2D Images & Curves

Topic 5

Week 4 – Jan. 30th, 2019

Topic 4: 2D Images and Curves

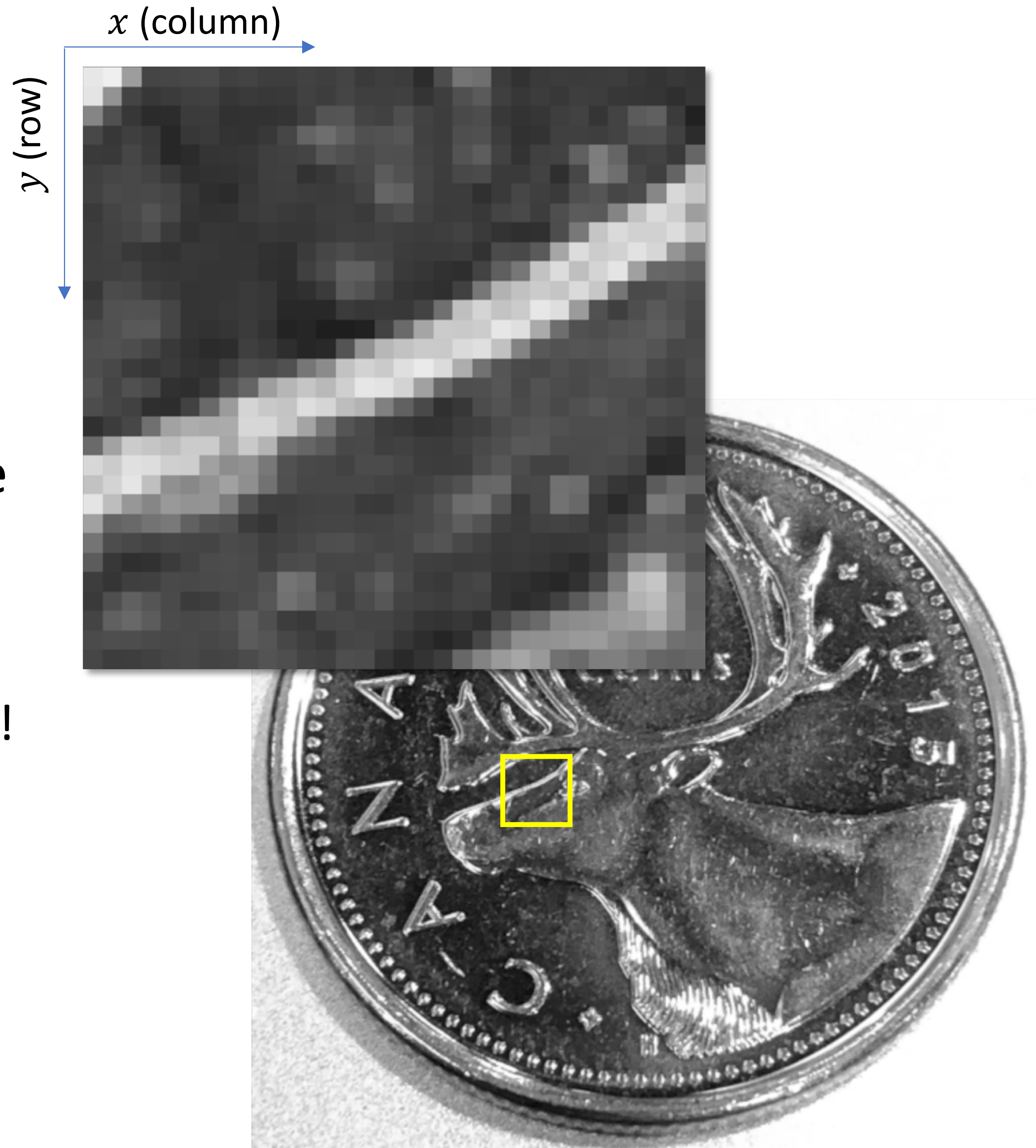
- **2D Image Patches**
- Image Gradients
- Edges
- Sobel Filter
- Second-Order Edges
- Canny Edges
- Circle Detection

Images are more than Pixels

- Up until now we have only looked at pixelwise operations
- Pixels by themselves are not overly informative!
- We need context

2D Image Patch

- An image patch (like an image) is 2D array of pixel intensities
- Pixels have row, column coordinate
- Often origin is at top left (varies)!
- How big is an image patch?
  - could be whole image, or single pixel!
  - typically small local neighbourhood



Tell me what you see

- On the next slide is an image
- Try to make note of the first thing you really look at…


BURGER KING
GER KING
Popeyes
Popeyes
LOUISIANA KITCHEN
LEFT TURN SIGNAL
TRANSIT SIGNAL
College

Street Scene

- Let’s look at just the brightness of this image (i.e. no colour)
- What is the first thing you look at in this image?
- What pixels are important?

Street Scene

- Probably one of the first things you noticed were the streetcar wires
- We are sensitive to these rapid changes brightness

2D Image Patch

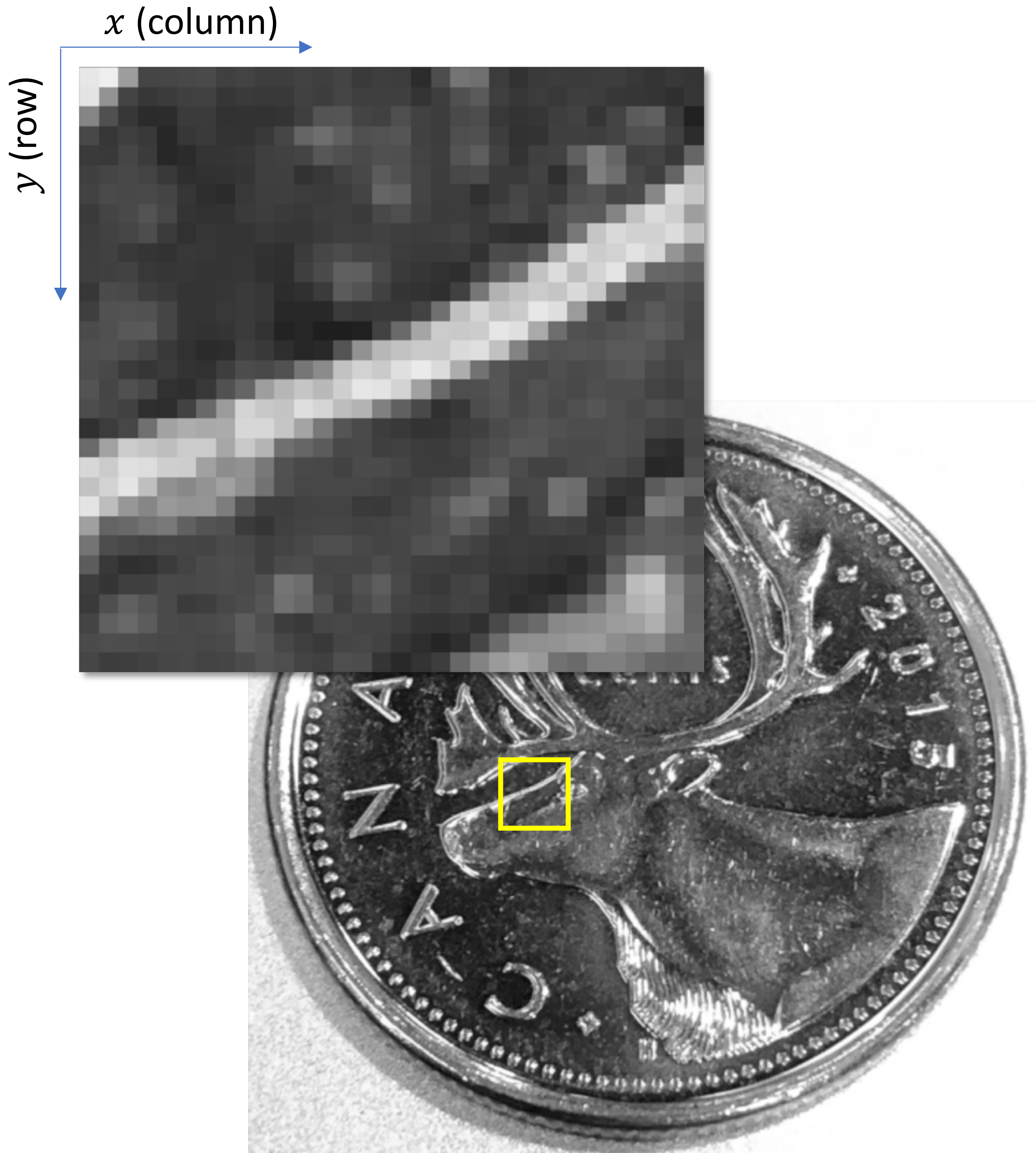
- Define the discrete function:

$$\mathbf{z} = \mathbf{I}(\mathbf{x}, \mathbf{y})$$

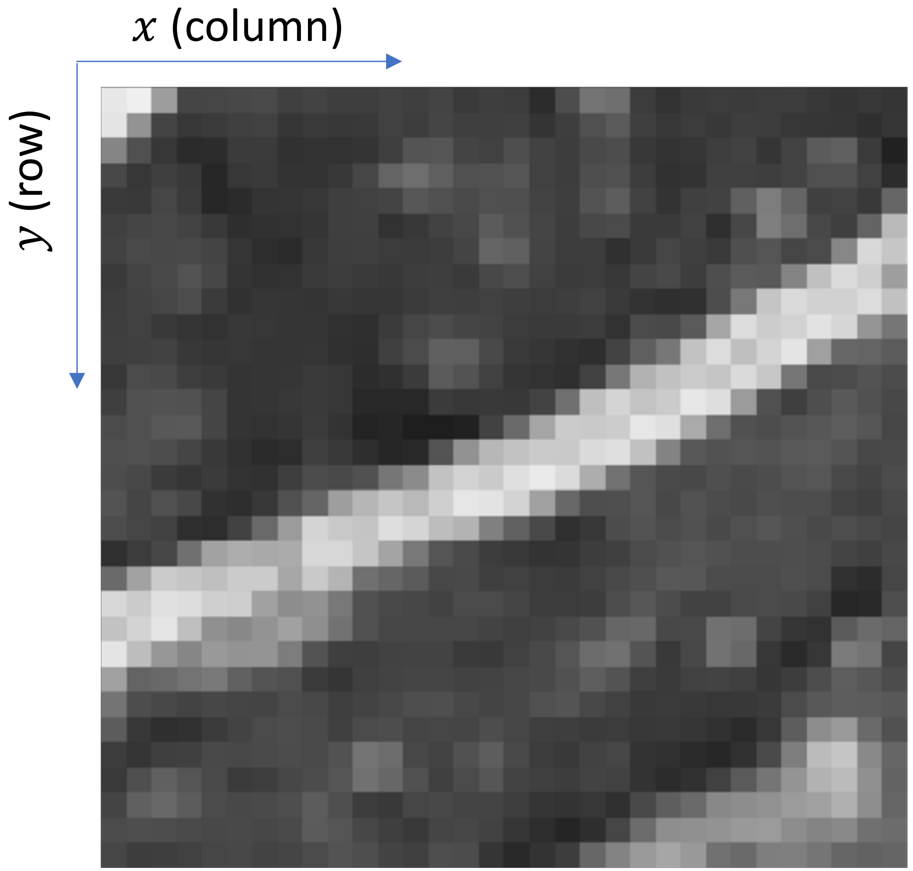
where $\mathrm{x} = 1, \ldots, W, \mathrm{y} = 1, \ldots, \mathrm{H}$ and $z \in [0, 1]$

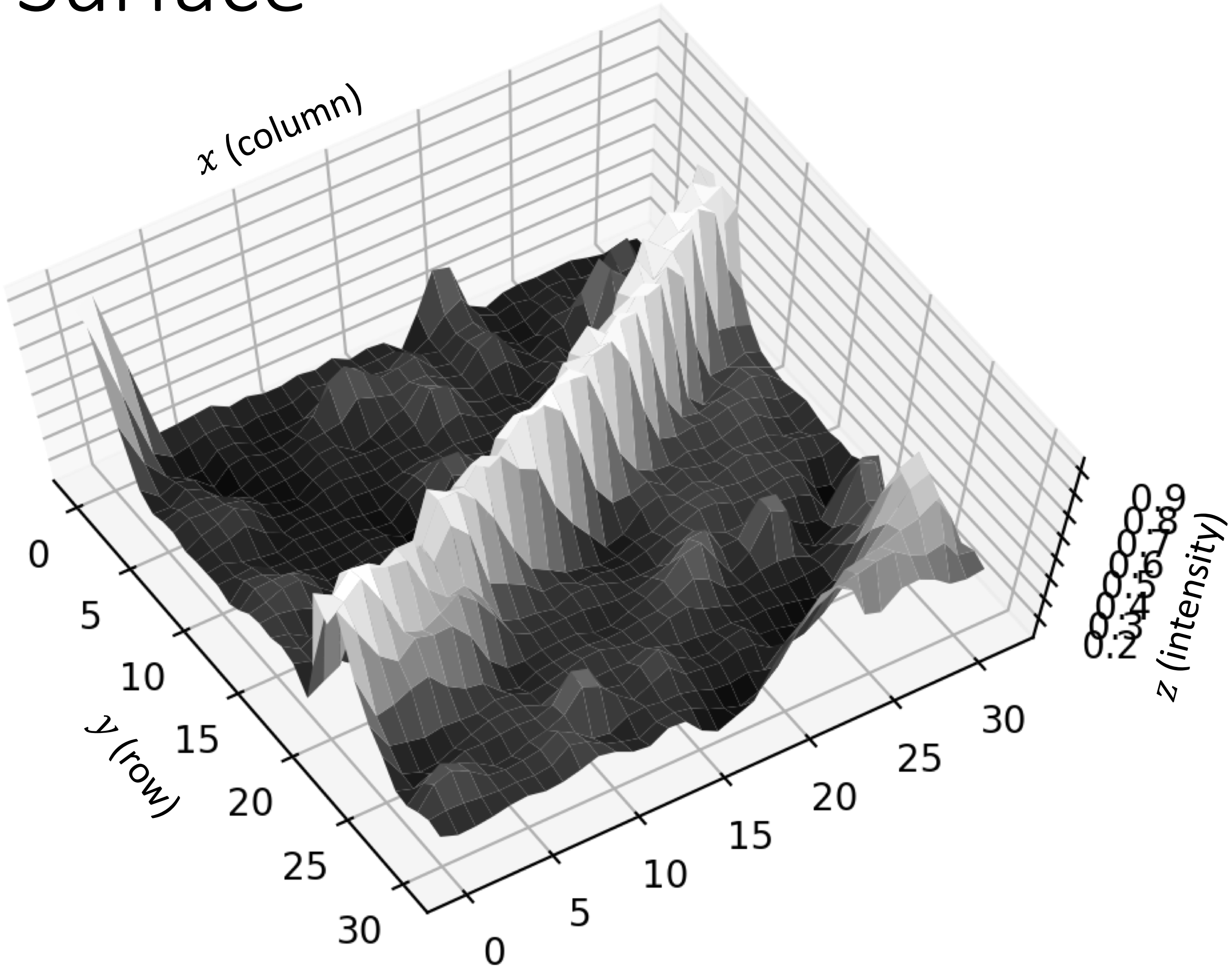
We would like to understand:

- How does this function **vary**?
- How do these intensity variations relate back to the **underlying scene**?



2D Image Patch as Surface

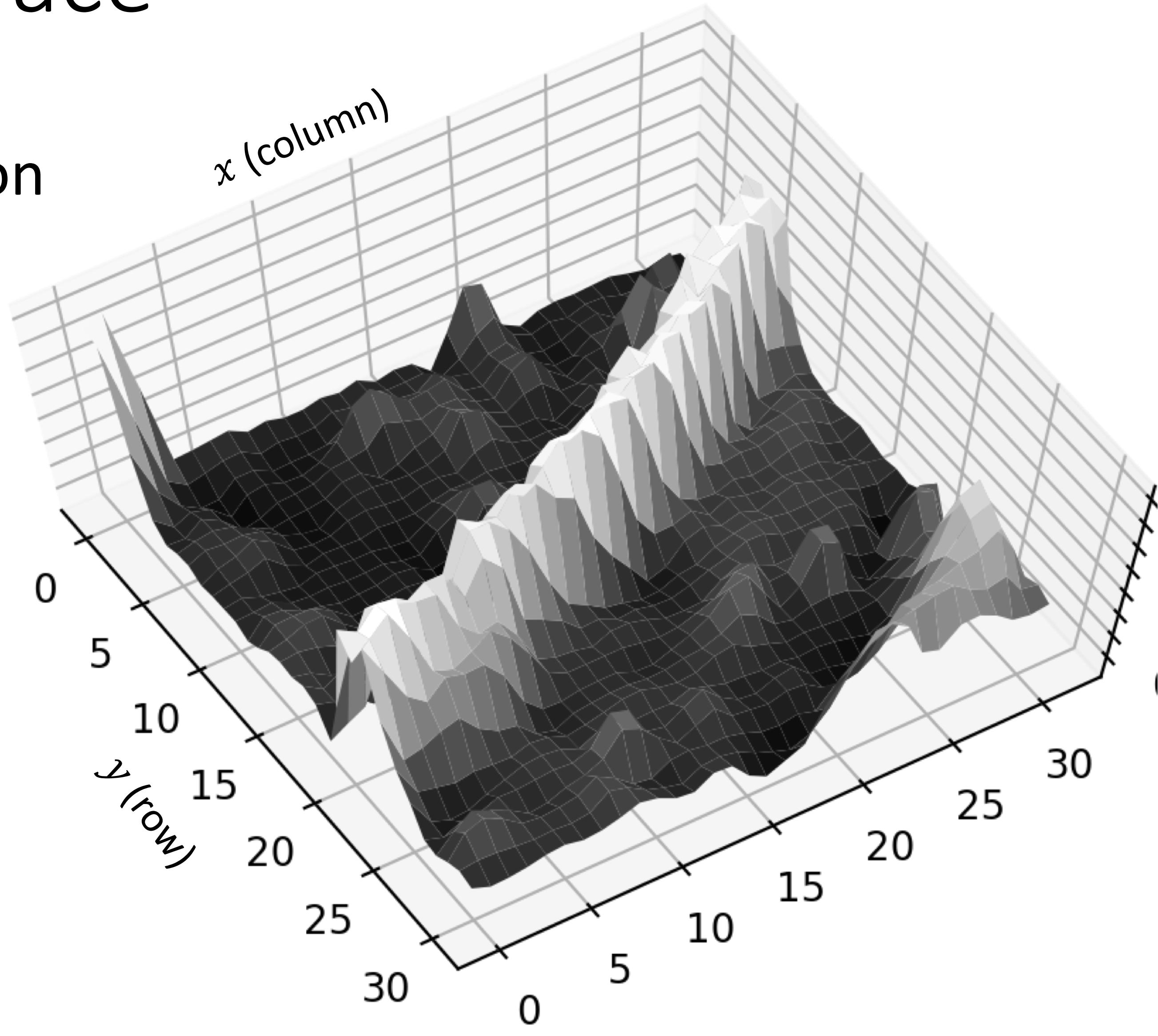




2D Image Patch as Surface

- Conceptually our intensity function $z = I(x, y)$ is a surface!
- In this context intensity is height

- How do we find the slope of the surface?

