CS559: Computer Graphics

Lecture 4: Image Filtering and Resampling

Li Zhang

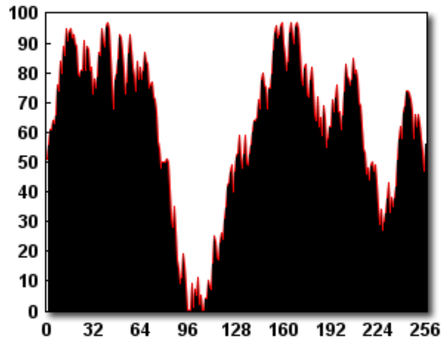
Spring 2010

Announcement

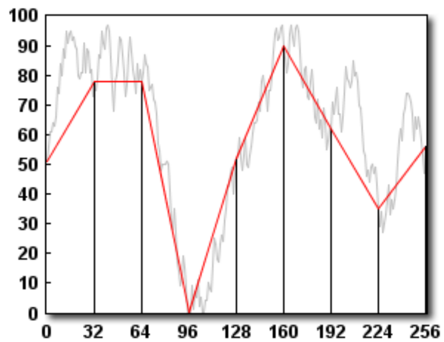
- Feb 3rd (this Wed) office hour moves to 4.30-5.30pm due to CS Department Faculty meeting 3.30-4.30.

Last time: Image Sampling and Filtering

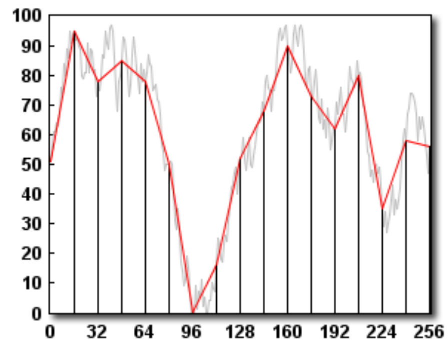
Continuous Function



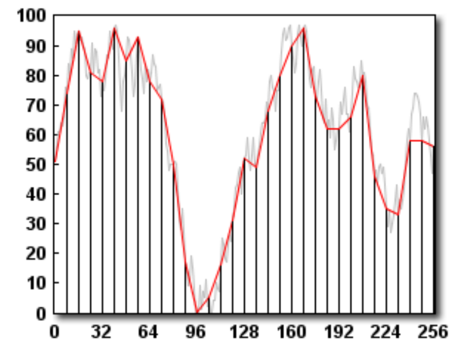
Discrete Samples



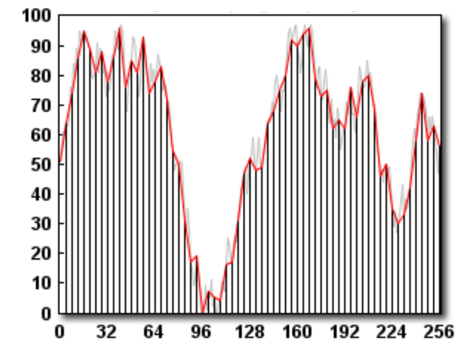
Sampling Period T = 32



Sampling Period T = 16



Sampling Period T = 8

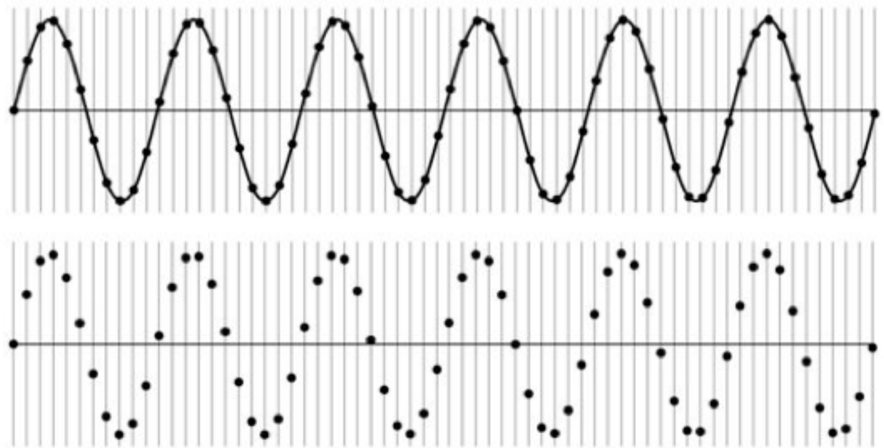


Sampling Period T = 4

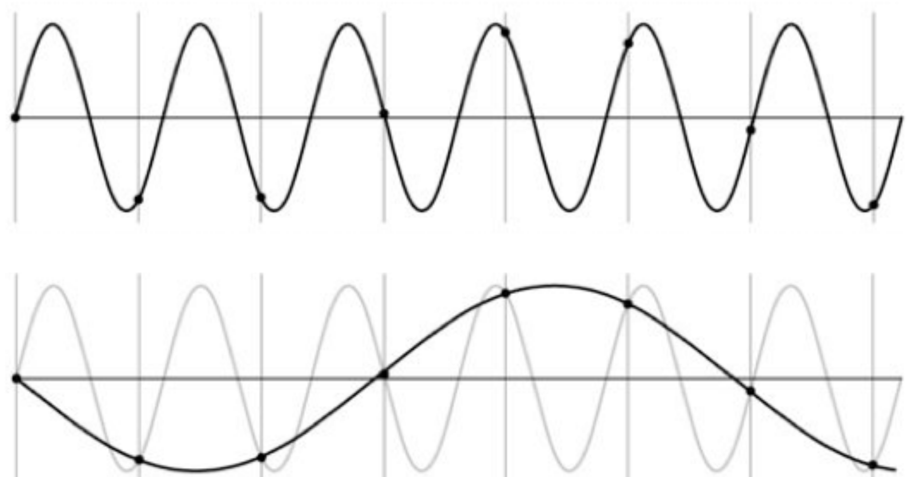
The denser the better, but at the expense of storage and processing power

Under-sampling

- Sampling a sine wave



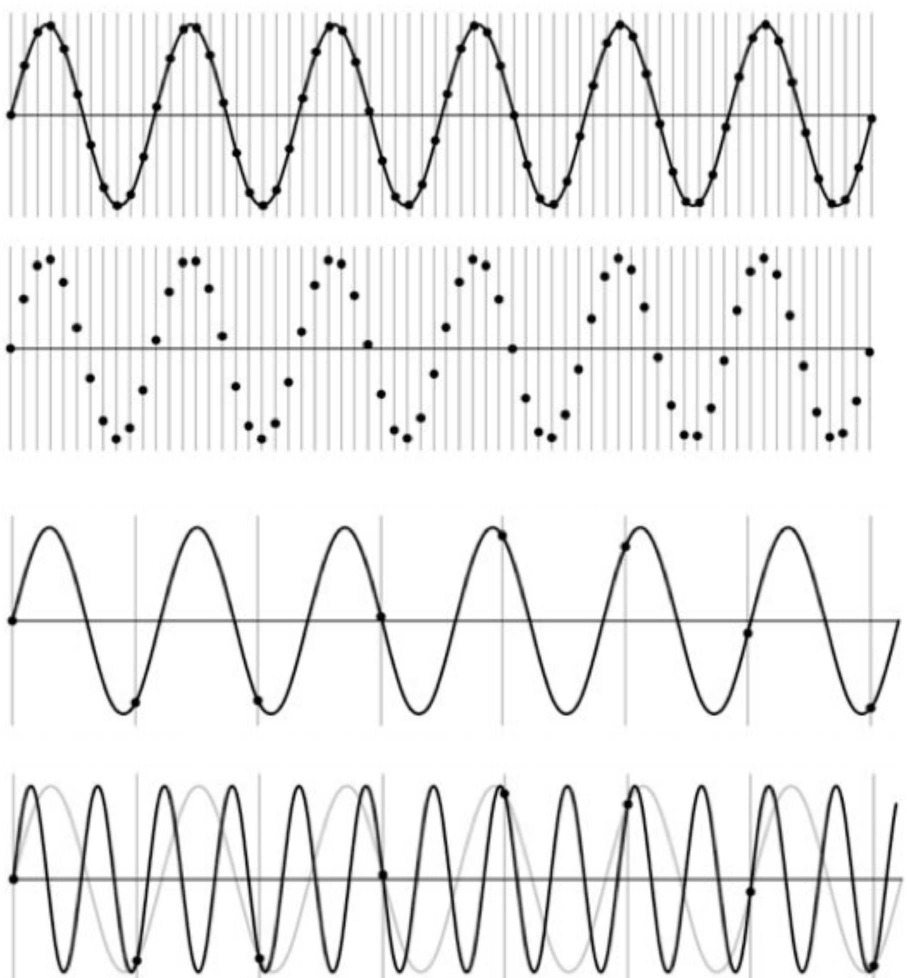
Some information loss



Ambiguous signal interpretation

Under-sampling

- Sampling a sine wave



Some information loss

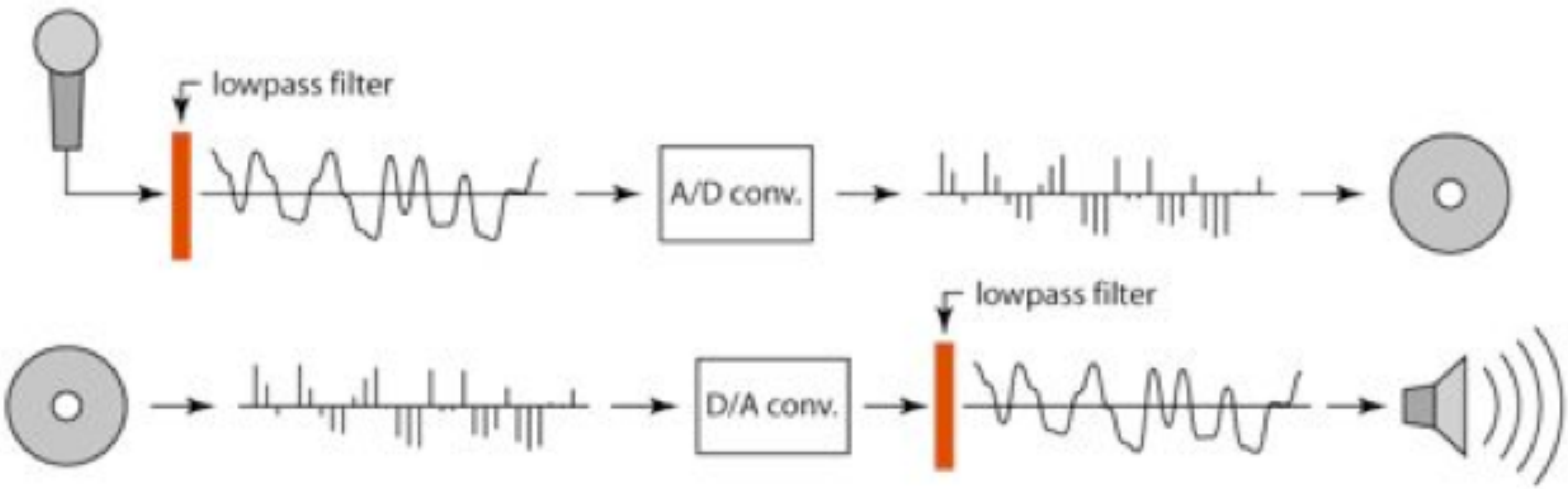
Ambiguous signal interpretation

Preventing aliasing

Introducing lowpass filters:

remove high frequency leaving only safe low frequencies

choose lowest frequency in reconstruction (disambiguate)



Discrete convolution

- Convolution: same idea but with *weighted* average

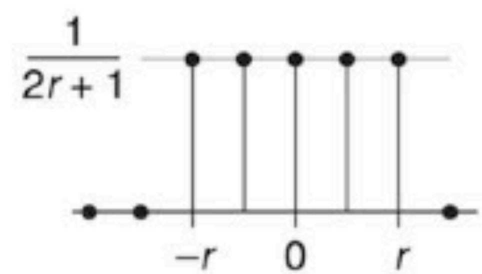
$$(a \star b)[i] = \sum_j a[j]b[i-j]$$

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 $a[j]$ 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



a box filter

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 = [..., 0, 0, 1, 0, 0, ...]

