Convolution

Steve Marschner
CS 4620
Cornell University

Linear, shift-invariant filters

- A fundamental kind of image processing operation
- Linear: preserves summation and scalar multiplication

$$f(I_1 + I_2) = f(I_1) + f(I_2)$$
$$f(aI) = af(I)$$

- Shift-invariant: commutes with shifting the image

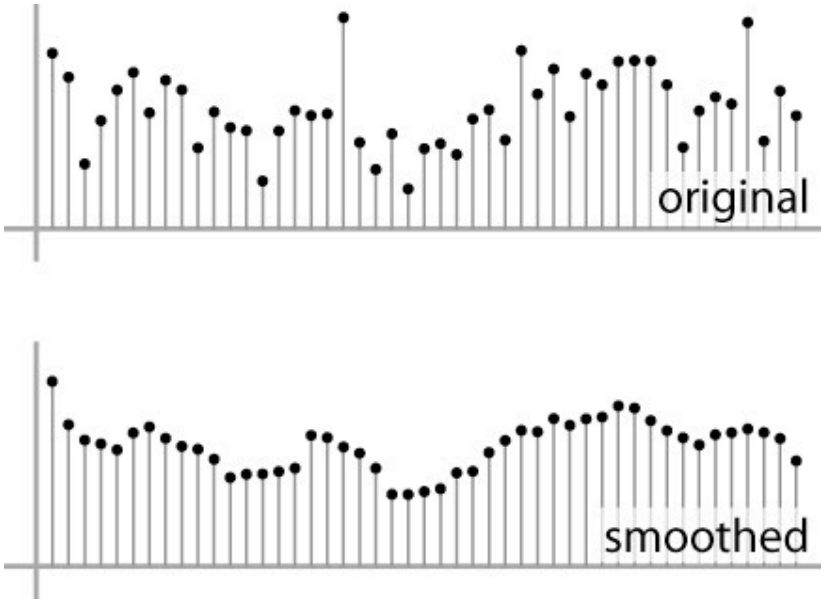
$$f(s_{\Delta x,\Delta y}(I)) = s_{\Delta x,\Delta y}(f(I))$$

(i.e. filter does the same thing at different places in the image)

- Surprisingly, all such operations can be computed with one simple algorithm: **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:

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

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)

- Visually: reflect *b* and slide it over so that *b*[0] lines up with *a*[j]
- This is all convolution is: it is a **moving weighted average**

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

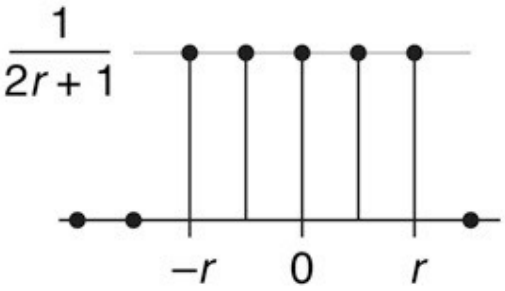
Filters

- 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

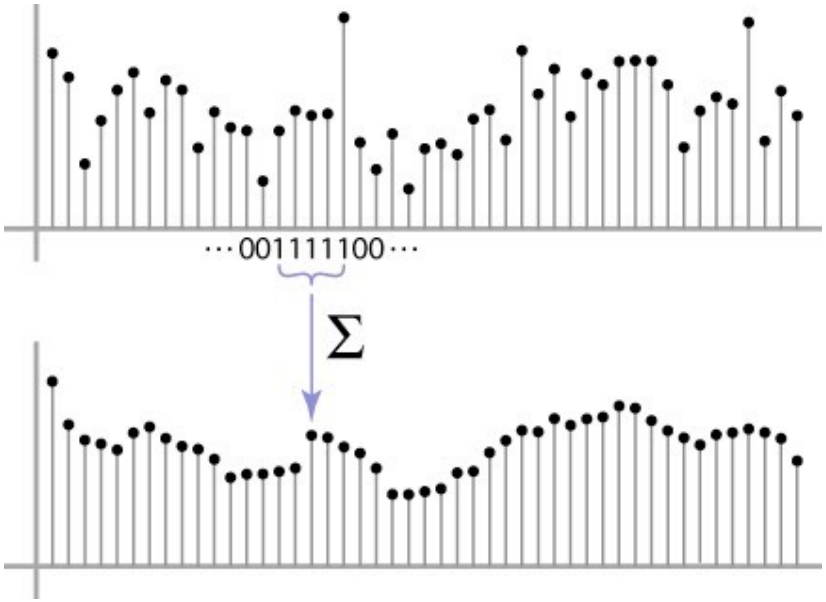
  since for images we usually want to treat left and right the same



