Reading

What is an image?

We can think of an image as a function, $f$, from $\mathbb{R}^2$ to $\mathbb{R}$:

- $f(x, y)$ gives the intensity of a channel at position $(x, y)$
- Realistically, we expect the image only to be defined over a rectangle, with a finite range:
  - $f: [a, b] \times [c, d] \to [0,1]$

A color image is just three functions pasted together. We can write this as a “vector-valued” function:

$$f(x, y) = \begin{bmatrix} r(x, y) \\ g(x, y) \\ b(x, y) \end{bmatrix}$$
Show example in scanalyze
Images as functions
What is a digital image?

In computer graphics, we usually operate on digital (discrete) images:

- **Sample** the space on a regular grid
- **Quantize** each sample (round to nearest integer)

If our samples are $\Delta$ apart, we can write this as:

$$ f[i,j] = \text{Quantize}\{ f(i \Delta, j \Delta) \} $$

\[\text{values between 0-255}\]
Image processing

An image processing operation typically defines a new image $g$ in terms of an existing image $f$.

The simplest operations are those that transform each pixel in isolation. These pixel-to-pixel operations can be written:

$$g(x,y) = t(f(x,y))$$

Examples: threshold, RGB $\rightarrow$ grayscale

Note: a typical choice for mapping to grayscale is to apply the YIQ television matrix and keep the $Y$.

$$[Y] = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{bmatrix} [R] [G] [B]$$

Note: gradients can be computed on $Y$
Let’s Enhance!
Noise

Image processing is also useful for noise reduction and edge enhancement. We will focus on these applications for the remainder of the lecture…

Common types of noise:

- **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
Average 2
Input #2
Average 4
Average 2
Ideal noise reduction
Ideal noise reduction
Why not just do that?

- People move
- Estimate motion before averaging
- Optical Flow
- Etc.
Practical noise reduction

How can we “smooth” away noise in a single image?

Filters

1) Mean = average
2) Gaussian
3) Median

Convolution

\[
I(x) \\
\downarrow \\
x_{-1} x_0 x_1 \\
\rightarrow x
\]

Is there a more abstract way to represent this sort of operation? Of course there is!

Yes, convolution.
Discrete convolution

One of the most common methods for filtering an image is called **discrete convolution**. (We will just call this “convolution” from here on.)

In 1D, convolution is defined as:

\[
g[n] = f[n] * h[n]
\]

\[
= \sum_{n'} f[n'] h[n - n']
\]

\[
= \sum_{n'} f[n'] \tilde{h}[n' - n]
\]

where \(\tilde{h}[n] = h[-n]\).

- filter \(h\) gets flipped
- if \(h\) symmetric \(\tilde{h} = h\)
\[ q(n) = \sum_{n'} f(n') h(n-n') = \sum_{n'} f(n) \widehat{h}(n'-n) \]  

\[ f: 0 1 0 \]
\[ h: 1 -1 0 \]
\[ \widehat{h} = 0 -1 1 \]

\[ f = (g \ast h) = (f \circ g) \ast h \]

We're doing flipping to preserve associativity.

Conv. w. impulse response (or delta func) outputs the function always go forward in time.

Why? Important for combining filters, say derivative-smoothing.
Since \( f \) and \( h \) are defined over finite regions, we can write them out in two-dimensional arrays:

<table>
<thead>
<tr>
<th>128</th>
<th>54</th>
<th>9</th>
<th>78</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>145</td>
<td>98</td>
<td>240</td>
<td>233</td>
<td>86</td>
</tr>
<tr>
<td>89</td>
<td>177</td>
<td>246</td>
<td>228</td>
<td>127</td>
</tr>
<tr>
<td>67</td>
<td>90</td>
<td>255</td>
<td>237</td>
<td>95</td>
</tr>
<tr>
<td>106</td>
<td>111</td>
<td>128</td>
<td>167</td>
<td>20</td>
</tr>
<tr>
<td>221</td>
<td>154</td>
<td>97</td>
<td>123</td>
<td>0</td>
</tr>
</tbody>
</table>

Note: This is not matrix multiplication!

