Frequency Domain Filtering
Spatial domain filtering

Frequency domain filtering

Sobel x
Frequency Domain Filtering

low-pass

band-pass
Frequency Domain Filtering
Original image  
Frequency magnitude  
Inverse Fourier transform
In the context of signal processing, the fourier transform plays a crucial role in converting signals from the time domain to the frequency domain. This transformation allows us to analyze the frequency components of a signal. Once the signal is represented in the frequency domain, we can apply filters, such as low-pass filters, to remove unwanted high-frequency components. After filtering, the inverse fourier transform is used to convert the filtered signal back into the time domain, resulting in a smoother or more refined representation of the original signal.
Inverse Fourier transform

Original image

Low-pass filter

Inverse Fourier transform

http://cns-alumni.bu.edu/~slehar/fourier/fourier.html
Inverse Fourier transform

Original image

High-pass filter

Inverse Fourier transform
Inverse Fourier transform

Original image

Band-pass filter

Inverse Fourier transform

http://cns-alumni.bu.edu/~slehar/ourier/ourier.html
http://cns-alumni.bu.edu/~slehar/fourier/fourier.html
Inverse Fourier transform

Original image

Band-pass filter

Inverse Fourier transform

http://cns-alumni.bu.edu/~slehar/fourier/fourier.html
Why does the Gaussian give a nice smooth image, but the square filter give edgy artifacts?

Gaussian filter

Box filter
Match the image to the Fourier magnitude image:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td><img src="image" alt="a" /></td>
<td><img src="image" alt="b" /></td>
<td><img src="image" alt="c" /></td>
<td><img src="image" alt="d" /></td>
<td><img src="image" alt="e" /></td>
</tr>
</tbody>
</table>

1. ![1](image)
2. ![2](image)
3. ![3](image)
4. ![4](image)
5. ![5](image)
Image Subsampling
This image is too big to fit on the screen. How would you reduce it to half its size?
Naive image sub-sampling

‘throw away even rows and columns’

What are the problems with this approach?
Why is the $1/4$ image so blocky (pixelated, aliased)?

How can we fix this?
Add Gaussian (lowpass) pre-filtering

Gaussian filtering
delete even rows
delete even columns

Gaussian filtering
delete even rows
delete even columns

What will the images look like scale to the same size?
Gaussian pre-filtering
Naive subsampling
This sequence of subsampled images is called the...

Gaussian image pyramid
Image Pyramids
What are image pyramids used for?

- Image compression
- Multi-scale texture mapping
- Image blending
- Multi-focus composites
- Noise removal
- Hybrid images
- Multi-scale detection
- Multi-scale registration
The Laplacian Pyramid as a Compact Image Code (1983)

Peter J. Burt and Edward H. Adelson
Constructing a Gaussian Pyramid

repeat
  filter
  subsample
until min resolution reached

Whole pyramid is only 4/3 the size of the original image!
Gaussian pyramid

What happens to the details of the image?

What is preserved at the higher scales?

How would you reconstruct the original image using the upper pyramid?
What happens to the details of the image?

What is preserved at the higher scales?

Not possible
What is lost between levels?
What does blurring take away?
We can retain the residuals with a ...
Laplacian pyramid

Retains the residuals (details) between pyramid levels

Can you reconstruct the original image using the upper pyramid?

What exactly do you need to reconstruct the original image?
Partial answer:

Level 0 = Level 1 (resized) + Level 0

Low frequency component

High frequency component
do\( (i = 0 : n\text{Scales}-1) \) 
\[
\begin{align*}
    l_i &= \text{blur}(f_i) \\
    h_i &= l_i - f_i \\
    f_{i+1} &= \text{subSamp2}(l_i)
\end{align*}
\]
Constructing the Laplacian Pyramid

```
do(i = 0 : nScales-1)
{
    l_i = blur(f_i)
    h_i = l_i - f_i
    f_{i+1} = subSamp2(l_i)
}
```
Constructing the Laplacian Pyramid

\[
\text{do( } i = 0 : nScales-1 \text{ )}
\]
\[
\begin{align*}
    l_i &= \text{blur}(f_i) \\
    h_i &= l_i - f_i \\
    f_{i+1} &= \text{subSamp2}(l_i)
\end{align*}
\]

What do you need to construct the original image?
What do you need to construct the original image?

