CS4620/5620: Lecture 26

Sampling and Anti-Aliasing

Announcements

- 4621
  - Next two Fridays
    - Friday, Nov 2 (splines), Nov 9 (animation)

Antialiasing

- Point sampling makes an all-or-nothing choice in each pixel
  - therefore steps are inevitable when the choice changes
  - discontinuities are bad
- On bitmap devices this is necessary
  - hence high resolutions required
  - 600+ dpi in laser printers to make aliasing invisible
- On continuous-tone devices we can do better

Antialiasing and resampling

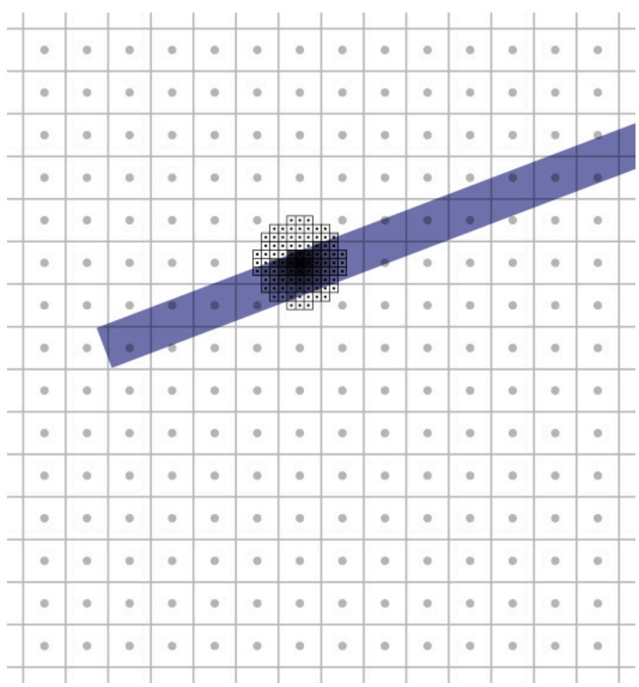
- Antialiasing by regular supersampling is *the same as* rendering a larger image and then resampling it to a smaller size
- So we can re-think this
  - one way: we're computing area of pixel covered by primitive
  - another way: we're computing average color of pixel
    - this way generalizes easily to arbitrary filters, arbitrary images

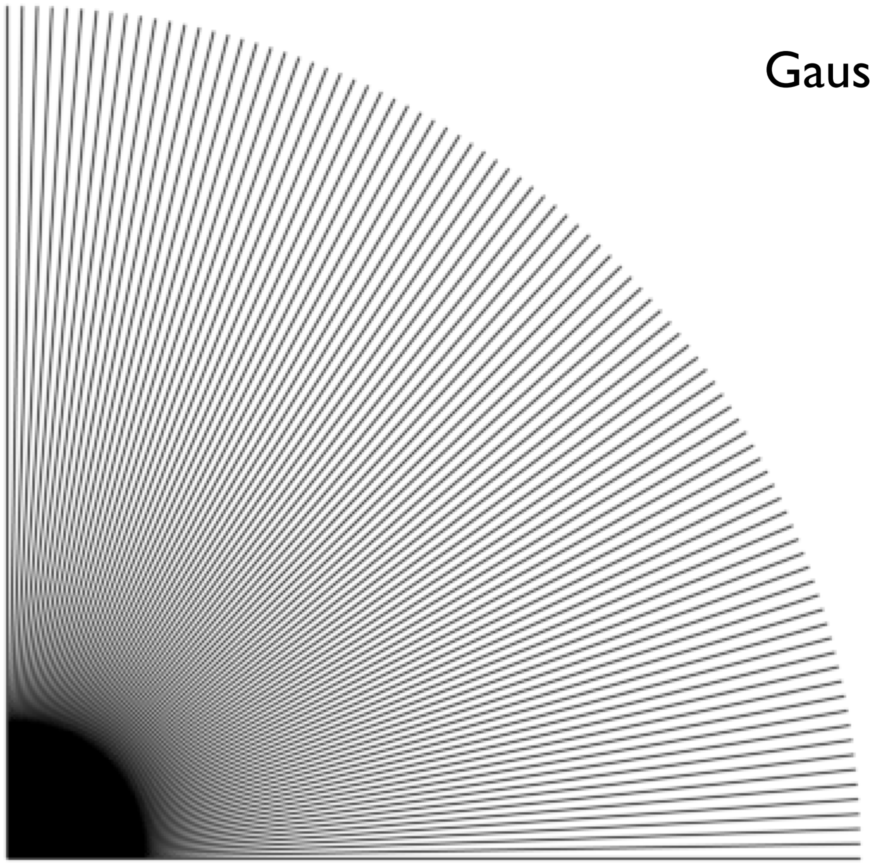
Weighted filtering

- Box filtering problem
  - Treats area near edge same as area near center
    - results in pixel turning on "too abruptly"
- Alternative: weight area by a smoother filter
  - unweighted averaging corresponds to using a box function

