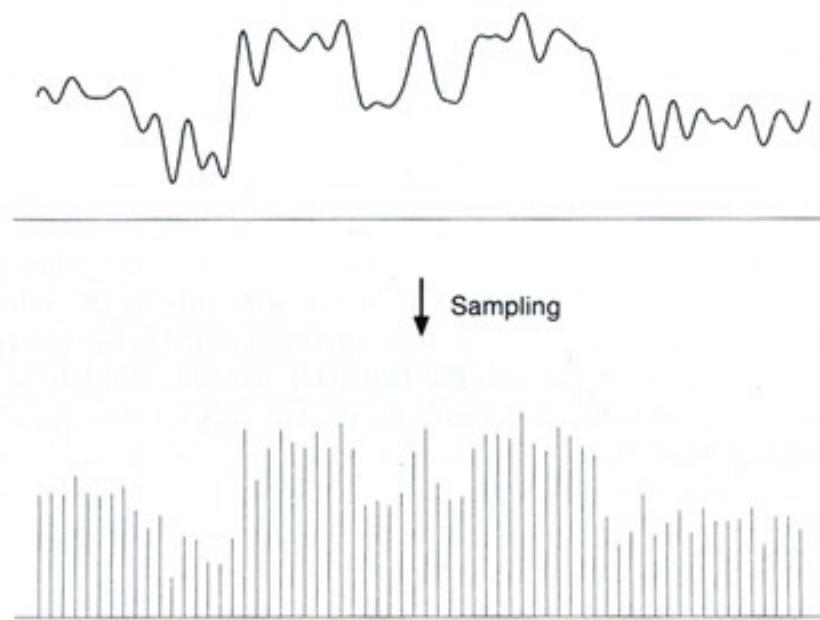


# **Sampling and reconstruction**

CS 5625 Lecture 2

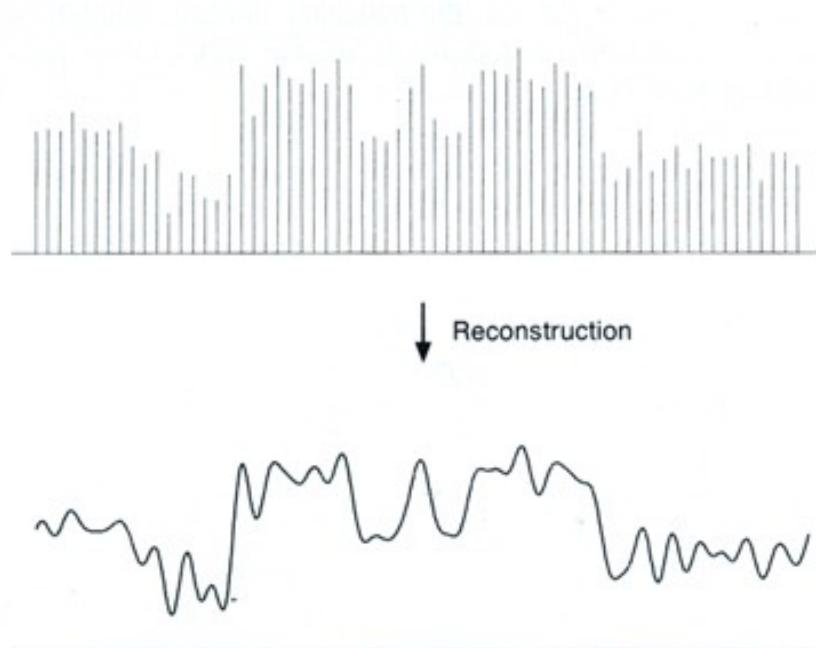
# Sampled representations

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



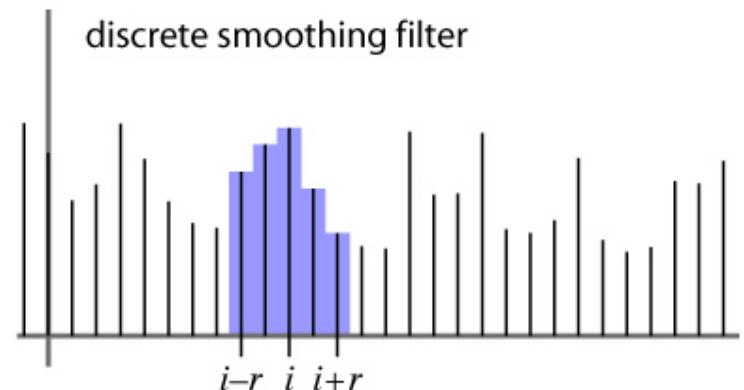
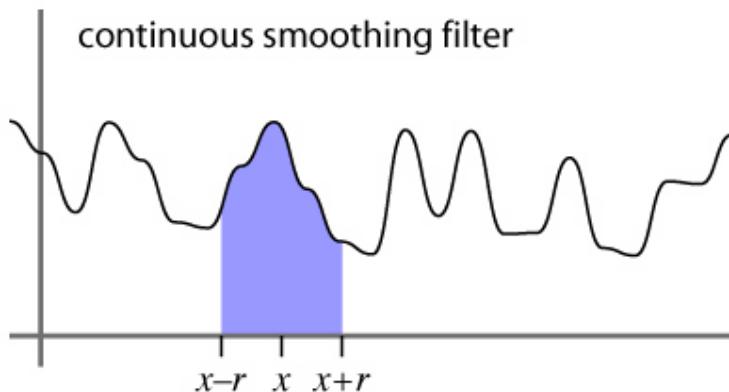
# Reconstruction

- 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

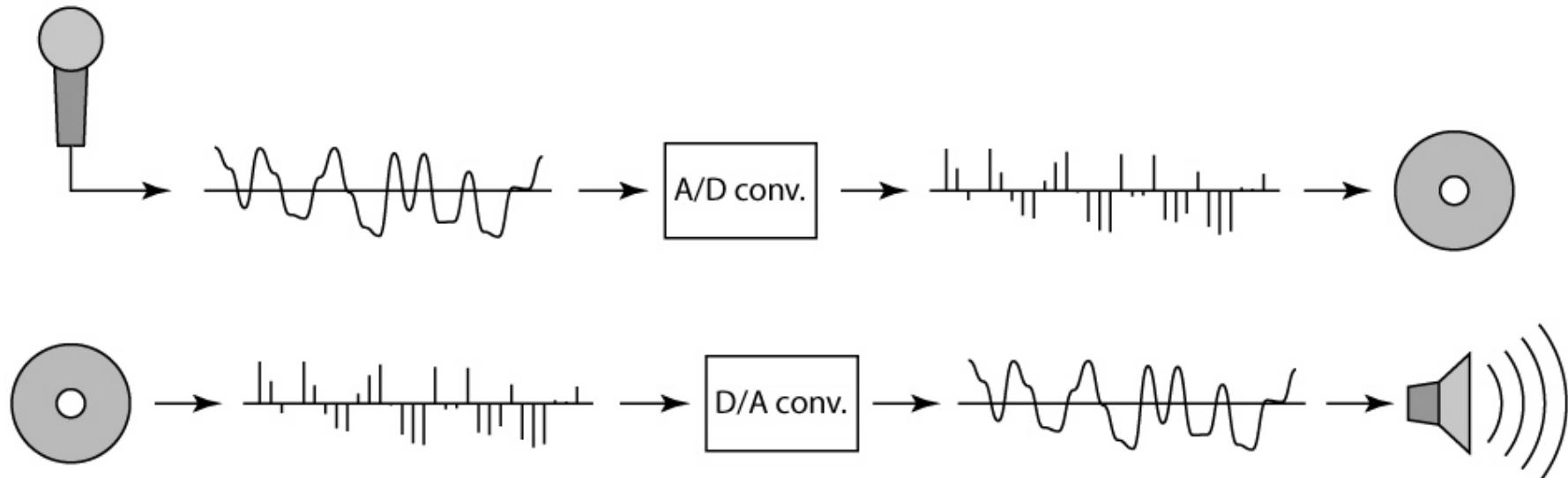


# **Roots of sampling**

- 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

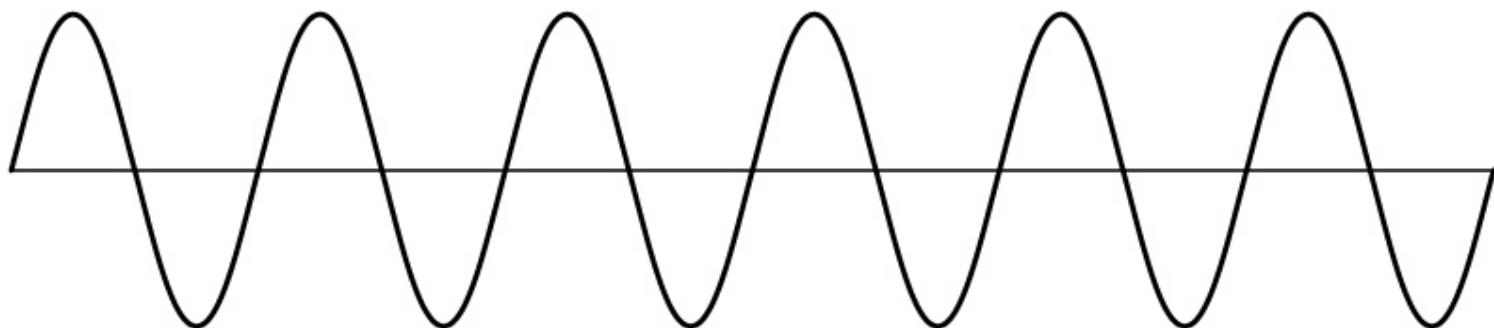
# Sampling in digital audio

- 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?



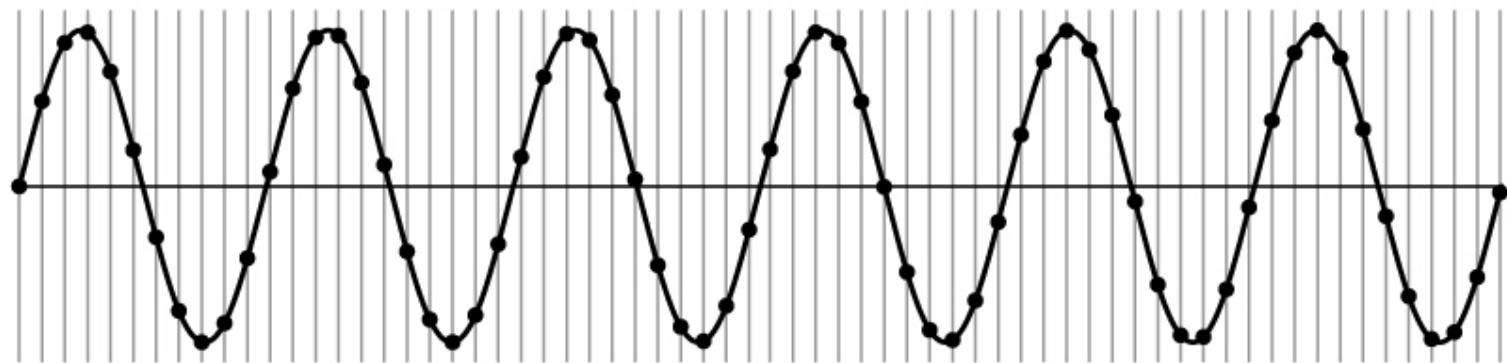
# Undersampling

- 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



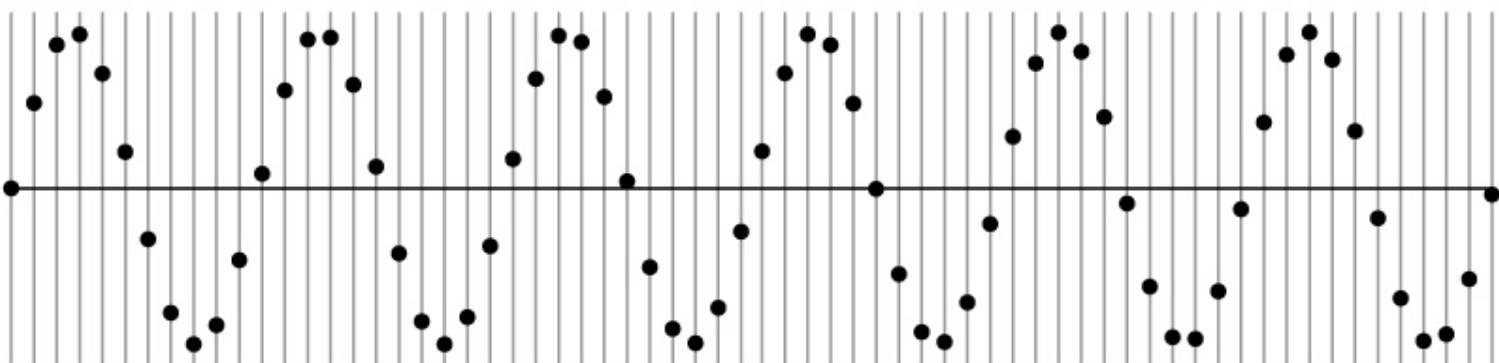
# Undersampling

- 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



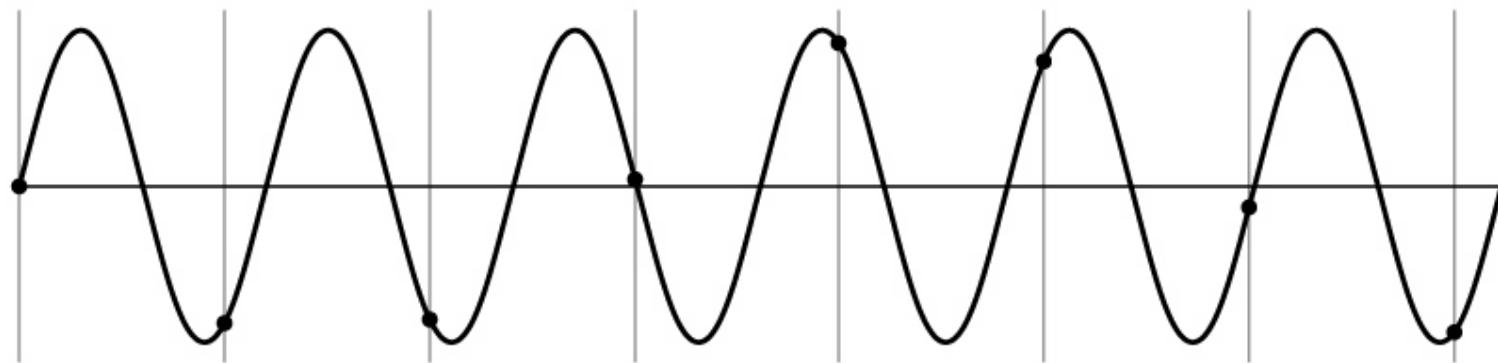
# Undersampling

- 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



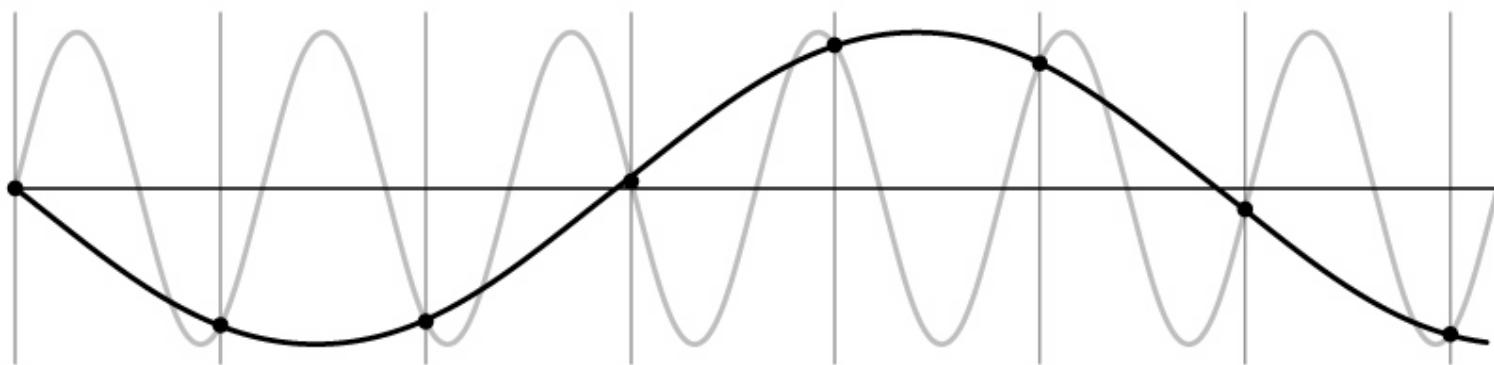
# Undersampling

- 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



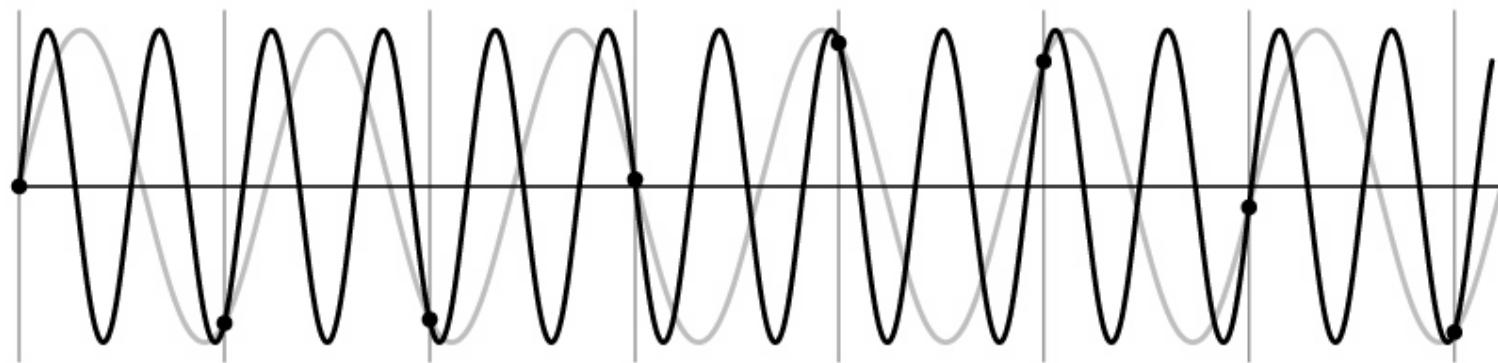
# Undersampling

- 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

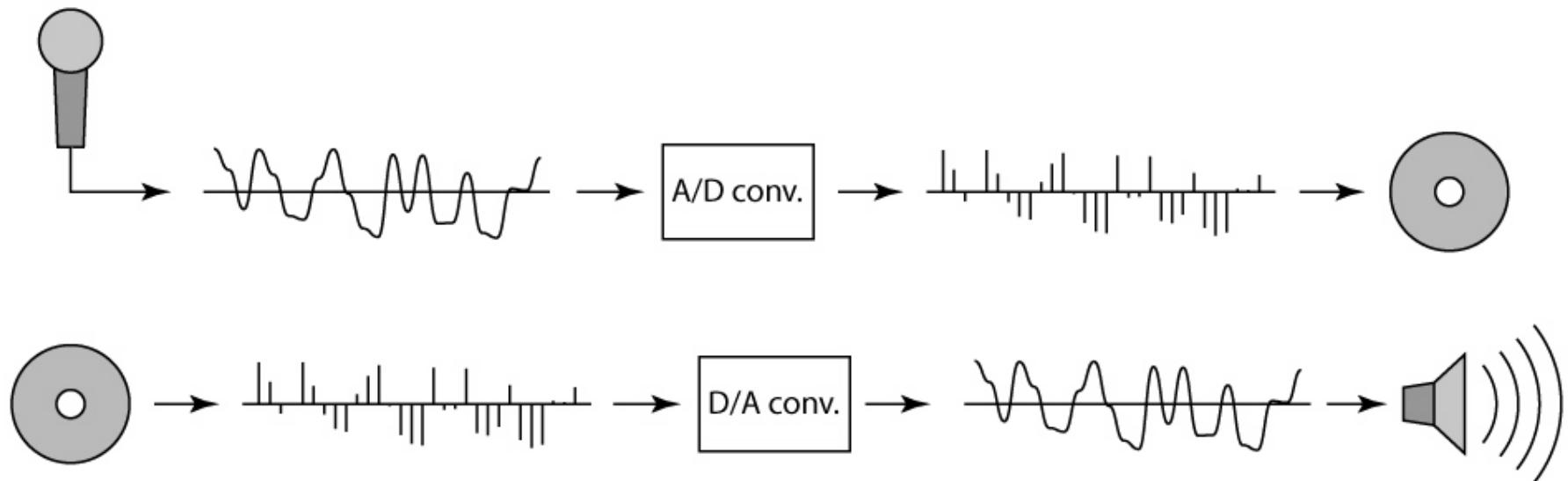


# Sneak preview

- Sampling creates copies of the signal at higher frequencies
- Aliasing is these frequencies leaking into the reconstructed signal
  - frequency  $f_s - x$  shows up as frequency  $x$
- The solution is filtering
  - during sampling, filter to keep the high frequencies out so they don't create aliases at the lower frequencies
  - during reconstruction, again filter high frequencies to avoid including high-frequency aliases in the output.

