CMPSCI 670: Computer Vision
Linear filtering

University of Massachusetts, Amherst
September 22, 2014

Instructor: Subhransu Maji
Today

• Administrivia:
  • Anyone had problems with submitting homework via edlab should email their homework to me (smaji@cs.umass.edu)
  • Late submission policy
    - Everyone has two late days for the entire semester. Beyond that you lose 15% of the homework per day.
  • Office hours this week: Thursday 3:45 - 4:45, CS 274

• Today’s lecture
  • Conclude photometric stereo, aka, shape from shading
  • Linear filtering
Diffuse reflection: Lambert’s law

\[ B = \rho (N \cdot S) = \rho \|S\| \cos \theta \]

- **B**: radiosity (total power leaving the surface per unit area)
- **\( \rho \)**: albedo (fraction of incident irradiance reflected by the surface)
- **N**: unit normal
- **S**: source vector (magnitude proportional to intensity of the source)
Photometric stereo (shape from shading)

• Can we reconstruct the shape of an object based on shading cues?

Luca della Robbia, Cantoria, 1438
Assume:

- A Lambertian object
- A *local shading model* (each point on a surface receives light only from sources visible at that point)
- A set of *known* light source directions
- A set of pictures of an object, obtained in exactly the same camera/object configuration but using different sources
- Orthographic projection

Goal: reconstruct object shape and albedo
Surface model: Monge patch

\[ z = f(x, y) \]

Image Plane

direction of projection

height

y

x

F&P 2nd ed., sec. 2.2.4
Image model

- **Known:** source vectors $S_j$ and pixel values $I_j(x,y)$
- **Unknown:** surface normal $N(x,y)$ and albedo $\rho(x,y)$
- Assume that the response function of the camera is a linear scaling by a factor of $k$
- Lambert’s law:

$$I_j(x,y) = k \rho(x,y) (N(x,y) \cdot S_j)$$

$$= (\rho(x,y)N(x,y)) \cdot (kS_j)$$

$$= g(x,y) \cdot V_j$$
For each pixel, set up a linear system:

\[
\begin{bmatrix}
I_1(x, y) \\
I_2(x, y) \\
\vdots \\
I_n(x, y)
\end{bmatrix}
= \begin{bmatrix}
V_1^T \\
V_2^T \\
\vdots \\
V_n^T
\end{bmatrix}
\begin{bmatrix}
\mathbf{g}(x, y)
\end{bmatrix}
\]

- Obtain least-squares solution for \( \mathbf{g}(x,y) \) (which we defined as \( \mathbf{N}(x,y) \rho(x,y) \))
- Since \( \mathbf{N}(x,y) \) is the unit normal, \( \rho(x,y) \) is given by the magnitude of \( \mathbf{g}(x,y) \)
- Finally, \( \mathbf{N}(x,y) = \mathbf{g}(x,y) / \rho(x,y) \)
Example

Recovered albedo

Recovered normal field

F&P 2nd ed., sec. 2.2.4
Recovering a surface from normals

Recall the surface is written as

\[(x, y, f(x, y))\]

This means the normal has the form:

\[
N(x, y) = \frac{1}{\sqrt{f_x^2 + f_y^2 + 1}} \begin{pmatrix} f_x \\ f_y \\ 1 \end{pmatrix}
\]

If we write the estimated vector \(g\) as

\[
g(x, y) = \begin{pmatrix} g_1(x, y) \\ g_2(x, y) \\ g_3(x, y) \end{pmatrix}
\]

Then we obtain values for the partial derivatives of the surface:

\[
f_x(x, y) = g_1(x, y) / g_3(x, y)
\]

\[
f_y(x, y) = g_2(x, y) / g_3(x, y)
\]

F&P 2nd ed., sec. 2.2.4
**Integrability:** for the surface $f$ to exist, the mixed second partial derivatives must be equal:

\[
\frac{\partial}{\partial y} \left( \frac{g_1(x, y)}{g_3(x, y)} \right) = \frac{\partial}{\partial x} \left( \frac{g_2(x, y)}{g_3(x, y)} \right)
\]

(in practice, they should at least be similar)

We can now recover the surface height at any point by integration along some path, e.g.

\[
f(x, y) = \int_0^x f_x(s, y) \, ds + \int_0^y f_y(x, t) \, dt + C
\]

(for robustness, should take integrals over many different paths and average the results)
Surface recovered by integration

F&P 2nd ed., sec. 2.2.4
Homework 2: Photometric stereo

Input

Estimated albedo

Estimated normals

Integrated height map
https://www.youtube.com/watch?v=S7gXih4XS7A
Linear filtering
Motivation: Image de-noising

- How can we reduce noise in a photograph?
• Let’s replace each pixel with a **weighted** average of its neighborhood

• The weights are called the **filter kernel**

• What are the weights for the average of a 3x3 neighborhood?

![Box filter image](image-url)

“box filter”

Source: D. Lowe
Defining convolution

Let \( f \) be the image and \( g \) be the kernel. The output of convolving \( f \) with \( g \) is denoted \( f \ast g \).

\[
(f \ast g)[m, n] = \sum_{k,l} f[m-k, n-l]g[k,l]
\]

Convention: kernel is “flipped”

MATLAB functions: \texttt{conv2}, \texttt{filter2}, \texttt{imfilter}
Key properties

- **Linearity:** \( \text{filter}(f_1 + f_2) = \text{filter}(f_1) + \text{filter}(f_2) \)

- **Shift invariance:** same behavior regardless of pixel location: \( \text{filter}(\text{shift}(f)) = \text{shift}(\text{filter}(f)) \)

- Theoretical result: any linear shift-invariant operator can be represented as a convolution
Properties in more detail

• Commutative: \( a * b = b * a \)
  • Conceptually no difference between filter and signal

• Associative: \( a * (b * c) = (a * b) * c \)
  • Often apply several filters one after another: \(((a * b_1) * b_2) * b_3)\)
  • This is equivalent to applying one filter: \(a * (b_1 * b_2 * b_3)\)

• Distributes over addition: \(a * (b + c) = (a * b) + (a * c)\)

• Scalars factor out: \(ka * b = a * kb = k(a * b)\)

• Identity: unit impulse \(e = [\ldots, 0, 0, 1, 0, 0, \ldots]\),
  \(a * e = a\)
What is the size of the output?

