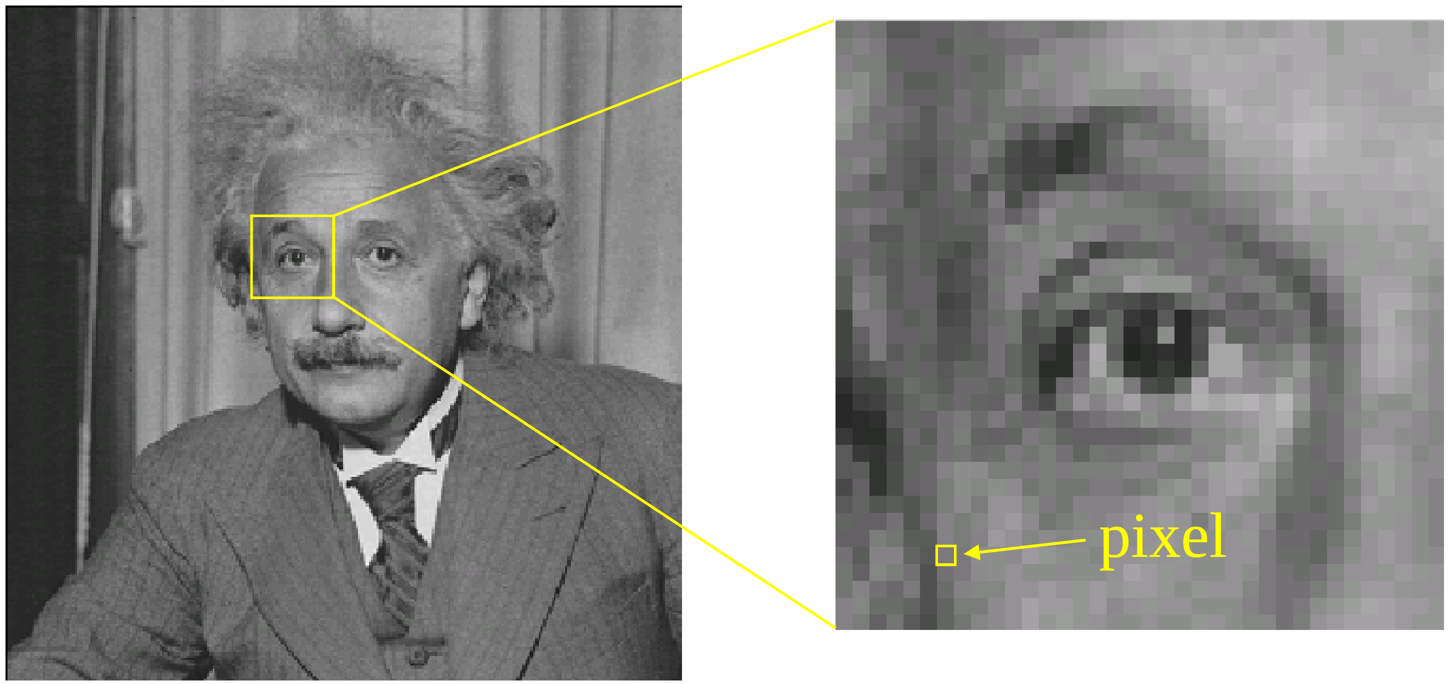
Topic 4:

Local analysis of image patches

- What do we mean by an image “patch”?
- Applications of local image analysis
- Visualizing 1D and 2D intensity functions

Local Image Patches

So far, we have considered pixels completely independently of each other (as RGB values or, as vectors [R, G, B])



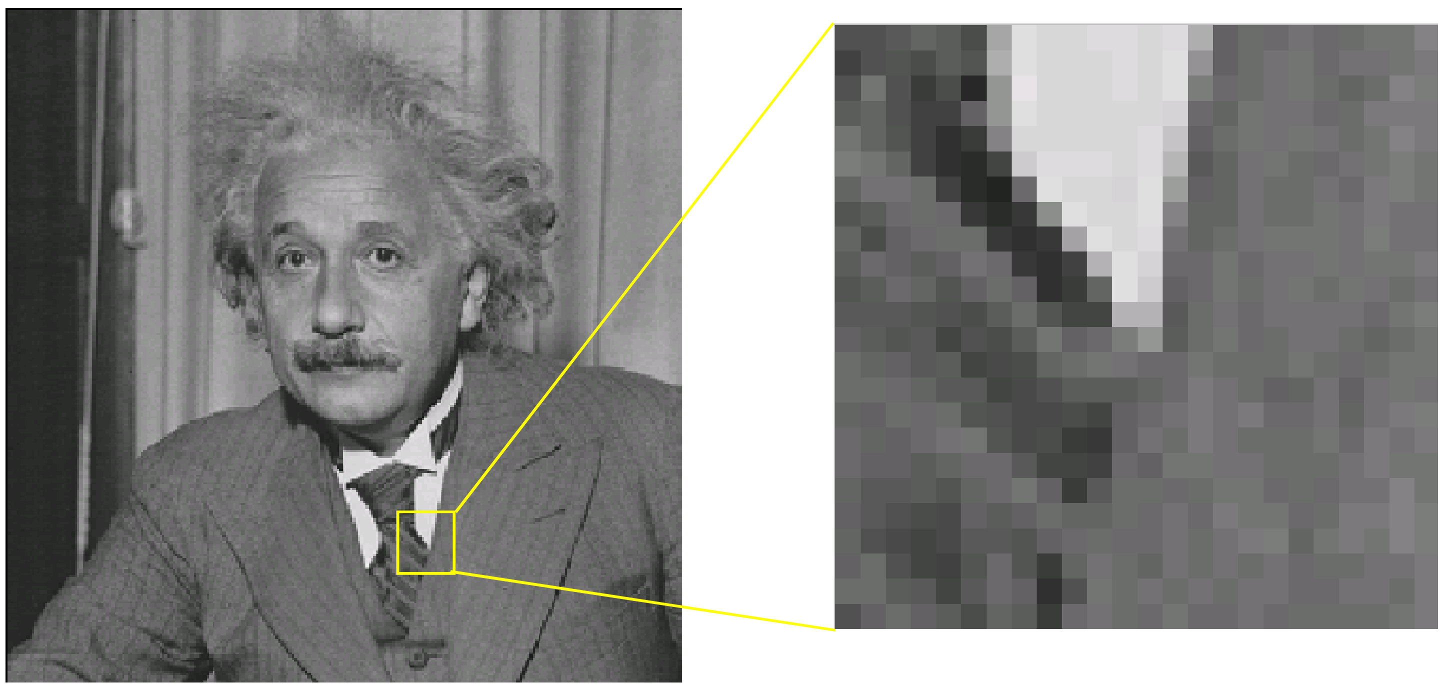
In reality, photos have a great deal of structure

This structure can be analyzed at a local level (eg., small groups of nearby pixels) or a global one (eg. entire image)

Local Image Patches

Qualitatively, we can think of many different types of patches in an image

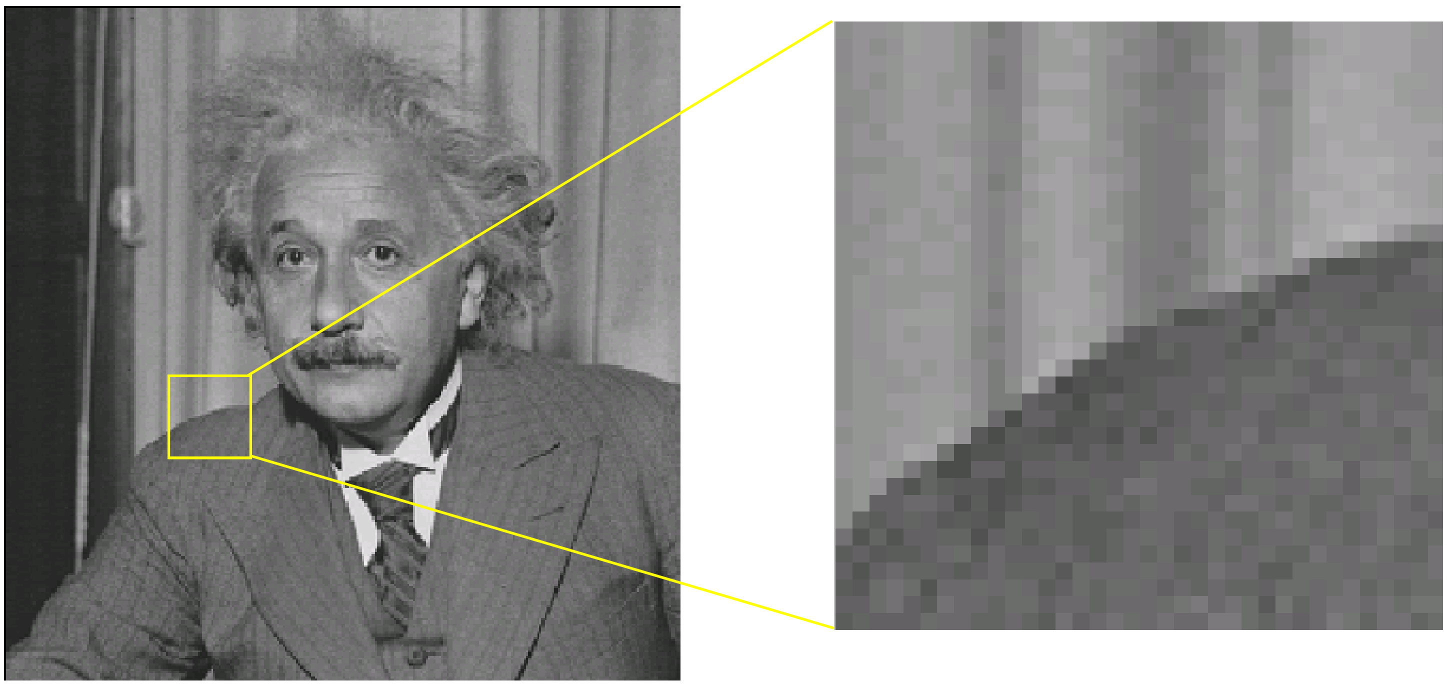
Patches corresponding to a “corner” in the image



Local Image Patches

Qualitatively, we can think of many different types of patches in an image

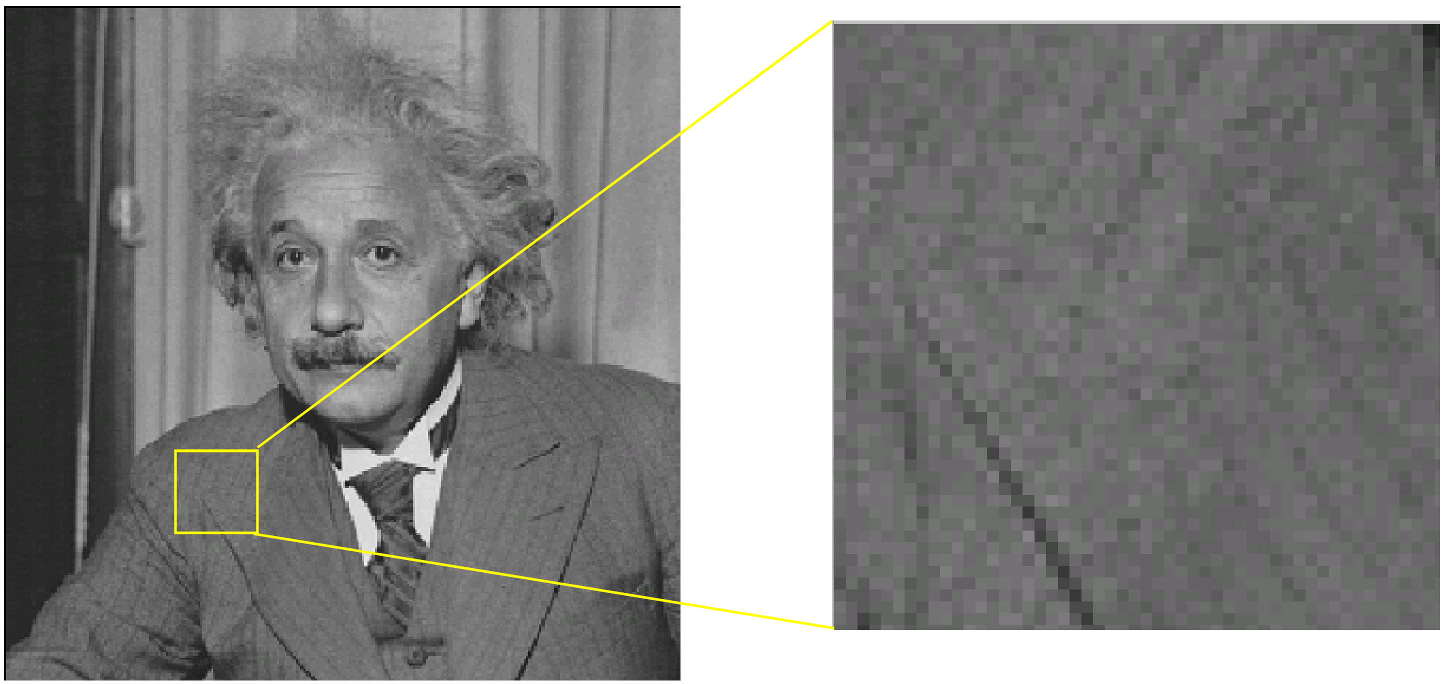
Patches corresponding to an "edge" in the image



Local Image Patches

Qualitatively, we can think of many different types of patches in an image

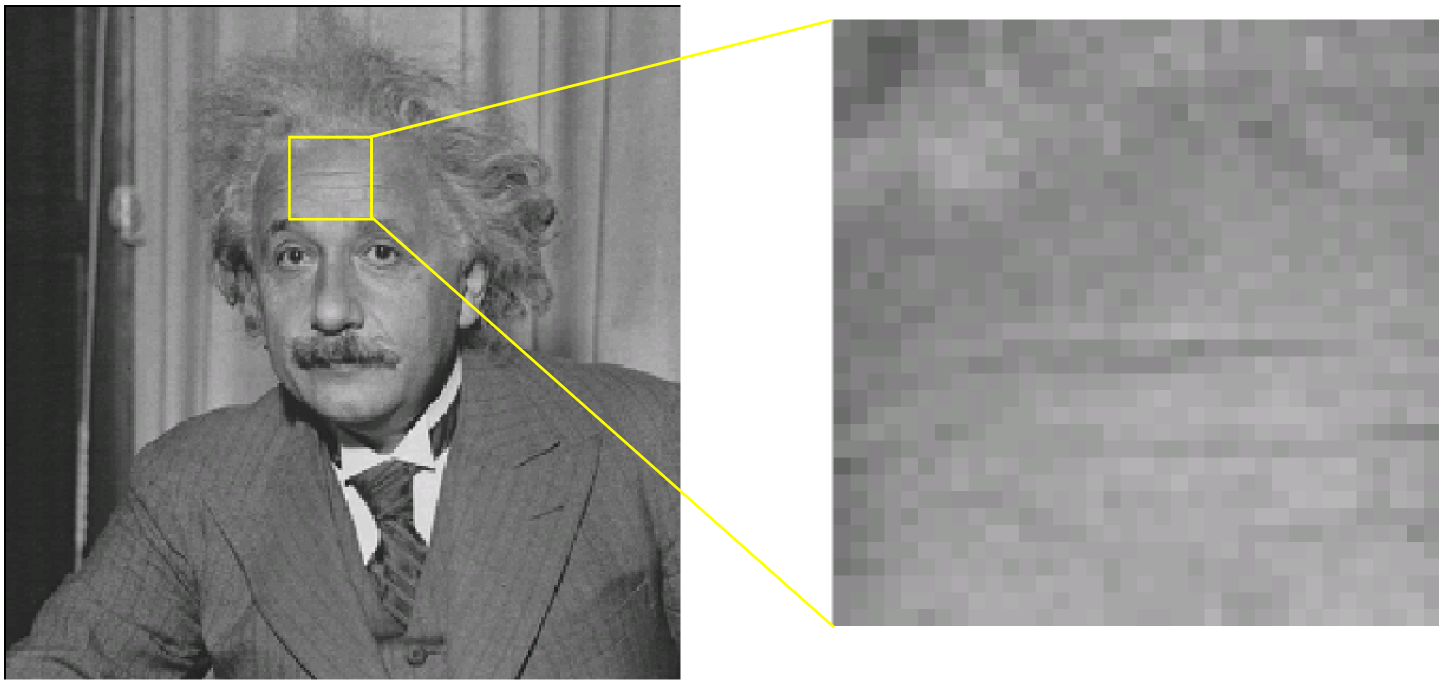
Patches of uniform texture



Local Image Patches

Qualitatively, we can think of many different types of patches in an image

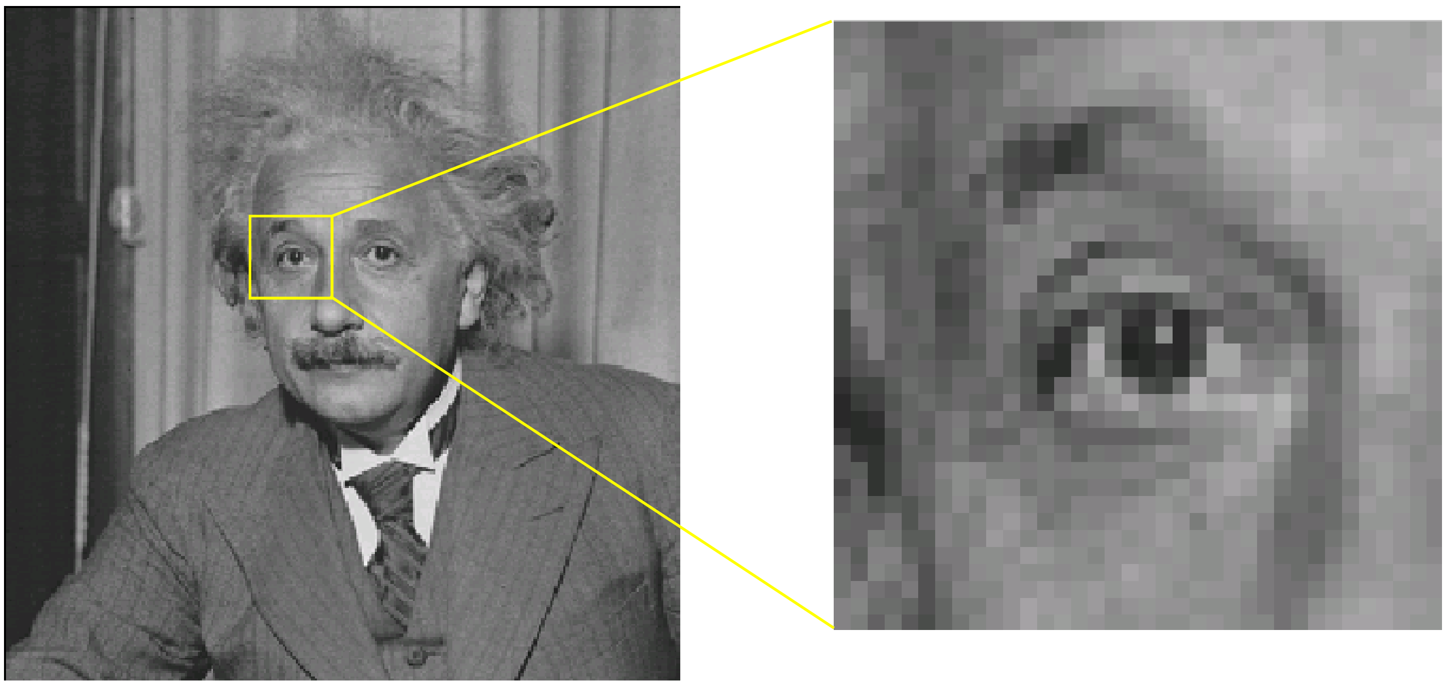
Patches that originate from a single surface



Local Image Patches

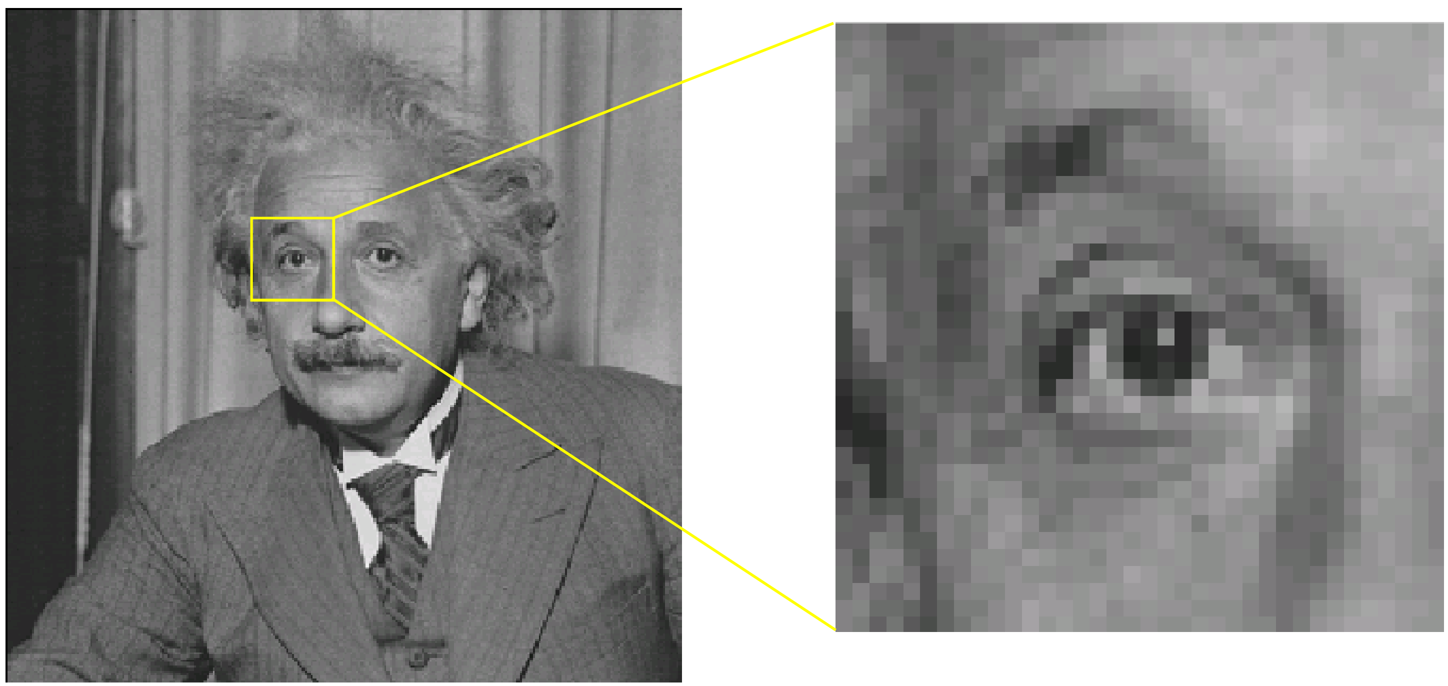
Qualitatively, we can think of many different types of patches in an image

Or patches with perceptually-significant "features"



Local Image Patches

When is a group of pixels considered a local patch?

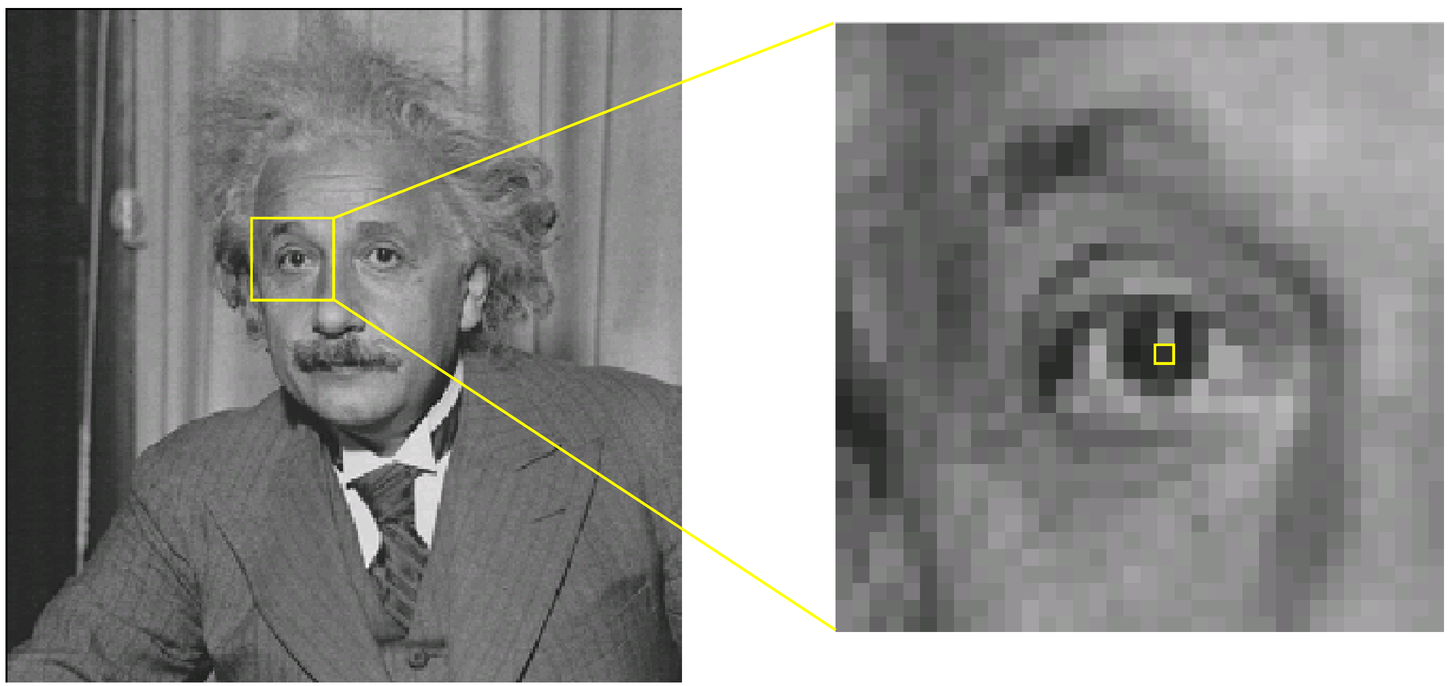


The notion of a patch is relative. It can be a single pixel

Local Image Patches

When is a group of pixels considered a local patch?

There is no answer to this question!

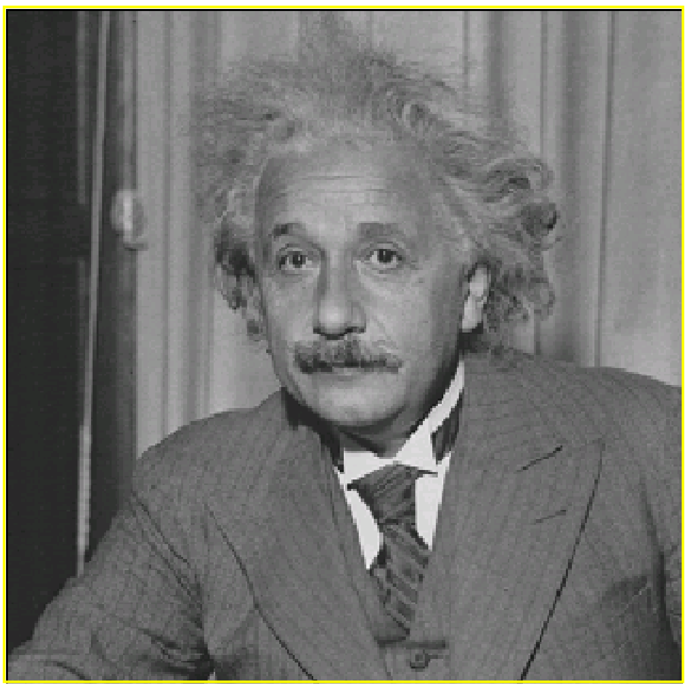


The notion of a patch is relative. It can be a single pixel

Local Image Patches

When is a group of pixels considered a local patch?