Q: What happens at the boundary of the image?
Boundary conditions

Reflection
Circular
Black

Chop the image
Ignore the filter on the sides
Use the image to find similar patches
Find many similar patches and average them
Photoshop example
Some properties of discrete convolution

One can show that convolution has some convenient properties. Given functions $a$, $b$, $c$:

$$a * b = b * a$$

$$(a * b) * c = a * (b * c)$$

$$a * (b + c) = a * b + a * c$$

We’ll make use of these properties later…
Convolution in 2D

In two dimensions, convolution becomes:

\[ g[n,m] = f[n,m] * h[n,m] \]
\[ = \sum_{m'} \sum_{n'} f[n',m'] h[n-n',m-m'] \]
\[ = \sum_{m'} \sum_{n'} f[n',m'] \tilde{h}[n'-n,m'-m] \]

where \( \tilde{h}[n,m] = h[-n,-m] \).
\[ f(n+1) - f(n) \]

\[
\begin{pmatrix}
1 & -1 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{pmatrix}
\]
Mean filters

How can we represent our noise-reducing averaging as a convolution filter (know as a **mean filter**)?

\[
\begin{bmatrix}
\h_{ij}
\end{bmatrix}
= \frac{1}{9}
\begin{bmatrix}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1
\end{bmatrix}
\]

**general case**

\[
h_{avg} = \frac{1}{nm \left( \begin{bmatrix} 1 \end{bmatrix} \right)} \\sum h_{ij} = 1
\]

\(n=\# cols\)
\(m=\# rows\)
Effect of mean filters

Gaussian noise

Salt and pepper noise

3x3

5x5

7x7

average filter does not work well for salt and pepper noise
Gaussian filters

Gaussian filters weigh pixels based on their distance from the center of the convolution filter. In particular:

\[ h[n,m] = \frac{e^{-(n^2+m^2)/(2\sigma^2)}}{C} \]

This does a decent job of blurring noise while preserving features of the image.

What parameter controls the width of the Gaussian? 

What happens to the image as the Gaussian filter kernel gets wider? gets blurrier

What is the constant \( C \)? What should we set it to?

\[
\text{normalization} \quad C = \sum e^{-(n^2+m^2)/2\sigma^2} \\
\text{(sum of elements of } h = 1)\]
Effect of Gaussian filters

Gaussian noise

Salt and pepper noise

3x3

5x5

7x7

works well

slightly better on salt & pepper than average but still not great
Median filters

A median filter operates over an $m \times m$ region by selecting the median intensity in the region.

What advantage does a median filter have over a mean filter?

Is a median filter a kind of convolution?
Effect of median filters

Gaussian noise

Salt and pepper noise

3x3

5x5

7x7

Note the edges
Q: how would you apply median on color images?

- Pick a neighborhood
- Average RGB of the pixels
- Choose the closest pixel to the average
Comparison: Gaussian noise

<table>
<thead>
<tr>
<th></th>
<th>Mean</th>
<th>Gaussian</th>
<th>Median</th>
</tr>
</thead>
<tbody>
<tr>
<td>3x3</td>
<td><img src="image" alt="3x3 Mean" /></td>
<td><img src="image" alt="3x3 Gaussian" /></td>
<td><img src="image" alt="3x3 Median" /></td>
</tr>
<tr>
<td>5x5</td>
<td><img src="image" alt="5x5 Mean" /></td>
<td><img src="image" alt="5x5 Gaussian" /></td>
<td><img src="image" alt="5x5 Median" /></td>
</tr>
<tr>
<td>7x7</td>
<td><img src="image" alt="7x7 Mean" /></td>
<td><img src="image" alt="7x7 Gaussian" /></td>
<td><img src="image" alt="7x7 Median" /></td>
</tr>
</tbody>
</table>
Comparison: salt and pepper noise
Bilateral filtering

Bilateral filtering is a method to average together nearby samples only if they are similar in value.
Bilateral filtering

We can also change the filter to something “nicer” like Gaussians:

Recall that convolution looked like this:

\[ g[n] = \sum_{n'} f[n']h[n-n'] \]

Bilateral filter is similar, but includes both range and domain filtering:

\[ g[n] = \frac{1}{C} \sum_{n'} f[n']h_{\sigma_s}[n-n'] h_{\sigma_r}(f[n] - f[n']) \]

and you have to normalize as you go:

\[ C = \sum_{n'} h_{\sigma_s}[n-n'] h_{\sigma_r}(f[n] - f[n']) \]
Input

\( \sigma_r = 0.1 \)

\( \sigma_r = 0.25 \)

\( \sigma_s = 2 \)

\( \sigma_s = 6 \)

Paris, et al. SIGGRAPH course notes 2007
RGB $\rightarrow$ YIQ

Compute the grayscale version of an image:

$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = M_{RGB \rightarrow YIQ} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$

where

$M_{RGB \rightarrow YIQ} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{bmatrix}$

Our visual system essentially encodes $Y$ at high spatial resolution, and $I$ and $Q$ at low spatial resolution.
RGB image

(R,0,0) (R,R,R)

(0,G,0) (G,G,G)