    $a \star e = a$
- Conceptually no distinction between filter and signal

Assuming zero padding outside the nonzero filter support

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

a

0	0	0
0	0.5	0
0	0	0.5

ɐ

0.5	0	0
0	0.5	0
0	0	0

62	79	23	119	120	105	4	0
10	10	9	62	12	78	34	0
10	58	197	46	46	0	0	48
176	135	5	188	191	68	0	49
2	1	1	29	26	37	0	77
0	89	144	147	187	102	62	208
255	252	0	166	123	62	0	31
166	63	127	17	1	0	99	30

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, …)

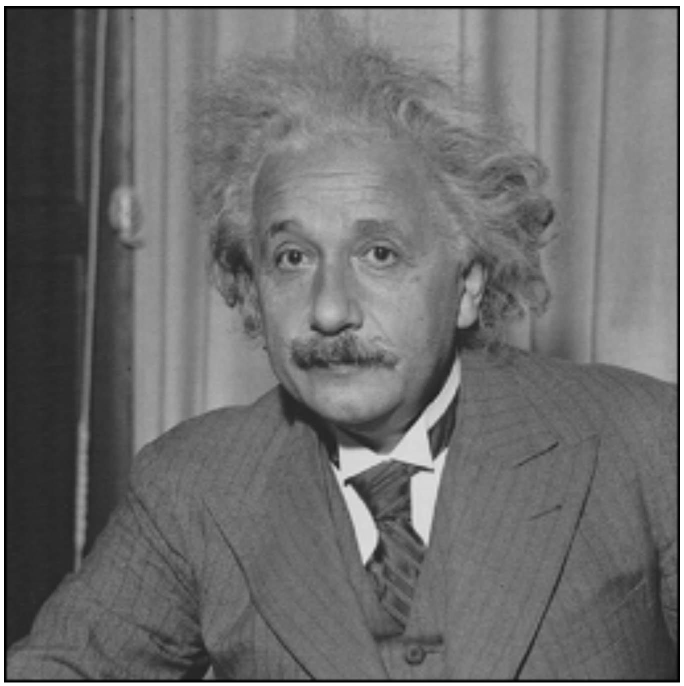
  sharpening (impulse minus blur)

- Usefulness of associativity

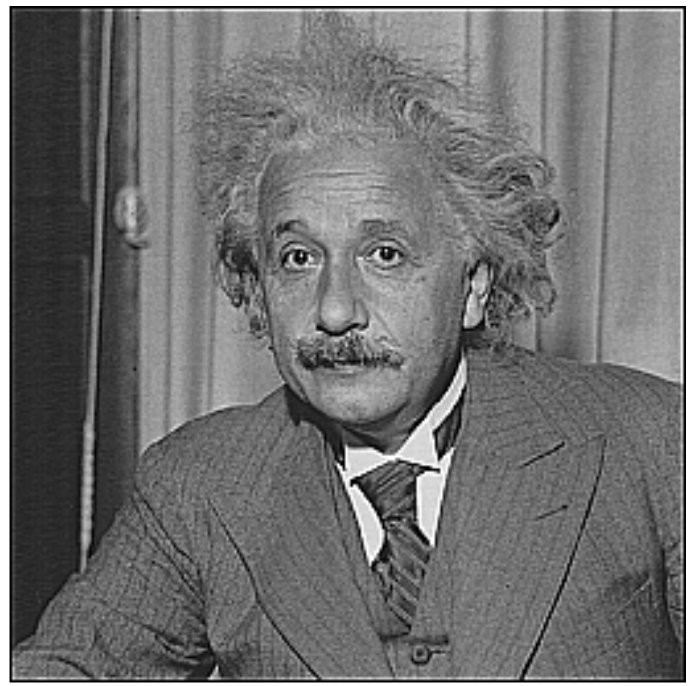
  often apply several filters one after another: $(((a * b_1) * b_2) * b_3)$

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

Sharpening by Filtering



before



after

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

1	4	6	4	1

1
4
6
4
1

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

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

first, convolve with this

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

Separable filtering

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

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

The filter can have rectangular shape as well.
For example 3x5.

second, convolve with this

first, convolve with this

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

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	1	0
0	2	0
0	1	0

The filter can have rectangular shape as well.
For example 3x5.

second, convolve with this

first, convolve with this

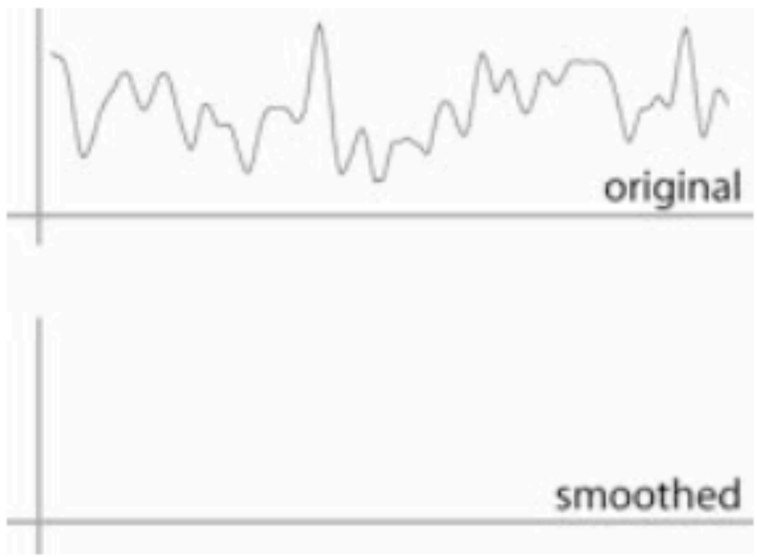
$$\sum_{i'} a_1[i'] \left(\sum_{j'} a_1[j'] b[i-i', j-j'] \right)$$

Today's topics

- Continuous Convolution
- Continuous-discrete convolution
- Resampling

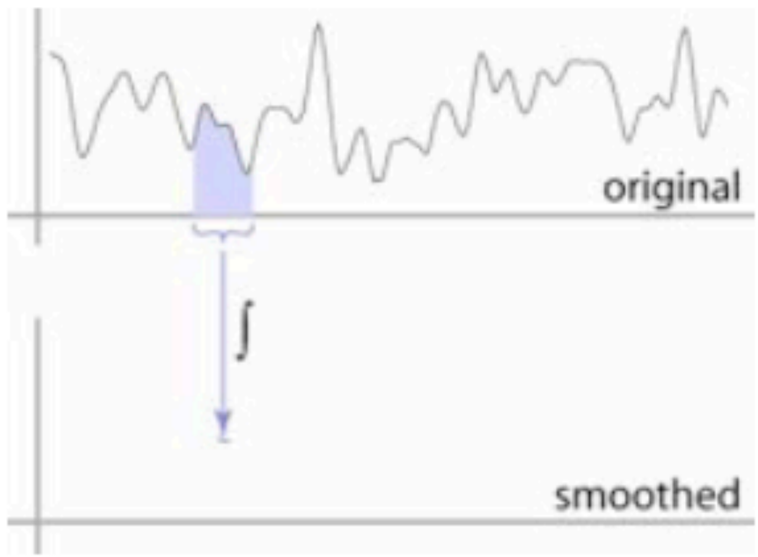
Continuous convolution: warm-up

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



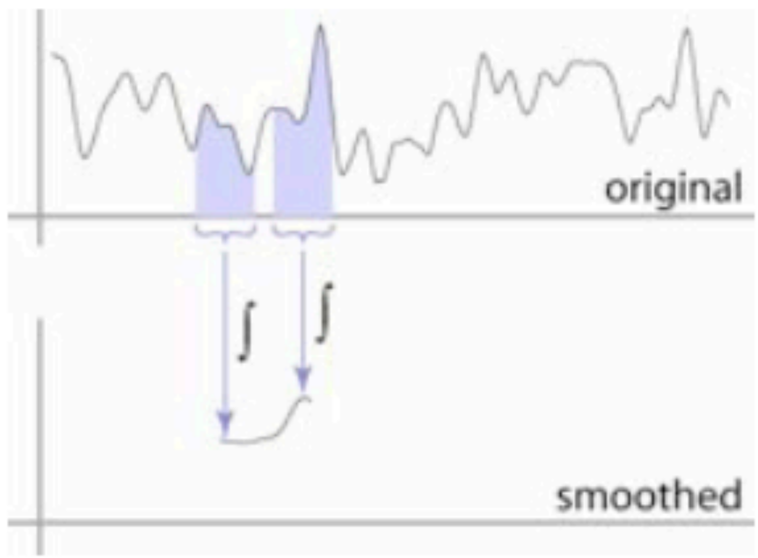
Continuous convolution: warm-up

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



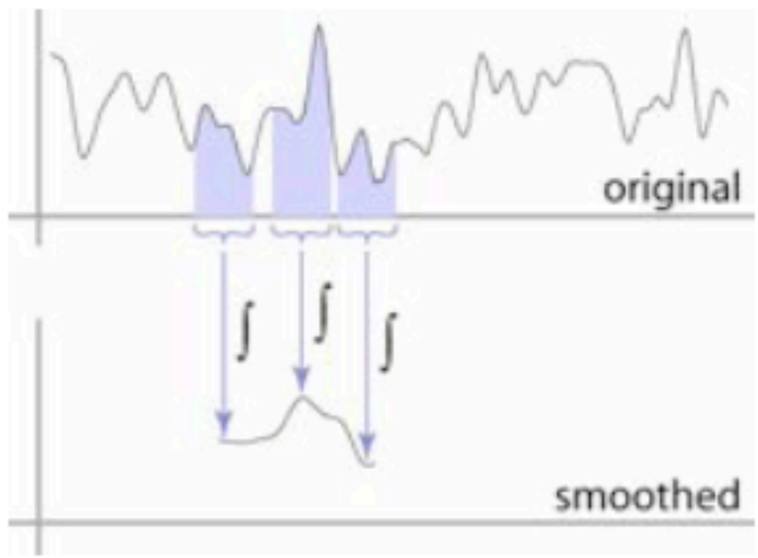
Continuous convolution: warm-up

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



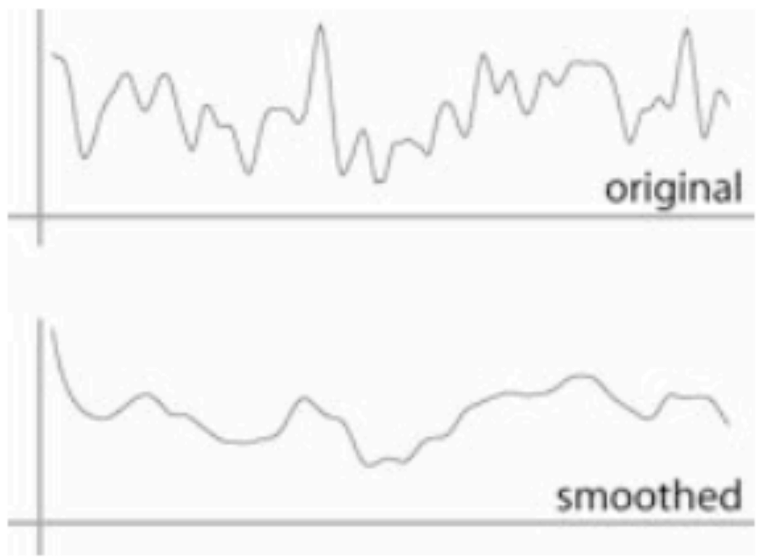
Continuous convolution: warm-up

- 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)