Weighted filtering by supersampling

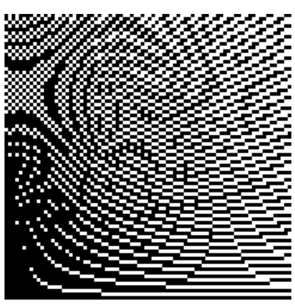
- Compute filtering integral by summing filter values for covered subpixels
- Simple, accurate
- But really slow



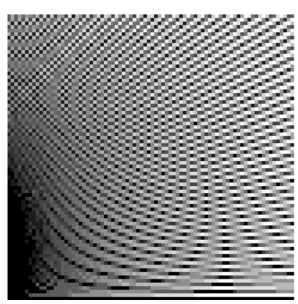


Gaussian filtering in action

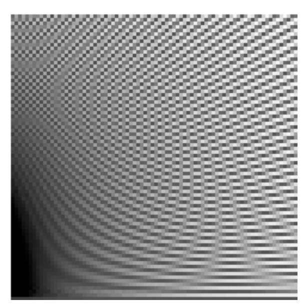
Filter comparison



Point sampling



Box filtering

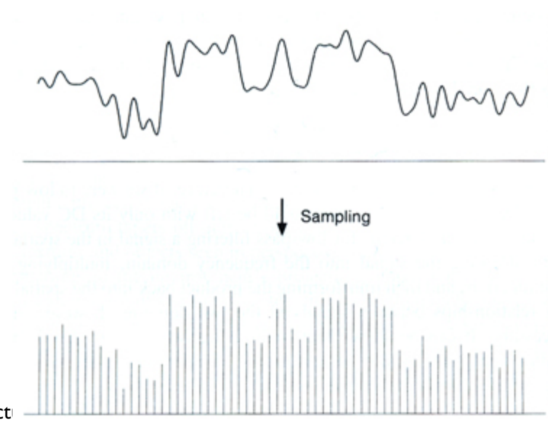


Gaussian filtering

Sampling Theory

Sampled representations

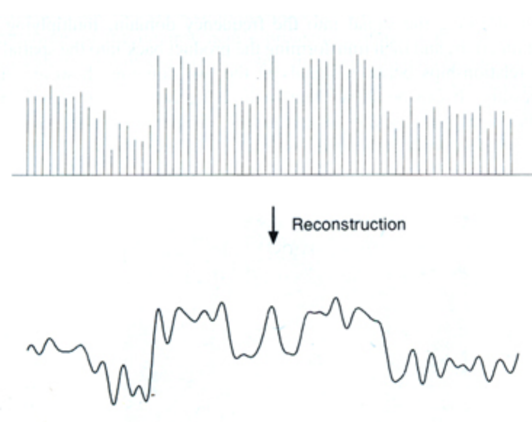
- How to store and compute with continuous functions?
- Common scheme for representation: samples
  - write down the function's values at many points
    - images, textures, etc.



[FvDFH fig.14.14b / Wolberg]

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



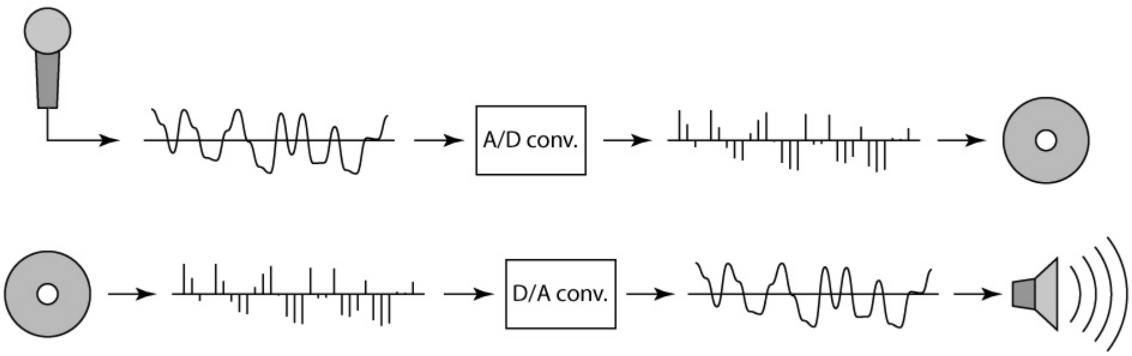
[FvDFH fig.14.14b / Wolberg]