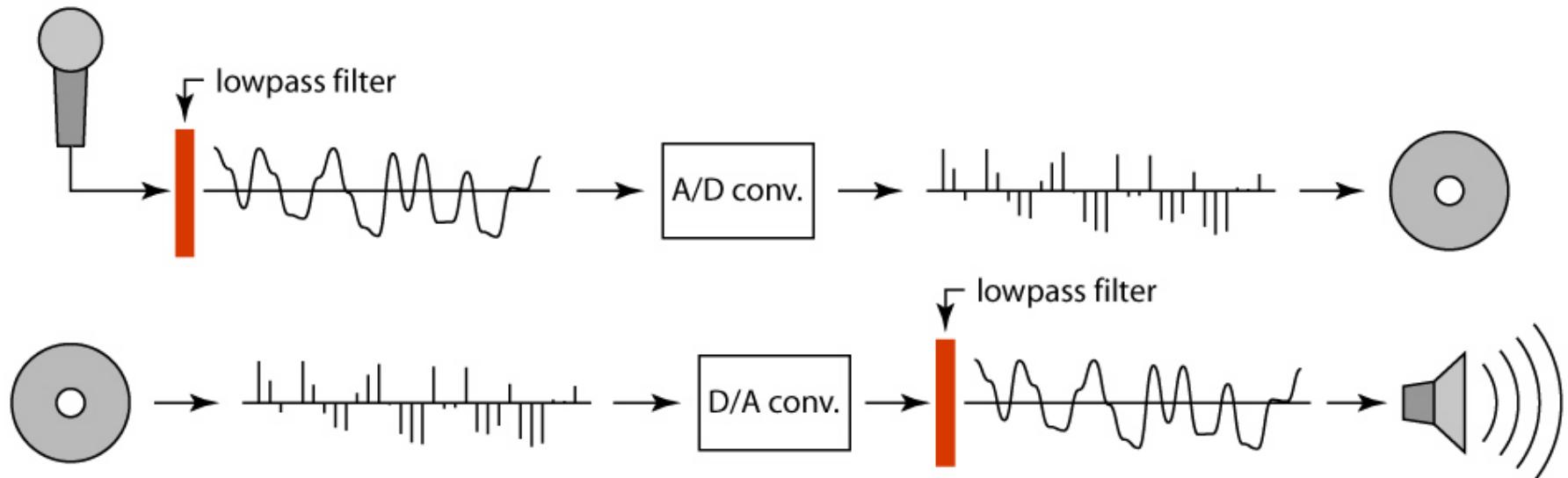
# Preventing aliasing

- 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)

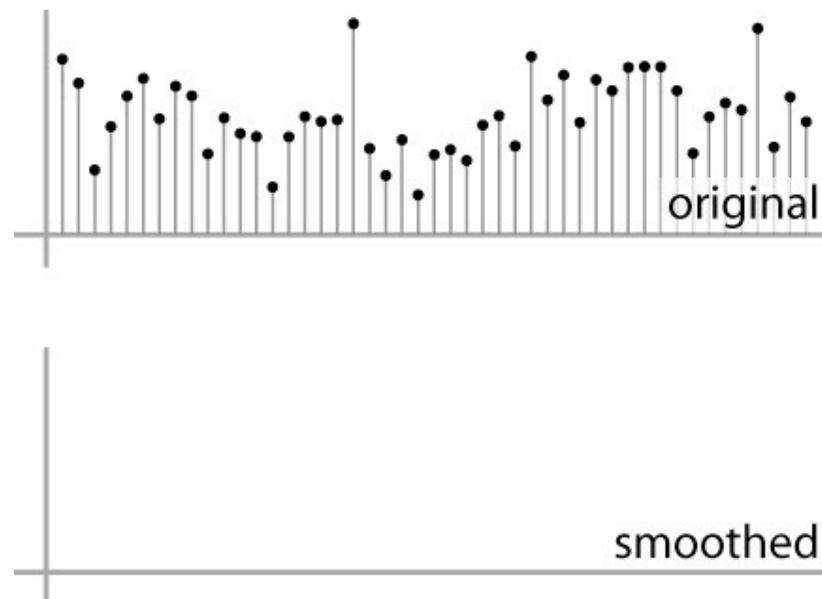


# Linear filtering: a key idea

- 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*

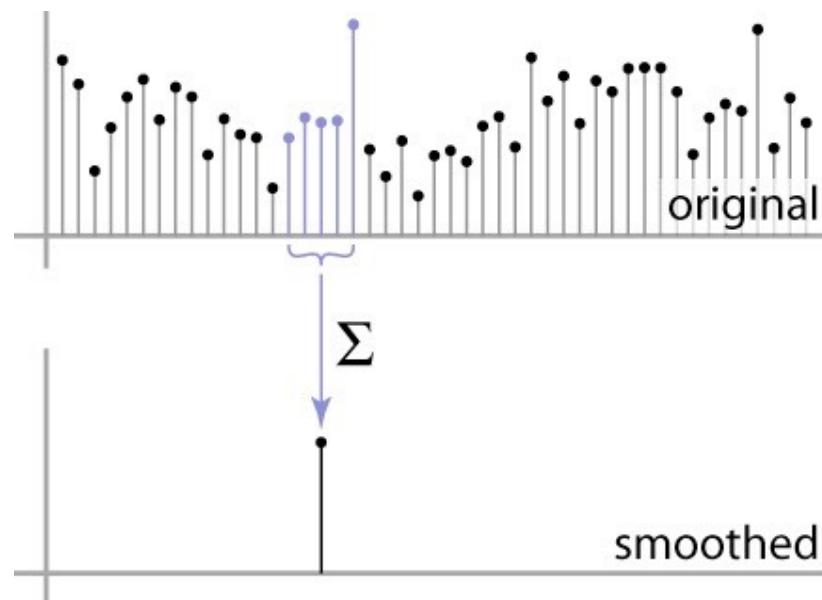
# Convolution warm-up

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



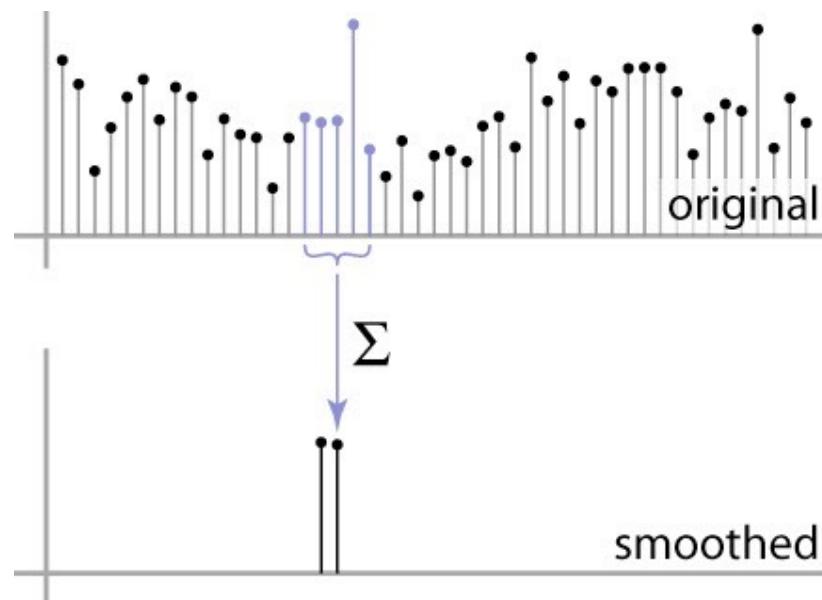
# Convolution warm-up

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



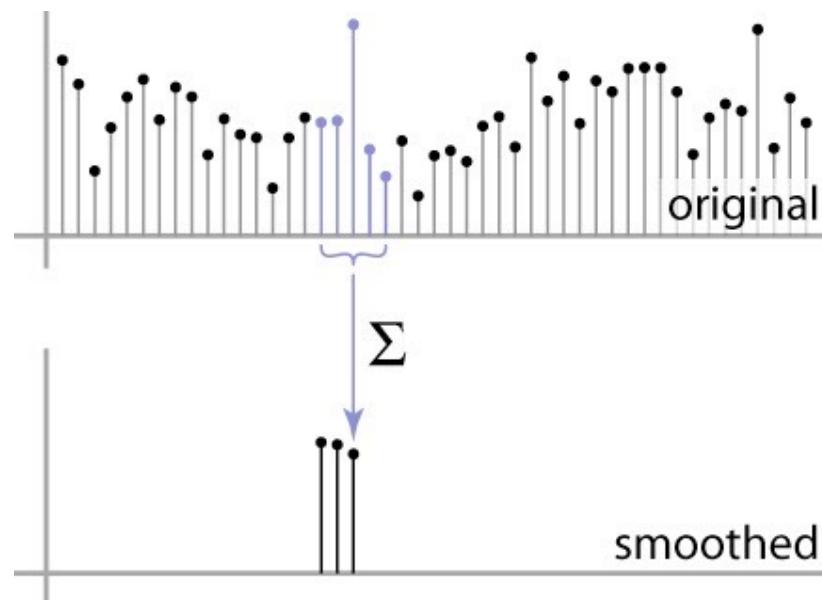
# Convolution warm-up

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



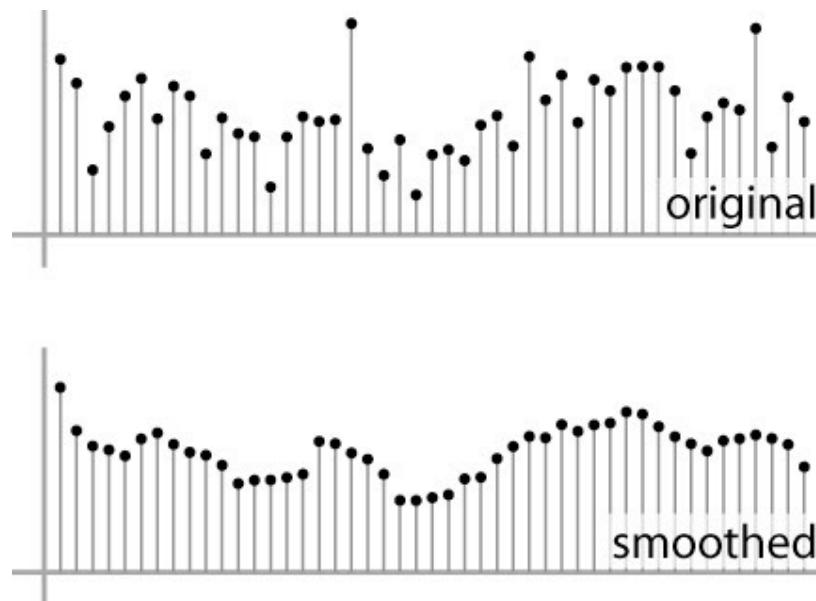
# Convolution warm-up

- 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

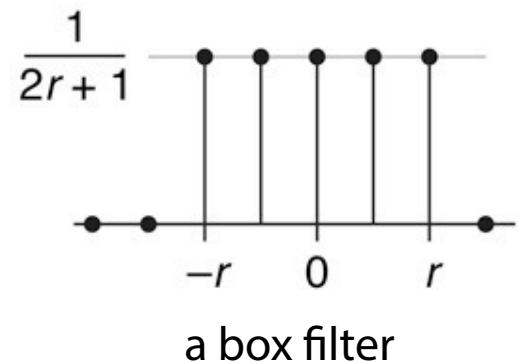
$$(a \star b)[i] = \sum_j a[j]b[i-j].$$

each sample gets its own weight (normally zero far away)

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

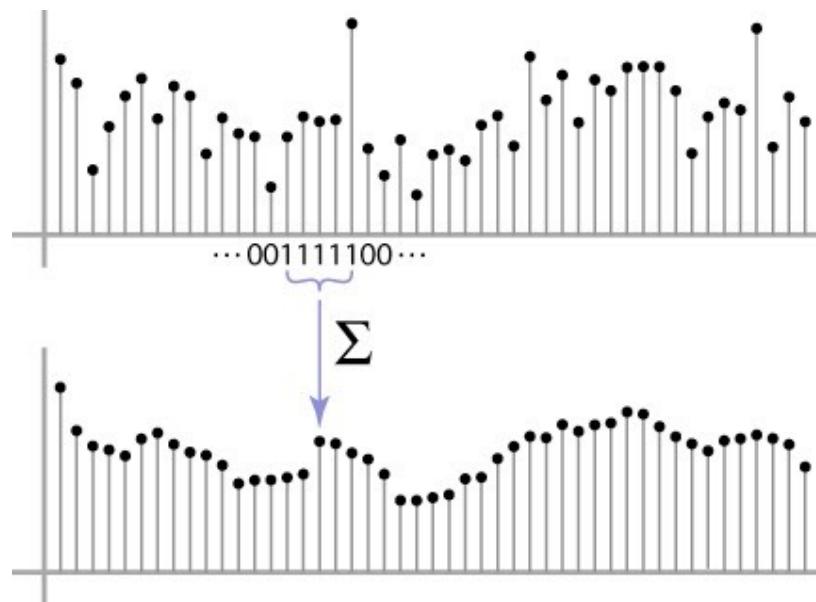
# Filters

- 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

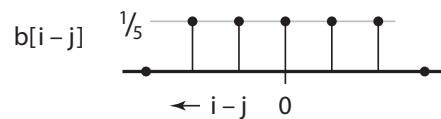
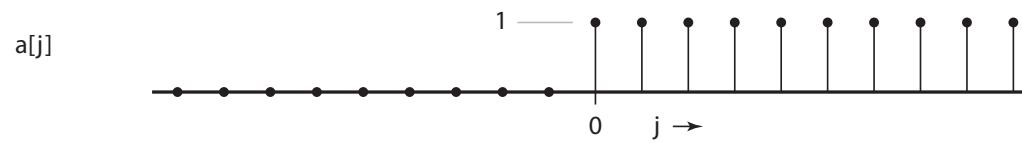


# Convolution and filtering

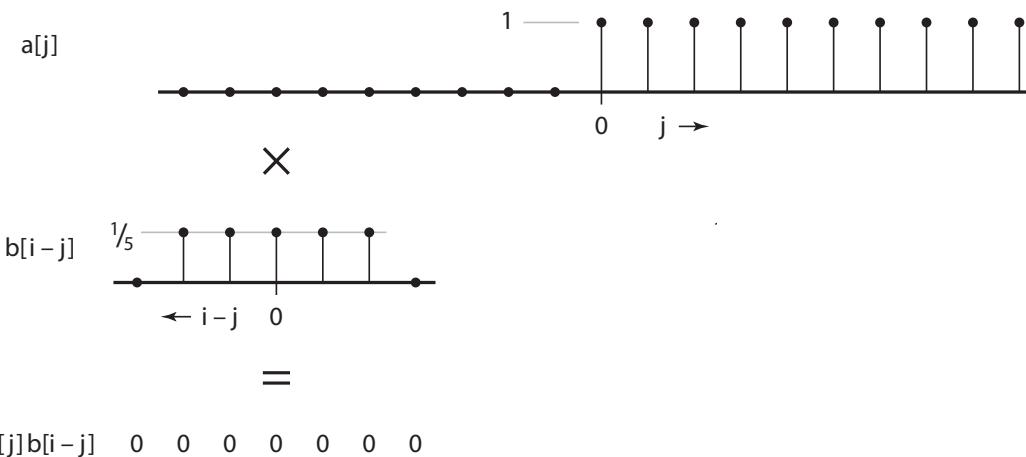
- Can express sliding average as convolution with a *box filter*
- $b_{\text{box}} = [\dots, 0, 1, 1, 1, 1, 1, 0, \dots]/5$