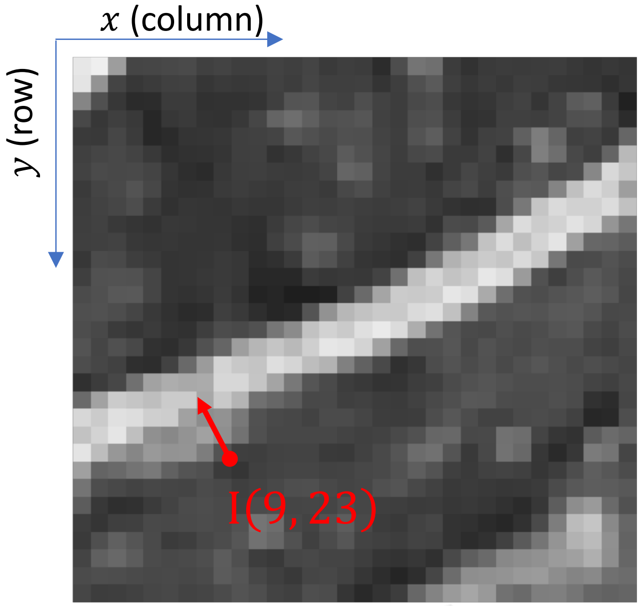


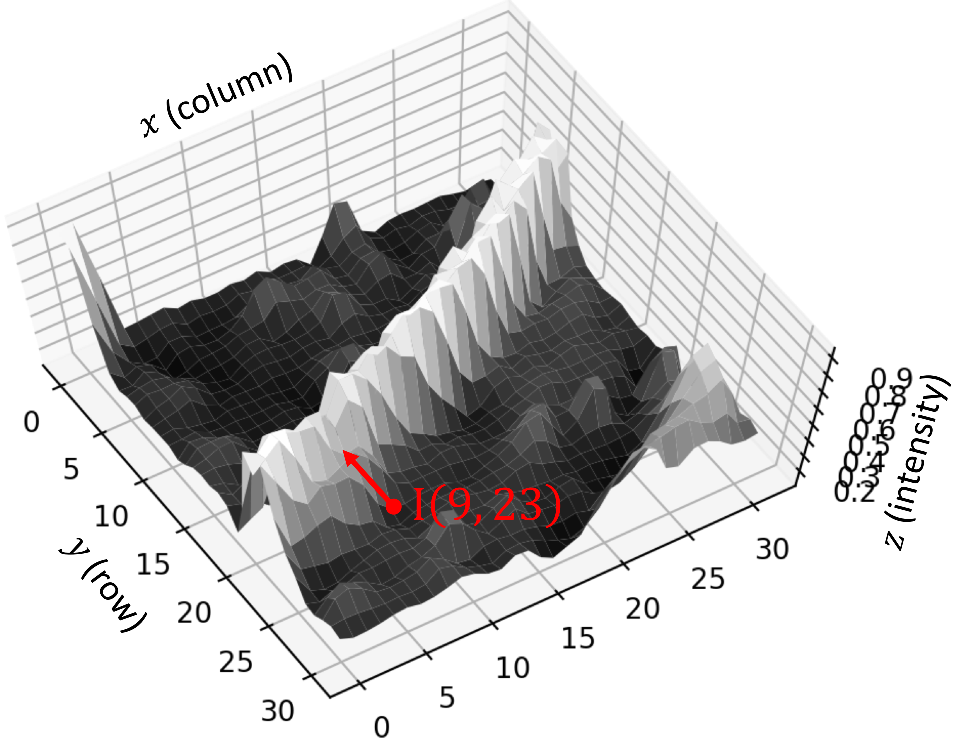
Vector Calculus Refresher

- We learned how to analyze the variation of (continuous) multivariate functions!
- Assume $f(x, y) = z$ is a 2D function, recall ∇f (**gradient** of f) is defined:

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

- ∇f tells us the direction, and magnitude of maximum increase at a given point





Topic 4: 2D Images and Curves

- 2D Image Patches
- **Image Gradients**
- Edges
- Sobel Filter
- Second-Order Edges
- Canny Edges
- Circle Detection

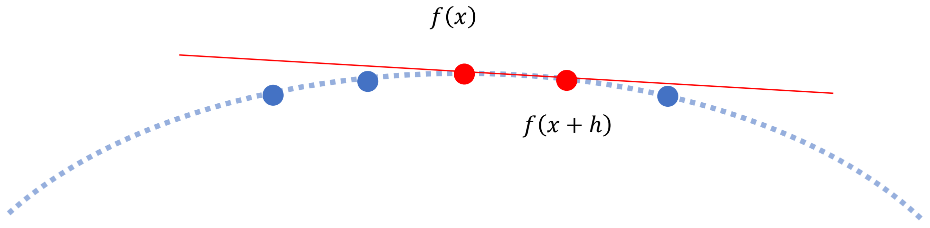
Discrete Gradients

- How do we calculate the partial derivatives of a discrete function? Let's look at 1D first:
- Recall the definition of the derivative as the limit:

$$\frac{df}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

- One option is to approximate the derivative using the smallest finite difference h:

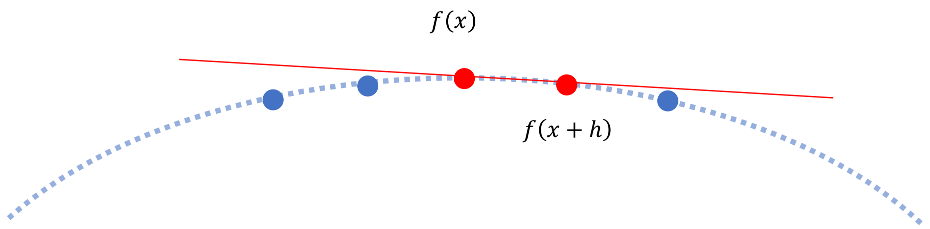
$$\frac{df}{dx} \approx \frac{f(x+h) - f(x)}{h}$$



Finite Differences

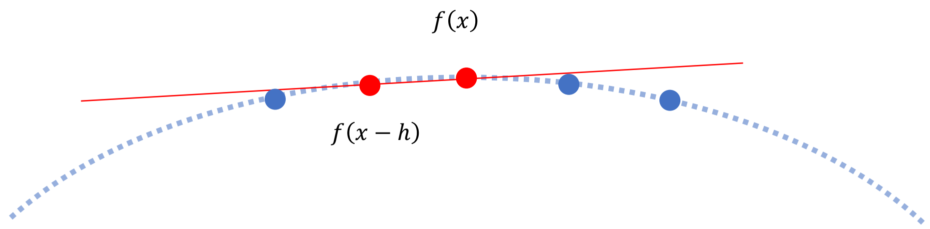
- Forward difference

$$\frac{df}{dx} \approx \frac{f(x+h) - f(x)}{h}$$



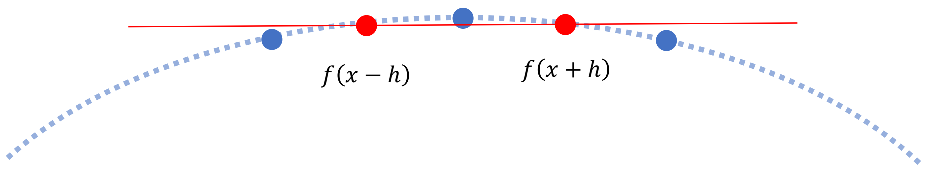
- Backward difference

$$\frac{df}{dx} \approx \frac{f(x) - f(x-h)}{h}$$



- Central difference

$$\frac{df}{dx} \approx \frac{f(x+h) - f(x-h)}{2h}$$



Discrete Image Gradients

- The gradient gives us two images, one for each partial derivative

$$\frac{\partial f}{\partial x}(x,y) \approx \frac{f(x+1,y) - f(x-1,y)}{2}$$

$$\frac{\partial f}{\partial y}(x,y) \approx \frac{f(x,y+1) - f(x,y-1)}{2}$$

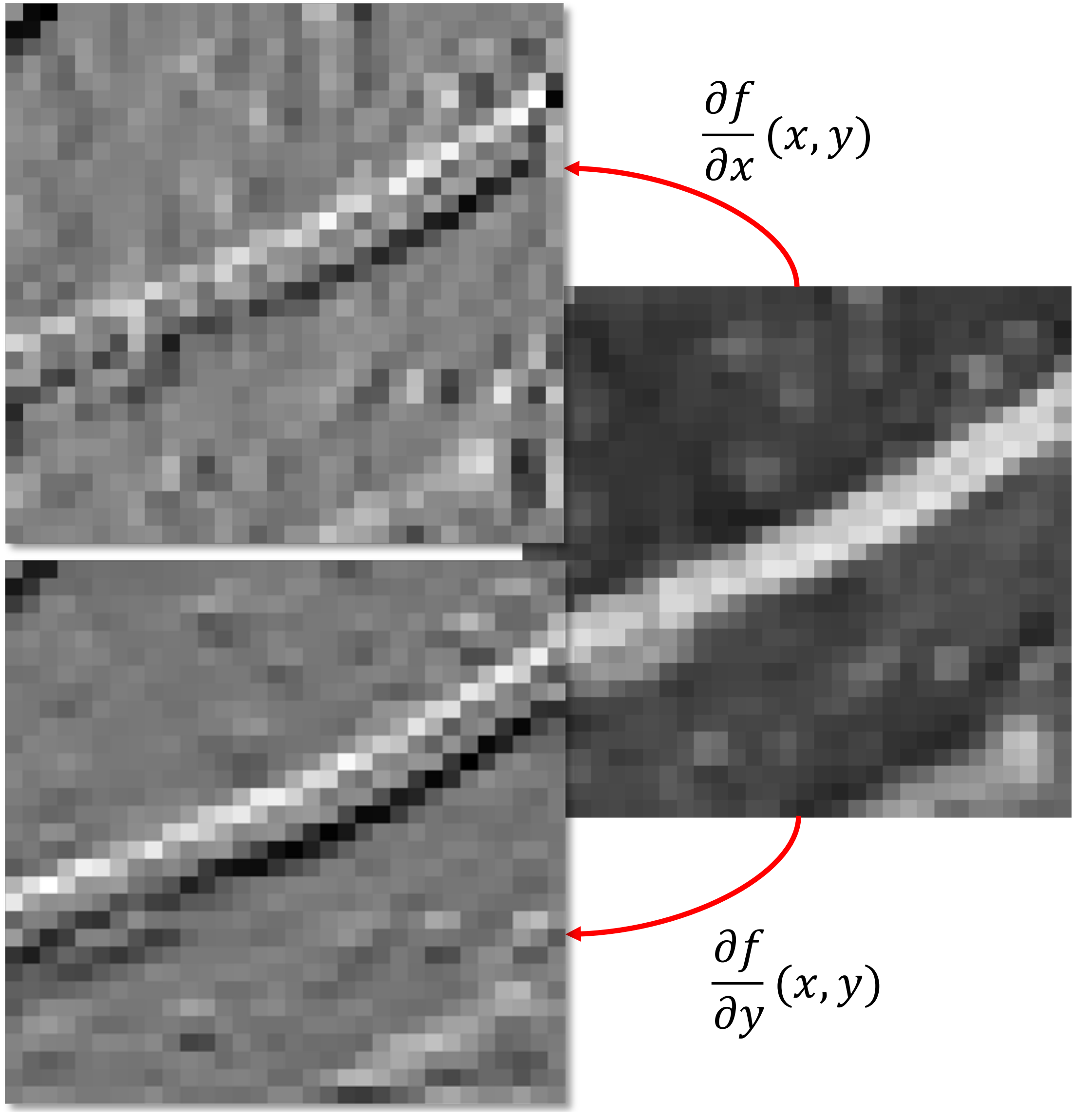
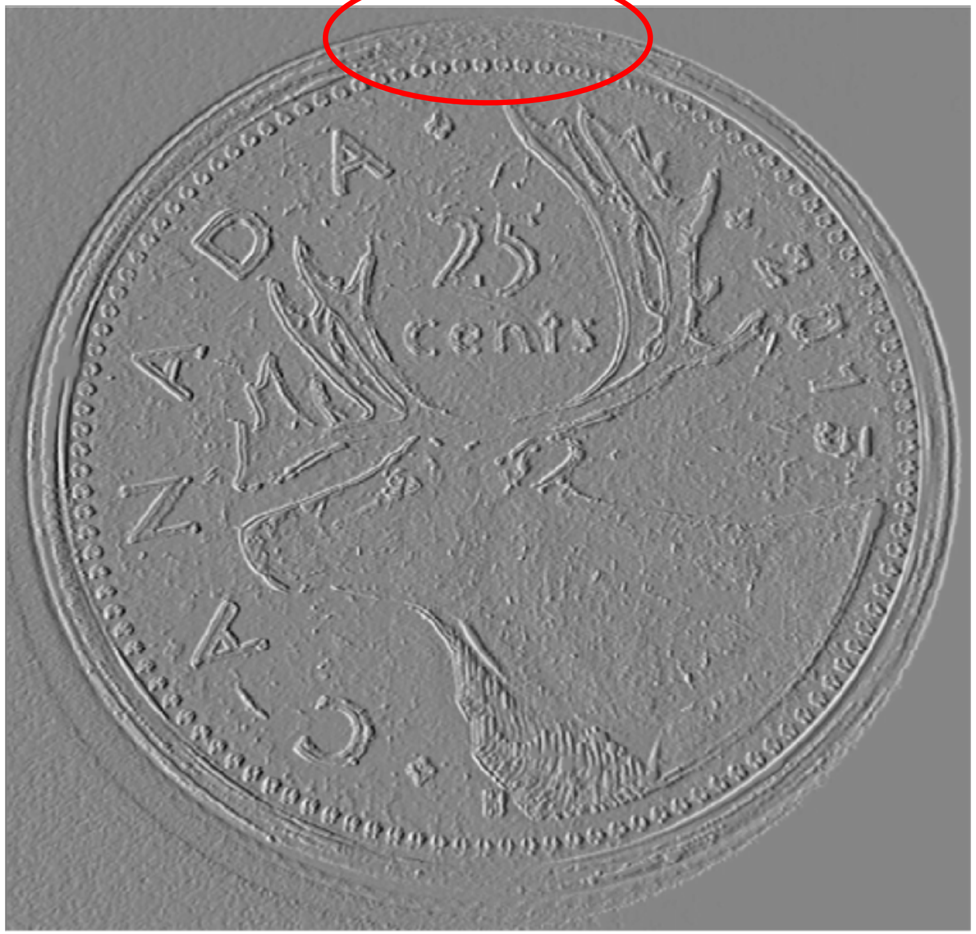
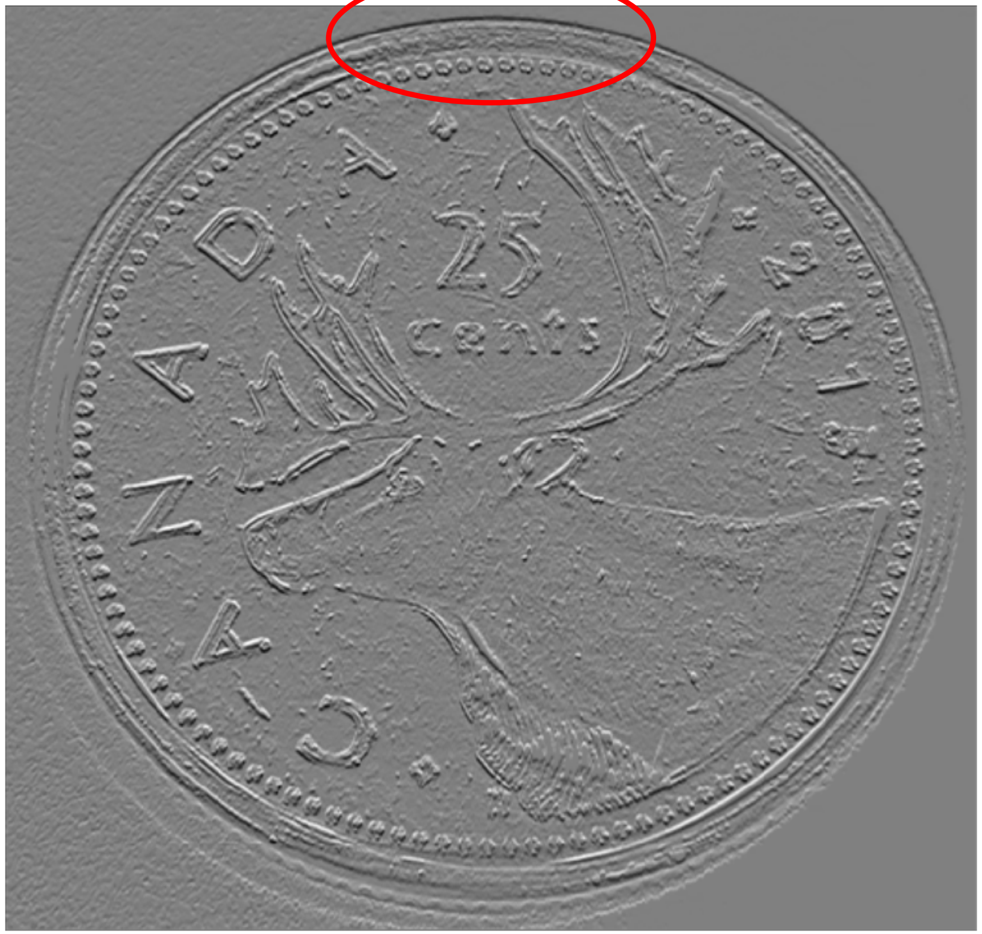


Image Gradients



$$\frac{\partial f}{\partial x}$$



$$\frac{\partial f}{\partial y}$$

Image Gradients

- Would like some measure of gradient for all components
- Since we have a vector for each pixel, can also look at the magnitude and direction of the gradient:

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

$$\theta(\mathrm{x},\mathrm{y}) = \tan^{-1}\left(\frac{\partial f}{\partial y}(x,y) / \frac{\partial f}{\partial x}(x,y)\right)$$