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)

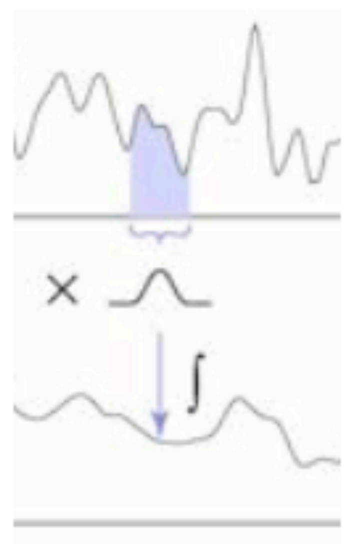
- Convolution just adds weights

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

weighting is now by a function

weighted integral is like weighted average

again bounds are set by support of *f(x)*



Let’s do a concrete example

Delta Function

- The counterpart of

0	0	0
0	1	0
0	0	0

for continuous convolution

Delta Function

- The counterpart of

0	0	0
0	1	0
0	0	0

for continuous convolution

- $(\delta \bigstar f)(x)=f(x)$

One more convolution

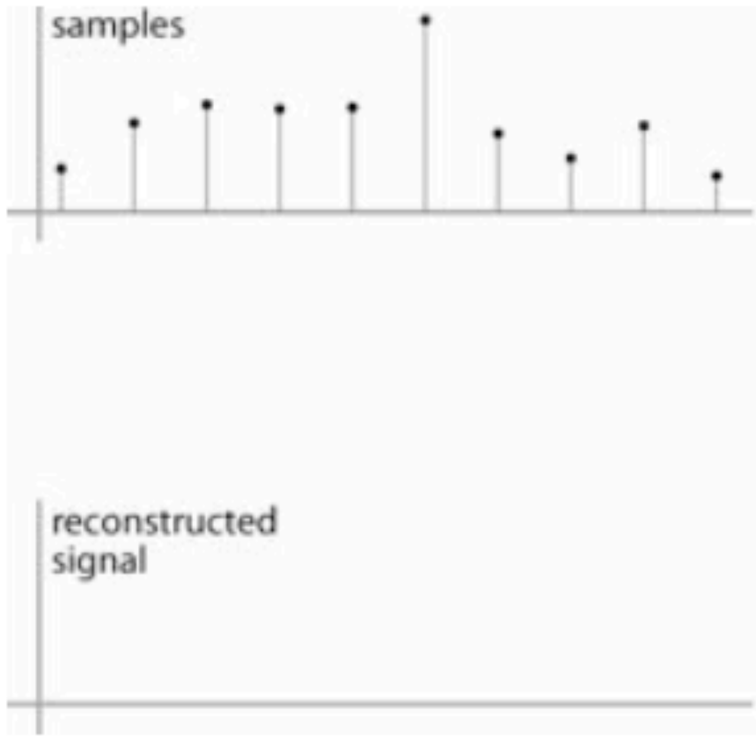
- Continuous–discrete convolution

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

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

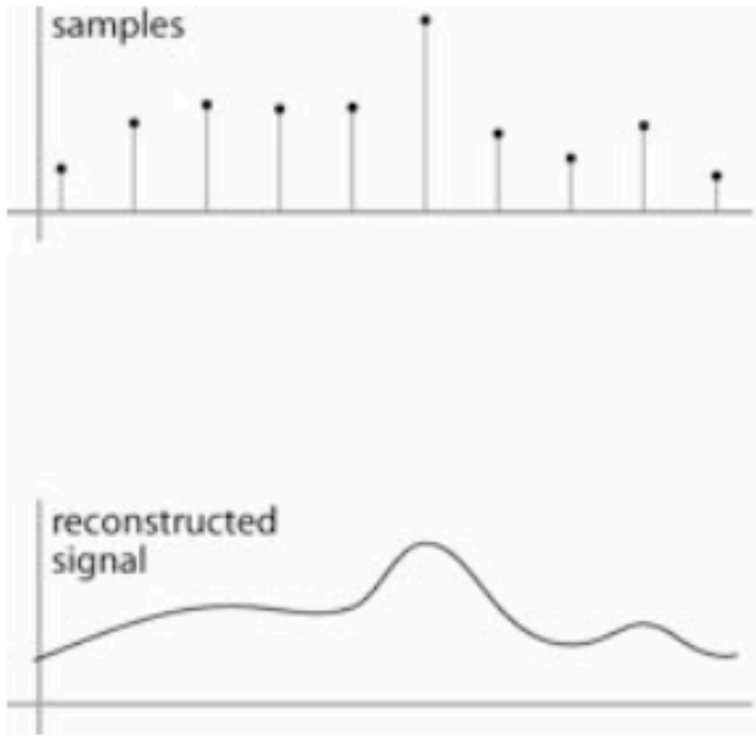
used for reconstruction and resampling

Continuous-discrete convolution



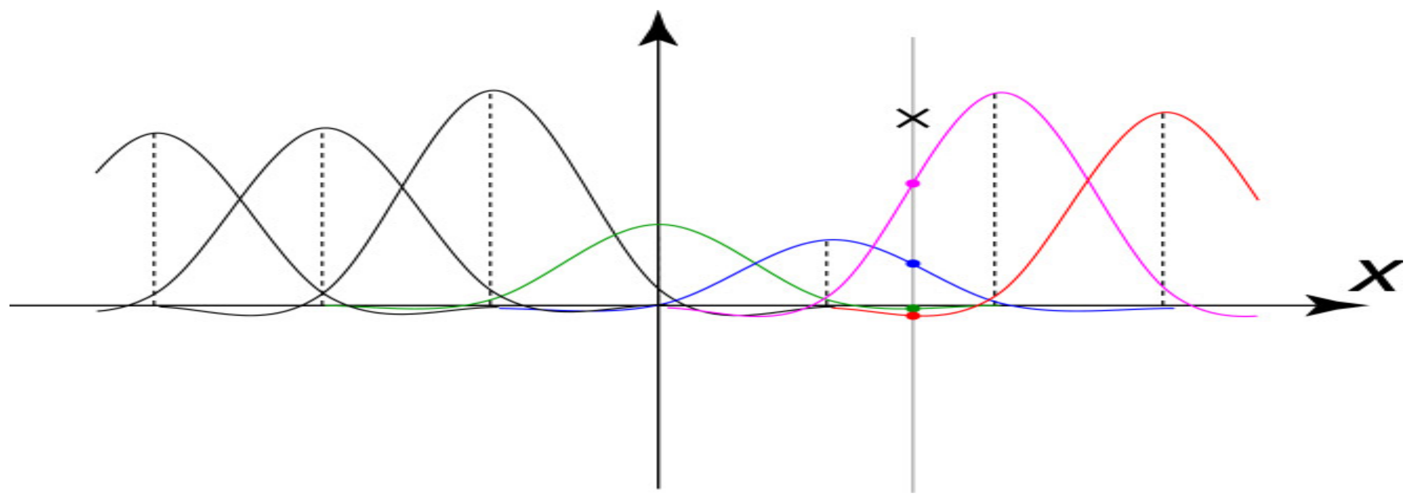
1. putting the flipped reconstruction filter at the desired location
2. evaluating at the original sample positions
3. taking products with the sample values themselves
4. summing it up

Continuous-discrete convolution



1. putting the flipped reconstruction filter at the desired location
2. evaluating at the original sample positions
3. taking products with the sample values themselves
4. summing it up

Another view on continuous-discrete convolution



Reconstruction (discrete-continuous convolution) as a sum of shifted copies of the filter

Same view also holds for discrete convolution

Resampling

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

Resampling

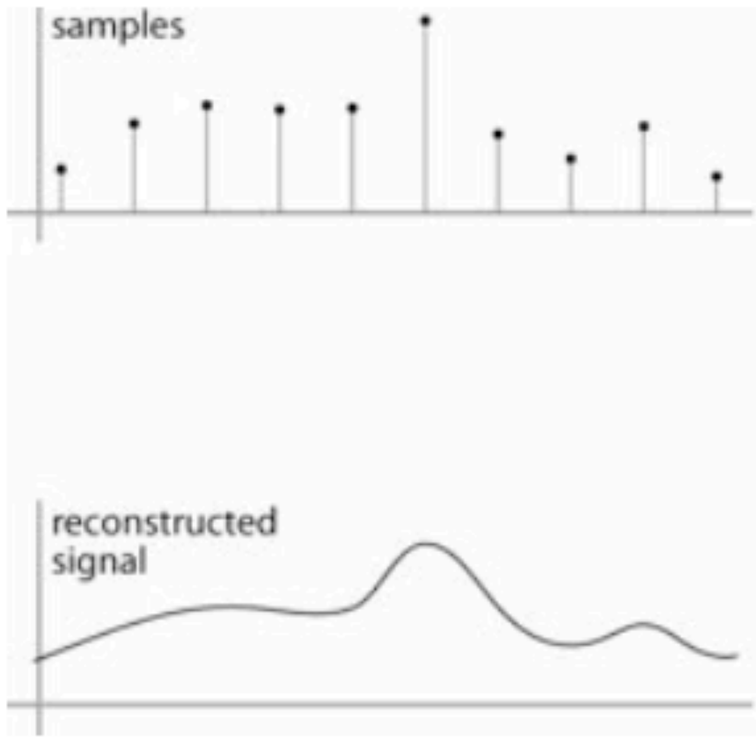
- Changing the sample rate
  - in images, this is enlarging and reducing
- Creating more samples:
  - increasing the sample rate
  - "upsampling"
  - "enlarging"

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"