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

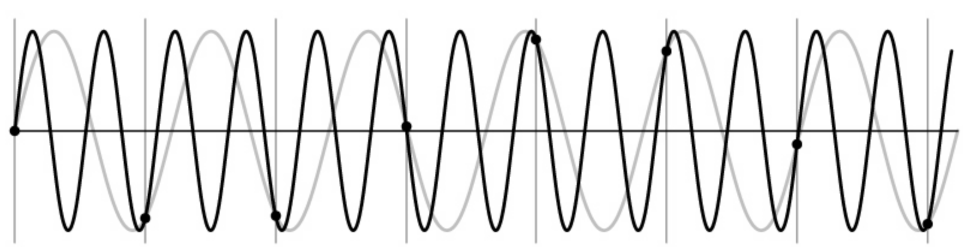
Sampling in digital audio

- 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?



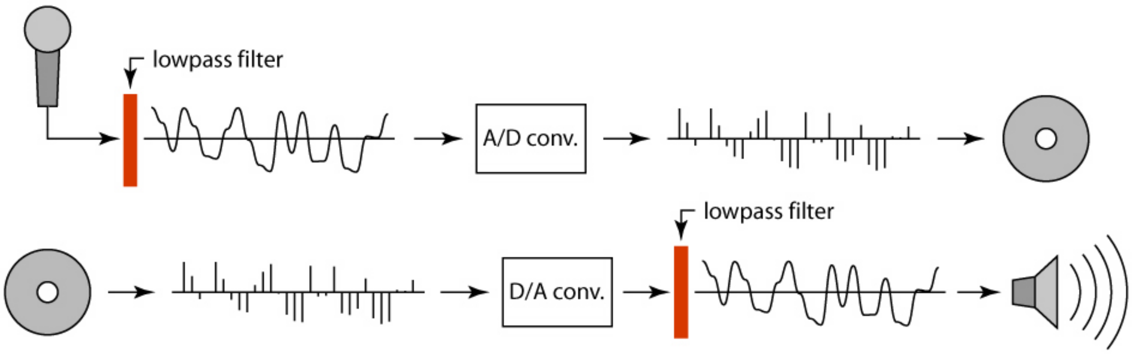
What is aliasing?

- 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



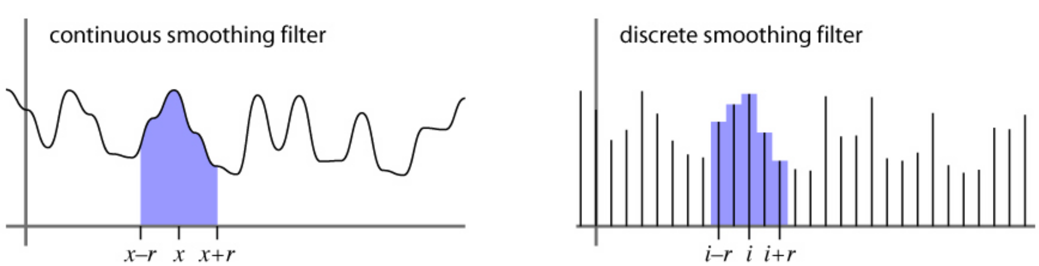
Preventing aliasing

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



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

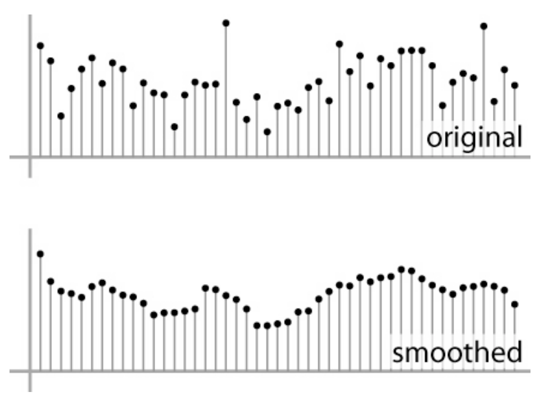


Linear filtering: a key idea

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

- Same moving average operation, expressed mathematically:

$$b_{\text{smooth}}[i] = \frac{1}{2r+1} \sum_{j=i-r}^{i+r} b[j]$$

Discrete convolution

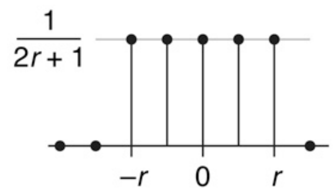
- Simple averaging: $$b_{\text{smooth}}[i] = \frac{1}{2r+1} \sum_{j=i-r}^{i+r} b[j]$$
  – every sample gets the same weight
- Convolution: same idea but with *weighted* average