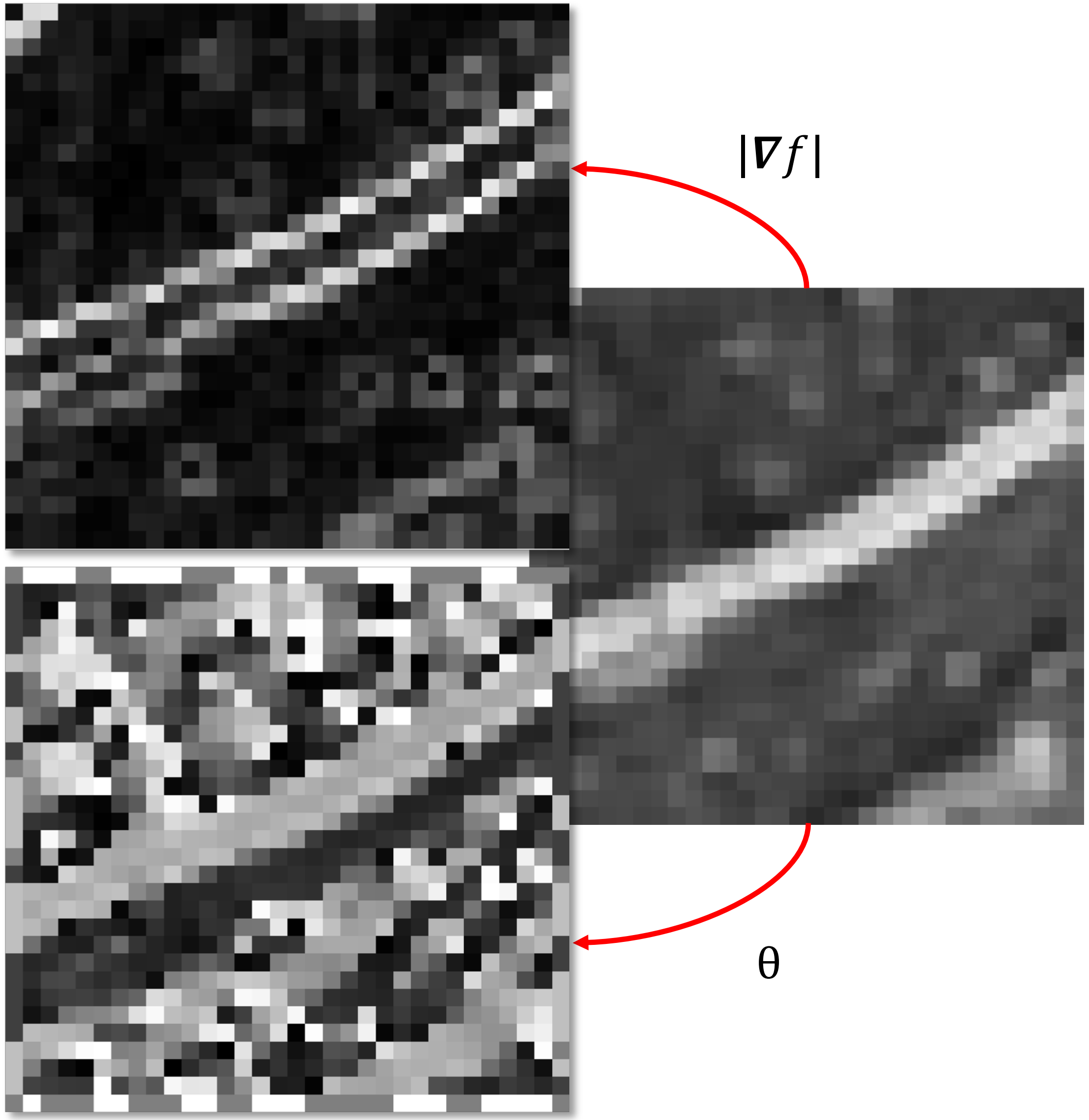


Image Gradients

$$|\nabla f(x,y)|_2 = \sqrt{\left(\frac{\partial f}{\partial x}(x,y)\right)^2 + \left(\frac{\partial f}{\partial y}(x,y)\right)^2}$$

- Square root makes this expensive
- Often instead calculated as:

$$|\nabla f(x,y)|_1 = \left|\frac{\partial f}{\partial x}(x,y)\right| + \left|\frac{\partial f}{\partial y}(x,y)\right|$$

- This is generally known as the Manhattan distance or l_1 norm

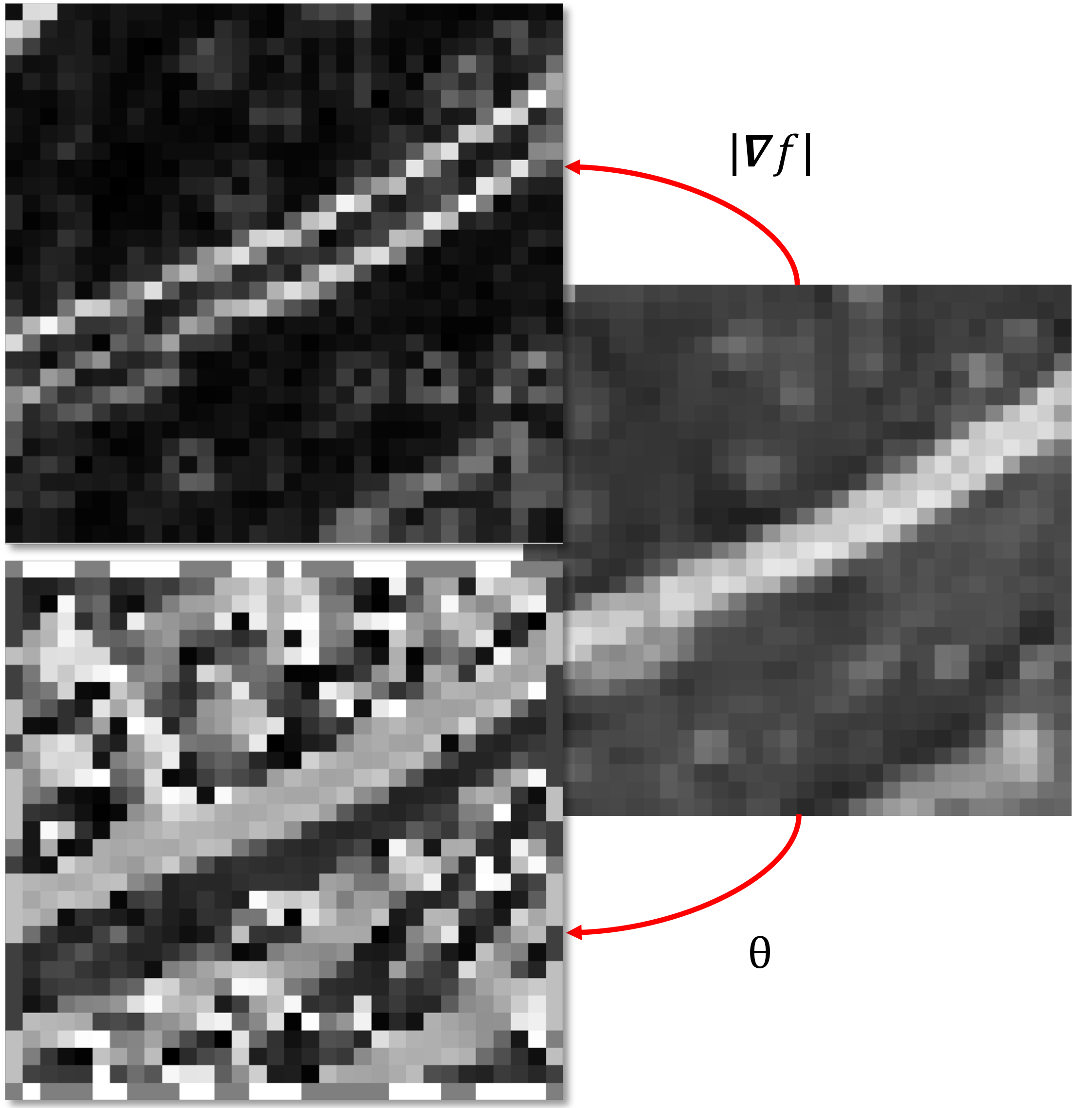
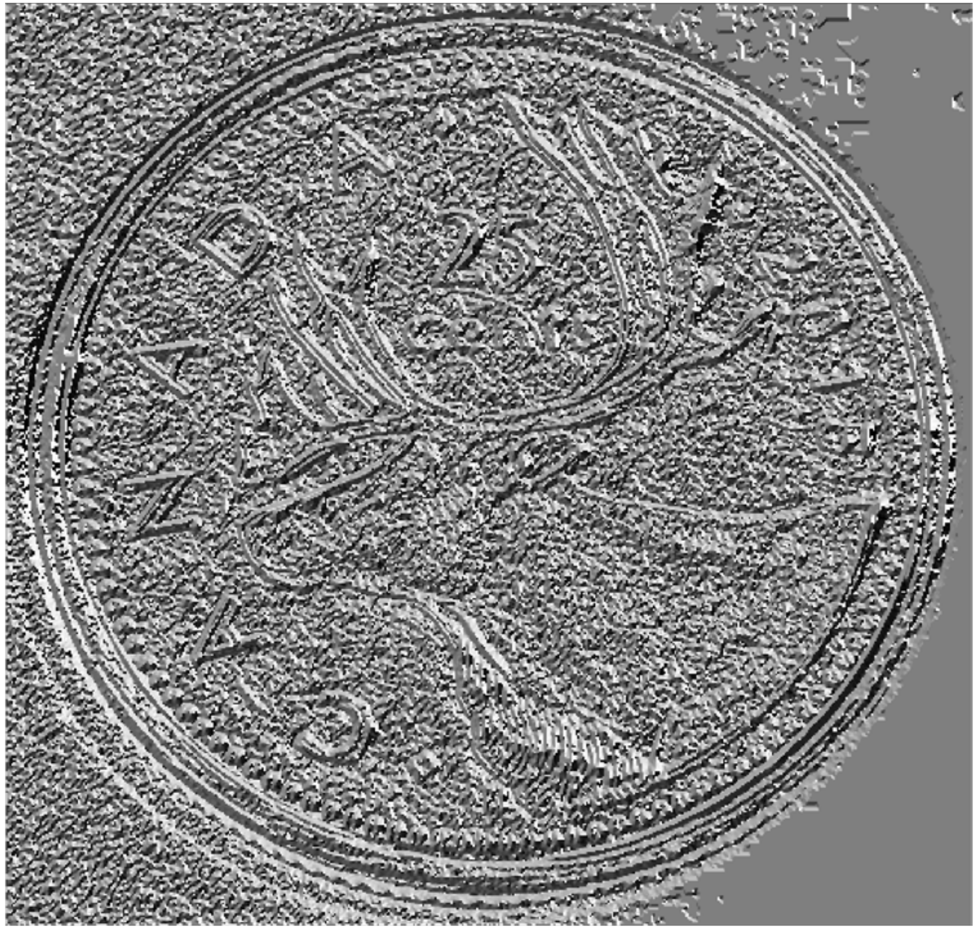


Image Gradients – Magnitude and Angle

$|\nabla f|$



θ

Image Gradients – Thresholding Magnitude

$|\boldsymbol{\nabla} f|$

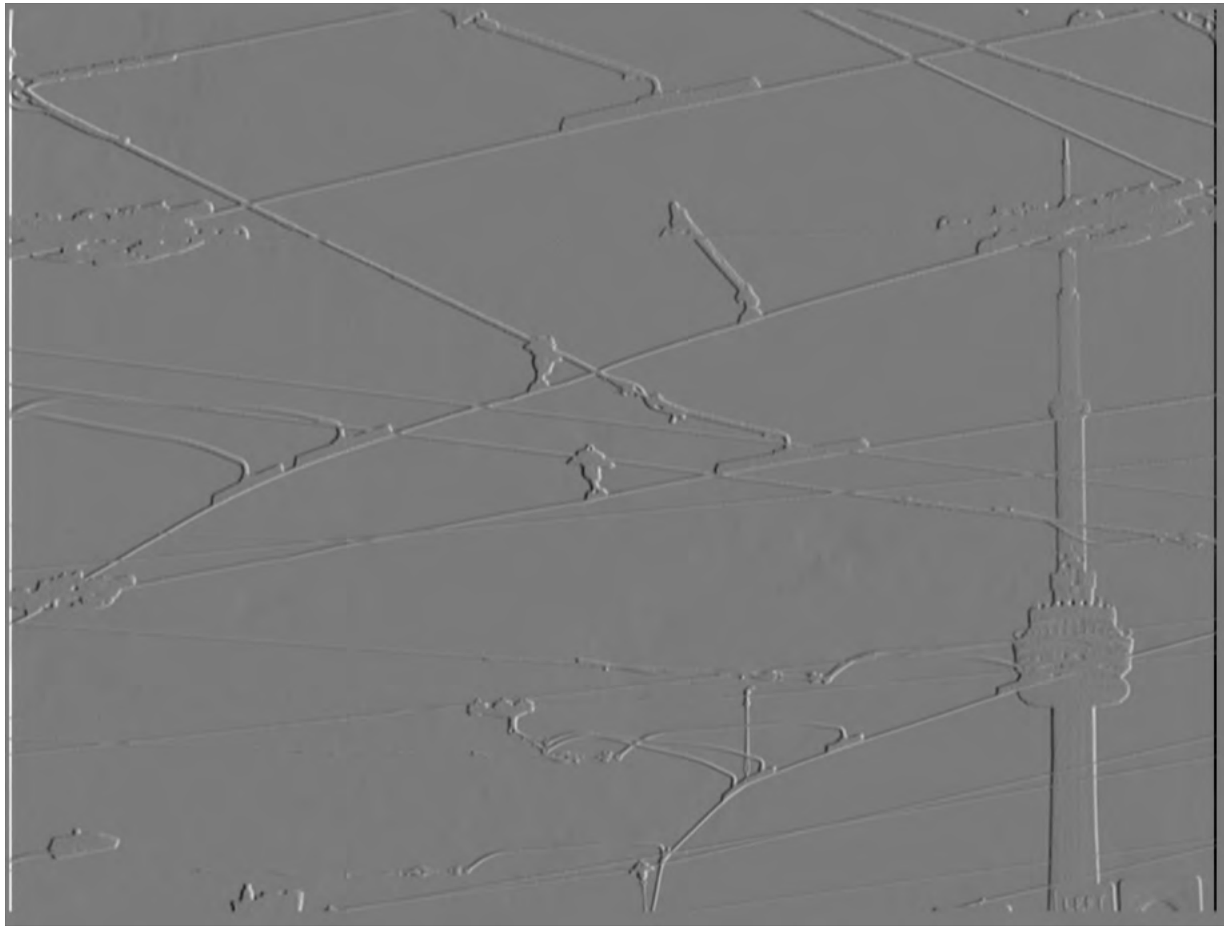
$|\boldsymbol{\nabla} f| > t$

Another Example – Street Scene

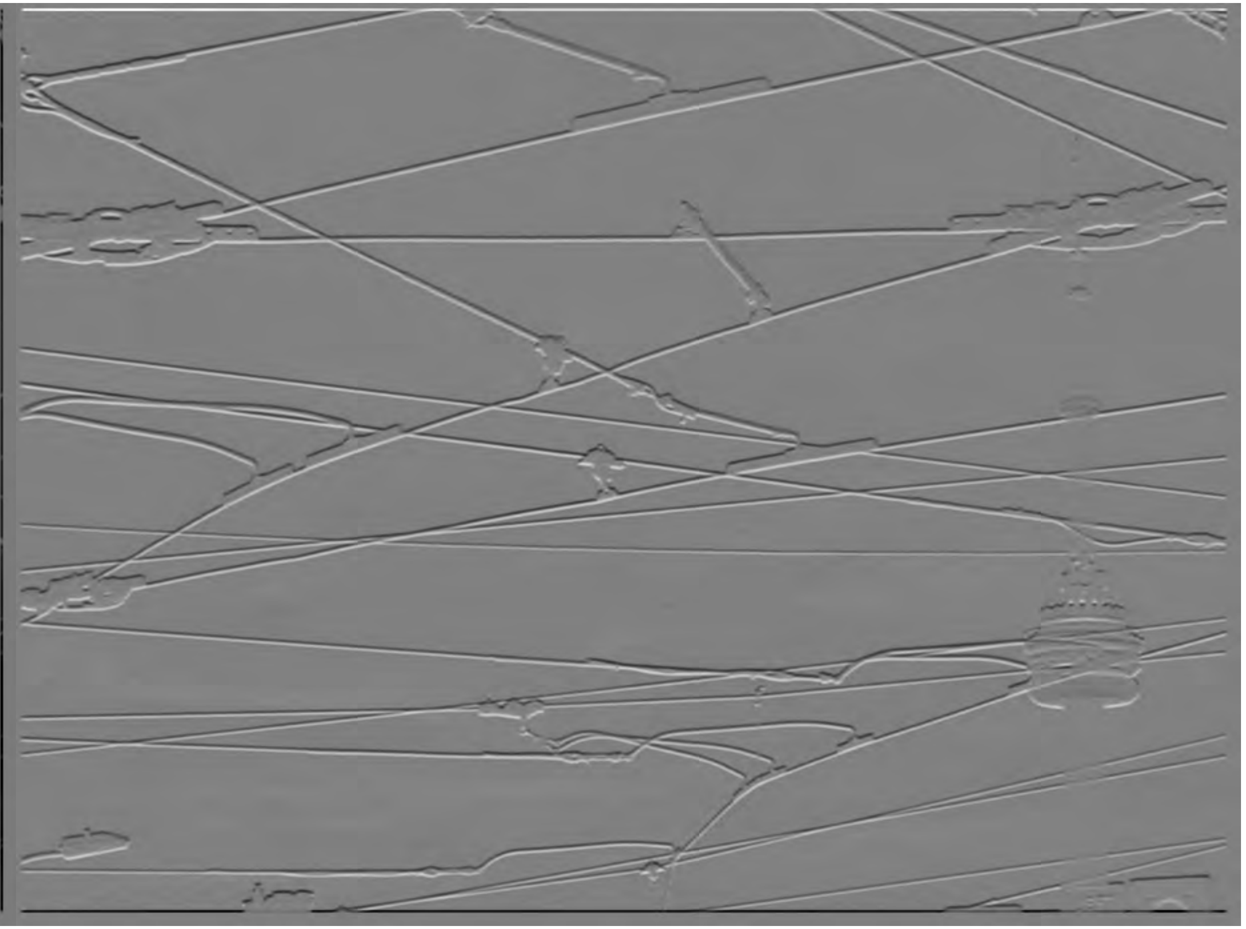
- Let’s look at a patch from a more typical scene