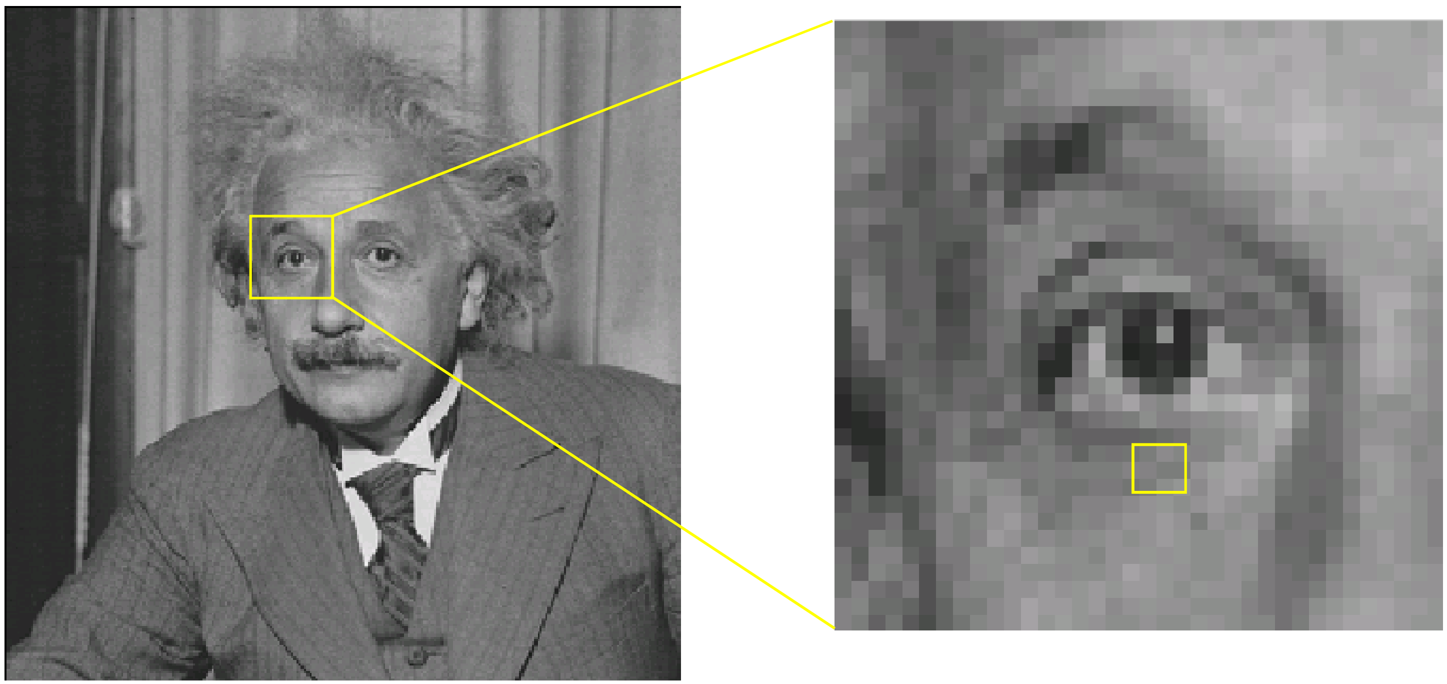
There is no answer to this question!



The notion of a patch is relative. It can be the entire image

Local Image Patches

We will begin with mathematical properties and methods that apply mostly to very small patches (e.g., 3x3)



… and eventually consider descriptions that apply to entire images

Topic 4:

Local analysis of image patches

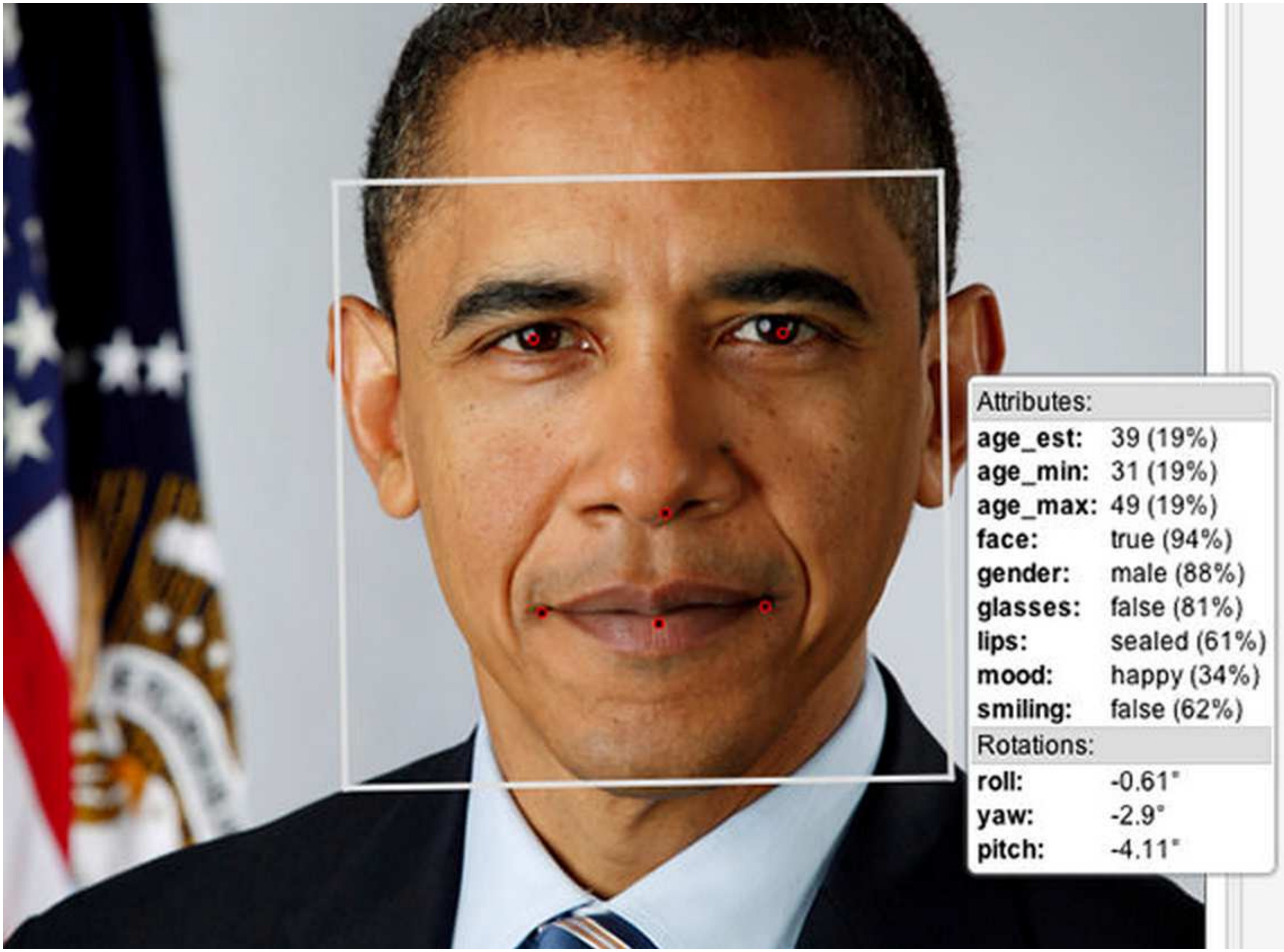
- What do we mean by an image “patch”?
- Applications of local image analysis
- Visualizing 1D and 2D intensity functions

Patches: Why Do We Care?

Many applications…

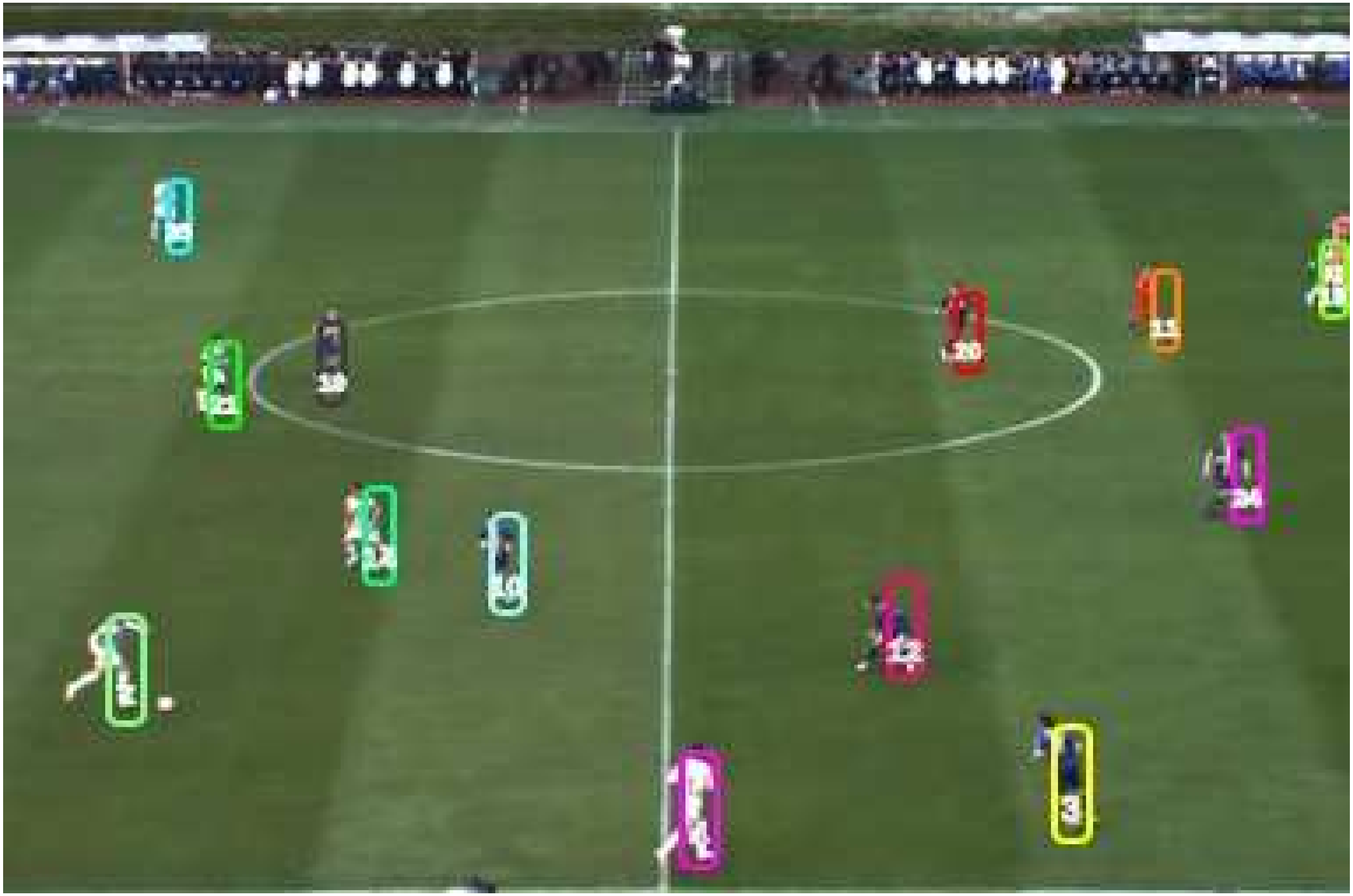
- Recognition
- Inspection
- Video-based tracking
- Special effects

Face Recognition and Analysis



http://petapixel.com/2012/03/30/facial-recognition-software-guesses-age-based-on-a-photo/

Tracking



M. Zervos, H. BenShitrit and P. Fua, Real time multi-object tracking using multiple cameras

Editing & Manipulating Photos

Object removal from a photo

Original

New

(Criminisi et al, CVPR 2003)

Editing & Manipulating Photos

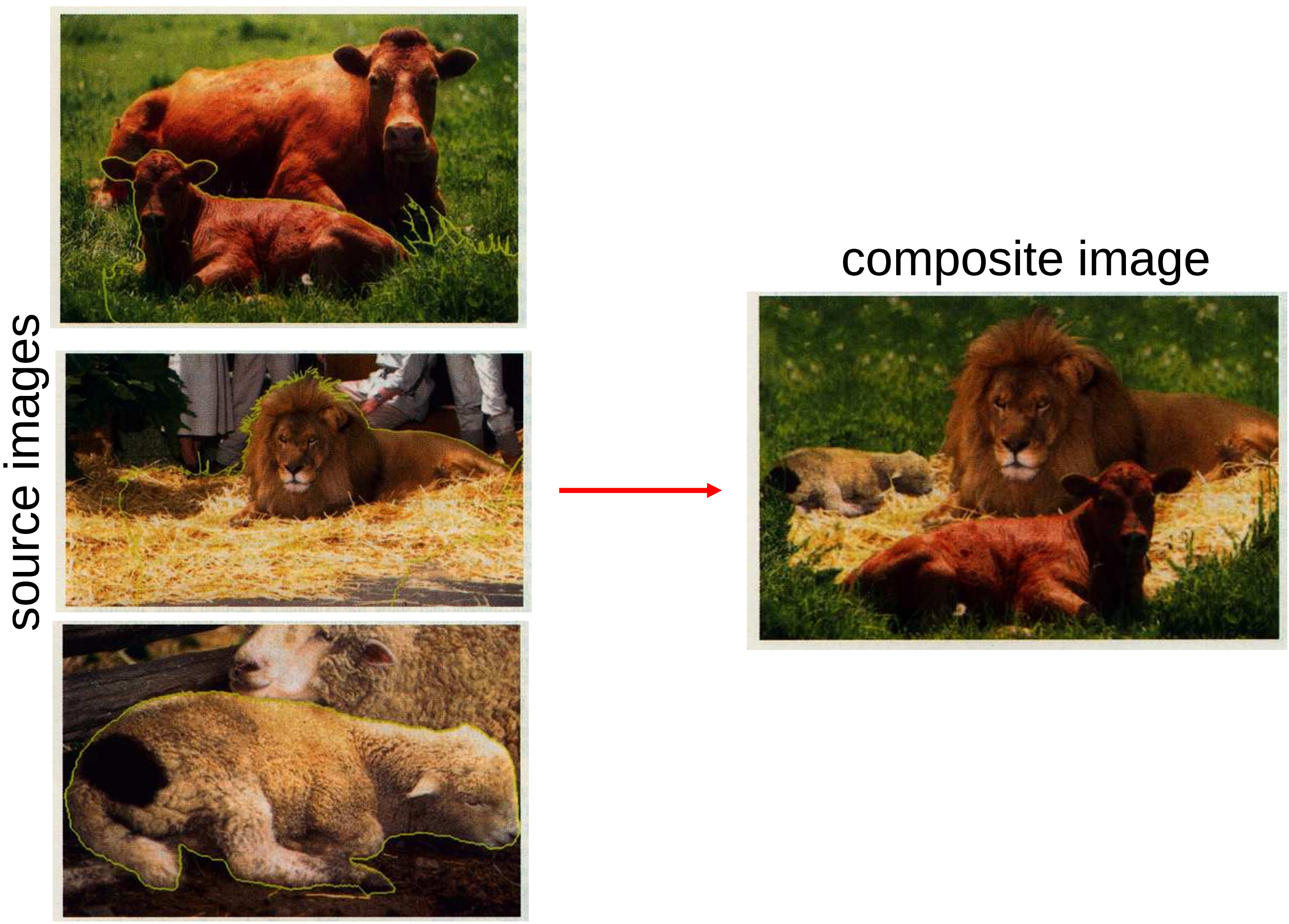
Colorization of black and white photos

Original (B&W) New (Color)

(Levin & Weiss, SIGGRAPH 2004)

Editing & Manipulating Photos

Scissoring objects from a photo



Giving Photos a "Painted" Look

From P. Litwinowicz's SIGGRAPH'97 paper "Processing Images and Videos for an Impressionist Effect"

Topic 4:

Local analysis of image patches

- What do we mean by an image “patch”?
- Applications of local image analysis
- Visualizing 1D and 2D image patches as intensity functions

Visualizing An Image as a Surface in 3D

Gray-scale image

y

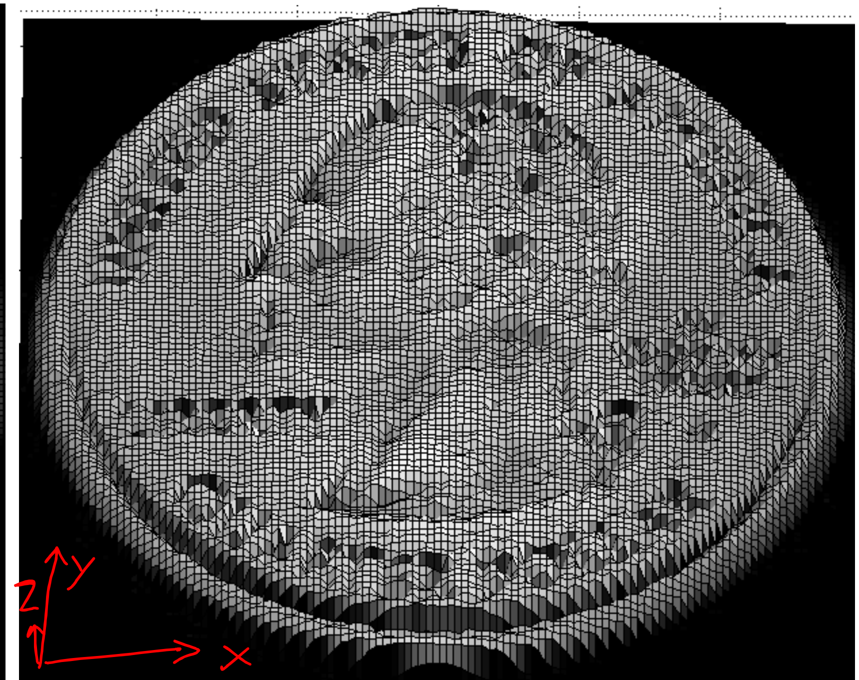
x

A gray-scale image is like a function I(x,y)

Image ⇔ Surface in 3D

Gray-scale image $I(x,y)$

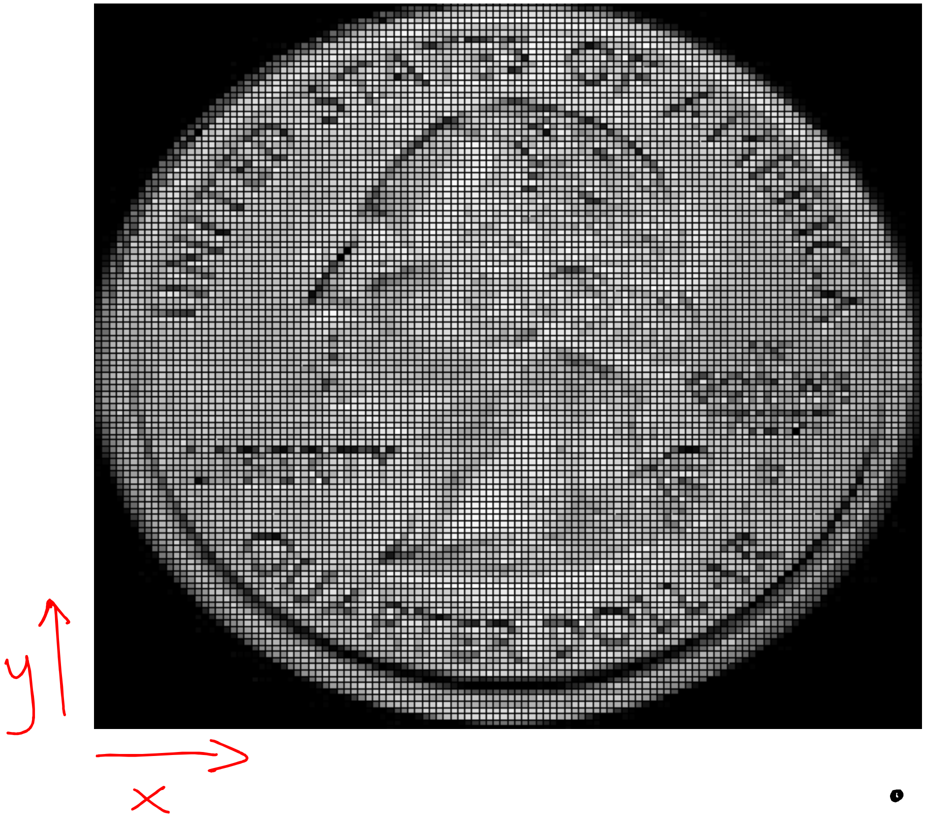
Surface $z = I(x,y)$



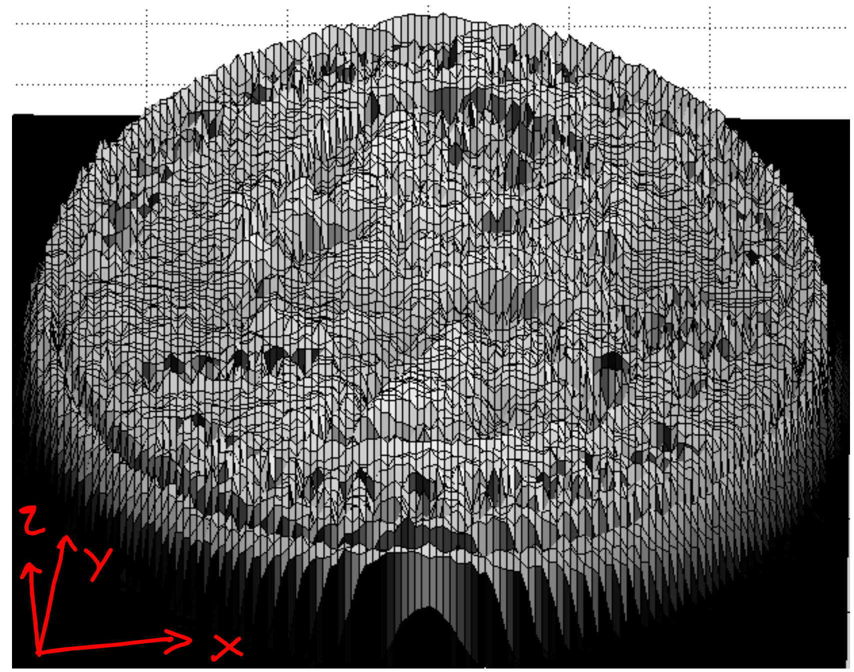
And we can visualize this function in 3D

Image ⇔ Surface in 3D

Gray-scale image $I(x,y)$



Surface $z = I(x,y)$



- The height of the surface at (x,y) is $I(x,y)$
- The surface contains point $(x, y, I(x,y))$

Image ⇔ Surface in 3D

Gray-scale image

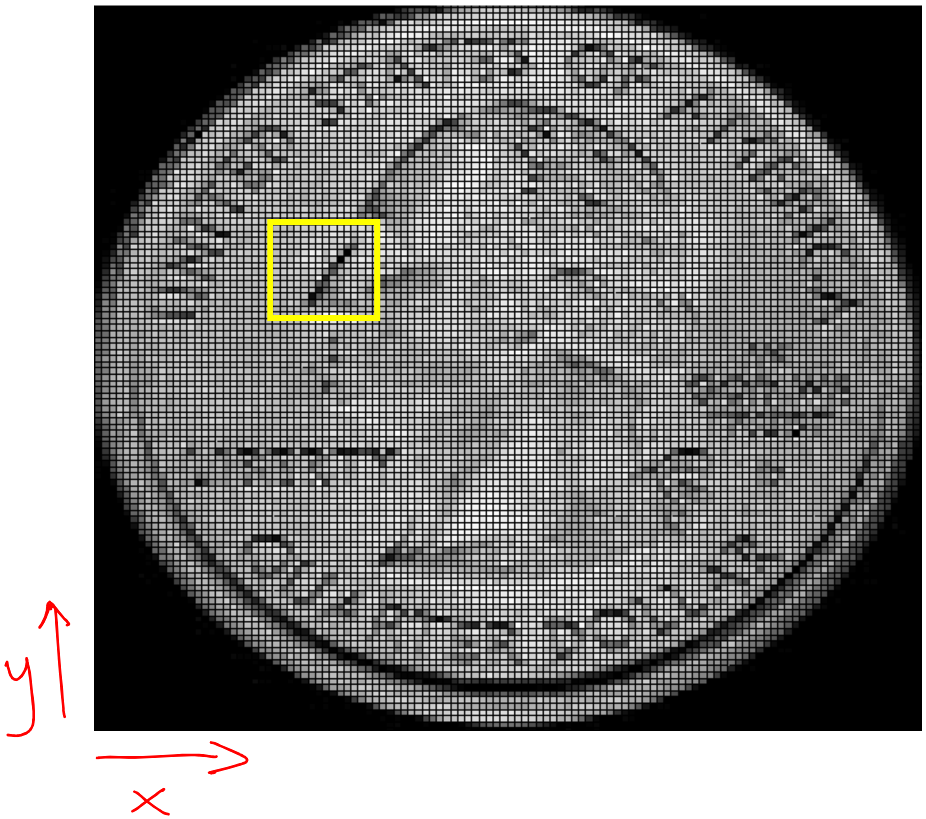
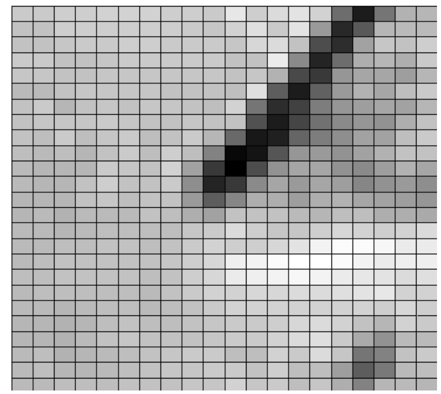


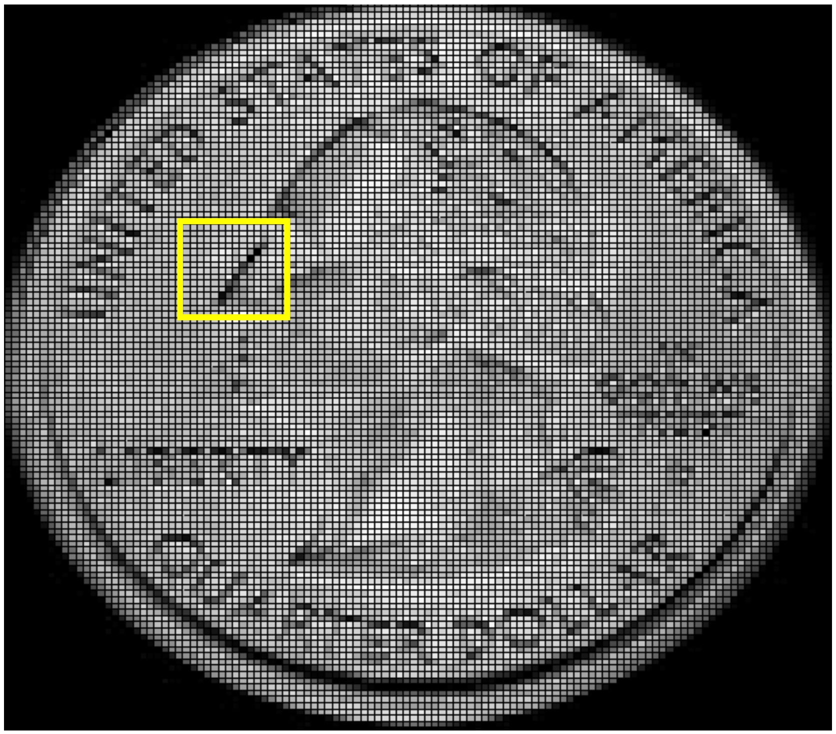
Image patch



The same
applies to image
patches

Image ⇔ Surface in 3D

Gray-scale image

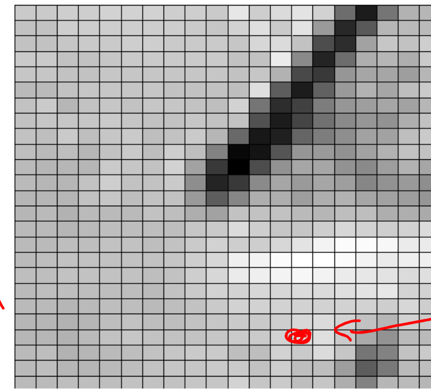


y

x

Patches have their own coordinate system.