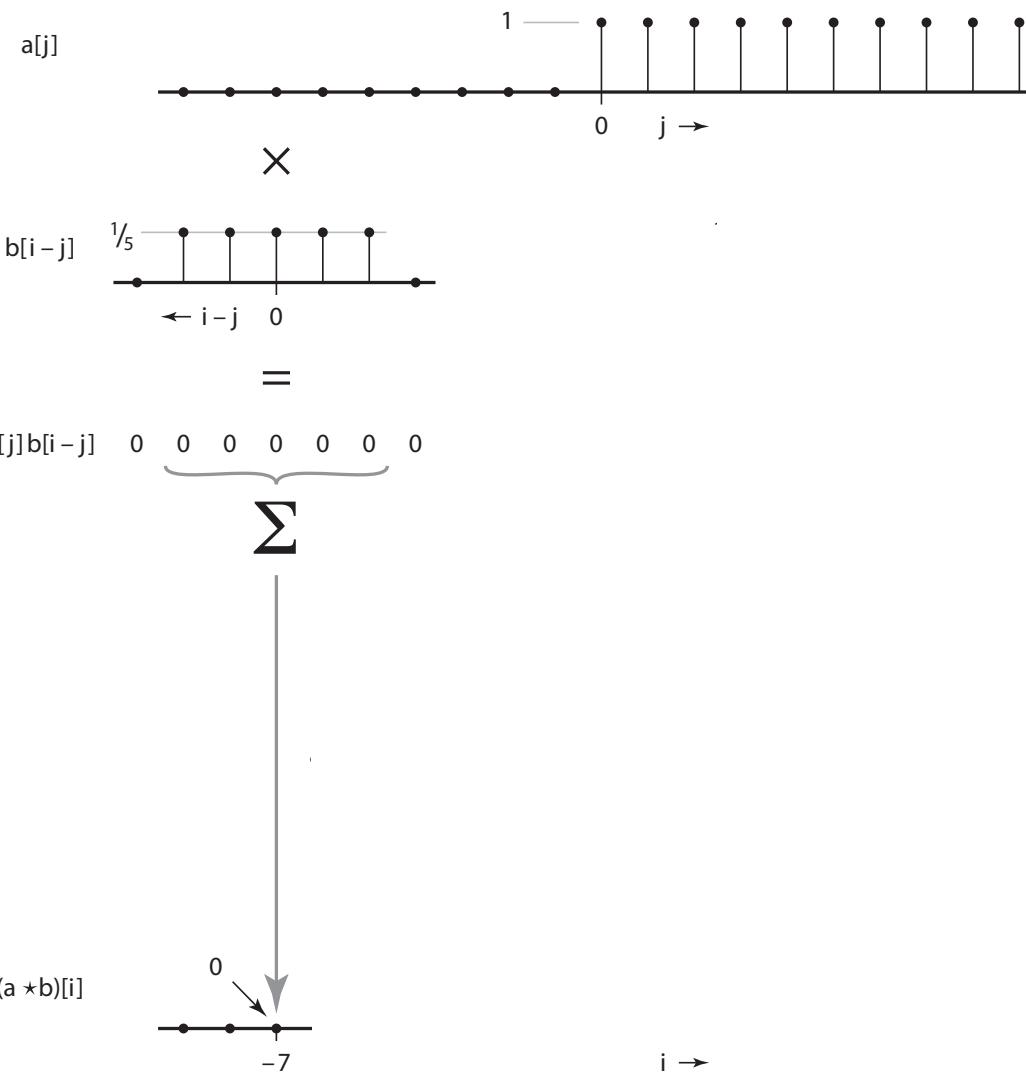
# Example: box and step



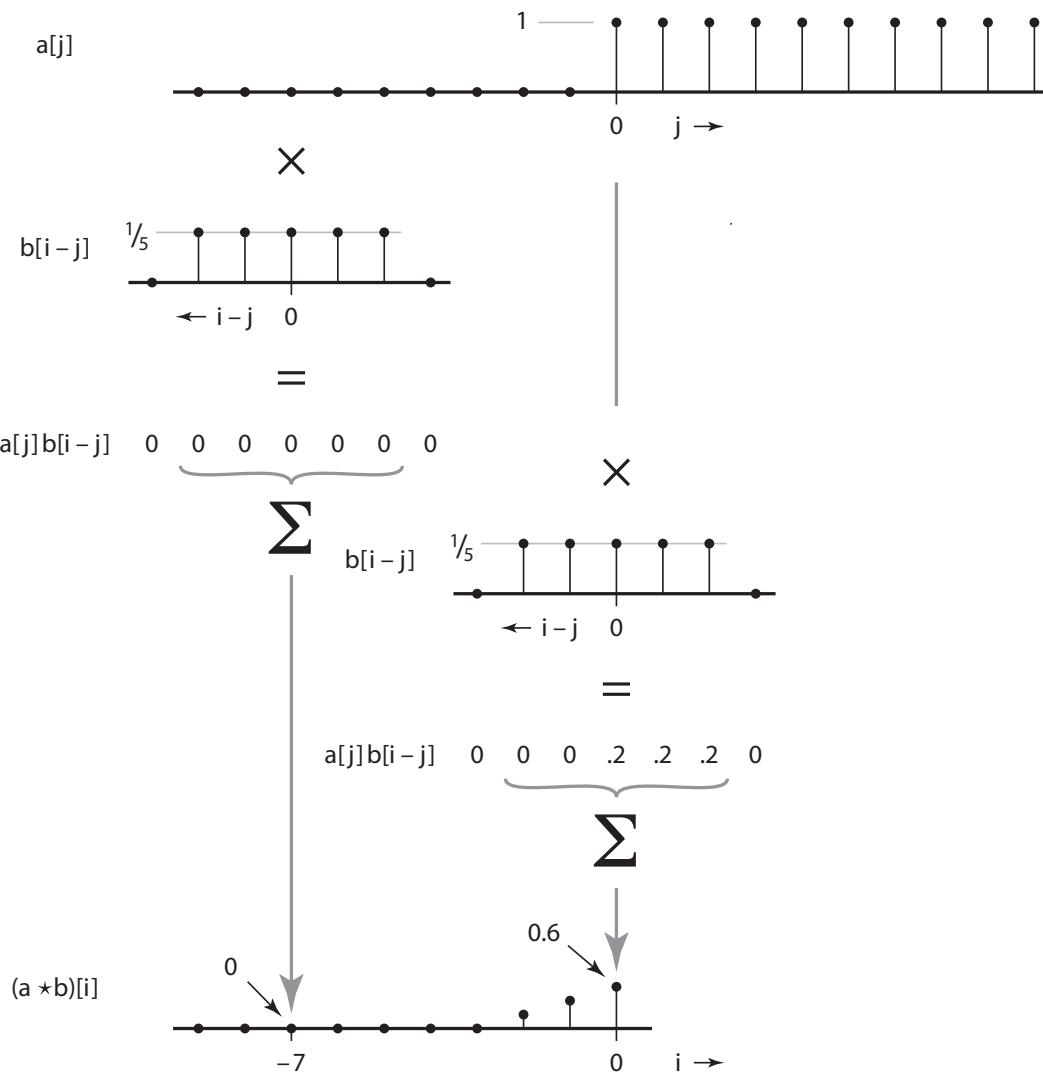
# Example: box and step



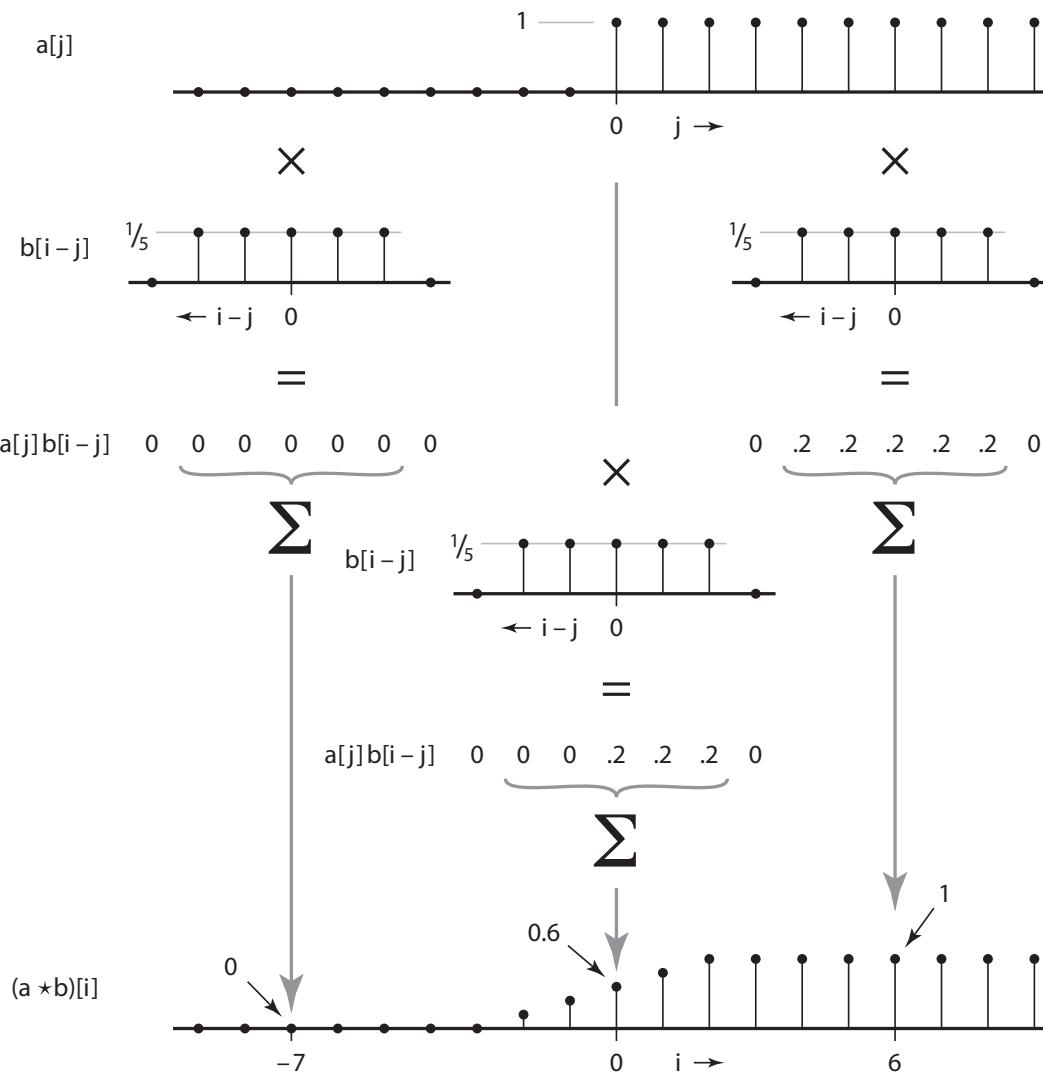
# Example: box and step



# Example: box and step

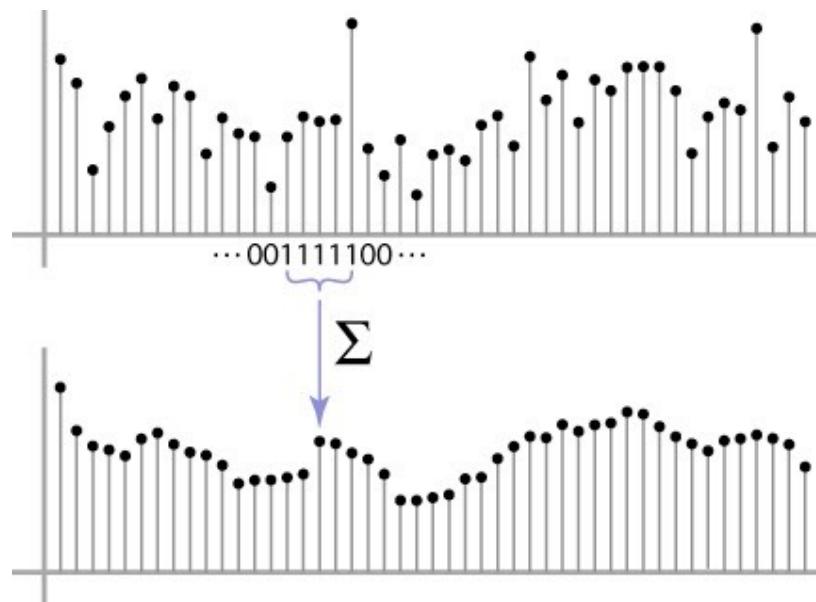


# Example: box and step



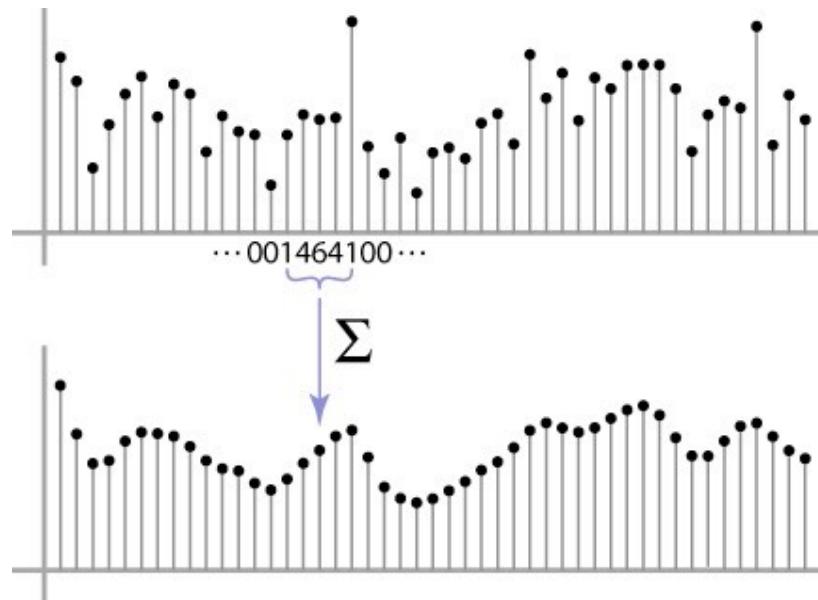
# Convolution and filtering

- 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:  $b = c \star a$
- Convolution is a multiplication-like operation
  - commutative  $a \star b = b \star a$
  - associative  $a \star (b \star c) = (a \star b) \star c$
  - distributes over addition  $a \star (b + c) = a \star b + a \star c$
  - scalars factor out  $\alpha a \star b = a \star \alpha b = \alpha(a \star b)$
  - identity: unit impulse  $e = [\dots, 0, 0, 1, 0, 0, \dots]$ 
$$a \star e = a$$
- Conceptually no distinction between filter and signal

# Discrete filtering in 2D

- Same equation, one more index

$$(a \star b)[i, j] = \sum_{i', j'} a[i', j'] b[i - i', j - j']$$

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

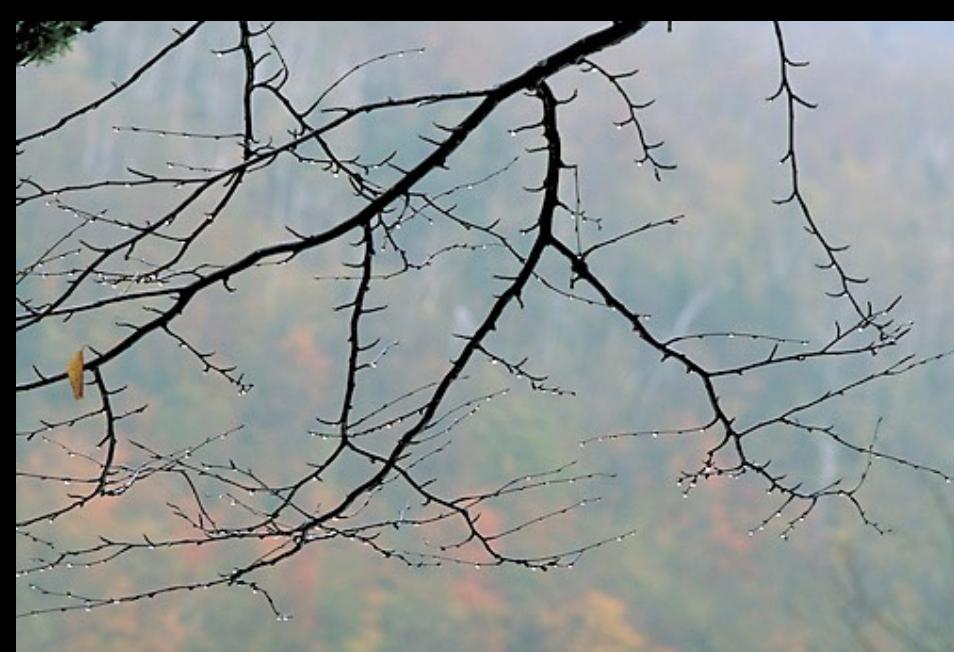
often apply several filters one after another:  $((a * b_1) * b_2) * b_3$

this is equivalent to applying one filter:  $a * (b_1 * b_2 * b_3)$



[Philip Greenspun]

original ▲ | ▼ box blur



sharpened ▲ | ▼ gaussian blur





[Philip Greenspun]

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:

$$\begin{aligned}(a_2 \star b)[i, j] &= \sum_{i'} \sum_{j'} a_2[i', j']b[i - i', j - j'] \\&= \sum_{i'} \sum_{j'} a_1[i']a_1[j']b[i - i', j - j'] \\&= \sum_{i'} a_1[i'] \left( \sum_{j'} a_1[j']b[i - i', j - j'] \right)\end{aligned}$$

# Separable filtering

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

|   |    |    |    |   |
|---|----|----|----|---|
| 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 |

=

|   |   |   |   |   |
|---|---|---|---|---|
| 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 |

\*

|   |   |   |   |   |
|---|---|---|---|---|
| 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 |

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

first, convolve with this



# Separable filtering

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

|   |    |    |    |   |
|---|----|----|----|---|
| 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 |

=

|   |   |   |   |   |
|---|---|---|---|---|
| 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 |

\*

|   |   |   |   |   |
|---|---|---|---|---|
| 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 |

second, convolve with this

first, convolve with this

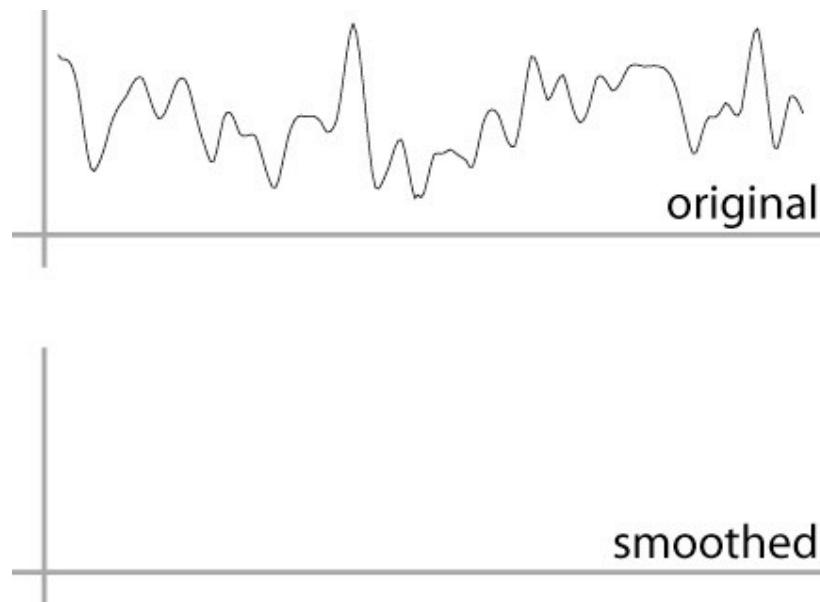
$$\sum_{i'} a_1[i'] \left( \sum_{j'} a_1[j'] b[i - i', j - j'] \right)$$

# Continuous convolution: warm-up

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

output is continuous

integration replaces summation

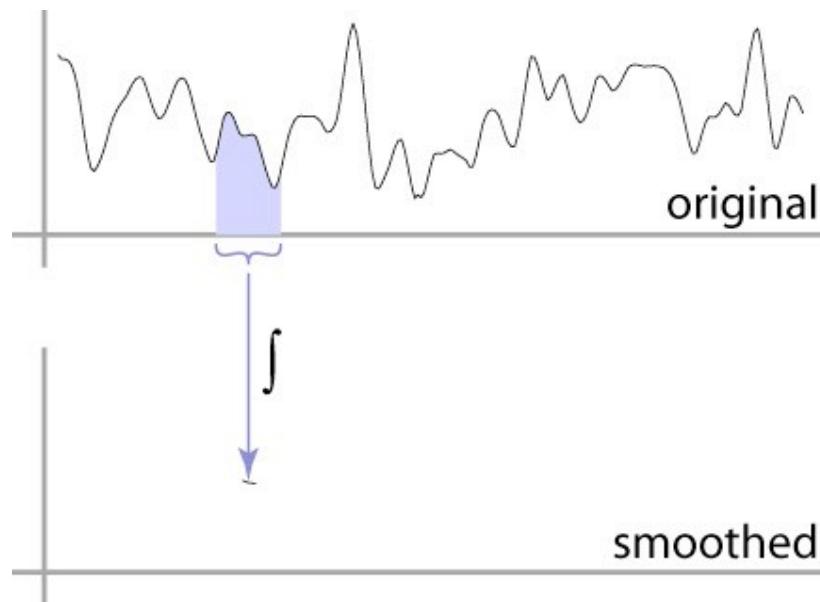


# Continuous convolution: warm-up

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

output is continuous

integration replaces summation

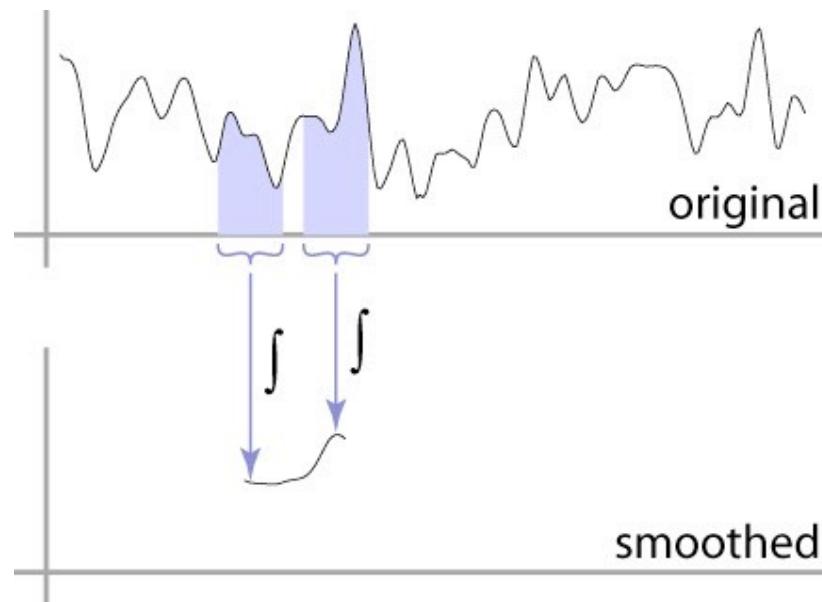


# Continuous convolution: warm-up

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

output is continuous

integration replaces summation

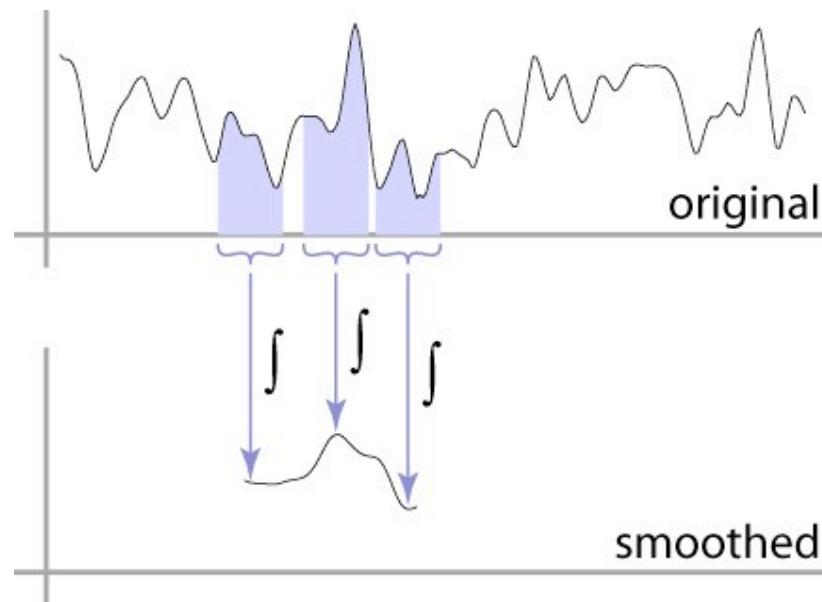


# Continuous convolution: warm-up

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

output is continuous

integration replaces summation

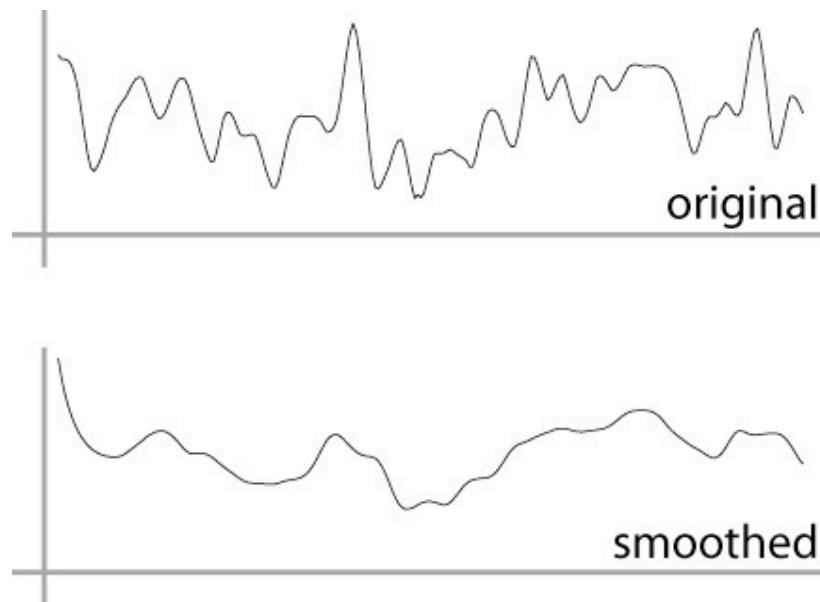


# Continuous convolution: warm-up

- 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)

