Image Interpolation

Computational Photography (CSCI 3240U)

Faisal Z. Qureshi
http://vclab.science.ontariotechu.ca

OntarioTech UNIVERSITY

How do we resize images?

Original

Upscaling

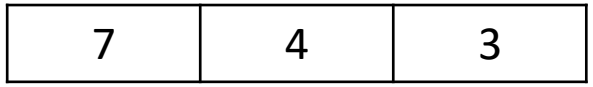
Downscaling

Let’s consider a 1D image

7	4	3

We want to increase its width by a factor of 2

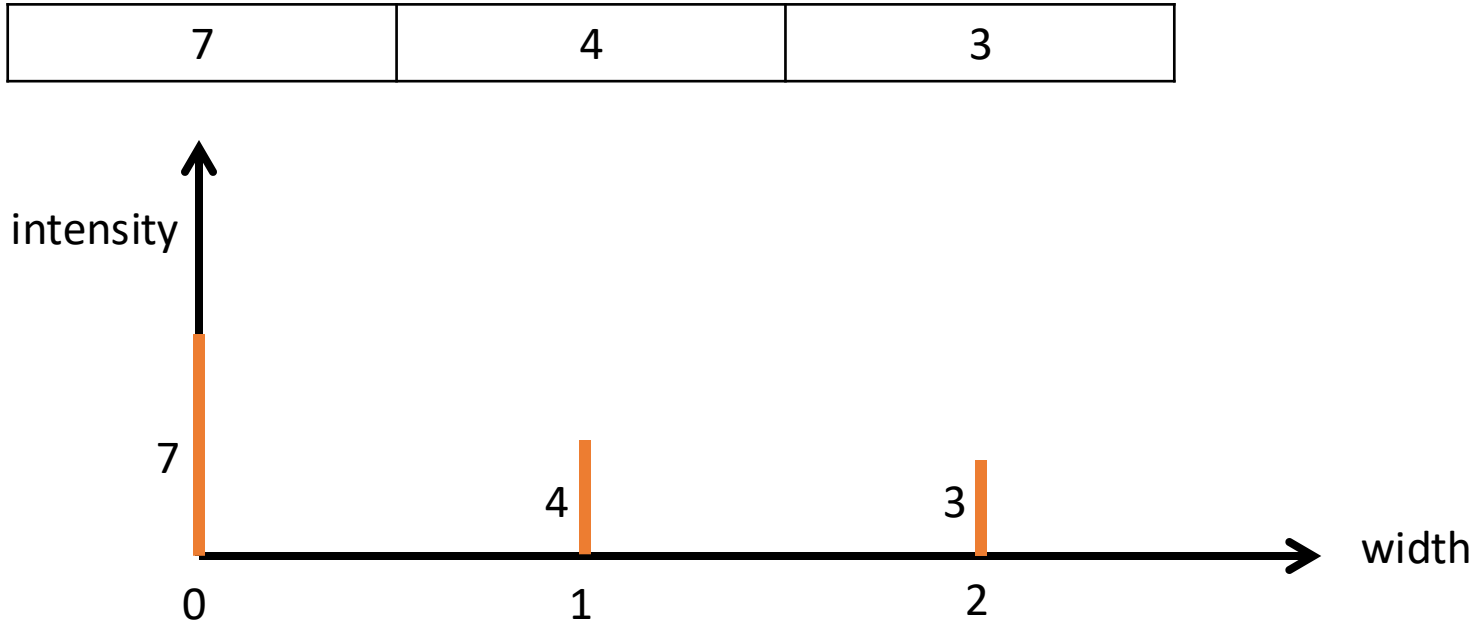
Let’s consider a 1D image



We want to increase its width by a factor of 2

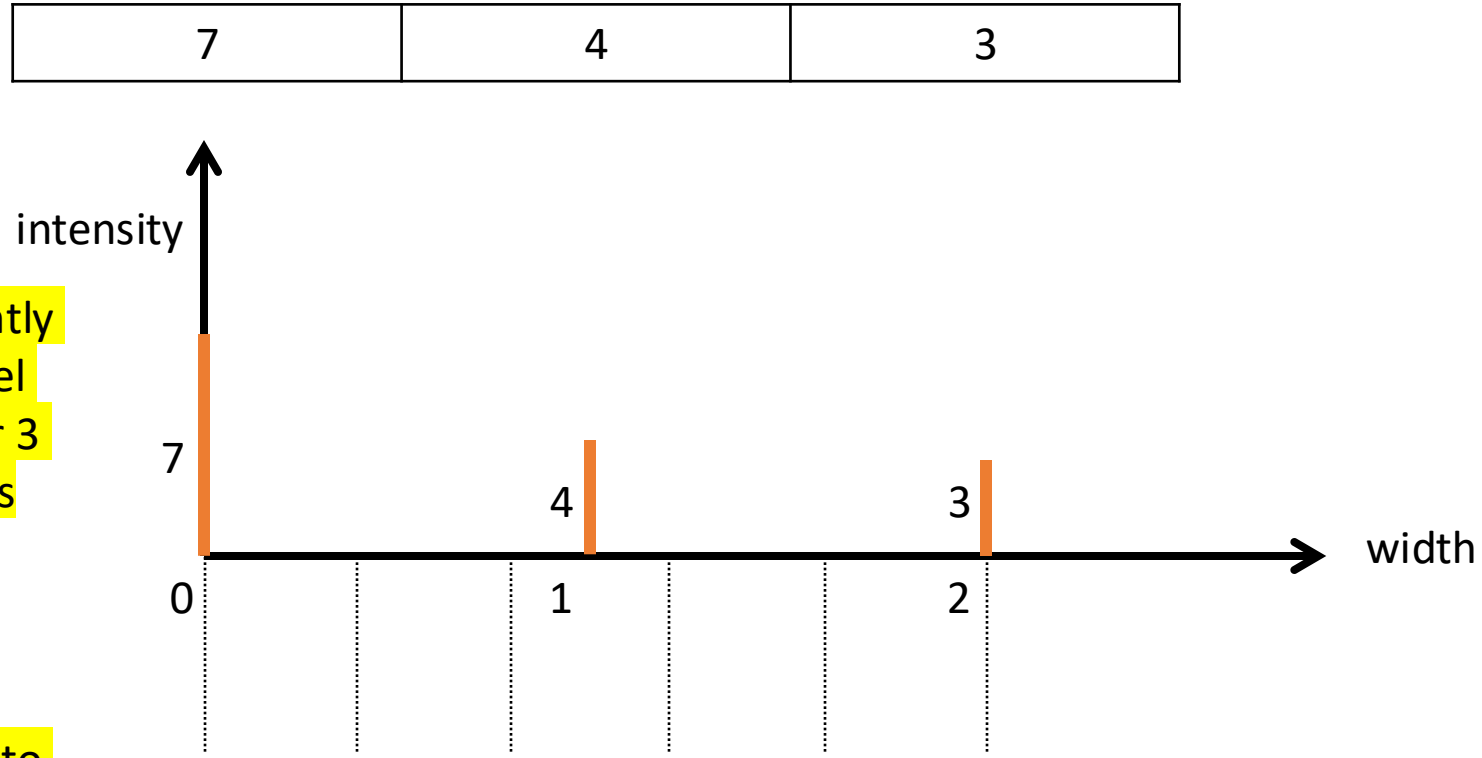
Upscaling

Resampling pixel locations



We currently have pixel values for 3 locations

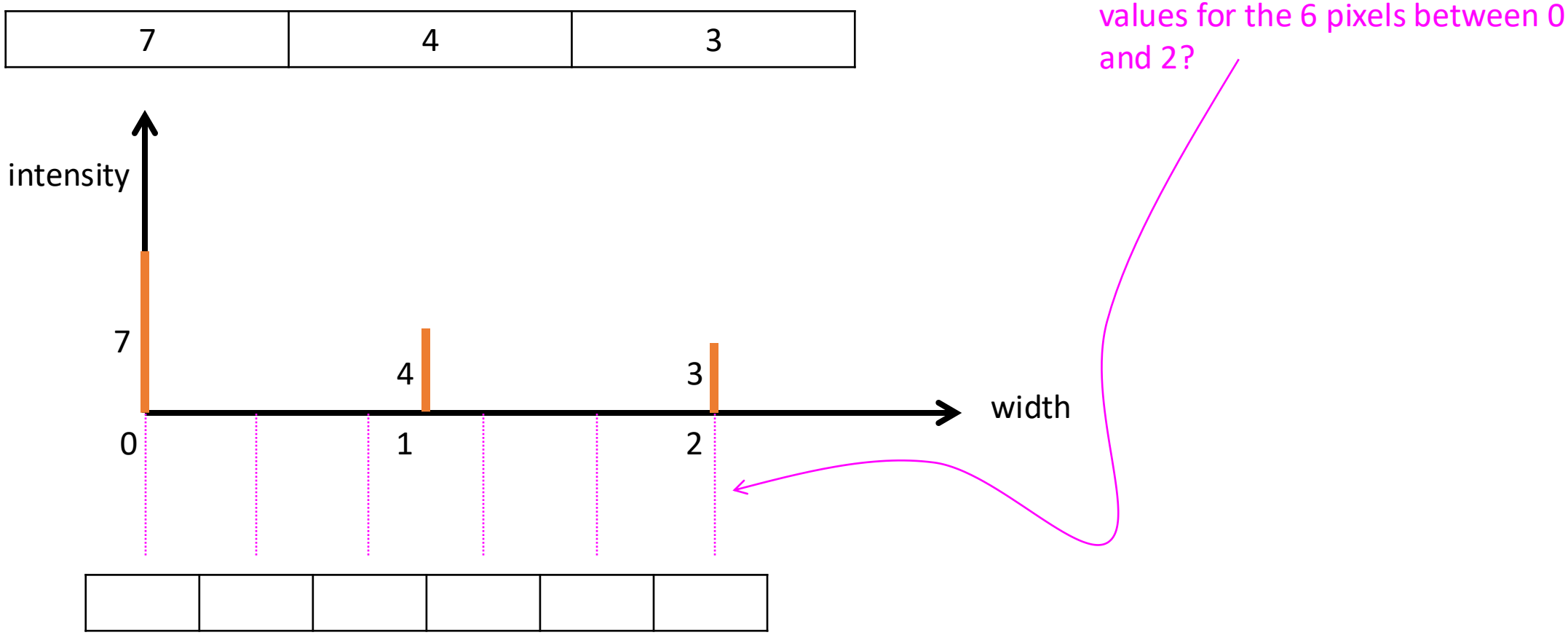
Resampling pixel locations



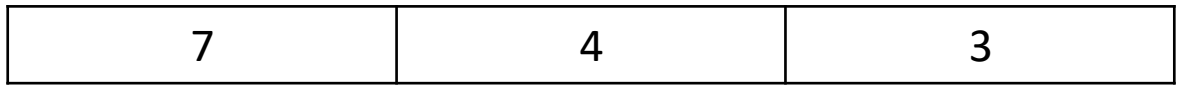
We currently have pixel values for 3 locations