Image patch

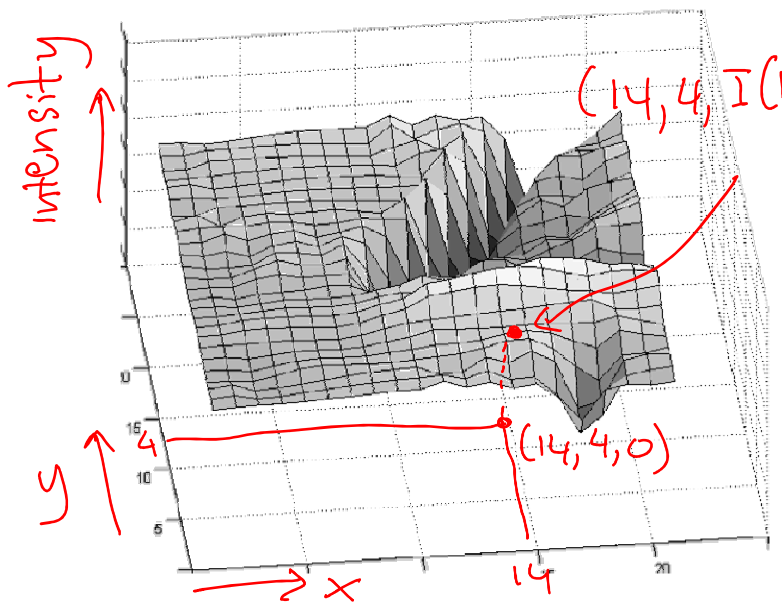


y

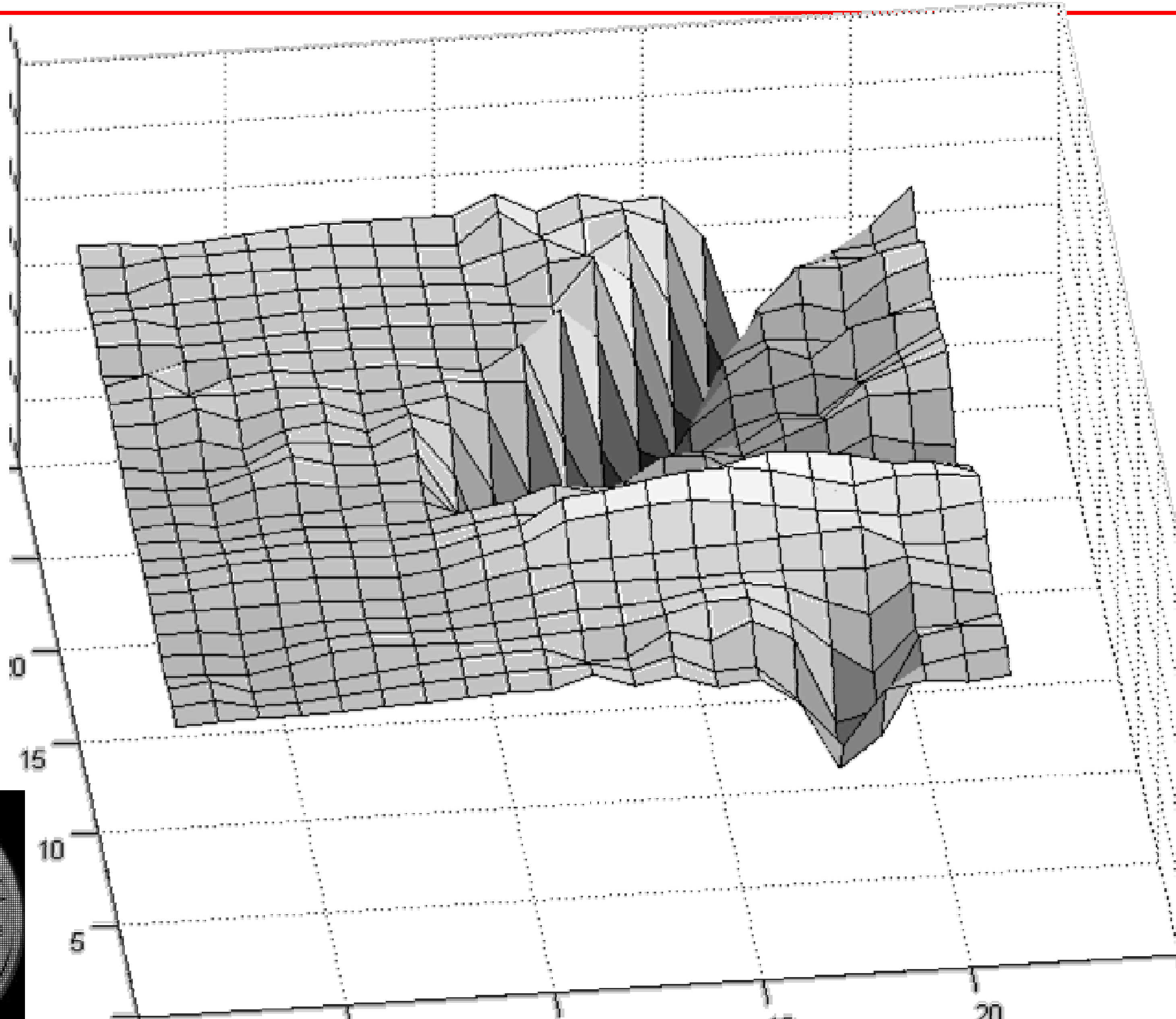
x

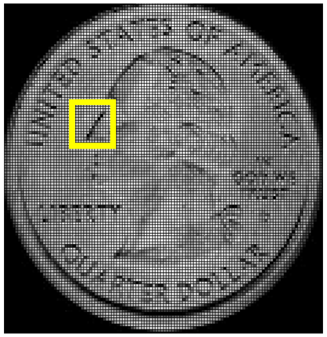
pixel (14, 4)

Surface patch $Z = I(x,y)$



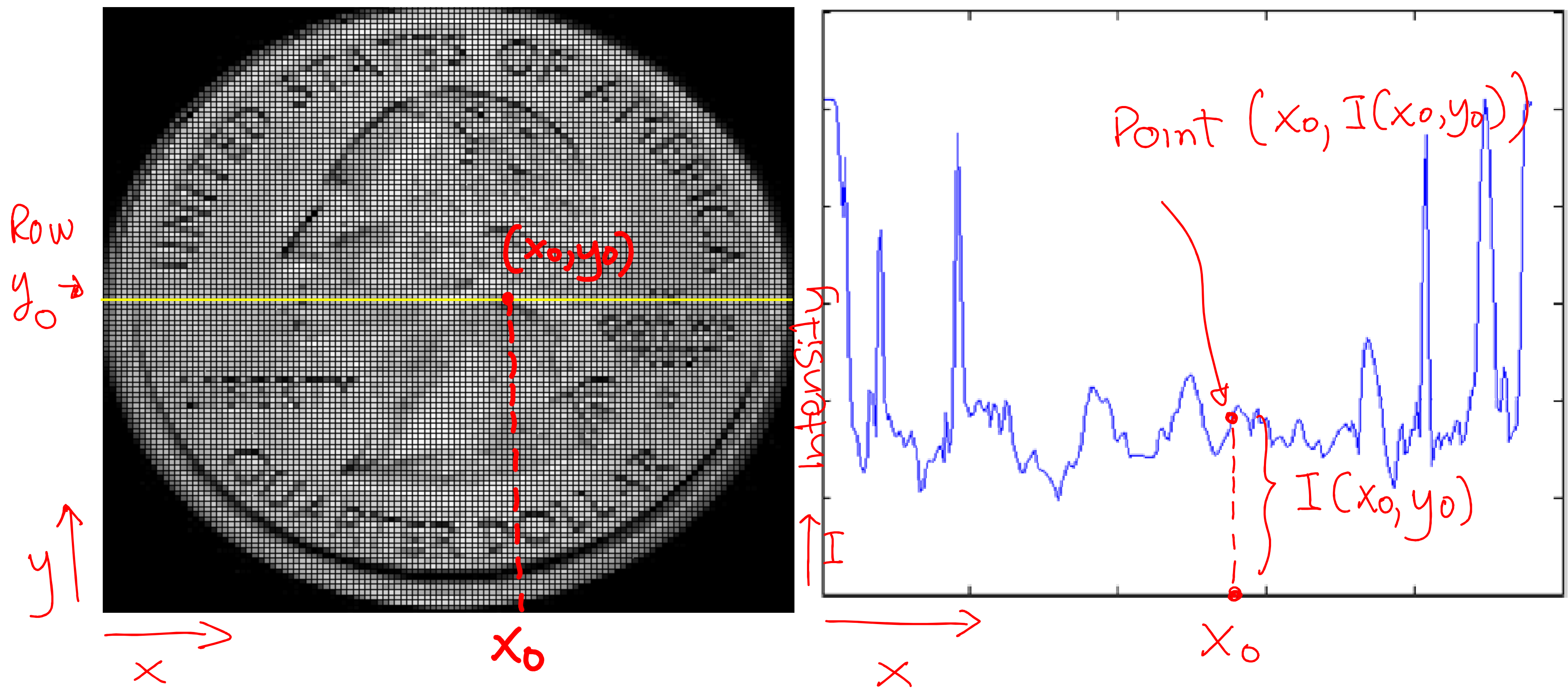
BTW, notice image noise





Visualizing a Row or Column as a Graph in 2D

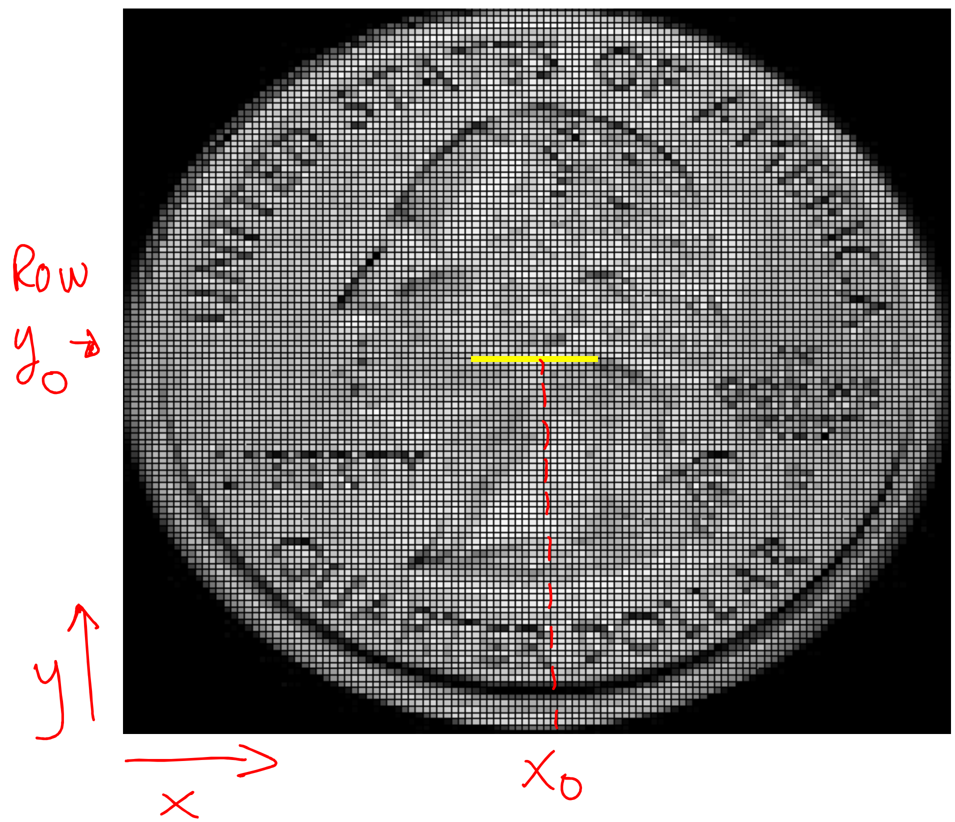
Gray-scale image | Graph in 2D



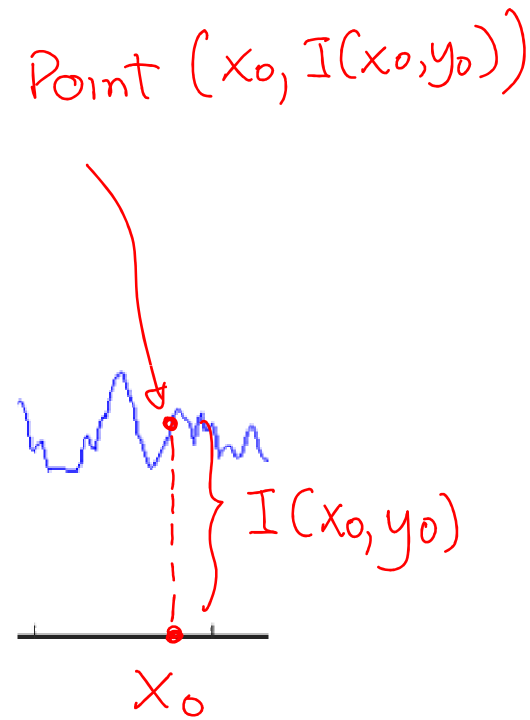
Another way of visualizing image data is as a graph in 2D

Image row or column ⇔ Graph in 2D

Gray-scale image



Graph in 2D



And of course, we can do this for a 1D patch.

Today we'll learn about

4.1. Today's lecture is about modeling image data taking into account more than one (potentially noisy) single pixel.

We will focus on 1D patches.

Methods include:

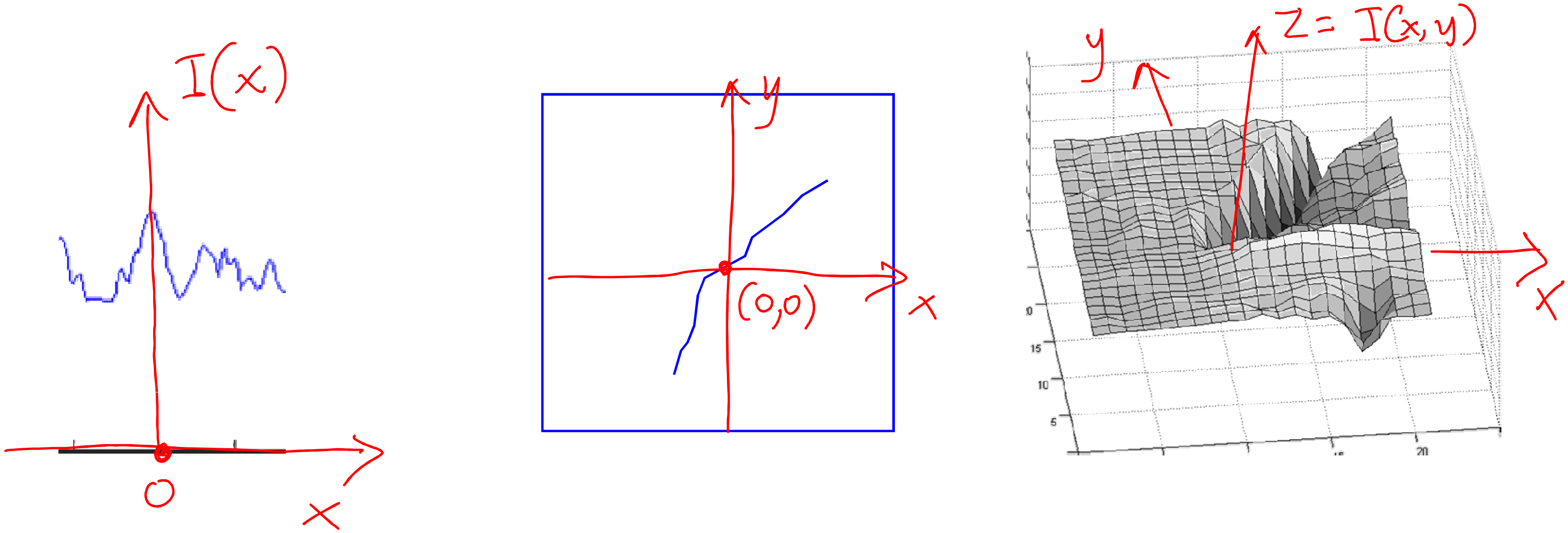
Computing derivatives of 1D patches using polynomial fitting via Least-squares, weighted least squares and RANSAC

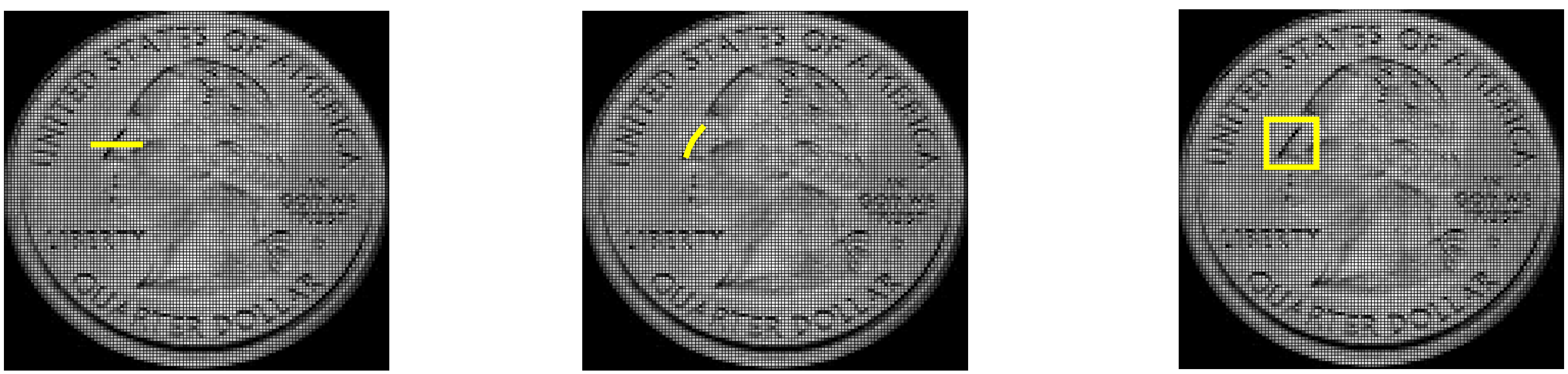
where are we, and what will come after?

- Subtopics:
  1. Local analysis of 1D image patches (today)
  2. Local analysis of 2D curve patches
  3. Local analysis of 2D image patches

Local Analysis of Image Patches: Outline

As graph in 2D | As curve in 2D | As surface in 3D





Topic 4:

Local analysis of image patches

- Subtopics:
  1. Local analysis of 1D image patches
  2. Local analysis of 2D curve patches
  3. Local analysis of 2D image patches

Topic 4.1:

Local analysis of 1D image patches

- Taylor series approximation of 1D intensity patches
    - Estimating derivatives of 1D intensity patches

      Least-squares fitting

      Weighted least-squares fitting

      Robust polynomial fitting: RANSAC

Topic 4.1:

Local analysis of 1D image patches

- Taylor series approximation of 1D intensity patches
  - Estimating derivatives of 1D intensity patches:

    Least-squares fitting

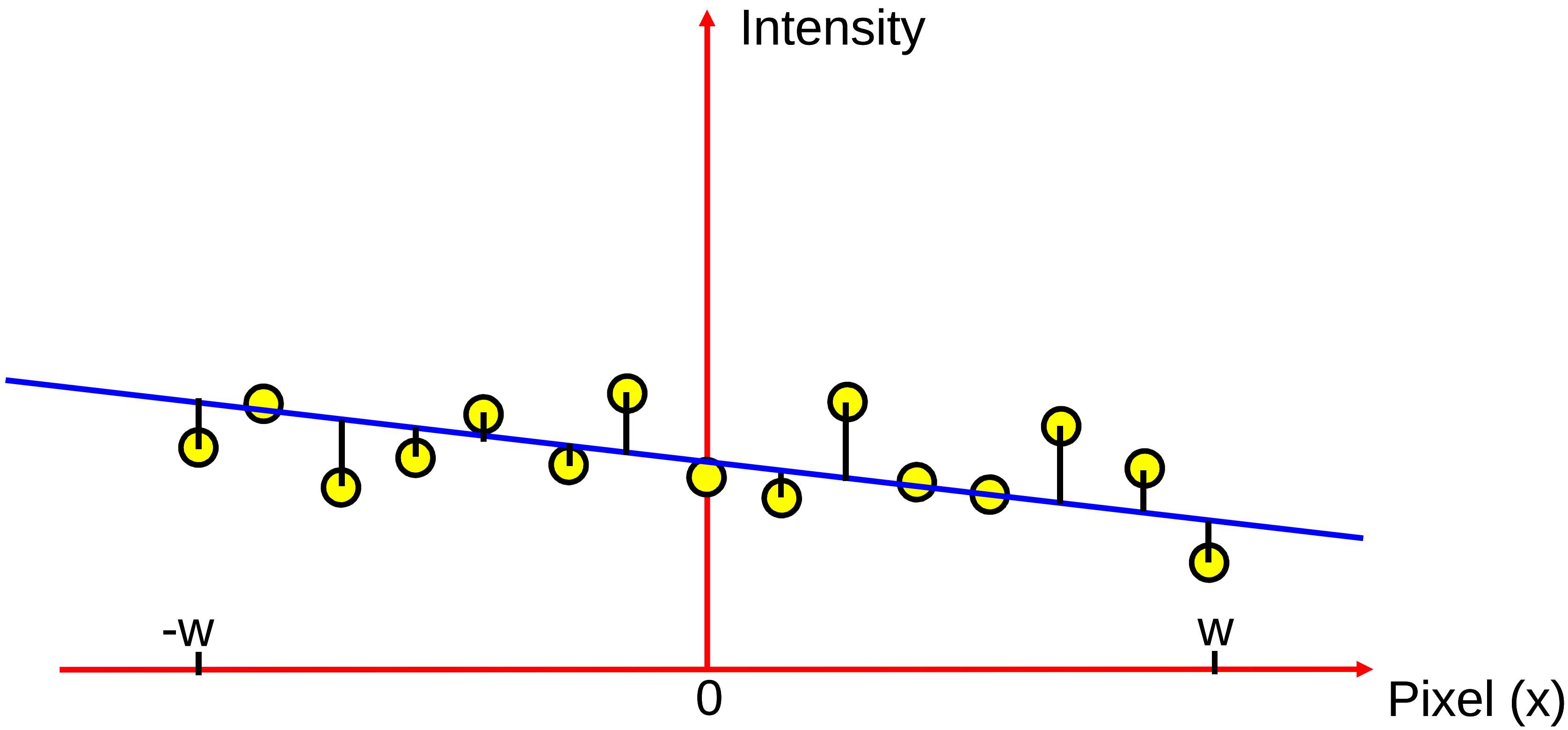
    Weighted least-squares fitting

    Robust polynomial fitting: RANSAC

Least-Squares Polynomial Fitting

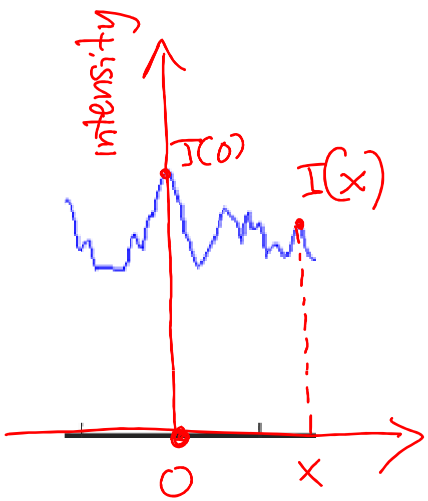
Taylor approximation: Fit a polynomial to the pixel intensities in a patch

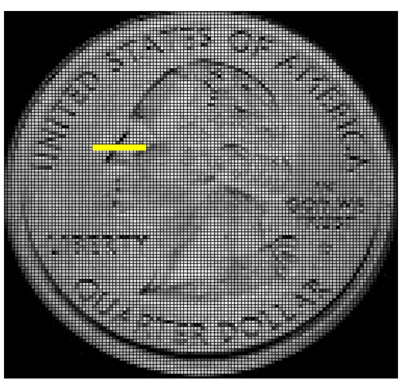
- All pixels contribute equally to estimate of derivative(s) at patch center (i.e., at x=0)



Taylor-Series Approximation of I(x)

As graph in 2D





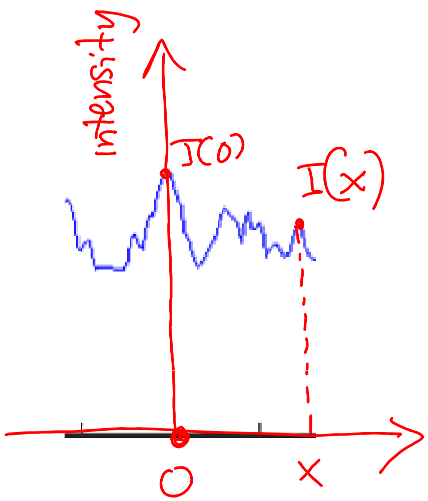
If we knew the derivatives of I(x) at x=0, we can approximate I(x) using the Taylor Series:

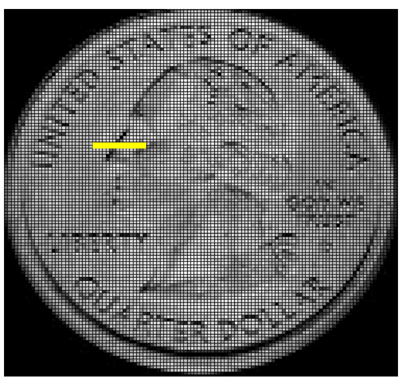
$$I(x) = \underbrace{\underbrace{\underbrace{I(0)}_{\text{0-th order approximation}} + x \cdot \frac{dI}{dx}(0)}_{\text{1st-order approx. of } I} + \frac{1}{2}x^2 \frac{d^2 I}{dx^2}(0)}_{\text{2nd-order approx of } I}$$

$$\underbrace{+ \cdots + \frac{1}{n!} x^n \frac{d^n I}{dx^n}(0)}_{\text{n-th order approx}} + R_{n+1}(x)$$

Taylor-Series Approximation of I(x)

As graph in 2D





If we knew the derivatives of I(x) at x=0, we can approximate I(x) using the Taylor Series:

$$I(x) = \underbrace{\underbrace{\underbrace{I(0)}_{\text{0-th order approximation}} + x \cdot \frac{dI}{dx}(0)}_{\text{1st-order approx. of } I} + \frac{1}{2} x^2 \frac{d^2 I}{dx^2}(0)}_{\text{2nd-order approx of } I}$$

$$\underbrace{+ \cdots + \frac{1}{n!} x^n \frac{d^n I}{dx^n}(0)}_{\text{n-th order approx}} + R_{n+1}(x)$$

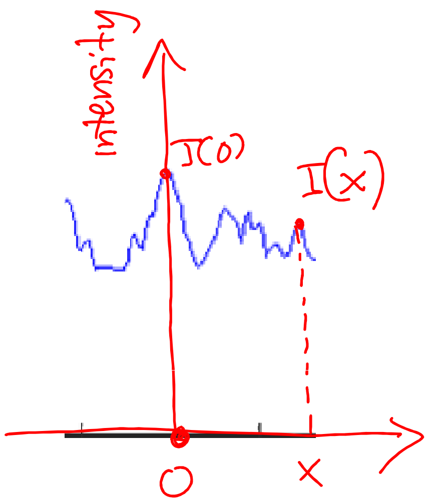
The residual $R_{n+1}(x)$ satisfies

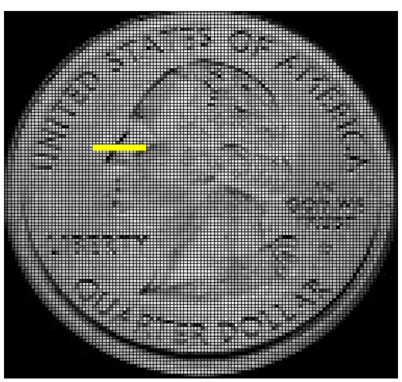
$$\lim_{x \to 0} R_{n+1}(x) = 0$$

?