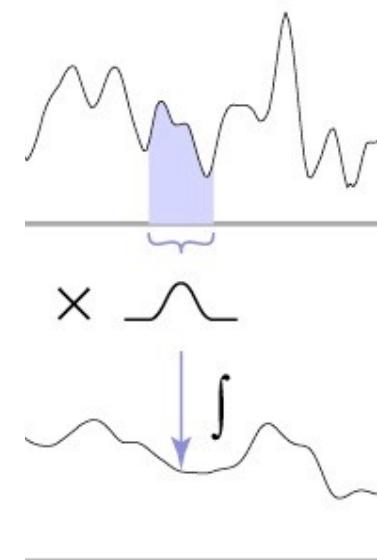
- Convolution just introduces weights

$$(f * g)(x) = \int_{-\infty}^{\infty} f(t)g(x-t) dt$$

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

$$(a \star f)(x) = \sum_i a[i]f(x - i)$$

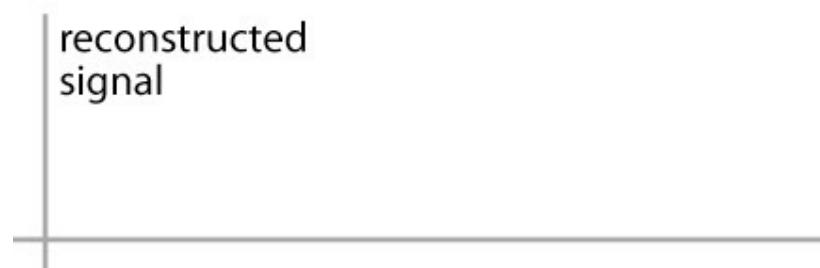
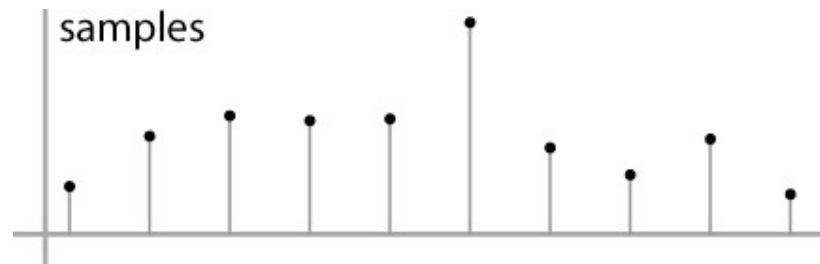
$$(a \star f)(x, y) = \sum_{i,j} a[i, j]f(x - i, y - j)$$

input: a sequence and a continuous function

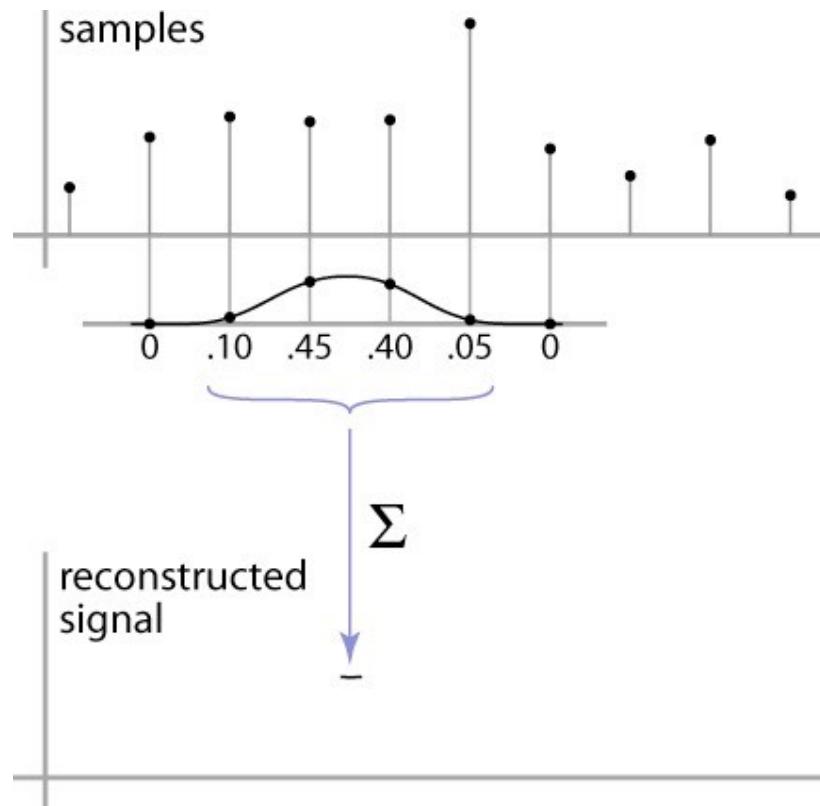
output: a continuous function

used for reconstruction and resampling

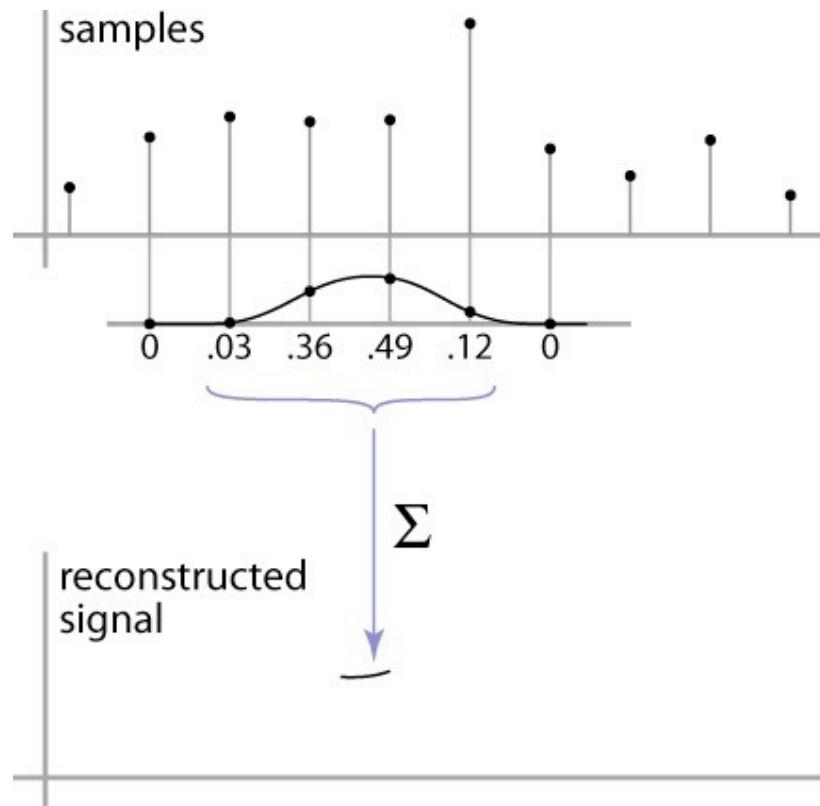
# Continuous-discrete convolution



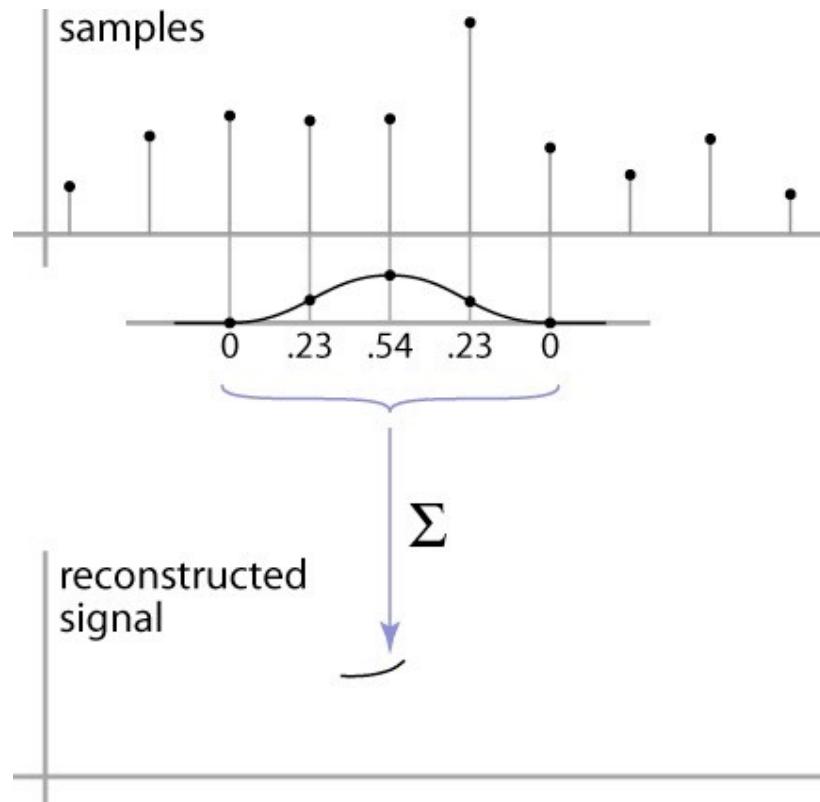
# Continuous-discrete convolution



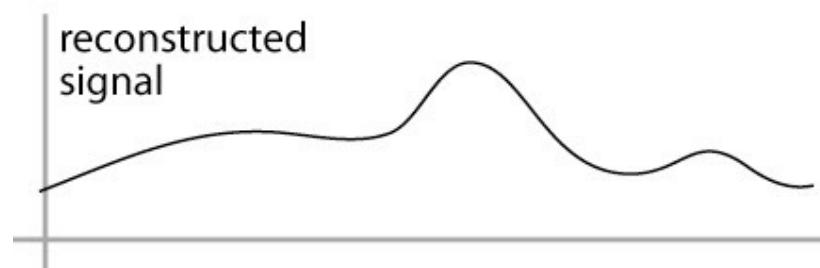
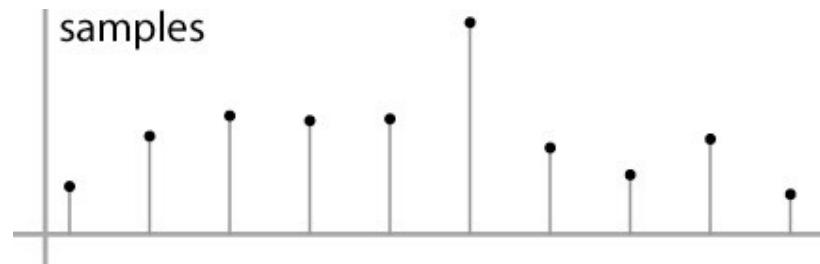
# Continuous-discrete convolution



# Continuous-discrete convolution



# Continuous-discrete convolution

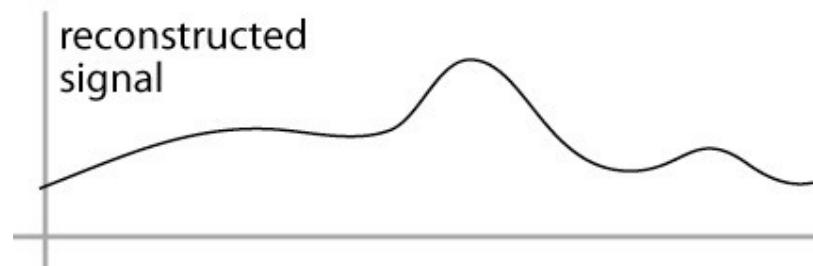
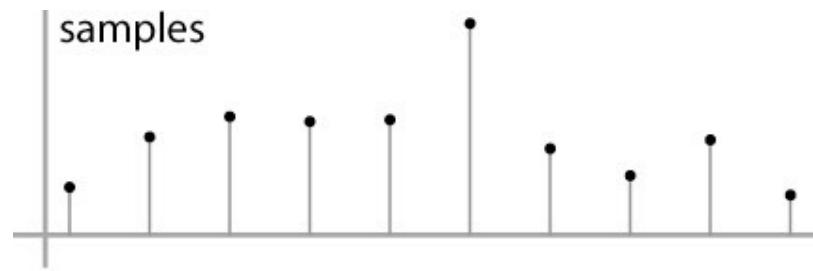


# Resampling

- 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”

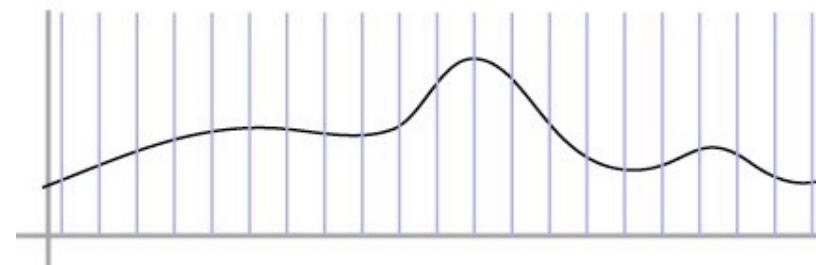
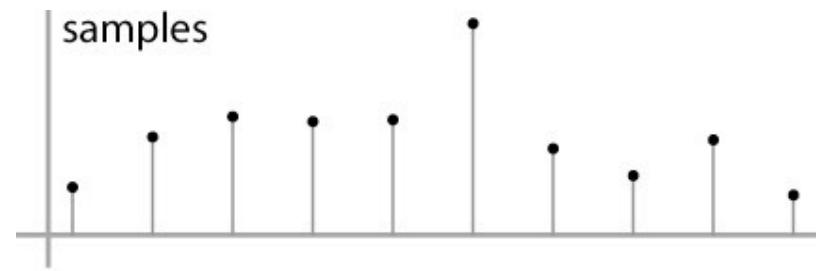
# Resampling

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



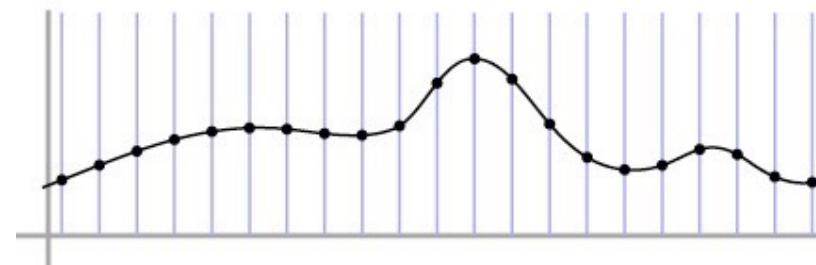
# Resampling

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



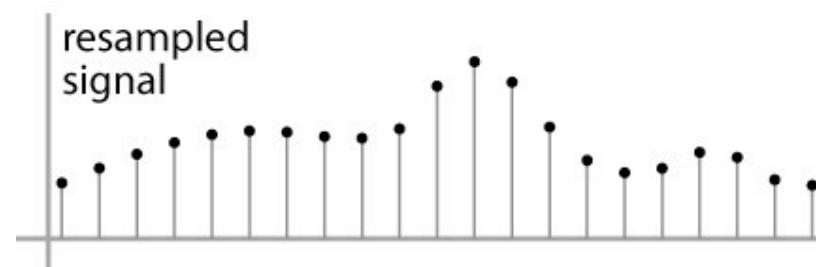
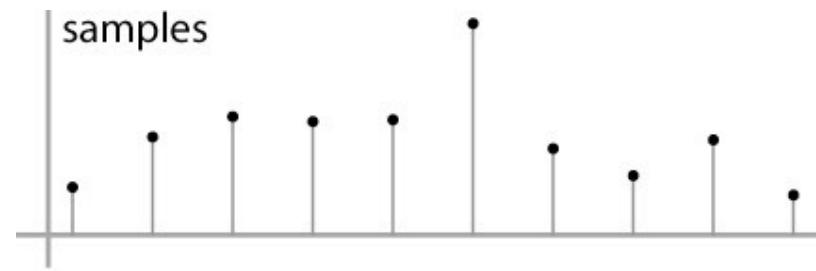
# Resampling

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



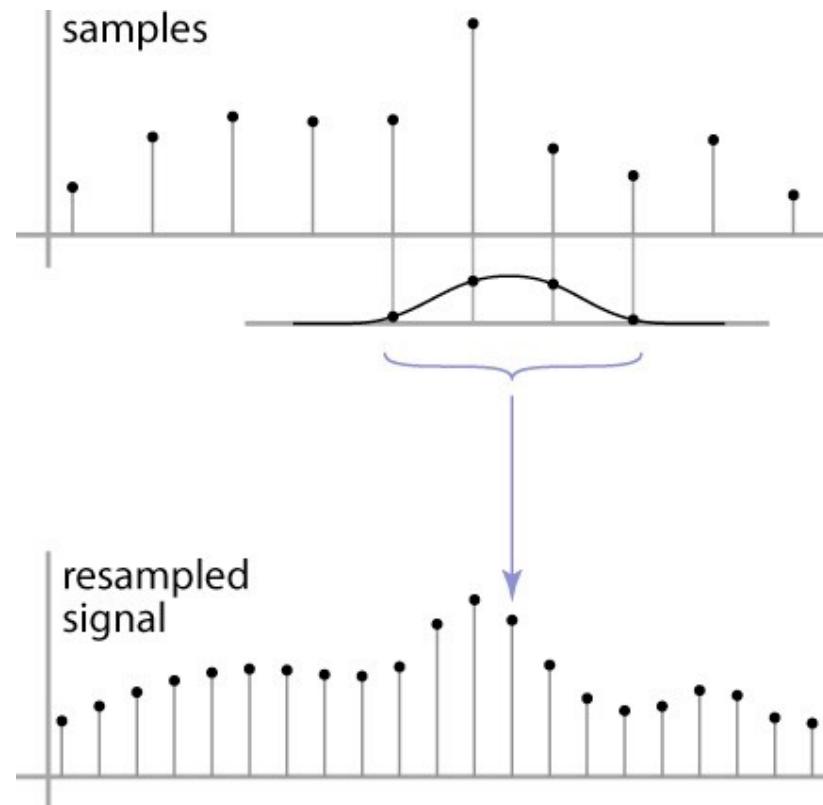
# Resampling

- 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



# Cont.-disc. convolution in 2D

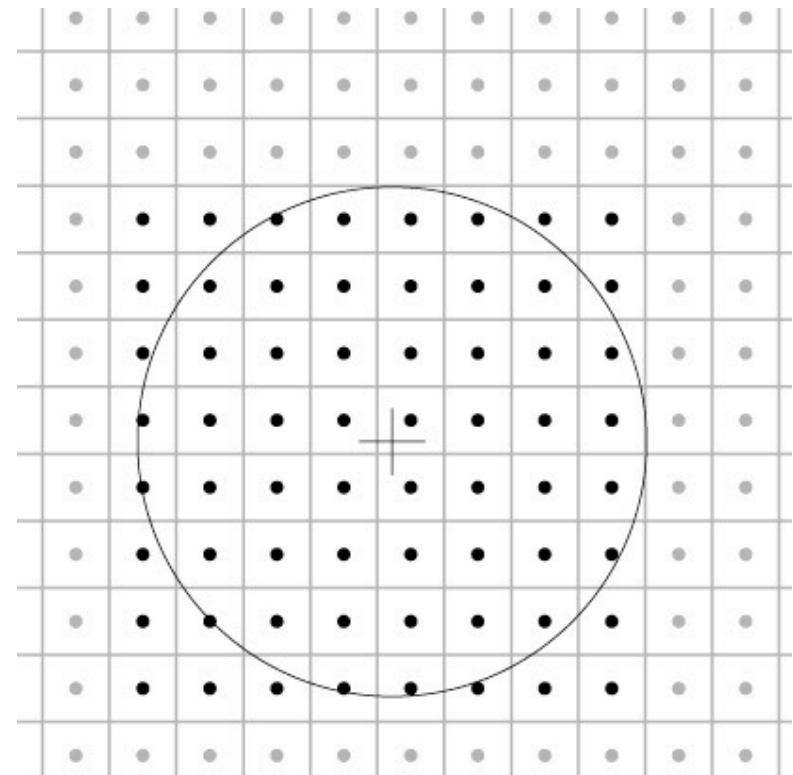
- same convolution—just two variables now

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

loop over nearby pixels,  
average using filter weight

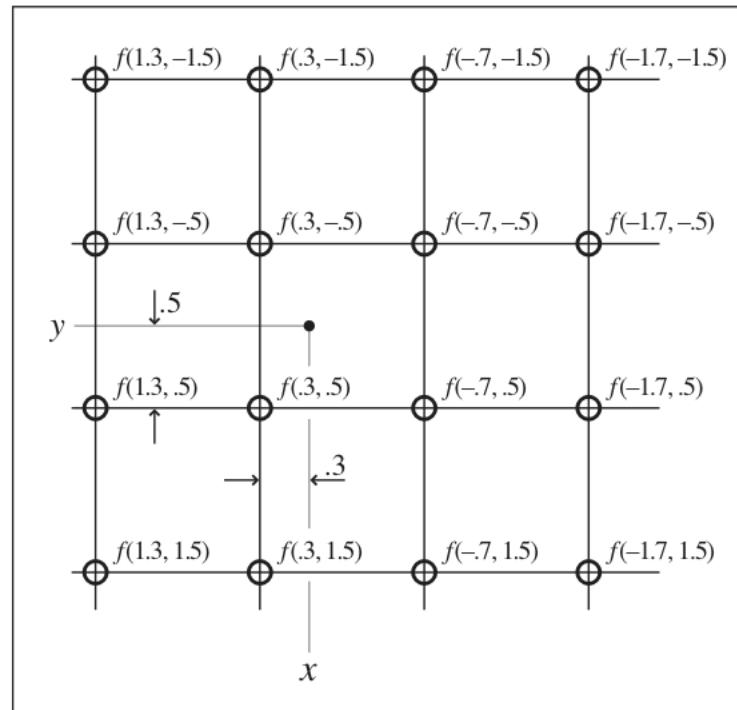
looks like discrete filter,  
but offsets are not integers  
and filter is continuous

remember placement of filter  
relative to grid is variable



# Cont.-disc. convolution in 2D

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



# 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:
$$f_2(x, y) = f_1(x)f_1(y)$$
- 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