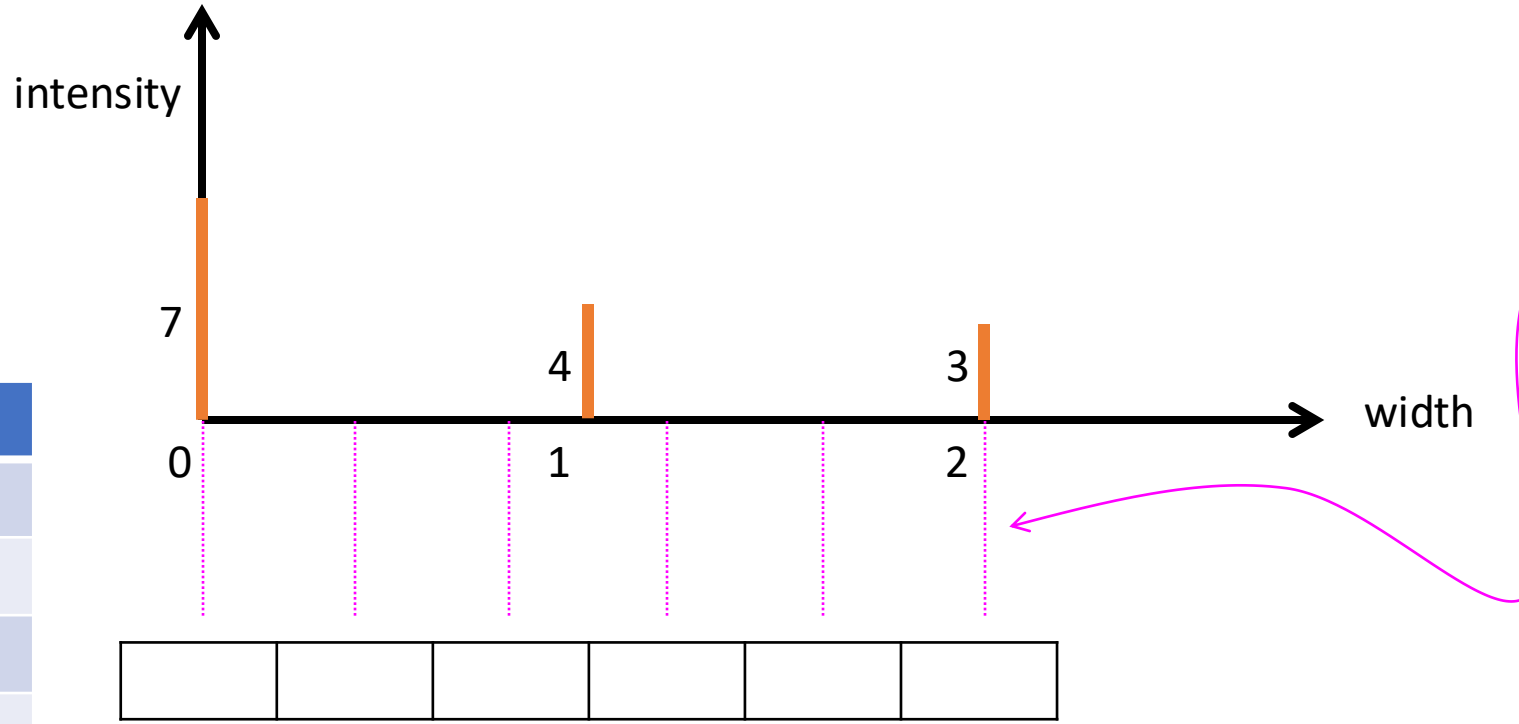
We need to (re-sample) at 6 locations

Resampling pixel locations

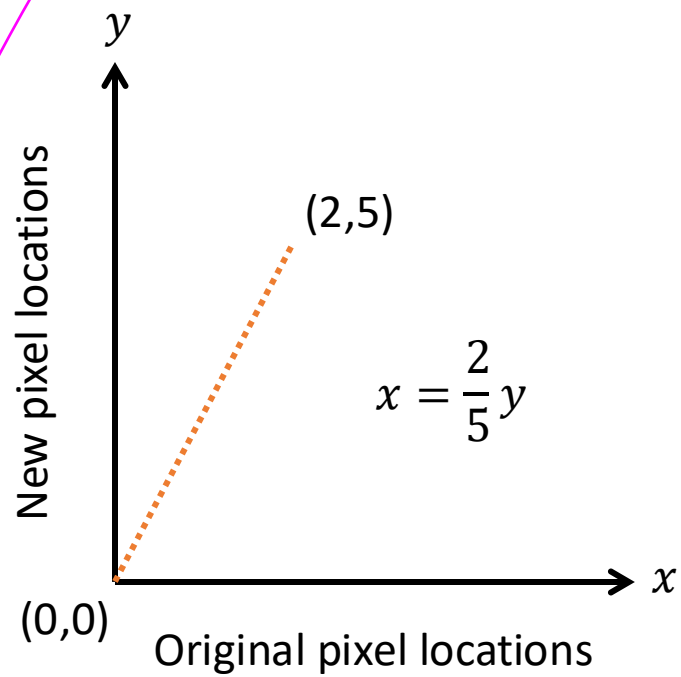


Resampling pixel locations



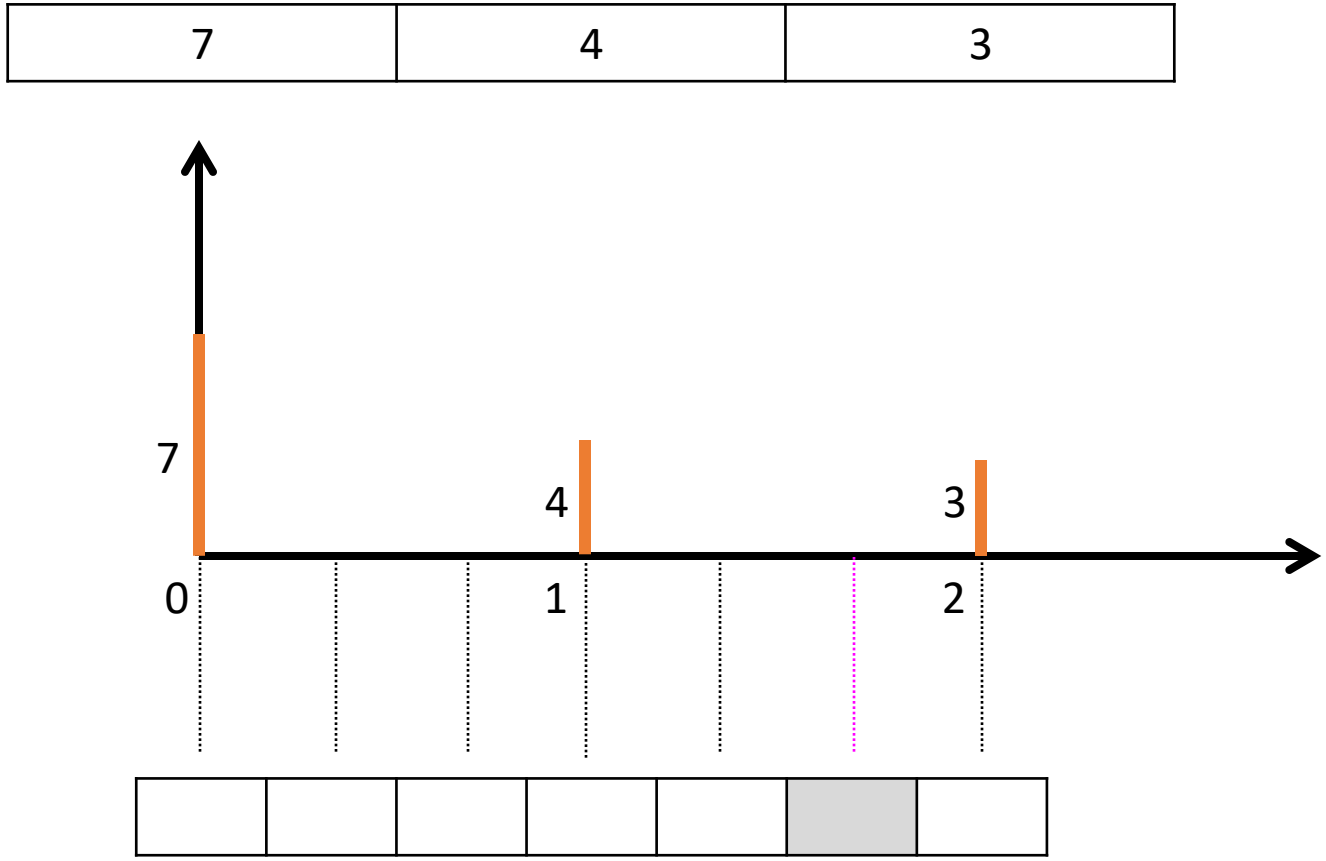