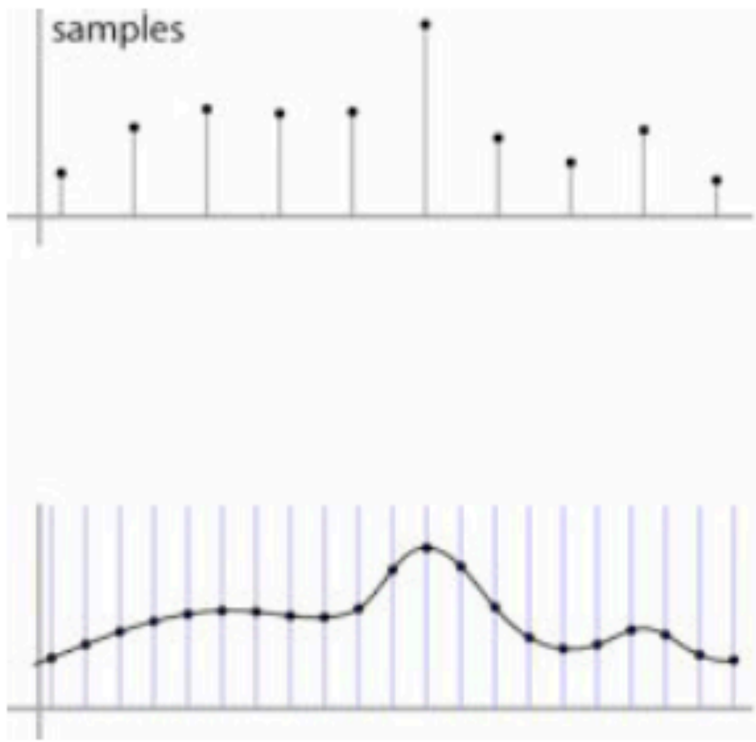
Resampling

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



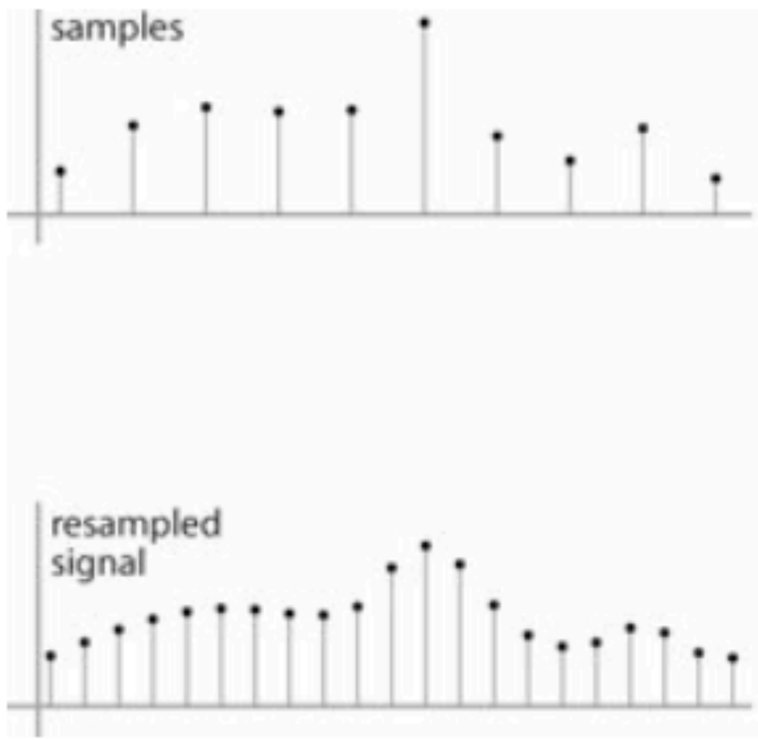
Resampling

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



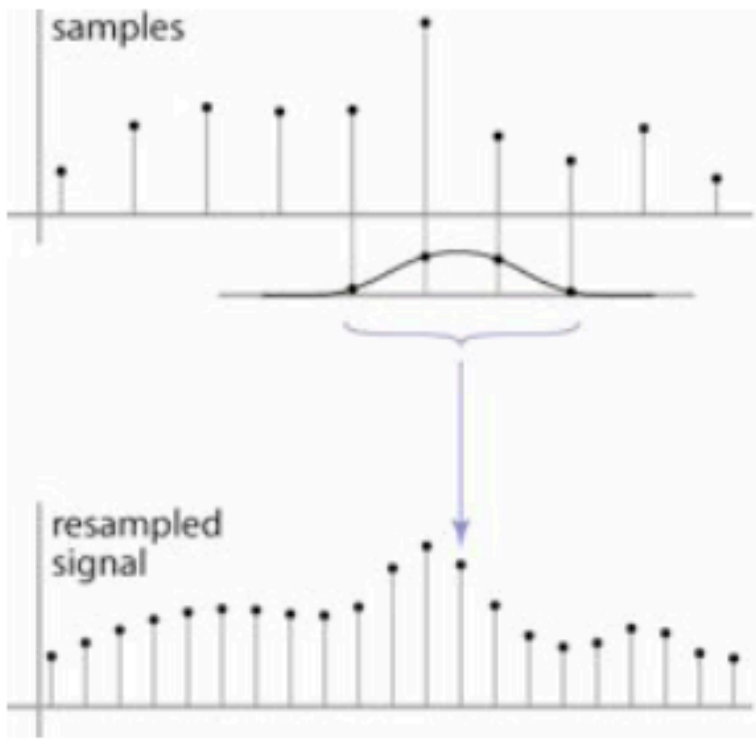
Resampling

- 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



And in pseudocode…

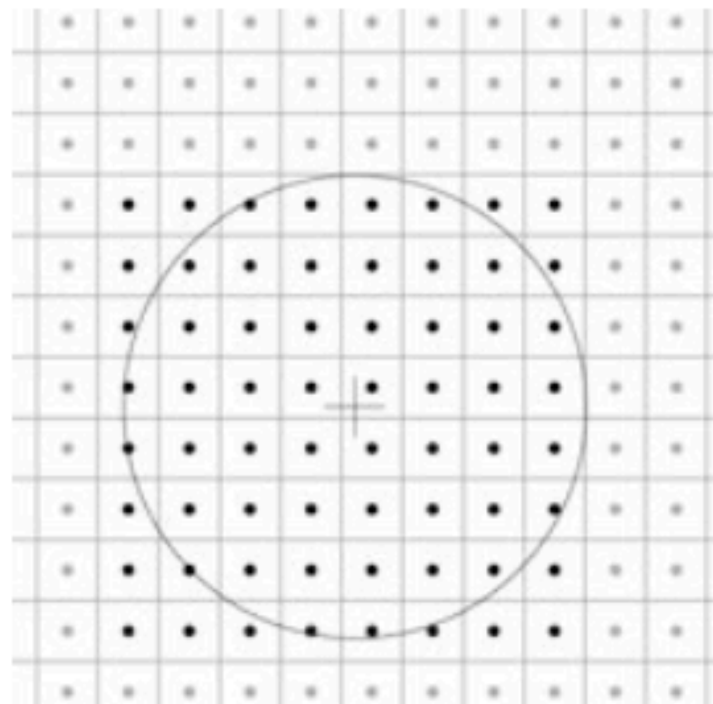
function reconstruct(sequence a, filter f, real x)
$s = 0$
$r = f$.radius
for $i = \lceil x - r \rceil$ to $\lfloor x + r \rfloor$ **do**
  $s = s + a[i]f(x - i)$
return s

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



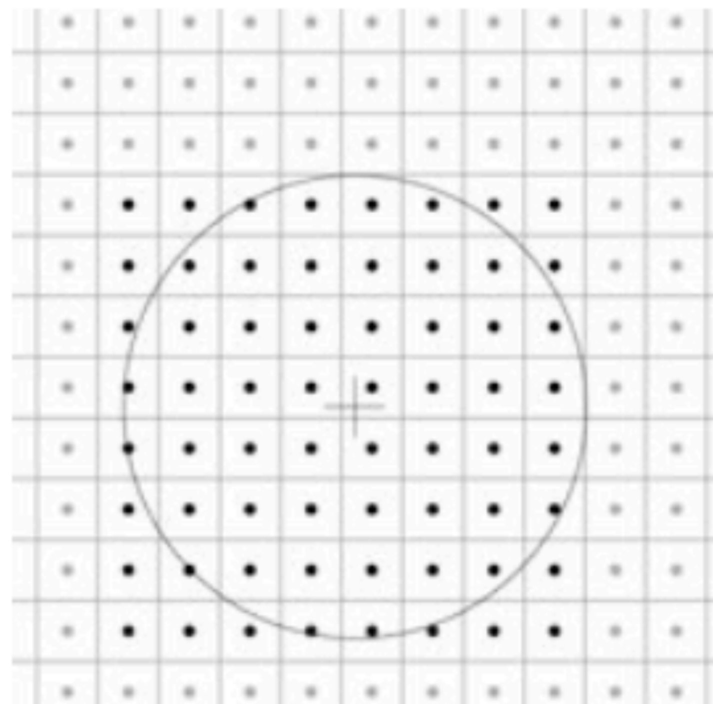
Cont.–disc. convolution in 2D

- 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



Cont.–disc. convolution in 2D

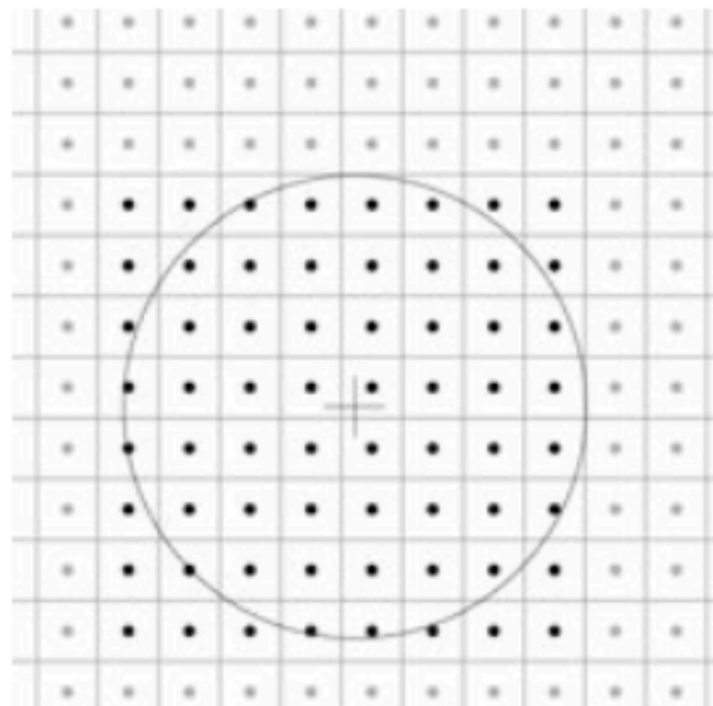
- same convolution—just two variables now

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

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

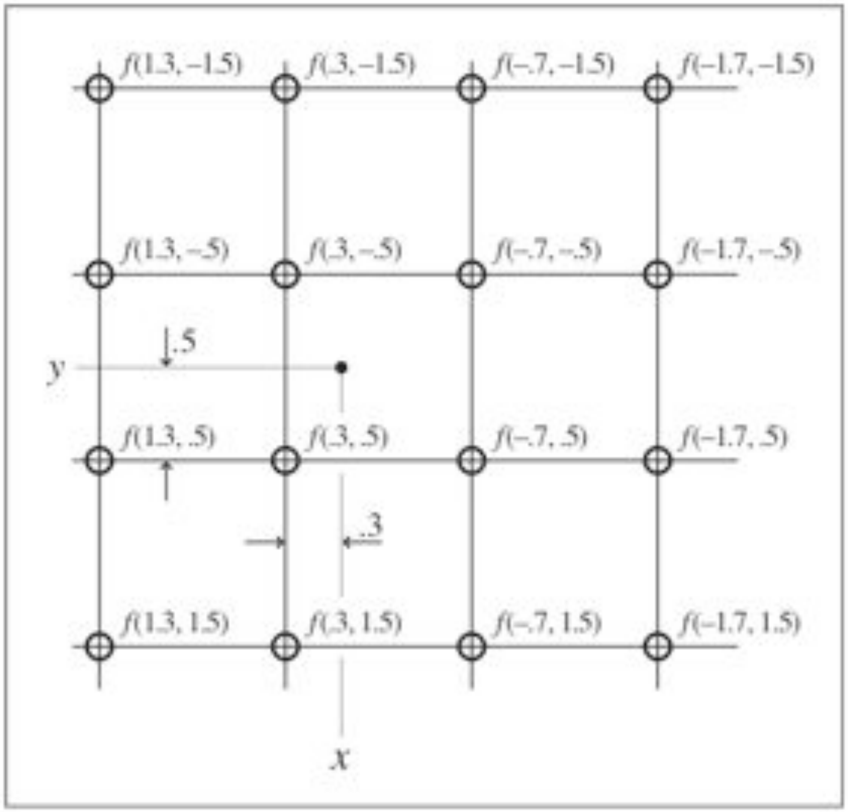
$$(a \star f)(x, y) = \sum_{i,j} a[i, j] f(x - i, y - j)$$

An Example:

Cont.–disc. convolution in 2D

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

An Example:



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:

$$f_2(x, y) = f_1(x) f_1(y)$$

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

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 src. and the other dimension of dest.