[Philip Greenspun]



two-stage resampling using a  
separable filter

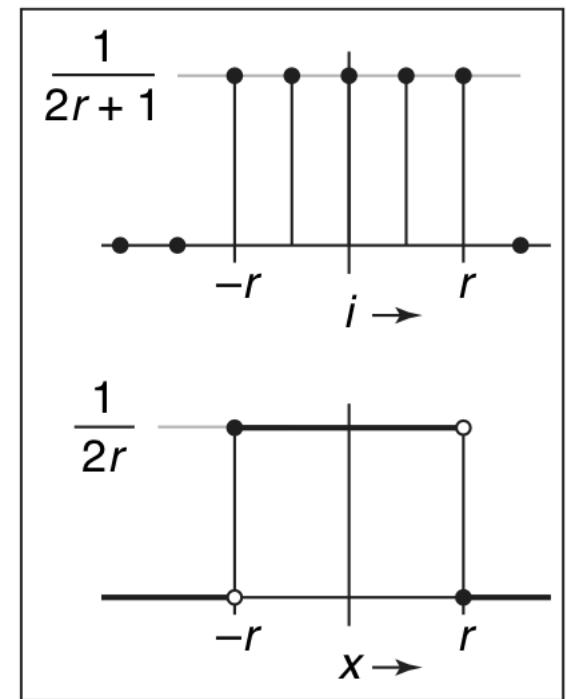
# A gallery of filters

- 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

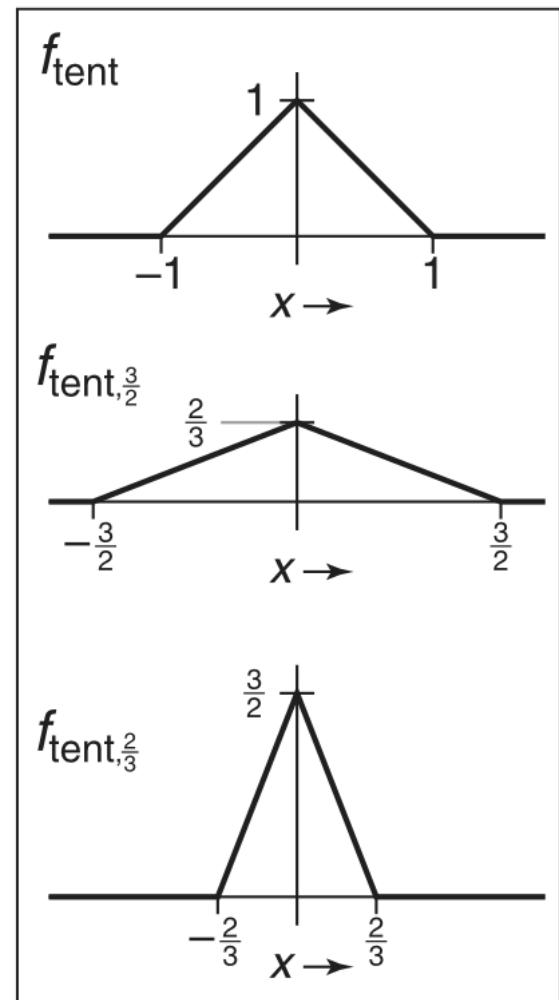
$$a_{\text{box},r}[i] = \begin{cases} 1/(2r+1) & |i| \leq r, \\ 0 & \text{otherwise.} \end{cases}$$

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

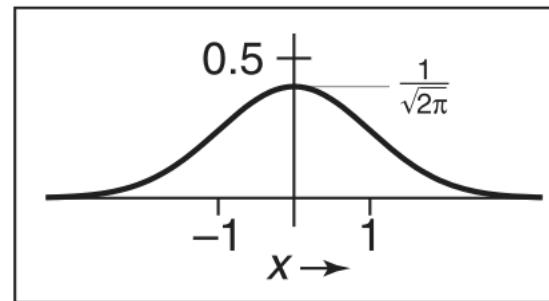


# Tent filter

$$f_{\text{tent}}(x) = \begin{cases} 1 - |x| & |x| < 1, \\ 0 & \text{otherwise;} \end{cases}$$
$$f_{\text{tent},r}(x) = \frac{f_{\text{tent}}(x/r)}{r}.$$

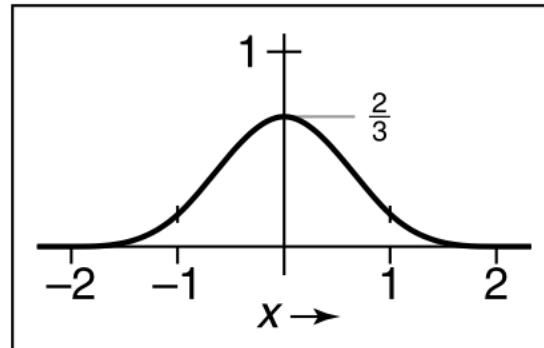


# Gaussian filter



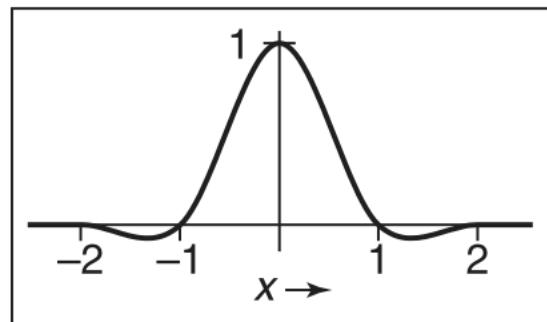
$$f_g(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}.$$

# B-Spline cubic



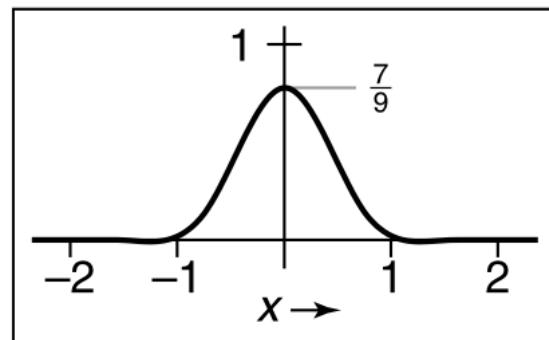
$$f_B(x) = \frac{1}{6} \begin{cases} -3(1 - |x|)^3 + 3(1 - |x|)^2 + 3(1 - |x|) + 1 & -1 \leq x \leq 1, \\ (2 - |x|)^3 & 1 \leq |x| \leq 2, \\ 0 & \text{otherwise.} \end{cases}$$

# Catmull-Rom cubic



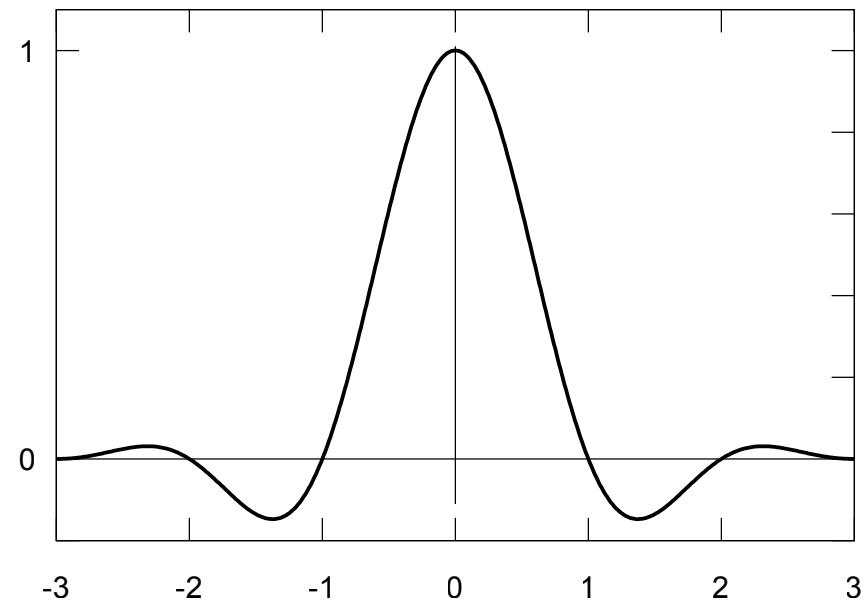
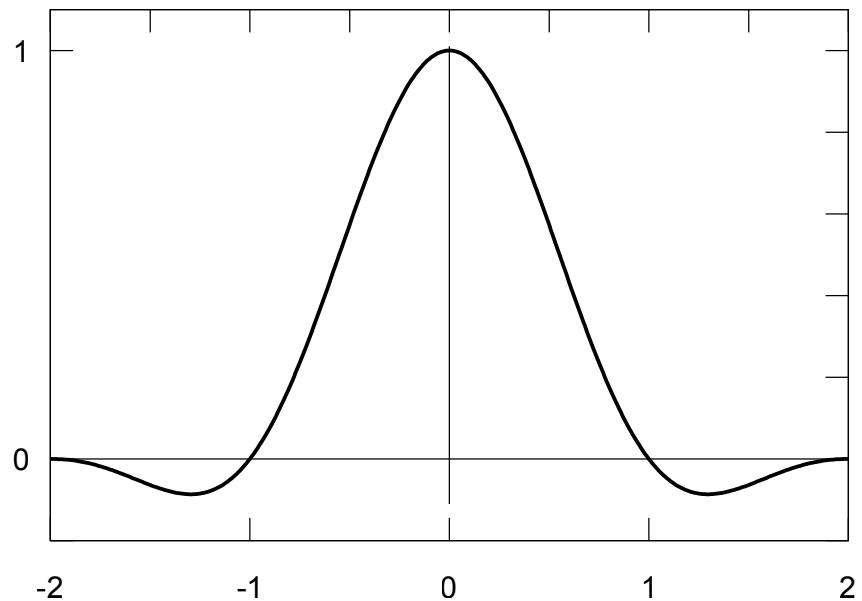
$$f_C(x) = \frac{1}{2} \begin{cases} -3(1 - |x|)^3 + 4(1 - |x|)^2 + (1 - |x|) & -1 \leq x \leq 1, \\ (2 - |x|)^3 - (2 - |x|)^2 & 1 \leq |x| \leq 2, \\ 0 & \text{otherwise.} \end{cases}$$

# Michell-Netravali cubic



$$\begin{aligned}f_M(x) &= \frac{1}{3}f_B(x) + \frac{2}{3}f_C(x) \\&= \frac{1}{18} \begin{cases} -21(1 - |x|)^3 + 27(1 - |x|)^2 + 9(1 - |x|) + 1 & -1 \leq x \leq 1, \\ 7(2 - |x|)^3 - 6(2 - |x|)^2 & 1 \leq |x| \leq 2, \\ 0 & \text{otherwise.} \end{cases}\end{aligned}$$

# Lanczos



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

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

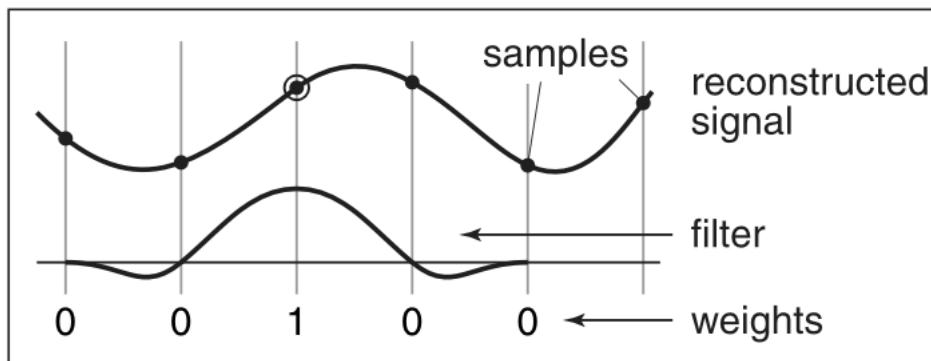
$$f_{L3}(x) = \begin{cases} \text{sinc}(x) \text{sinc}(x/3) & |x| < 3 \\ 0 & \text{otherwise} \end{cases}$$

# 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  $\text{output}[i, j] = \text{input}[\text{round}(x(i)), \text{round}(y(j))]$
  - $x(i)$  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

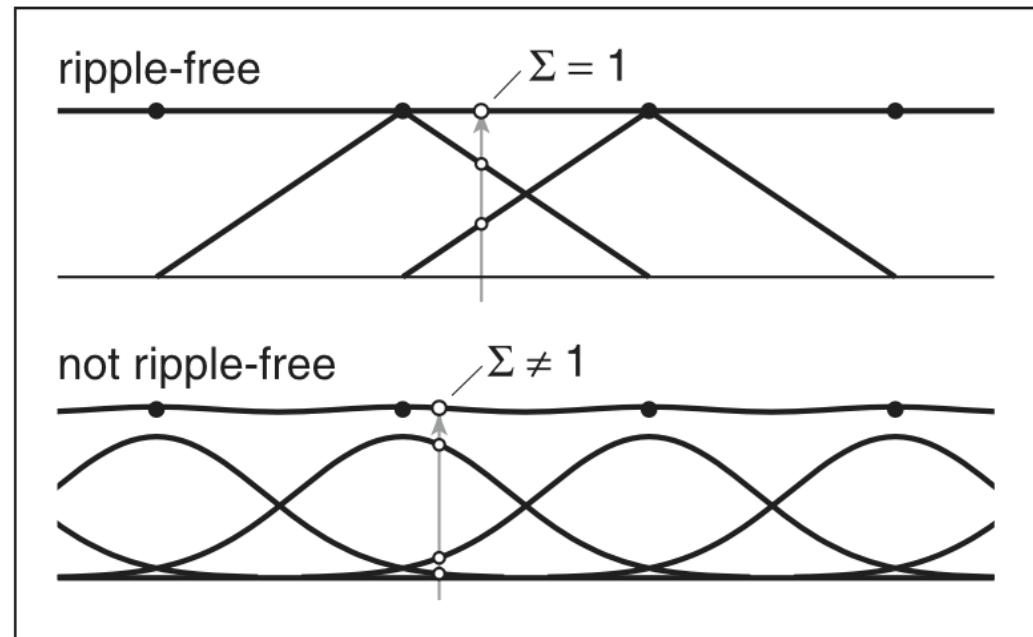
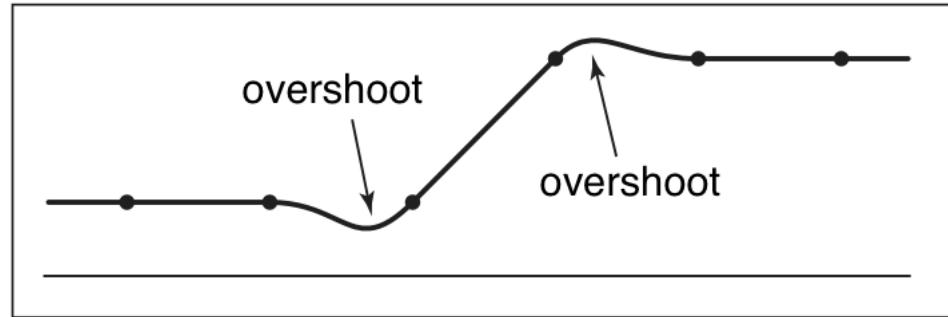


interpolating filter used for reconstruction

# Ringing, overshoot, ripples

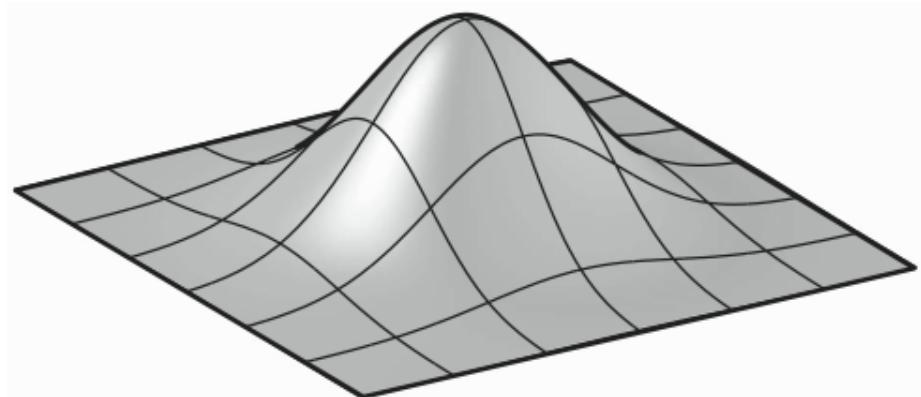
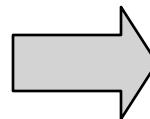
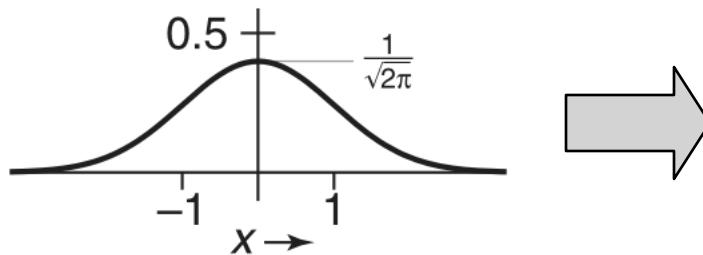
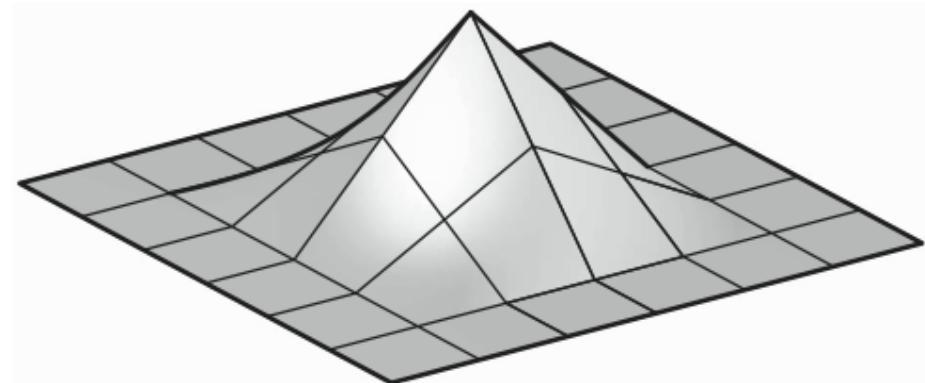
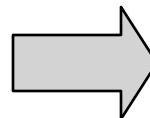
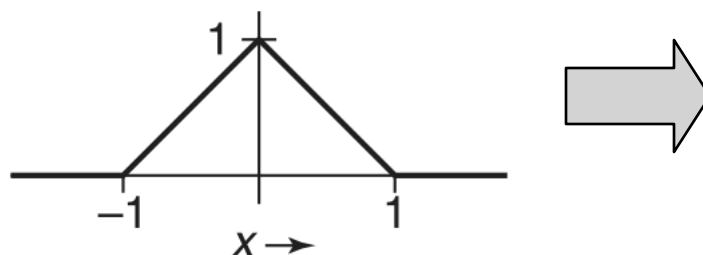
- Overshoot
  - caused by negative filter values
- Ripples
  - constant in, non-const. out
  - ripple free when:

$$\sum_i f(x + i) = 1 \quad \text{for all } x.$$



# 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



[Philip Greenspun]

# Yucky details

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

need to extrapolate

methods:

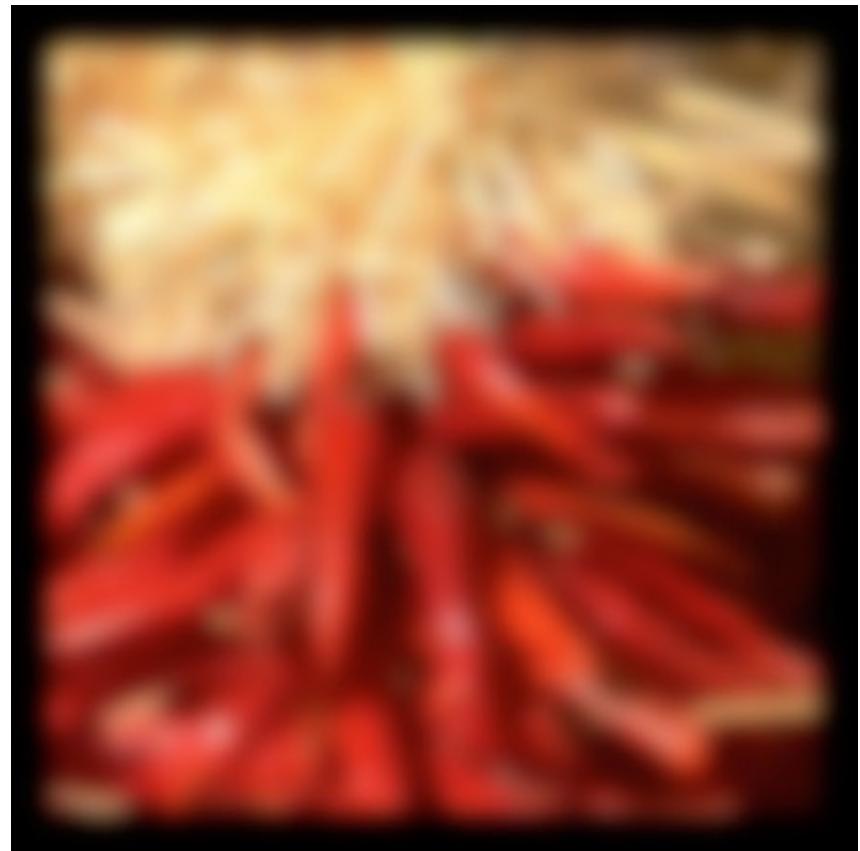
- clip filter (black)
- wrap around
- copy edge
- reflect across edge
- vary filter near edge



[Philip Greenspun]

# Yucky details

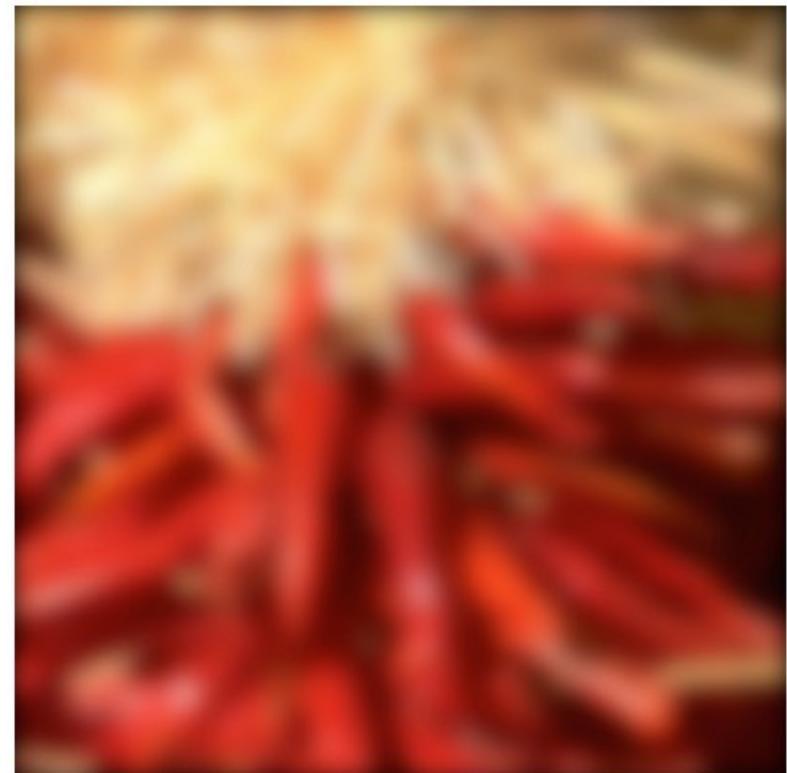
- What about near the edge?
  - the filter window falls off the edge of the image
  - need to extrapolate
  - methods:
    - clip filter (black)
    - wrap around
    - copy edge
    - reflect across edge
    - vary filter near edge



