More on Image Processing
Sampling and Reconstruction

Sampling

Reconstruction (Blending)
Sampling and Reconstruction

Original signal

↓ Sampling

Sampled signal

↓ Reconstruction

Reconstructed signal
Sampling Theory

- How many samples are required to represent a given signal without loss of information?
- What signals can be reconstructed without loss for a given sampling rate?
Aliasing

- Aliasing is what happens when we use too few samples: we cannot reconstruct the original function:
Aliasing 2
Aliasing 3
Spatial Aliasing

- Artifacts due to limited spatial resolution

“Jaggies”
Spectral Analysis

- So our image (a function $f(x,y)$) describes how the signal “changes” over $x$ and $y$
- Aliasing occurs when we use too few samples
- What is enough?
  - The more an image changes, the more we need to sample it.
- How do we measure how fast a signal changes?
  - Frequencies
Spectral Analysis

- **Spatial domain:**
  - Function: $f(x)$
  - Filtering: convolution

- **Frequency domain:**
  - Function: $F(u)$
  - Filtering: multiplication

\[
\begin{align*}
  f(x) &= \int_{-\infty}^{\infty} F(u)e^{+i2\pi ux} \, du \\
  F(u) &= \int_{-\infty}^{\infty} f(x)e^{-i2\pi xu} \, dx
\end{align*}
\]
Fourier Transform (1D)

Any signal can be written as a sum of periodic functions.
Fourier Transform (1D)

- Fourier transform:

\[ F(\mu) = \int_{-\infty}^{\infty} f(t) e^{-j2\pi t \mu} \, dt \]

- Inverse Fourier transform:

\[ f(t) = \int_{-\infty}^{\infty} F(\mu) e^{j2\pi \mu t} \, d\mu \]
Extending to 2D

- Let \( f(t,z) \) be a function of two variables, then

\[
F(\mu, \nu) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(t, z)e^{-j2\pi(\mu t + \nu z)} d\mu dv
\]

\[
f(t, z) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(\mu, \nu)e^{j2\pi(\mu t + \nu z)} d\mu dv
\]

- Like before we can extend this to discrete functions and images
Some images and their transforms
Discrete Fourier Transform

- In our case (images) we do not have a continuous function.

- We won’t go into detail, but it can be shown that

\[ \tilde{F}(\mu) = \frac{1}{\Delta T} \sum_{n=-\infty}^{\infty} F(\mu - \frac{n}{\Delta T}) \]

- Sampling \(\rightarrow\) Can’t capture all frequencies (i.e. all details).
  - Therefore \(f(t)\) will be called \textit{Band-limited}. 
The Fourier Transform of Sampled Functions

- We can recover \( f(t) \) from its sampled version if we can isolate \( F(\mu) \)

- The Sampling Theorem:
  - No information is lost if the sample rate is greater than twice the highest frequency
  - *Nyquist* rate
Nyquist–Shannon Sampling Theorem

- A signal can be reconstructed from its samples, if the original signal has no frequencies above 1/2 the sampling frequency - Shannon
- The minimum sampling rate for band-limited function is called “Nyquist rate”

A signal is bandlimited if its highest frequency is bounded. The frequency is called the bandwidth.
Reconstruction

- Piecewise constant
- Linear Interpolation
- Higher order interpolations
Piecewise Constant

Problem: discontinuous!

$F(x)$
Images?

2D piecewise constant function
Piecewise-Linear

Problem: not smooth!
Higher Order?

$Larger \ support: \ more \ information \ from \ neighborhood$
More Problems: Noise & Outliers
What Is Noise?

Usually appears in higher frequencies

![Graph showing original and noisy signals]
Noise in 2D

Again: higher frequencies
Solution:
Use Neighborhood Information

- Neighborhood of 1:

\[ f^{new}(x_i) = \frac{1}{3} \left[ f(x_{i-1}) + f(x_i) + f(x_{i+1}) \right] \]

- Neighborhood of k:

\[ f^{new}(x_i) = \frac{1}{2k+1} \sum_{j=-k}^{+k} f(x_{i+j}) \]

Larger support: more information from neighborhood
Effect: Smoothing!