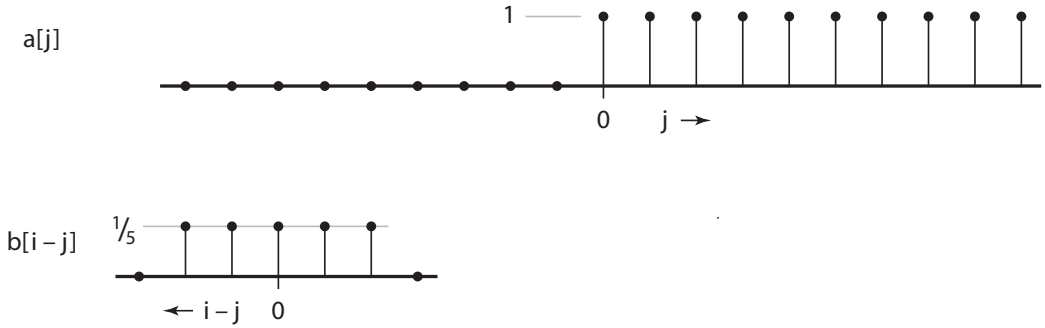
a box filter

Convolution and filtering

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

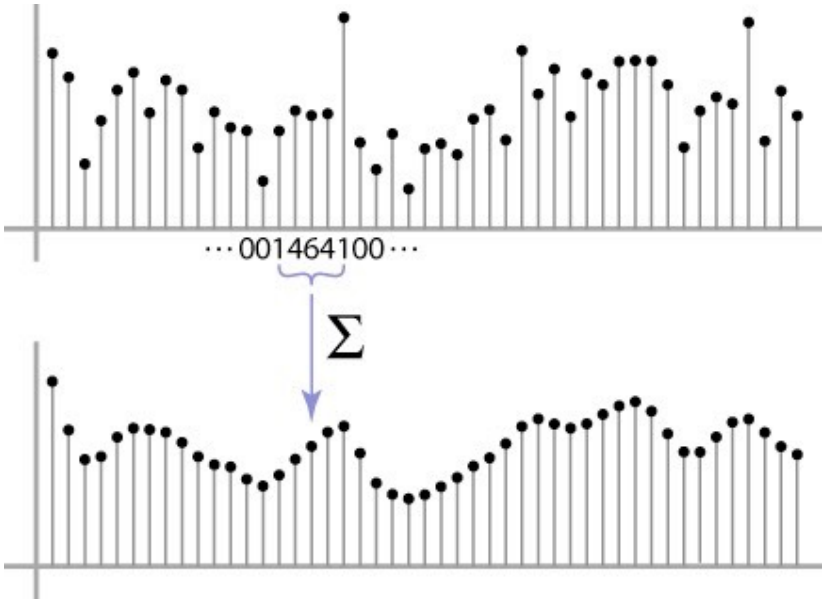


Example: box and step



Convolution and filtering

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



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 $\alpha a \star b = a \star \alpha b = \alpha(a \star b)$