[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



[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



[Philip Greenspun]

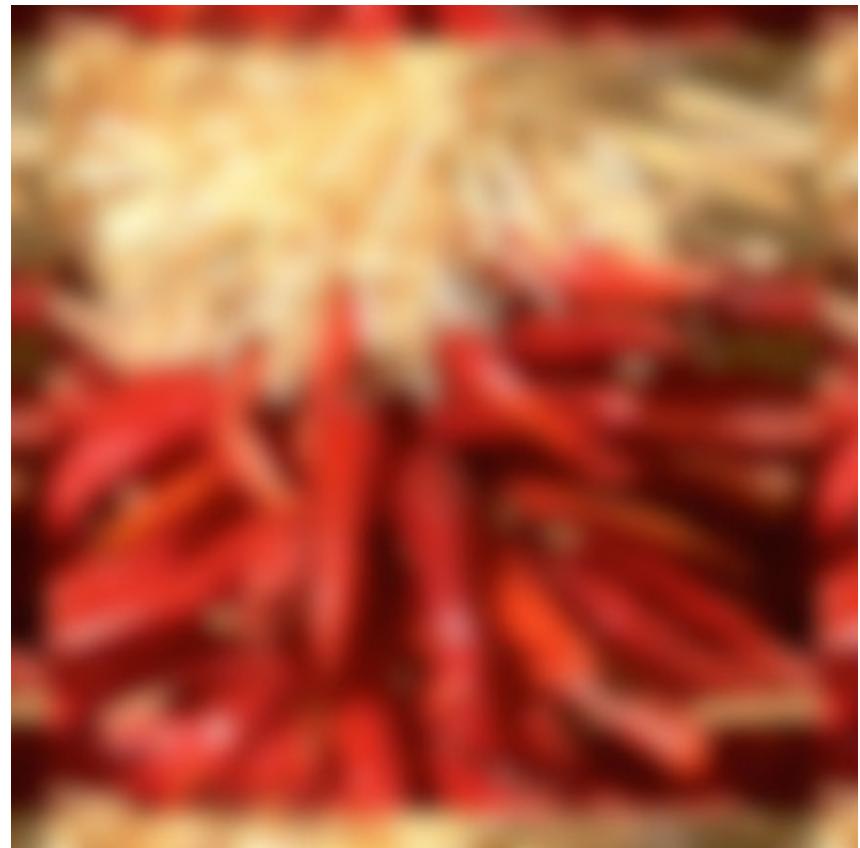
# Yucky details

- 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



[Philip Greenspun]

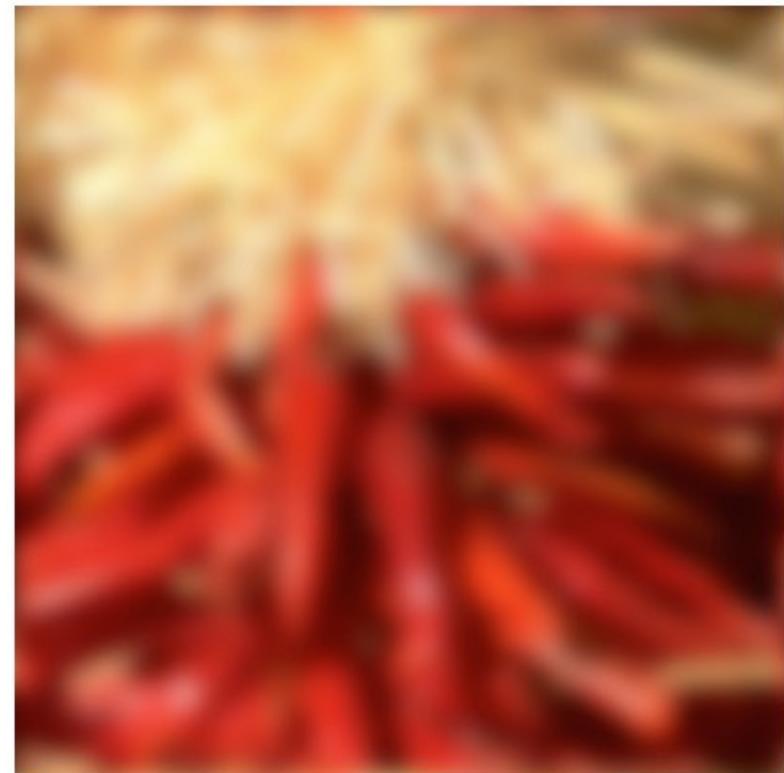
# **Yucky details**

- 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



[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



[Philip Greenspun]

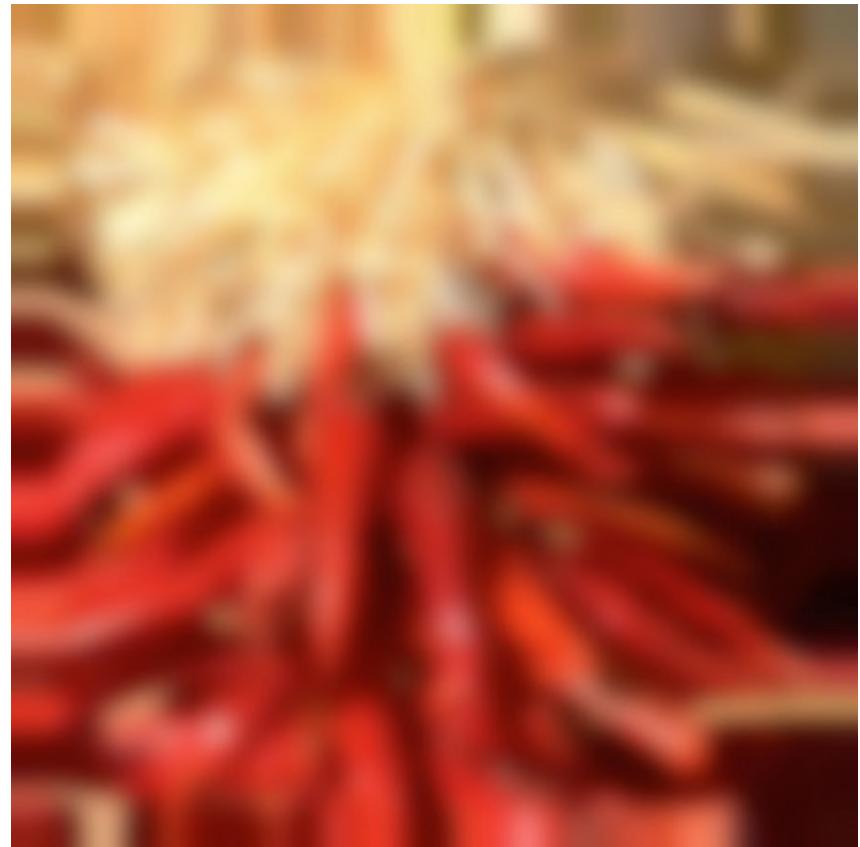
# Yucky details

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

need to extrapolate

methods:

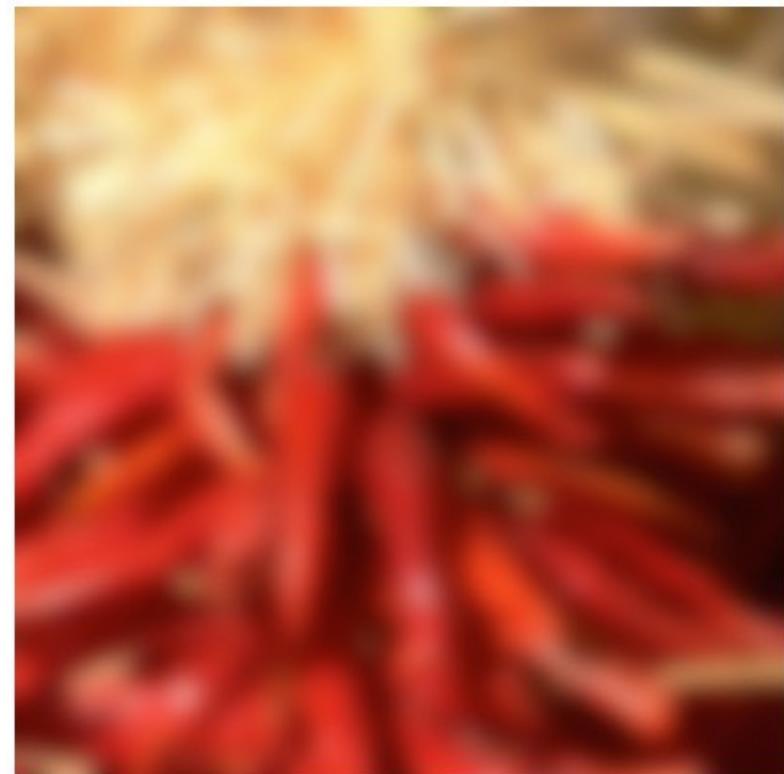
- clip filter (black)
- wrap around
- copy edge
- reflect across edge
- vary filter near edge



[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



[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



[Philip Greenspun]

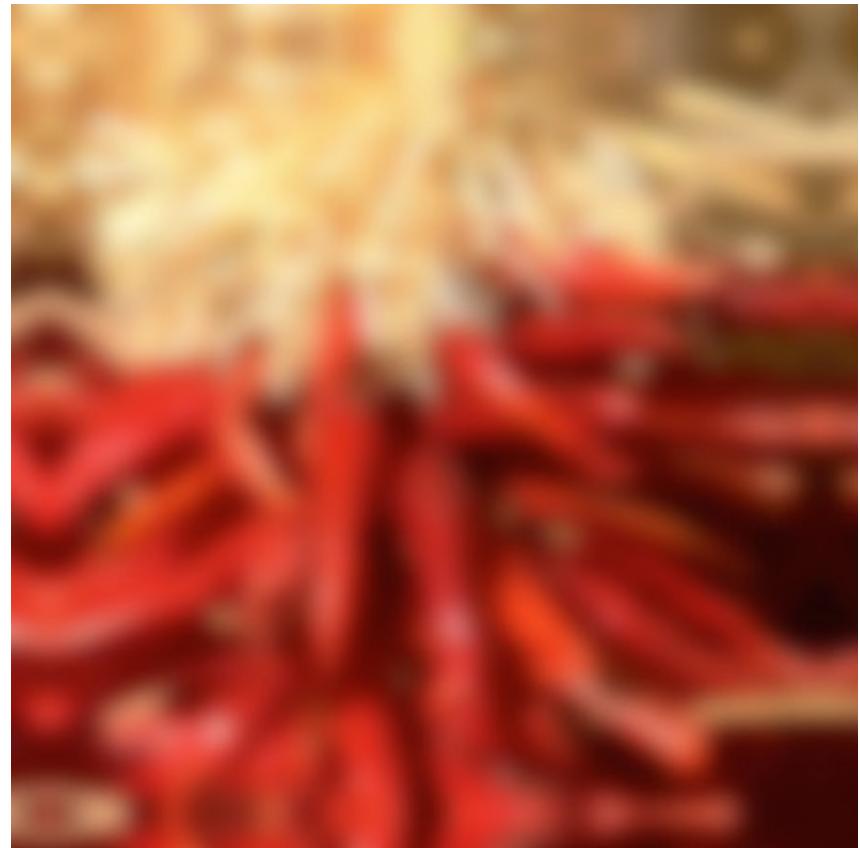
# Yucky details

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

need to extrapolate

methods:

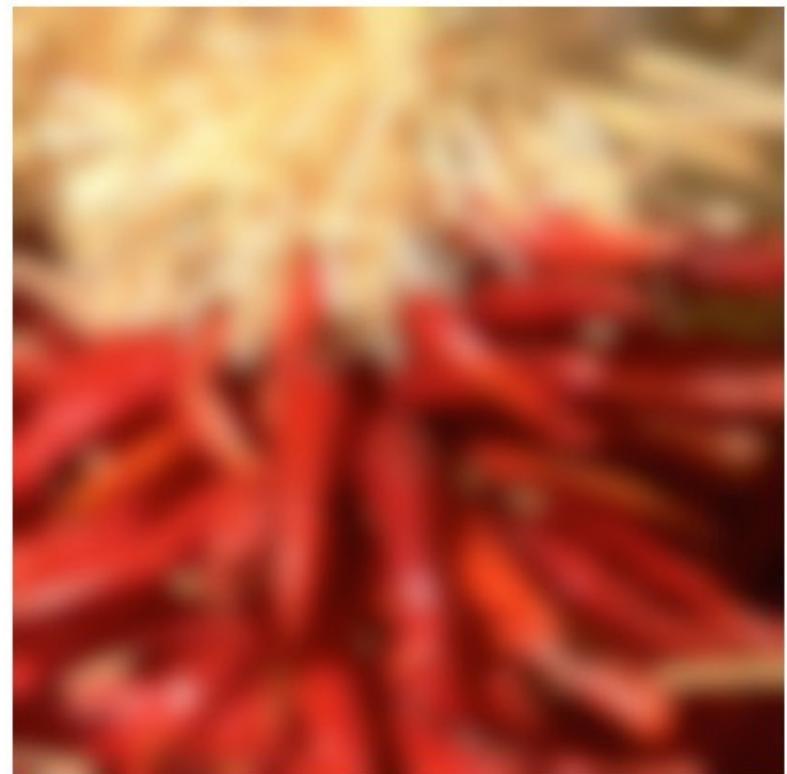
- clip filter (black)
- wrap around
- copy edge
- reflect across edge
- vary filter near edge



[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

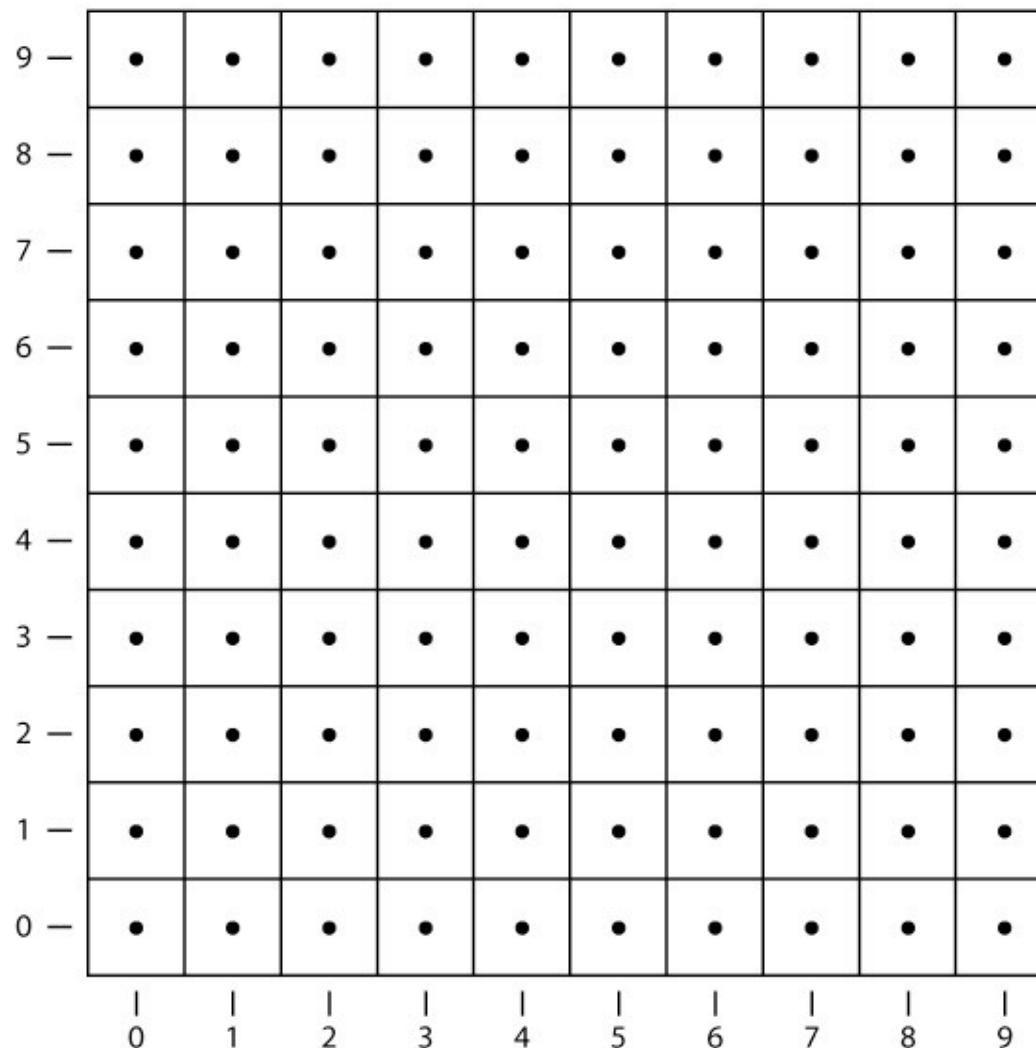


[Philip Greenspun]

