Filtering Images in The Frequency Domain

Treating Images in the Fourier Domain:

\[ f(x,y) \rightarrow h(x,y) \rightarrow g(x,y) \]

Pixel Domain \[ g(x, y) = f(x, y) \ast h(x, y) \] Convolution

Freq. Domain \[ G(\omega_x, \omega_y) = F(\omega_x, \omega_y) \times H(\omega_x, \omega_y) \] Multiplication

The Convolution Property of the Fourier Transform
Filter Types:

Ideal Low-pass Filters:

\[ H(\omega_x, \omega_y) = \begin{cases} 1 & \sqrt{\omega_x^2 + \omega_y^2} \leq c \\ 0 & \text{else} \end{cases} \]

Cutoff Frequency

Note Undesirable Ringing Effects:

Filter Types:

Other Ideal Filters:

All ideal filters suffer from the ringing (Gibbs) phenomenon
Alternatives to Ideal Filters:

Butterworth Filters:

\[ H(\omega_x, \omega_y) = \frac{1}{1 + \left(\frac{\omega_x^2 + \omega_y^2}{c^2}\right)^{2n}} \]

Gaussian Filters:

\[ H(\omega_x, \omega_y) = e^{-\frac{\omega_x^2 + \omega_y^2}{2\sigma^2}} \]

Control parameters

Advantage

\[ h(x, y) = \sqrt{2\pi\sigma^2} e^{-2\pi^2\sigma^2(x^2+y^2)} \]

MATLAB
High-pass derived Filters:

Butterworth Filters:

\[
H(\omega_x, \omega_y) = \frac{1}{1 + \left(\frac{c^2}{\omega_x^2 + \omega_y^2}\right)^{2n}}
\]

Gaussian Filters:

\[
H(\omega_x, \omega_y) = 1 - \frac{\omega_x^2 + \omega_y^2}{2\sigma^2}
\]

\[
h(x, y) = \delta(x, y) - \sqrt{2\pi \sigma e^{-\frac{2\pi^2 \sigma^2 (x^2 + y^2)}}}
\]

Cutoff Frequency of non-ideal Filters:

Define Power:

\[
P(\omega_x, \omega_y) = |H(\omega_x, \omega_y)|^2
\]

\[
\alpha = 100\% \times \frac{\iint_{(\omega_x, \omega_y) \in \Omega} P(\omega_x, \omega_y) \, d\omega_x \, d\omega_y}{\iint_{(\omega_x, \omega_y) \in \text{Full}} P(\omega_x, \omega_y) \, d\omega_x \, d\omega_y}
\]

A cutoff frequency can be defined by setting alpha.
Homomorphic Filtering:
Recall the Simple Model of (Gray) Image Formation:

- Can distinguish two components:
  - **Illumination** incident on the object: $0 < i(x,y) < \text{Inf}$
  - **Reflectance** function of the object: $0 < r(x,y) < 1$

\[ f(x,y) = i(x,y)r(x,y) \iff F(\omega_x, \omega_y) = I(\omega_x, \omega_y) \ast R(\omega_x, \omega_y) \]

Typically low freq. content

Typically high freq. content

Since we don’t have access generally to the individual I or R components, we can’t design filters for treating them separately.

---

Linear filters in the log domain

**SOLUTION:**
Take log of the image, filter the result, then take exponential

\[
\log f_L(x, y) = \log i_L(x, y) + \log r_L(x, y) \\
F_L(\omega_x, \omega_y) = I_L(\omega_x, \omega_y) + R_L(\omega_x, \omega_y)
\]

**APPLICATION:**
- Improve (Compress) Dynamic Range
- Enhance Contrast
Linear filter in the log domain

- Filter $F_L(\omega_x, \omega_y)$ with a linear filter $H_L(\omega_x, \omega_y)$ such that
  - Dampen low frequencies
  - Enhance high frequencies

Highpass filter in the log domain

---

Sampling and Aliasing

Recall the 1-D scenario:

$$p(t) = \sum_{i=-\infty}^{\infty} \delta(t - iT)$$

$$s(t) = l(t) \sum_{i=-\infty}^{\infty} \delta(t - iT) = \sum_{i=-\infty}^{\infty} l(t) \delta(t - iT)$$

Scaled and shifted delta functions
Sampling and Aliasing

Recall the 1-D scenario:

Time domain:
\[ p(t) = \sum_{i=-\infty}^{\infty} \delta(t - iT) \]

Frequency domain:
\[ P(\omega) = \frac{2\pi}{T} \sum_{i=-\infty}^{\infty} \delta(\omega - \frac{2\pi i}{T}) \]

Spectra of: Cont. Signal Sampling function Sampled Signal

Sampling Theorem:
If a 1-d signal is bandlimited to frequency W, then if it is sampled with a sufficiently high rate (higher than 2W), its spectral replica do not overlap, and it can be reconstructed without loss by linear time-invariant filtering.
Sampling and Aliasing in 2-D

In 2-D, sampling also leads to spectral replication.

