] b[i - i', j - j'] \right)
\]
Continuous convolution: warm-up

- Can apply sliding-window average to a continuous function just as well
  
  output is continuous
  
  integration replaces summation

![](original.png)

![](smoothed.png)
Continuous convolution: warm-up

- Can apply sliding-window average to a continuous function just as well
  - output is continuous
  - integration replaces summation
Continuous convolution: warm-up

- Can apply sliding-window average to a continuous function just as well
  - output is continuous
  - integration replaces summation
Continuous convolution: warm-up

• Can apply sliding-window average to a continuous function just as well
  
  output is continuous
  
  integration replaces summation
Continuous convolution: warm-up

- Can apply sliding-window average to a continuous function just as well

  output is continuous

  integration replaces summation
Continuous convolution

- Sliding average expressed mathematically:

\[
g_{\text{smooth}}(x) = \frac{1}{2r} \int_{x-r}^{x+r} g(t) \, dt
\]

note difference in normalization (only for box)

- Convolution just introduces weights

\[
(f \ast g)(x) = \int_{-\infty}^{\infty} f(t)g(x - t) \, dt
\]

weighting is now by a function

weighted integral is like weighted average
again bounds are set by support of \(f(x)\)
One more convolution

- Continuous–discrete convolution

\[
(a \ast f)(x) = \sum_{i} a[i] f(x - i)
\]

\[
(a \ast f)(x, y) = \sum_{i, j} a[i, j] f(x - i, y - j)
\]

input: a sequence and a continuous function
output: a continuous function
used for reconstruction and resampling
Continuous-discrete convolution

samples

reconstructed
signal
Continuous-discrete convolution
Continuous-discrete convolution

samples

reconstructed signal

\[ \sum \]
Continuous-discrete convolution
Continuous-discrete convolution

samples

reconstructed signal
And in pseudocode...

```plaintext
function reconstruct(sequence a, filter f, real x)
  s = 0
  r = f.radius
  for i = ⌊x - r⌋ to ⌊x + r⌋ do
    s = s + a[i] * f(x - i)
  return s
```
Resampling

- Changing the sample rate
  in images, this is enlarging and reducing

- Creating more samples:
  increasing the sample rate
  “upsampling”
  “enlarging”

- Ending up with fewer samples:
  decreasing the sample rate
  “downsampling”
  “reducing”
Resampling

- Reconstruction creates a continuous function
  forget its origins, go ahead and sample it

![Graph showing samples and reconstructed signal]
Resampling

- Reconstruction creates a continuous function
  forget its origins, go ahead and sample it
Resampling

- Reconstruction creates a continuous function
  forget its origins, go ahead and sample it
Resampling

- Reconstruction creates a continuous function forget its origins, go ahead and sample it.
Resampling

- Reconstruction creates a continuous function
  
  forget its origins, go ahead and sample it
And in pseudocode...

function resample(sequence a, filter f, real $x_0$, real $\Delta x$, int $N$)
allocate output sequence $b$ of length $N$
for $j = 0$ to $N - 1$ do
    $b[j] = \text{reconstruct}(a, f, x + j\Delta x)$
return $b$
Defining the source interval

- Convenient: define samples using an interval to be resampled, and \( N \)
- Exactly how to fill an interval with \( N \) samples? Desiderata:
  - Sample pattern should be centered in the interval (prevents shifting of content as you change resolution)
  - Sample patterns of adjacent intervals should tile (makes it meaningful to break up a signal into pieces)
  - \( N \) all the way down to 1 should work
- Solution: think of breaking into \( N \) subintervals and centering a sample in each one
  - Sample \( i \) goes at \( x_l + \frac{i + 0.5}{N} (x_r - x_l) \)
  - E.g. first sample at \( x_0 = x_l + \frac{1}{2} \Delta x \)
  - E.g. last sample at \( x_l + (N - \frac{1}{2}) \Delta x \)
And in pseudocode...

```plaintext
function resample(sequence a, filter f, real x_l, real x_r, int N)
allocate output sequence b of length N
for j = 0 to N - 1 do
  b[j] = reconstruct(a, f, x_l + (j + 1/2)(x_r - x_l)/N)
return b
```

Note that this expands into a double loop
Cont.–disc. convolution in 2D

- same convolution—just two variables now

\[(a \ast f)(x, y) = \sum_{i,j} a[i, j] f(x - i, y - j)\]

loop over nearby pixels, average using filter weight

looks like discrete filter, but offsets are not integers and filter is continuous

remember placement of filter relative to grid is variable

support of reconstruction filter

pixel locations

sample point for reconstruction
Cont.–disc. convolution in 2D

\[(a \ast f)(x, y) = \sum_{i,j} a[i, j] f(x - i, y - j)\]