(1) Residuals
What do you need to construct the original image?

(1) Residuals

(2) Smallest image

$f_2$

$h_1$

$h_0$

$f_0$
do (i = nScales-1:-1:0) 
{
    l_i = upSamp2(f_{i+1})
    f_i = h_i + l_i
}
output: f_0
Why is it called the Laplacian Pyramid?

Difference of Gaussians approximates the Laplacian

unit - Gaussian \approx Laplacian

Image Gradients and Gradient Filtering

16-385 Computer Vision
What is an image edge?
Recall that an image is a 2D function $f(x)$.
How would you detect an edge?
What kinds of filter would you use?
The ‘Sobel’ filter

\[
\begin{array}{ccc}
1 & 0 & -1 \\
2 & 0 & -2 \\
1 & 0 & -1 \\
\end{array}
\]

a derivative filter
(with some smoothing)

*Filter returns large response on vertical or horizontal lines?*
The ‘Sobel’ filter

<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>

a derivative filter
(with some smoothing)

*Filter returns large response on vertical or horizontal lines?*

*Is the output always positive?*
The ‘Sobel’ filter

\[
\begin{array}{ccc}
1 & 2 & 1 \\
0 & 0 & 0 \\
-1 & -2 & -1
\end{array}
\]

a derivative filter
(with some smoothing)

Responds to horizontal lines

Output can be positive or negative
How do you visualize negative derivatives/gradients?
Derivative in X direction

Visualize with scaled absolute value

Derivative in Y direction
The ‘Sobel’ filter

\[
\begin{array}{ccc}
1 & 0 & -1 \\
2 & 0 & -2 \\
1 & 0 & -1 \\
\end{array}
\]

Where does this filter come?
Do you remember this from high school?

\[ f'(x) = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h} \]
Do you remember this from high school?

The derivative of a function \( f \) at a point \( x \) is defined by the limit

\[
 f'(x) = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h}
\]

Approximation of the derivative when \( h \) is small

This definition is based on the ‘forward difference’ but ...
it turns out that using the ‘central difference’ is more accurate

\[ f'(x) = \lim_{h \to 0} \frac{f(x + 0.5h) - f(x - 0.5h)}{h} \]

How do we compute the derivative of a discrete signal?

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>20</td>
<td>10</td>
<td>200</td>
<td>210</td>
<td>250</td>
</tr>
</tbody>
</table>
it turns out that using the ‘central difference’ is more accurate

\[
f'(x) = \lim_{h \to 0} \frac{f(x + 0.5h) - f(x - 0.5h)}{h}
\]

How do we compute the derivative of a discrete signal?

<table>
<thead>
<tr>
<th>10</th>
<th>20</th>
<th>10</th>
<th>200</th>
<th>210</th>
<th>250</th>
<th>250</th>
</tr>
</thead>
</table>

\[
f'(x) = \frac{f(x + 1) - f(x - 1)}{2} = \frac{210 - 10}{2} = 100
\]

ID derivative filter
Decomposing the Sobel filter

\[
\begin{bmatrix}
1 & 0 & -1 \\
2 & 0 & -2 \\
1 & 0 & -1 \\
\end{bmatrix}
= 
\begin{bmatrix}
1 \\
2 \\
1 \\
\end{bmatrix}
\begin{bmatrix}
1 & 0 & -1 \\
\end{bmatrix}
\]

Sobel

What this?
Decomposing the Sobel filter

\[
\begin{bmatrix}
1 & 0 & -1 \\
2 & 0 & -2 \\
1 & 0 & -1 \\
\end{bmatrix}
\]

Sobel

= 

\[
\begin{bmatrix}
1 \\
2 \\
1 \\
\end{bmatrix}
\]

weighted average and scaling

\[
\begin{bmatrix}
1 & 0 & -1 \\
\end{bmatrix}
\]
Decomposing the Sobel filter

\[
\begin{bmatrix}
1 & 0 & -1 \\
2 & 0 & -2 \\
1 & 0 & -1 \\
\end{bmatrix}
= 
\begin{bmatrix}
1 \\
2 \\
1 \\
\end{bmatrix}
\]

weighted average and scaling

What this?
Decomposing the Sobel filter

\[
\begin{bmatrix}
1 & 0 & -1 \\
2 & 0 & -2 \\
1 & 0 & -1 \\
\end{bmatrix}
= 
\begin{bmatrix}
1 \\
2 \\
1 \\
\end{bmatrix}
\]

Sobel

What this?

\[
\begin{bmatrix}
1 & 0 & -1 \\
\end{bmatrix}
\]

x-derivative

weighted average and scaling
The Sobel filter only returns the x and y edge responses.