Does not pass through data points (good or bad?)
Smoothing Out Noise

Noisy

Denoising
Tradeoff

- Larger neighborhood smoothes out noise but also removes details!
Filtering

- Convolution
  - Each output pixel is a linear combination of input pixels in neighborhood with weights prescribed by a filter
Filtering an Image

Filter $H$

Image $H$

[Diagram showing the filtering process with an input, output, and a kernel (flipped).]
Linear Filters

Example: Box Filter (Mean)

\[ H \ast F = G \]
Filtering with different neighborhood size

Original (noisy)

5

50

10

200
Weighted Averaging

- Closer neighbors are more important than far away ones!
- Better results are achieved if we use weighted averaging instead of uniform averaging
- The weights are proportional to the distance between the pixels:

$$I_{\text{new}}(p) = \sum_{q \in N(p)} w(|p - q|)I(q)$$
Gaussian Function

\[ f(x) = e^{-\frac{(x-\mu)^2}{2\sigma^2}} \]
2D Gaussian Function

2D Gaussian

\[ g(x) = e^{-\frac{x^2 + y^2}{2\sigma^2}} \]
Creating 3x3 Discrete Filters

Large $\sigma$

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

$1/16$

Small $\sigma$

\[
\begin{array}{ccc}
1 & 8 & 1 \\
8 & 64 & 8 \\
1 & 8 & 1 \\
\end{array}
\]

$1/100$
Creating 5x5 Discrete Filter
# Discrete 5x5 Gaussian Filter

## Large $\sigma$

<table>
<thead>
<tr>
<th></th>
<th>0.0030</th>
<th>0.0133</th>
<th>0.0219</th>
<th>0.0133</th>
<th>0.0030</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0133</td>
<td>0.0596</td>
<td>0.0983</td>
<td>0.0596</td>
<td>0.0133</td>
<td></td>
</tr>
<tr>
<td>0.0219</td>
<td>0.0983</td>
<td>0.1621</td>
<td>0.0983</td>
<td>0.0219</td>
<td></td>
</tr>
<tr>
<td>0.0133</td>
<td>0.0596</td>
<td>0.0983</td>
<td>0.0596</td>
<td>0.0133</td>
<td></td>
</tr>
<tr>
<td>0.0030</td>
<td>0.0133</td>
<td>0.0219</td>
<td>0.0133</td>
<td>0.0030</td>
<td></td>
</tr>
</tbody>
</table>

## Small $\sigma$

<table>
<thead>
<tr>
<th></th>
<th>0.0000</th>
<th>0.0000</th>
<th>0.0002</th>
<th>0.0000</th>
<th>0.0000</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0000</td>
<td>0.0113</td>
<td>0.0837</td>
<td>0.0113</td>
<td>0.0000</td>
<td></td>
</tr>
<tr>
<td>0.0002</td>
<td>0.0837</td>
<td>0.6187</td>
<td>0.0837</td>
<td>0.0002</td>
<td></td>
</tr>
<tr>
<td>0.0000</td>
<td>0.0113</td>
<td>0.0837</td>
<td>0.0113</td>
<td>0.0000</td>
<td></td>
</tr>
<tr>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0002</td>
<td>0.0000</td>
<td>0.0000</td>
<td></td>
</tr>
</tbody>
</table>

## Simple

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>1</th>
<th>4</th>
<th>7</th>
<th>4</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>16</td>
<td>26</td>
<td>16</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>26</td>
<td>41</td>
<td>26</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>26</td>
<td>16</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>7</td>
<td>4</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$$\frac{1}{273}$$

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>16</td>
<td>8</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$$\frac{1}{100}$$
Low-Pass Filtering for Anti-aliasing

