CS448f: Image Processing For Photography and Vision

Fast Filtering Continued
Filtering by Resampling

• This looks like we just zoomed a small image

• Can we filter by downsampling then upsampling?
Filtering by Resampling
Filtering by Resampling

• Downsampling with rect (averaging down)
• Upsampled with linear interpolation
Use better upsampling?

- Downsampld with rect (averaging down)
- Upsampled with bicubic interpolation
Use better downsampling?

- Downsampled with tent filter
- Upsampled with linear interpolation
Use better downsampling?

- Downsampled with bicubic filter
- Upsampled with linear interpolation
Best Resampling

- Downsampled, blurred, then upsampled with bicubic filter
Best Resampling

- Equivalent to downsampled, then upsampled with a blurred bicubic filter
What's the point?

- Q: If we can blur quickly without resampling, why bother resampling?

- A: Memory use

- Store the blurred image at low res, sample it at higher res as needed.
Recap: Fast Linear Filters

1) Separate into a sequence of simpler filters
   - e.g. Gaussian is separable across dimension
   - and can be decomposed into rect filters

2) Separate into a sum of simpler filters
Recap: Fast Linear Filters

3) Separate into a sum of easy-to-precompute components (integral images)
   - great if you need to compute lots of different filters

4) Resample
   - great if you need to save memory

5) Use feedback loops (IIR filters)
   - great, but hard to change the std.dev of your filter
Histogram Filtering

• The fast rect filter
  – maintained a sum
  – updated it for each new pixel
  – didn't recompute from scratch

• What other data structures might we maintain and update for more complex filters?
Histogram Filtering

• The min filter, max filter, and median filter
  – Only care about what pixel values fall into neighbourhood, not their location
  – Maintain a histogram of the pixels under the filter window, update it as pixels enter and leave
Histogram Updating
Histogram Updating

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

+ + +
+ + +
+ + +
+ + +
+ + +
+ + +
+ + +
Histogram Updating
Histogram-Based Fast Median

• Maintain:
  – hist = Local histogram
  – med = Current Median
  – lt = Number of pixels less than current median
  – gt = Number of pixels greater than current median
Histogram-Based Fast Median

• while (lt < gt):
  – med--
  – Update lt and gt using hist

• while (gt < lt):
  – med++
  – Updated lt and gt using hist
Histogram-Based Fast Median

• Complexity?
• Extend this to percentile filters?
• Max filters? Min filters?
Use of a min filter: dehazing
Large min filter
Difference (brightened)
Weighted Blurs

- Perform a Gaussian Blur weighted by some mask
- Pixels with low weight do not contribute to their neighbors
- Pixels with high weight do contribute to their neighbors
Weighted Blurs

• Can be expressed as:

\[
O(x) = \frac{\sum_{x' = x-f}^{x+f} I(x').e^{-(\sigma_1(I(x)-I(x')))^2}.w(x')}{\sum_{x' = x-f}^{x+f} e^{-(\sigma_1(I(x)-I(x')))^2}.w(x')}
\]

• Where \( w \) is some weight term

• How can we implement this quickly?
Weighted Blurs

• Use homogeneous coordinates for color!
• Homogeneous coordinates uses (d+1) values to represent d-dimensional space
• All values of the form \([a.r, a.g, a.b, a]\) are equivalent, regardless of \(a\).
• To convert back to regular coordinates, divide through by the last coordinate
Weighted Blurs

- This is red: [1, 0, 0, 1]
- This is the same red: [37.3, 0, 0, 37.3]
- This is dark cyan: [0, 3, 3, 6]
- This is undefined: [0, 0, 0, 0]
- This is infinite: [1, 5, 2, 0]
Weighted Blurs

• Addition of homogeneous coordinates is weighted averaging

\[
\begin{bmatrix}
    x.r_0 & x.g_0 & x.b_0 & x \\
    y.r_1 & y.g_1 & y.b_1 & y
\end{bmatrix}
+ \begin{bmatrix}
    x.r_0 & x.g_0 & x.b_0 & x \\
    y.r_1 & y.g_1 & y.b_1 & y
\end{bmatrix}
= \begin{bmatrix}
    (x.r_0 + y.r_1)/(x+y) & (x.g_0 + y.g_1)/(x+y) & (x.b_0 + y.b_1)/(x+y) & (x+y)
\end{bmatrix}
\]
Weighted Blurs

• Often the weight is called alpha and used to encode transparency, in which case this is known as “premultiplied alpha”.
• We’ll use it to perform weighted blurs.
Weight:
Result:
Why bother with uniform weights?