How do we compute location values for the 6 pixels between 0 and 2?



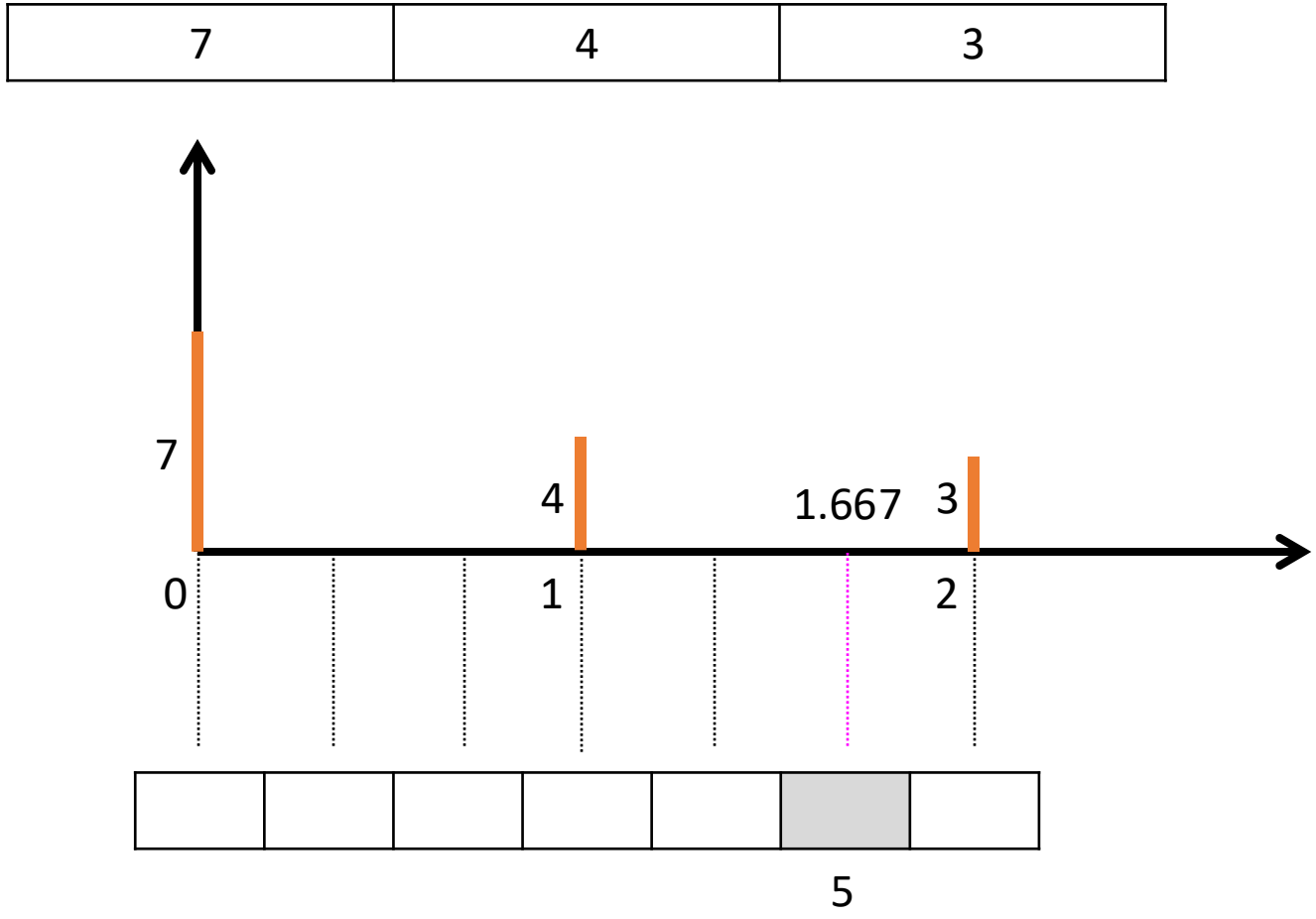
y	x
0	0
1	2/5
2	4/5
3	6/5
4	8/5
5	1

Resampling pixel locations



What is the location (between 0 and 2) of the shaded pixel?