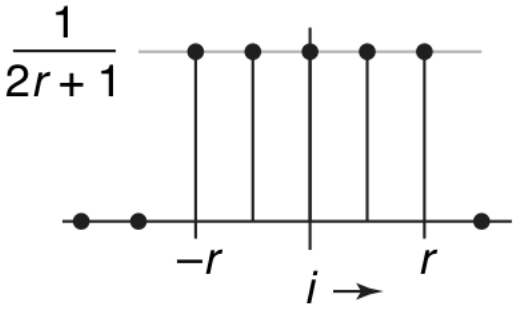
  identity: unit impulse e = […, 0, 0, 1, 0, 0, …]

  $$a \star e = a$$

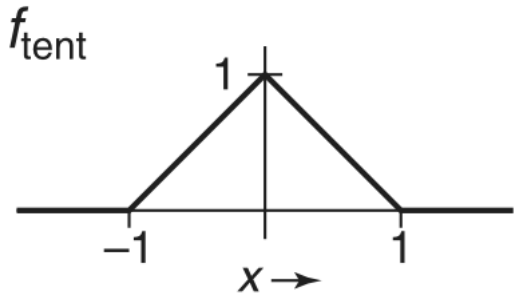
- Conceptually no distinction between filter and signal

Some useful filters

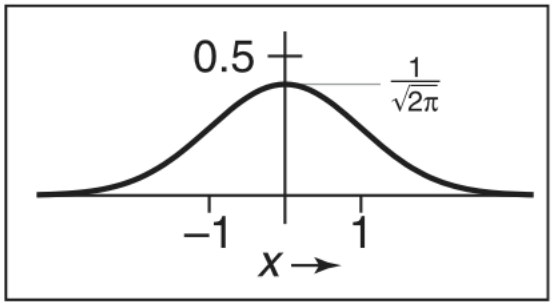
Box: $$a_{\text{box},r}[i] = \begin{cases} 1/(2r+1) & |i| \le r, \\ 0 & \text{otherwise.} \end{cases}$$



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



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



Discretizing filters

- A filter is usually written down as a continuous function
- This function is sampled to get the filter weights

  (the box was an exception)
- It is useful to scale filters to change their size
- It is necessary to trim infinite filters (e.g. gaussian) to represent them
- Typical approach: 2 parameters, radius r and scale s (names vary)

$$b[i] = \begin{cases} \dfrac{f(i/s)}{\sum_{i=-r}^{r} f(i/s)} & i \leq r \\ 0 & \text{otherwise} \end{cases}$$

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)

  feature detection (edges, corners, …)

  in convolutional neural networks (CNNs)

- 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)
$s = 0$
$r = a$.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

Building 2D filters

- Almost always, we build 2D filters from 1D filters like this:

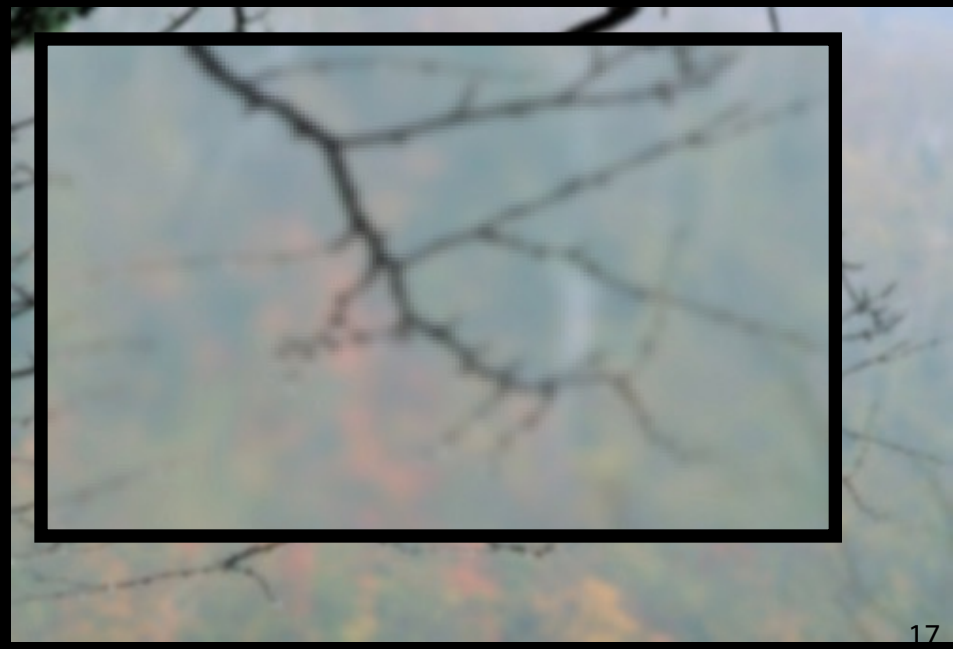
$$a_2[i,j] = a_1[i]a_1[j]$$

- This is called a "separable" filter

[Philip Greenspun]