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

  – each sample gets its own weight (normally zero far away)
- This is all convolution is: it is a **moving weighted average**

Filters

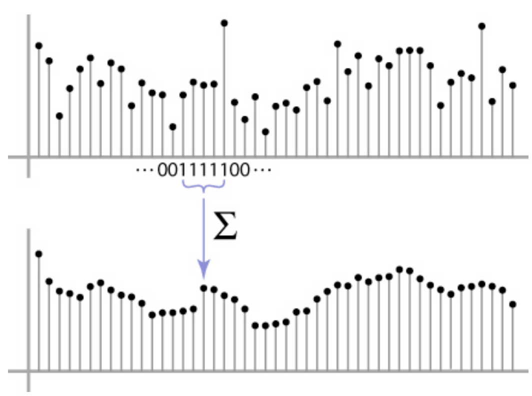
- Sequence of weights $a[j]$ is called a *filter*
- Filter is nonzero over its *region of support*
  – usually centered on zero: support radius r
- Filter is *normalized* so that it sums to 1.0
  – this makes for a weighted average, not just any old weighted sum
- Most filters are symmetric about 0
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



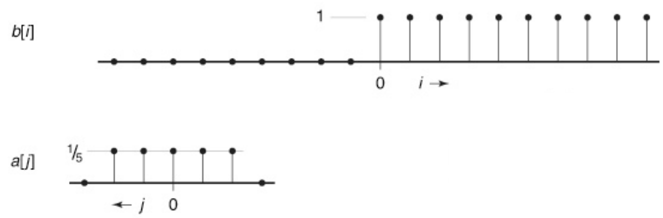
a box filter

Convolution and filtering

- Can express sliding average as convolution with a *box filter*
- $a_{\text{box}} = [\ldots, 0, 1, 1, 1, 1, 1, 0, \ldots]$

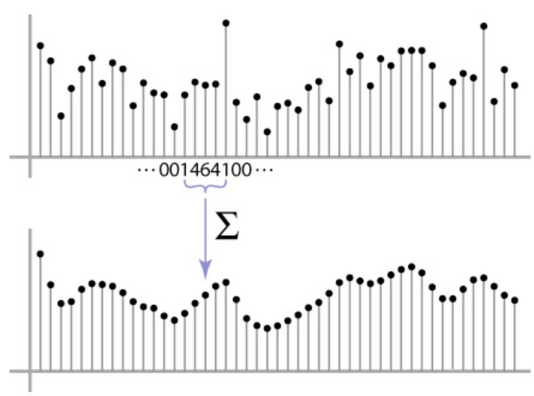


Example: box and step



Convolution and filtering

- Convolution applies with any sequence of weights
- Example: Bell curve (Gaussian-like) […, 1, 4, 6, 4, 1, …]/16



And in pseudocode...

function convolve(sequence a, sequence b, int r, int i)
  $s = 0$
  for $j = -r$ to r
    $s = s + a[j]b[i - j]$
  return s

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, gaussian, …)
  – sharpening
- 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...

function convolve2d(filter2d a, filter2d b, int i, int j)
$s = 0$
$r = a.\text{radius}$
for $i' = -r$ to r **do**
  for $j' = -r$ to r **do**
    $s = s + a[i'][j']b[i - i'][j - j']$
return s

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:

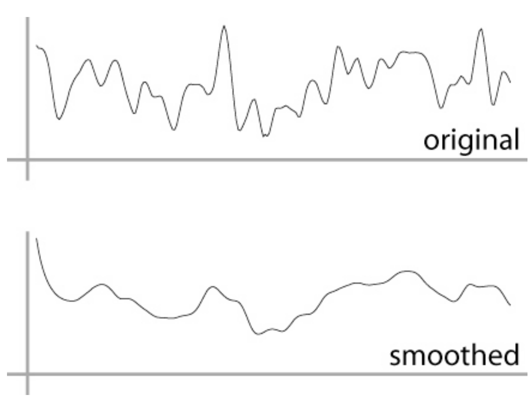
$$(a_2 \star b)[i,j] = \sum_{i'} \sum_{j'} a_2[i',j']b[i-i',j-j']$$
$$= \sum_{i'} \sum_{j'} a_1[i']a_1[j']b[i-i',j-j']$$
$$= \sum_{i'} a_1[i'] \left(\sum_{j'} a_1[j']b[i-i',j-j'] \right)$$