Resampling pixel locations



What is the location (between 0 and 2) of the shaded pixel?

Given

Last pixel location in original image = 2
Last pixel location in resulting image = 6

Use the following relationship (that we developed in previous slides):

$$x = \frac{2}{6}y$$

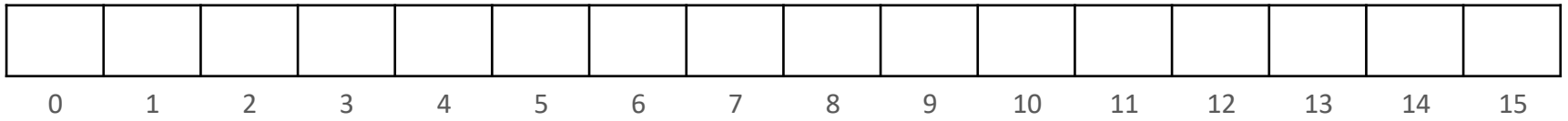
Sample location is

$$x = \frac{2}{6}(5) = 1.667$$

Resampling pixel locations

Consider a 16-pixel 1D image. You are asked to resize it to a 5-pixel 1D image. What is the location of pixel 2 (between 0 and 15) of the new image?

Original image

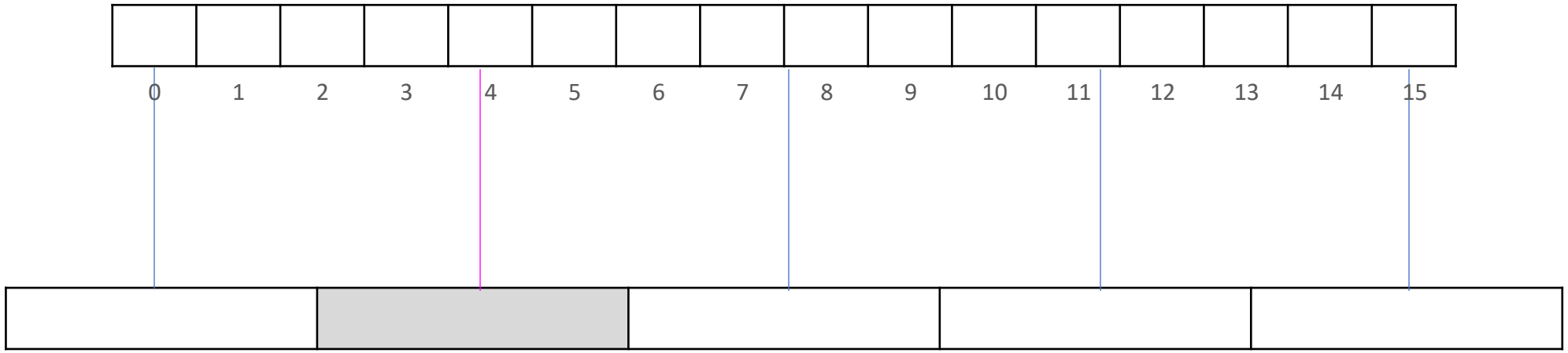


Resized image

Downscaling

Resampling pixel locations

Consider a 16-pixel 1D image. You are asked to resize it to a 5-pixel 1D image. What is the location of pixel 2 (between 0 and 15) of the new image?



Given

Last pixel location in original image = 15
Last pixel location in resulting image = 4