$\boldsymbol{f}$

Gradient – Street Scene

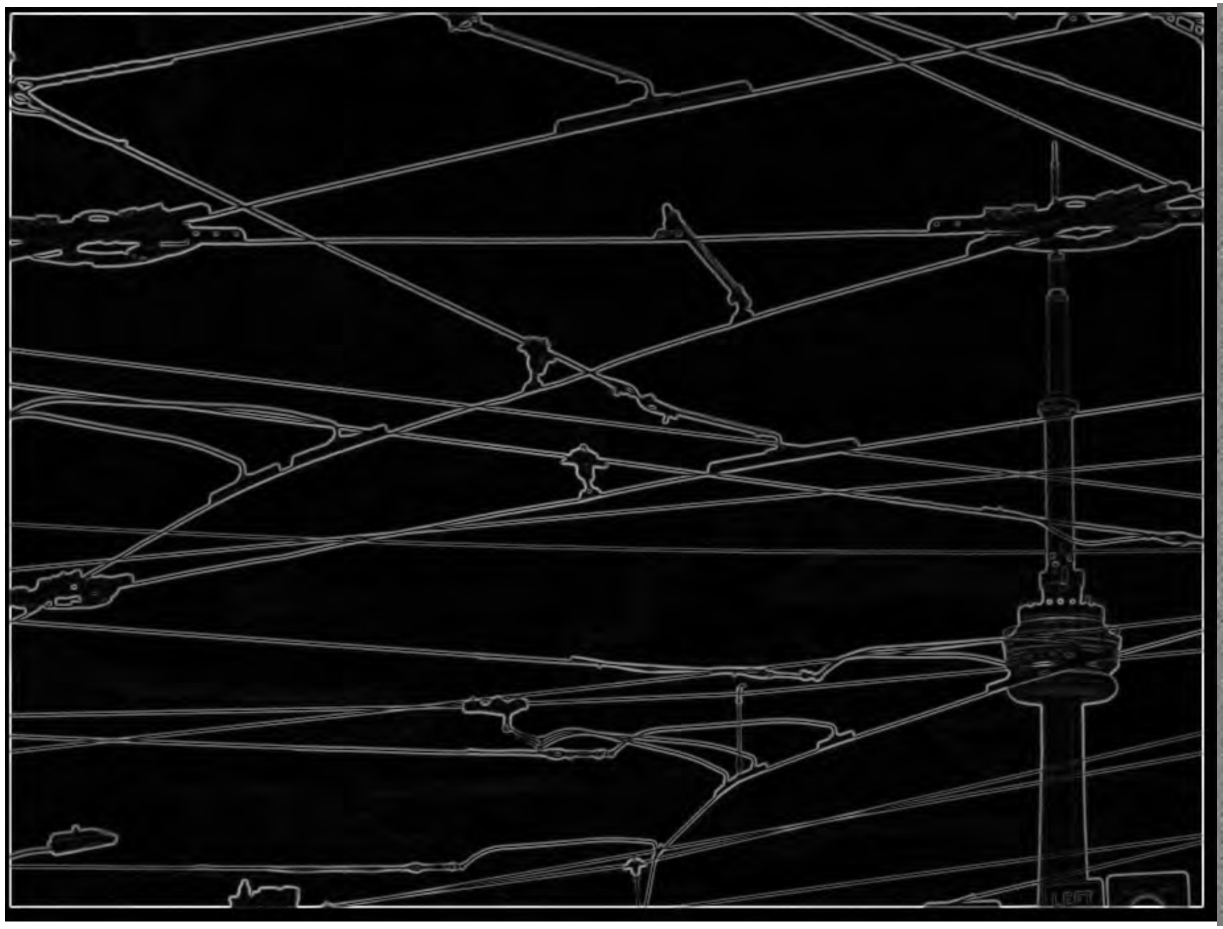


$\frac{\partial f}{\partial x}$



$\frac{\partial f}{\partial y}$

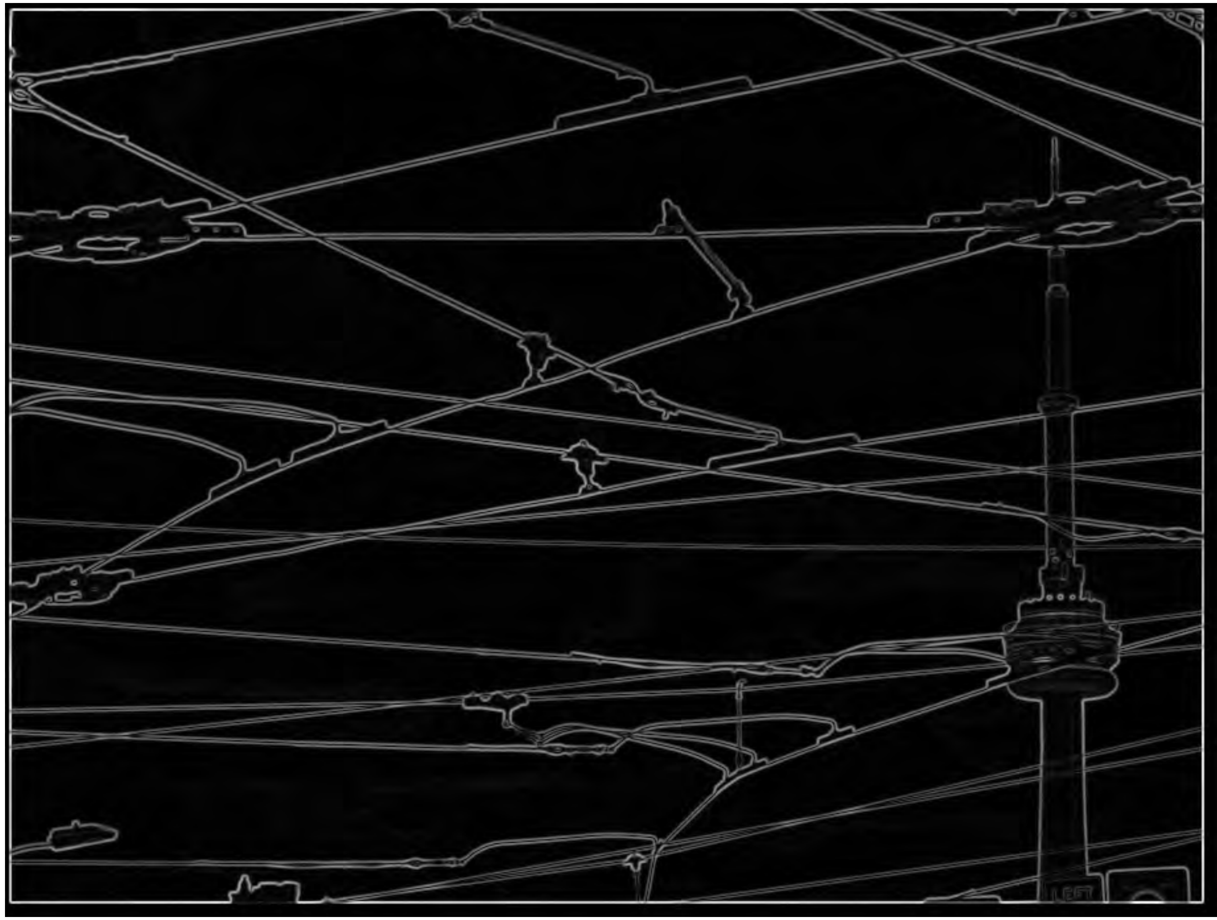
Gradient – Street Scene



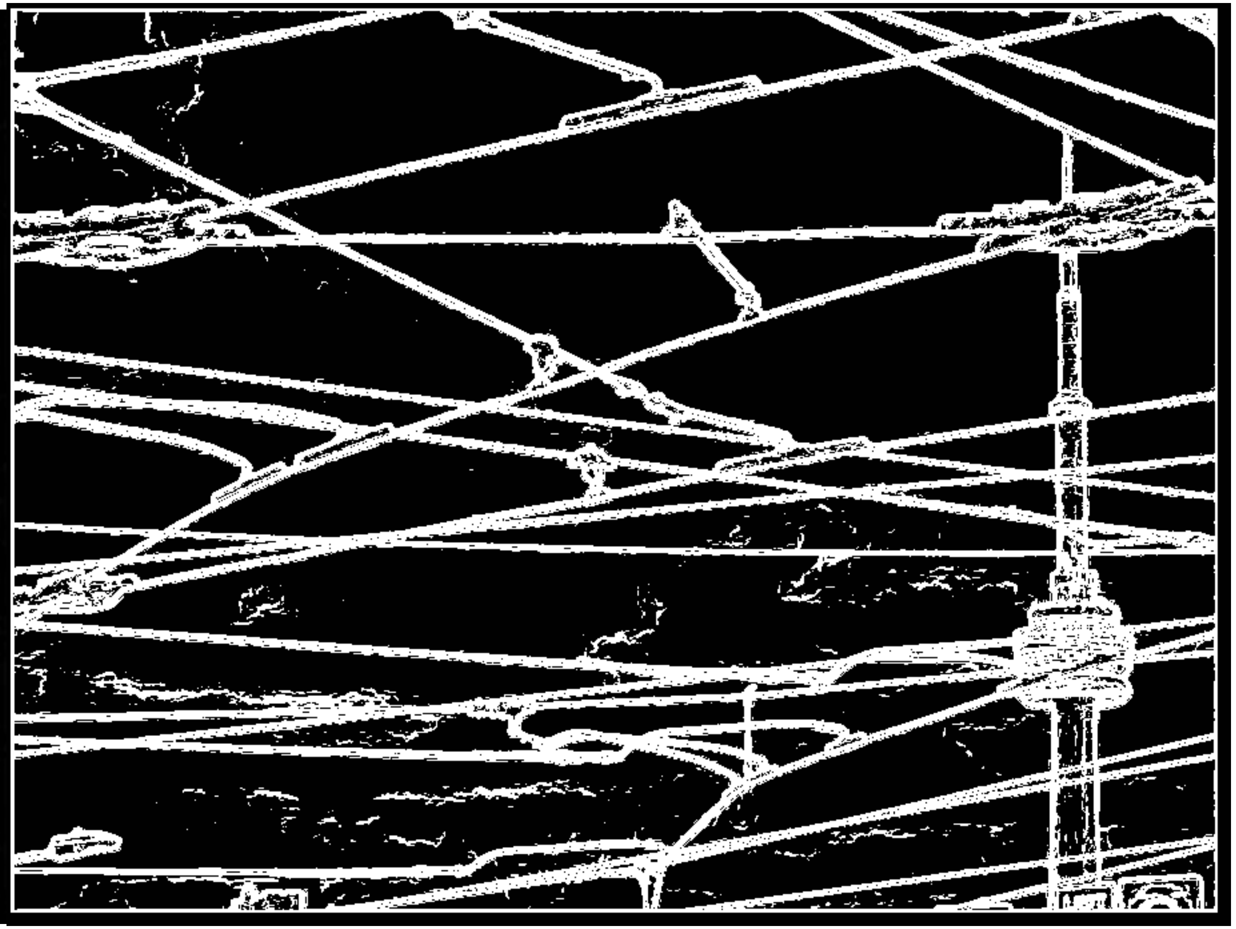
$|\nabla f|$

$\boldsymbol{\theta}$

Gradient – Street Scene



$|\boldsymbol{\nabla} f|$



$|\boldsymbol{\nabla} f| > t$

Topic 4: 2D Images and Curves

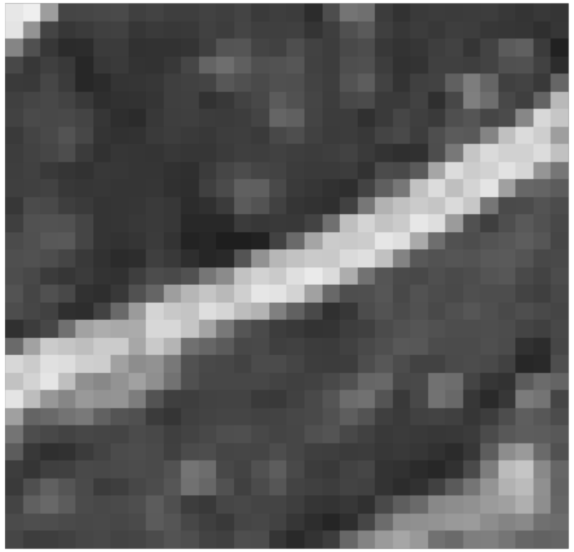
- 2D Image Patches
- Image Gradients
- **Edges**
- Sobel Filter
- Second-Order Edges
- Canny Edges
- Circle Detection

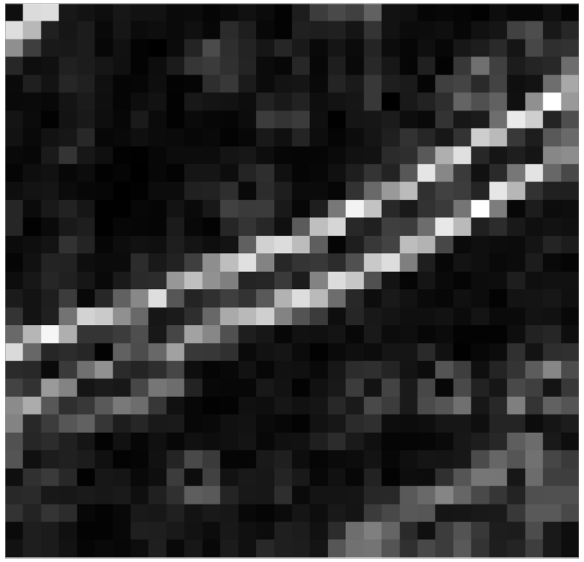
Edges

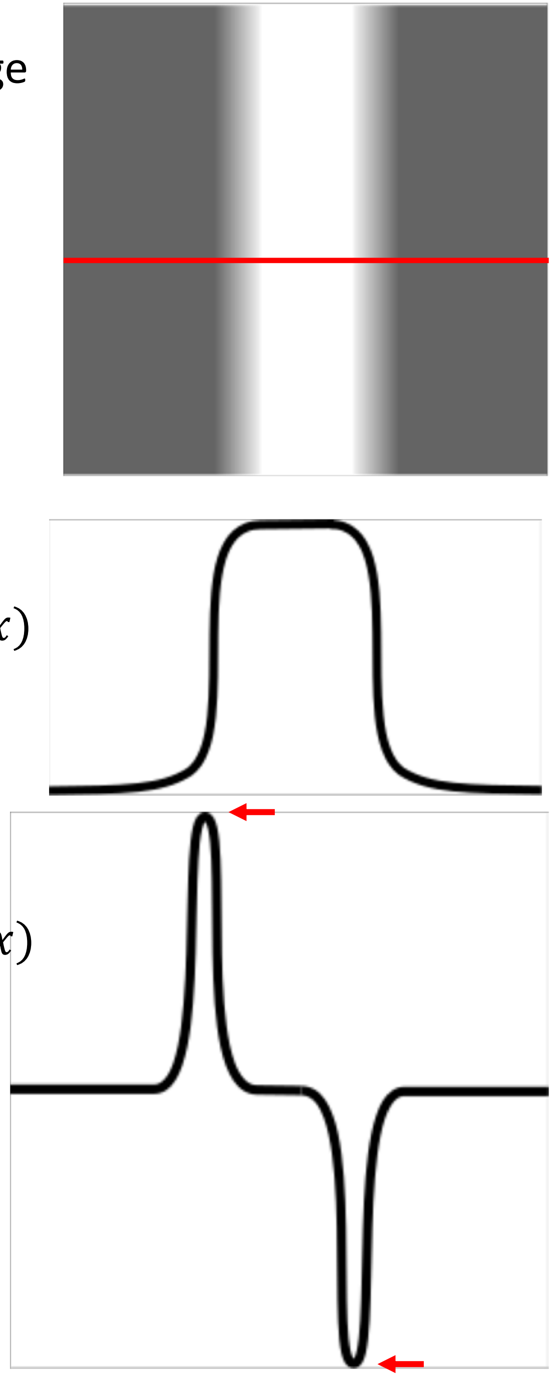
- Humans can recognize the content of this image from these edges
- This should be surprising! We've removed most of the original content of the image – 1 bit image from 8 bit!
- Edges are **salient** – i.e. they contain lots of information about the scene
- What information are edges preserving from the scene?

What is an Edge?

- Edges arise from a rapid change or discontinuity in image intensity
- The edges we find with a gradient filter, arise from the extrema of the first derivative

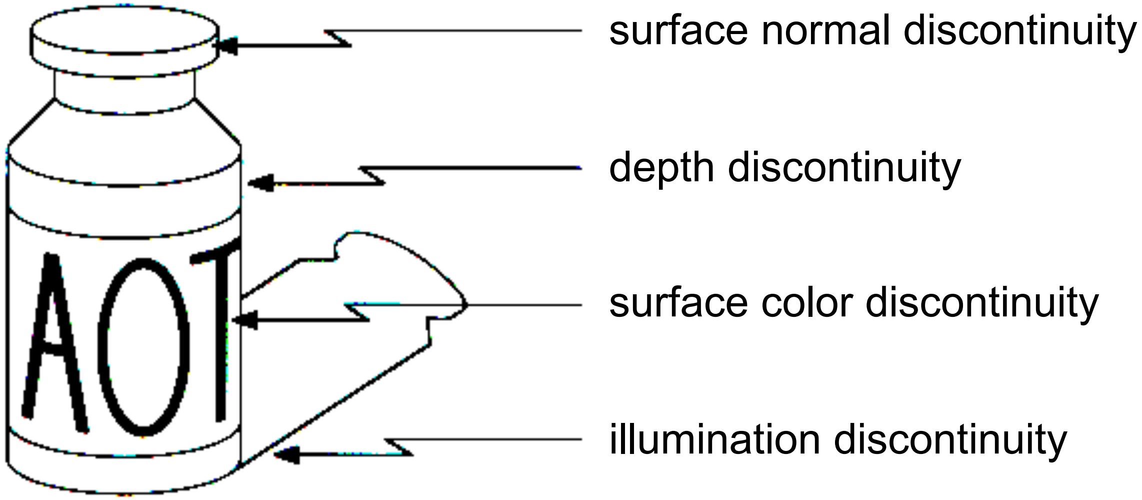




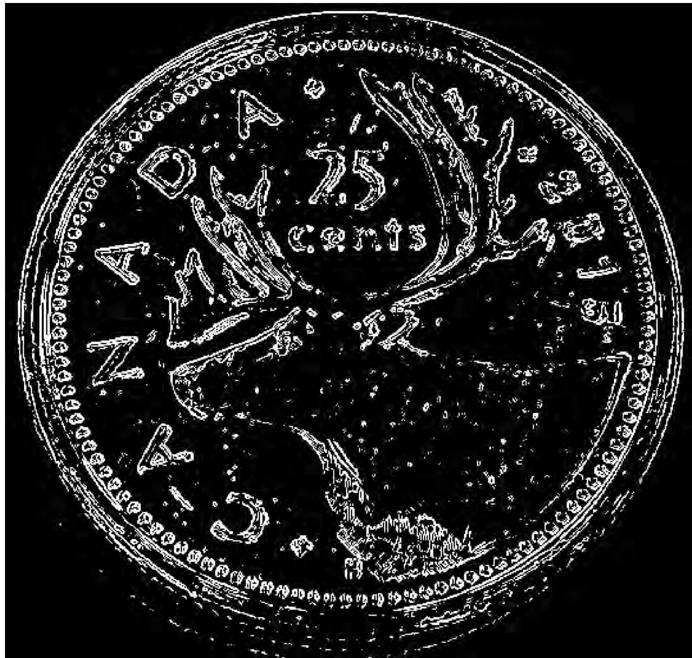


What is an Edge?