[Philip Greenspun]

two-stage resampling using a separable filter

A gallery of filters

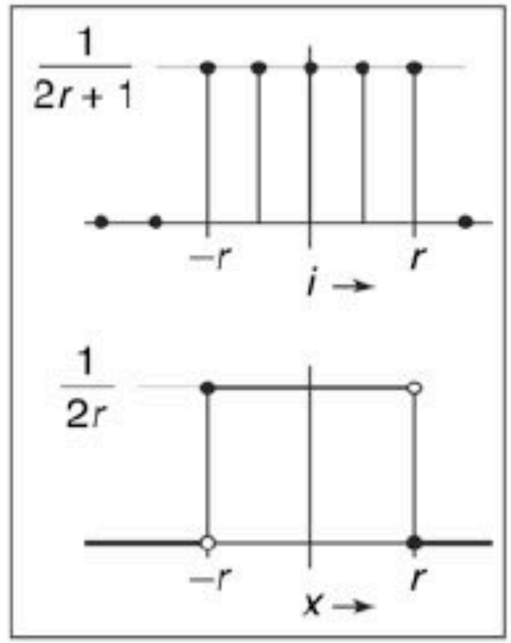
- Box filter
  - Simple and cheap
- Tent filter
  - Linear interpolation
- Gaussian filter
  - Very smooth antialiasing filter
- B-spline cubic
  - Very smooth
- Catmull-rom cubic
  - Interpolating
- Mitchell-Netravali cubic
  - Good for image upsampling

Let’s take a break

Box filter

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

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



Discontinuous Reconstruction

How to use box filter

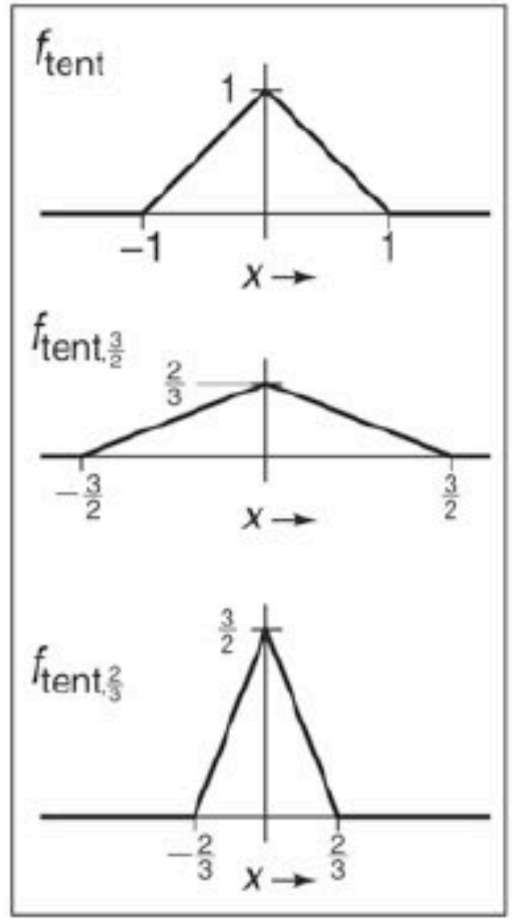
- Method 1

- Method 2

Tent filter

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

$$f_{\text{tent},r}(x) = \frac{f_{\text{tent}}(x/r)}{r}.$$



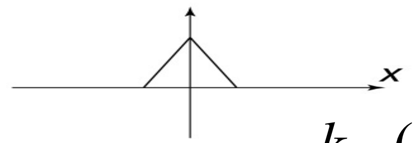
How to use tent filter

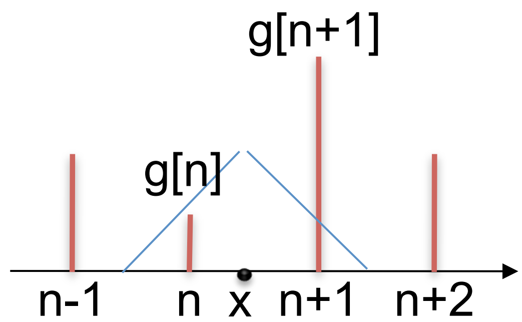
- Method 1

- Method 2

Reconstruction using 1D tent filter

1D example:





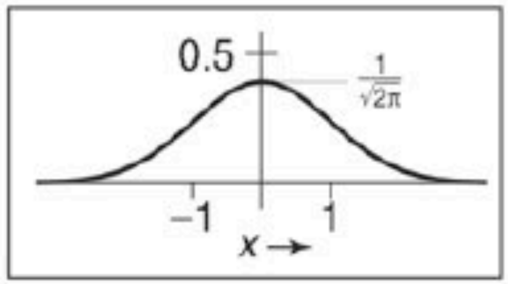
$$k_{1D}(x) = \begin{cases} 1 - |x| & |x| < 1 \\ 0 & \text{otherwise} \end{cases}$$

$$\Delta x = x - n$$

$$f(x) = g[n] \cdot (1 - \Delta x) + g[n+1] \cdot \Delta x$$

Tent filter reconstruction: Zero-order continuity
Use only one multiplication?

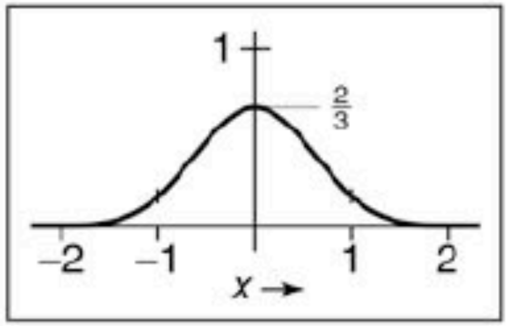
Gaussian filter



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

Infinitely smooth, negligible beyond [-3,3]

B-Spline cubic



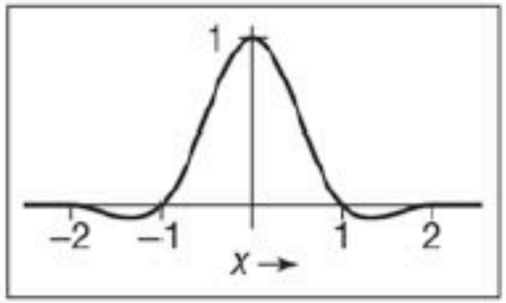
$$f_B(x) = \frac{1}{6} \begin{cases} -3(1-|x|)^3 + 3(1-|x|)^2 + 3(1-|x|) + 1 & -1 \leq x \leq 1, \\ (2-|x|)^3 & 1 \leq |x| \leq 2, \\ 0 & \text{otherwise.} \end{cases}$$

C2 Smoothness

Can be obtained by convolving a box filter four times

What's the problem to use it as a reconstruction filter?

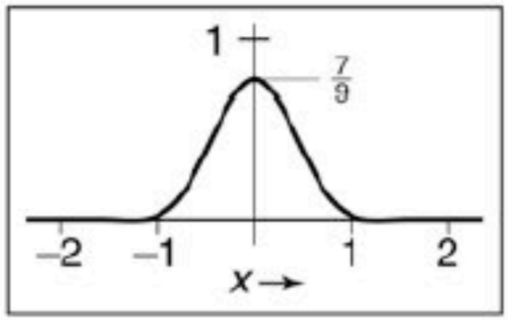
Catmull-Rom cubic



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

C1 Smoothness
It interpolates samples: “connecting the dots”

Michell-Netravali cubic



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

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

All-around best choice [Mitchell & Netravali 1988]

Effects of reconstruction filters

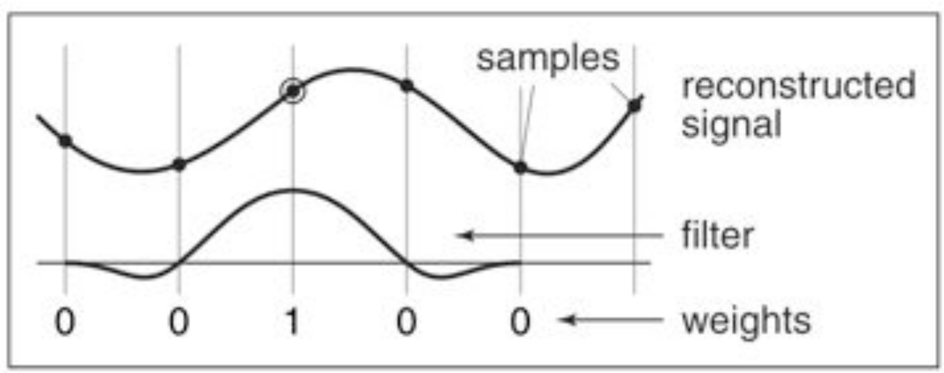
- For some filters, the reconstruction process winds up implementing a simple algorithm
- Box filter (radius 0.5): nearest neighbor sampling
  - box always catches exactly one input point
  - it is the input point nearest the output point
  - so output[*i*, *j*] = input[round(*x*(*i*)), round(*y*(*j*))]
    - *x*(*i*) computes the position of the output coordinate *i* on the input grid

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] = input[round($x(i)$), 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 Kernels

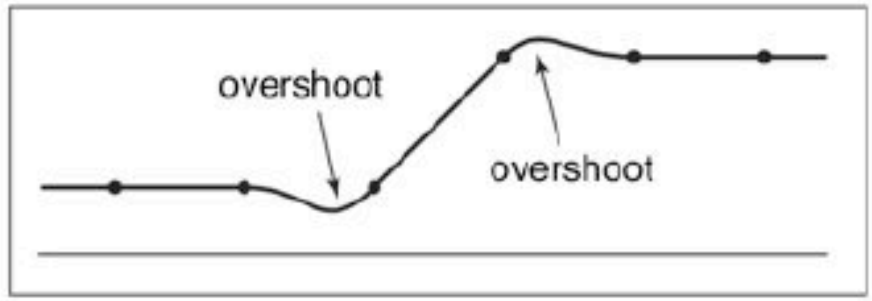
- Filter, Impulse Response, or kernel function, same concept but different names
- Degrees of continuity
- Interpolating or no
- Ringing or overshooting



Interpolating filter for reconstruction

Ringing, overshoot, ripples

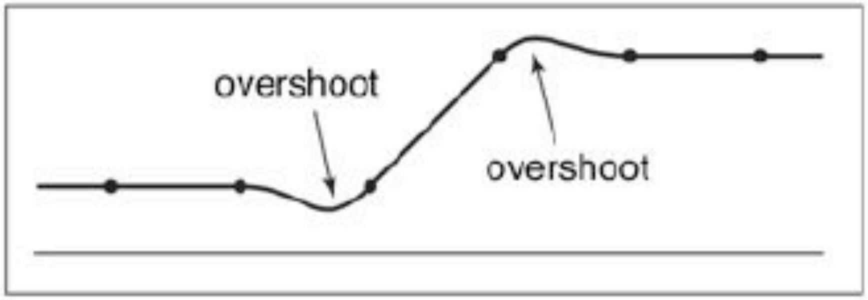
- Overshoot
  - caused by
    - negative filter values