- In effect we can pre-filter the image by blurring and then sample.
- This removes high frequency information and consequently, lowers the minimum sampling rate ("Nyquist rate").
- This is called “Low pass filtering”.
- Note: we trade aliasing for blurring!
More Linear Filters

Original

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

= Identical image
More Linear Filters

Original

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

\[ \ast \]

Shifted left
By 1 pixel
More Linear Filters

\[ \text{Original} \ast \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \text{Blur (with a box filter)} \]
More Linear Filters

Original

\(*\frac{1}{16}\)

Blur (with a Gaussian filter)
Box vs. Gaussian Filters

Difference
Smoothing Filter = Low Pass Filter!

- Gaussian filtering blurs the image!
- It removes the “high-frequency” components from the image and therefore is called a low-pass filter

Original

Blurred
Blur (lowpass filters)

- We can either take a uniform kernel (mean filter)

\[
\frac{1}{9} \quad \frac{1}{9} \quad \frac{1}{9} \\
\frac{1}{9} \quad \frac{1}{9} \quad \frac{1}{9} \\
\frac{1}{9} \quad \frac{1}{9} \quad \frac{1}{9}
\]

- Or a Gaussian kernel

\[
\frac{1}{16} \quad \frac{2}{16} \quad \frac{1}{16} \\
\frac{2}{16} \quad \frac{4}{16} \quad \frac{2}{16} \\
\frac{1}{16} \quad \frac{2}{16} \quad \frac{1}{16}
\]

- A Gaussian kernel tends to provide gentler smoothing and preserve edges better
What Does Blurring Take Way?

Let's add it back:

\[ \text{original} - \text{smoothed} = \text{details} \]

\[ \text{original} + \alpha = ? \]
Sharpening

\[ I + \alpha(I - I \ast H) = (1 + \alpha)I - \alpha(I \ast H) \]

Example for \( \alpha = 1 \):
Sharpening Filter

$$\begin{bmatrix}
0 & 0 & 0 \\
0 & 2 & 0 \\
0 & 0 & 0
\end{bmatrix} \ast \left( \begin{bmatrix}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1
\end{bmatrix} - \frac{1}{9} \right) = \text{Sharpening}
$$

Sharpening filter (accentuates edges)
Sharpening Filter

\[(1 + \alpha)I - \alpha(I * H) = I * ([1 + \alpha]e - H)\]
Edge Detection

- Convolve with a filter that finds differences between neighbor pixels

Original  Detect edges

Filter = \[
\begin{bmatrix}
-1 & -1 & -1 \\
-1 & +8 & -1 \\
-1 & -1 & -1 \\
\end{bmatrix}
\]
**Sharpen**

- Sum detected edges with original image

Original  Sharpened

Filter = $\begin{bmatrix} -1 & -1 & -1 \\ -1 & +9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$
Sharpening + noise

(and noise enhancement!)
Separating Scales

- Small scale = details (& noise)
- Large scale = “structure”
- How can we separate the two?
  - Smooth the image by filtering
    \[ I^S = G*I \]
  - Take the difference between the two:
    \[ D = I - I^S \]
One Level

\[ I = I_S + D \]
We can continue in this manner and represent an image as a one-parameter family of smoothed images (+ their residuals)
Residual Image

- Mostly “small scale” or “high frequency”
- Interestingly, contains a lot of information
- Can contain noise but also – edges!
Edges

- Caused by a variety of factors:

  - surface normal discontinuity
  - depth discontinuity
  - surface color discontinuity
  - illumination discontinuity
Finding Edges

- Until now: Smooth and take the difference
- Characterize what an edge is?
  - Edge is a discontinuity (large change) in color
  - Color is “function value” of image so edge is large derivatives in image
Finding Edges

- Edges = discontinuity of various forms
- Function discontinuity $\rightarrow$ large derivatives
Characterizing Edges

- An edge is a place of rapid change in the image intensity function.

