Computational Photography

Prof. Feng Liu

Spring 2016

http://www.cs.pdx.edu/~fliu/courses/cs510/

04/05/2016
Last Time

- Digital Camera
  - History of Camera
  - Controlling Camera
- Photography Concepts
Today

- Filters and its applications

noisy image

naïve denoising
Gaussian blur

better denoising
edge-preserving filter

Slide credit: Sylvain Paris and Frédo Durand
The raster image (pixel matrix)
The raster image (pixel matrix)
Perception of Intensity

Slide credit: T. Adelson
Perception of Intensity
Color Image

Slide credit: D. Hoiem
Image Filtering

- Image filtering: compute function of local neighborhood at each pixel position
- One type of “Local operator,” “Neighborhood operator,” “Window operator”

Useful for:
- Enhancing images
  - Noise reduction, smooth, resize, increase contrast, etc.
- Extracting information from images
  - Texture, edges, distinctive points, etc.
- Detecting patterns
  - Template matching, e.g., eye template
Bokeh: Blur in out-of-focus regions of image

Image Correlation Filtering

- Select a filter $g$
  - $g$ is called a filter, mask, kernel, or template
- Center filter $g$ at each pixel in image $f$
- Multiply weights by corresponding pixels
- Set resulting value in output image $h$
- Linear filtering is sum of dot product at each pixel position
- Filtering operation called cross-correlation, and denoted $h = f \otimes g$

Slide credit: C. Dyer
Example: Box Filter

\[
g[\cdot, \cdot] = \frac{1}{9} \begin{bmatrix}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
\end{bmatrix}
\]

Slide credit: David Lowe
Image Filtering

\[ f[\cdot,\cdot] \]

\[ h[\cdot,\cdot] \]

\[ h[m,n] = \sum_{k,l} g[k,l] f[m+k,n+l] \]
Image Filtering

\[ f[\ldots] \]

\[ h[\ldots] \]

\[ h[m, n] = \sum_{k,l} g[k, l] f[m + k, n + l] \]

Credit: S. Seitz
Image Filtering

\[ f[\ldots] \]

\[ h[\ldots] \]

\[ h[m,n] = \sum_{k,l} g[k,l] f[m+k,n+l] \]

Credit: S. Seitz
Image Filtering

\[ f[\ldots] \]

\[ h[\ldots] \]

\[ h[m,n] = \sum_{k,l} g[k,l] f[m+k,n+l] \]

Credit: S. Seitz
**Image Filtering**

Let $f[\cdot, \cdot]$ be the input image, $g[\cdot, \cdot]$ be the filter kernel, and $h[\cdot, \cdot]$ be the filtered image. The filtering operation can be expressed as:

$$h[m, n] = \sum_{k,l} g[k,l] f[m+k, n+l]$$

Credit: S. Seitz
Image Filtering

\[ f[\cdot, \cdot] \]

\[ h[\cdot, \cdot] \]

\[ h[m,n] = \sum_{k,l} g[k,l] f[m+k, n+l] \]

Credit: S. Seitz
Image Filtering

\[ f[\cdot, \cdot] \quad h[\cdot, \cdot] \]

\[ h[m, n] = \sum_{k,l} g[k, l] f[m + k, n + l] \]

Credit: S. Seitz
Image Filtering

\[ h[m,n] = \sum_{k,l} g[k,l] f[m+k,n+l] \]

Credit: S. Seitz
Box Filter

What does it do?

• Replaces each pixel with an average of its neighborhood

• Achieves smoothing effect (i.e., removes sharp features)

• Weaknesses:
  • Blocky results
  • Axis-aligned streaks

Slide credit: David Lowe
Smoothing with Box Filter

Slide credit: C. Dyer
Properties of Smoothing Filters

- **Smoothing**
  - Values all positive
  - Sum to 1 $\Rightarrow$ constant regions same as input
  - Amount of smoothing proportional to mask size
  - Removes “high-frequency” components
  - “low-pass” filter

Slide credit: C. Dyer
Gaussian Filtering

- Weight contributions of neighboring pixels by nearness

- Constant factor at front makes volume sum to 1

- Convolve each row of image with 1D kernel to produce new image; then convolve each column of new image with same 1D kernel to yield output image

\[ G_\sigma = \frac{1}{2\pi \sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}} \]

<table>
<thead>
<tr>
<th></th>
<th>0.003</th>
<th>0.013</th>
<th>0.022</th>
<th>0.013</th>
<th>0.003</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.013</td>
<td>0.059</td>
<td>0.097</td>
<td>0.059</td>
<td>0.013</td>
<td></td>
</tr>
<tr>
<td>0.022</td>
<td>0.097</td>
<td>0.159</td>
<td>0.097</td>
<td>0.022</td>
<td></td>
</tr>
<tr>
<td>0.013</td>
<td>0.059</td>
<td>0.097</td>
<td>0.059</td>
<td>0.013</td>
<td></td>
</tr>
<tr>
<td>0.003</td>
<td>0.013</td>
<td>0.022</td>
<td>0.013</td>
<td>0.003</td>
<td></td>
</tr>
</tbody>
</table>

5 x 5, \( \sigma = 1 \)

Slide credit: Christopher Rasmussen
Smoothing with a Gaussian

- Smoothing with a box actually doesn’t compare at all well with a defocused lens
- Most obvious difference is that a single point of light viewed in a defocused lens looks like a fuzzy blob; but the averaging process would give a little square
- Gaussian is *isotropic* (i.e., rotationally symmetric)