[Philip Greenspun]

two-stage resampling using a separable filter

Continuous convolution

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



Continuous convolution

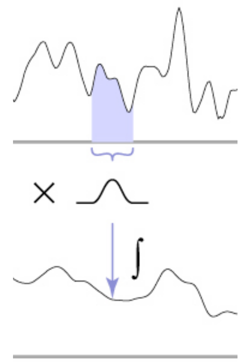
- Sliding average expressed mathematically:

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

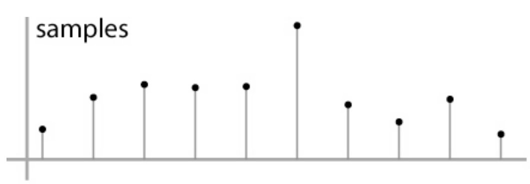
  – note difference in normalization (only for box)
- Convolution just adds weights

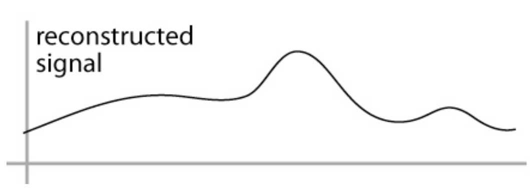
$$(f \star g)(x) = \int_{-\infty}^{\infty} f(t)g(x-t)dt$$

  – weighting is now by a function
  – weighted integral is like weighted average
  – again bounds are set by support of *f(x)*



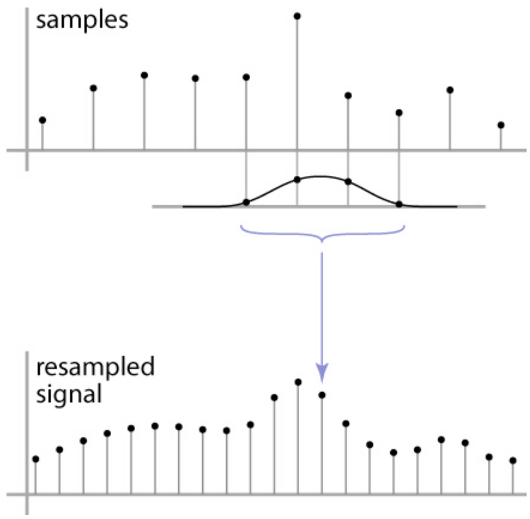
Continuous-discrete convolution





Resampling

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



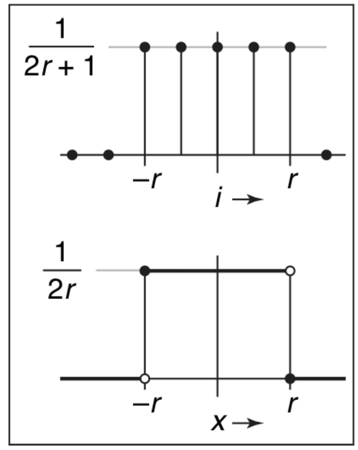
A gallery of filters

- Box filter
  - Simple and cheap
- Tent filter
  - Linear interpolation
- Gaussian filter
  - Very smooth antialiasing filter
- B-spline cubic
  - Very smooth
- ...

Box filter

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

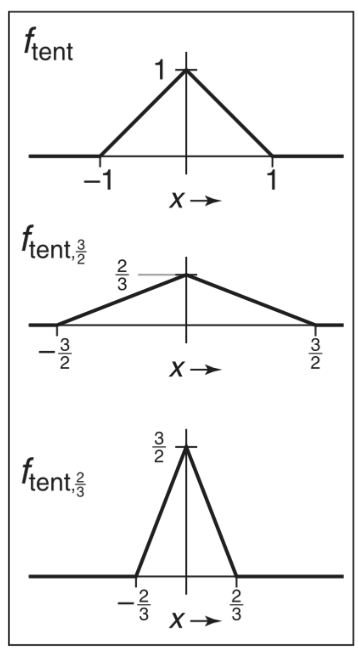
$$f_{\text{box},r}(x) = \begin{cases} 1/(2r) & -r \leq x < r, \\ 0 & \text{otherwise.} \end{cases}$$