original ▲ | ▼ box blur

sharpened ▲ | ▼ gaussian blur



Optimization: separable filters

- basic alg. is $O(r^2)$: large filters get expensive fast!
- definition: $a_2[i,j]$ 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:

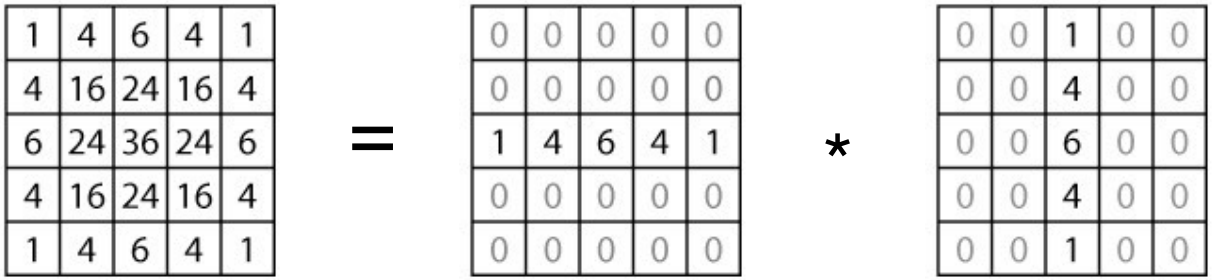
$$(a_2 \star b)[i,j] = \sum_{i'}\sum_{j'} a_2[i',j']b[i-i',j-j']$$

$$= \sum_{i'}\sum_{j'} a_1[i']a_1[j']b[i-i',j-j']$$

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

Separable filtering

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



second, convolve with this

first, convolve with this

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

Yucky details

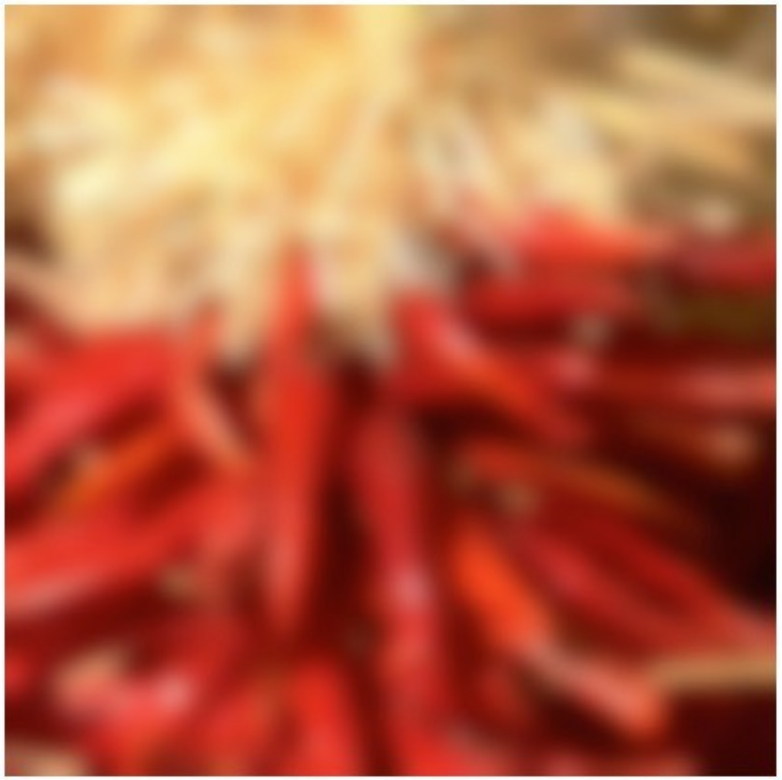
- What about near the edge?

  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]