How can you compute the image gradient?
How do you compute the image gradient?

Choose a derivative filter

\[
S_x = \begin{bmatrix}
1 & 0 & -1 \\
2 & 0 & -2 \\
1 & 0 & -1
\end{bmatrix}
\]

\[
S_y = \begin{bmatrix}
1 & 2 & 1 \\
0 & 0 & 0 \\
-1 & -2 & -1
\end{bmatrix}
\]

What is this filter called?

Run filter over image

\[
\frac{\partial f}{\partial x} = S_x \otimes f
\]

\[
\frac{\partial f}{\partial y} = S_y \otimes f
\]

What are the dimensions?

Image gradient

\[
\nabla f = \left[ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right]
\]

What are the dimensions?
Matching that Gradient!

(a) \[ \nabla f = \left[ 0, \frac{\partial f}{\partial y} \right] \]

(b) \[ \nabla f = \left[ \frac{\partial f}{\partial x}, 0 \right] \]

(c) \[ \nabla f = \left[ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right] \]
Image Gradient

Gradient in x only

\[ \nabla f = \left[ \frac{\partial f}{\partial x}, 0 \right] \]

Gradient in y only

\[ \nabla f = \left[ 0, \frac{\partial f}{\partial y} \right] \]

Gradient in both x and y

\[ \nabla f = \left[ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right] \]

Gradient direction

\(?\)

Gradient magnitude

\(?\)
Image Gradient

Gradient in x only
\[ \nabla f = \left[ \frac{\partial f}{\partial x}, 0 \right] \]

Gradient in y only
\[ \nabla f = \left[ 0, \frac{\partial f}{\partial y} \right] \]

Gradient in both x and y
\[ \nabla f = \left[ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right] \]

Gradient direction
\[ \theta = \tan^{-1} \left( \frac{\frac{\partial f}{\partial y}}{\frac{\partial f}{\partial x}} \right) \]

Gradient magnitude
\[ ||\nabla f|| = \sqrt{\left( \frac{\partial f}{\partial x} \right)^2 + \left( \frac{\partial f}{\partial y} \right)^2} \]

How does the gradient direction relate to the edge?

What does a large magnitude look like in the image?
Common ‘derivative’ filters

<table>
<thead>
<tr>
<th></th>
<th>Sobel</th>
<th></th>
<th>Scharr</th>
<th></th>
<th>Roberts</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><img src="image" alt="Sobel" /></td>
<td><img src="image" alt="Scharr" /></td>
<td><img src="image" alt="Roberts" /></td>
<td><img src="image" alt="Robert" /></td>
<td></td>
</tr>
<tr>
<td><img src="image" alt="Sobel" /></td>
<td><img src="image" alt="Scharr" /></td>
<td><img src="image" alt="Roberts" /></td>
<td><img src="image" alt="Robert" /></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
How do you find the edge from this signal?
How do you find the edge from this signal?

Intensity plot

Use a derivative filter!
How do you find the edge from this signal?

Intensity plot

Use a derivative filter!

Derivative plot

What happened?
How do you find the edge from this signal?

Intensity plot

Use a derivative filter!

Derivative plot

Derivative filters are sensitive to noise
Don’t forget to smooth before running derivative filters!
Laplace filter
A.K.A. Laplacian, Laplacian of Gaussian (LoG), Marr filter, Mexican Hat Function
Laplace filter
A.K.A. Laplacian, Laplacian of Gaussian (LoG), Marr filter, Mexican Hat Function
Laplace filter

A.K.A. Laplacian, Laplacian of Gaussian (LoG), Marr filter, Mexican Hat Function
\[ f'(x) = \lim_{h \to 0} \frac{f(x + 0.5h) - f(x - 0.5h)}{h} \]

derivative filter \[
\begin{array}{ccc}
1 & 0 & -1
\end{array}
\]

\[ f''(x) \approx \frac{\delta^2_h[f](x)}{h^2} = \frac{f(x + h) - 2f(x) + f(x - h)}{h^2}.
\]