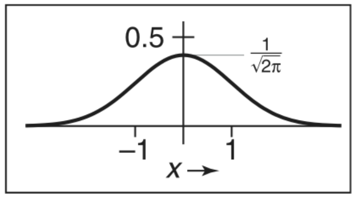
Tent filter

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

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



Gaussian filter



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

Resampling

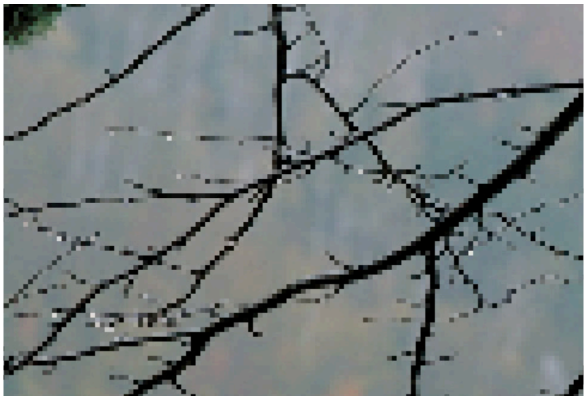
- Changing the sample rate
  - in images, this is enlarging and reducing
- Creating more samples:
  - increasing the sample rate
  - "upsampling"
  - "enlarging"
- Ending up with fewer samples:
  - decreasing the sample rate
  - "downsampling"
  - "reducing"

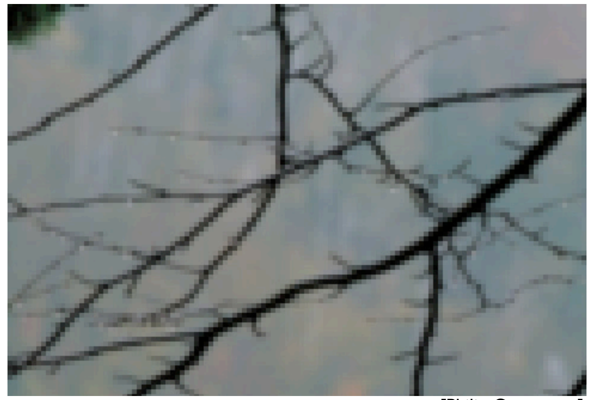
Reducing and enlarging

- 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]



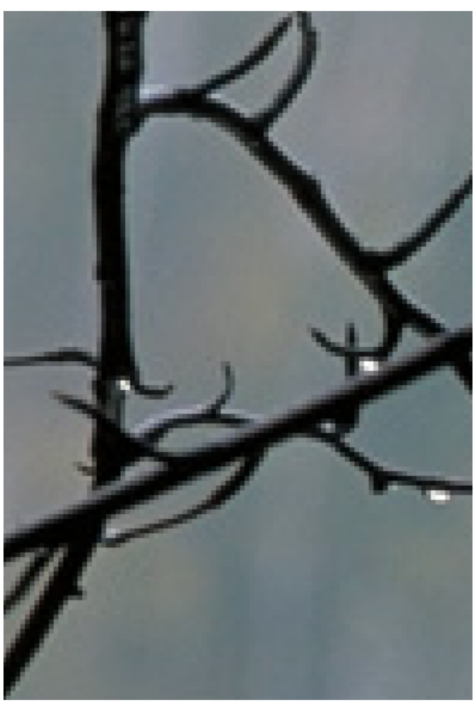


[Philip Greenspun]