- MATLAB: \texttt{filter2(g, f, \textit{shape})}
  - \textit{shape} = ‘full’: output size is sum of sizes of f and g
  - \textit{shape} = ‘same’: output size is same as f
  - \textit{shape} = ‘valid’: output size is difference of sizes of f and g
Annoying 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

Source: S. Marschner
What about near the edge?

- the filter window falls off the edge of the image
- need to extrapolate
- methods (MATLAB):
  - clip filter (black): `imfilter(f, g, 0)`
  - wrap around: `imfilter(f, g, 'circular')`
  - copy edge: `imfilter(f, g, 'replicate')`
  - reflect across edge: `imfilter(f, g, 'symmetric')`
Practice with linear filters

Original

Source: D. Lowe
Practice with linear filters

Original

Filtered (no change)

Source: D. Lowe
Practice with linear filters

Original

\[
\begin{array}{ccc}
0 & 0 & 0 \\
0 & 0 & 1 \\
0 & 0 & 0 \\
\end{array}
\]
Practice with linear filters

Original

Shifted left
By 1 pixel

Source: D. Lowe
Practice with linear filters

Original

Source: D. Lowe
Practice with linear filters

Original

Blur (with a box filter)

Source: D. Lowe
Practice with linear filters

Original

(Note that filter sums to 1)
Practice with linear filters

Original

Sharpening filter
- Accentuates differences with local average

Source: D. Lowe
Sharpening

before

after

Source: D. Lowe
Sharpening

What does blurring take away?

Let’s add it back:
Smoothing with box filter revisited

- What’s wrong with this picture?
- What’s the solution?

Source: D. Forsyth
Smoothing with box filter revisited

• What’s wrong with this picture?
• What’s the solution?
  • To eliminate edge effects, weight contribution of neighborhood pixels according to their closeness to the center

“fuzzy blob”
Gaussian Kernel

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

- Constant factor at front makes volume sum to 1 (can be ignored when computing the filter values, as we should renormalize weights to sum to 1 in any case)

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

Source: C. Rasmussen
Gaussian Kernel

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

- Standard deviation \( \sigma \): determines extent of smoothing

Source: K. Grauman
Choosing kernel width

• The Gaussian function has infinite support, but discrete filters use finite kernels
Choosing kernel width

- Rule of thumb: set filter half-width to about $3\sigma$
Gaussian vs. box filtering
Gaussian filters

• Remove high-frequency components from the image (*low-pass filter*)

• Convolution with self is another Gaussian
  • So can smooth with small-\(\sigma\) kernel, repeat, and get same result as larger-\(\sigma\) kernel would have
  • Convolving two times with Gaussian kernel with std. dev. \(\sigma\) is same as convolving once with kernel with std. dev. \(\sigma \sqrt{2}\)

• *Separable* kernel
  • Factors into product of two 1D Gaussians
  • Discrete example:

\[
\begin{bmatrix}
1 & 2 & 1 \\
2 & 4 & 2 \\
1 & 2 & 1
\end{bmatrix}
= 
\begin{bmatrix}
1 \\
2 \\
1
\end{bmatrix}
\begin{bmatrix}
1 & 2 & 1
\end{bmatrix}
\]

Source: K. Grauman 41
Separability of the Gaussian filter

\[ G_{\sigma}(x, y) = \frac{1}{2\pi\sigma^2} \exp\left( -\frac{x^2 + y^2}{2\sigma^2} \right) \]

\[ = \left( \frac{1}{\sqrt{2\pi\sigma}} \exp\left( -\frac{x^2}{2\sigma^2} \right) \right) \left( \frac{1}{\sqrt{2\pi\sigma}} \exp\left( -\frac{y^2}{2\sigma^2} \right) \right) \]

The 2D Gaussian can be expressed as the product of two functions, one a function of \( x \) and the other a function of \( y \).

In this case, the two functions are the (identical) 1D Gaussian.
Why is separability useful?

• Separability means that a 2D convolution can be reduced to two 1D convolutions (one among rows and one among columns)

• What is the complexity of filtering an $n \times n$ image with an $m \times m$ kernel?
  
  - $O(n^2 m^2)$

• What if the kernel is separable?
  
  - $O(n^2 m)$
• **Salt and pepper noise**: contains random occurrences of black and white pixels

• **Impulse noise**: contains random occurrences of white pixels

• **Gaussian noise**: variations in intensity drawn from a Gaussian normal distribution

Source: S. Seitz
Gaussian noise

- Mathematical model: sum of many independent factors
- Good for small standard deviations
- Assumption: independent, zero-mean noise
Reduction Gaussian noise

Smoothing with larger standard deviations suppresses noise, but also blurs the image.
Reducing salt-and-pepper noise

3x3  5x5  7x7

What’s wrong with the results?
Alternative idea: Median filtering

- A **median filter** operates over a window by selecting the median intensity in the window.

```
  10 15 20
  23 90 27
  33 31 30
```

- Is median filtering linear?

Source: K. Grauman 48
• What advantage does median filtering have over Gaussian filtering?
  • Robustness to outliers
Median filter

Salt-and-pepper noise

Median filtered

MATLAB: `medfilt2(image, [h w])`

Source: M. Hebert