Laplace filter \[
? \quad ? \quad ?
\]
first-order finite difference

\[ f'(x) = \lim_{h \to 0} \frac{f(x + 0.5h) - f(x - 0.5h)}{h} \]

derivative filter \[ \begin{array}{ccc} 1 & 0 & -1 \end{array} \]

second-order finite difference

\[ f''(x) \approx \frac{\delta_h^2[f](x)}{h^2} = \frac{f(x + h) - 2f(x) + f(x - h)}{h^2}. \]

Laplace filter \[ \begin{array}{ccc} 1 & -2 & 1 \end{array} \]
Zero crossings are more accurate at localizing edges
Second derivative is noisy
2D Laplace filter

1D Laplace filter

2D Laplace filter
2D Laplace filter

1  -2  1

1D Laplace filter

?  ?  ?
?  ?  ?
?  ?  ?

2D Laplace filter

hint
If the Sobel filter approximates the first derivative, the Laplace filter approximates ....?
What's different between the two results?
Zero crossings are more accurate at localizing edges
(but not very convenient)
$$h_\sigma(u, v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2 + v^2}{2\sigma^2}}$$

Gaussian

$$\frac{\partial}{\partial x} h_\sigma(u, v)$$

Derivative of Gaussian

$$\nabla^2 h_\sigma(u, v)$$

Laplacian of Gaussian

2D Gaussian Filters
Filtering vs Convolution

16-385 Computer Vision
Filters we have learned so far …

The ‘Box’ filter

\[
\begin{array}{ccc}
1 & 1 & 1 \\
\frac{1}{9} & 1 & 1 \\
1 & 1 & 1
\end{array}
\]

Gaussian filter

\[
\begin{array}{ccc}
1 & 2 & 1 \\
\frac{1}{16} & 2 & 4 & 2 \\
1 & 2 & 1
\end{array}
\]

Sobel filter

\[
\begin{array}{ccc}
1 & 0 & -1 \\
2 & 0 & -2 \\
1 & 0 & -1
\end{array}
\]

Laplace filter

\[
\begin{array}{ccc}
0 & 1 & 0 \\
1 & -4 & 1 \\
0 & 1 & 0
\end{array}
\]
Filtering vs Convolution

Filtering (cross-correlation)

\[ h = g \otimes f \]

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

Convolution

\[ h = g * f \]

\[ h[m, n] = \sum_{k,l} g[k, l] f[m - k, n - 1] \]

Credit: Steve Seitz
Filtering vs Convolution

**filtering**
\[ h = g \otimes f \]

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

(filter flipped vertically and horizontally)

**convolution**
\[ h = g \ast f \]

(image)
\[ h[m, n] = \sum_{k,l} g[k, l] f[m - k, n - 1] \]
Filtering vs Convolution

**Filtering**

(hcross-correlation)

\[ h = g \otimes f \]

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

**Convolution**

\[ h = g \ast f \]

\[ h[m, n] = \sum_{k,l} g[k, l] f[m - k, n - 1] \]

Suppose \( g \) is a Gaussian filter.

How does convolution differ from filtering?

| 1 2 1 |
| 2 4 2 |
| 1 2 1 |

Recall...
Commutative

\[ a \ast b = b \ast a \,.
\]

Associative

\[ (((a \ast b_1) \ast b_2) \ast b_3) = a \ast (b_1 \ast b_2 \ast b_3) \]

Distributes over addition

\[ a \ast (b + c) = (a \ast b) + (a \ast c) \]

Scalars factor out

\[ \lambda a \ast b = a \ast \lambda b = \lambda (a \ast b) \]

**Derivative Theorem of Convolution**

\[ \frac{\partial}{\partial x} (h \ast f) = (\frac{\partial}{\partial x} h) \ast f \]
Derivative Theorem of Convolution

\[
\frac{\partial}{\partial x} (h \ast f) = \left( \frac{\partial}{\partial x} h \right) \ast f
\]
Recall ...

**Gaussian**

**Input**

**Smoothing input**

**Derivative**

**Output**