- Edges have a variety of causes!
- Each type gives us different information about the scene
- Difficult to distinguish edges from different causes



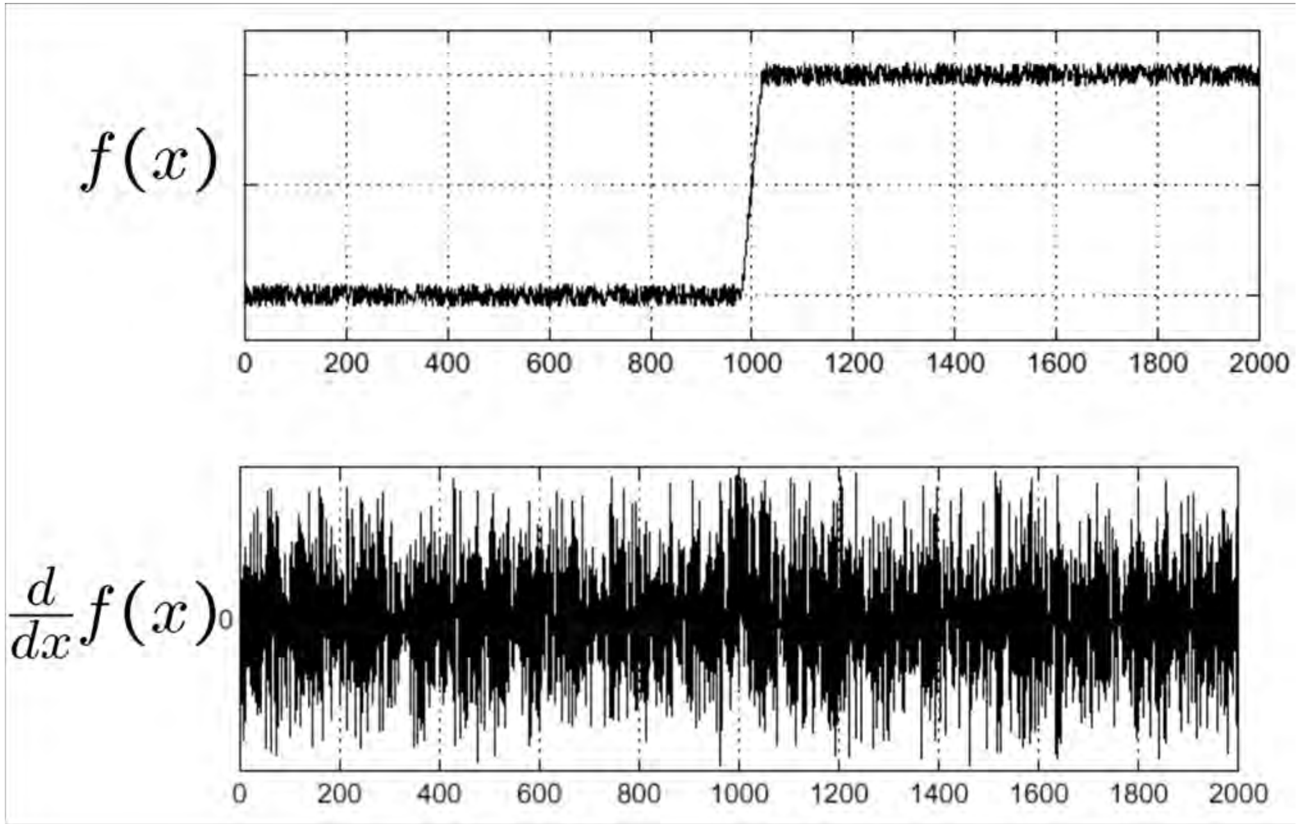
© Steve Seitz



Gradient and Noise

- In reality, our image is going to be a noisy sampling of the underlying function

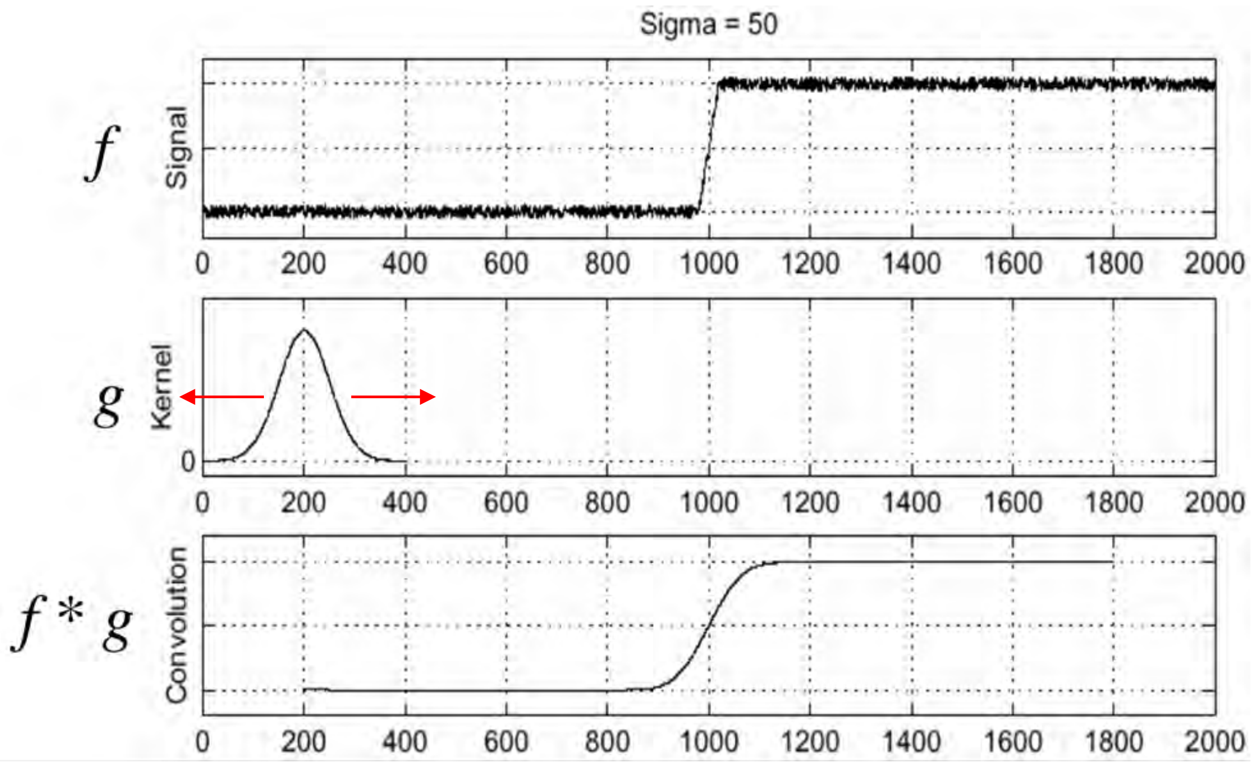
- Gradient is very sensitive to this!
    - Can find the edge in gradient? →



© Steve Seitz

Gradient and Noise

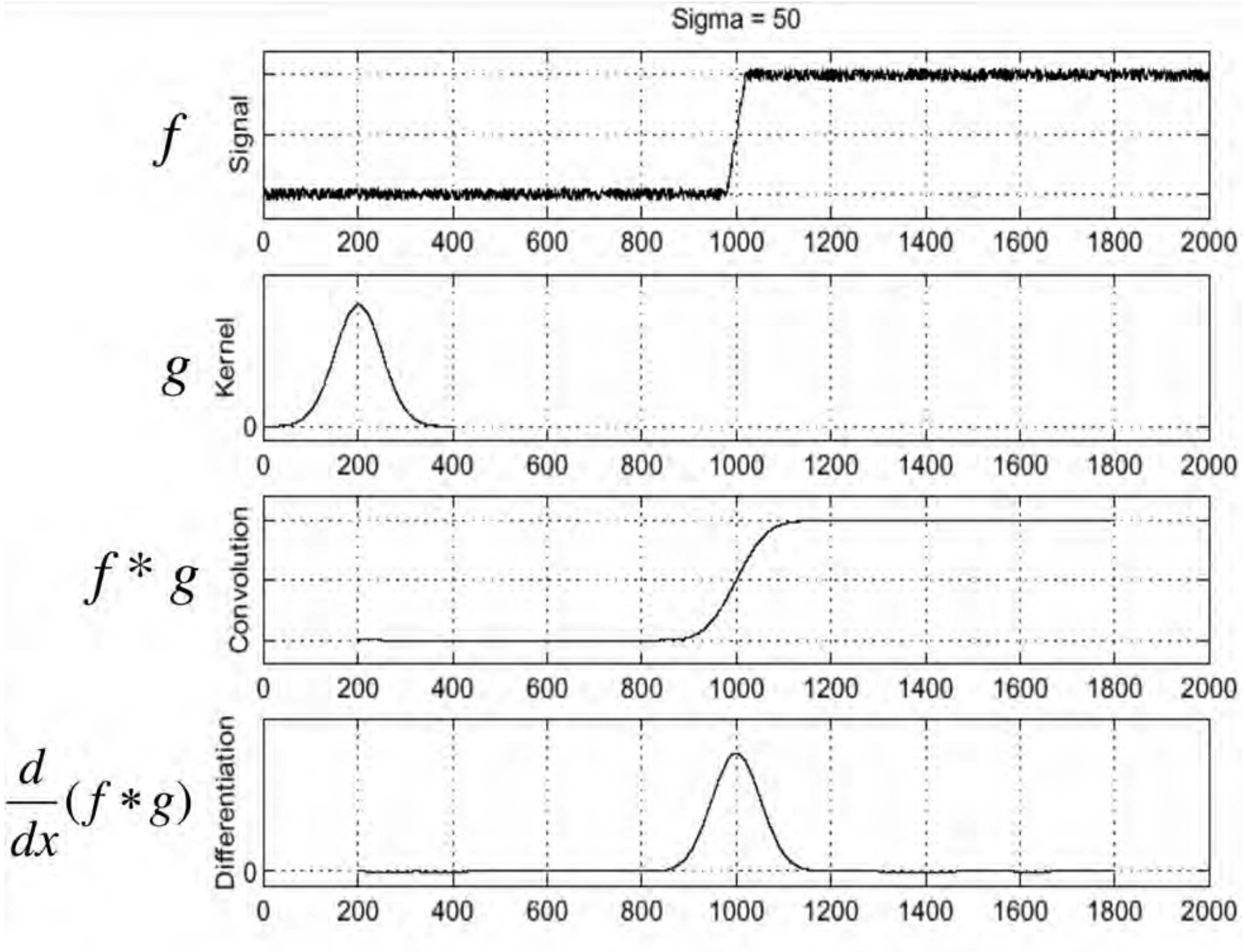
- Let's first try to approximate the underlying function better
- Recall from last week:
  - Approximate each point by a weighted function of its neighbouring points (e.g. Gaussian)
  - Apply as sliding window across function (this is called *convolution* "$*$", as we will see later)



© Steve Seitz

Gradient and Noise - Smoothing

- Averaging avoids random noise
- Gives us a smoother approximation of the function
- Our gradient of the smoothed function gives us nicer edges!
- How do we do this smoothing in practice?



© Steve Seitz

Topic 4: 2D Images and Curves

- 2D Image Patches
- Image Gradients
- Edges
- **Sobel Filter**
- Second-Order Edges
- Canny Edges
- Circle Detection

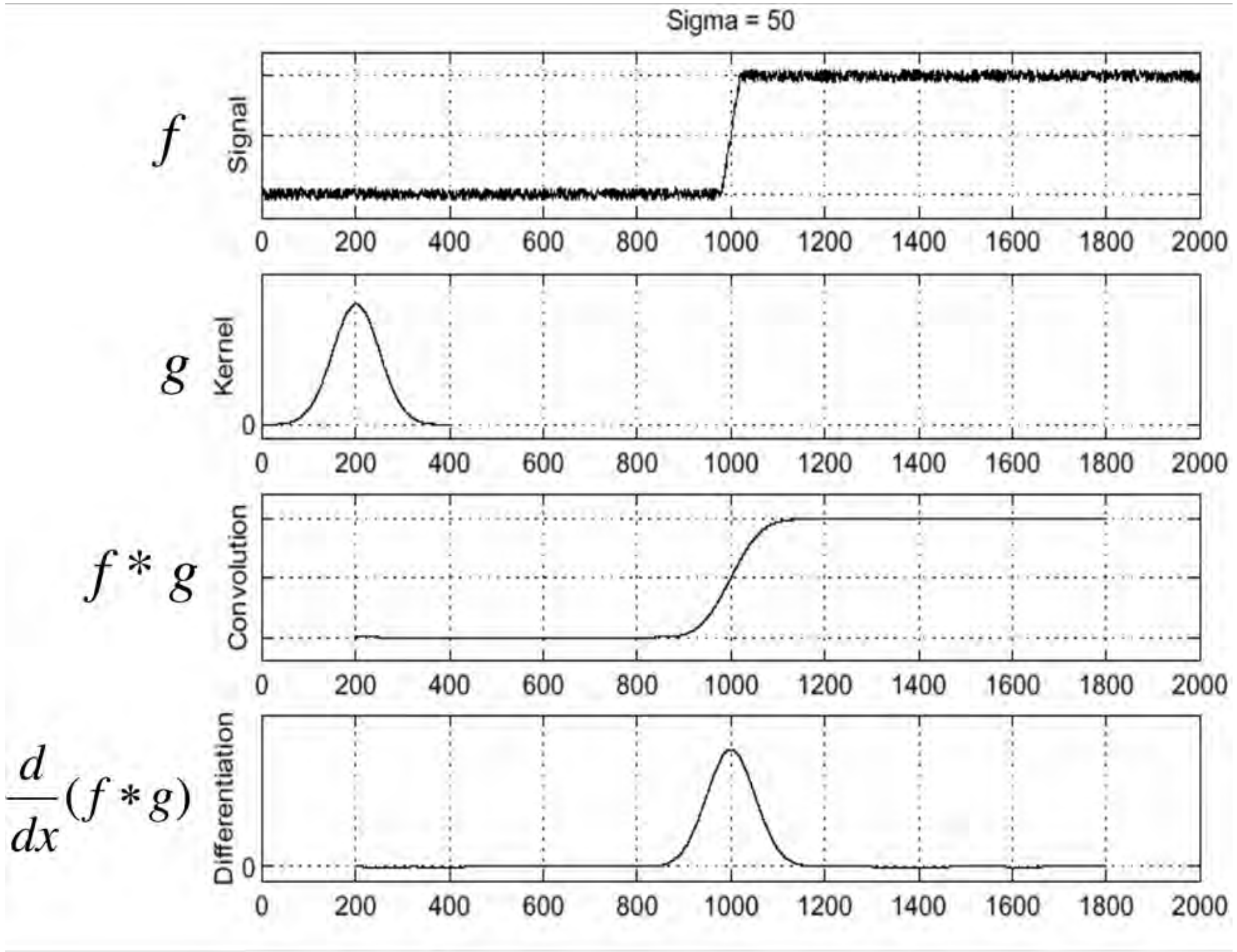
Smoothing/Averaging Filter

- Let's use a simple centre-weighted average:

$$g(x) = \frac{f(x-1) + 2f(x) + f(x+1)}{4}$$

- We apply this to every pixel, using the a sliding window of 3 pixels
- We can represent this as operation for **each pixel location** x:

$$g(x) = \underbrace{\frac{1}{4}\begin{bmatrix}1 & 2 & 1\end{bmatrix}}_{\text{filter}} \underbrace{\begin{bmatrix} f(x-1) \\ f(x) \\ f(x+1) \end{bmatrix}}_{\text{sliding window centred at } x}$$



Convolutional Filters

- So the smoothing operation is:

$$g(x) = \frac{1}{4}\begin{bmatrix}1 & 2 & 1\end{bmatrix}\begin{bmatrix}f(x-1)\\ f(x)\\ f(x+1)\end{bmatrix}$$

- Similarly, the gradient is:

$$f'(x) = \frac{1}{2}\underbrace{\begin{bmatrix}-1 & 0 & 1\end{bmatrix}}_{\text{filter}}\underbrace{\begin{bmatrix}f(x-1)\\ f(x)\\ f(x+1)\end{bmatrix}}_{\text{sliding window centred at } x}$$

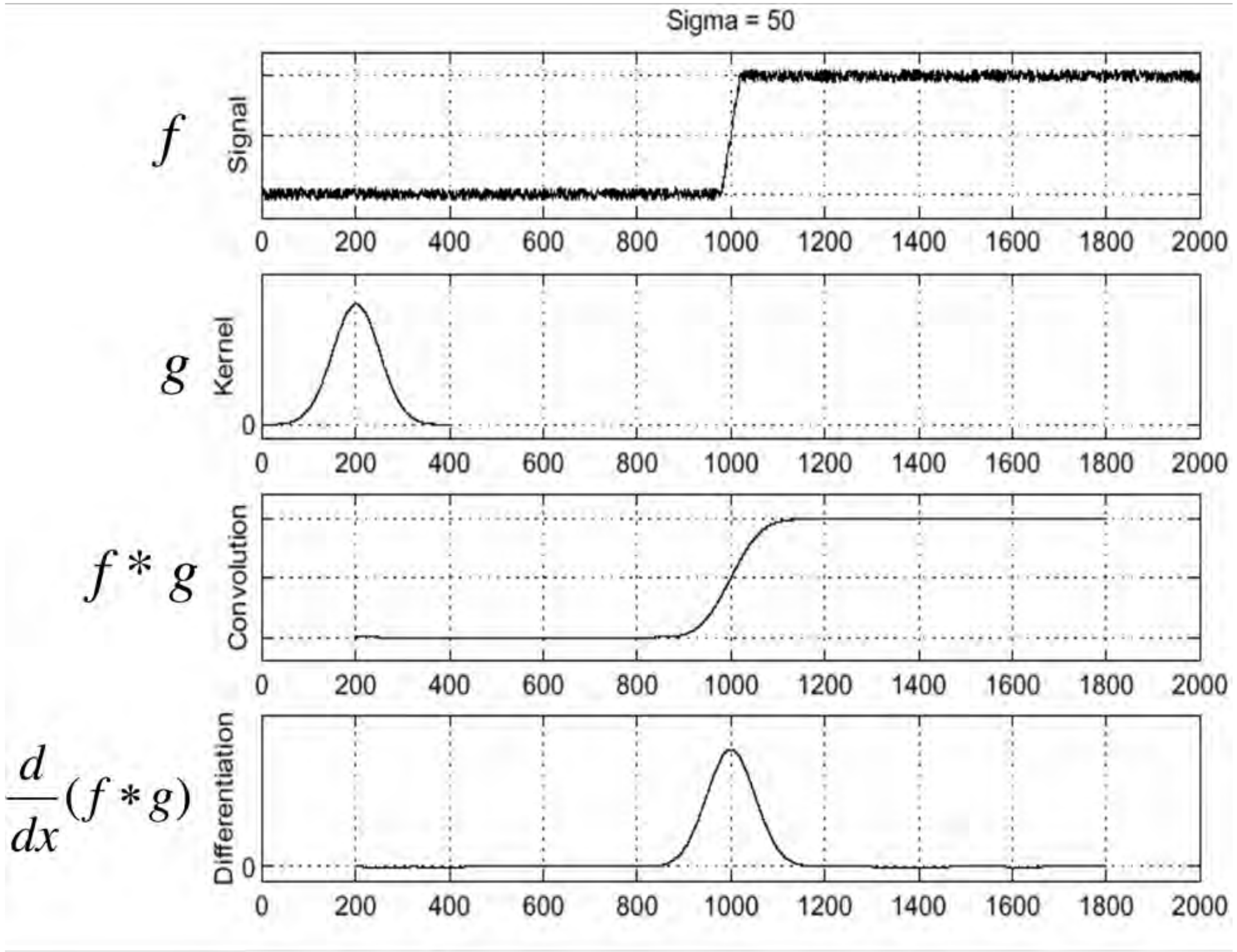


Image Filters

- Many operations for images can be represented as linear filters
- Filters are a linear combination of the neighbourhood pixels
- In general, a 3x3 filter centred on the image at $I(x, y)$ will calculate:
- *operator is convolution **not** matrix multiplication

$$\begin{array}{|c|c|c|}\hline a_0 & a_1 & a_2 \\ \hline a_3 & a_4 & a_5 \\ \hline a_6 & a_7 & a_8 \\ \hline\end{array} * \begin{array}{|c|c|c|}\hline x_0 & x_1 & x_2 \\ \hline x_3 & x_4 & x_5 \\ \hline x_6 & x_7 & x_8 \\ \hline\end{array} \Rightarrow \begin{array}{|c|c|c|}\hline & & \\ \hline & y_4 & \\ \hline & & \\ \hline\end{array}$$

$$y = \sum a_i x_{8-i}$$

Image Filters

- Creates a new image based on linear combination of local neighbourhood
- $*$ is the convolution operator **not** matrix multiplication

Identity

$$\begin{bmatrix}0&0&0\\0&1&0\\0&0&0\end{bmatrix} * \begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix} \Rightarrow \begin{bmatrix} & & \\ & 5 & \\ & & \end{bmatrix}$$

Horizontal Shift

$$\begin{bmatrix}0&0&0\\1&0&0\\0&0&0\end{bmatrix} * \begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix} \Rightarrow \begin{bmatrix} & & \\ & 6 & \\ & & \end{bmatrix}$$

Smoothing Filter + Gradient

- We want to calculate $\frac{\delta}{\delta x} * (g * \ f(x, y))$, i.e. the gradient of the smoothed image
- In 2D for a single direction (e.g. x) we would apply our two filters:

$$\frac{\delta}{\delta x} = \frac{1}{2}\begin{bmatrix}-1 & 0 & 1\end{bmatrix}, \qquad g = \frac{1}{4}\begin{bmatrix}1 & 2 & 1\end{bmatrix}$$

- It turns out that the convolution operator is associative however!
- i.e. $\frac{\delta}{\delta x} * \left(g * \ f(x, y)\right) \equiv \left(\boldsymbol{\frac{\delta}{\delta x}} * \boldsymbol{g}\right) * f(x, y)$

$$\frac{\delta}{\delta x} * g = \frac{1}{2}\begin{bmatrix}1\\2\\1\end{bmatrix}\frac{1}{4}\begin{bmatrix}-1 & 0 & 1\end{bmatrix} = \underbrace{\frac{1}{8}\begin{bmatrix}\mathbf{-1} & \mathbf{0} & \mathbf{1}\\ \mathbf{-2} & \mathbf{0} & \mathbf{2}\\ \mathbf{-1} & \mathbf{0} & \mathbf{1}\end{bmatrix}}_{\textbf{Sobel Filter (x direction)}}$$