Taylor-Series Approximation of I(x)

As graph in 2D





If we knew the derivatives of I(x) at x=0, we can approximate I(x) using the Taylor Series:

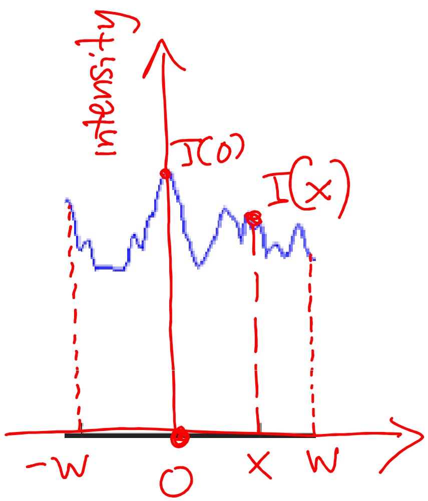
$$I(x) = \underbrace{\underbrace{\underbrace{I(0)}_{\text{0-th order approximation}} + x \cdot \frac{dI}{dx}(0)}_{\text{1st-order approx. of } I} + \frac{1}{2}x^2 \frac{d^2 I}{dx^2}(0)}_{\text{2nd-order approx of } I}$$

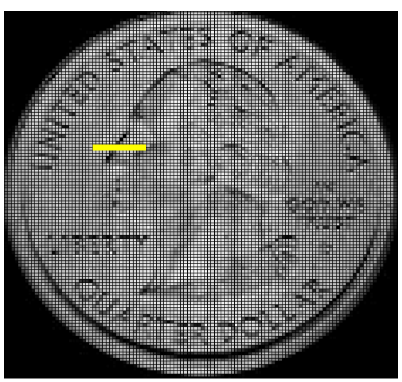
$$\underbrace{+ \cdots + \frac{1}{n!} x^n \frac{d^n I}{dx^n}(0)}_{\text{n-th order approx}} + R_{n+1}(x)$$

The approximation is best at the origin and degrades from there.

Taylor-Series Approximation of I(x)

As graph in 2D





The n-th order Taylor series expansion of I(x), near the patch center (x=0) can then be written in matrix form as:

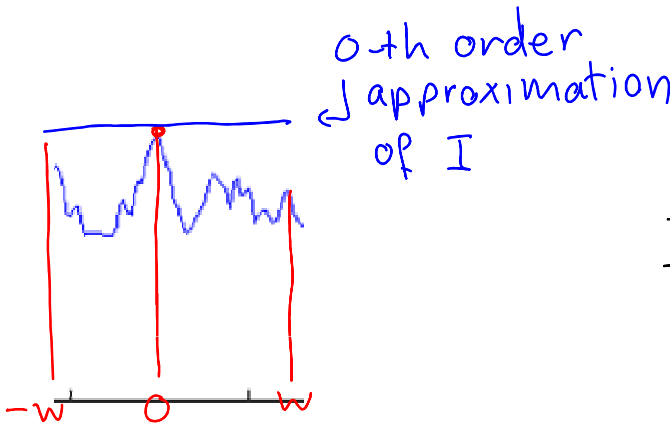
$$I(x) \simeq \begin{bmatrix} 1 & x & \frac{1}{2}x^2 & \frac{1}{6}x^3 & \cdots & \frac{1}{n!}x^n \end{bmatrix} \begin{bmatrix} I(0) \\ \frac{dI}{dx}(0) \\ \frac{dI}{dx^2}(0) \\ \vdots \\ \frac{d^nI}{dx^n}(0) \end{bmatrix}$$

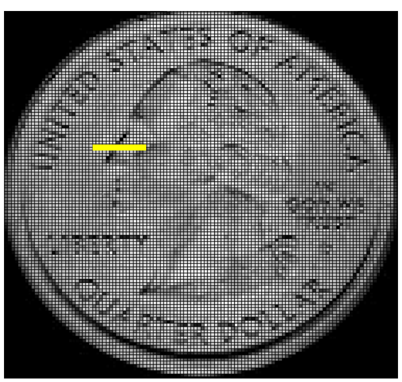
Note that an approximated value for I(x) will depend on n+1 coefficients: the intensity derivatives at I(0)

Taylor-Series Approximation of I(x)

As graph in 2D

Example: 0^{th} order approximation



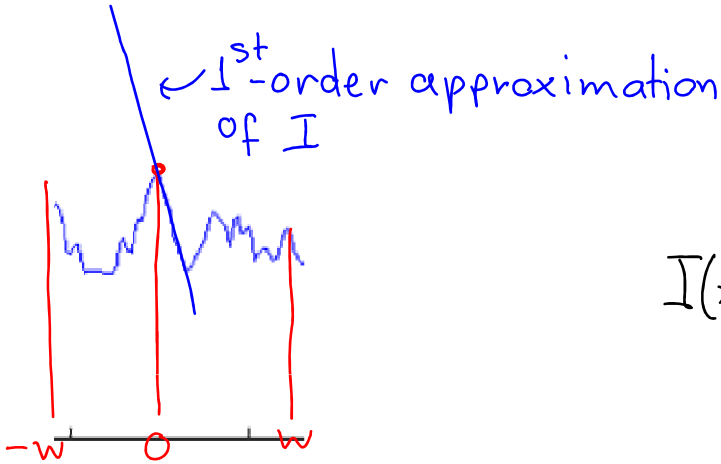


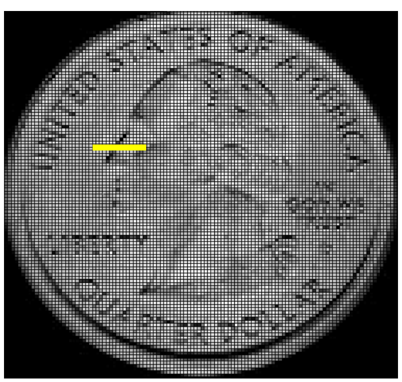
$$I(x) \approx \begin{bmatrix} 1 & & & & \end{bmatrix} \begin{bmatrix} I(0) \\ \frac{dI}{dx}(0) \\ \frac{dI}{dx^2}(0) \\ \vdots \\ \frac{d^nI}{dx^n}(0) \end{bmatrix}$$

$$I(x) = I(0)$$

Taylor-Series Approximation of I(x)

As graph in 2D





Example: 1st order approximation

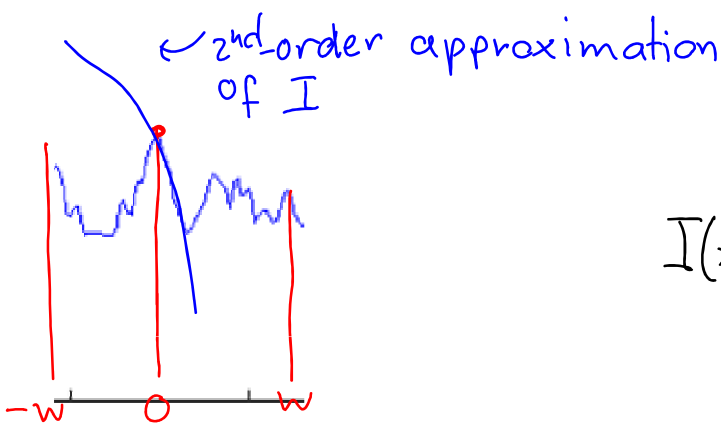
$$I(x) \approx \begin{bmatrix} 1 & x \end{bmatrix} \begin{bmatrix} I(0) \\ \frac{dI}{dx}(0) \end{bmatrix}$$

$$I(x) = I(0) + x \cdot \frac{dI}{dx}(0)$$

Taylor-Series Approximation of I(x)

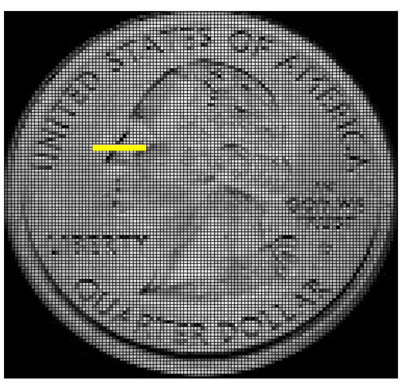
As graph in 2D

Example: 2nd order approximation



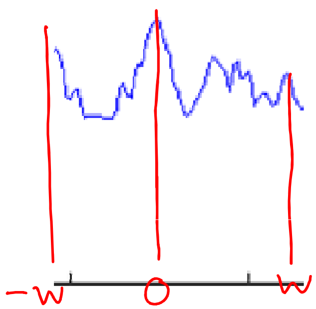
$$I(x) \approx \begin{bmatrix} 1 & x & \frac{1}{2}x^2 \end{bmatrix} \begin{bmatrix} I(0) \\ \frac{dI}{dx}(0) \\ \frac{dI}{dx^2}(0) \end{bmatrix}$$

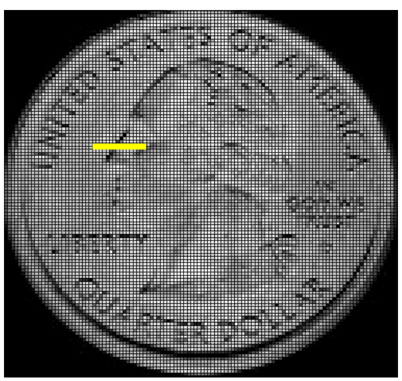
$$I(x) = I(0) + x \cdot \frac{dI}{dx}(0) + \frac{x^2}{2}\frac{d^2I}{dx^2}(0)$$



Taylor-Series Approximation of I(x)

As graph in 2D





And so on...

$$I(x) \approx \begin{bmatrix} 1 & x & \frac{1}{2}x^2 & \frac{1}{6}x^3 & \cdots & \frac{1}{n!}x^n \end{bmatrix} \begin{bmatrix} I(0) \\ \frac{dI}{dx}(0) \\ \frac{dI}{dx^2}(0) \\ \vdots \\ \frac{d^nI}{dx^n}(0) \end{bmatrix}$$

Taylor-Series Approximation of I(x)

But do we know the derivatives?

$$\begin{bmatrix} I(0) \\ \frac{dI}{dx}(0) \\ \frac{dI}{dx^2}(0) \\ \vdots \\ \frac{d^nI}{dx^n}(0) \end{bmatrix}$$

Taylor-Series Approximation of I(x)

$$\begin{bmatrix} I(0) \\ \frac{dI}{dx}(0) \\ \frac{dI}{dx^2}(0) \\ \vdots \\ \frac{d^n I}{dx^n}(0) \end{bmatrix}$$

But do we know the derivatives?

No, but we can estimate them!

Taylor-Series Approximation of I(x)

And can we estimate them for the entire row?

Taylor-Series Approximation of I(x)

And can we estimate them for the entire row?
Yes, but pixel by pixel.

In fact…

Applying the same operation on multiple patches

A “sliding window” algorithm is a common approach to patch-based operations

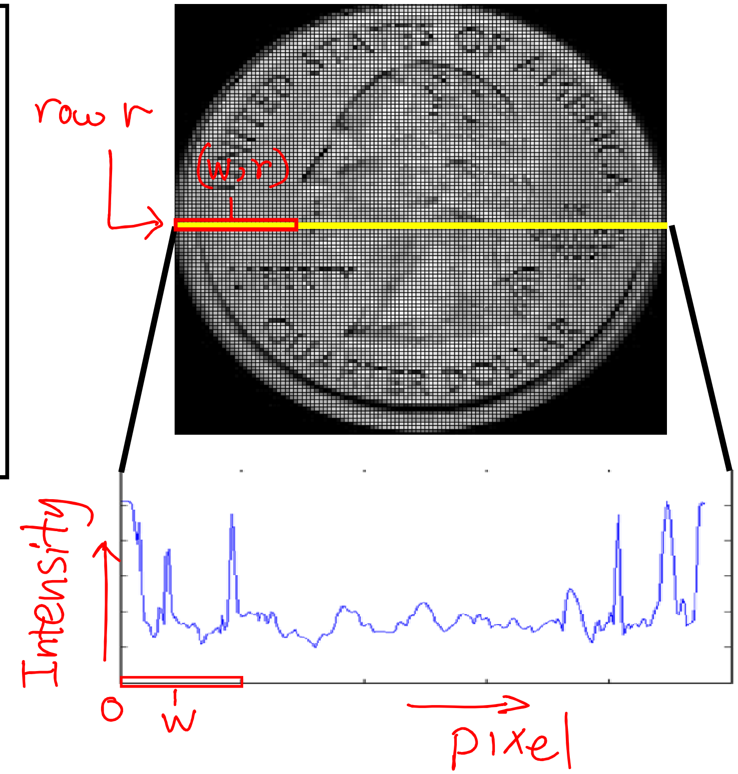
The algorithm goes as follows:

Applying the same operation on multiple patches

A "sliding window" algorithm is a common approach to patch-based operations

The algorithm goes as follows:

1. Define a "pixel window" using a window size and a window center.

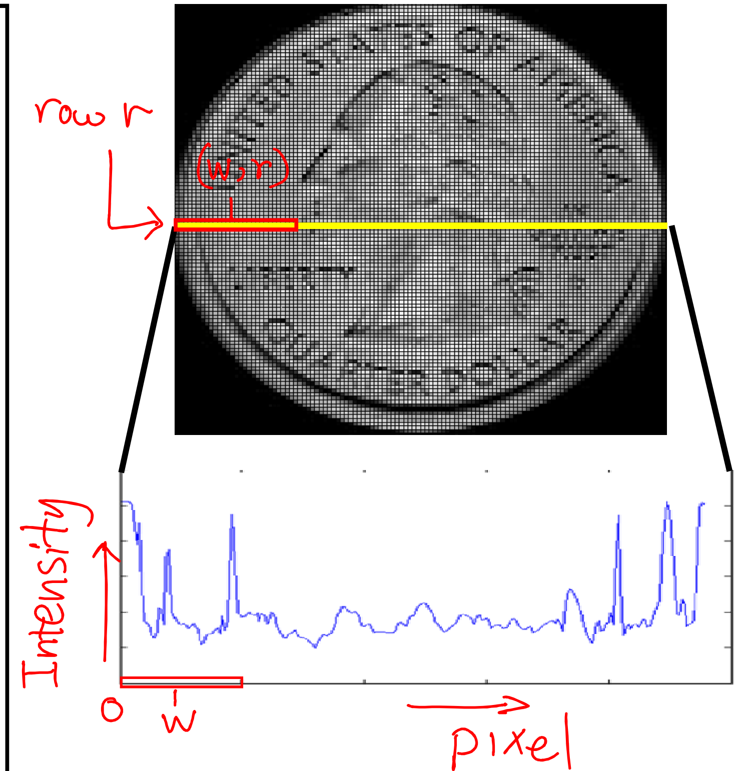


Applying the same operation on multiple patches

A "sliding window" algorithm is a common approach to patch-based operations

The algorithm goes as follows:

1. Define a "pixel window" using a window size and a window center.
2. Apply whatever operation in mind to that patch
3. Move the window center one pixel to define a new window
4. Repeat steps 1-3

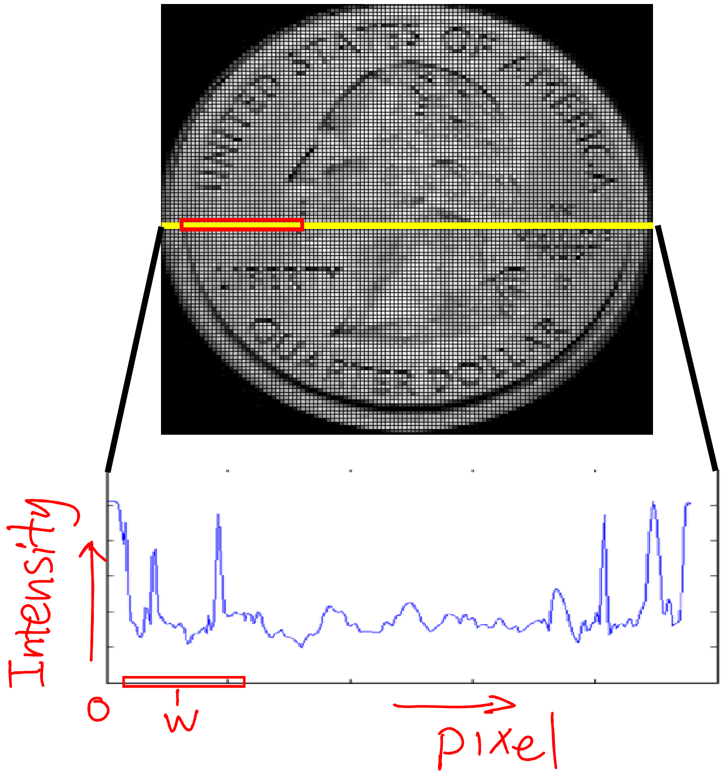


Applying the same operation on multiple patches

A "sliding window" algorithm is a common approach to patch-based operations

The algorithm goes as follows:

1. Define a "pixel window" using a window size and a window center.
2. Apply whatever operation in mind to that patch
3. Move the window center one pixel to define a new window
4. Repeat steps 1-3

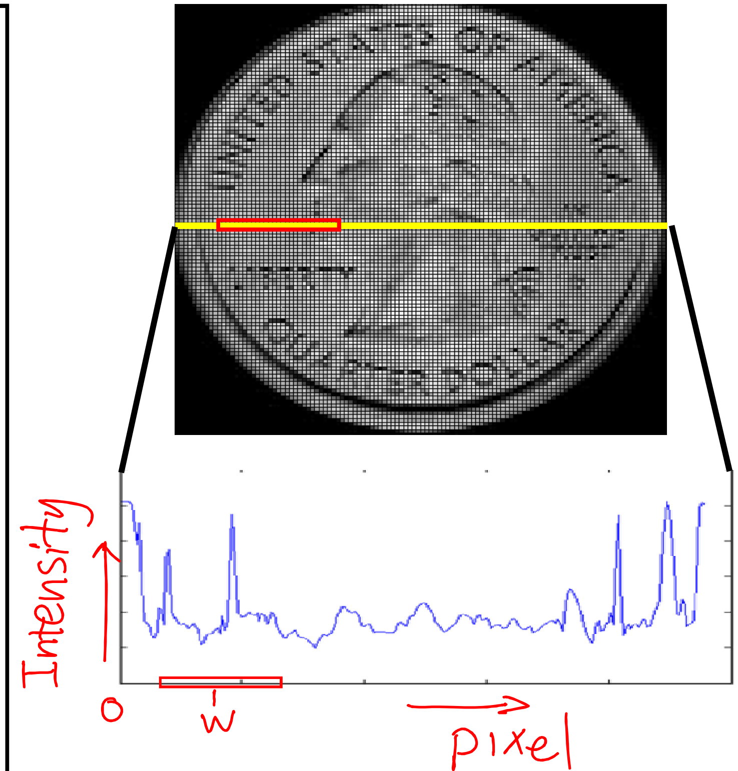


Applying the same operation on multiple patches

A "sliding window" algorithm is a common approach to patch-based operations

The algorithm goes as follows:

1. Define a "pixel window" using a window size and a window center.
2. Apply whatever operation in mind to that patch
3. Move the window center one pixel to define a new window
4. Repeat steps 1-3

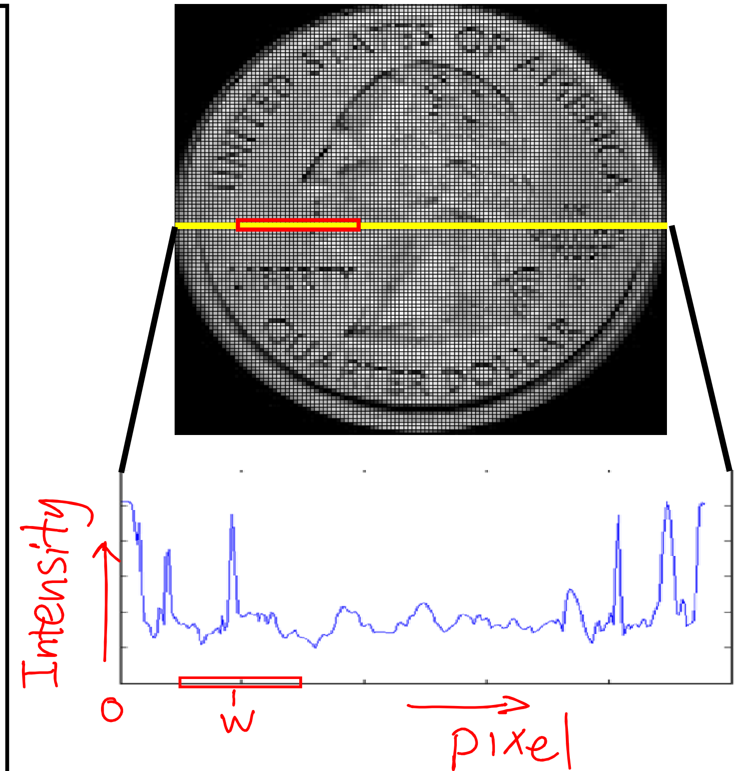


Applying the same operation on multiple patches

A "sliding window" algorithm is a common approach to patch-based operations

The algorithm goes as follows:

1. Define a "pixel window" using a window size and a window center.
2. Apply whatever operation in mind to that patch
3. Move the window center one pixel to define a new window
4. Repeat steps 1-3

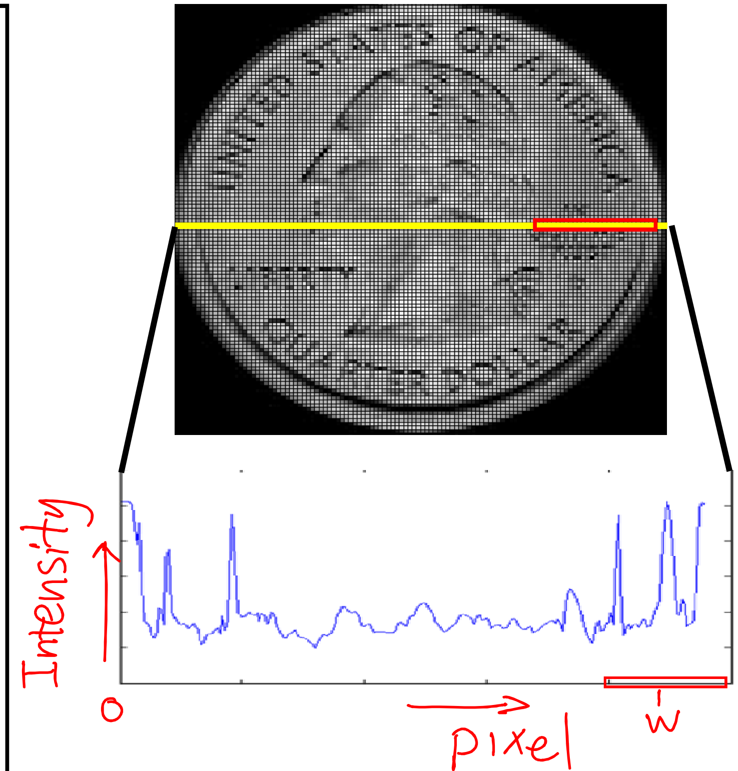


Applying the same operation on multiple patches

A "sliding window" algorithm is a common approach to patch-based operations

The algorithm goes as follows:

1. Define a "pixel window" using a window size and a window center.
2. Apply whatever operation in mind to that patch
3. Move the window center one pixel to define a new window
4. Repeat steps 1-3



Estimating Derivatives For Image Row r

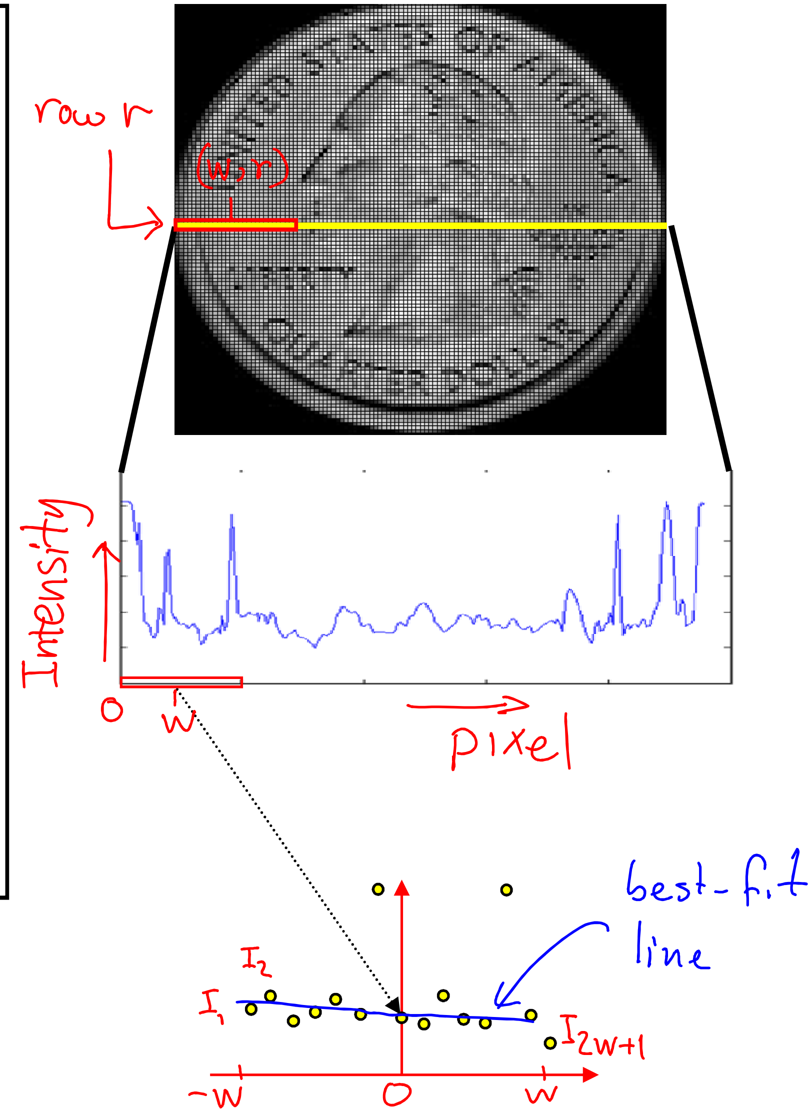
"Sliding window" algorithm:

- Define a "pixel window" centered at pixel (w,r)
- Fit n-degree poly to window's intensities (usually n=1 or 2)
- Assign the poly's derivatives at x=0 to pixel at window's center
- "Slide" window one pixel over, so that it is centered at pixel (w+1,r)
- Repeat 1-4 until window reaches right image border