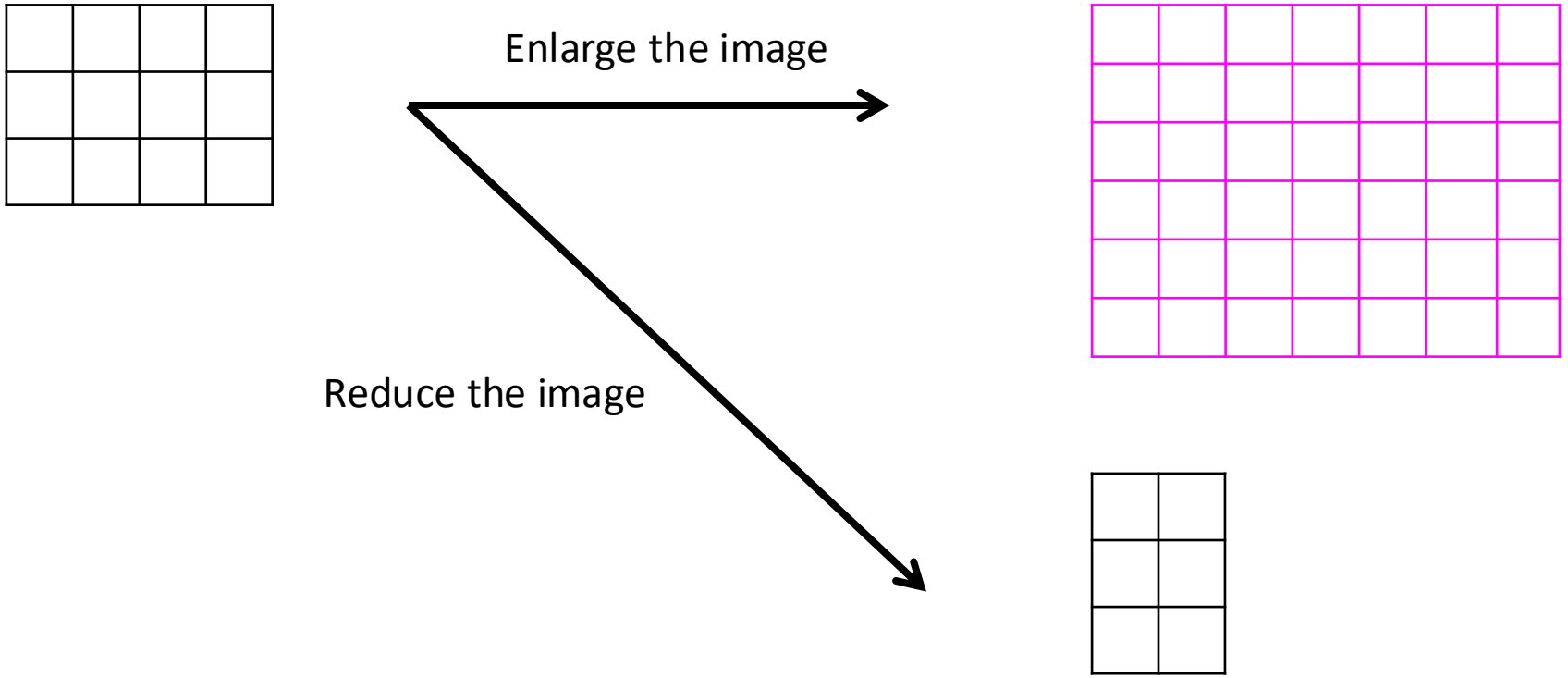
Use the relationship developed earlier

$$x = \frac{15}{4}y$$

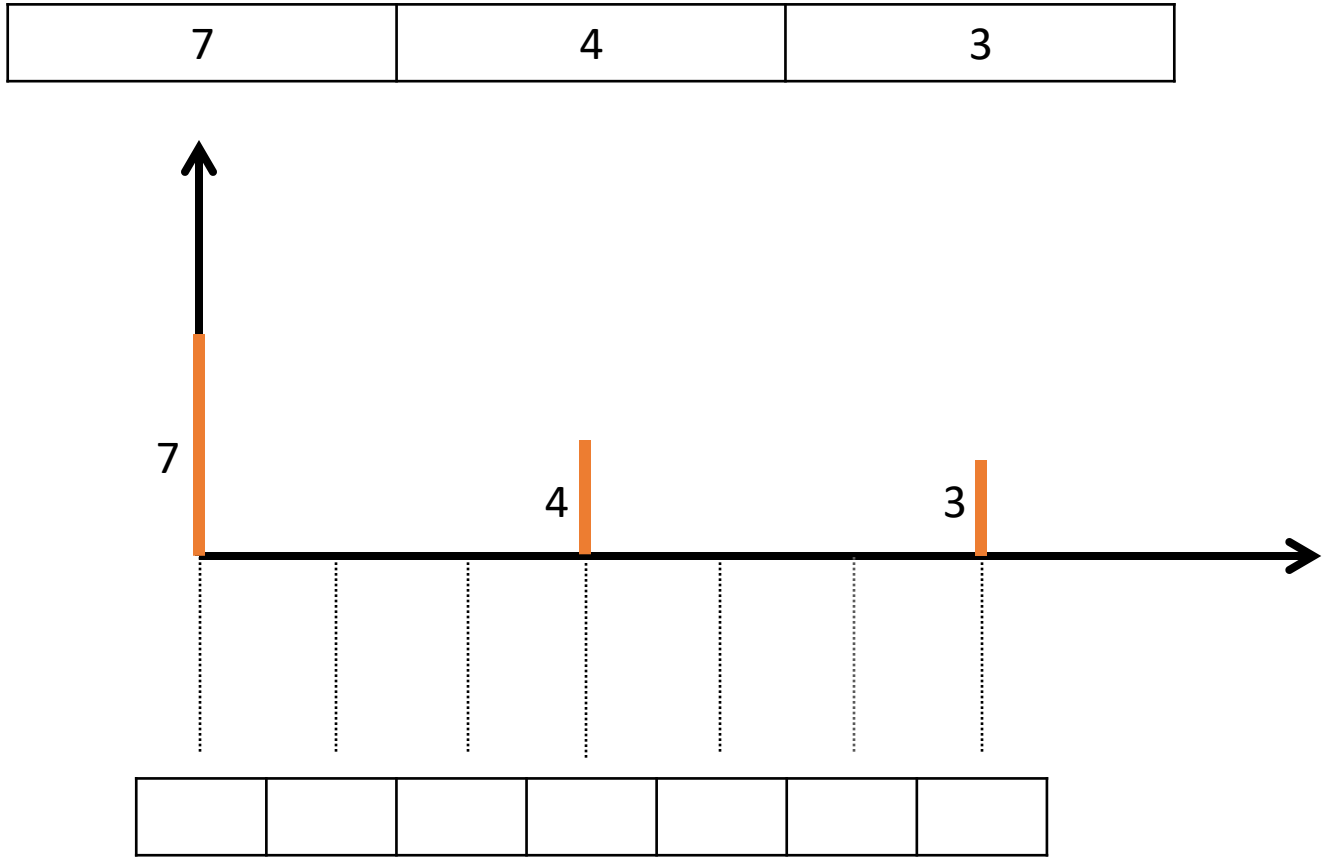
Sample location is

$$x = \frac{15}{4}(1) = 3.75 \approx 4$$

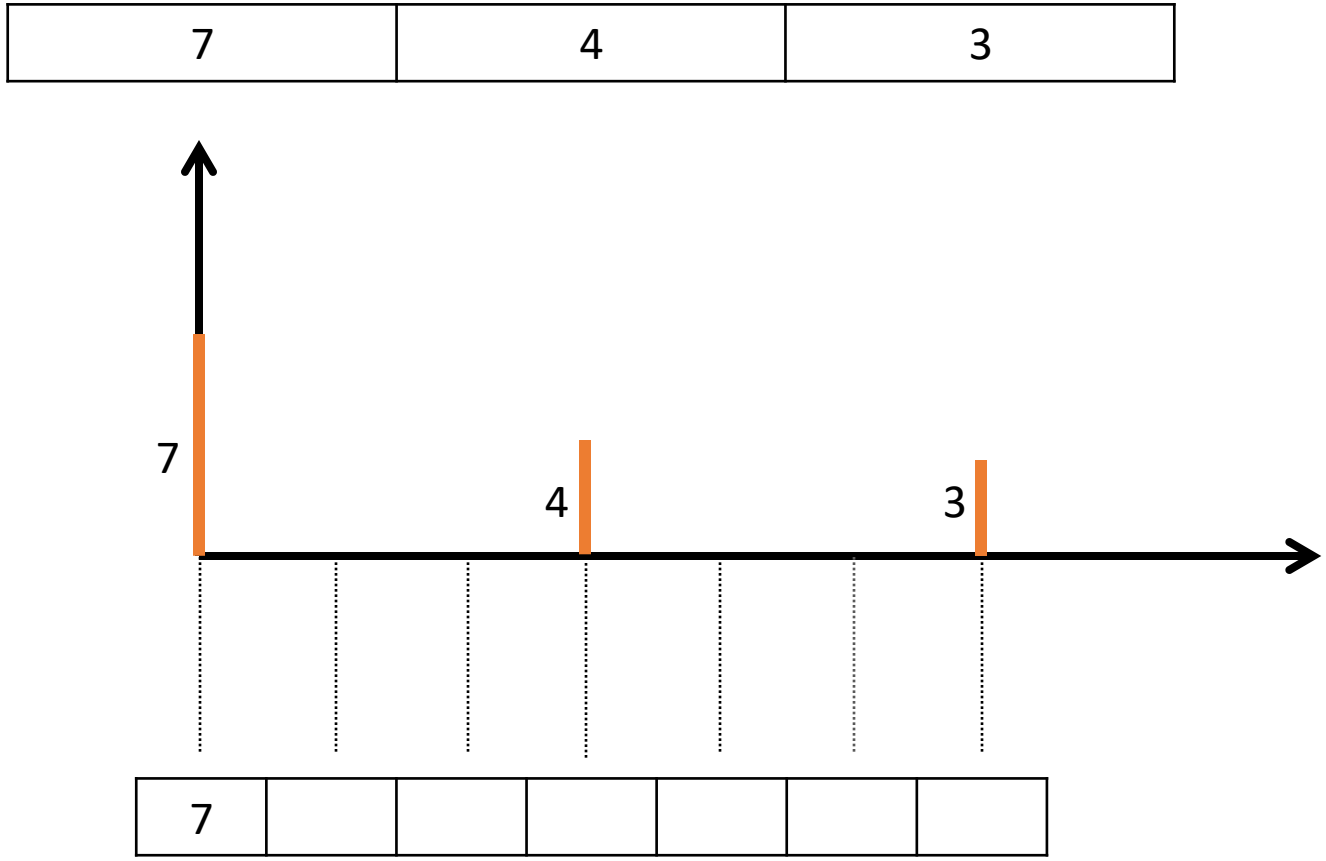
Resampling pixel locations (in 2D)



How to compute new pixel values?