Sobel Filter for Edges

$$Sobel_x = \frac{1}{8}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$Sobel_y = \frac{1}{8}\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

- Sobel filter is **the** standard image gradient filter
- There are many others however!
- Typically the normalization (1/8) term is ignored

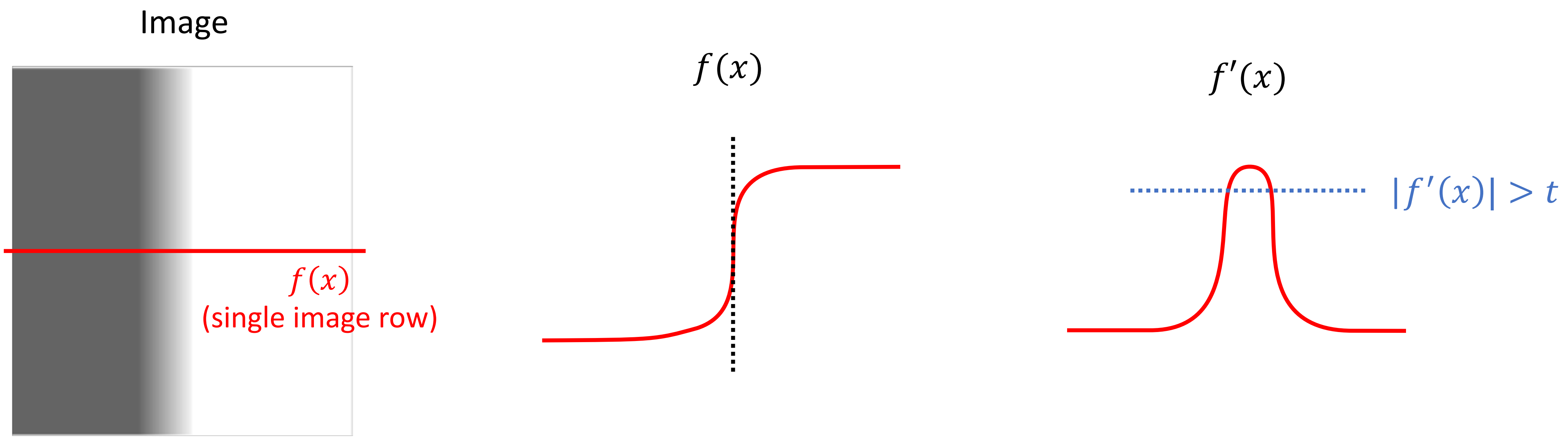
Intermission

Topic 4: 2D Images and Curves

- 2D Image Patches
- Image Gradients
- Edges
- Sobel Filter
- **Second-Order Edges**
- Canny Edges
- Circle Detection

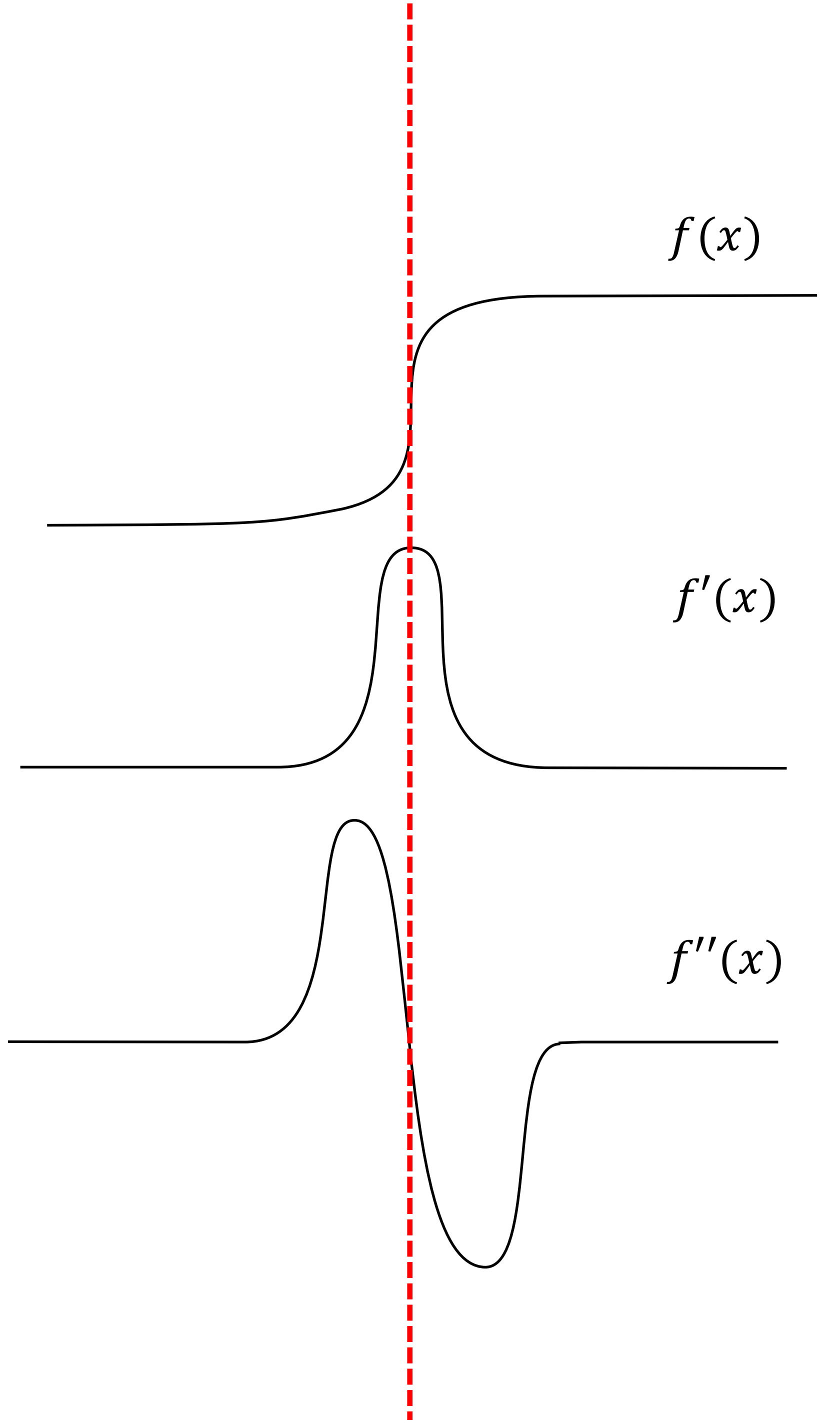
Issues with Gradient Edges

- Using gradient filters to find edges is not ideal:
  - Edge from gradient magnitude is “thick”, ideally as localized to one pixel
  - This is because we threshold the gradient magnitude
  - Can possibly include several pixels around the true edge location



Finding Better Edges

- What we are interested in finding is a single edge, at the exact extrema of $f'(x)$
- This is actually where zero crossing of the second derivative $f''(x)$ is!
- Using the second derivative, we can potentially **localize** edges better than with the gradient



Going back to 2D

- Recall the gradient is defined:

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

- In 2D the second derivative is called the **Laplacian** (∇^2):

$$\nabla^2 f = \nabla \cdot \nabla f = \begin{bmatrix} \frac{\partial}{\partial x} & \frac{\partial}{\partial y} \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \end{bmatrix} = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

Discrete 2nd Order Derivative

- Just as with the first derivative, there are many finite difference approximations of the second derivative
- When based on the central difference, the approximation for $f(x)$:

$$\frac{d^2 f}{dx^2} \approx f(x+1) - 2f(x) + f(x-1)$$

Derivation of finite differences for 2nd order derivatives:
https://math.stackexchange.com/questions/210264/second-derivative-formula-derivation
Derivation of finite differences for 2nd order derivatives in terms of Taylor series:
https://geometrictools.com/Documentation/FiniteDifferences.pdf

Discrete Laplacian – 2D

$$\frac{d^2 f}{dx^2} \approx f(x+1, y) - 2f(x, y) + f(x-1, y)$$

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

- Therefore, we can approximate the Laplacian with:

$$\nabla^2 f \approx f(x+1, y) + f(x, y+1) + f(x-1, y) + f(x, y-1) - 4f(x, y)$$

Discrete Laplacian Filter

$$\nabla^2 f \approx f(x+1,y)+f(x,y+1)+f(x-1,y)+f(x,y-1)-4f(x,y)$$

- Let’s represent this in convolutional filter form:

$$\nabla^2 f \approx \begin{bmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

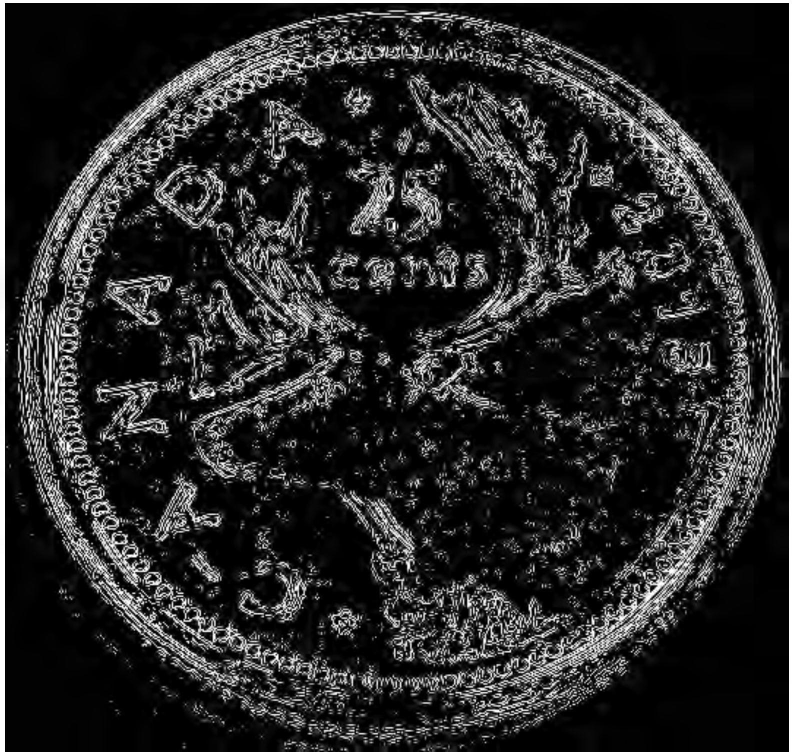
$$= \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Laplacian of Gaussian

- Just as with the gradient, the Laplacian is highly sensitive to noise
- Typically smooth the image before applying the Laplacian with Gaussian
- Unlike with gradient we do not get direction of edge!

Image

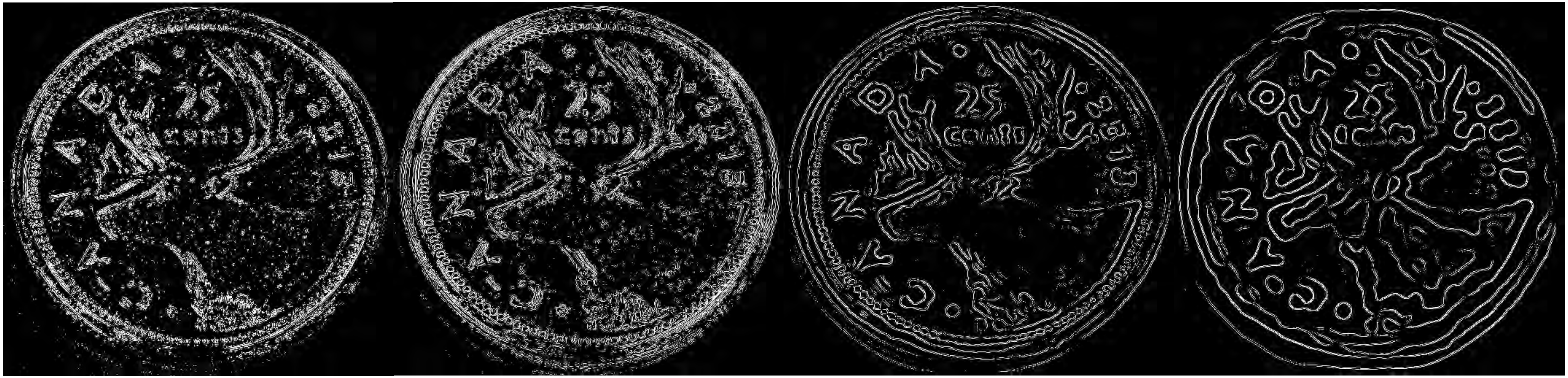
Smoothed image



Laplacian zero-crossings

Laplacian of Gaussian

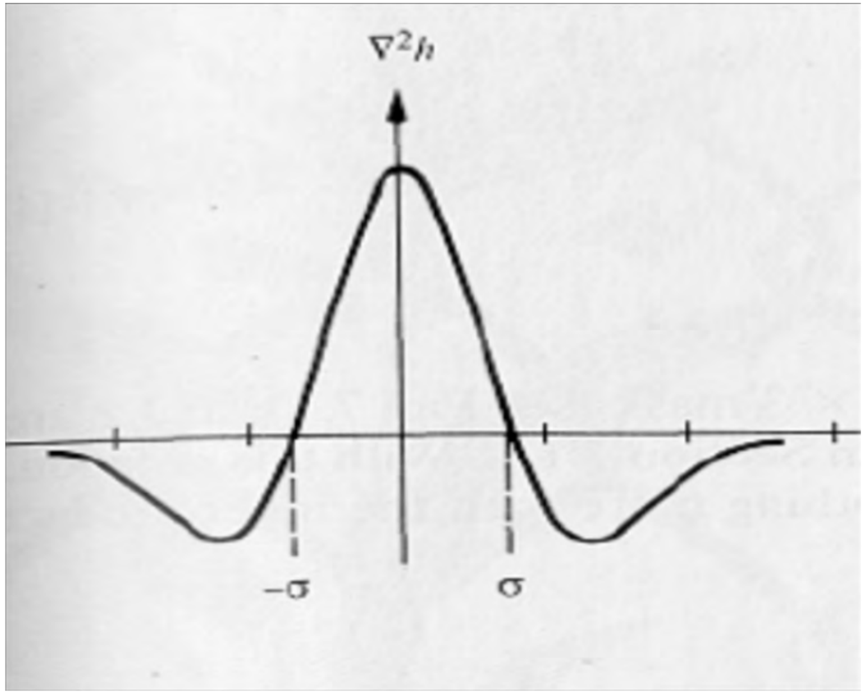
- This is often called the Laplacian of Gaussian
- Using Gaussian filters of different σ results in very different edges
- In fact, we see edges of different scales – this will be important later!

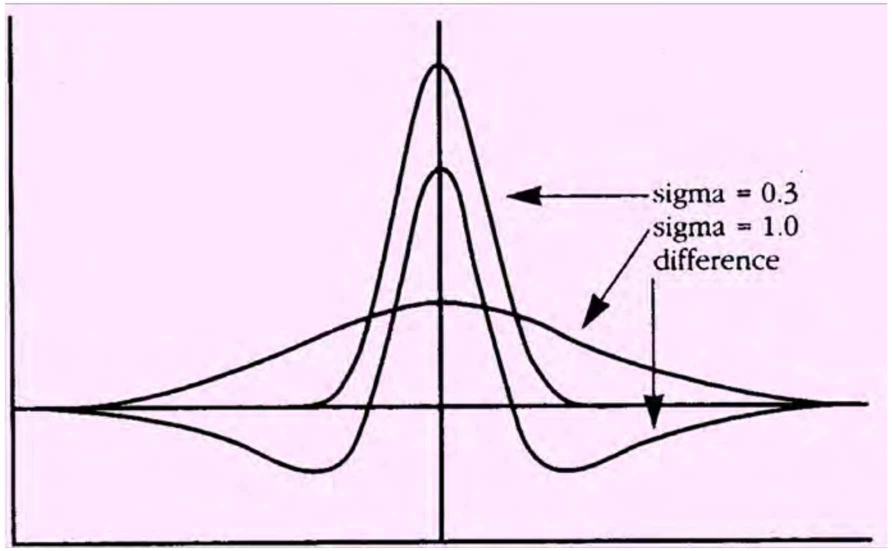


$\sigma = 3$ $\sigma = 9$ $\sigma = 27$ $\sigma = 49$

Difference of Gaussians

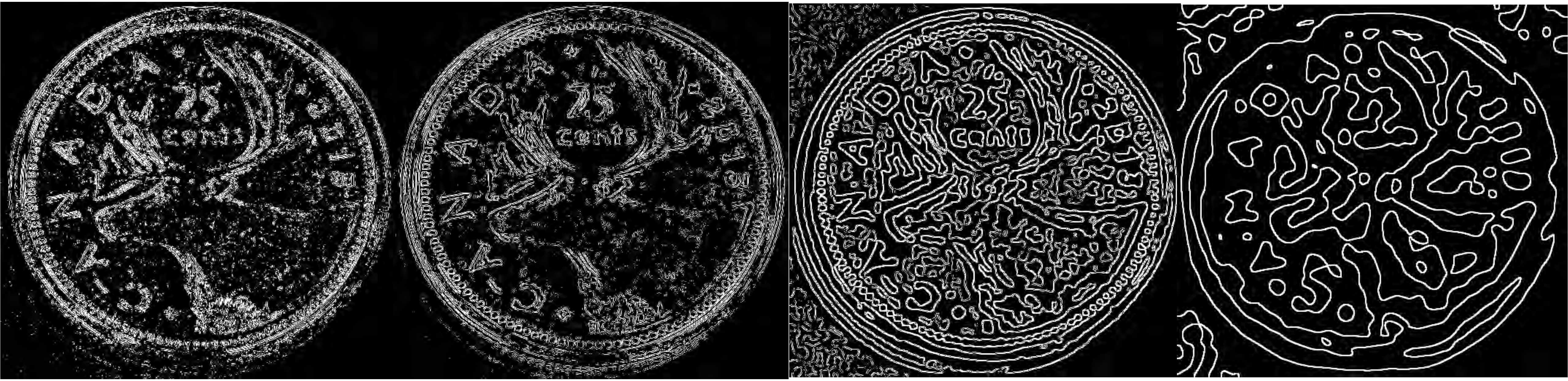
- The Laplacian of Gaussians is well approximated by simply subtracting two Gaussian functions
- This is called “Difference of Gaussians”





Difference of Gaussians

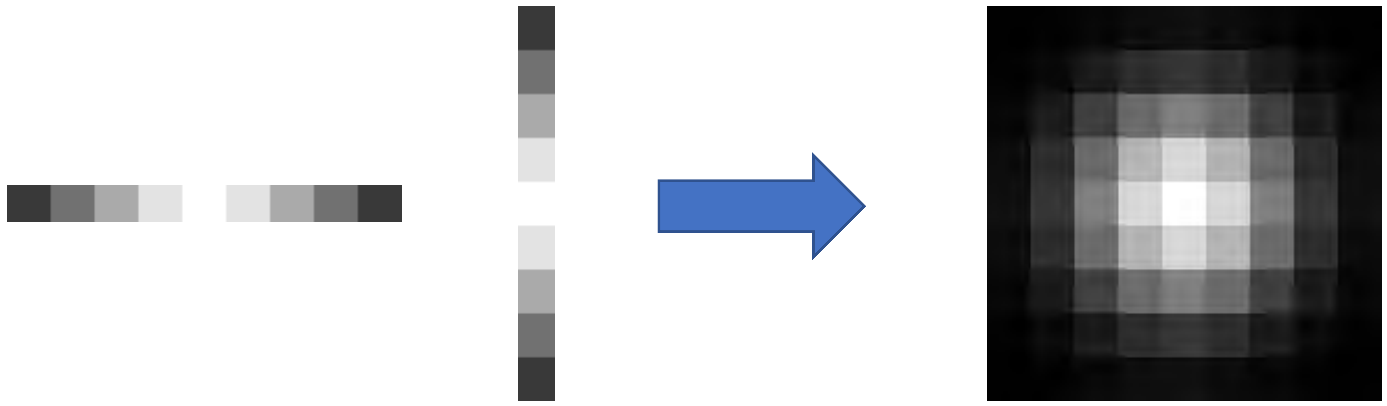
- Typically use $\sigma_2 \approx 1.6\,\sigma_1$ to approximate LoG
- Notice again that we get edges from different scales with increasing σ