Mathematically, this can be represented as:

\[ p(x, y) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} \delta(x - mX, y - nY) \]

where \(\delta(x, y)\) is the Dirac delta function, and \(X, Y\) are the sampling intervals in the spatial domain.

In the frequency domain, the replicated spectrum is given by:

\[ \tilde{p}(\omega_x, \omega_y) = \frac{1}{X} \sum_{m=-\infty}^{\infty} \frac{1}{Y} \sum_{n=-\infty}^{\infty} \delta(\omega_x - m\frac{2\pi}{X}, \omega_y - n\frac{2\pi}{Y}) \]
Sampling and Aliasing in 2-D

If an is bandlimited to a set $S$, then if it is sampled with a sufficiently high rate (directionally higher than $2\times$radius of $S$), then its spectral replica do not overlap, and it can be reconstructed without loss by linear shift-invariant filtering.

Sampling Theorem (2-D):

Many baseband spectra may be reconstructed with the same sampling grid. (Tiling)
Examples and Demos of Aliasing in 2-D

- Ptolemy Demo
- Chalmers Demos

Image Interpolation
Sampling and interpolation in 1D

1. Original signal (continuous)

2. Sampled signal (discrete)

3. Interpolated signal (continuous)

4. Resampled signal (discrete)

Matlab’s interp1() function

Sampling and interpolation in 2D

1. Original surface (continuous)

2. Sampled surface (discrete)

3. Interpolated surface (continuous)

4. Resampled surface (discrete)

Matlab’s interp2() function
When is interpolation useful?

- Zoom; Rotation; In general, spatial transformation
  - The “interpolation” task of digital image processing is transformation from sampled images to resampled images

Any image-related device and software involve interpolation. E.g., television, video games, medical instruments, graphics

Nearest-neighbor interpolation

- The value of the nearest pixel is copied
  - Original (sampled) grid
  - Resampled grid

Copied
Bilinear interpolation

- The values of the four nearest samples are linearly weighted along both axes
  - Original (sampled) grid
  - Resampled grid

\[ g = (1 - p)(1 - q)f_1 + (1 - p)qf_2 + p(1 - q)f_3 + pqf_4 \]

The convolution theorem

- Filtering operation is
  - Convolution in the spatial domain
  - Multiplication in the frequency domain

\[ g(x, y) = f(x, y) * h(x, y) \]
\[ \Leftrightarrow G(\omega_x, \omega_y) = F(\omega_x, \omega_y) \cdot H(\omega_x, \omega_y) \]

- And vice versa

\[ g(x, y) = f(x, y) \cdot h(x, y) \]
\[ \Leftrightarrow G(\omega_x, \omega_y) = F(\omega_x, \omega_y) * H(\omega_x, \omega_y) \]
Sampling

Spatial domain

$f(x, y)$

$\hat{h}(x, y) = \sum_{m,n=-\infty}^{\infty} \delta(x - mX, y - nY)$

Multiplication with a “nailbed”

Frequency domain

$F(\omega_x, \omega_y)$

$H(\omega_x, \omega_y) = \frac{2\pi X}{XY}$

Convolution with a “nailbed”

(Picture taken and modified from “Lecture 8–Filtering in the frequency domain”)

Aliasing

Aliasing

"Reconstruction filter"

"Interpolation filter"
Sampling theorem

• If the original image is sampled at a rate higher than twice the highest frequency of it (the Nyquist rate), then replicated spectra do not overlap and it can be reconstructed without loss.

How to recover the original?
How to recover the original?

Solution
Extract the central spectrum
(This is point-wise multiplication, hence achieved by linear filtering)

Sinc filter

- Extracting the central spectrum is achieved by multiplying a box in the frequency domain
- Its spatial counterpart is called the sinc function

\[ h(x, y) = \text{sinc}(x)\text{sinc}(y) \]

\[ \text{sinc}(x) = \frac{\sin(\pi x)}{\pi x} \]
Separable filters

- A filter is separable when
  \[ h(x, y) = h_1(x)h_2(y) \]
- Separable filters can be applied by two 1D filtering operations; first along one axis, then along the other axis

(Lehmann et al., 1999)

Ideal sampling in 1D (frequency domain)

Original frequency

Sampling

Reconstruction

Box in frequency

Sampled frequency
Ideal sampling in 1D (spatial domain)

Original band-limited signal

Sampling

Interpolation

Sampled points

Sinc function

Infinite support

\[ \text{sinc}(x) = \frac{\sin(\pi x)}{\pi x} \]

Is that all? — No!