Estimating Derivatives For Image Row r

"Sliding window" algorithm:

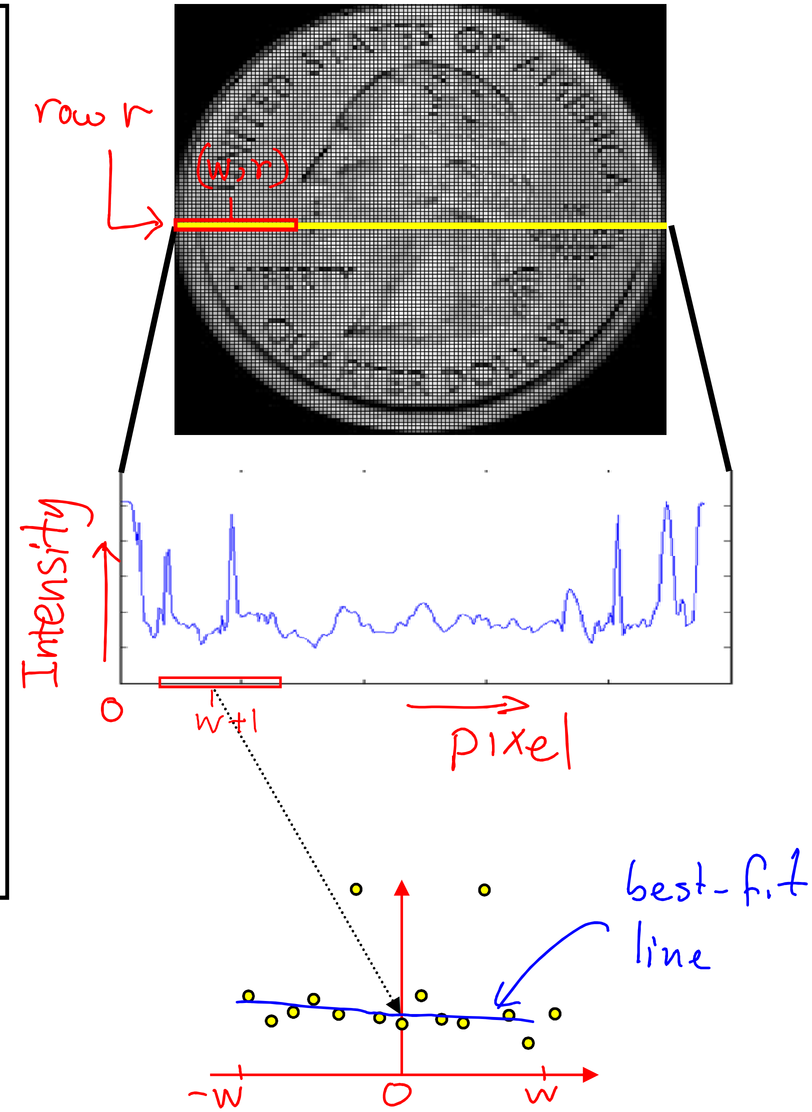
- Define a "pixel window" centered at pixel (w,r)
- Fit n-degree poly to window's intensities (usually n=1 or 2)
- Assign the poly's derivatives at x=0 to pixel at window's center
- "Slide" window one pixel over, so that it is centered at pixel (w+1,r)
- Repeat 1-4 until window reaches right image border



Estimating Derivatives For Image Row r

"Sliding window" algorithm:

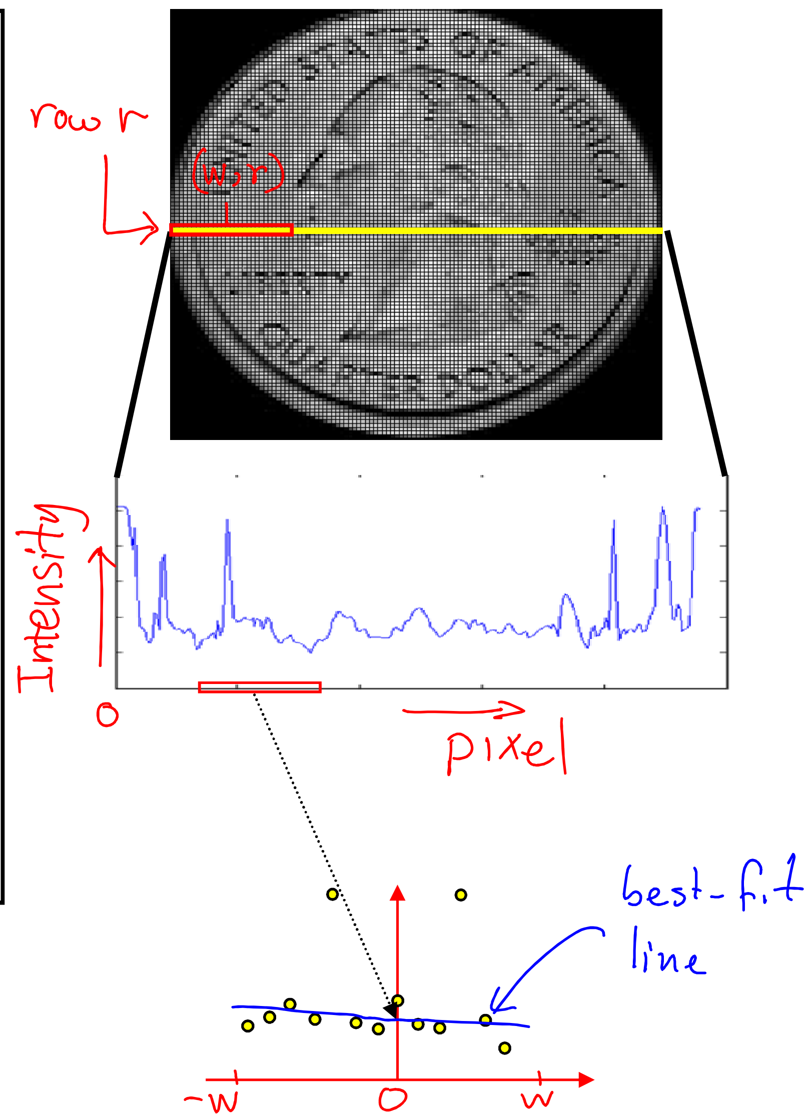
- Define a "pixel window" centered at pixel (w,r)
- Fit n-degree poly to window's intensities (usually n=1 or 2)
- Assign the poly's derivatives at x=0 to pixel at window's center
- "Slide" window one pixel over, so that it is centered at pixel (w+1,r)
- Repeat 1-4 until window reaches right image border



Estimating Derivatives For Image Row r

"Sliding window" algorithm:

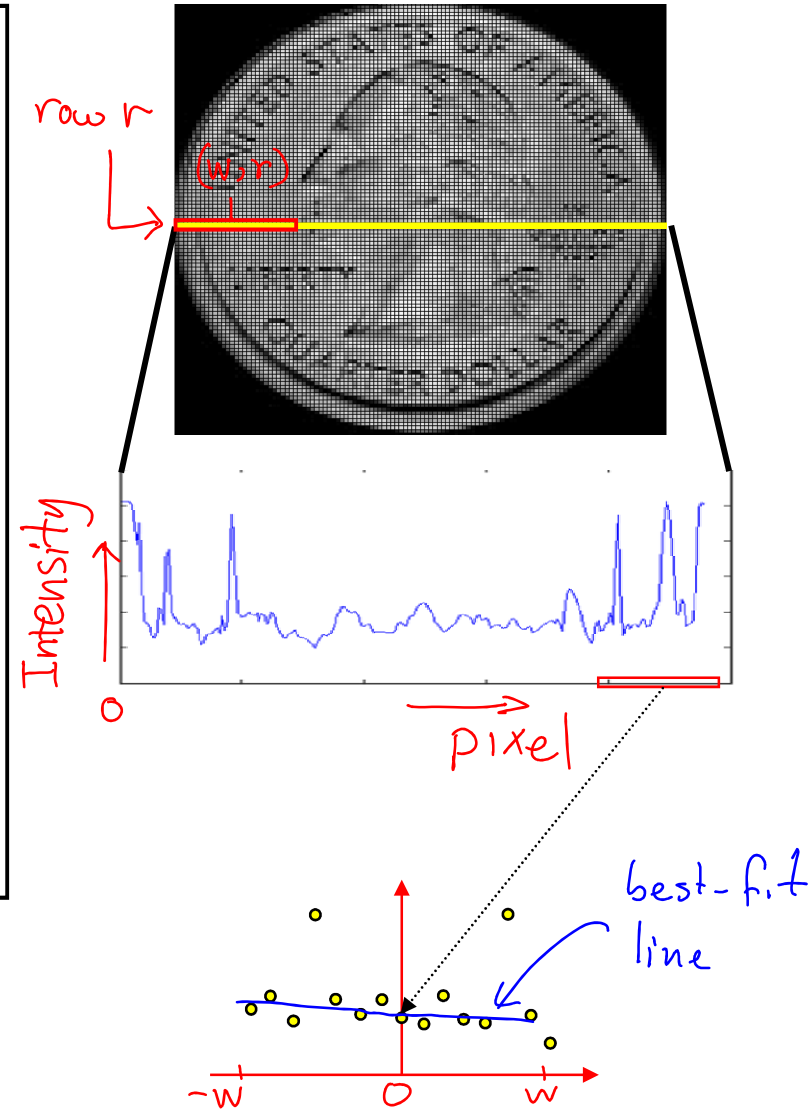
- Define a "pixel window" centered at pixel (w,r)
- Fit n-degree poly to window's intensities (usually n=1 or 2)
- Assign the poly's derivatives at x=0 to pixel at window's center
- "Slide" window one pixel over, so that it is centered at pixel (w+1,r)
- Repeat 1-4 until window reaches right image border



Estimating Derivatives For Image Row r

"Sliding window" algorithm:

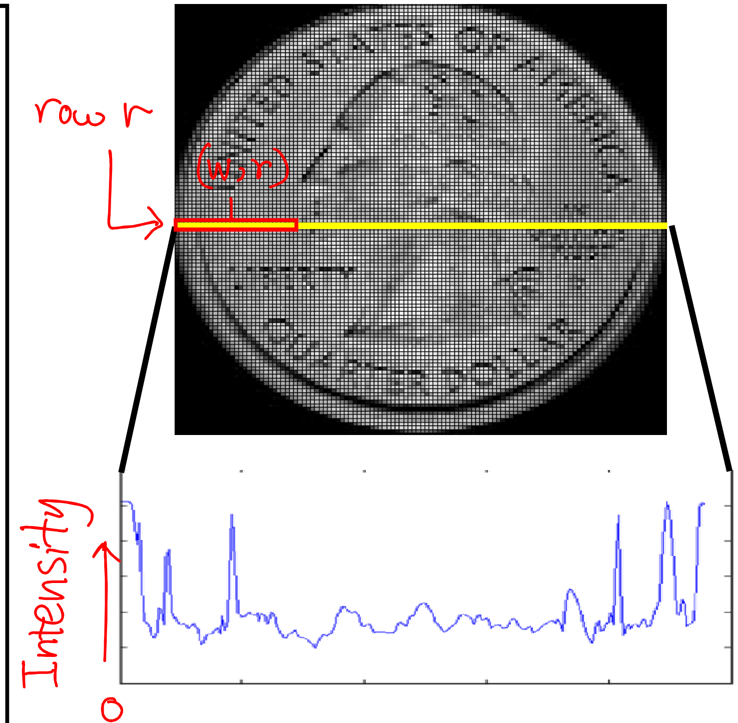
- Define a "pixel window" centered at pixel (w,r)
- Fit n-degree poly to window's intensities (usually n=1 or 2)
- Assign the poly's derivatives at x=0 to pixel at window's center
- "Slide" window one pixel over, so that it is centered at pixel (w+1,r)
- Repeat 1-4 until window reaches right image border



Estimating Derivatives For Image Row r

"Sliding window" algorithm:

- Define a "pixel window" centered at pixel (w,r)
- Fit n-degree poly to window's intensities (usually n=1 or 2)
- Assign the poly's derivatives at x=0 to pixel at window's center
- "Slide" window one pixel over, so that it is centered at pixel (w+1,r)
- Repeat 1-4 until window reaches right image border



$$\underbrace{\frac{dI}{dx}(0)}_{\text{patch}} \longleftrightarrow \underbrace{\frac{dI}{dx}(w)}_{\text{image}}$$

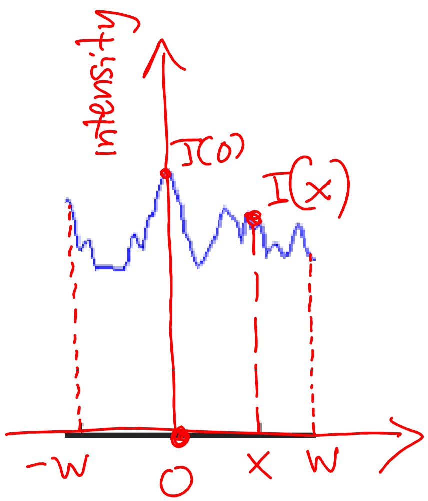
Topic 4.1:

Local analysis of 1D image patches

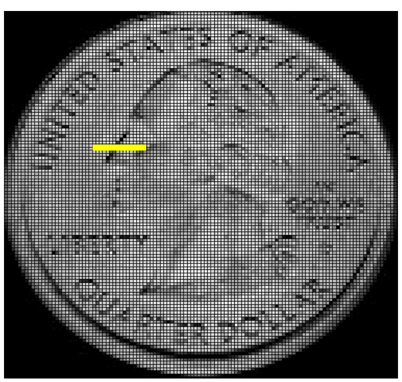
- Taylor series approximation of 1D intensity patches
- Estimating derivatives of 1D intensity patches:
  - Least-squares fitting
  - Weighted least-squares fitting
  - Robust polynomial fitting: RANSAC

Taylor-Series Approximation of I(x)

As graph in 2D

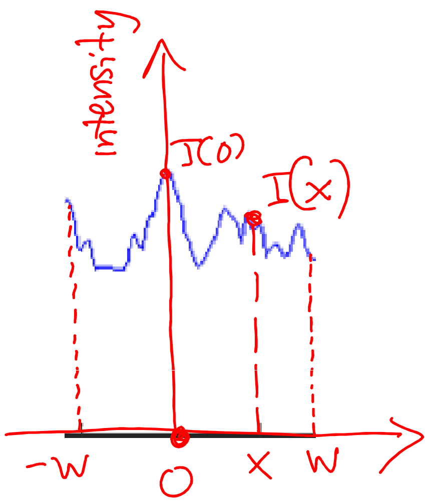


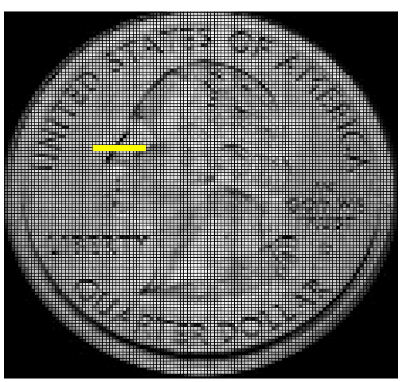
How to estimate the Taylor series approximation from image data?



Taylor-Series Approximation of I(x)

As graph in 2D





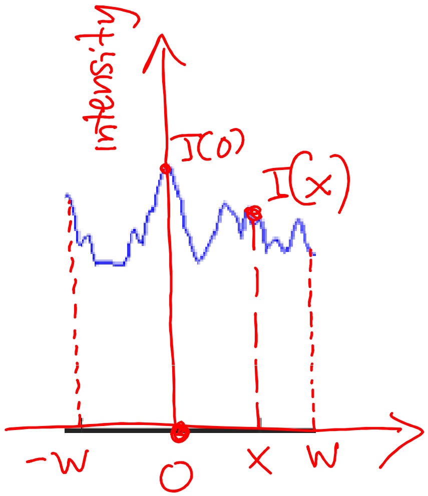
Surprise!

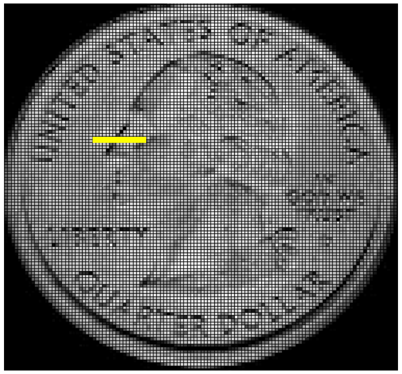
The n^{th} degree Taylor approximation can be estimated using a linear system of equations (which we can represent in matrix form).

This is Least Squares!

Taylor-Series Approximation of I(x)

As graph in 2D





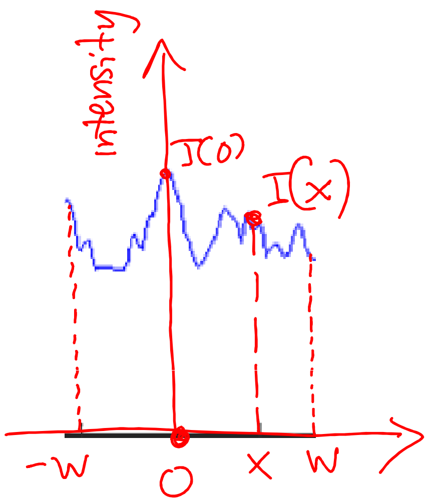
We know that the Taylor series is:

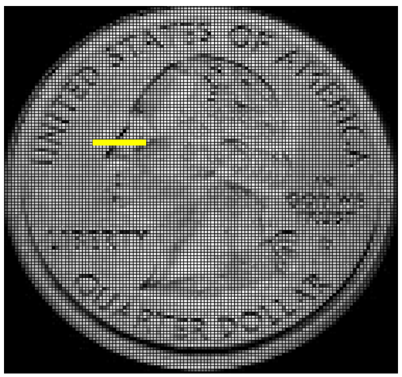
$$I(x) = I(0) + x \cdot \frac{dI}{dx}(0) + \frac{1}{2}x^2 \frac{d^2 I}{dx^2}(0)$$

$$+ \cdots + \frac{1}{n!} x^n \frac{d^n I}{dx^n}(0) + R_{n+1}(x)$$

Taylor-Series Approximation of I(x)

As graph in 2D





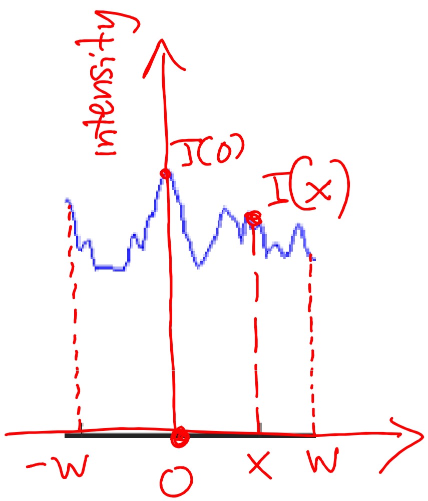
We know that the Taylor series is:

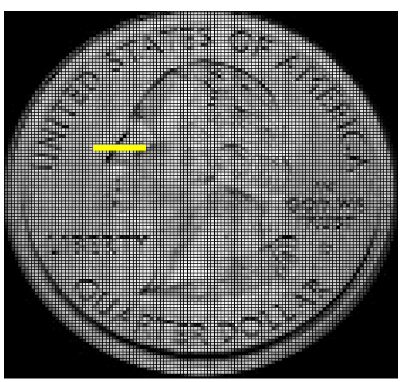
$$I(x) = I(0) + x \cdot \frac{dI}{dx}(0) + \frac{1}{2} x^2 \frac{d^2 I}{dx^2}(0) + \cdots + \frac{1}{n!} x^n \frac{d^n I}{dx^n}(0)$$

The derivatives are unknown

Taylor-Series Approximation of I(x)

As graph in 2D





We know that the Taylor series is:

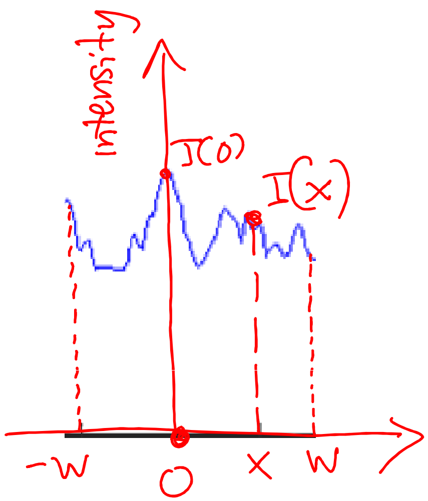
$$I(x) = I(0) + x\frac{dI}{dx}(0) + \frac{1}{2}x^2\frac{d^2I}{dx^2}(0)$$

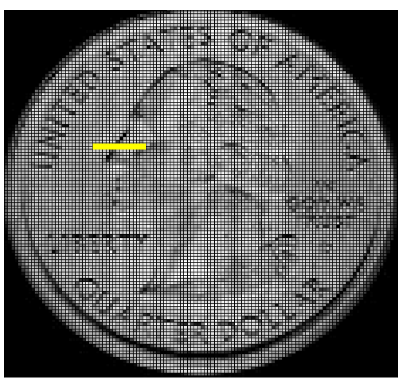
$$+ \cdots + \frac{1}{n!}x^n\frac{d^nI}{dx^n}(0)$$

But the coefficients are known

Taylor-Series Approximation of I(x)

As graph in 2D



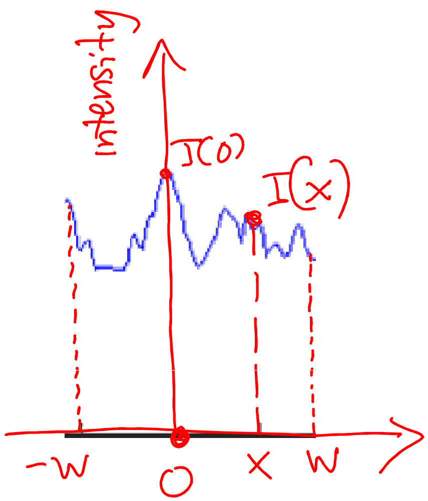


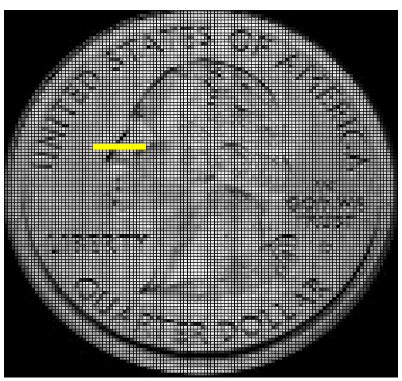
The n-th order Taylor series expansion of I(x), near the patch center (x=0) can then be written in matrix form as:

$$I(x) \simeq \begin{bmatrix} 1 & x & \frac{1}{2}x^2 & \frac{1}{6}x^3 & \cdots & \frac{1}{n!}x^n \end{bmatrix} \begin{bmatrix} I(0) \\ \frac{dI}{dx}(0) \\ \frac{dI}{dx^2}(0) \\ \vdots \\ \frac{d^nI}{dx^n}(0) \end{bmatrix}$$

Taylor-Series Approximation of I(x)

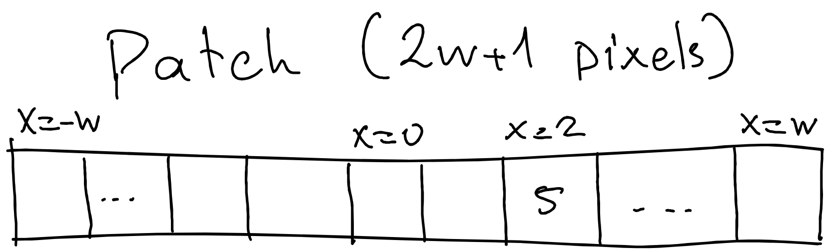
As graph in 2D





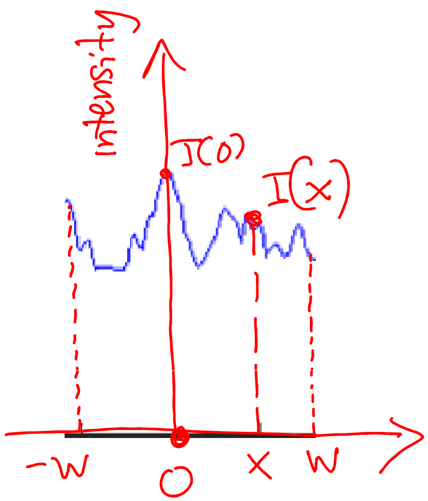
The n-th order Taylor series expansion of I(x), near the patch center (x=0) can then be written in matrix form as:

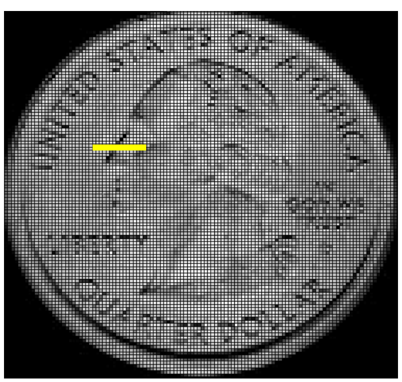
$$I(x) \simeq \begin{bmatrix} 1 & x & \frac{1}{2}x^2 & \frac{1}{6}x^3 & \cdots & \frac{1}{n!}x^n \end{bmatrix} \begin{bmatrix} I(0) \\ \frac{dI}{dx}(0) \\ \frac{dI}{dx^2}(0) \\ \vdots \\ \frac{d^nI}{dx^n}(0) \end{bmatrix}$$



Taylor-Series Approximation of I(x)

As graph in 2D





The n-th order Taylor series expansion of I(x), near the patch center (x=0) can then be written in matrix form as:

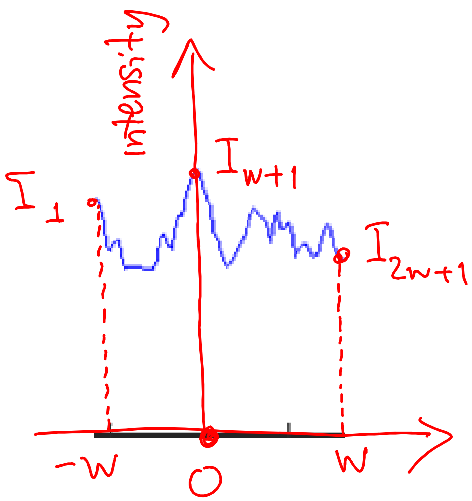
$$I(x) \approx \begin{bmatrix} 1 & x & \frac{1}{2}x^2 & \frac{1}{6}x^3 & \cdots & \frac{1}{n!}x^n \end{bmatrix} \begin{bmatrix} I(0) \\ \frac{dI}{dx}(0) \\ \frac{dI}{dx^2}(0) \\ \vdots \\ \frac{d^nI}{dx^n}(0) \end{bmatrix}$$

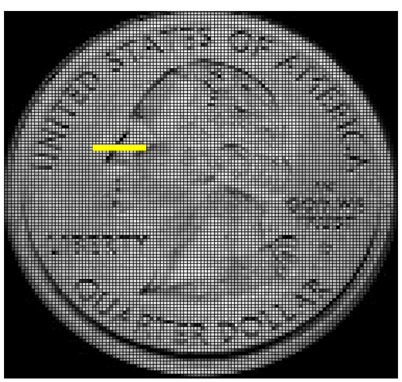
for $x \in (-w, w)$

2w+1 equations to estimate n+1 unknowns

Least-Squares Polynomial Fitting of I(x)