$\sigma_1 = 3, \sigma_2 = 5$ $\quad$ $\sigma_1 = 9, \sigma_2 = 15$ $\quad$ $\sigma_1 = 21\ \sigma_2 = 33$ $\quad$ $\sigma_1 = 49\ \sigma_2 = 75$

Difference of Gaussians

- Gaussian filter is **separable**, can compute it with 1 x k + k x 1 filters:



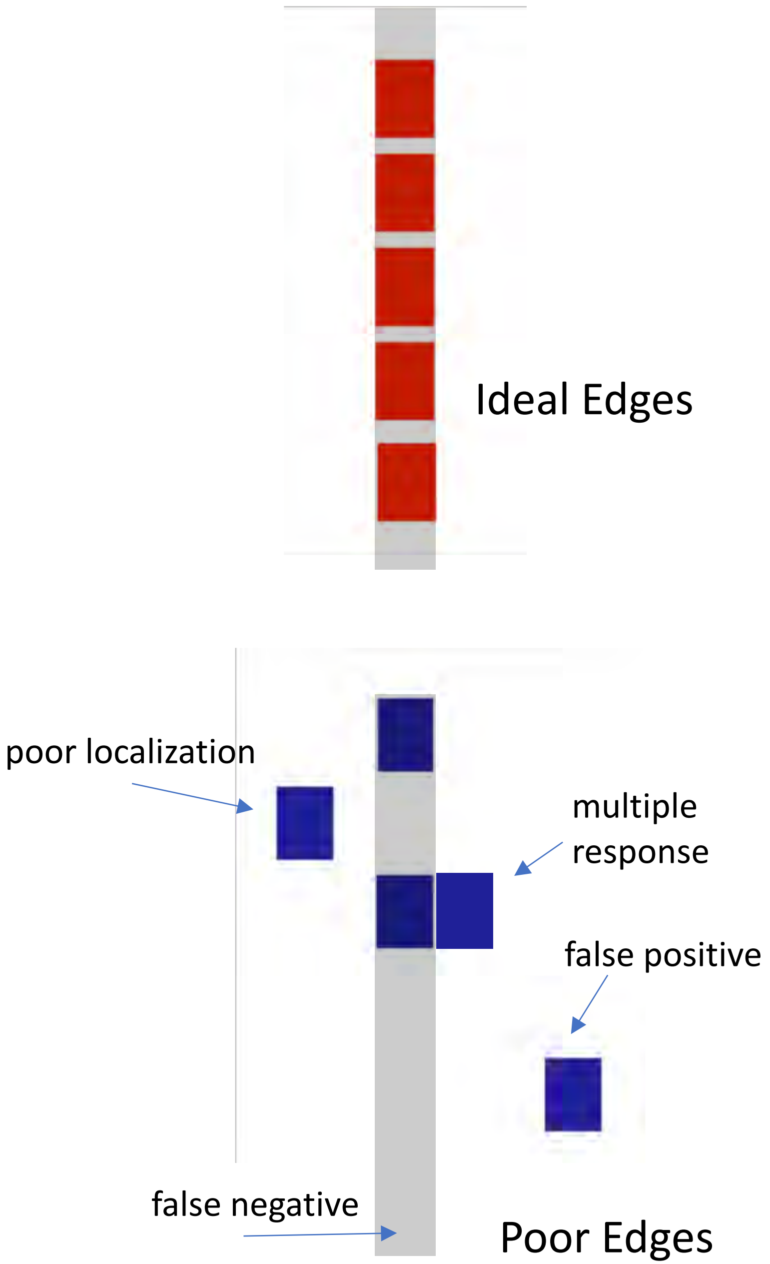
- Actually Laplacian of Gaussian also has separable representation – but requires 4 filters instead of 2
- The Laplacian of Gaussian is relatively expensive to compute

Topic 4: 2D Images and Curves

- 2D Image Patches
- Image Gradients
- Edges
- Sobel Filter
- Second-Order Edges
- **Canny Edges**
- Circle Detection

Ideal Edge Detector

- We are still far from the ideal edge detector…
- Good Detection:
  - Want to minimize false edges, i.e. false positives
  - Want to minimize true edges missed, i.e. false negatives
- Good Localization
  - Want to find edges as close to true edge location as possible
- Single Response
  - Want to find only one pixel for any single true edge location



Derived from work of L. Fei-Fei

Canny Edge Detector

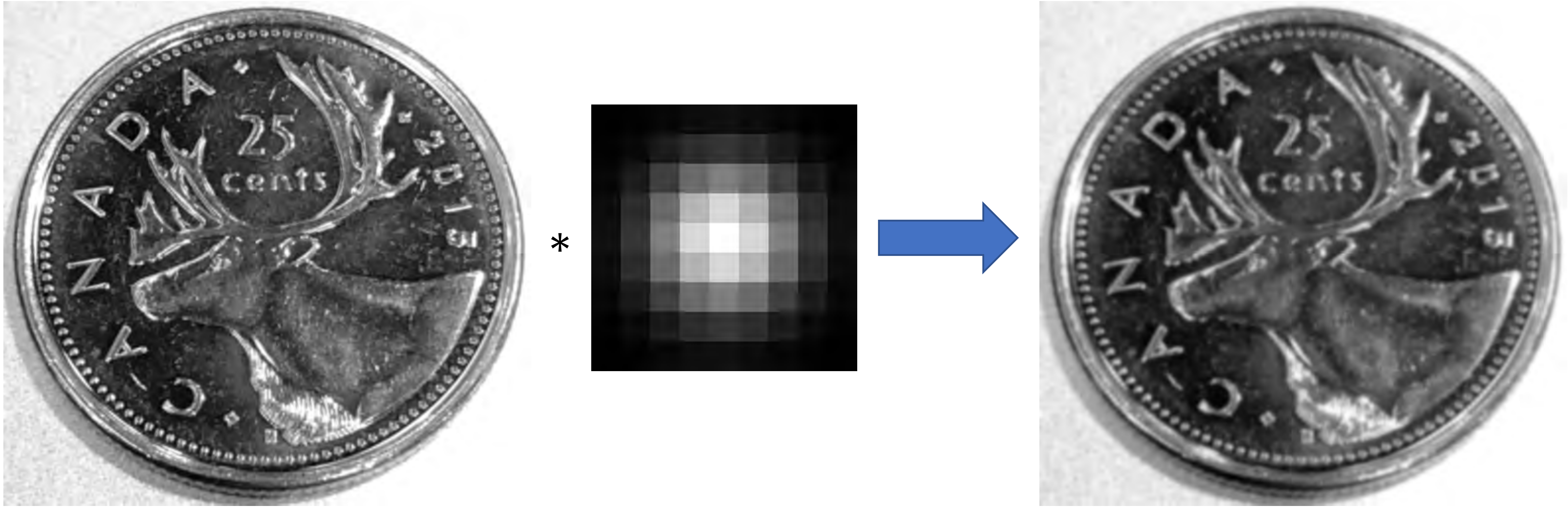
- First introduced by John Canny in his 1983 M.Sc. Thesis
- Still the most widely used edge detector today!

- Outline:
  - Step 1: Filter image with the derivative of Gaussian
  - Step 2: Calculate magnitude and orientation of resulting gradient
  - Step 4: Thin out thick edges (“non-maxima suppression”)
  - Step 3: Hysteresis thresholding:
    - Use initial high threshold to find beginning of edge curve
    - Find remaining pixels belonging to edge curve with second, lower threshold

J. Canny, A Computational Approach To Edge Detection, IEEE Trans. Pattern Analysis and Machine Intelligence, 8:679-714, 1986.
https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html

Canny Step 1 – Derivative of Gaussian

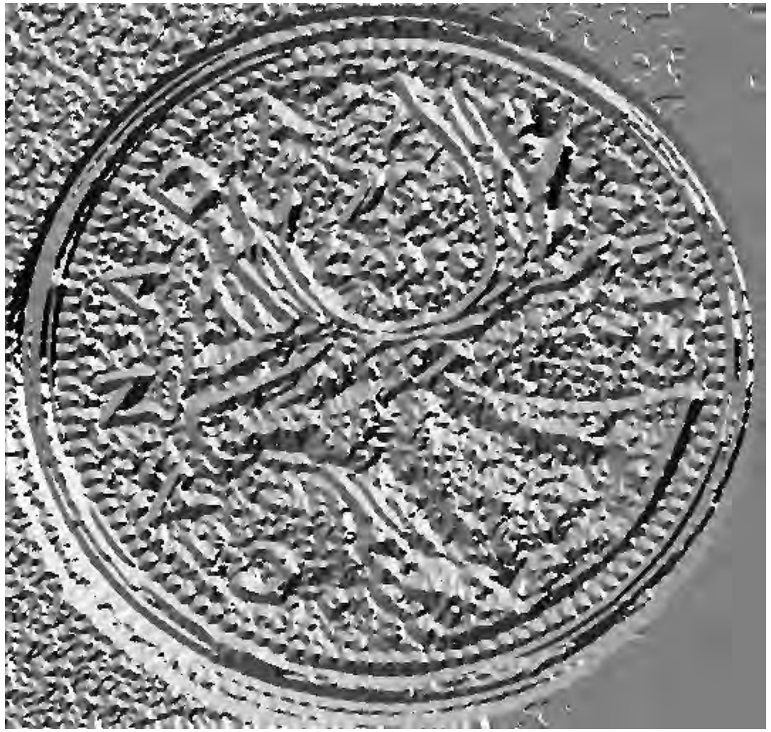
- Almost the exact same as Sobel, except we use **Gaussian** to smooth
  - Recall: Sobel uses a simple weighted average instead



Canny Step 2 – Calculate Magnitude/Direction

- Calculate gradient (central difference) magnitude and direction
- Direction is typically discretized to 4 possible angles (0, 45, 90, 135)

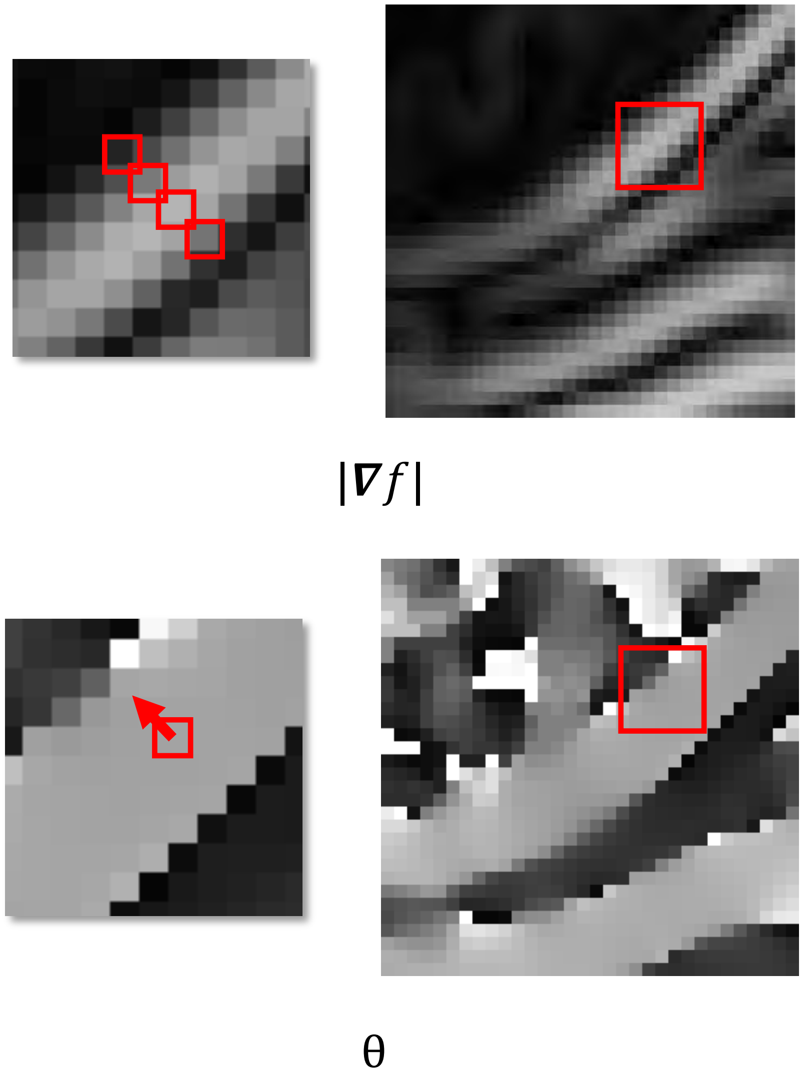
$|\nabla f|$



θ

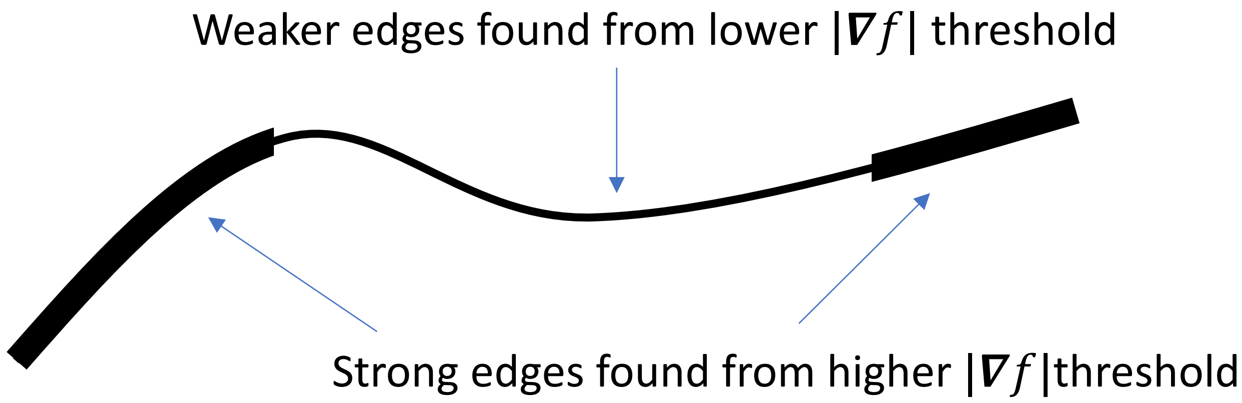
Canny Step 3 – Non-Maxima Suppression

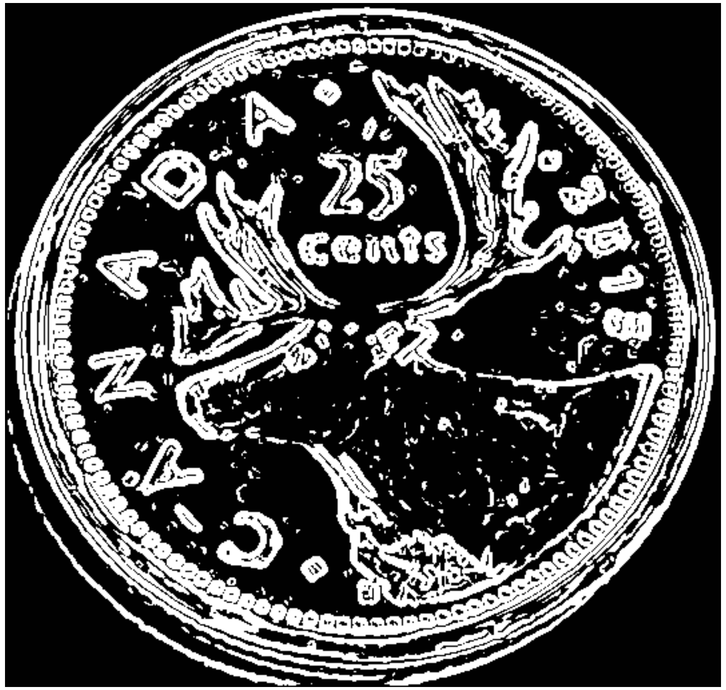
- Remember we only want a single pixel response for each edge!
- "Non-Maxima Suppression"
  - Step 1: Compare the gradient magnitude along the non-zero pixels in the edge mask pixels in the +/- gradient direction (i.e. perpendicular to the edge)
  - Step 2: Keep largest gradient magnitude found, zero out others



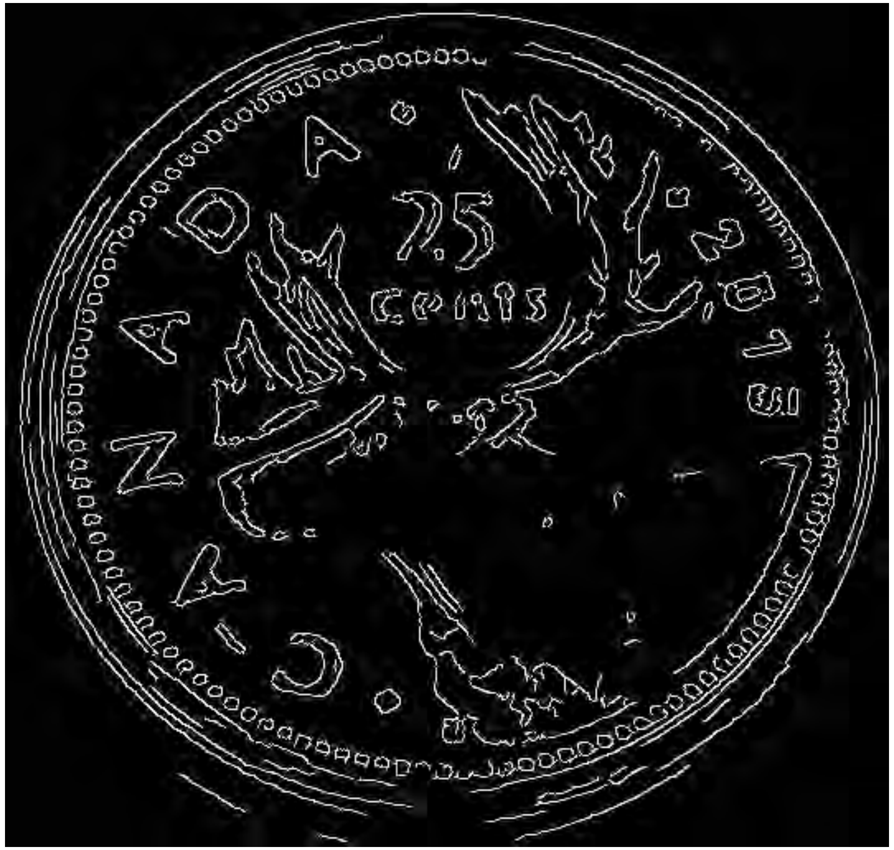
Canny Step 4 – Hysteresis

- Uses two thresholds – upper and lower
  - Canny recommended a *upper*:*lower* ratio between 2:1 and 3:1
- Find initial edge points using high threshold
- Use lower threshold to find other edge points that agree with the edge curve





Canny Edges



Topic 4: 2D Images and Curves

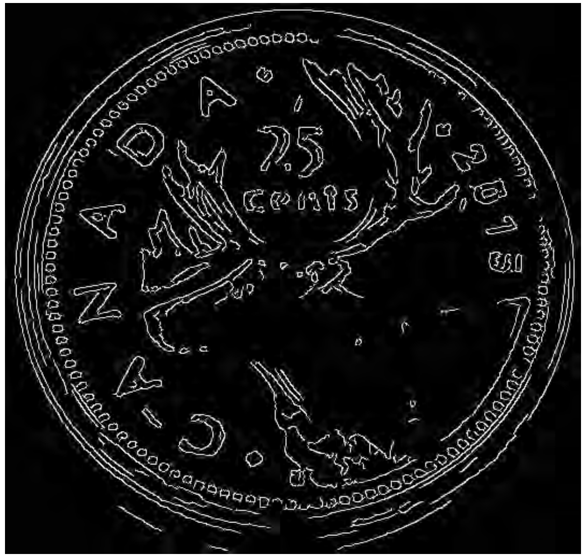
- 2D Image Patches
- Image Gradients
- Edges
- Sobel Filter
- Second-Order Edges
- Canny Edges
- **Circle Detection**

Putting it all together!