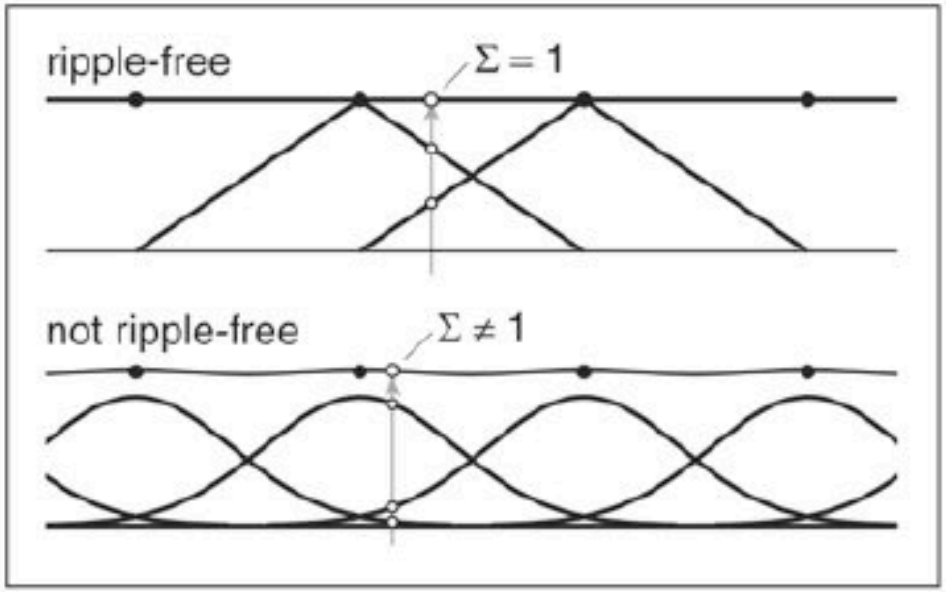


Ringing, overshoot, ripples

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

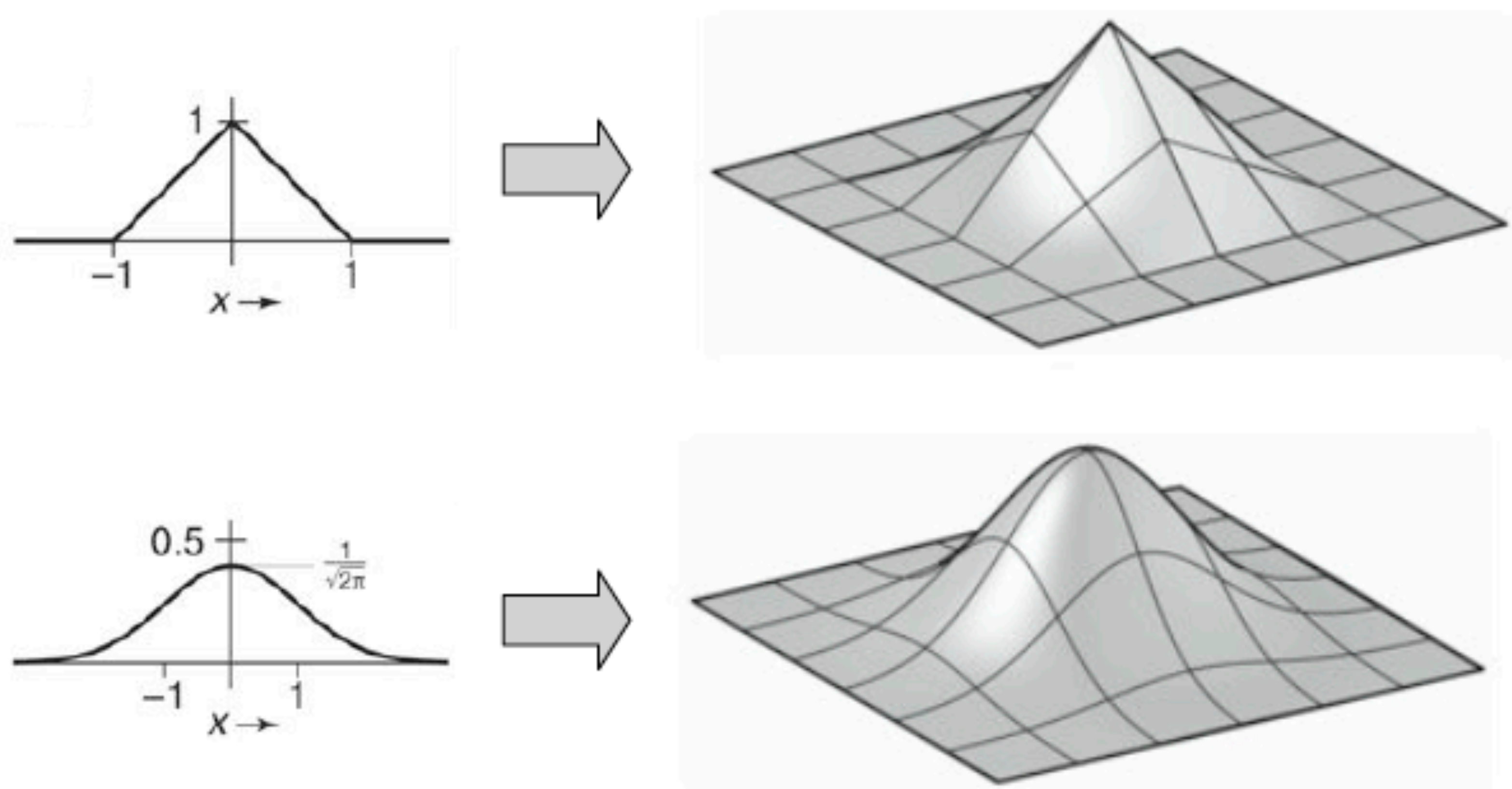
$$\sum_i f(x+i) = 1 \quad \text{for all } x.$$





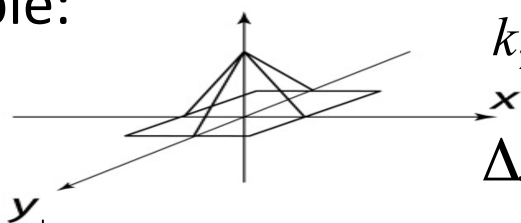
Constructing 2D filters

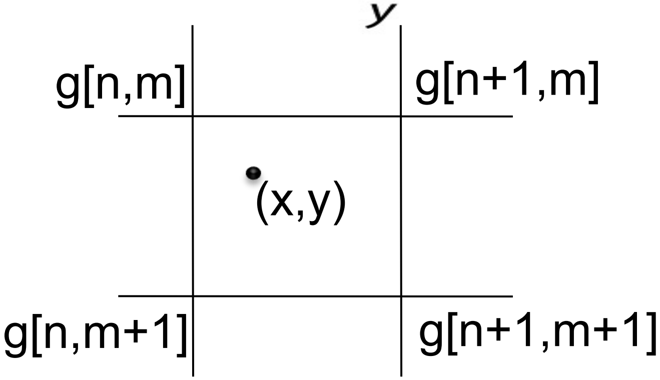
- Separable filters (most common approach)



Reconstruction filter Examples in 2D

2D example:





$$k_{2D}(x,y) = \begin{cases} (1-|x|)(1-|y|) & |x|<1, |y|<1 \\ 0 & \text{otherwise} \end{cases}$$

$$\Delta x = x - n, \Delta y = y - m$$

$$\begin{aligned} f(x,y) = & \; g[n,m]\cdot(1-\Delta x)\cdot(1-\Delta y) \\ & + g[n+1,m]\cdot\Delta x\cdot(1-\Delta y) \\ & + g[n,m+1]\cdot(1-\Delta x)\cdot\Delta y \\ & + g[n+1,m+1]\cdot\Delta x\cdot\Delta y \end{aligned}$$

How to simplify the calculation?

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

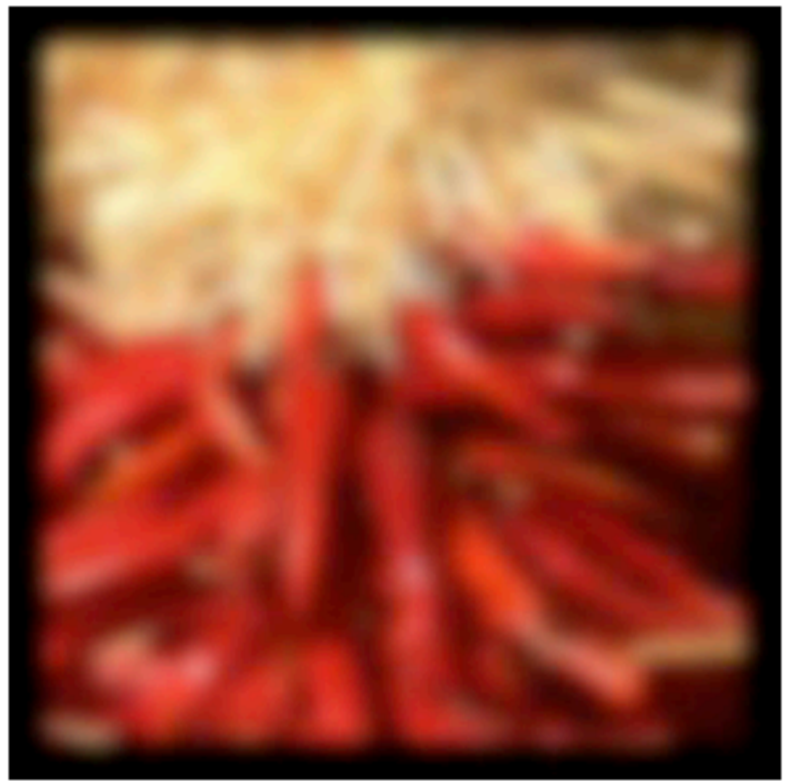
  methods:

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

[Philip Greenspun]

Yucky details

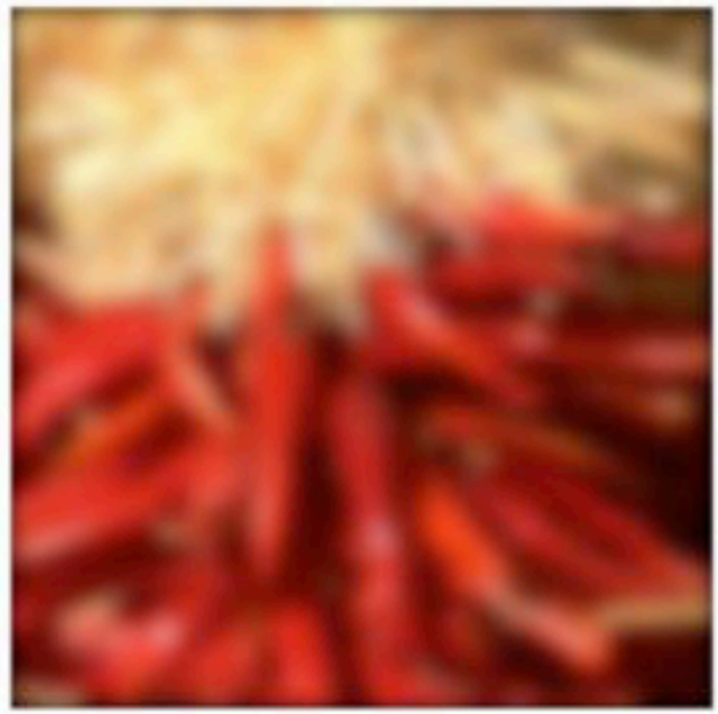
- What about near the edge?
  - the filter window falls off the edge of the image
  - need to extrapolate
  - methods:
    - clip filter (black)
    - wrap around
    - copy edge
    - reflect across edge
    - vary filter near edge



[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]