As graph in 2D





The equations define the system:

$$I_{(2w+1)\times 1} = X_{(2w+1)\times(n+1)}\, d_{(n+1)\times 1}$$

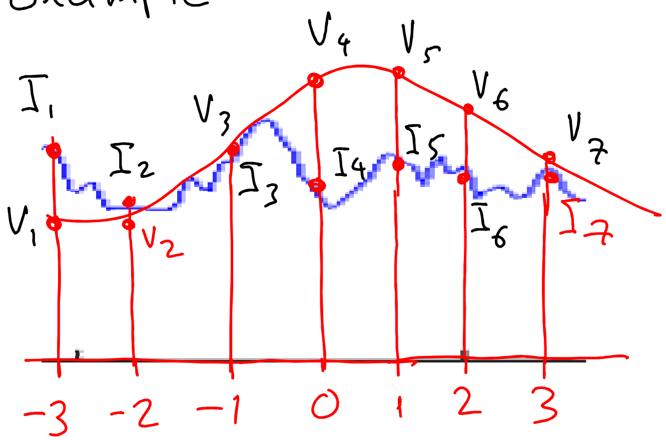
intensities (known) | positions (known) | derivatives (unknown)

Solving linear system in terms of d minimizes the "fit error"

$$\| I - Xd \|^2$$

Least-Squares Polynomial Fitting of I(x)

Example



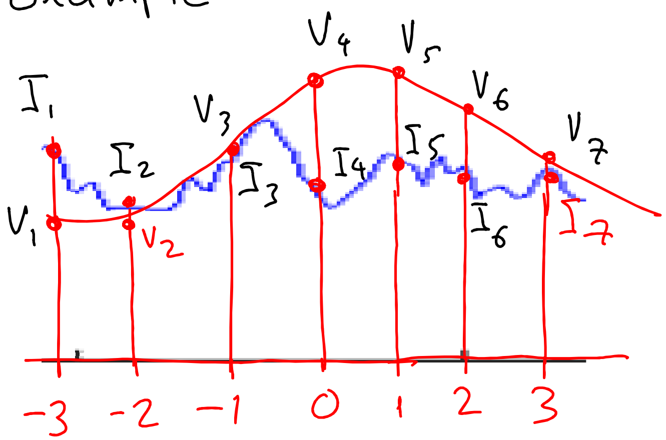
$$\| I - Xd \|^2$$

Solution d is called a least-squares fit

We could then do v=Xd to get an estimate for all pixels in the patch in (-w, ..., 0, ..., w)

Least-Squares Polynomial Fitting of I(x)

Example



$$\| I - Xd \|^2$$

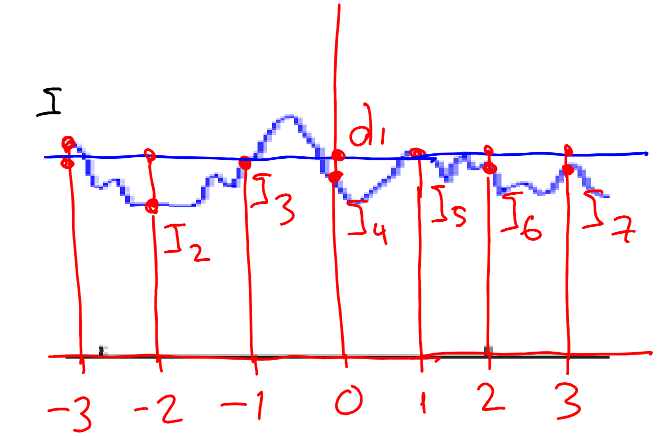
Solution d is called a <u>least-squares fit</u>

- This solution minimizes the 2-norm (i.e. the length) of the error vector $(I-v)$:

$$\left(\sum_{i=1}^{2w+1} (I_i - v_i)^2 \right)^{1/2}$$

0^{th}-Order (Constant) Estimation of I(x)

Special case:



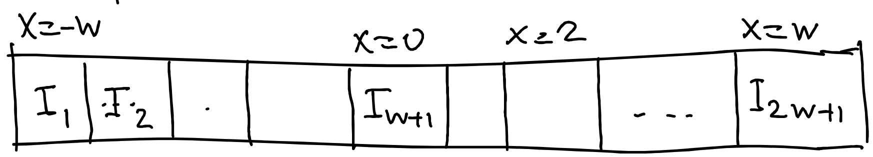
- Solution minimizes

$$\sum_{i=1}^{2w+1} (I_i - d_1)^2$$

- Solution is the mean intensity of the patch:

$$d_1 = \frac{1}{2w+1} \sum_{i=1}^{2w+1} I_i$$

Patch (2w+1 pixels)



$$I_{(2w+1)\times 1} = X_{(2w+1)\times 1}\, d_{1\times 1}$$

intensities (known)

positions (known)

one unknown (equal to I(0))

$$\begin{bmatrix} I_1 \\ I_2 \\ \vdots \\ I_n \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} [d_1]$$

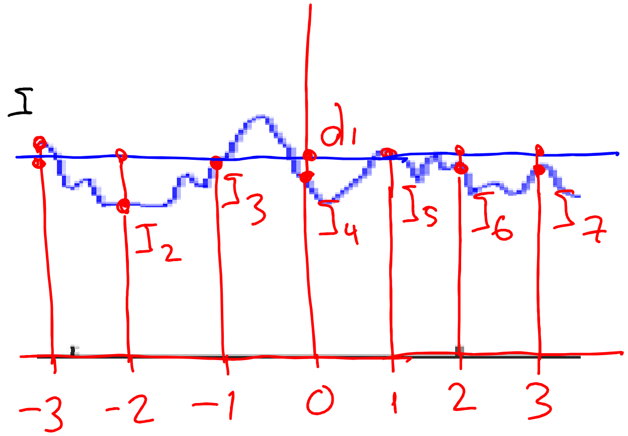
$d_1 \leftarrow I(0)$

Solving linear system in terms of d minimizes the "fit error"

$$\| I - Xd \|^2$$

0^{th}-Order (Constant) Estimation of I(x)

Special case:



- Solution minimizes

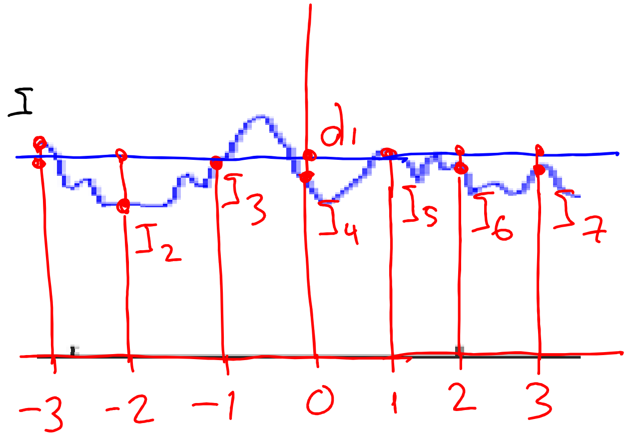
$$\sum_{i=1}^{2w+1} (I_i - d_1)^2$$

- Solution is the mean intensity of the patch:

$$d_1 = \frac{1}{2w+1} \sum_{i=1}^{2w+1} I_i$$

0^{th}-Order (Constant) Estimation of I(x)

Special case:



- Solution minimizes
$$\sum_{i=1}^{2w+1} (I_i - d_1)^2$$

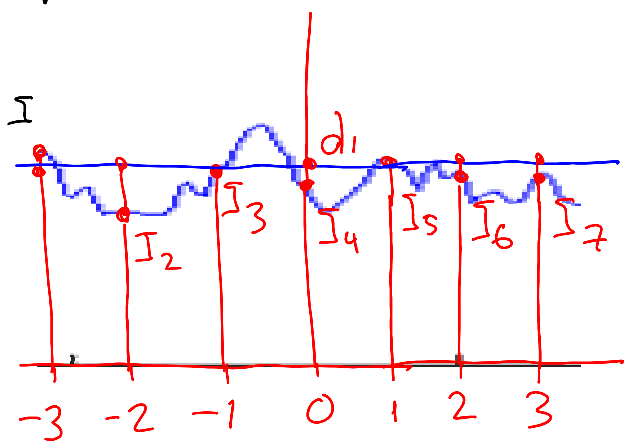
- Solution is the mean intensity of the patch:
$$d_1 = \frac{1}{2w+1} \sum_{i=1}^{2w+1} I_i$$

Proof

- Let $E(x) = \sum_{i=1}^{2w+1} (I_i - x)^2$

0^{th}-Order (Constant) Estimation of I(x)

Special case:



- Solution minimizes

$$\sum_{i=1}^{2w+1} (I_i - d_1)^2$$

- Solution is the mean intensity of the patch:

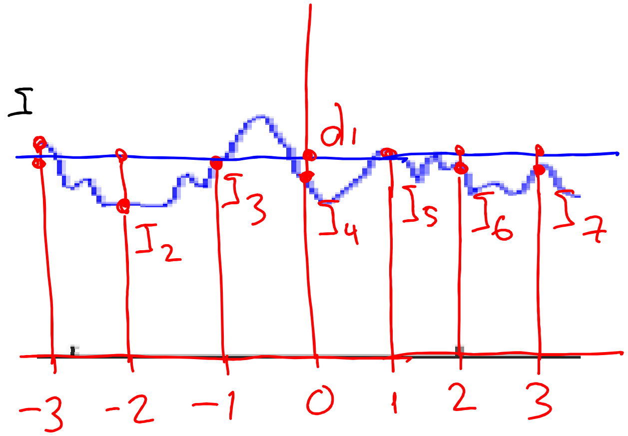
$$d_1 = \frac{1}{2w+1} \sum_{i=1}^{2w+1} I_i$$

Proof

- Let $E(x) = \sum_{i=1}^{2w+1} (I_i - x)^2$
- At the minimum of $E(x_i)$, the derivative $\frac{d}{dx_i} E(x)$ must be zero

0^{th}-Order (Constant) Estimation of I(x)

Special case:



- Solution minimizes

$$\sum_{i=1}^{2w+1} (I_i - d_1)^2$$

- Solution is the mean intensity of the patch:

$$d_1 = \frac{1}{2w+1} \sum_{i=1}^{2w+1} I_i$$

Proof

- Let $E(x) = \sum_{i=1}^{2w+1} (I_i - x)^2$
- At the minimum of $E(x_i)$, the derivative $\frac{d}{dx} E(x)$ must be zero
- $$\frac{d}{dx} E(x) = \sum_{i=1}^{2w+1} \frac{d}{dx} \left[(I_i - x)^2 \right]$$

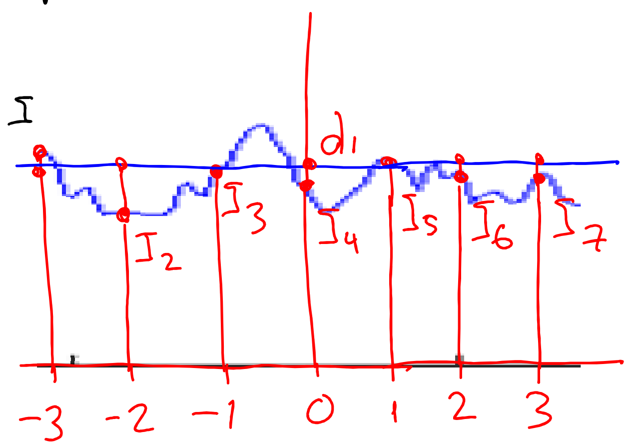
$$= \sum_{i=1}^{2w+1} 2(I_i - x) \cdot (-1)$$

$$= -2 \left[\sum_{i=1}^{2w+1} (I_i - x) \right]$$

$$= -2 \left(\sum_{i=1}^{2w+1} I_i \right) + 2(2w+1)x$$

0^{th}-Order (Constant) Estimation of I(x)

Special case:



- Solution minimizes

$$\sum_{i=1}^{2w+1} (I_i - d_1)^2$$

- Solution is the mean intensity of the patch:

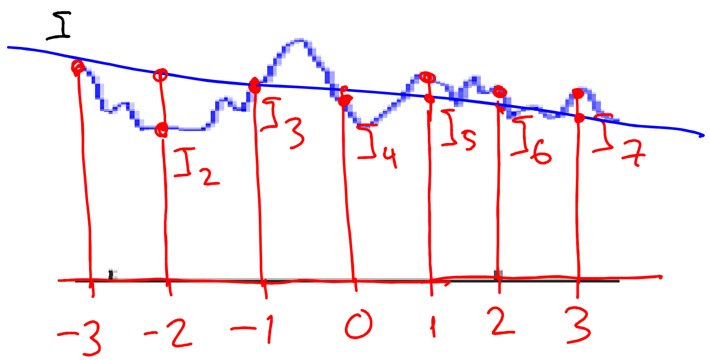
$$d_1 = \frac{1}{2w+1} \sum_{i=1}^{2w+1} I_i$$

Proof

- Let $E(x) = \sum_{i=1}^{2w+1} (I_i - x)^2$
- At the minimum of $E(x_i)$, the derivative $\frac{d}{dx} E(x)$ must be zero
- $$\frac{d}{dx} E(x) = \sum_{i=1}^{2w+1} \frac{d}{dx} \left[(I_i - x)^2 \right]$$
$$= \sum_{i=1}^{2w+1} 2 (I_i - x) \cdot (-1)$$
$$= -2 \left[\sum_{i=1}^{2w+1} (I_i - x) \right]$$
$$= -2 \left(\sum_{i=1}^{2w+1} I_i \right) + 2(2w+1) x$$
- $$\frac{d}{dx} E(x) = 0 \iff x = \frac{1}{2w+1} \left(\sum_{i=1}^{2w+1} I_i \right)$$

1st-Order (Linear) Estimation of I(x)

Special case:

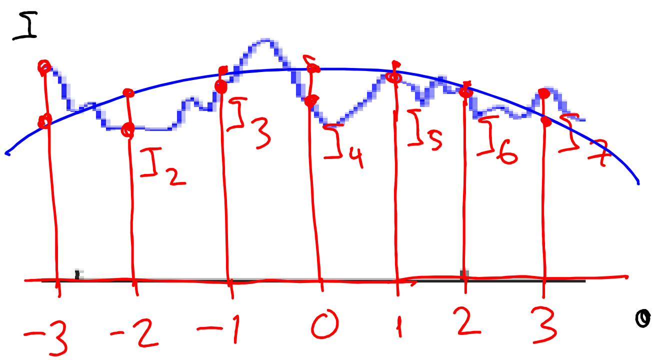


- Solution minimizes sum of "vetrical" distances between line and image intensities

- Gives us an estimate of $I(0)$ and $\frac{dI}{dx}(0)$ (i.e. value & derivative at 0)

2nd-Order (Quadratic) Estimation of I(x)

Special case:

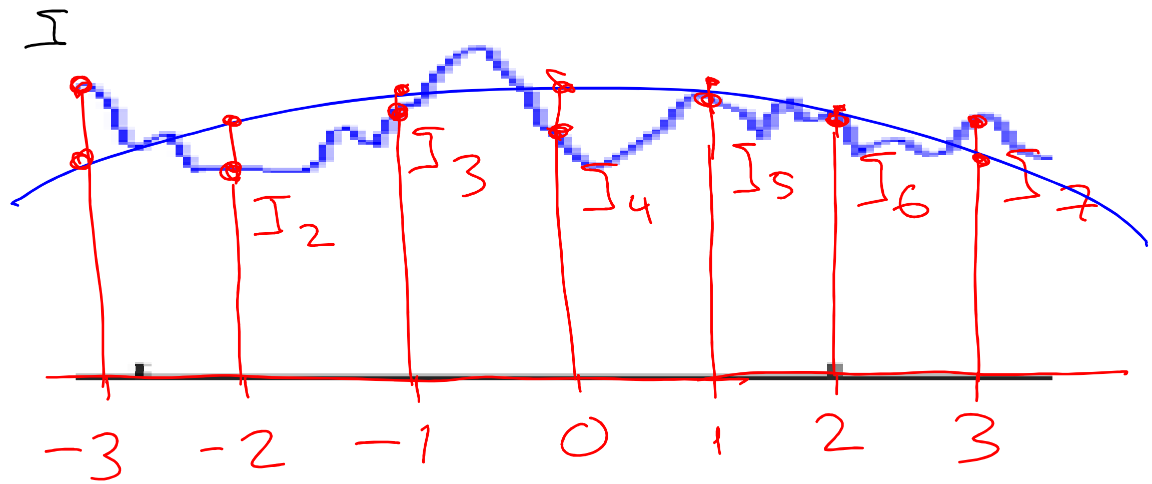


- Fits a parabola/hyperbola/ellipse
- Gives us an estimate of 1st & 2nd image derivative at patch center

$$\begin{bmatrix} I_1 \\ I_2 \\ \vdots \\ I_n \end{bmatrix} = \begin{bmatrix} 1 & -3 & 9/2 \\ 1 & -2 & 2 \\ \vdots & \vdots & \\ 1 & 3 & 9/2 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ d_3 \end{bmatrix}$$

$d_3 \leftarrow \frac{d^2 I}{dx^2}(0)$

2nd-Order (Quadratic) Estimation of I(x)



Note how all pixels in the window contribute equally to the estimate around the center of the window!

Topic 4.1:

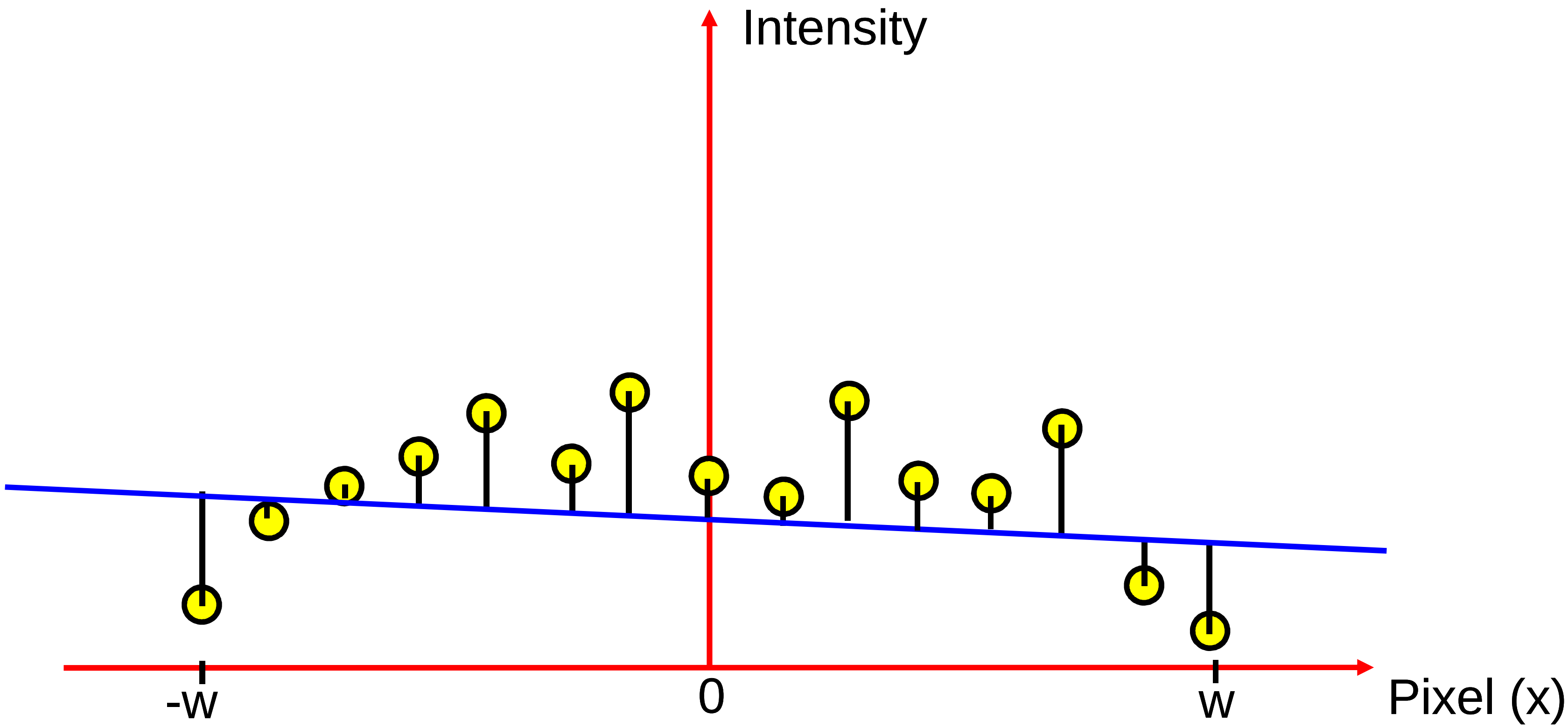
Local analysis of 1D image patches

- Taylor series approximation of 1D intensity patches
- Estimating derivatives of 1D intensity patches:
  - Least-squares fitting
  - Weighted least-squares fitting
  - Robust polynomial fitting: RANSAC

Weighted Least Squares Polynomial Fitting

Scenario #1:

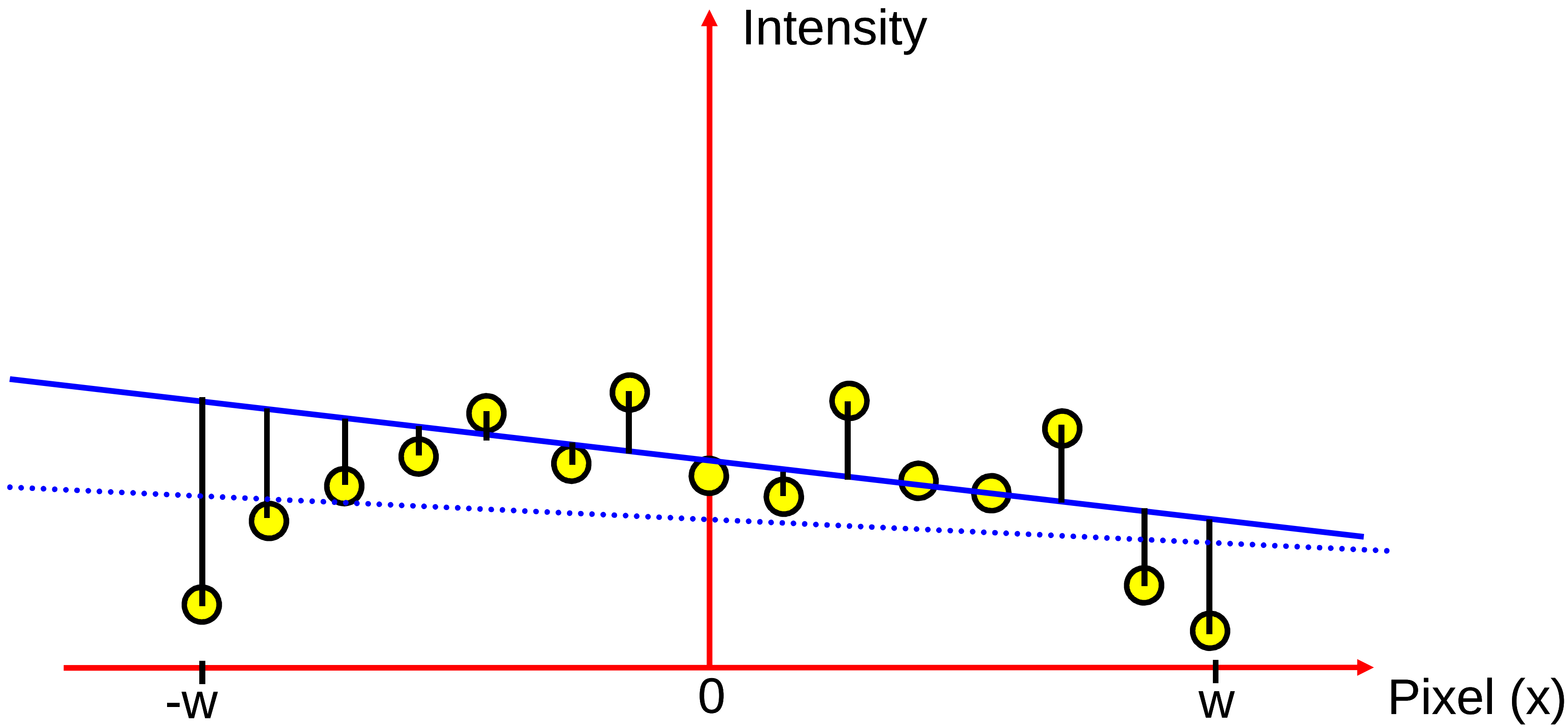
- Fit polynomial to ALL pixel intensities in a patch



Weighted Least Squares Polynomial Fitting

Scenario #2:

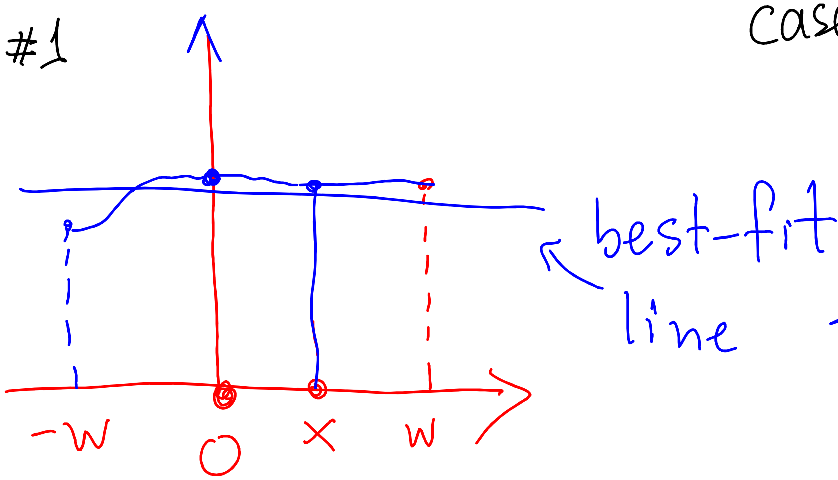
- Fit polynomial to all the pixel intensities in the patch
- Pixels contribute to estimate of derivative(s) at center according to a weight function $\Omega(x)$



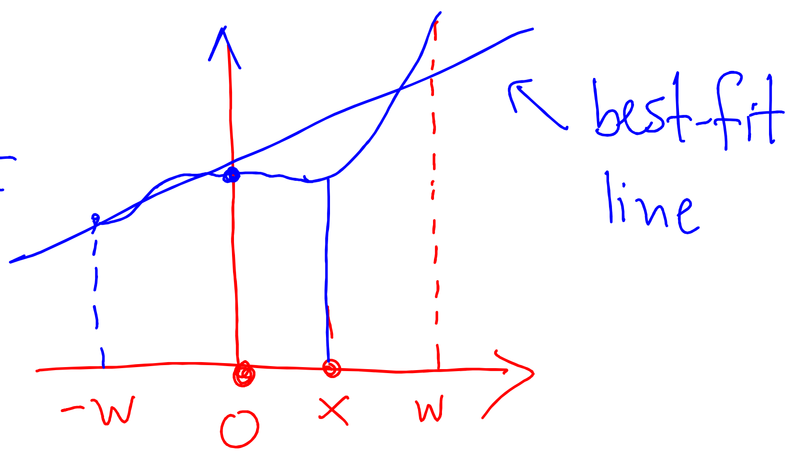
Polynomial Fitting: A Linear Formulation

Q: Will the estimate of $\frac{dI}{dx}(0)$ be the same or different in the two cases below? (assume a 1^{st} order fit)

case #1



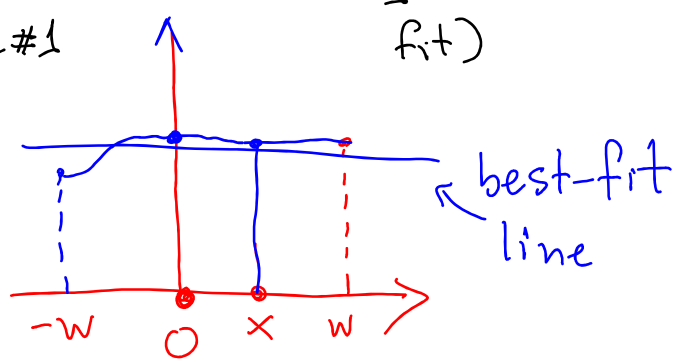
case #2



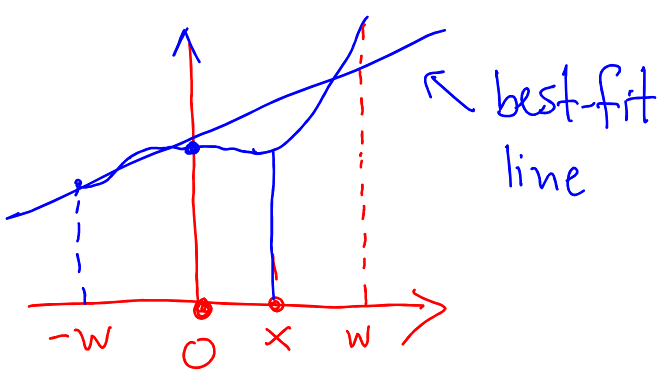
Polynomial Fitting: A Linear Formulation

Q: Will the estimate of $\frac{dI}{dx}(0)$ be the same or different in the two cases below? (assume a 1^{st} order fit)

case #1



case #2

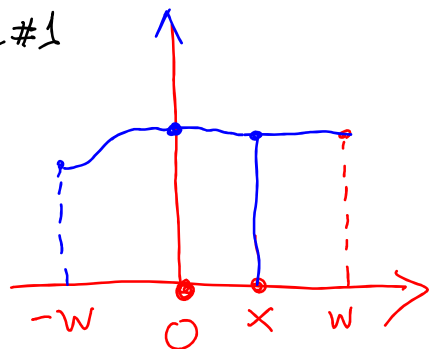


Ans: the values will differ because all patch pixels contribute equally to the linear system!