Well... at least it gets rid of the sum of the weights term in the denominator of all of these equations:

$$O(x) = \sum_{x' = x - f}^{x + f} I(x').e^{-\left(\sigma_1(I(x) - I(x'))^2\right)}$$
Weight:
Result: Like a max filter but faster
Weight:
Result: Like a min filter but faster
Weight:
Result: A blur that ignores the dog
In ImageStack:

• Convert to homogeneous coordinates:
  - ImageStack -load dog1.jpg -load mask.png
    -multiply -load mask.png -adjoin c ... 

• Perform the blur
  - ... -gaussianblur 4 ... 

• Convert back to regular coordinates
  - ... -evalchannels “[0]/[3]” “[1]/[3]” “[2]/[3]” 
    -save output.png
The Bilateral Filter

• Pixels are mixed with nearby pixels that have a similar value

\[
O(x) = \sum_{x' = x - f}^{x + f} I(x') \cdot e^{-(\sigma_1 (I(x) - I(x'))^2)} \cdot e^{-(\sigma_2 (x - x')^2)}
\]

• Is this a weighted blur?

\[
w(x) = e^{-(\sigma_1 (I(x) - I(x'))^2)}
\]
The Bilateral Filter

\[
O(x) = \sum_{x' \leq x-f}^{x+f} I(x').e^{-(\sigma_1 (I(x)-I(x'))^2)}.e^{-(\sigma_2 (x-x')^2)}
\]

• No, there’s no single weight per pixel 😞
• What if we picked a fixed intensity level \(a\), and computed:

\[
O(x) = \sum_{x' \leq x-f}^{x+f} I(x').e^{-(\sigma_1 (a-I(x'))^2)}.e^{-(\sigma_2 (x-x')^2)}
\]
The Bilateral Filter

\[ O(x) = \sum_{x' = x - f}^{x + f} I(x') \cdot e^{-(\sigma_1(a - I(x'))^2)} \cdot e^{-(\sigma_2(x - x')^2)} \]

- This formula is correct when \( I(x) = a \)
- And is just a weighted blur, where the weight is:

\[ w(x') = e^{-(\sigma_1(a - I(x'))^2)} \]
The Bilateral Filter

• So we have a formula that only works for pixel values close to a
• How can we extend it to work for all pixel values?
The Bilateral Filter

- 1) Pick lots of values of $a$
- 2) Do a weighted blur at each value
- 3) Each output pixel takes its value from the blur with the closest $a$
  - or interpolate between the nearest 2 $a$’s

- Fast Bilateral Filtering for the Display of High-Dynamic-Range Images
  - Durand and Dorsey 2002
  - Used an FFT to do the blur for each value of $a$
The Bilateral Filter

- Here’s a better way to think of it:
- We can combine the exponential terms...

\[
O(x) = \sum_{x'=x-f}^{x+f} I(x').e^{-(\sigma_1 (a-I(x'))^2)} .e^{-(\sigma_2 (x-x')^2)}
\]

\[
O(x) = \sum_{x'=x-f}^{x+f} I(x').e^{-(\sigma_1 (I(x)-I(x'))^2+\sigma_2 (x-x')^2)}
\]
Linearizing the Bilateral Filter

• The product of an 1D gaussian and an 2D gaussian across different dimensions is a single 3D gaussian.
• So we're just doing a weighted 3D blur
• Axes are:
  – image x coordinate
  – image y coordinate
  – pixel value
The Bilateral Grid – Step 1

Chen et al SIGGRAPH 07

• Take the 2D image $\text{Im}(x, y)$
• Create a 3D volume $V(x, y, z)$, such that:
  – Where $\text{Im}(x, y) = z$, $V(x, y, z) = (z, 1)$
  – Elsewhere, $V(x, y, z) = (0, 0)$
The Bilateral Grid – Step 2

- Blur the 3D volume (using a fast blur)
The Bilateral Grid – Step 3

- Slice the volume at z values corresponding to the original pixel values
Comparison

Input

Regular blur

Bilateral Grid Slice
Pixel Influence

- Each pixel blurred together with
  - those nearby in space (x coord on this graph)
  - and value (y coord on this graph)
Bilateral Grid = Local Histogram Transform

- Take the weight channel:

- Blur in space (but not value)
Bilateral Grid = Local Histogram Transform

• One column is now the histogram of a region around a pixel!