Example showing filter weights used to compute a reconstructed value at a single point.
And in pseudocode...

```plaintext
function reconstruct(sequence a, filter f, real x, real y)
  s = 0
  for j = [y - f.ry] to [y + f.r_y] do
    for i = [x - f_RX] to [x + f.r_x] do
      s = s + a[i, j] * f(x - i, y - j)
  return s
```
Defining the source rectangle

- exactly the same way we defined the source interval in 1D
- except now there is an interval \((l, r)\) in \(x\) and an interval \((b, t)\) in \(y\)
- intervals in \(x\) and \(y\) constitute a rectangle
- sample points are positioned with the same considerations (at centers of a grid of sub-rectangles)
And in pseudocode...

```plaintext
function resample(image a, filter f, rectangle r, int w, int h)
allocate output image b of size (w, h)
for j = 0 to h - 1 do
    y = r.b + (j + 1/2)(r.t - r.b)/h
    for i = 0 to w - 1 do
        x = r.l + (i + 1/2)(r.r - r.l)/w
        b[i, j] = reconstruct(a, f, x, y)
return b
```

Note that this expands into a quadruple loop
Separable filters for resampling

- just as in filtering, separable filters are useful
  separability in this context is a statement about a continuous filter, rather than a discrete one:
  \[ f_2(x, y) = f_1(x)f_1(y) \]
- with a separable filter the region of support is rectangular
- all widely used resampling filters are separable
  there are good reasons best explained with frequency domain arguments
  (it’s easiest to design separable filters to suppress grid artifacts)
Optimized separable resampling

- for larger filters, separability provides an opportunity for an optimization
- resample in two passes, one resampling each row and one resampling each column
- intermediate storage required: product of one dimension of source and the other dimension of destination
two-stage resampling using a separable filter

\[ O(r^2 N_{dst} M_{dst}) \]

\[ O(rN_{src} M_{dst}) \]
A gallery of filters

- Box filter
  Simple and cheap
- Tent filter
  Linear interpolation
- Gaussian filter
  Very smooth antialiasing filter
- B-spline cubic
  Very smooth
- Catmull-rom cubic
  Interpolating
- Mitchell-Netravali cubic
  Good for image upsampling
Box filter

\[ a_{\text{box},r}[i] = \begin{cases} 
  1/(2r + 1) & |i| \leq r, \\
  0 & \text{otherwise}. 
\end{cases} \]

\[ f_{\text{box},r}(x) = \begin{cases} 
  1/(2r) & -r \leq x < r, \\
  0 & \text{otherwise}. 
\end{cases} \]
Tent filter

\[ f_{\text{tent}}(x) = \begin{cases} 
1 - |x| & |x| < 1, \\
0 & \text{otherwise}; 
\end{cases} \]

\[ f_{\text{tent},r}(x) = \frac{f_{\text{tent}}(x/r)}{r}. \]
Gaussian filter

\[ f_g(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}. \]
B-Spline cubic

\[ f_B(x) = \frac{1}{6} \begin{cases} -3(1 - |x|)^3 + 3(1 - |x|)^2 + 3(1 - |x|) + 1 & -1 \leq x \leq 1, \\ (2 - |x|)^3 & 1 \leq |x| \leq 2, \\ 0 & \text{otherwise}. \end{cases} \]
Catmull-Rom cubic

\[ f_C(x) = \frac{1}{2} \begin{cases} 
-3(1 - |x|)^3 + 4(1 - |x|)^2 + (1 - |x|) & \text{if } -1 \leq x \leq 1, \\
(2 - |x|)^3 - (2 - |x|)^2 & \text{if } 1 \leq |x| \leq 2, \\
0 & \text{otherwise.} 
\end{cases} \]
Michell-Netravali cubic

\[ f_M(x) = \frac{1}{3} f_B(x) + \frac{2}{3} f_C(x) \]

\[ = \frac{1}{18} \begin{cases} 
-21(1 - |x|)^3 + 27(1 - |x|)^2 + 9(1 - |x|) + 1 & -1 \leq x \leq 1, \\
7(2 - |x|)^3 - 6(2 - |x|)^2 & 1 \leq |x| \leq 2, \\
0 & \text{otherwise.}
\end{cases} \]
Lanczos

\[ f_{L2}(x) = \begin{cases} \text{sinc}(x) \text{sinc}(x/2) & |x| < 2 \\ 0 & \text{otherwise} \end{cases} \]

\[ \text{sinc}(x) = \frac{\sin(\pi x)}{\pi x} \]

\[ f_{L3}(x) = \begin{cases} \text{sinc}(x) \text{sinc}(x/3) & |x| < 3 \\ 0 & \text{otherwise} \end{cases} \]
Effects of reconstruction filters

- For some filters, the reconstruction process winds up implementing a simple algorithm