Weighted Least-Squares Estimation of I(x)

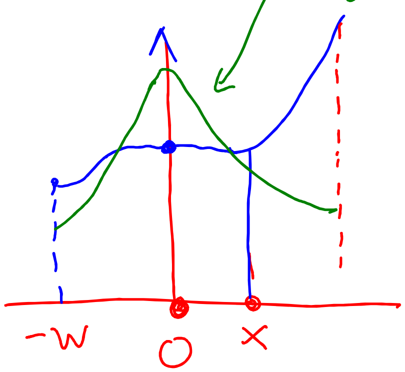
Q: How can we bias our estimate of $\frac{d}{dx}I(0)$ toward the patch center?

case #1



weight function $\Omega(x)$
(e.g. $\Omega(x) = e^{-x^2}$)

case #2

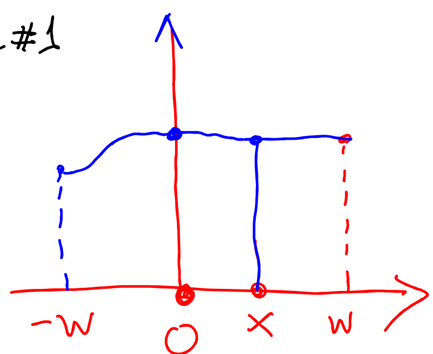


Idea: Weigh pixels near center more than pixels away from it

Weighted Least-Squares Estimation of I(x)

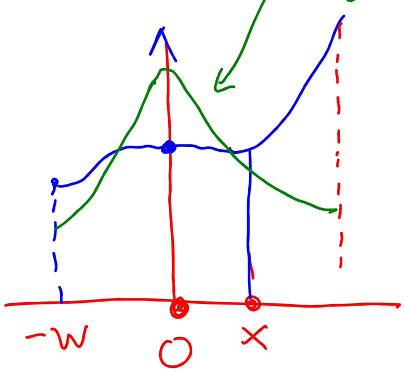
Q: How can we bias our estimate of $\frac{d}{dx} I(0)$ toward the patch center?

case #1



weight function $\Omega(x)$
(e.g. $\Omega(x) = e^{-x^2}$)

case #2



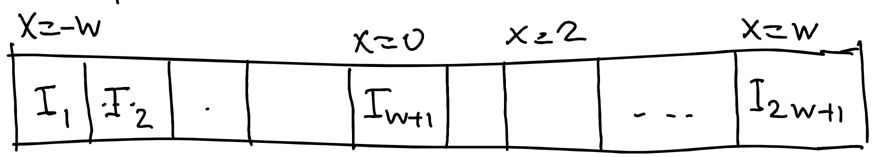
New equation for pixel x:

$$\Omega(x) I(x) = \Omega(x) \cdot \begin{bmatrix} 1 & x & \frac{1}{2}x^2 & \frac{1}{6}x^3 & \cdots & \frac{1}{n!}x^n \end{bmatrix} \begin{bmatrix} I(0) \\ \frac{dI}{dx}(0) \\ \frac{dI}{dx^2}(0) \\ \vdots \\ \frac{d^nI}{dx^n}(0) \end{bmatrix}$$

Weighted Least-Squares Estimation of I(x)

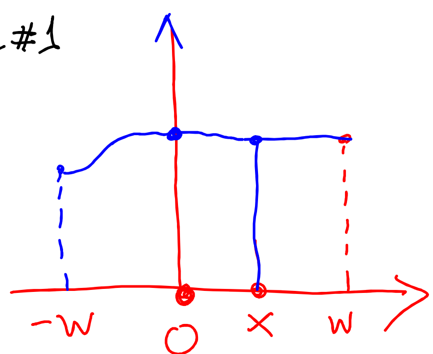
Q: How can we bias our estimate of $\frac{d}{dx} I(0)$ toward the patch center?

Patch (2w+1 pixels)



$$\begin{bmatrix} \Omega_1 & & 0 \\ & \ddots & \\ 0 & & \Omega_{2w+1} \end{bmatrix} I = \begin{bmatrix} \Omega_1 & & \\ & \ddots & \\ & & \Omega_{2w+1} \end{bmatrix} X d$$

case #1

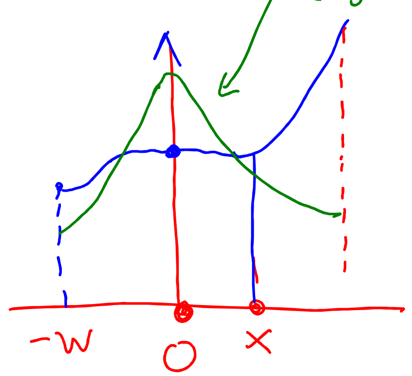


Solution d minimizes the norm

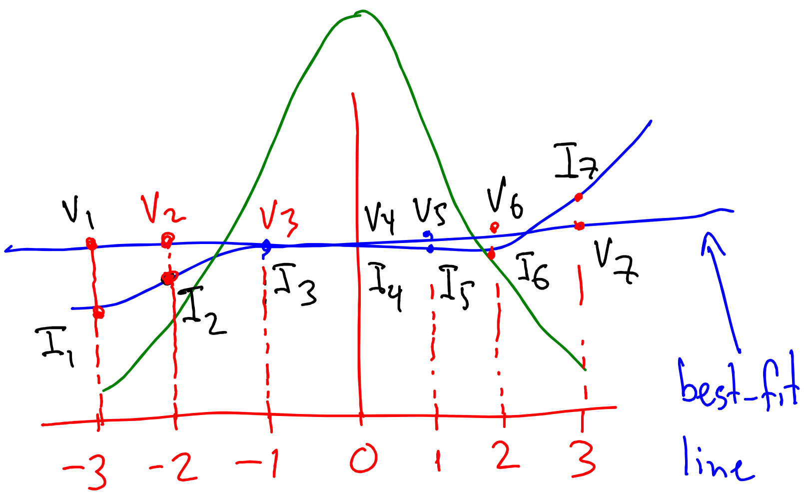
$$\left\| \begin{bmatrix} \Omega_1 & & 0 \\ & \ddots & \\ 0 & & \Omega_{2w+1} \end{bmatrix} (I - Xd) \right\|^2$$

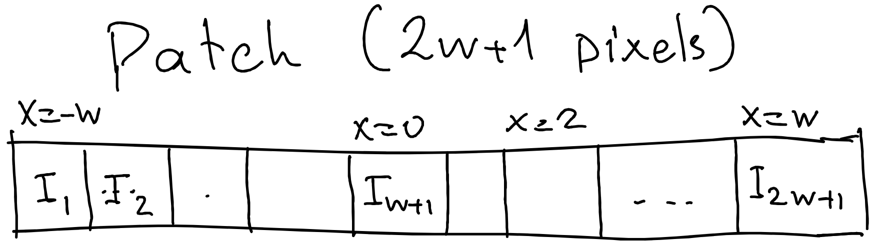
weight function $\Omega(x)$ (e.g. $\Omega(x) = e^{-x^2}$)

case #2



Weighted Least-Squares Estimation of I(x)





$$\begin{bmatrix} \Omega_1 & & 0 \\ & \ddots & \\ 0 & & \Omega_{2w+1} \end{bmatrix} I = \begin{bmatrix} \Omega_1 & & \\ & \ddots & \\ & & \Omega_{2w+1} \end{bmatrix} X d$$

We could then do v=Xd to get an estimate of I(x) for all pixels in the patch in (-w, ..., 0, ..., w).

- This solution minimizes the 2-norm (i.e. the length) of the weighted error vector:

$$\left(\sum_{i=1}^{2w+1} \left[\Omega_i (I_i - v_i) \right]^2 \right)^{1/2}$$

Topic 4.1:

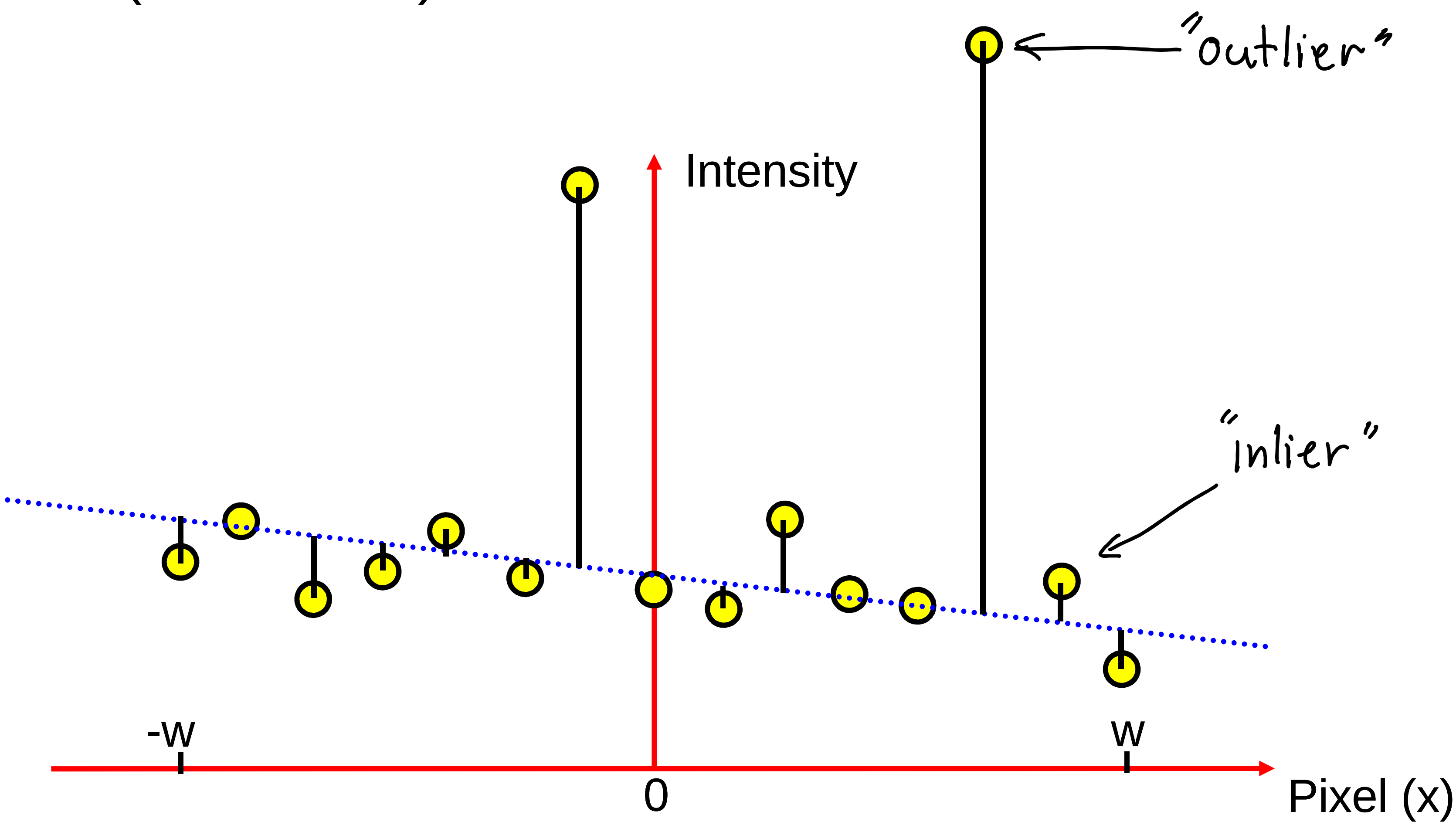
Local analysis of 1D image patches

- Taylor series approximation of 1D intensity patches
- Estimating derivatives of 1D intensity patches:
  - Least-squares fitting
  - Weighted least-squares fitting
  - Robust polynomial fitting: RANSAC

Robust Polynomial Fitting

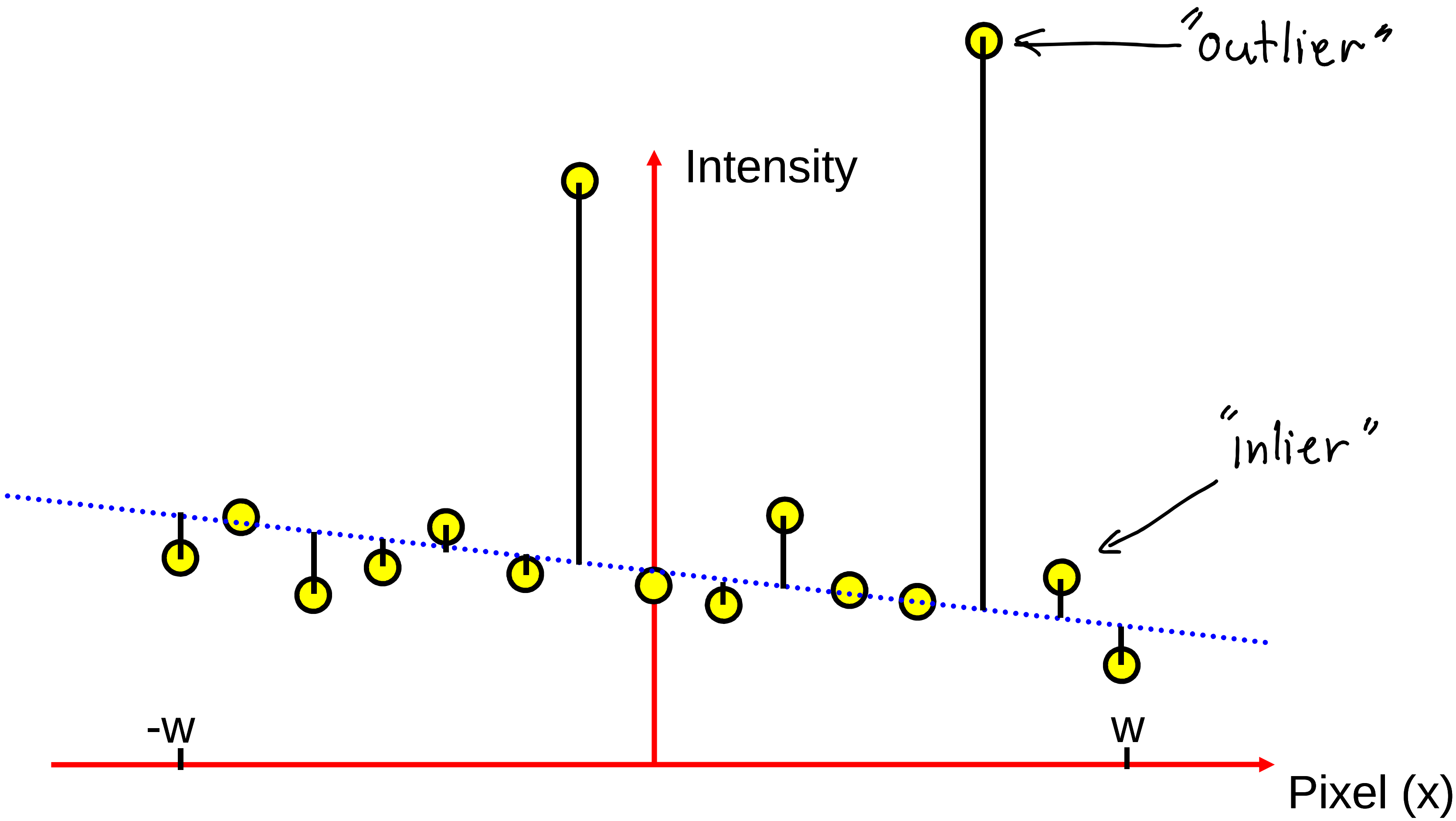
Scenario #3:

- Fit polynomial only to SOME pixel intensities in a patch (the “inliers”)



Robust Polynomial Fitting

But how can we tell between inliers and outliers?



Robust Polynomial Fitting

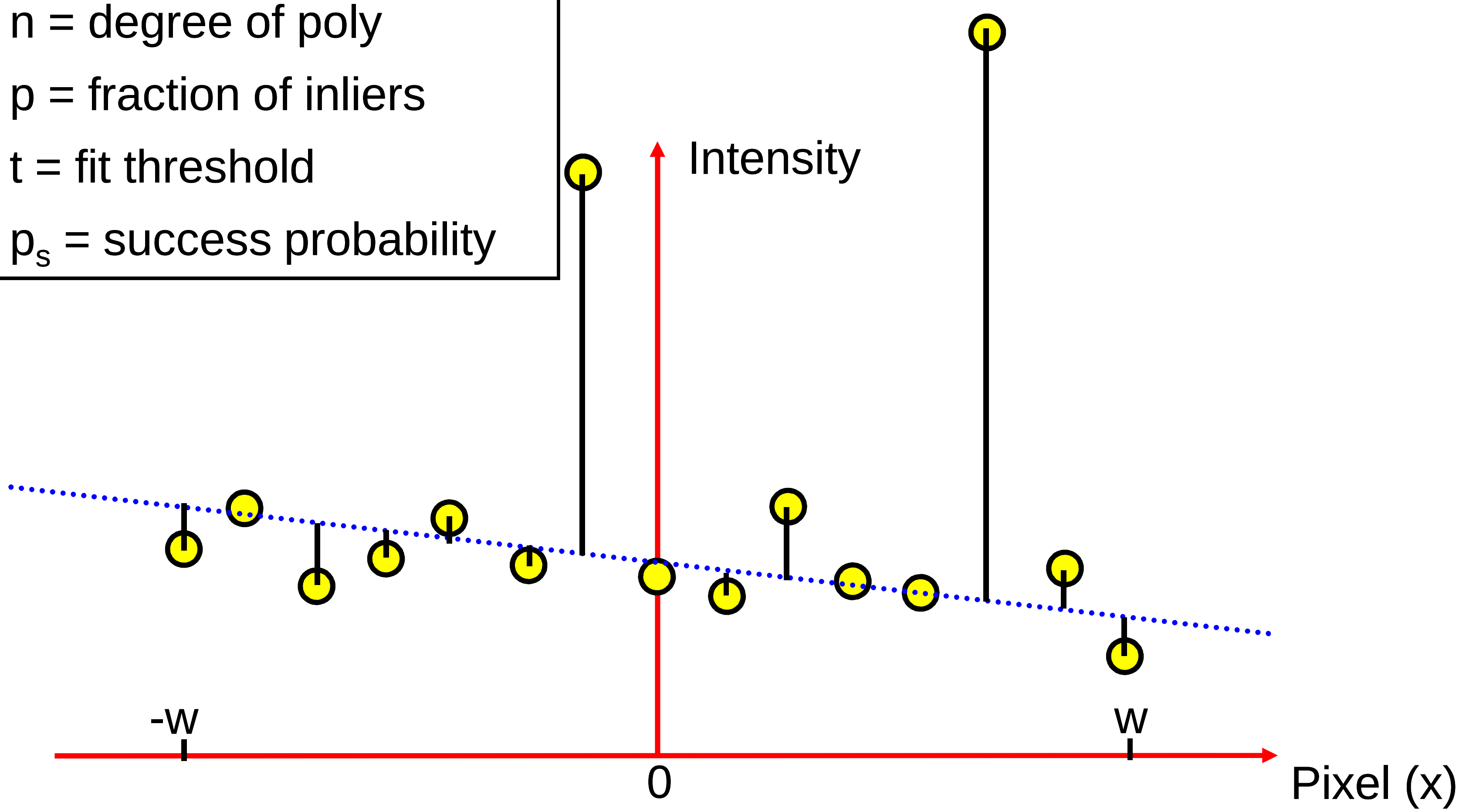
We can’t. At least not before we fit a model.

Polynomial Fitting Using RANSAC

Here's our problem: find the inliers, fit a polynomial to them:

Given:

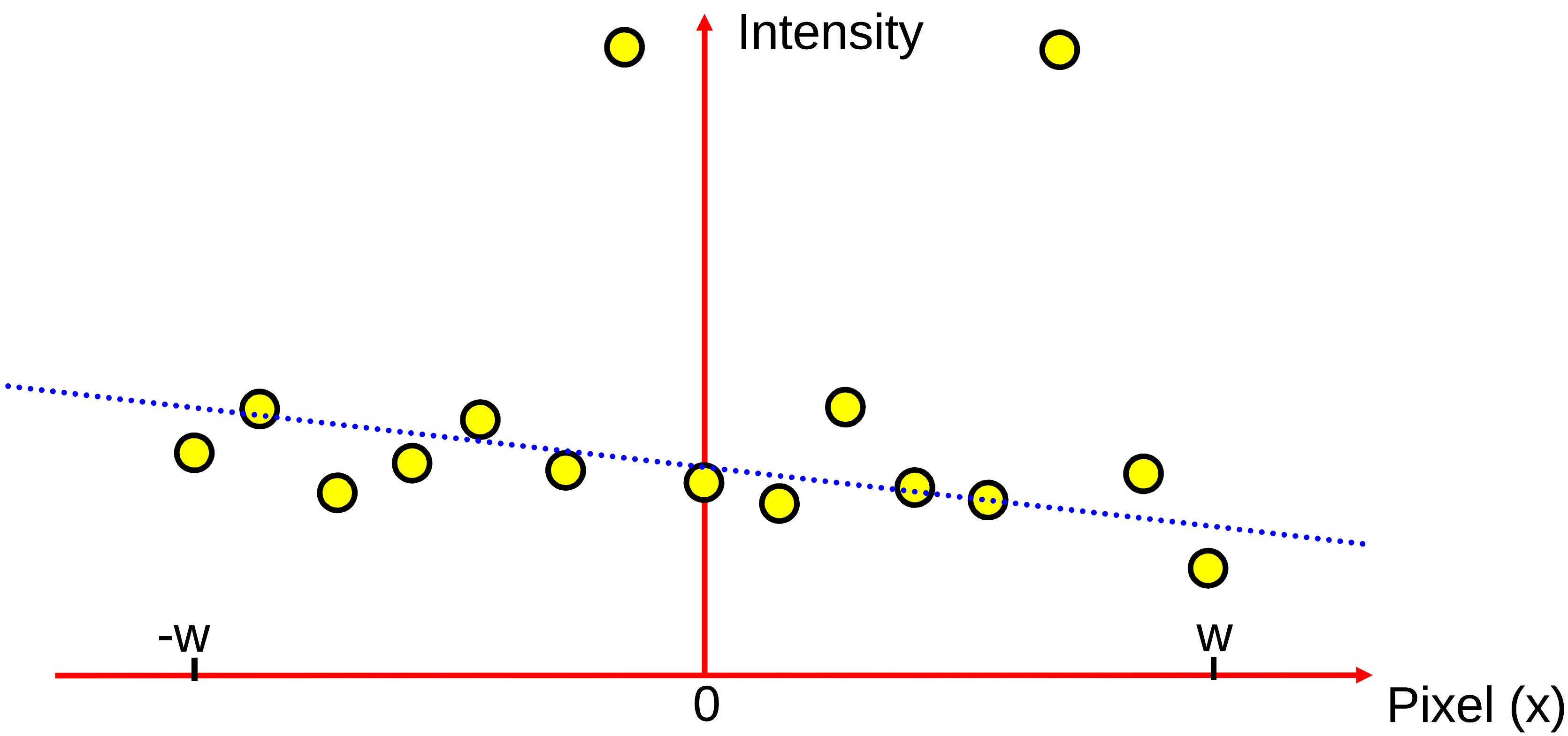
- n = degree of poly
- p = fraction of inliers
- t = fit threshold
- p_s = success probability



RANSAC Algorithm

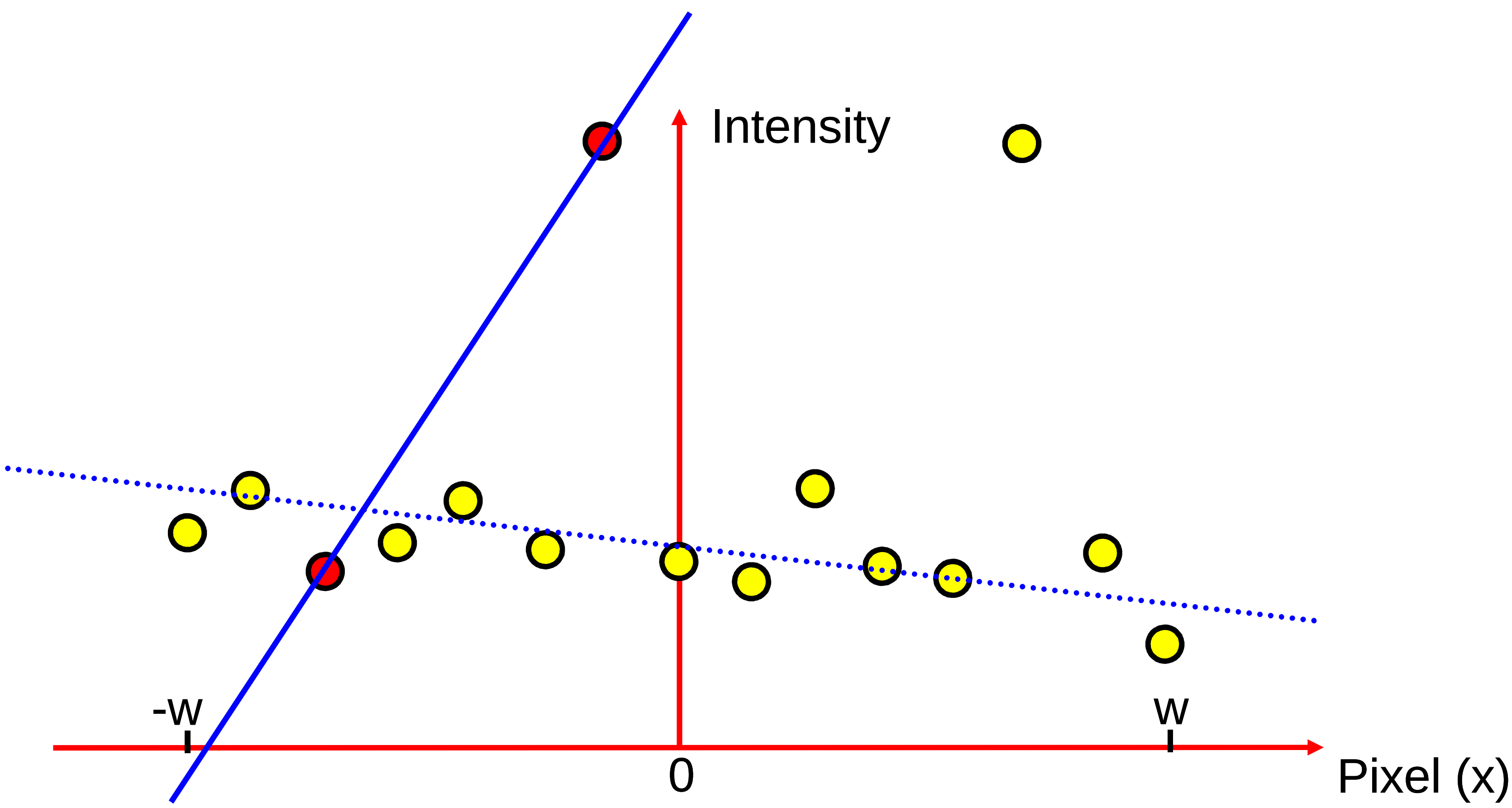
Example: Line fitting using RANSAC (i.e., n=2 unknown polynomial coefficients)