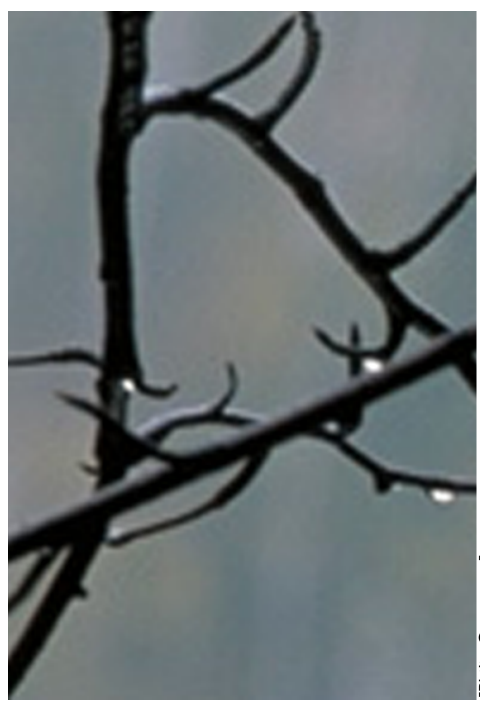
by dropping pixels

gaussian filter

250 pixel width



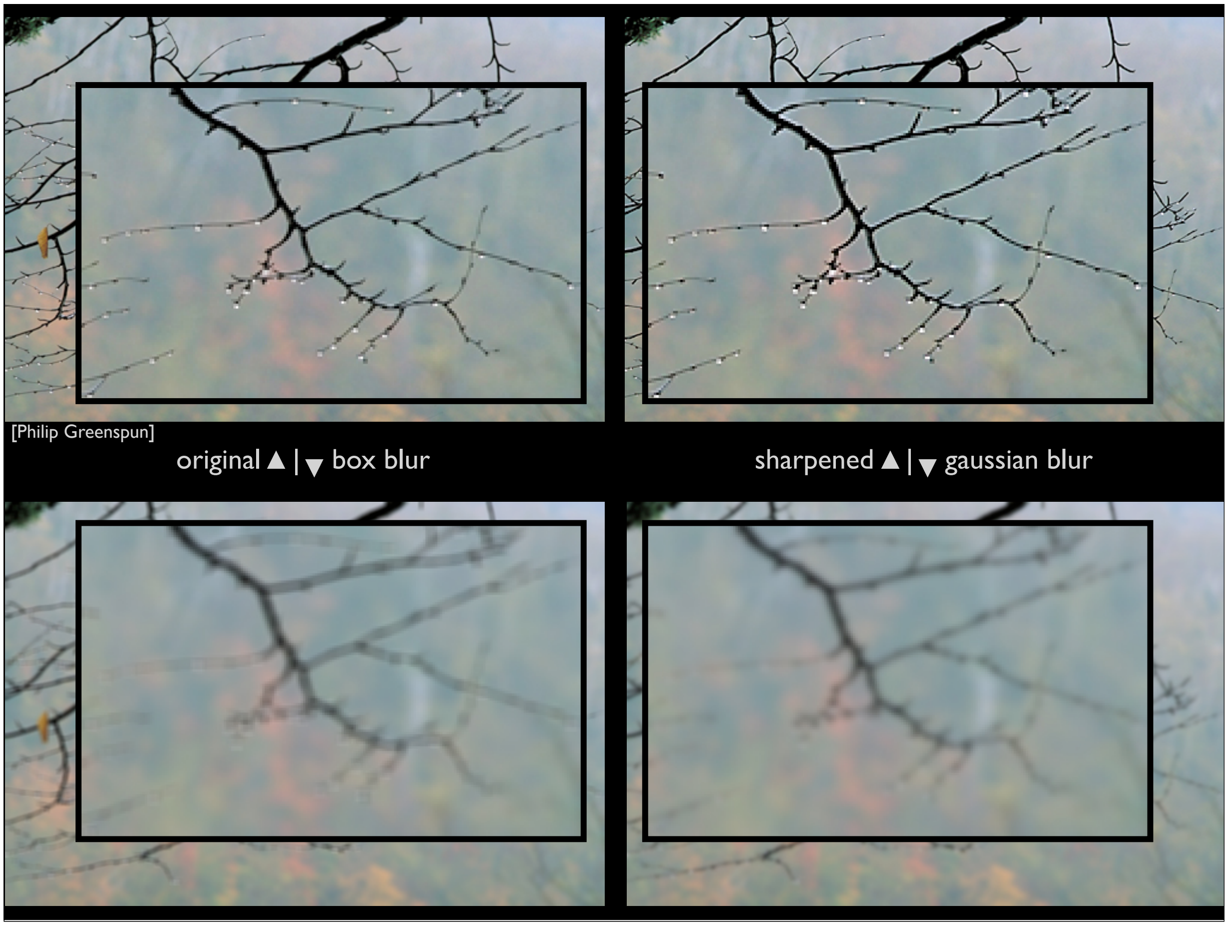
box reconstruction filter



bicubic reconstruction filter

[Philip Greenspun]

4000 pixel width



Types of artifacts

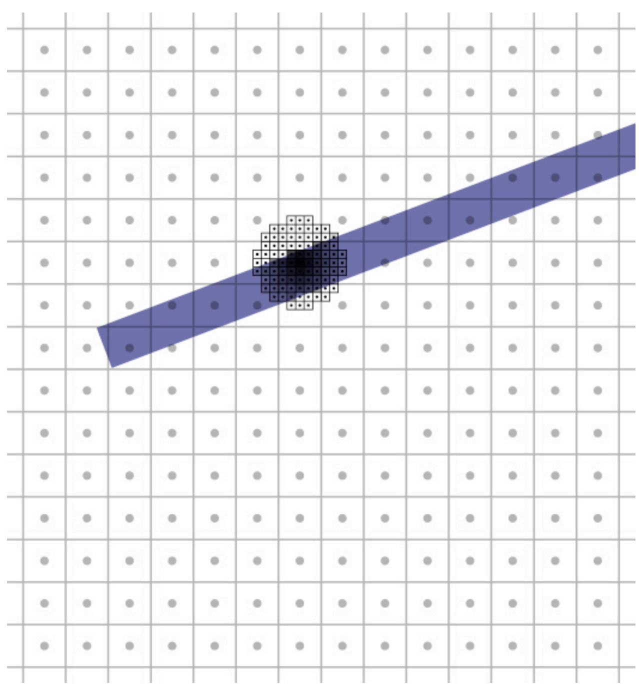
- Garden variety
  - what we saw in this natural image
  - fine features become jagged or sparkle
- Moiré patterns
  - caused by repetitive patterns in input
  - produce large-scale artifacts; highly visible
- Aliasing
- How do I know what filter is best at preventing aliasing?
  - practical answer: experience
  - theoretical answer: there is another layer of cool math behind all this based on Fourier transforms