- We create a startup (it’s the thing to do)
- Our pitch is a new app: couchchange™
  - Users upload phone images of their couch
  - We return an image highlighting where the coins is
- How do we do this?
  - Hint: you’ve actually seen everything you need already

Finding Circles

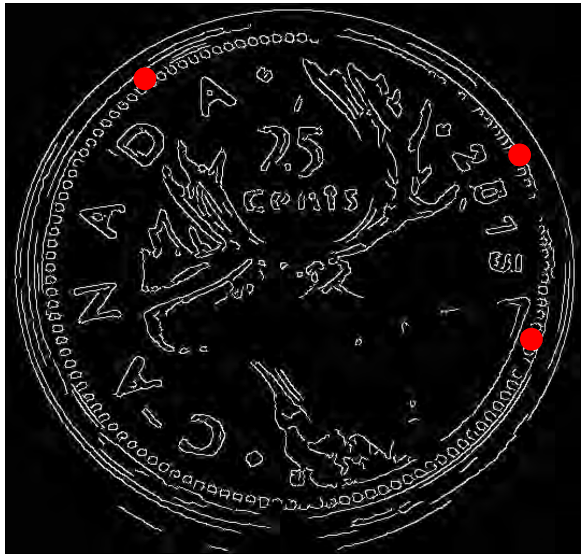
- Assumptions:
  - Images of a planar surface with coins
  - Surface is perpendicular to the camera
    - (i.e. quarters look like circles, not ellipses)
- Step 1: Find Canny edges
  - This will give us the edges for the object outline
  - We will also get lots of extraneous edges however!
- Step 2: Fit a model of a circle to the edge points
  - We need a robust method to do this…



RANSAC Circle Detection

- We want to fit a model of a circle to our edge points
- Circle centred at (a, b) of radius r:

$$(x - a)^2 + (y - b)^2 = r^2$$



  - Use a set of randomly sampled points to give an initial estimate of the model
  - Need a minimum of 3 points to find parameters for a circle

RANSAC Circle Detection – Initial Circle

- Calculate an initial estimate of our circle parameters using our 3 initial points - many approaches to this (see reference)!
- Use our circle equations, gives us 3 equations, 3 unknowns:

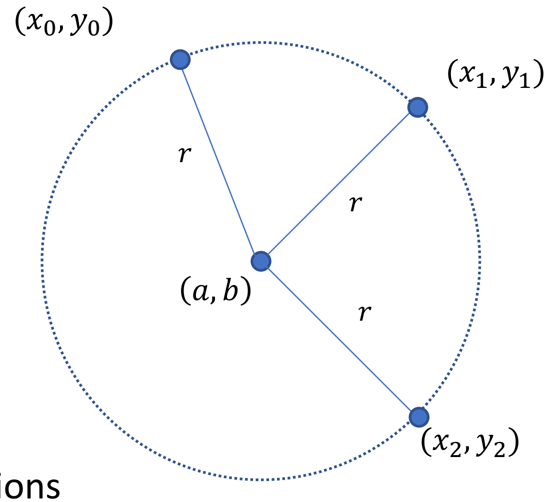
$$(x_0 - a)^2 + (y_0 - b)^2 = r^2$$
$$(x_1 - a)^2 + (y_1 - b)^2 = r^2$$
$$(x_2 - a)^2 + (y_2 - b)^2 = r^2$$

However it looks like a quadratic system of equations! This is simpler than it seems, if we write it as:

$$(x_0 - a)^2 + (y_0 - b)^2 = (x_1 - a)^2 + (y_1 - b)^2$$
$$(x_1 - a)^2 + (y_1 - b)^2 = (x_2 - a)^2 + (y_2 - b)^2$$

Nicely the quadratic terms a^2, b^2cancel out, and we are left with two simple (but very lengthy) equations for calculating (a, b)!
We can then use our circle equation to calculate r:

$$r = \sqrt{(x_0 - a)^2 + (y_0 - b)^2}$$



https://qc.edu.hk/math/Advanced%20Level/circle%20given%203%20points.htm

RANSAC Circle Detection – Random Sample

- Checking for consensus is expensive
- We need to ensure our initial points are reasonable
- The initial points should not be too close – why?
- The initial points should not be co-linear – why?

RANSAC Circle Detection – Consensus

- We now have a circle centred at (a, b) of radius r:

$$(x - a)^2 + (y - b)^2 = r^2$$

  - Inliers to our proposed model should be within a certain distance threshold of the circle's circumference:

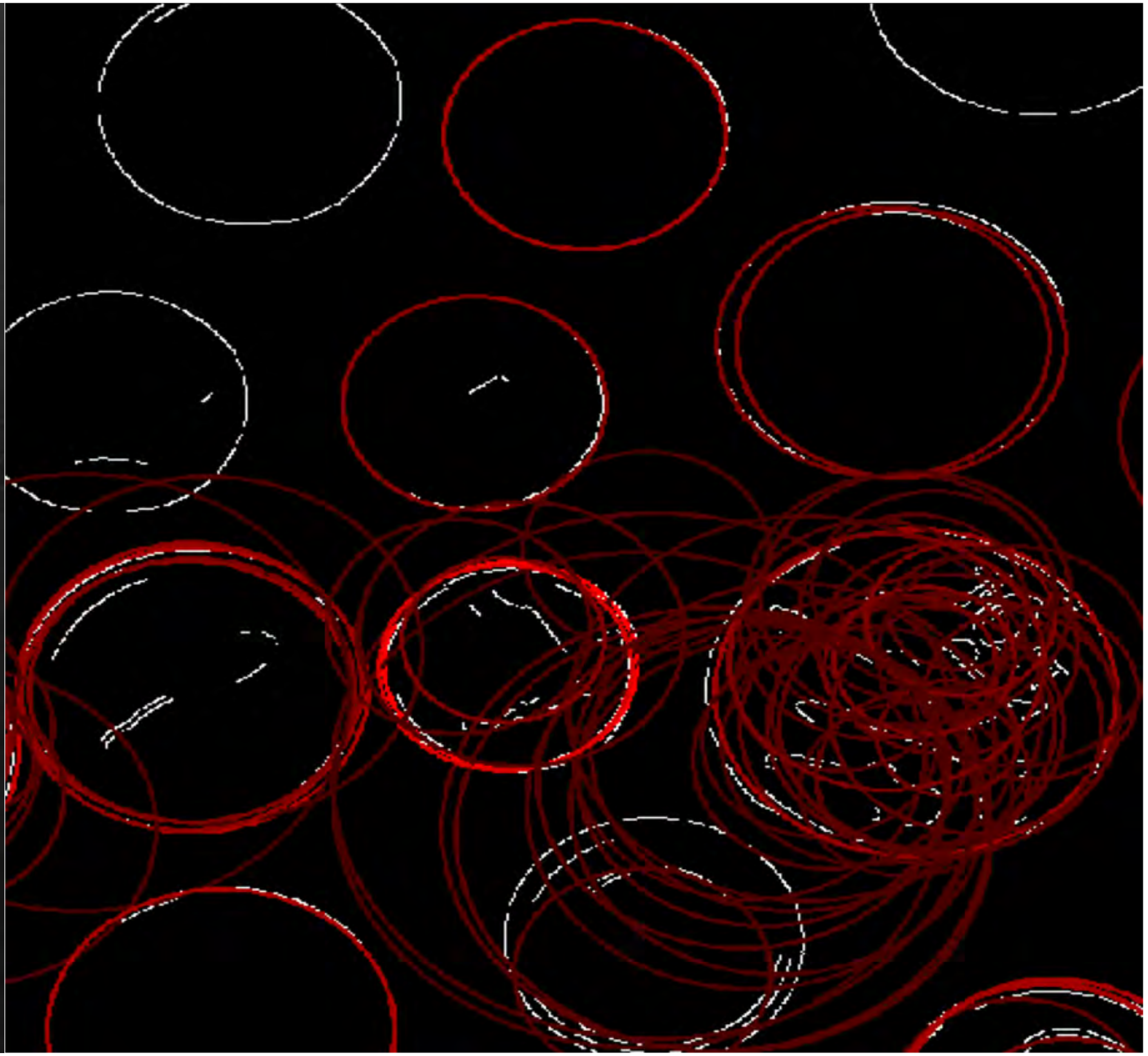
$$\sqrt{(x_{\text{inlier}} - a)^2 + (y_{\text{inlier}} - b)^2} - r^2 \leq d_{inlier}$$

  - All other points are deemed outliers!
  - Re-fit our model to our new complete set of inliers, to get a better estimate of the parameters

RANSAC Circle Detection – Consensus

- Count # of inlier edges we have for our re-fit model
- The larger the # of edges, the better the model!
- Decide if we have a good circle – if so, save it
  - Set threshold relative to circumference of circle, otherwise we bias towards large circles!
- Continue looking for better models until we reach iteration limit

RANSAC Circle Detection – Multiple Circles



Hough Transform

- Another common method of finding circles is the Hough transform
- Instead of fitting a model to a set of initial points, considers all possible models for each point
- We do this by voting in the parameter space
- For example, for a circle of **known** radius r:

$$(x - \boldsymbol{a})^2 + (y - \boldsymbol{b})^2 = r^2$$

- Our parameter space is the 2D space of all possible centres (a, b)

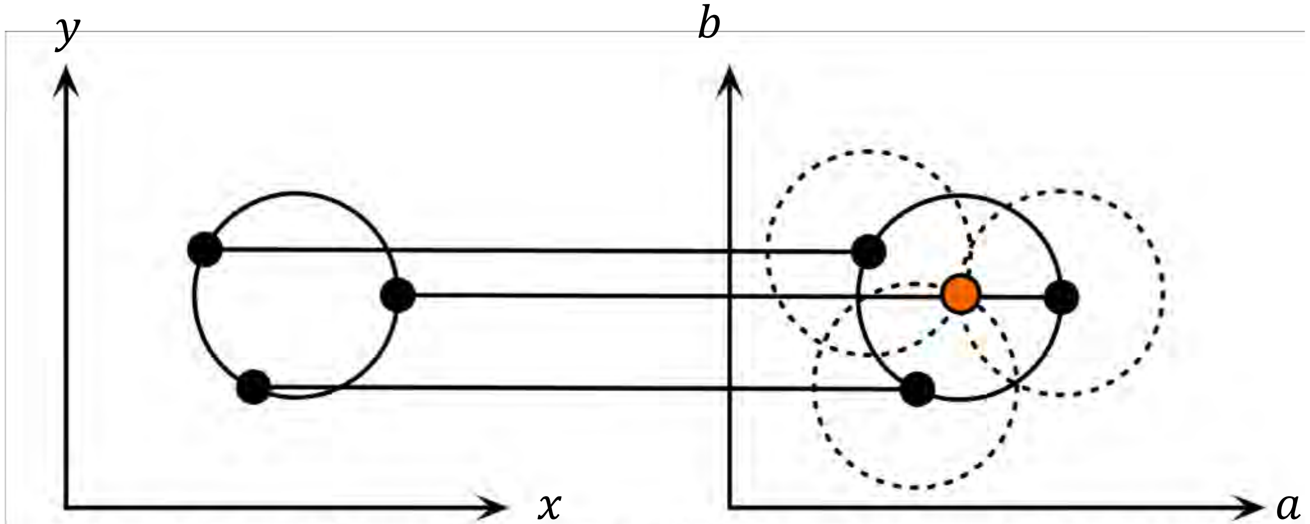
Hough Transform

- We can also represent a circle with the parameterization:

$$x = a + r\cos\theta$$
$$y = b + r\sin\theta$$

where $\theta \in [0, 2\pi]$

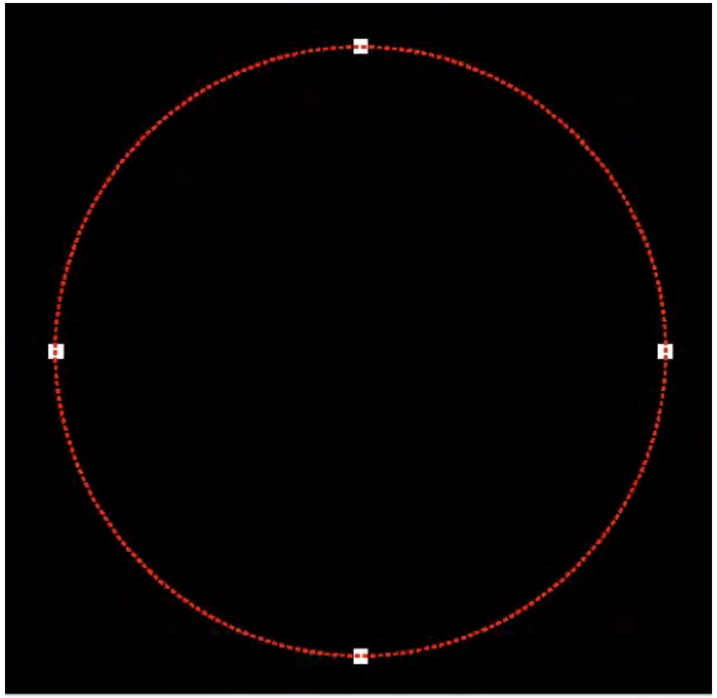
- Each edge pixel in the image space (x, y) corresponds to many possible circles, i.e. parameters (a, b)

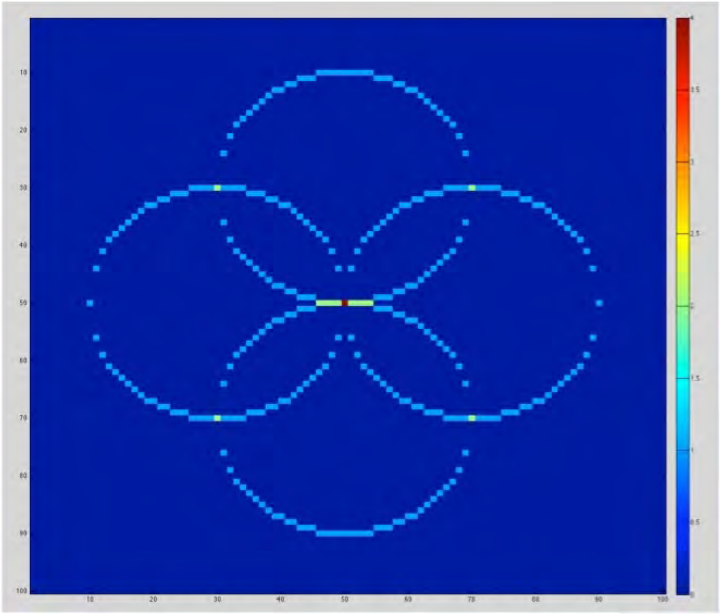


Each point in geometric space (left) generates a circle in parameter space (right). The circles in parameter space intersect at the (a, b) that is the center in geometric space.

Hough Transform

- Each edge pixel in the image space (x, y) votes for all possible circle centres (a, b) in parameter space that could have caused it
- The parameter space accumulates votes for all edge points
- Circle parameters that match the most edge pixels gain the most votes!
- Peaks of the Hough Transform are chosen as candidate circles

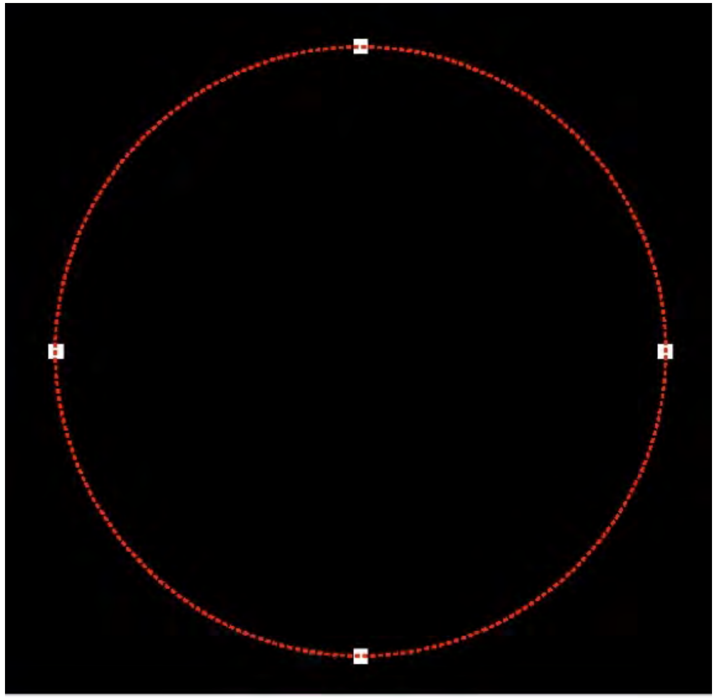


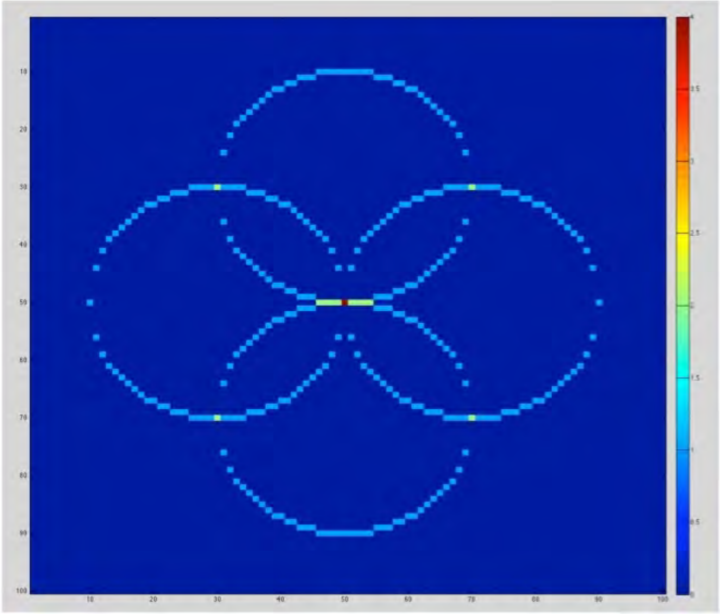


From Wikipedia

Hough Transform

- Disadvantages:
  - Not as robust to outliers/noise
  - Memory used to store discretized parameter space is considerable
  - Does not scale to models with many parameters





Overview of Topic 5

- Today we learned an early computer vision pipeline:
  - Given raw image input
  - Find salient local features (e.g. edges)
  - Match a model (encoding our assumptions) to our extracted features
- We went from a **raw image** to some **understanding** of the image

End of Topic 5