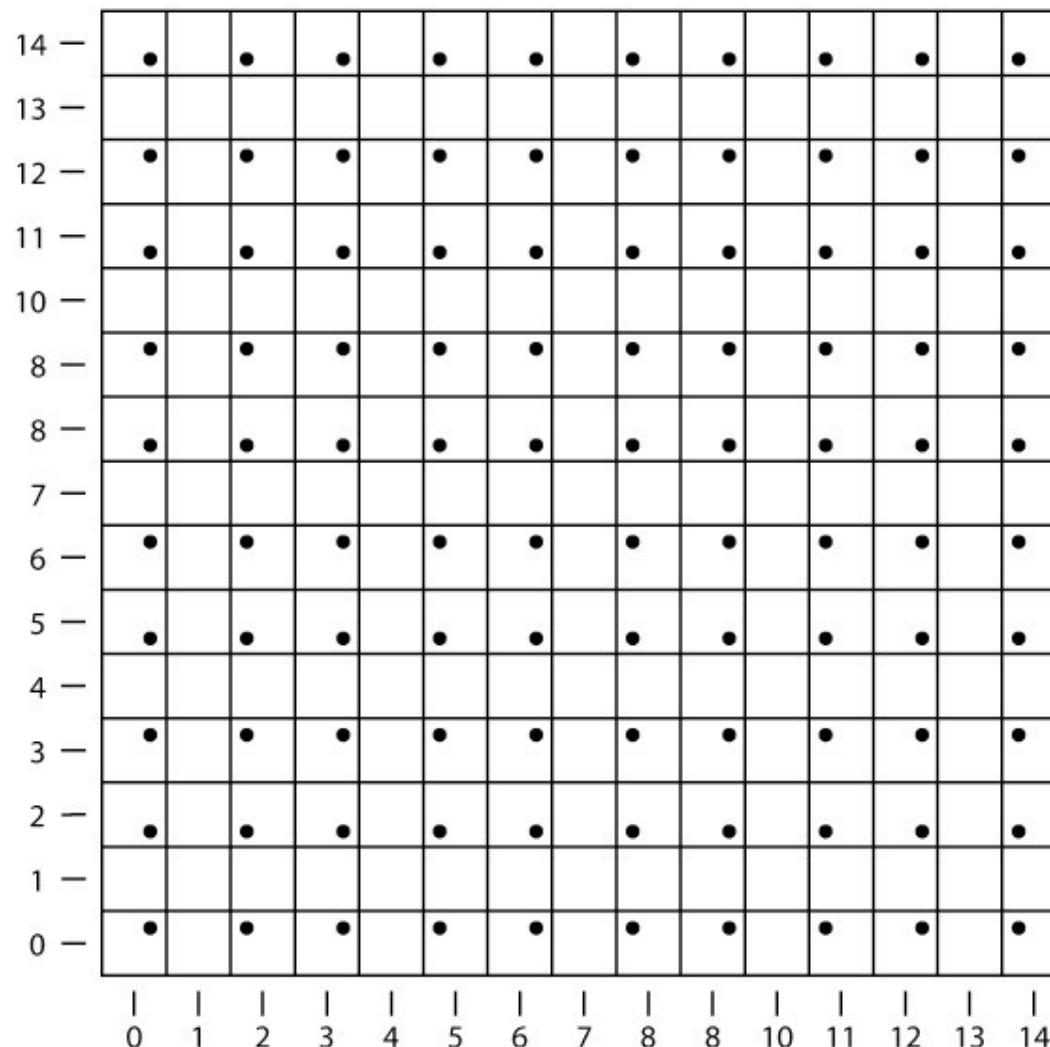
# Reducing and enlarging

- 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

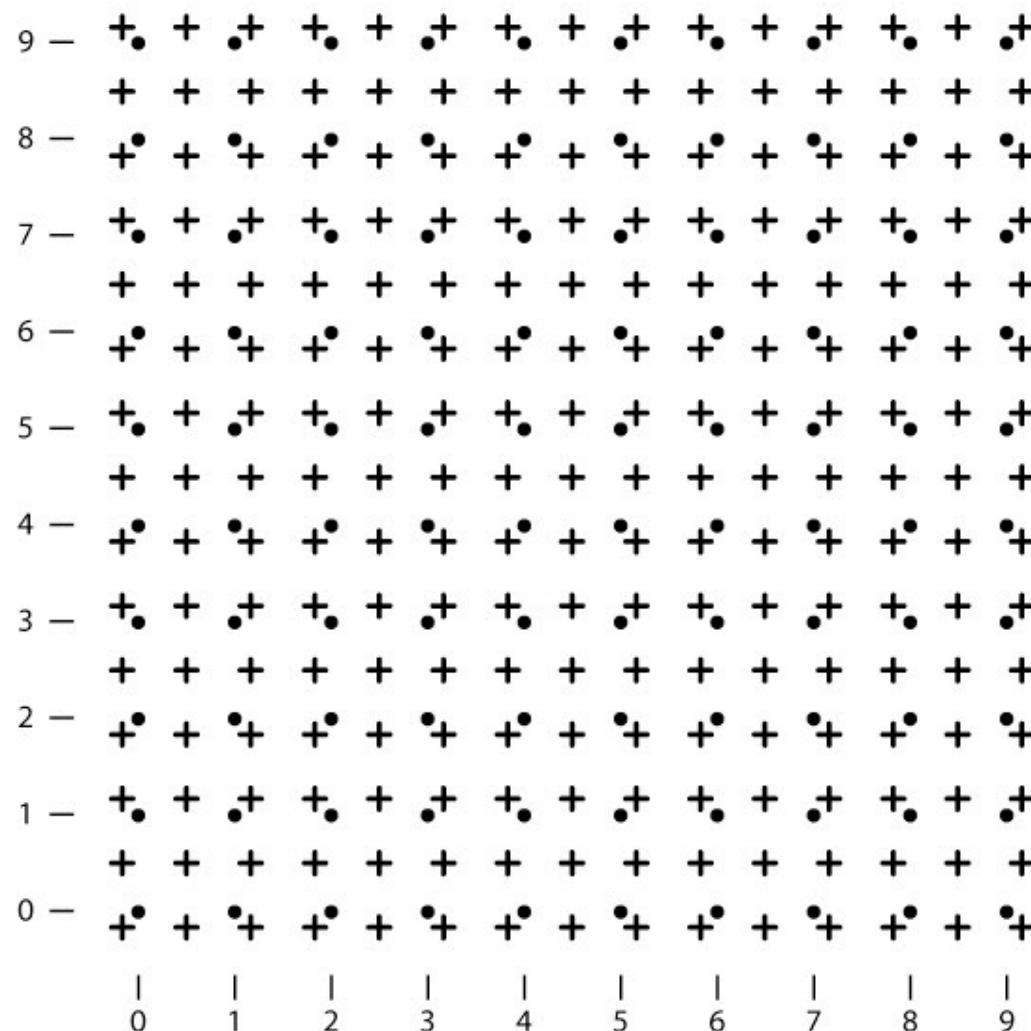
# Resampling example



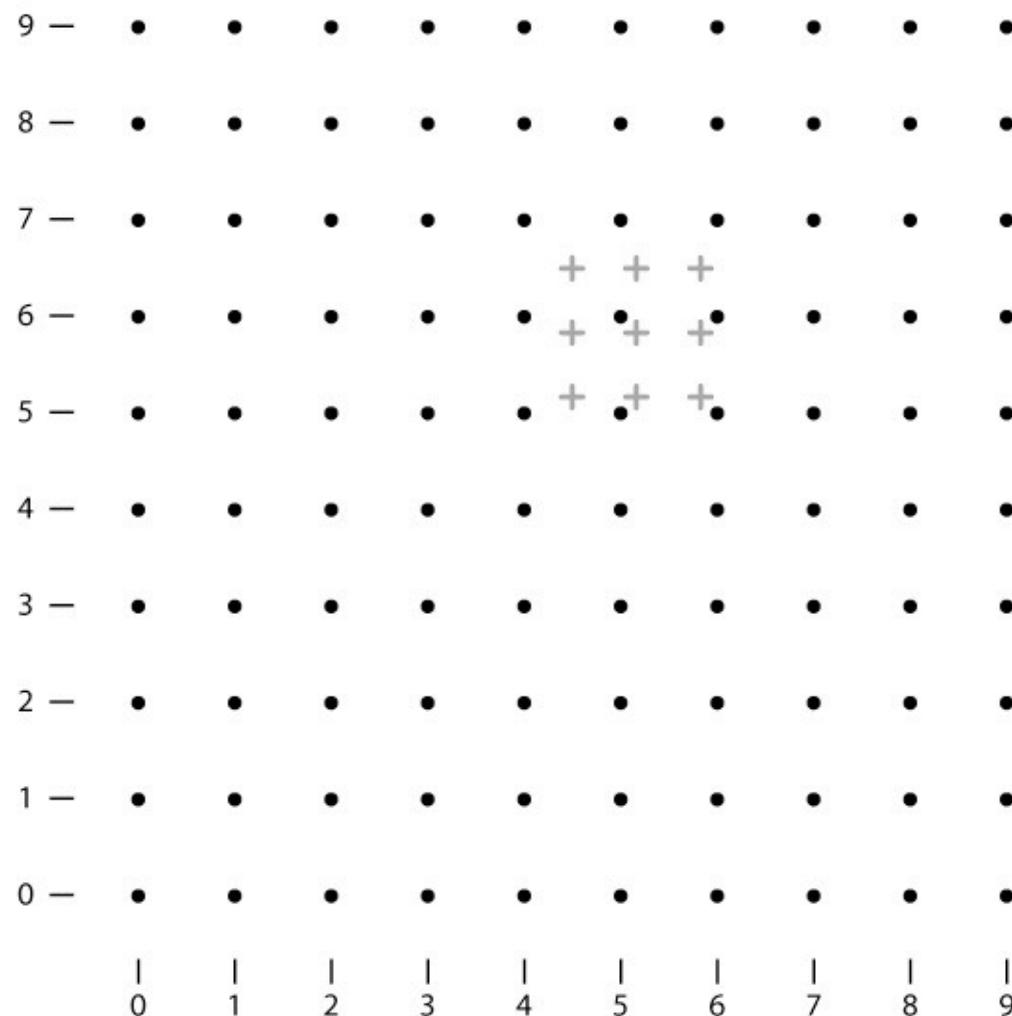
# Resampling example



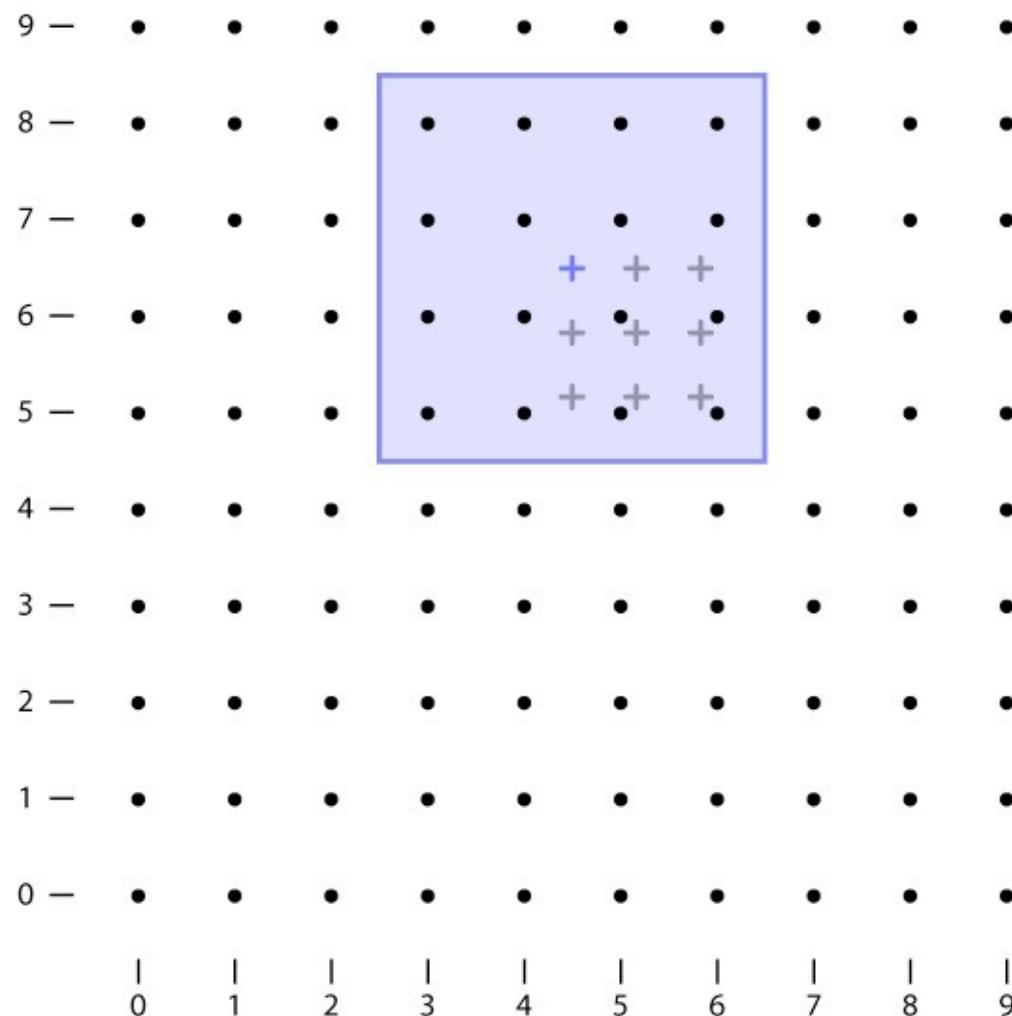
# Resampling example



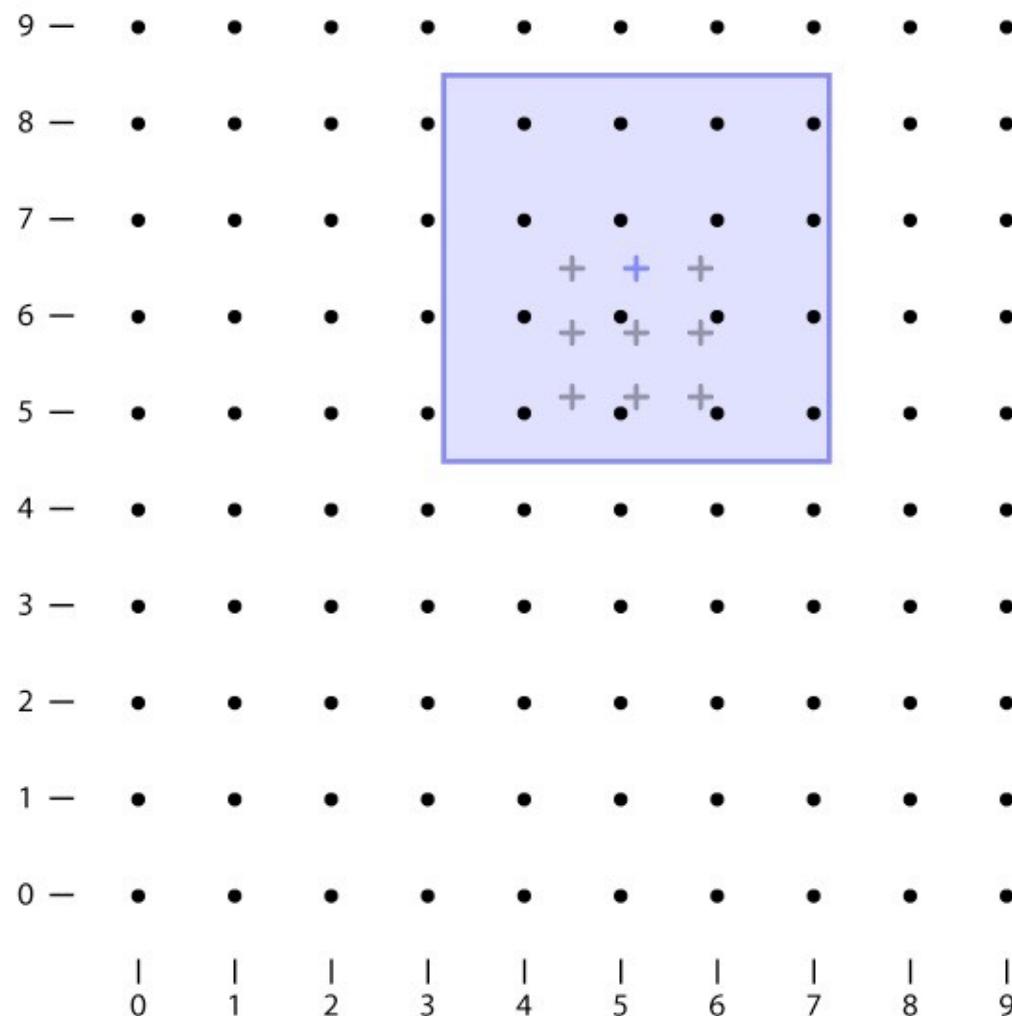
# Resampling example



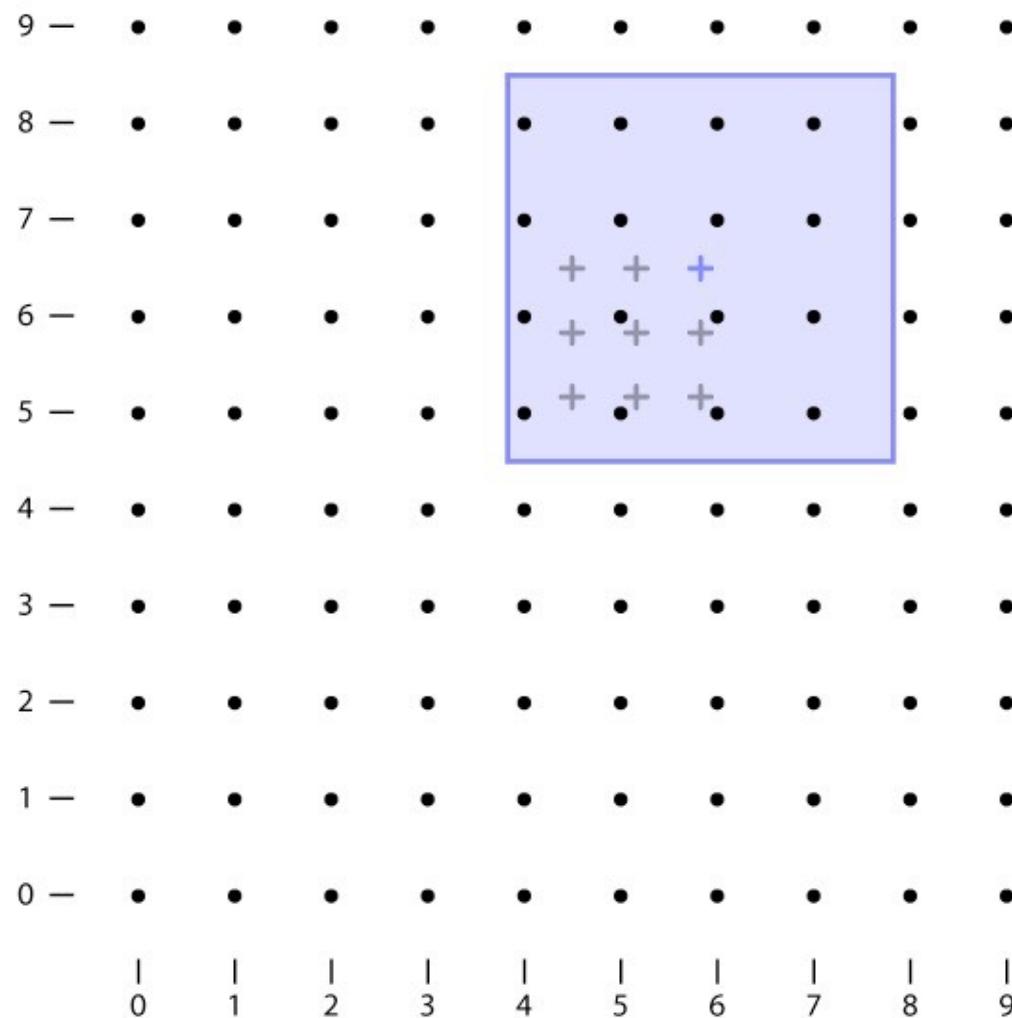
# Resampling example



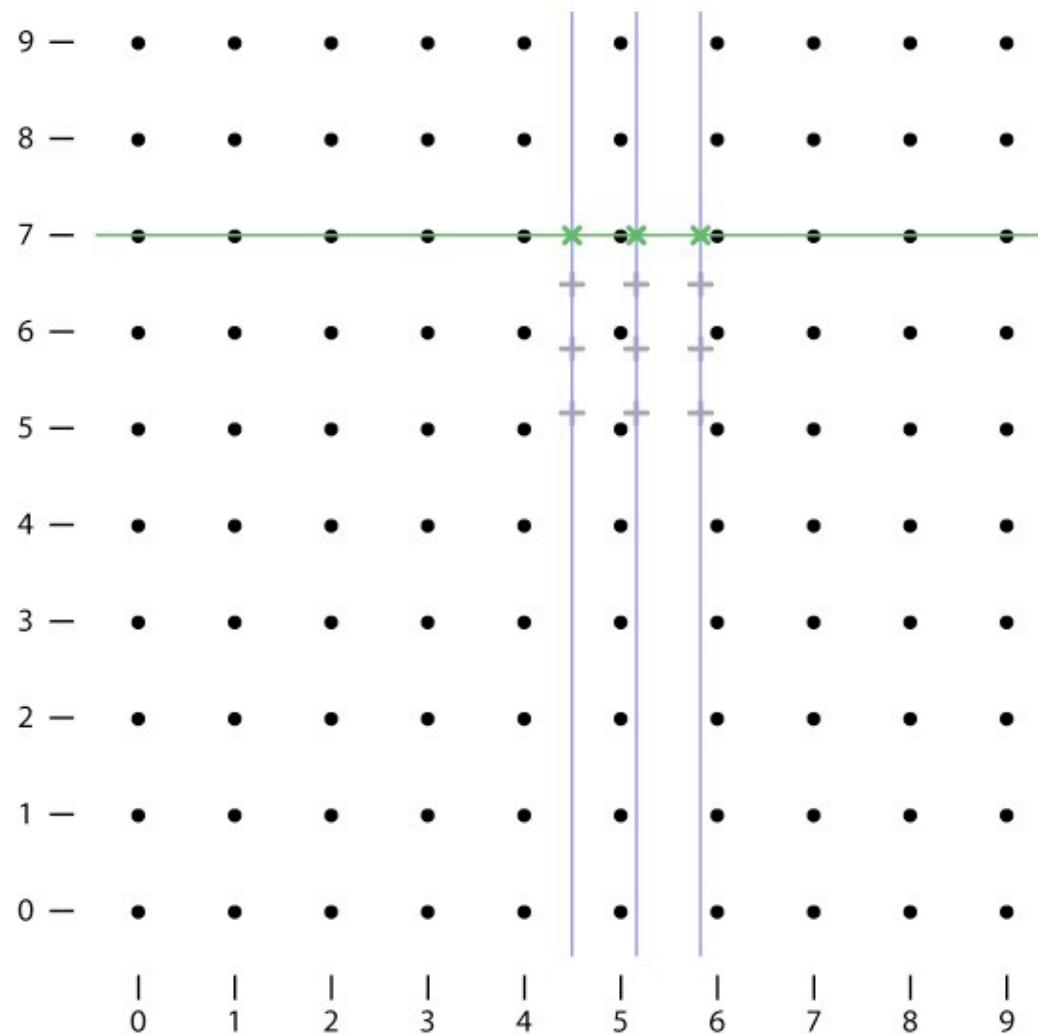
# Resampling example



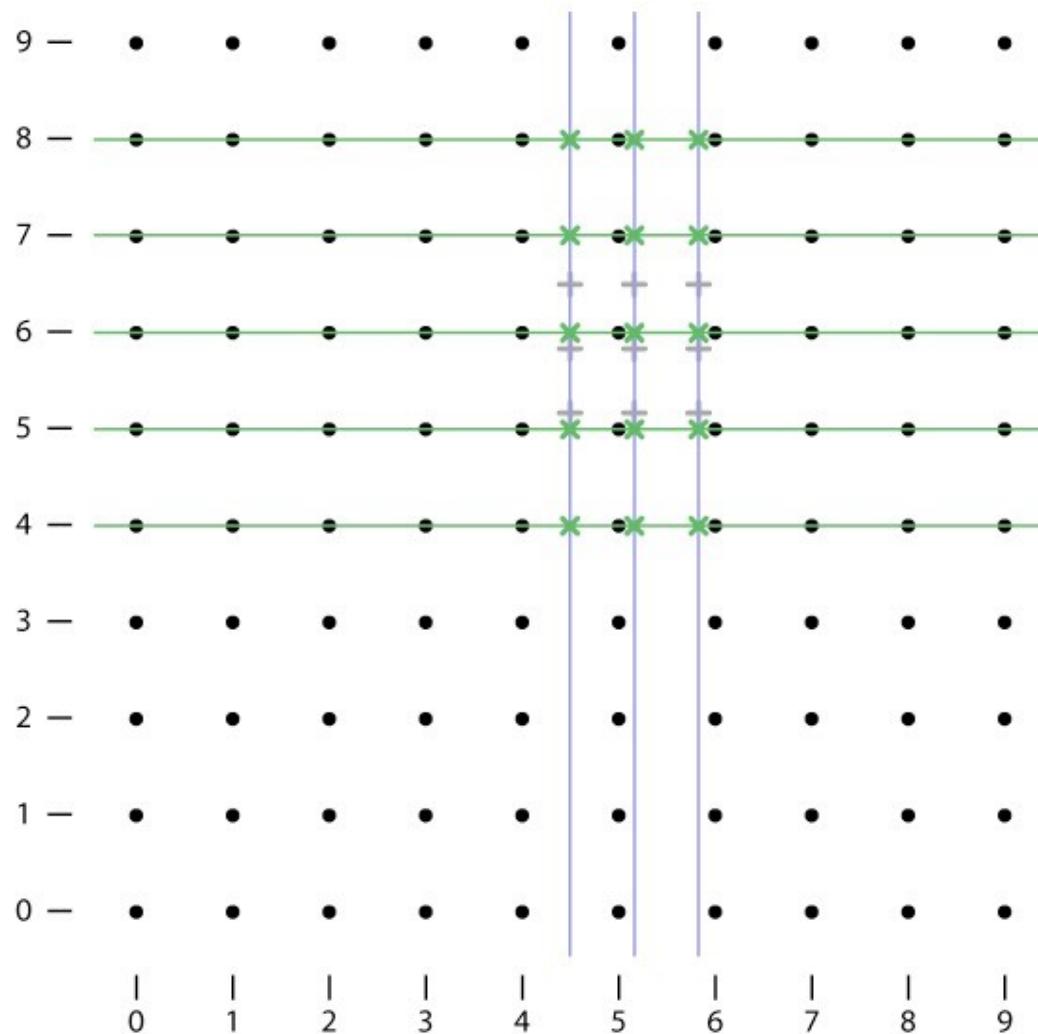
# Resampling example



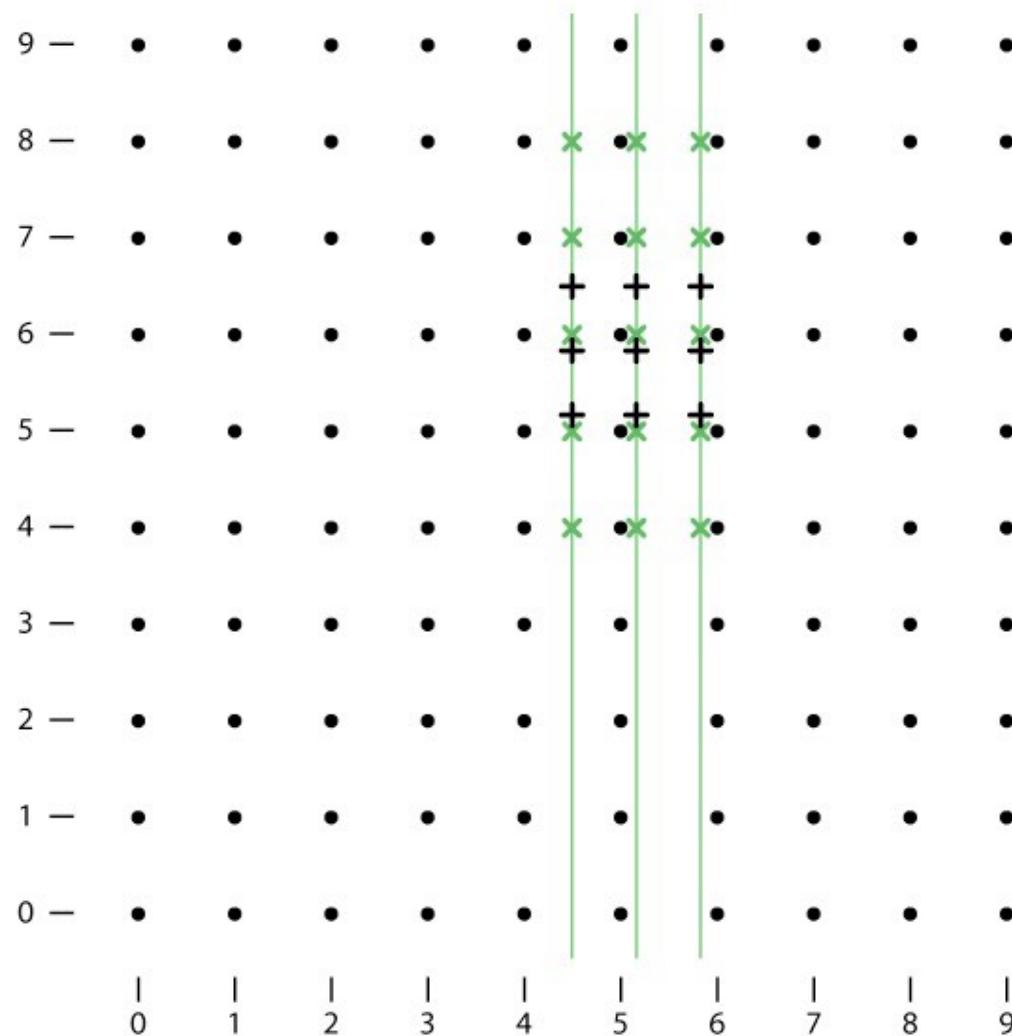
# Resampling example



# Resampling example

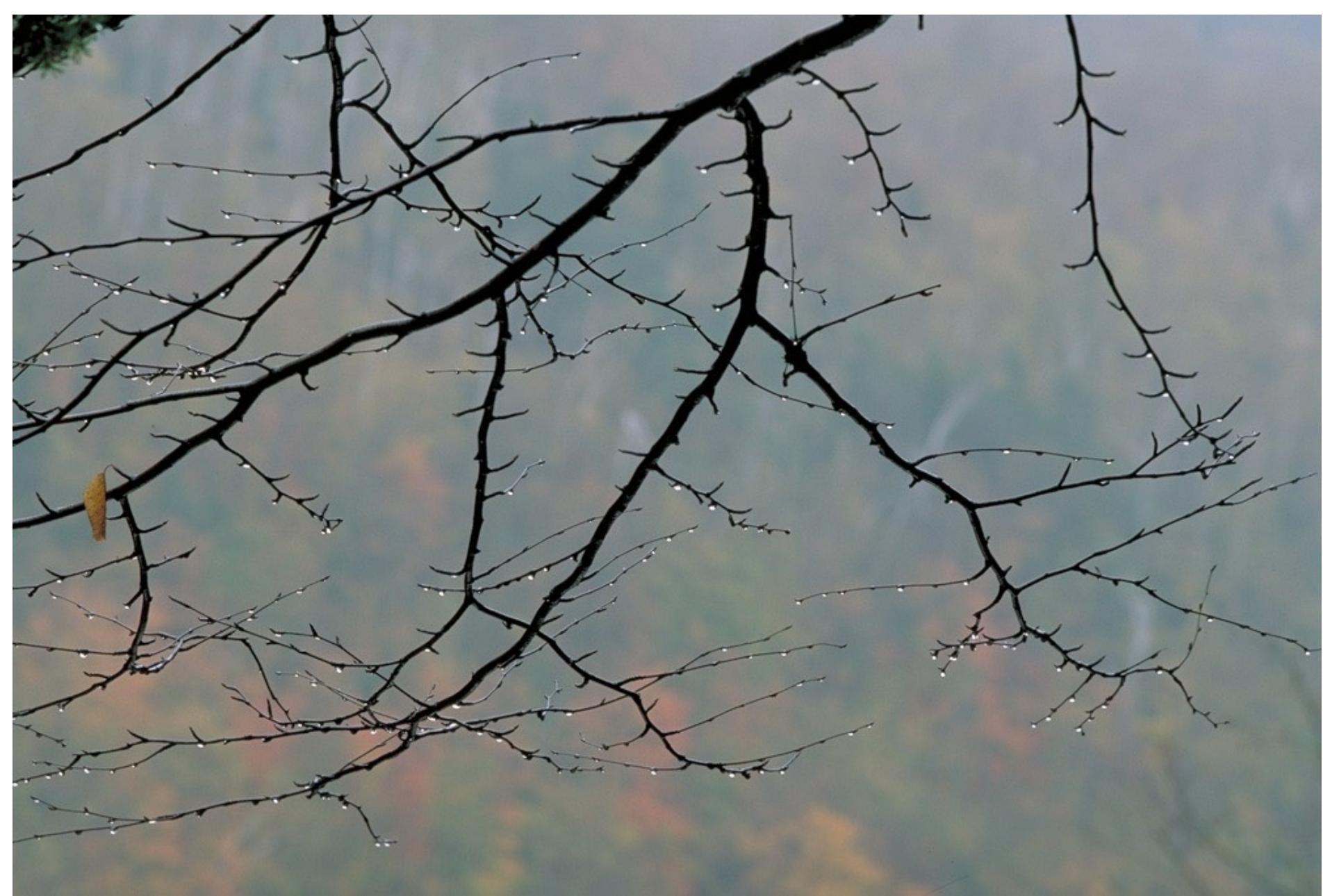


# Resampling example



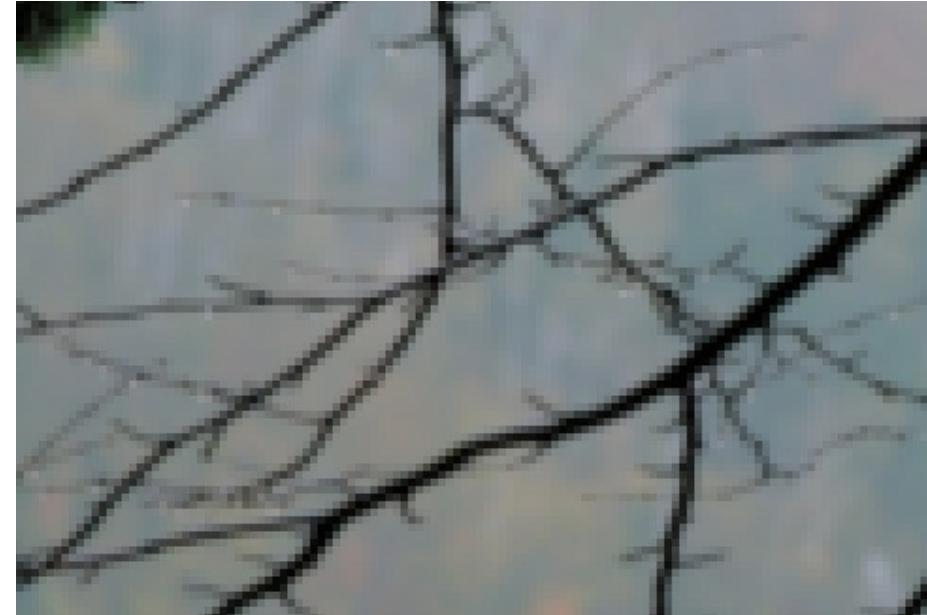