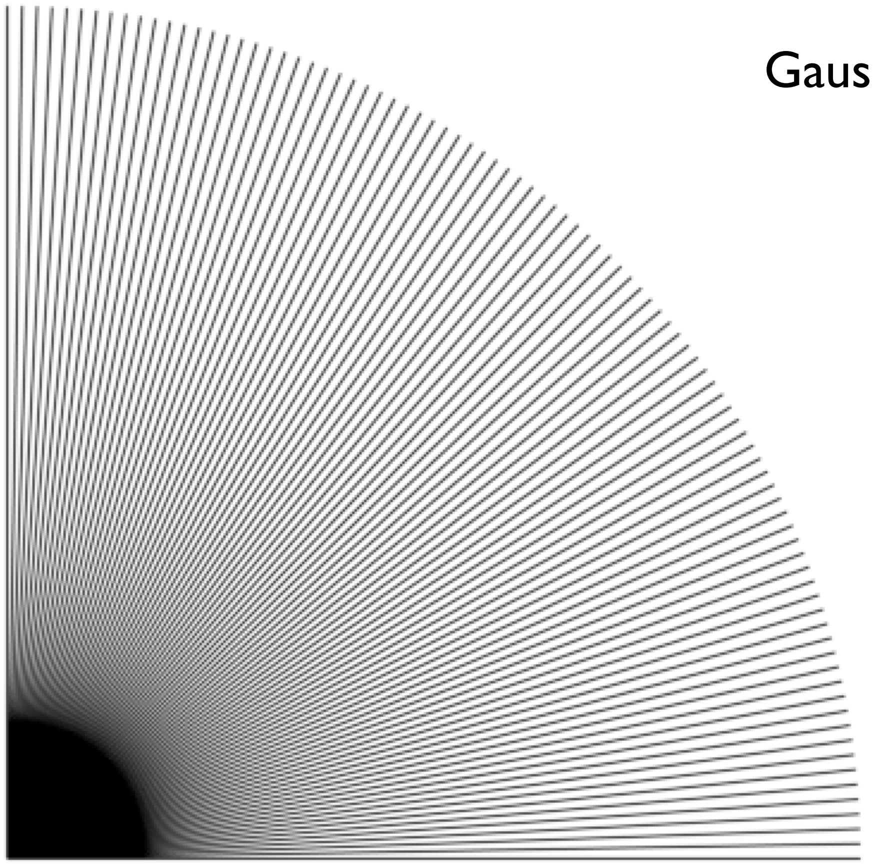
Again: Weighted filtering for line drawing

- Box filtering problem: treats area near edge same as area near center
  - results in pixel turning on "too abruptly"
- Alternative: weight area by a smoother filter
  - unweighted averaging corresponds to using a box function
  - sharp edges mean high frequencies
    - so want a filter with good extinction for higher freqs.
  - a Gaussian is a popular choice of smooth filter
  - important property: normalization (unit integral)

Weighted filtering by supersampling

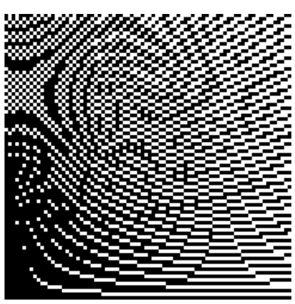
- Compute filtering integral by summing filter values for covered subpixels
- Simple, accurate
- But really slow



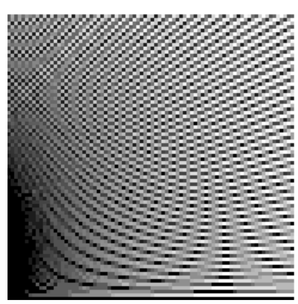


Gaussian filtering in action

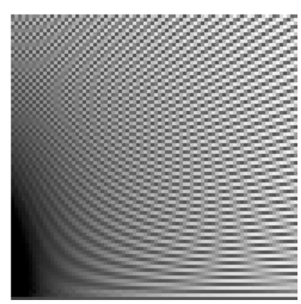
Filter comparison



Point sampling



Box filtering



Gaussian filtering