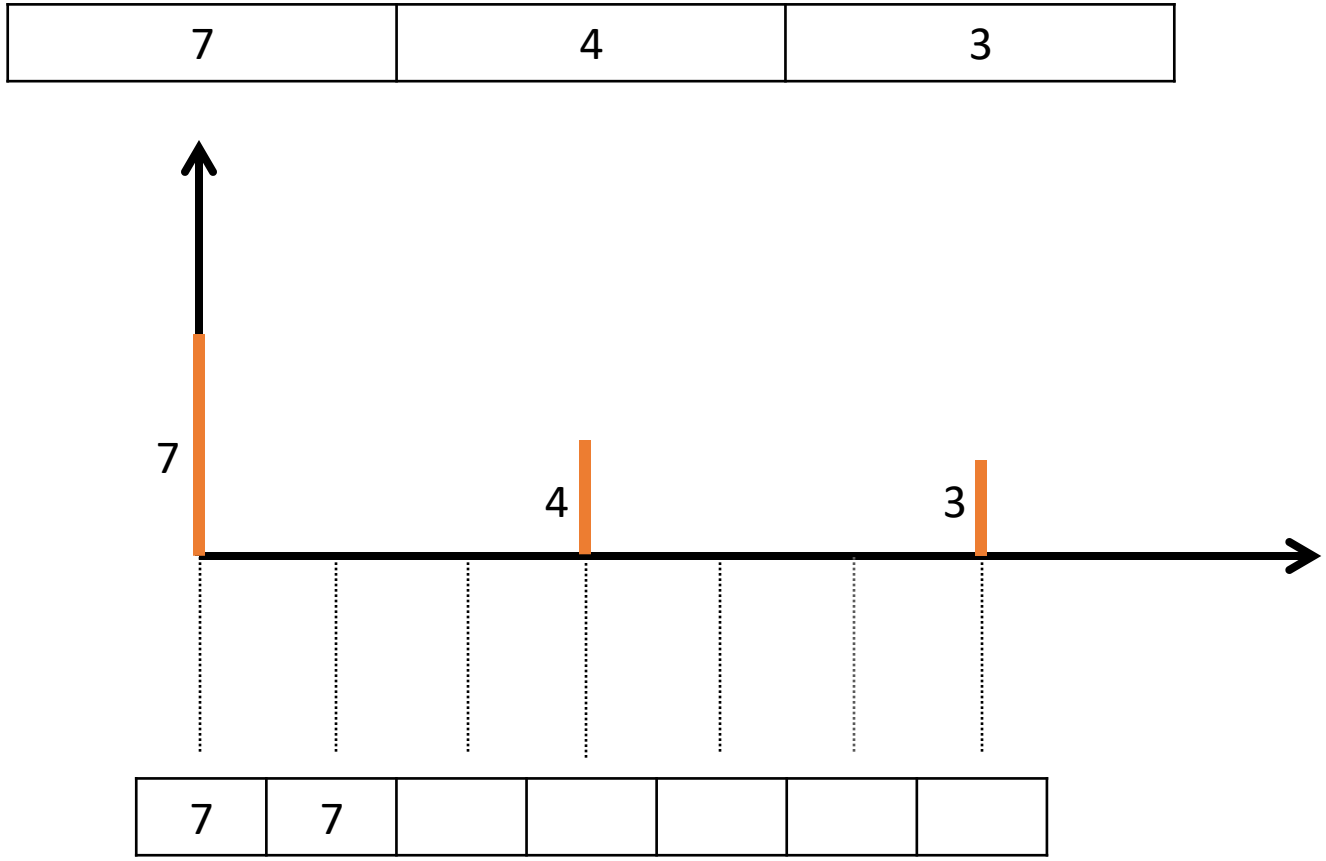


How to compute new pixel values?



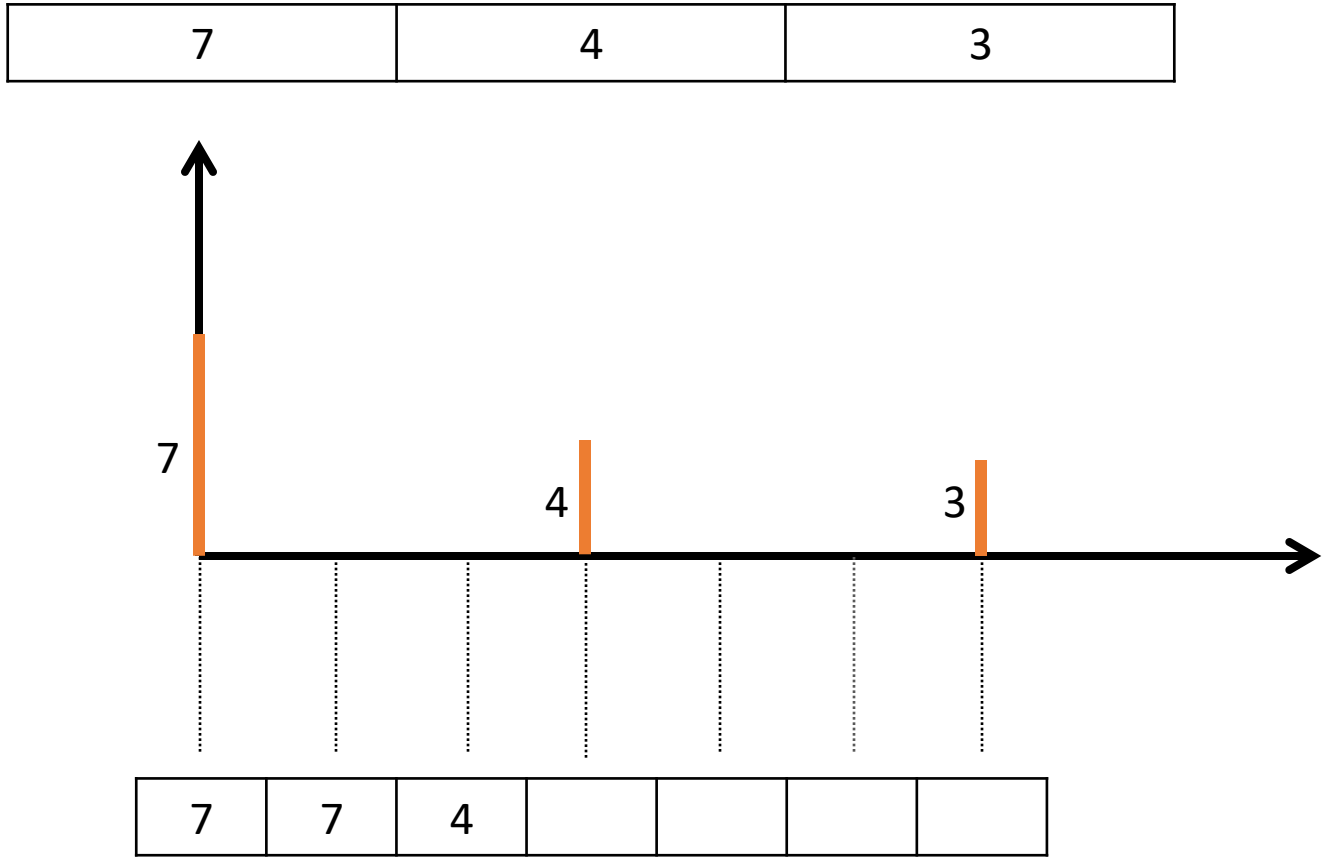
Nearest-Neighbor Interpolation

How to compute new pixel values?