# Reducing and enlarging

- 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



1000 pixel width

[Philip Greenspun]



[Philip Greenspun]

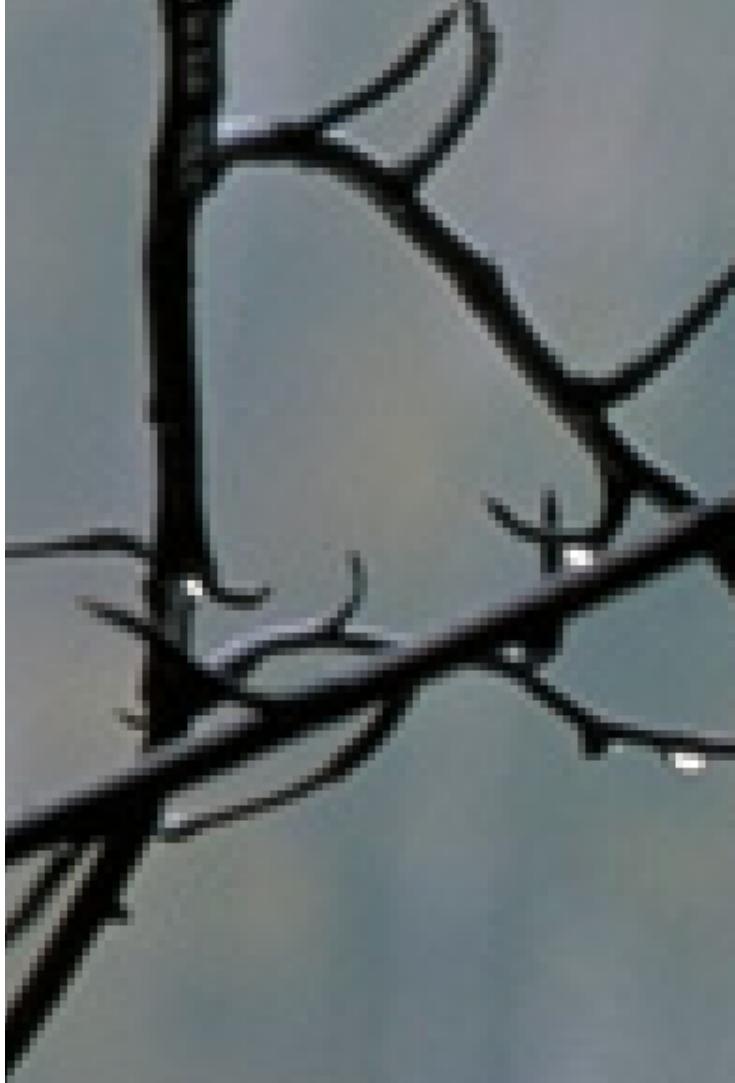


by dropping pixels

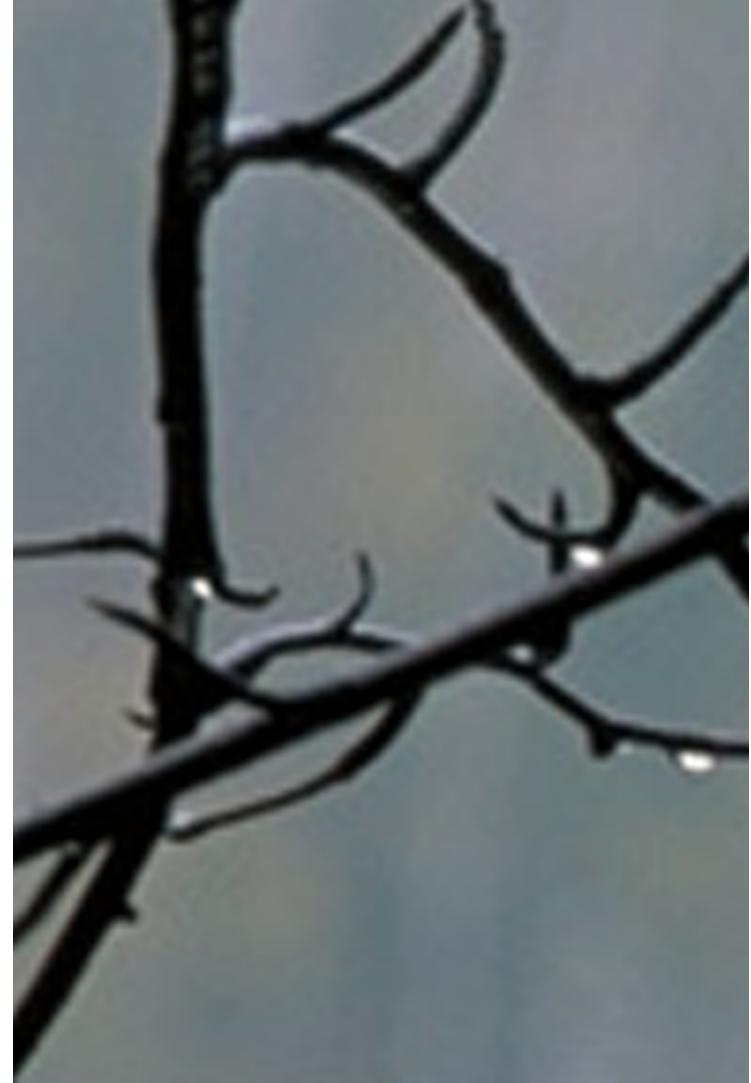


gaussian filter

250 pixel width



box reconstruction filter



bicubic reconstruction filter

4000 pixel width

[Philip Greenspun]

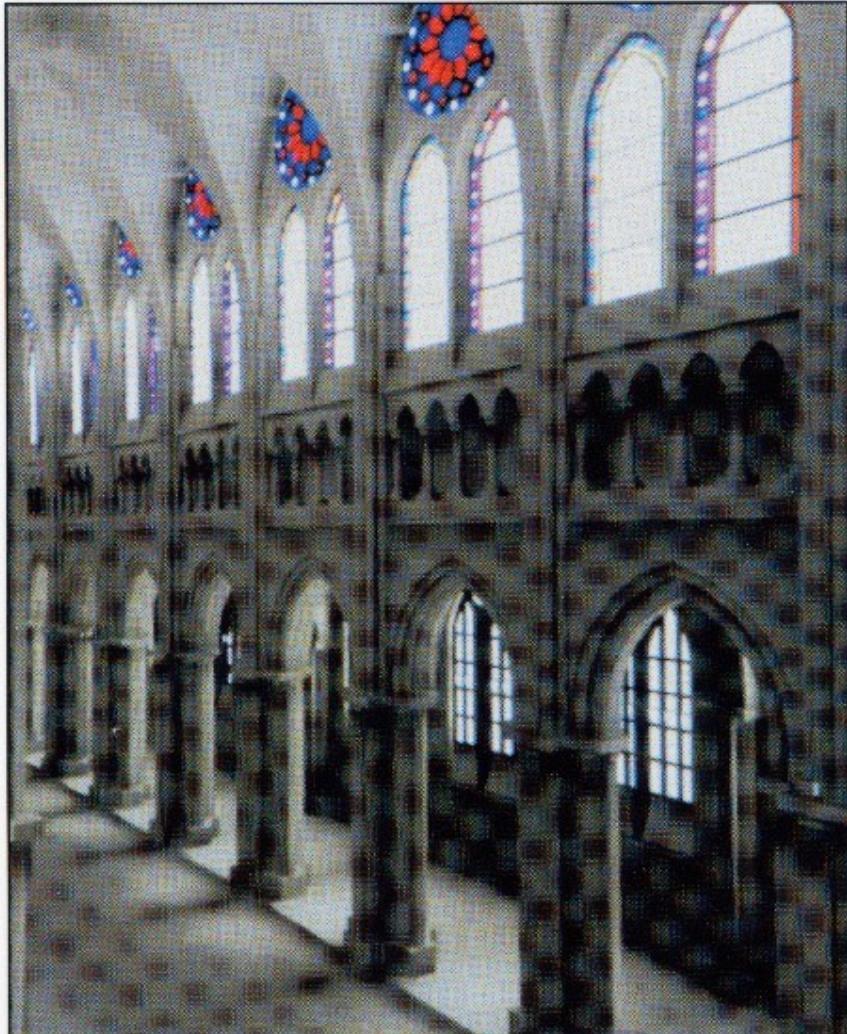
# Types of artifacts

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

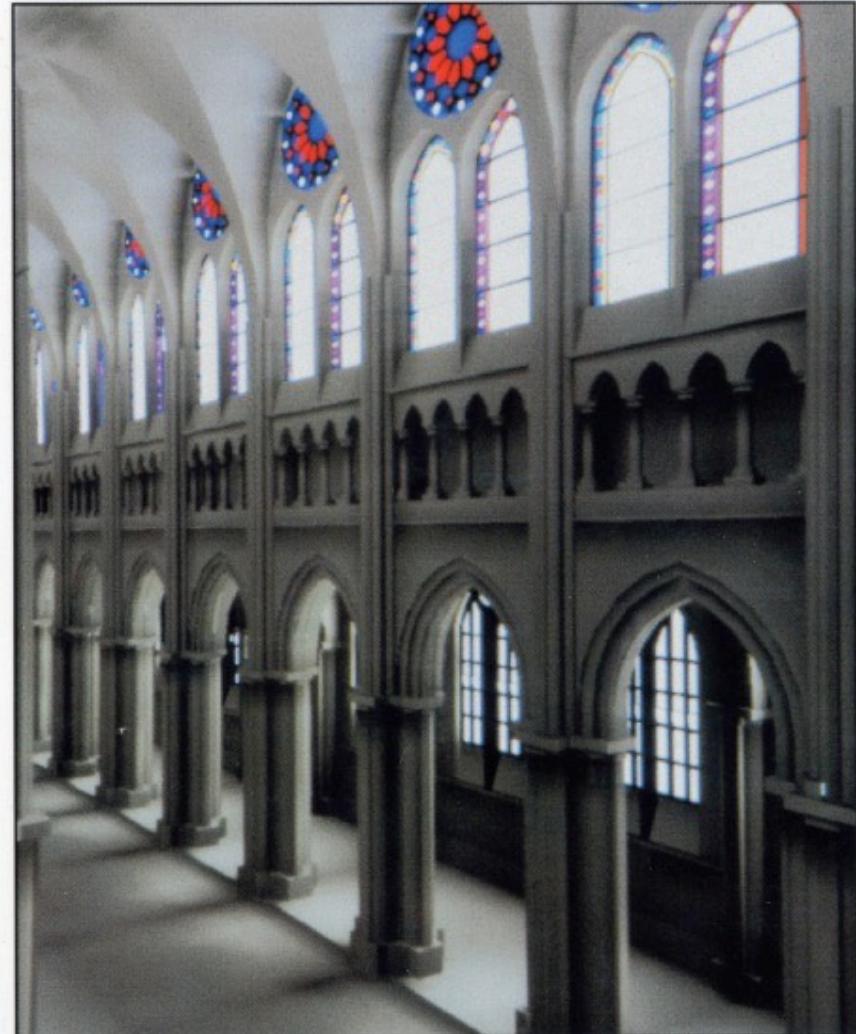


[Hearn & Baker cover]

600ppi scan of a color halftone image



by dropping pixels



gaussian filter

downsampling a high resolution scan

[Hearn & Baker cover]

# 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