• If we blur in value too, it’s just a histogram with fewer buckets
• Useful for median, min, max filters as well.
The Elephant in the Room

• Why hasn’t anyone done this before?

• For a 5 megapixel image at 3 bytes per pixel, the bilateral grid with 256 value buckets would take up:
  \[-5 \times 1024 \times 1024 \times (3+1) \times 256 = 5120 \text{ Megabytes}\]

• But wait, we never need the original grid, just the original grid blurred...
Use Filtering by Resampling!

• Construct the bilateral grid at low resolution
  – Use a good downsampling filter to put values in the grid
  – Blur the grid with a small kernel (eg 5x5)
  – Use a good upsampling filter to slice the grid

• Complexity?
  – Regular bilateral filter: $O(w*h*f*f)$
  – Bilateral grid implementation:
    • time: $O(w*h)$
    • memory: $O(w/f * h/f * 256/g)$
Use Filtering by Resampling!

• A Fast Approximation of the Bilateral Filter using a Signal Processing Approach
  – Paris and Durand 2006
Dealing with Color

• I’ve treated value as 1D, it’s really 3D
• The bilateral grid should hence really be 5D
• Memory usage starts to go up...
• Cost of splatting and slicing = $2^d$
• Most people just use distance in luminance instead of full 3D distance
  – values in grid are 3D colors (4 bytes per entry)
  – positions of values is just the 1D luminance
    = \frac{(R+G+B)}{3}
Bilateral Grid Demo and Video
Using distance in 3D

vs

Just using distance in luminance

Input

Full Bilateral

Luminance Only Bilateral

Same luminance
There is a disconnect between positions and values

- **Values** in the bilateral grid are the things we want to blur

- **Positions** (and hence distances) in the bilateral grid determine which values we mix

- So we could, for example, get the positions from one image, and the values from another
Joint Bilateral Filter

Input Image

Reference Image

Result
Joint Bilateral Application

- Flash/No Flash photography
- Take a photo with flash (colors look bad)
- Take a photo without flash (noisy)
- Use the edges from the flash photo to help smooth the blurry photo
- Then add back in the high frequencies from the flash photo

- Digital Photography with Flash and No-Flash Image Pairs
  
  *Petschnigg et al, SIGGRAPH 04*
No Flash:
Joint Bilateral Upsample

*Kopf et al, SIGGRAPH 07*

- Say we’ve computed something expensive at low resolution (e.g., tonemapping, or depth)
- We want to use the result at the original resolution
- Use the original image as the positions
- Use the low res solution as the values
- Since the bilateral grid is low resolution anyway, just:
  - read in the low res values at positions given by the downsampled high res image
  - slice using the high res image
Joint Bilateral Upsample Example

- Low resolution depth, high resolution color
- Depth edges probably occur at color edges

Figure 4: Stereo Depth: The low resolution depth map is shown at left. The top right row shows details from the upsampled maps using different methods. Below each detail image is a corresponding 3d view from an offset camera using the upsampled depth map.
Non-Local Means

• Average each pixel with other pixels that have similar local neighborhoods

• Slow as hell
Think of it this way:

• Blur pixels with other pixels that are nearby in patch-space
• Can use a bilateral grid!
  – Except dimensionality too high
  – Not enough memory
  – Splatting and Slicing too costly ($2^d$)
• Solution: Use a different data structure to represent blurry high-D space
• (video)
Key Ideas

• Filtering (even bilateral filtering) is $O(w \times h)$

• You can also filter by downsampling, possibly blurring a little, then upsampling

• The bilateral grid is a local histogram transform that’s useful for many things