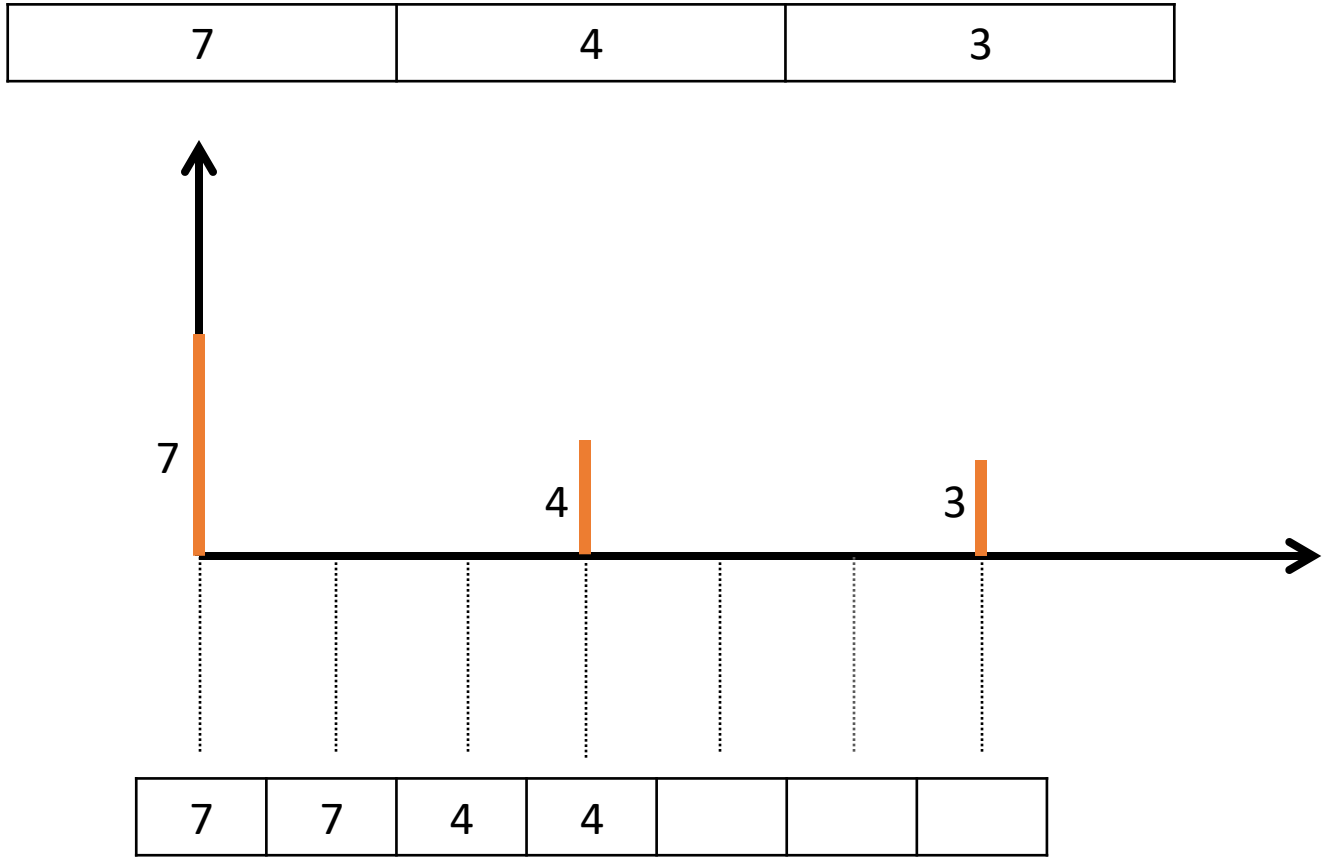
Nearest-Neighbor Interpolation

How to compute new pixel values?



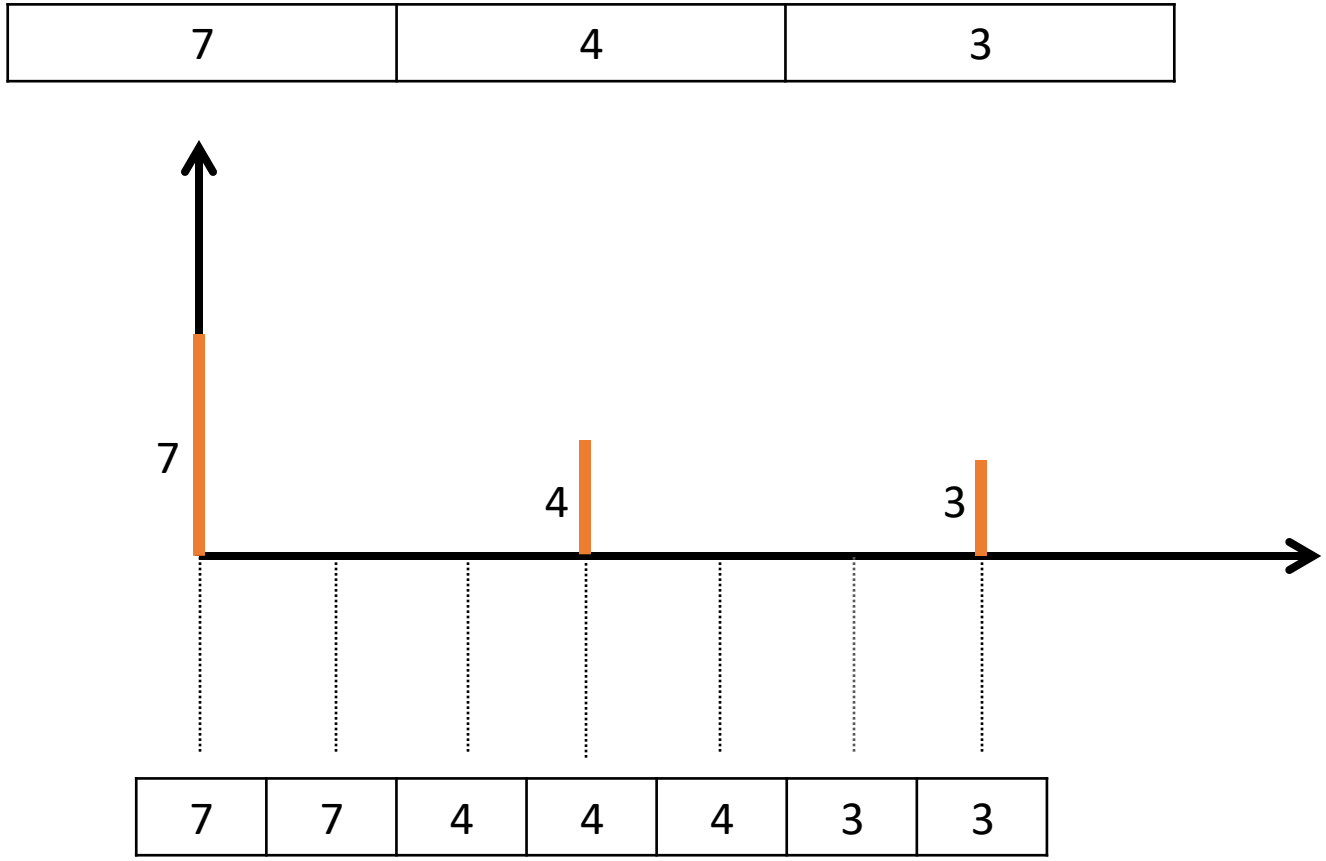
Nearest-Neighbor Interpolation

How to compute new pixel values?



Nearest-Neighbor Interpolation

How to compute new pixel values?