What is an image derivative?

edges correspond to extrema of derivative
On a 2D Grid (Image)

\[
\frac{\partial}{\partial x} f(i, j) \cong \frac{f(i, j) - f(i, j - 1)}{1} = f(i, j) - f(i, j - 1)
\]

\[
\frac{\partial}{\partial y} f(i, j) \cong \frac{f(i, j) - f(i - 1, j)}{1} = f(i, j) - f(i - 1, j)
\]
Gradient Vector

- For a differentiable function $f(x,y)$ the gradient vector at point $(x_0,y_0)$ is defined as:

$$\nabla f(x_0,y_0) = \left( \frac{\partial f}{\partial x}(x_0,y_0), \frac{\partial f}{\partial y}(x_0,y_0) \right)$$

- Or in general:

$$\nabla f = \left( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right)$$

- It can be shown that this is the direction of the most rapid increase/decrease of the function
Gradient Example

\[ z = f(x,y) = x^2 + y^2 \]
The Image Gradient Vector

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

The gradient points in the direction of most rapid increase in intensity

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

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

The edge strength is given by the gradient magnitude:

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

The gradient direction is given by:

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

Source: Steve Seitz
Derivatives as a Linear Filters

The gradient of an image is a vector:

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

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

- or -

\[ \|
\n= \left| \frac{\partial f}{\partial x} \right| + \left| \frac{\partial f}{\partial y} \right| \]
Image Derivatives and Gradient
Effects of noise

Where is the edge?

Noisy input image

\[ f(x) \]

\[ \frac{d}{dx} f(x) \]
Solution: smooth first

\[ f \] Signal

\[ h \] Kernel

\[ f \ast h \] Convolution

\[ \frac{d}{dx} (f \ast h) \] Differentiation

Sigma = 50
Associativity of Convolution

- To find edges, look for peaks in:
  \[ \frac{d}{dx} (f \ast h) \]

- Differentiation is a convolution, and convolution is associative:
  \[ \frac{d}{dx} (f \ast h) = f \ast \frac{d}{dx} h \]

- This saves us one operation!
2D Edge Detection Filters

Gaussian

\[ h_\sigma(u, v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}} \]

derivative of Gaussian (x)

\[ \frac{\partial}{\partial x} h_\sigma(u, v) \]
…Sharpening Filter

scaled impulse — Gaussian \(\approx\) Laplacian of Gaussian
DOG: Derivative of Gaussian

$x$-direction

$y$-direction
Finding Edges: Canny
1. Gaussian filter to smooth image
2. Find intensity gradients of image
Thresholding

Where is the Edge?
Non-maximum Suppression - Edge Thinning

- Check if pixel is local maximum along gradient direction

By interpolating pixels p and r
Non-maximum Suppression - Edge Thinning

1. Compare edge strength of current pixel with edge strength of pixel in positive and negative gradient directions.

2. If strength of current pixel is largest compared to other pixels in mask with same direction, value will be preserved.

3. Otherwise, value will be suppressed.
Non-maximum Suppression
Canny Edge Detector

1. Filter image with derivative of Gaussian
2. Find magnitude and orientation of gradient
3. Apply non-maximum suppression
4. Linking and thresholding (hysteresis):
   - Define two thresholds: low and high
   - Use the high threshold to start edge curves and the low threshold to continue them
Different $\sigma$

original  Canny with $\sigma = 1$  Canny with $\sigma = 2$
Quantization

- Reduce intensity resolution
  - Frame buffers have limited number of bits per pixel
  - Physical devices have limited dynamic range
Uniform Quantization

- Images with decreasing bits per pixel:
Uniform Quantization

- $P(x,y) = \text{round}(I(x,y))$

$P(x,y) - 2$ bits per pixel
Reducing effects of Quantization

- Dithering
  - Random dither
  - Ordered dither
  - Error diffusion dither

- Halftoning
  - Classical halftoning
Dithering

- Distribute errors among pixels
  - Exploit spatial integration in our eye
  - Display greater range of perceptible intensities