- A Gaussian gives a good model of a fuzzy blob
- It closely models many physical processes (the sum of many small effects)
What does Blurring take away?

original

Slide credit: C. Dyer
What does Blurring take away?

smoothed (5x5 Gaussian)

Slide credit: C. Dyer
Smoothing with Gaussian Filter
Smoothing with Box Filter
Gaussian blur
Gaussian Filters

- What parameters matter here?
- **Standard deviation**, $\sigma$, of Gaussian: determines extent of smoothing

$\sigma = 2$ with 30 x 30 kernel

$\sigma = 5$ with 30 x 30 kernel

Source: D. Hoiem
Effect of $\sigma$

- $\sigma = 1$
- $\sigma = 3$
- $\sigma = 5$

Slide credit: C. Dyer
for sigma=1:3:10
    h = fspecial('gaussian', hsize, sigma);
    out = imfilter(im, h);
    imshow(out);
    pause;
end

Parameter $\sigma$ is the “scale” / “width” / “spread” of the Gaussian kernel, and controls the amount of smoothing.
Gaussian filters

\[ \sigma = 1 \text{ pixel} \]
\[ \sigma = 5 \text{ pixels} \]
\[ \sigma = 10 \text{ pixels} \]
\[ \sigma = 30 \text{ pixels} \]
Gaussian Filters

- What parameters matter here?
- **Size** of kernel or mask

$\sigma = 5$ with $10 \times 10$ kernel

$\sigma = 5$ with $30 \times 30$ kernel

Slide credit: C. Dyer
How big should the filter be?

- Gaussian function has infinite “support” but need a finite-size kernel
- Values at edges should be near 0
- $\sim 98.8\%$ of area under Gaussian in mask of size $5\sigma \times 5\sigma$
- In practice, use mask of size $2k+1 \times 2k+1$ where $k \approx 3\sigma$
- Normalize output by dividing by sum of all weights

Slide credit: C. Dyer
Sharpening Filters

Original

(Note that filter sums to 1)

Source: D. Lowe
Sharpening Filters

- Sharpen an out of focus image by subtracting a multiple of a blurred version

Source: D. Lowe
Sharpening
Sharpening by Unsharp Masking

- \[ h = f + k(f^* g) \] where \( k \) is a small positive constant and \( g = \)

\[
\begin{array}{ccc}
0 & 1 & 0 \\
1 & -4 & 1 \\
0 & 1 & 0
\end{array}
\]
called a Laplacian mask

- Called **unsharp masking** in photography

Figure 6.32. (a) Original image. (b) Blurred image. (c) Difference between first two. (d) Enhanced image.
Sharpening using Unsharp Mask Filter

Original

Filtered result

Slide credit: C. Dyer
Application: Edge Detection

Vertical Edge (absolute value)
Application: Edge Detection

Sobel

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>-2</td>
<td>-1</td>
</tr>
</tbody>
</table>

Horizontal Edge (absolute value)

Slide credit: C. Dyer
Application: Hybrid Images

Application: XDoG Filters

\[ D_X(\sigma, k, \tau) = G(\sigma) - \tau \cdot G(k \cdot \sigma) \]

\[ E_X(\sigma, k, \tau, \epsilon, \varphi) = \begin{cases} 1, & \text{if } D_X(\sigma, k, \tau) < \epsilon \\ 1 + \text{tanh}(\varphi \cdot (D_X(\sigma, k, \tau))), & \text{otherwise.} \end{cases} \]

Many methods have been proposed to make a photo look like a painting.

Today we look at one: *Painterly-Rendering with Brushes of Multiple Sizes*.

Basic ideas:
- Build painting one layer at a time, from biggest to smallest brushes.
- At each layer, add detail missing from previous layer.


Slide credit: S. Chenney
function paint(sourceImage, $R_1 \ldots R_n$) // take source and several brush sizes
{
    canvas := a new constant color image
    // paint the canvas with decreasing sized brushes
    for each brush radius $R_i$, from largest to smallest do
    {
        // Apply Gaussian smoothing with a filter of size const * radius
        // Brush is intended to catch features at this scale
        referenceImage = sourceImage * $G(fs \ R_i)$
        // Paint a layer
        paintLayer(canvas, referenceImage, $R_i$)
    }
    return canvas
}
Algorithm 2

procedure paintLayer(canvas, referenceImage, R) // Add a layer of strokes
{
  S := a new set of strokes, initially empty
  D := difference(canvas, referenceImage) // euclidean distance at every pixel
      for x=0 to imageWidth stepsize grid do // step in size that depends on brush radius
        for y=0 to imageHeight stepsize grid do {
          // sum the error near (x,y)
          M := the region (x-grid/2..x+grid/2, y-grid/2..y+grid/2)
          areaError := sum(D_{i,j} for i,j in M) / grid^2
          if (areaError > T) then {
            // find the largest error point
            (x1,y1) := max D_{i,j} in M
            s := makeStroke(R, x1, y1, referenceImage)
            add s to S
          }
        }
  paint all strokes in S on the canvas, in random order
}
Results

Original

Biggest brush added

Medium brush added

Finest brush added

Slide credit: S. Chenney
Next Time

- More Filters
- De-noise