- Box filter (radius 0.5): nearest neighbor sampling
  box always catches exactly one input point
  it is the input point nearest the output point
  so output\([i, j] = \text{input}[\text{round}(x(i)), \text{round}(y(j))]\]
  \(x(i)\) computes the position of the output coordinate \(i\) on the input grid

- Tent filter (radius 1): linear interpolation
  tent catches exactly 2 input points
  weights are \(a\) and \((1 - a)\)
  result is straight-line interpolation from one point to the next
Properties of filters

- Degree of continuity
- Impulse response
- Interpolating or no
- Ringing, or overshoot

interpolating filter used for reconstruction
Ringing, overshoot, ripples

- **Overshoot**
  - caused by negative filter values

- **Ripples**
  - constant in, non-const. out
  - ripple free when:

\[
\sum_i f(x + i) = 1 \quad \text{for all } x.
\]
Constructing 2D filters

- Separable filters (most common approach)
Yucky details

• What about near the edge?
  
  the filter window falls off the edge of the image
  
  need to extrapolate

methods:
  
  • clip filter (black)
  • wrap around
  • copy edge
  • reflect across edge
  • vary filter near edge
Yucky details

• What about near the edge?
  
  the filter window falls off the edge of the image
  need to extrapolate

methods:
  • clip filter (black)
  • wrap around
  • copy edge
  • reflect across edge
  • vary filter near edge
What about near the edge?

the filter window falls off the edge of the image

need to extrapolate

methods:

- clip filter (black)
- wrap around
- copy edge
- reflect across edge
- vary filter near edge
Yucky details

- What about near the edge?
  - the filter window falls off the edge of the image
  - need to extrapolate

methods:
  - clip filter (black)
  - wrap around
  - copy edge
  - reflect across edge
  - vary filter near edge
Yucky details

• What about near the edge?
  the filter window falls off the edge of the image
  need to extrapolate
methods:
  • clip filter (black)
  • wrap around
  • copy edge
  • reflect across edge
  • vary filter near edge
Yucky details

- What about near the edge?
  - the filter window falls off the edge of the image
  - need to extrapolate

  methods:
  - clip filter (black)
  - wrap around
  - copy edge
  - reflect across edge
  - vary filter near edge
Yucky details

- What about near the edge?
  
  the filter window falls off the edge of the image
  
  need to extrapolate
  
  methods:
  
  - clip filter (black)
  - wrap around
  - copy edge
  - reflect across edge
  - vary filter near edge
Yucky details

- What about near the edge?
  - the filter window falls off the edge of the image
  - need to extrapolate

methods:
  - clip filter (black)
  - wrap around
  - copy edge
  - reflect across edge
  - vary filter near edge
Yucky details

- What about near the edge?
  the filter window falls off the edge of the image
need to extrapolate
methods:
  - clip filter (black)
  - wrap around
  - copy edge
  - reflect across edge
  - vary filter near edge
Yucky details

• What about near the edge?

  the filter window falls off the edge of the image
  need to extrapolate

  methods:
  • clip filter (black)
  • wrap around
  • copy edge
  • reflect across edge
  • vary filter near edge
Yucky details

- What about near the edge?
  the filter window falls off the edge of the image
  need to extrapolate
  methods:
    - clip filter (black)
    - wrap around
    - copy edge
    - reflect across edge
    - vary filter near edge
Yucky details

- What about near the edge?
  - the filter window falls off the edge of the image
  - need to extrapolate
    - clip filter (black)
    - wrap around
    - copy edge
    - reflect across edge
    - vary filter near edge

[Philip Greenspun]
Yucky details

• What about near the edge?
  the filter window falls off the edge of the image
  need to extrapolate
methods:
  • clip filter (black)
  • wrap around
  • copy edge
  • reflect across edge
  • vary filter near edge
Reducing and enlarging

- Very common operation
  - devices have differing resolutions
  - applications have different memory/quality tradeoffs
- Also very commonly done poorly
- Simple approach: drop/replicate pixels
- Correct approach: use resampling
by dropping pixels

by dropping pixels

gaussian filter

gaussian filter

250 pixel width

250 pixel width
box reconstruction filter

bicubic reconstruction filter

4000 pixel width
Types of artifacts

- Garden variety
  what we saw in this natural image
  fine features become jagged or sparkle

- Moiré patterns
600ppi scan of a color halftone image
downsampling a high resolution scan

by dropping pixels

gaussian filter
Types of artifacts

- Garden variety
  what we saw in this natural image
  fine features become jagged or sparkle

- Moiré patterns
  caused by repetitive patterns in input
  produce large-scale artifacts; highly visible

- These artifacts are aliasing just like in the audio example earlier

- How do I know what filter is best at preventing aliasing?
  practical answer: experience
  theoretical answer: there is another layer of cool math behind all this
  - based on Fourier transforms
  - provides much insight into aliasing, filtering, sampling, and reconstruction