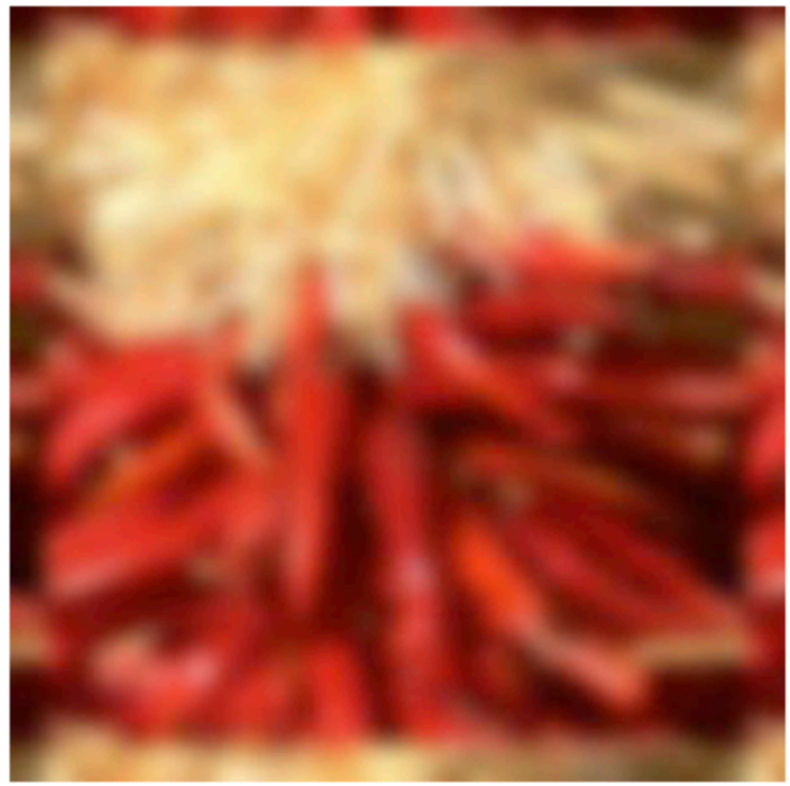
Yucky details

- 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

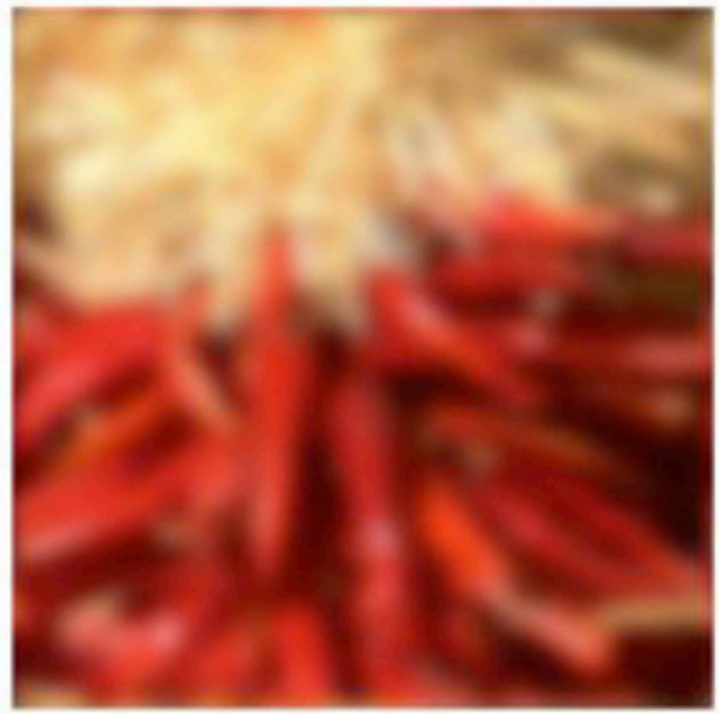
- What about near the edge?
  - the filter window falls off the edge of the image
  - need to extrapolate
  - methods:
    - clip filter (black)
    - wrap around
    - copy edge
    - reflect across edge
    - vary filter near edge



[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

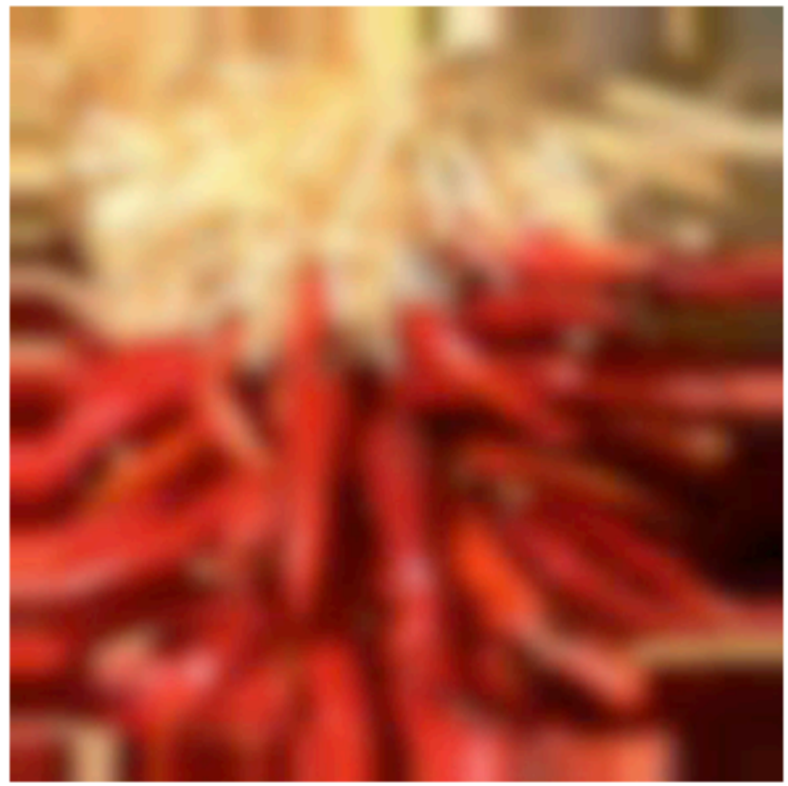
  methods:

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

[Philip Greenspun]

Yucky details

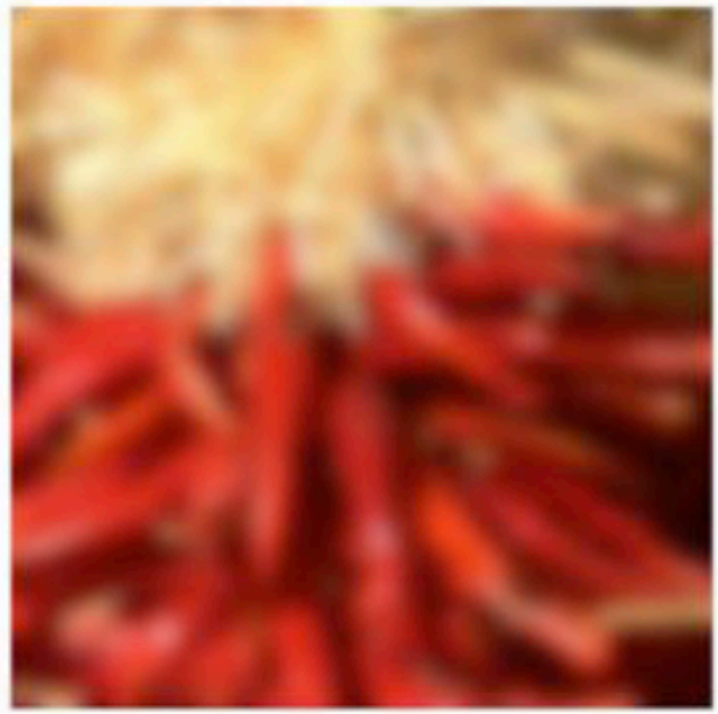
- What about near the edge?
  - the filter window falls off the edge of the image
  - need to extrapolate
  - methods:
    - clip filter (black)
    - wrap around
    - copy edge
    - reflect across edge
    - vary filter near edge



[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]

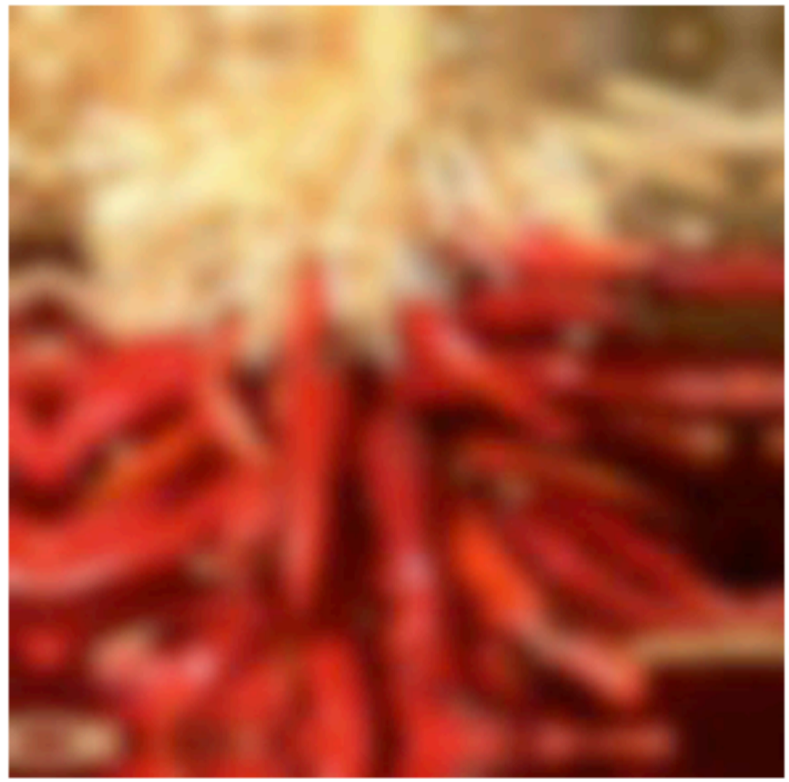
Yucky details

- 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

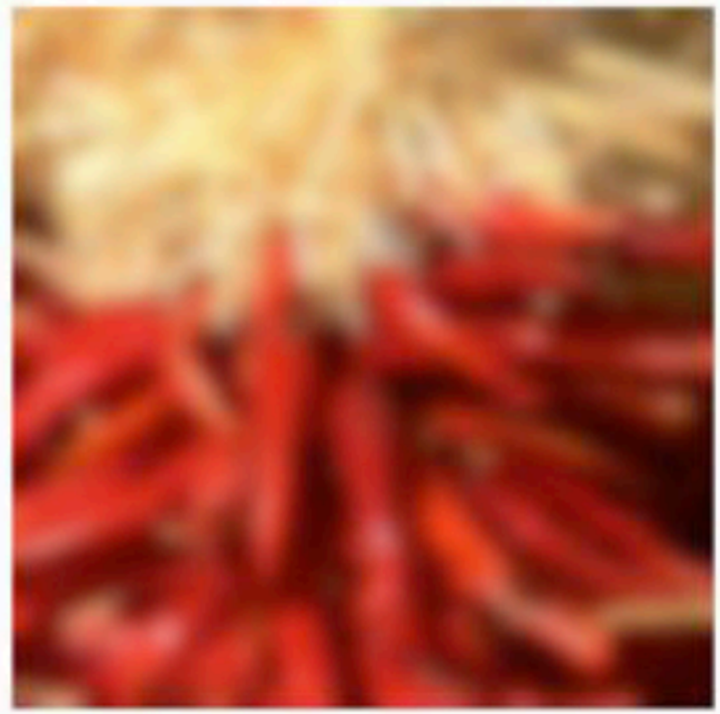
- What about near the edge?
  - the filter window falls off the edge of the image
  - need to extrapolate
  - methods:
    - clip filter (black)
    - wrap around
    - copy edge
    - reflect across edge
    - vary filter near edge