Original (8 bits)
Uniform Quantization (1 bit)
Floyd-Steinberg Dither (1 bit)
Random Dither

- Randomize quantization errors
  - Errors appear as noise

\[ P(x, y) = \text{trunc}(I(x, y) + \text{noise}(x, y) + 0.5) \]
Random Dither

Original (8 bits)

Uniform Quantization (1 bit)

Random Dither (1 bit)
Ordered Dither

- Pseudo-random quantization errors
  - Matrix stores pattern of thresholds

\[
D_2 = \begin{bmatrix}
3 & 1 \\
0 & 2
\end{bmatrix}
\]

For each pixel \((x,y)\)

oldpixel = \(I(x,y) + D(x \mod n, y \mod n)\)

\(P(x,y) = \text{find\_closest\_color}(\text{oldpixel})\)
Ordered Dither

- Bayer’s ordered dither matrices

\[
D_n = \begin{bmatrix}
4D_{n/2} + D_2(1,1)U_{n/2} & 4D_{n/2} + D_2(1,2)U_{n/2} \\
4D_{n/2} + D_2(2,1)U_{n/2} & 4D_{n/2} + D_2(2,2)U_{n/2}
\end{bmatrix}
\]

\[
D_2 = \begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix} \quad D_4 = \begin{bmatrix} 15 & 7 & 13 & 5 \\ 3 & 11 & 1 & 9 \\ 12 & 4 & 14 & 6 \\ 0 & 8 & 2 & 10 \end{bmatrix}
\]

Idea: organize successive integers such that the average distance between two successive numbers in the map is as large as possible
Ordered Dither

- An example
  - Palette consists of 8 red tones, 8 green tones and their combinations (64 colors)
  - Original image had 19600 colors

Undithered  Dithered
Ordered Dither

Original (8 bits)

Random Dither (1 bit)

Ordered Dither (1 bit)
Error Diffusion Dither

- Spread quantization error over neighbor pixels

\[ \alpha + \beta + \gamma + \delta = 1.0 \]
Floyd-Steinberg Algorithm

for (x = 0; x < width; x++) {
    for (y = 0; y < height; y++) {
        P(x,y) = trunc(I(x,y) + 0.5)
        e = I(x,y) - P(x,y)
        I(x,y+1) += \alpha * e;
        I(x+1,y-1) += \beta * e;
        I(x+1,y) += \gamma * e;
        I(x+1,y+1) += \delta * e;
    }
}

Error Diffusion Dither

Original (8 bits)  Random Dither (1 bit)  Ordered Dither (1 bit)  Floyd-Steinberg Dither (1 bit)
More examples

<table>
<thead>
<tr>
<th>Original</th>
<th>Threshold</th>
<th>Random</th>
<th>Bayer</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Original" /></td>
<td><img src="image2" alt="Threshold" /></td>
<td><img src="image3" alt="Random" /></td>
<td><img src="image4" alt="Bayer" /></td>
</tr>
<tr>
<td>Floyd-Steinberg</td>
<td>Jarvice, Judice &amp; Ninke</td>
<td>Stucki</td>
<td>Burkes</td>
</tr>
</tbody>
</table>
Classical Halftoning

- Use dots of varying size to represent intensities
  - Area of dots proportional to intensity in image

\[ I(x,y) \]

\[ P(x,y) \]
Halftone patterns

- Use cluster of pixels to represent intensity
  - Trade spatial resolution for intensity resolution

Figure 14.37 from H&B
Halftone patterns

- How many intensities in a $n \times n$ cluster?

![Halftone patterns diagram](image)

Figure 14.37 from H&B
Classical Halftoning

Newspaper Image

From New York Times, 9/21/99