- Step 1: Randomly choose n pixels from the patch



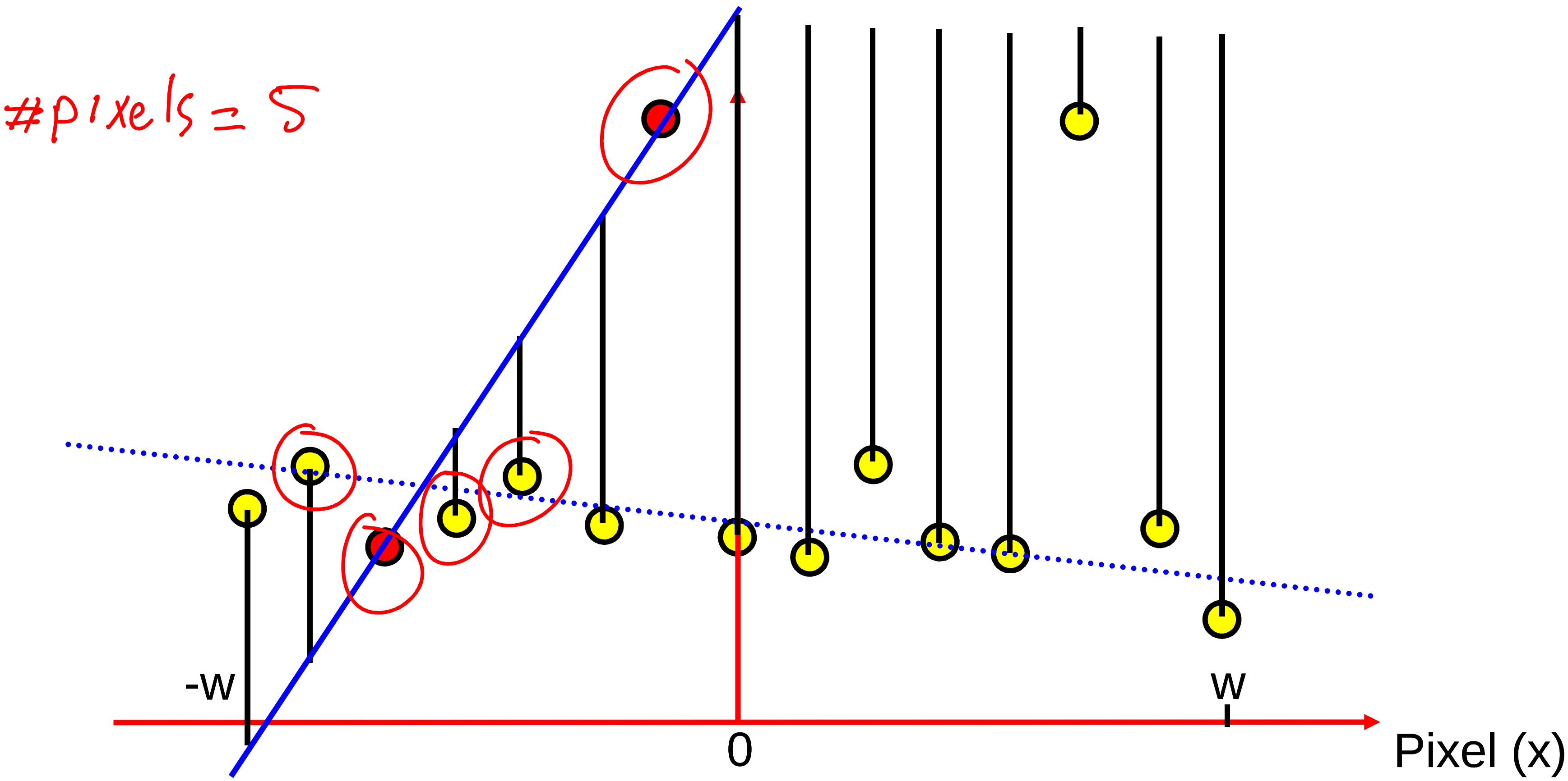
RANSAC Algorithm

Step 2: Fit the poly using the chosen pixels/intensities



RANSAC Algorithm

Step 3: Count pixels with vertical distance < threshold t



RANSAC Algorithm

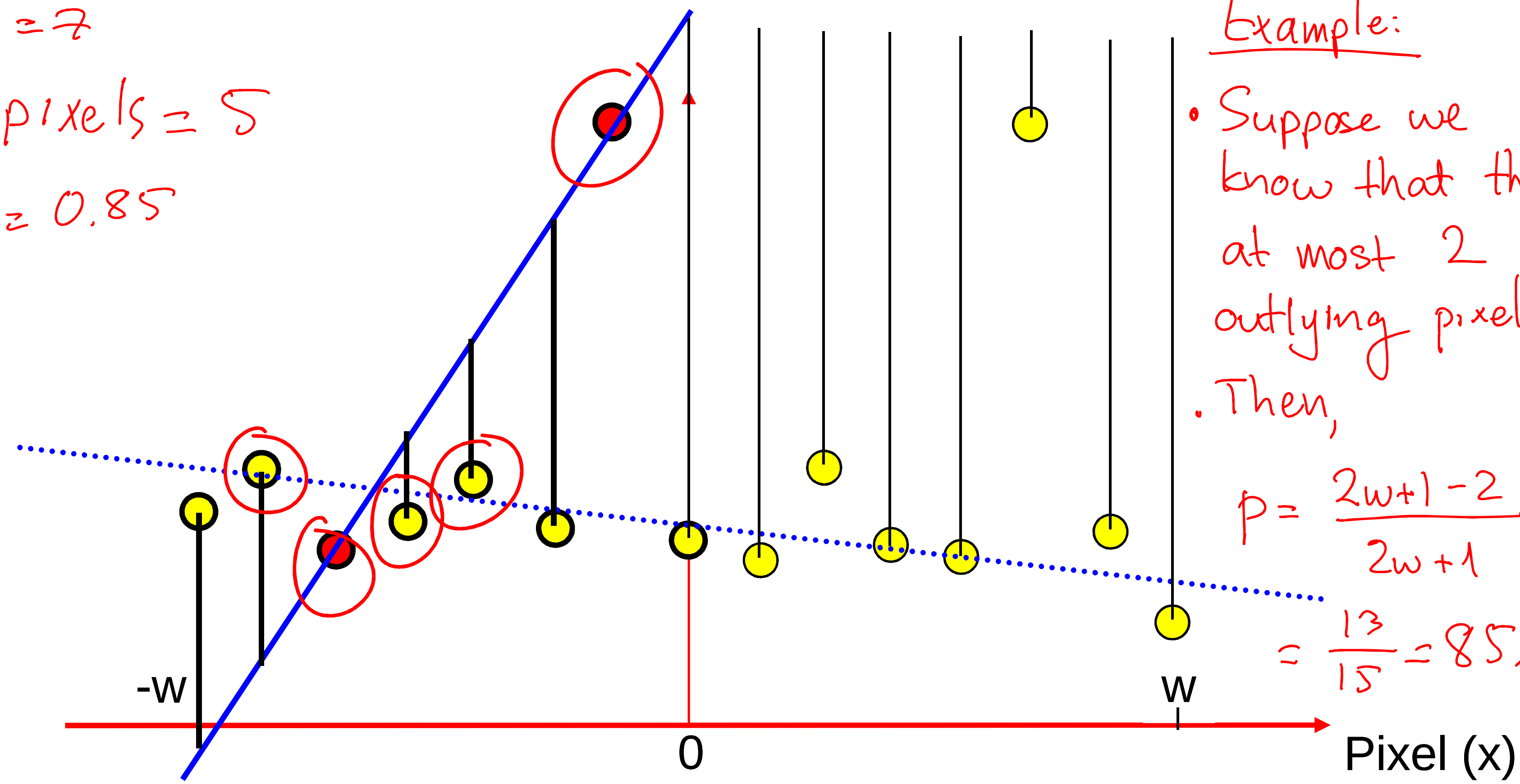
$= (2w+1)\cdot p$ pixels

- Step 4: If there aren't "enough" such pixels, REPEAT (not more than K times)

$w = 7$

$\#\text{pixels} = 5$

$p = 0.85$



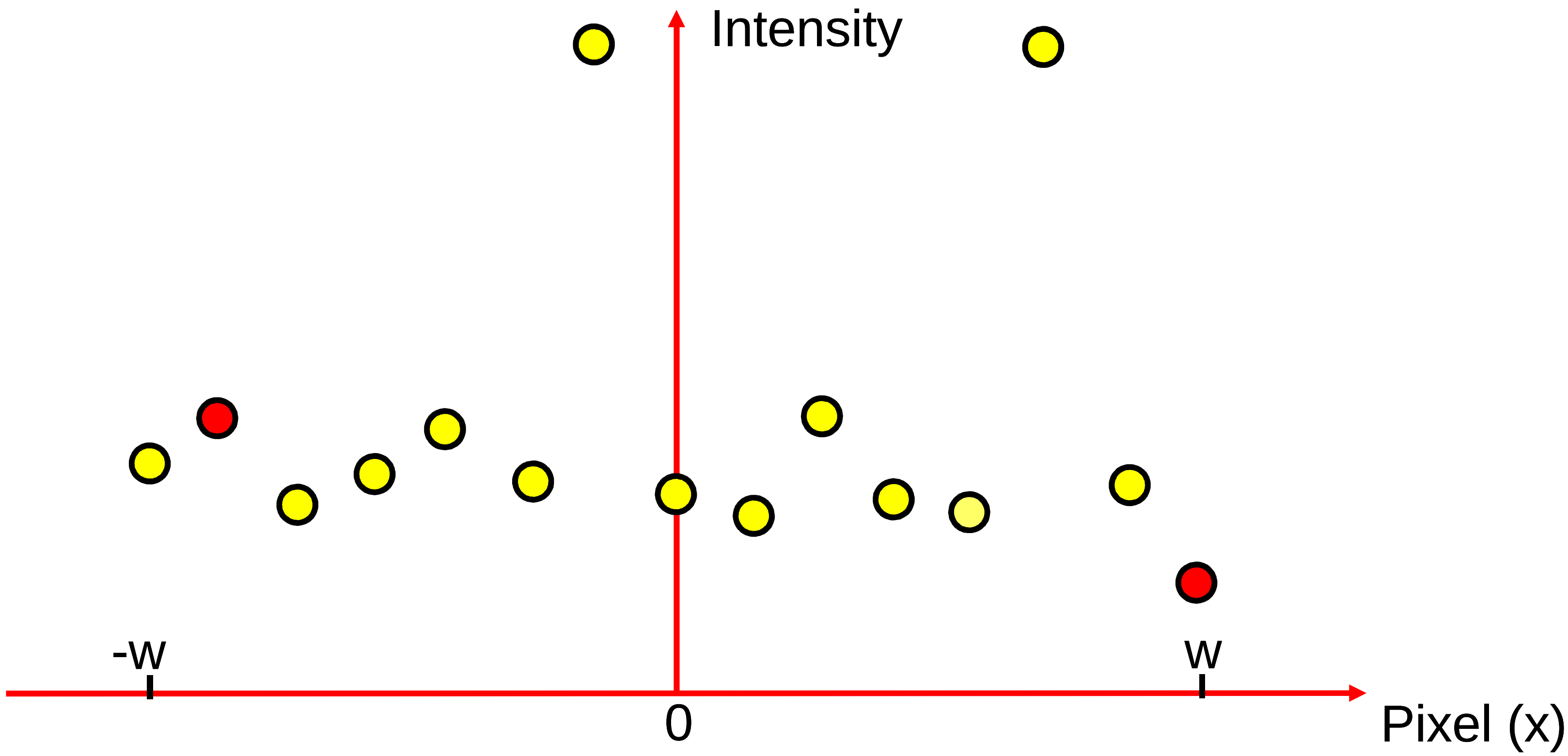
Example:

- Suppose we know that there at most 2 outlying pixels.
- Then,

$$p = \frac{2w+1-2}{2w+1} = \frac{13}{15} = 85\%$$

RANSAC Algorithm

How about these two?



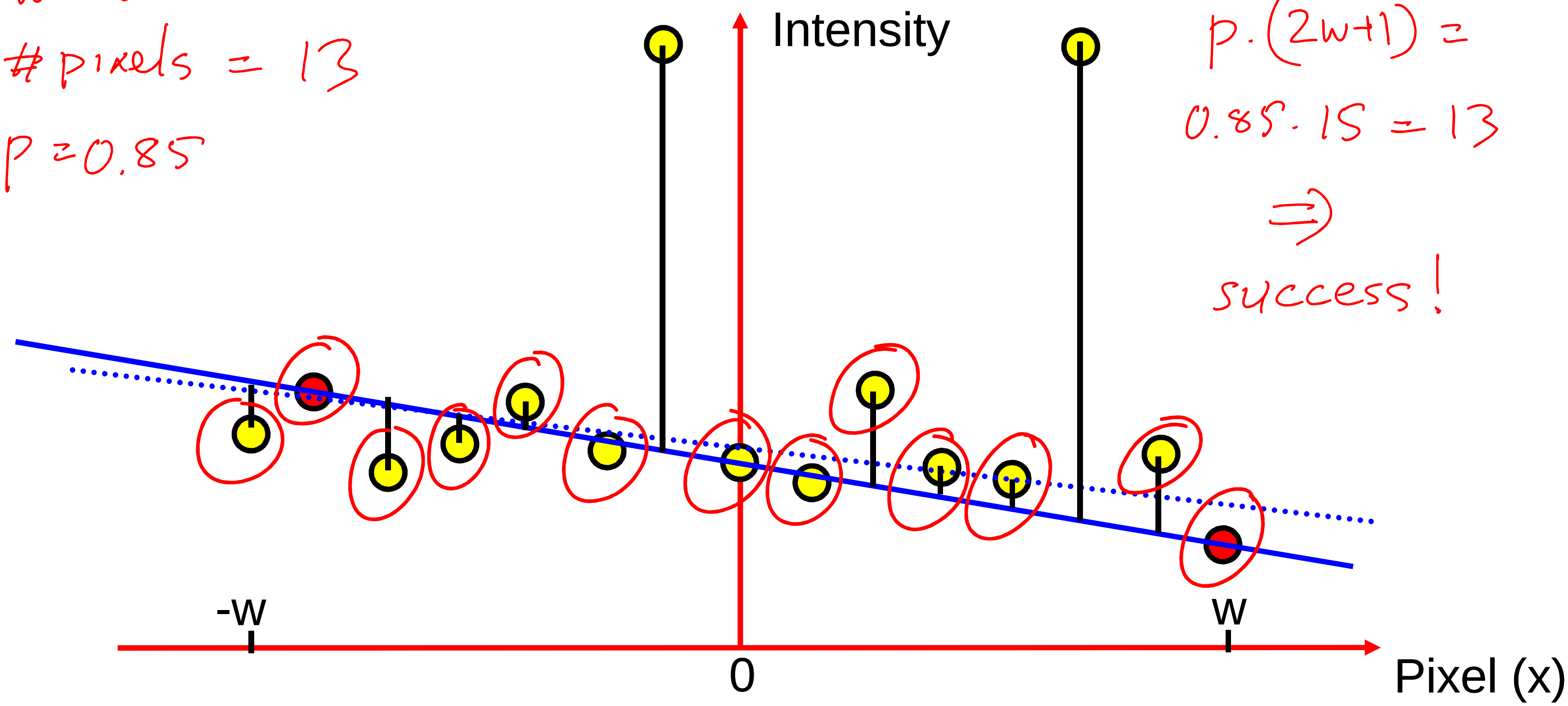
RANSAC Algorithm

Step 4: If there are "enough" such pixels, STOP
Label them as "inliers" & do a least-squares fit
to the INLIER pixels only

$w = 7$

$\#\text{pixels} = 13$

$p = 0.85$

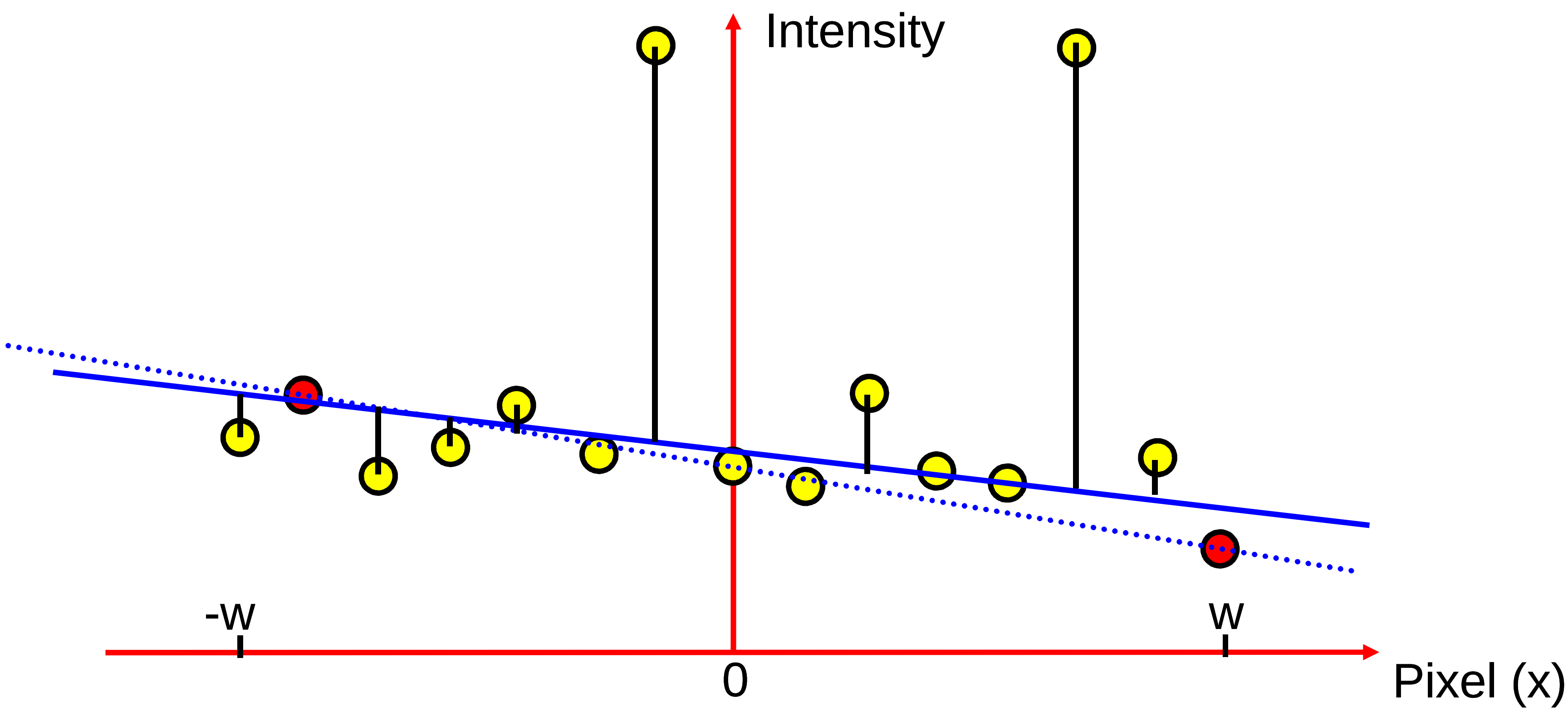


$p \cdot (2w+1) = 0.85 \cdot 15 = 13$

$\Rightarrow$ success!

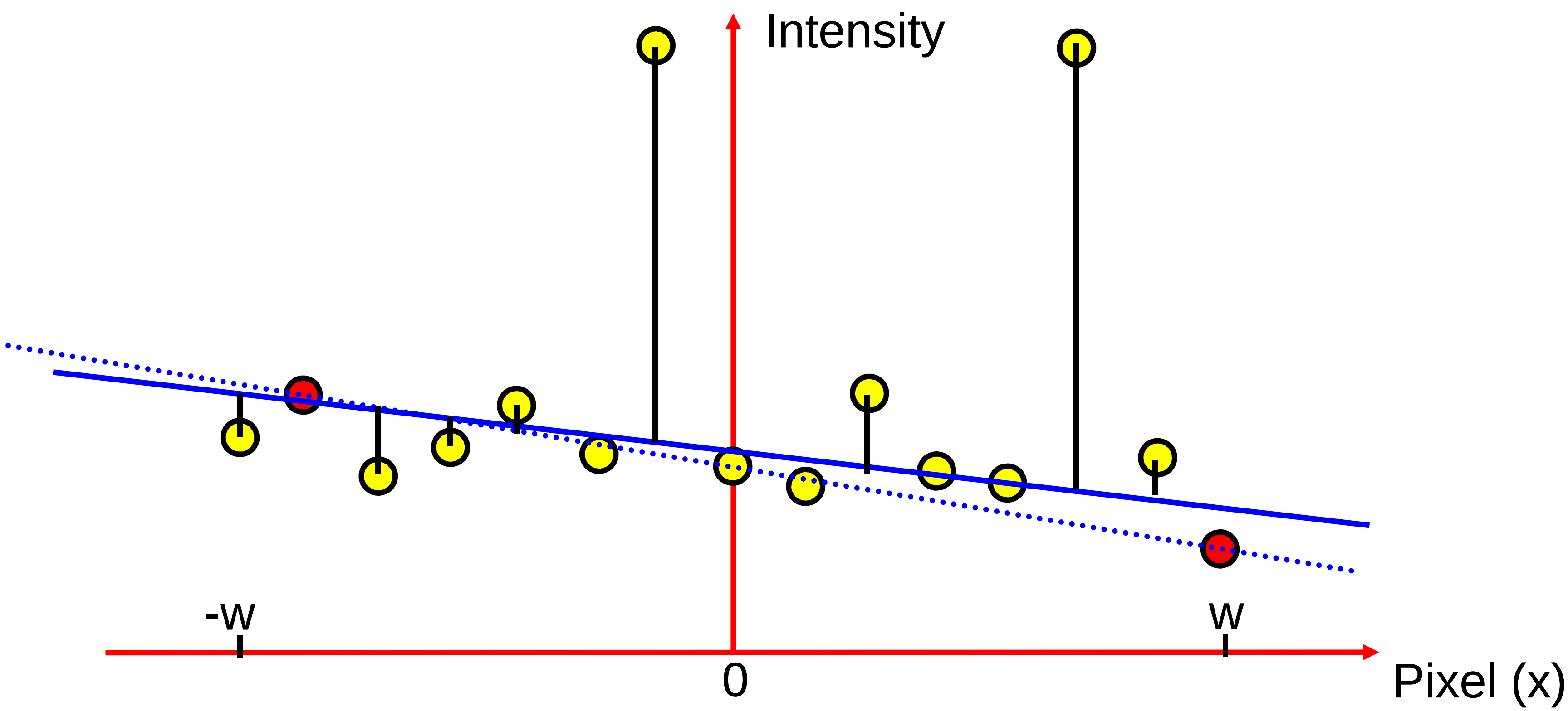
RANSAC Algorithm

Step 4: If there are "enough" such pixels, STOP
Label them as "inliers" & do a least-squares fit
to the INLIER pixels only



RANSAC Algorithm

Eventually, after “enough” trials, there must be some likelihood of having chosen n+1 inliers to fit the model.



RANSAC Algorithm

Eventually, after “enough” trials, there must be some likelihood of having chosen n+1 inliers to fit the model.

How many trials are enough then?

RANSAC Algorithm

Given:

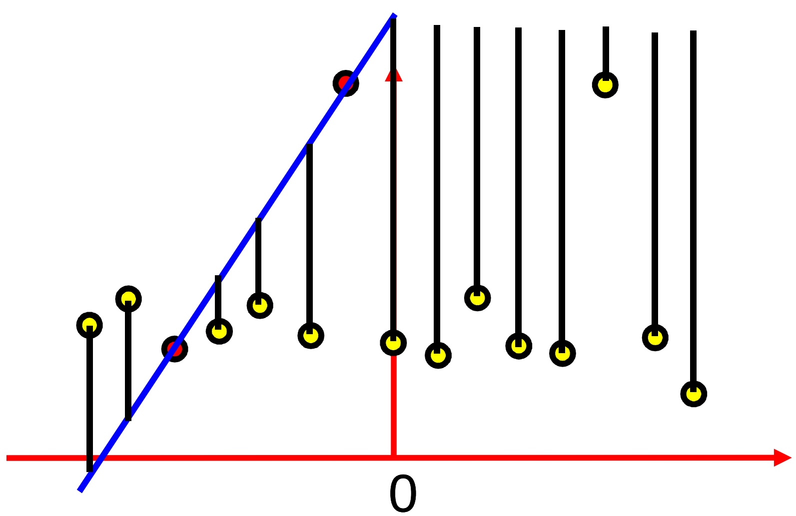
- n = degree of poly
- p = fraction of inliers
- t = fit threshold
- p_s = success probability

Repeat at most K times:

1. Randomly choose n+1 pixels
2. Fit n-degree poly
3. Count pixels whose vertical distance from poly is < t
4. If there are at least (2w+1)p pixels, EXIT LOOP
   a. Label them as inliers
   b. Fit n-degree poly to all inlier pixels

Q: What should K be?

- Probability we chose an inlier pixel: p
- Probability we chose $(n+1)$ inlier pixels: p^{n+1}
- Prob at least 1 outlier chosen: $1-p^{n+1}$
- Prob at least 1 outlier chosen in all K trials: $(1-p^{n+1})^K$



RANSAC Algorithm

Given:

- n = degree of poly
- p = fraction of inliers
- t = fit threshold
- p_s = success probability

Repeat at most K times:

1. Randomly choose n+1 pixels
2. Fit n-degree poly
3. Count pixels whose vertical distance from poly is < t
4. If there are at least (2w+1)p pixels, EXIT LOOP
   a. Label them as inliers
   b. Fit n-degree poly to all inlier pixels

Q: What should K be?

- Probability we chose an inlier pixel: p
- Probability we chose $(n+1)$ inlier pixels: p^{n+1}
- Prob at least 1 outlier chosen: $1-p^{n+1}$
- Prob at least 1 outlier chosen in all K trials: $(1-p^{n+1})^K$
- Failure probability: $(1-p^{n+1})^K$
- Success probability $p_s = 1-(1-p^{n+1})^K$
- By taking logs on both sides

$$\boxed{K = \frac{\log(1-p_s)}{\log(1-p^{n+1})}}$$