[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

Practical upsampling

- This can also be viewed as:
  1. putting the reconstruction filter at the desired location
  2. evaluating at the original sample positions
  3. taking products with the sample values themselves
  4. summing it up

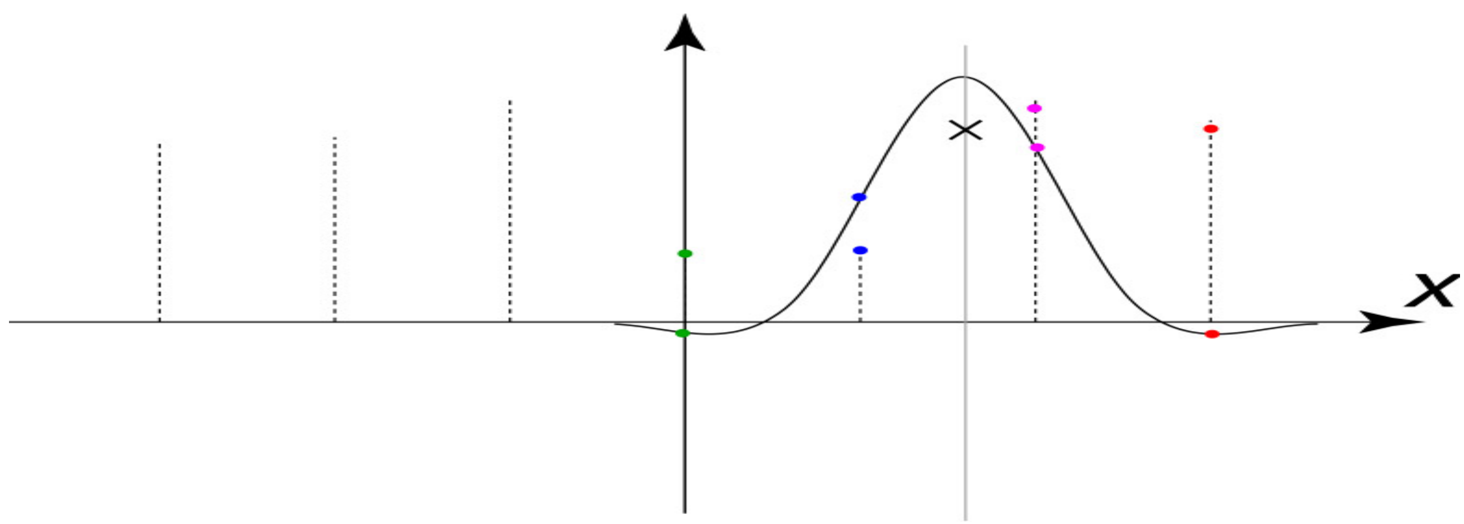
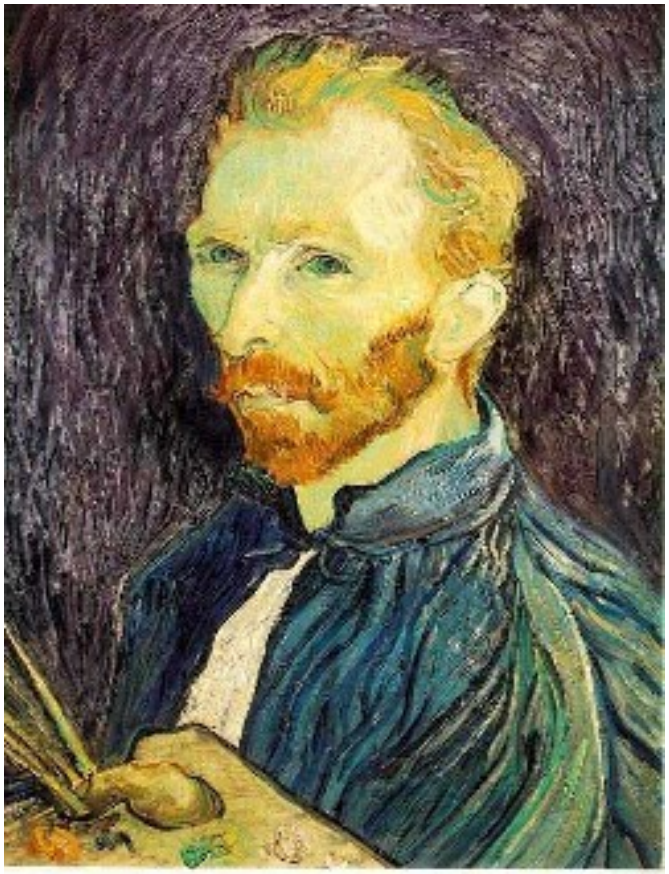


Image Downsampling

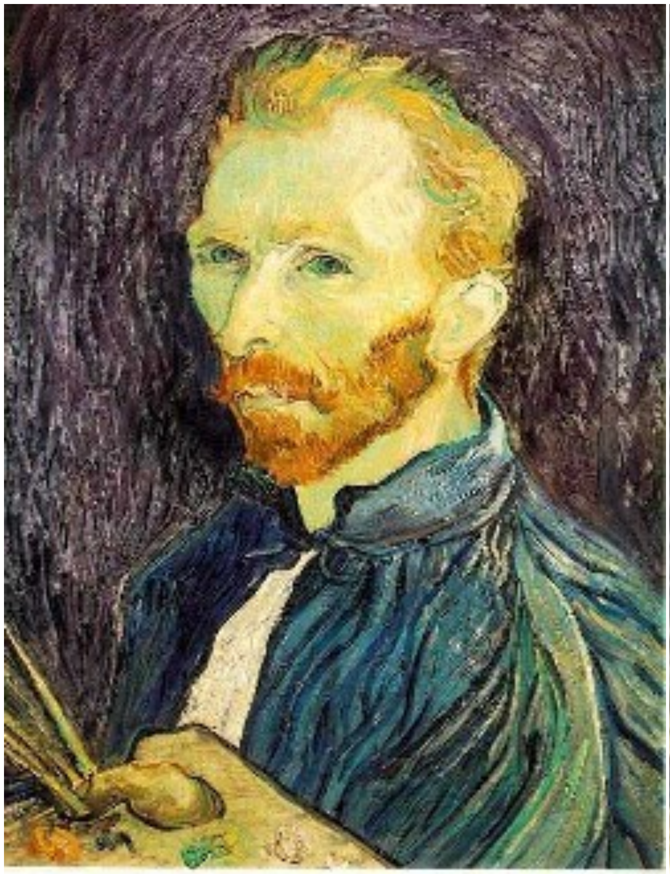


1/4

1/8

Throw away every other row and column to create a 1/2 size image
- called *image sub-sampling*

Image sub-sampling



1/2

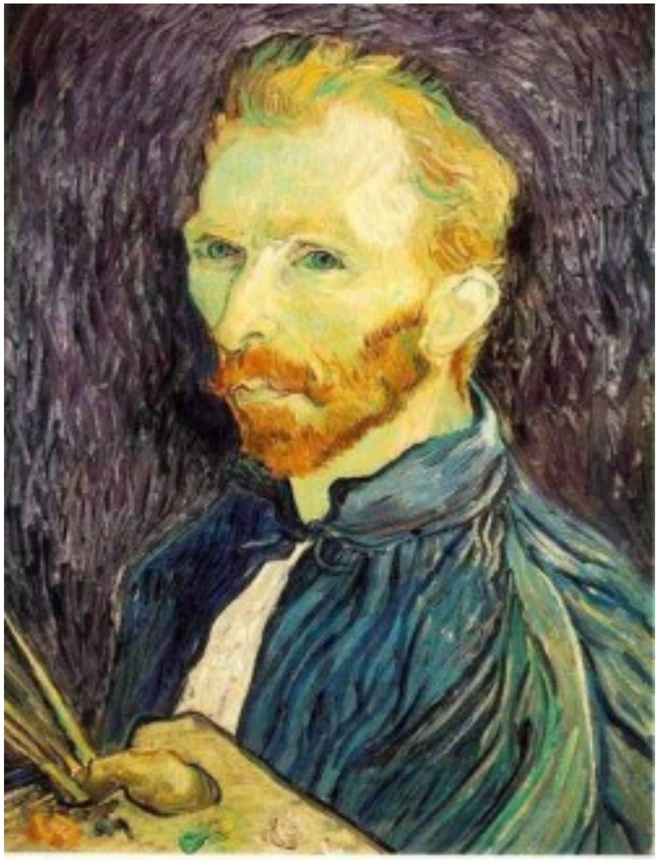
1/4 (2x zoom)

1/8 (4x zoom)

Why does this look so crufty?

Minimum Sampling requirement is not satisfied – resulting in **Aliasing effect**

Subsampling with Gaussian pre-filtering



Gaussian 1/2

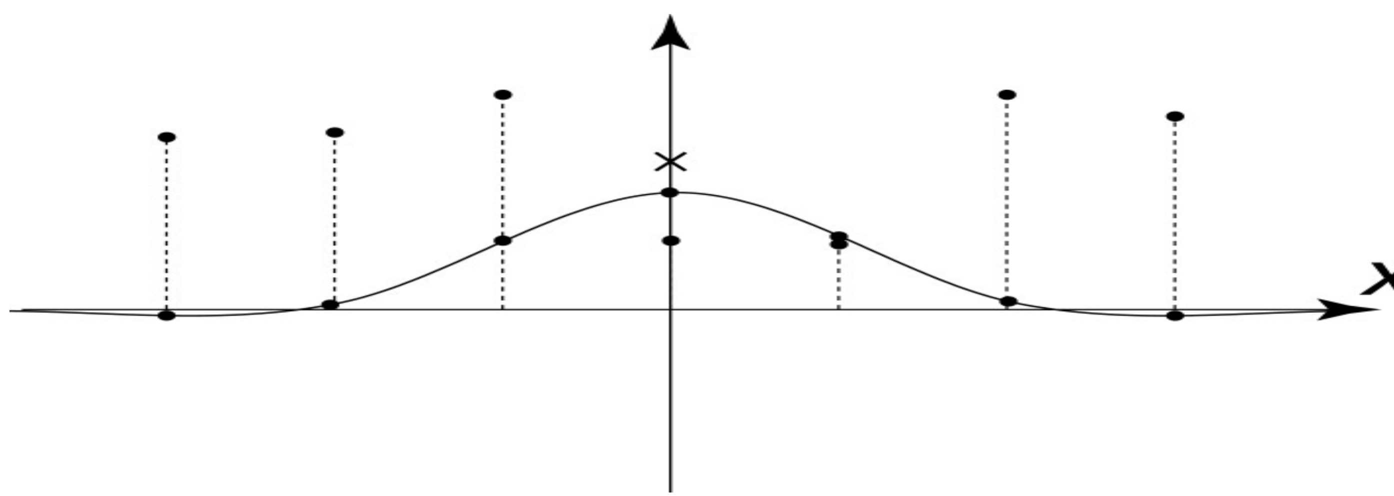
G 1/4

G 1/8

- Solution: filter the image, *then* subsample

Practical downsampling

- Downsampling is similar, but filter has larger support and smaller amplitude.
- Operationally:
  1. Choose reconstruction filter in downsampled space.
  2. Compute the downsampling rate, *d*, ratio of new sampling rate to old sampling rate
  3. Stretch the filter by 1/*d* and scale it down by *d*
  4. Follow upsampling procedure (previous slides) to compute new values (need normalization)



Filter Choice: speed vs quality

Box filter: very fast

Tent filter: moderate quality

Cubic filter: excellent quality, for example Mitchell filter.