(0,0,B) (B,B,B)
RGB $\rightarrow$ YIQ

\[ \begin{align*}
M_{RGB \rightarrow YIQ} &
\end{align*}\]
RGB $\rightarrow$ YIQ

$M_{RGB\rightarrow YIQ}$
RGB $\rightarrow$ YIQ $\rightarrow$ RGB

\[ M_{RGB \rightarrow YIQ} \]

\[ M_{-1} \]

Compressed loss

Compressed wave
Blurring the Y channel

RGB → Y blur → Y

I (+128) → No change → I (+128)

Q (+128) → No change → Q (+128)

RGB
Blurring the I channel

- RGB → I (+128) → No change → Y
- Q (+128) → No change → Y
- I (+128) → I blur → blurry Y
- RGB
Blurring the Q channel

- RGB
- I (+128) → No change → Y
- Q (+128) → Q blur → Y
- Q (+128) → No change → RGB
Blur comparison

INPUT

RGB

OUTPUT

RGB after Y blur

RGB after I blur

RGB after Q blur
Sharpen comparison

INPUT

RGB

OUTPUT

RGB after Y sharpen

RGB after I sharpen

RGB after Q sharpen
Edge detection

One of the most important uses of image processing is **edge detection:**

- Really easy for humans
- Really difficult for computers

- Fundamental in computer vision
- Important in many graphics applications
a.9/1 liked to detect there.
Types of edges

1D

Q: How might you detect an edge in 1D?

\[ f(x) : \left| \frac{df}{dx} \right| > \text{threshold} \]

\[ h_c = [1 \ 0 \ -1] \]
Gradients

The **gradient** is the 2D equivalent of the derivative:

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

Properties of the gradient

- It's a vector
- Points in the direction of maximum increase of \( f \)
- Magnitude is rate of increase

How can we approximate the gradient in a discrete image?

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

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

\[
\left[ \frac{\partial^2 f}{\partial y \partial x} + \frac{\partial^2 f}{\partial x \partial y} \right]
\]

\[
\frac{\partial f}{\partial x} \approx h_x \times f
\]

\[
\frac{\partial f}{\partial y} \approx h_y \times f
\]
Less than ideal edges
Steps in edge detection

Edge detection algorithms typically proceed in three or four steps:

- **Filtering**: cut down on noise
- **Enhancement**: amplify the difference between edges and non-edges
- **Detection**: use a threshold operation
- **Localization** (optional): estimate geometry of edges as 1D contours that can pass between pixels
Edge enhancement

A popular gradient filter is the **Sobel operator**:

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

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

We can then compute the magnitude of the vector \((\tilde{s}_x, \tilde{s}_y)\).

Note that these operators are conveniently "pre-flipped" for convolution, so you can directly slide these across an image without flipping first.
Results of Sobel edge detection

- **Original**
- **Smoothed**
- **Sx + 128**
- **Sy + 128**
- **Magnitude**
- **Threshold = 64**
- **Threshold = 128**

There is a tradeoff here - how to choose the threshold.

If \( \| \nabla f \| > \text{thresh} \) then \( \Rightarrow \text{edge} \)
Second derivative operators

The Sobel operator can produce thick edges. Ideally, we’re looking for infinitely thin boundaries.

An alternative approach is to look for local extrema in the first derivative: places where the change in the gradient is highest.

Q: A peak in the first derivative corresponds to what in the second derivative?

Q: How might we write this as a convolution filter?
Localization with the Laplacian

An equivalent measure of the second derivative in 2D is the Laplacian:

\[ \nabla^2 f(x, y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \]

Using the same arguments we used to compute the gradient filters, we can derive a Laplacian filter to be:

\[
\Delta = \begin{bmatrix}
0 & 1 & 0 \\
1 & -4 & 1 \\
0 & 1 & 0
\end{bmatrix}
\]

Zero crossings in a Laplacian filtered image can be used to localize edges.
Localization with the Laplacian

Original

Smoothed

Zoom in Laplacian

+128 is just for visualization of negative edges.
Sharpening with the Laplacian

Why does the sign make a difference?

How can you write the filter that makes the sharpened image?
\[ f(x) \]

\[ |f'(x)| > \text{threshold} \]

\[ f''(x) = 0 \]

\[ f(x, y) \]

\[ \nabla \cdot \nabla f = \left( \frac{\partial^2 f}{\partial x^2} \right) + \left( \frac{\partial^2 f}{\partial y^2} \right) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = \Delta^2 f \]
Summary

What you should take away from this lecture:

- The meanings of all the boldfaced terms.
- How noise reduction is done
- How discrete convolution filtering works
- The effect of mean, Gaussian, and median filters
- What an image gradient is and how it can be computed
- How edge detection is done
- What the Laplacian image is and how it is used in either edge detection or image sharpening