- We have the ideal reconstruction filter. Is that all?
- No!
- Practical issues:
  - Fourier transform is often too heavy to compute in order to meet a demand for fast processing
  - Sinc has an infinite support and its computation is lengthy

We find that it’s valuable to have interpolation filters that are locally supported in the spatial domain. We’ll review such filters in the following slides.
Sinc

- 1 at the origin, 0 at the other integer points
- Positive from 0 to 1, negative from 1 to 2, positive from 2 to 3, and so on (oscillating)
- Infinite support (though not shown)

Nearest-neighbor

- The rough extreme for approximating sinc
- Support size is 1 x 1 (very fast computation)
- Bad frequency response
Linear

- Triangle corresponds to linear interpolation because the values of neighbor pixels are weighted by their distance
- Support size is 2 x 2
- Remark: Triangle is a convolution of two boxes

Cubic

- Negative between 1 and 2
- Support size is 4 x 4
Truncated sinc (5 x 5)

- Sinc truncated at ±2.5, giving 5 x 5 support
- Sudden truncation causes overshoots

Truncated sinc (6 x 6)

- Sinc truncated at ±3, giving 6 x 6 support
- Sudden truncation causes overshoots
There are hundreds of interpolators...

- Lehman et al. [1] report quantitative comparison of 31 interpolation kernels
- There are even more in the literature...
- Only several of them are implemented in Matlab

Matlab exercise

- Simple zoom (and shrink)
  - imresize()
- Rotation
  - imrotate()
- General spatial transformation
  - maketform(), imtransform()
  - A bit tricky
- Transformation of coordinates
  - To use interp2(), we have to transform coordinates
imresize()

```matlab
%% Read the original image
f = rgb2gray(im2double(imread('lena.tiff')));

%% Zoom
factor = 3;
gzoom_box = imresize(f, factor, 'box');
gzoom_cubic = imresize(f, factor, 'cubic');
figure(1)
subplot(1, 3, 1)
imagesc(gzoom_box); axis off; axis image; colormap(gray(256))
title('Box')
subplot(1, 3, 2)
imagesc(gzoom_cubic); axis off; axis image; colormap(gray(256))
title('Cubic')
subplot(1, 3, 3)
imagesc(gzoom_box - gzoom_cubic); axis off; axis image; colormap(gray(256))
title('Difference')
```

imrotate()

```matlab
%% Rotation
angle = 30;
grot_nearest = imrotate(f, angle, 'nearest');
grot_bicubic = imrotate(f, angle, 'bicubic');
figure(2)
subplot(1, 3, 1)
imagesc(grot_nearest); axis off; axis image; colormap(gray(256))
title('Nearest')
subplot(1, 3, 2)
imagesc(grot_bicubic); axis off; axis image; colormap(gray(256))
title('Bicubic')
subplot(1, 3, 3)
imagesc(grot_nearest - grot_bicubic); axis off; axis image; colormap(gray(256))
title('Difference')```
%% Affine transformation
A = [1.25, 0.35, 0;
     0.20, 0.80, 0;
     0.00, 0.00, 1];
tform = maketform('affine', A);
gtform_nearest = imtransform(f, tform, 'nearest');
gtform_bicubic = imtransform(f, tform, 'bicubic');

figure(3)
subplot(1, 3, 1)
imagesc(gtform_nearest); axis off; axis image; colormap(gray(256))
title('Nearest')
subplot(1, 3, 2)
imagesc(gtform_bicubic); axis off; axis image; colormap(gray(256))
title('Bicubic')
subplot(1, 3, 3)
imagesc(gtform_nearest - gtform_bicubic); axis off; axis image; colormap(gray(256))
title('Difference')

%% Use interp2() to interpolation
% We have to have transformed "coordinates" to resample
M = 32;
[x, y] = meshgrid(1:M);  % Oriignal grid coordinates
[xg, yg] = tformfwd(tform, x, y);  % Transform according to tform
xg = xg - (max(xg(:)) - M)/2;
yg = yg - (max(yg(:)) - M)/2;

figure(4)
scatter(x(:), y(:), 'b')
hold on
scatter(xg(:), yg(:), 'r')
hold off
axis tight
title('Original and resampling points')
% Do it for image f
M = size(f, 1);
[x, y] = meshgrid(1:M);  % Original grid coordinates
[xg, yg] = tformfwd(tform, x, y);  % Transform according to tform
xg = xg - (max(xg(:)) - M)/2;
yg = yg - (max(yg(:)) - M)/2;
gp2_nearest = interp2(x, y, f, xg, yg, 'nearest');
gp2_cubic = interp2(x, y, f, xg, yg, 'cubic');

figure(5)
s subplot(1, 3, 1)
imagesc(gp2_nearest); axis off; axis image; colormap(gray(256));
title('Nearest')
s subplot(1, 3, 2)
imagesc(gp2_cubic); axis off; axis image; colormap(gray(256))
title('Cubic')
s subplot(1, 3, 3)
imagesc(gp2_nearest - gp2_cubic); axis off; axis image; colormap(gray(256))
title('Difference')