Nearest-Neighbor Interpolation

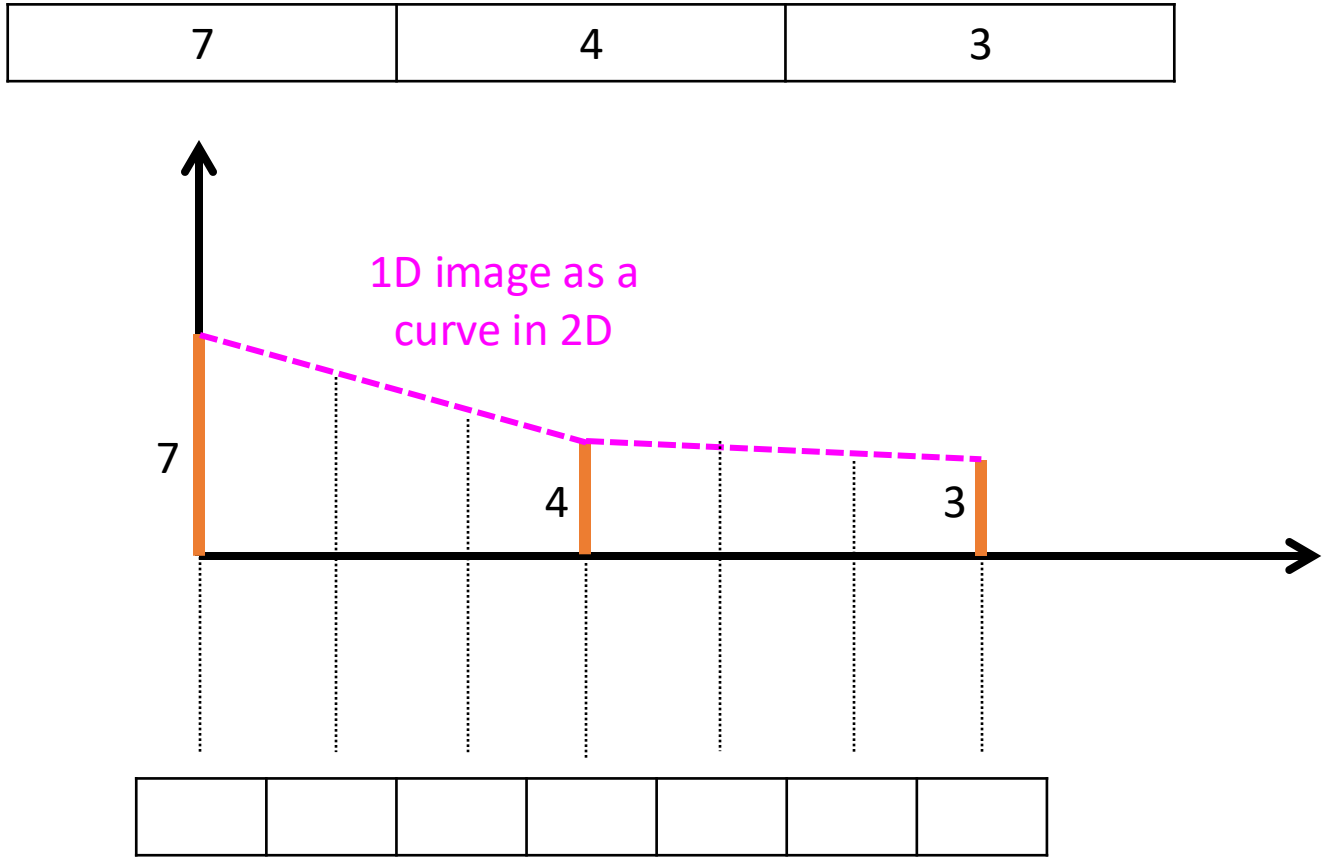
Nearest-Neighbor Interpolation

- Easy to implement.
- Results in blocky or pixelated results
- Does not consider neighboring pixels
- Losses details and smoothness
- Use other methods, e.g., bilinear, bicubic, etc., for higher-quality image resizing

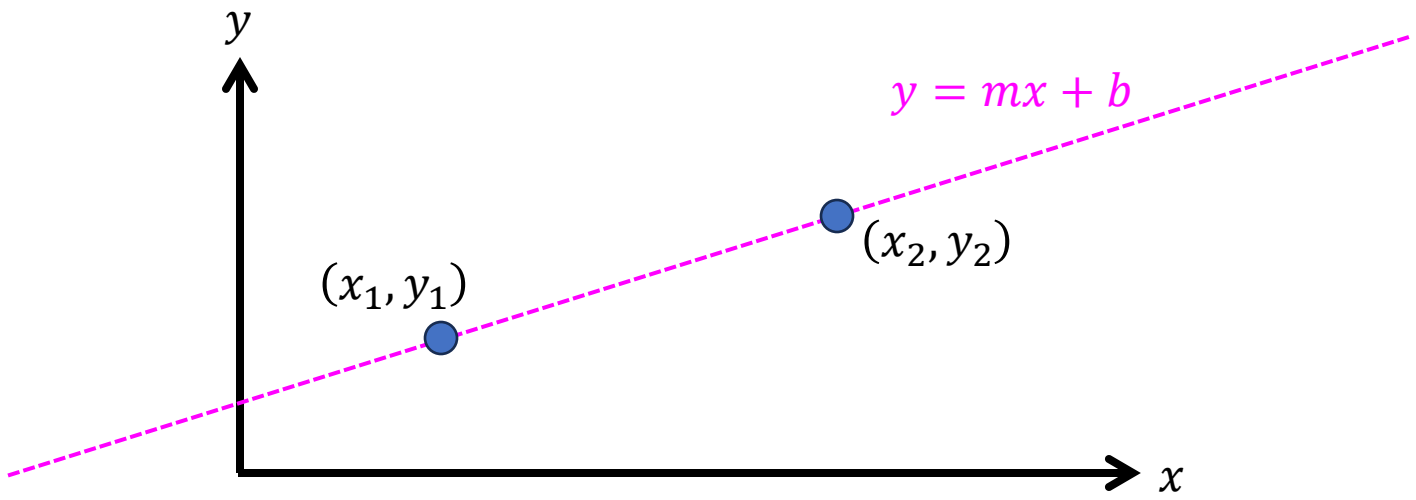
Nearest-Neighbor Interpolation

Nearest-Neighbor Interpolation

Linear Interpolation



2D Line Fitting

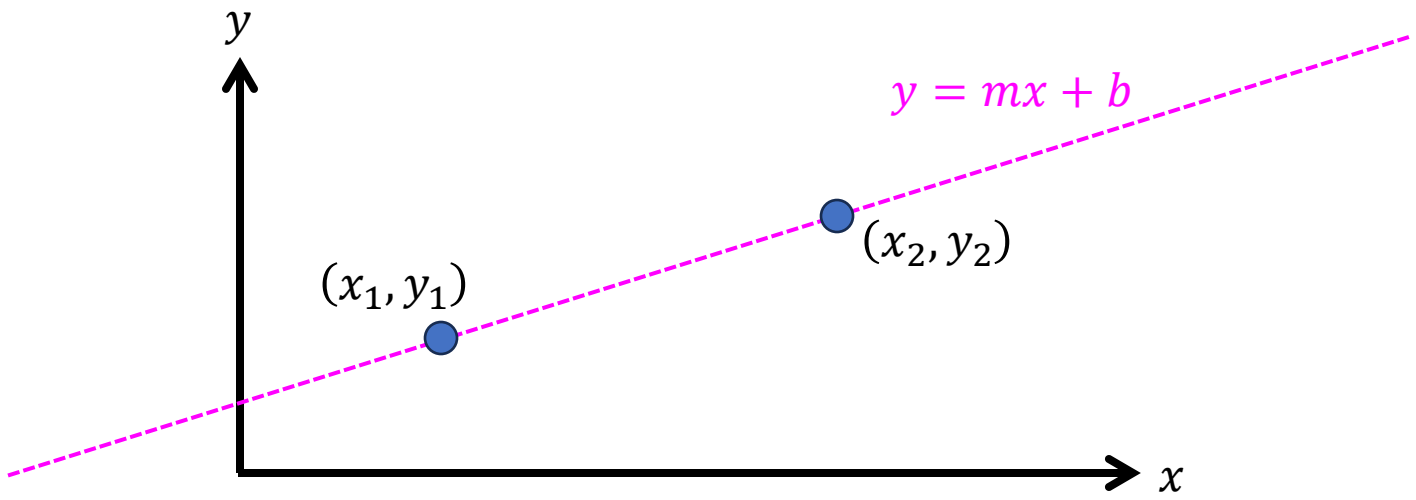


2D Line Fitting

A line between (x_1, y_1) and (x_2, y_2) is given by:

$$\frac{y - y_1}{y_2 - y_1} = \frac{x - x_1}{x_2 - x_1}$$

2D Line Fitting



2D Line Fitting

A line between (x_1, y_1) and (x_2, y_2) is given by:

$$\frac{y - y_1}{y_2 - y_1} = \frac{x - x_1}{x_2 - x_1}$$

Re-writing yields

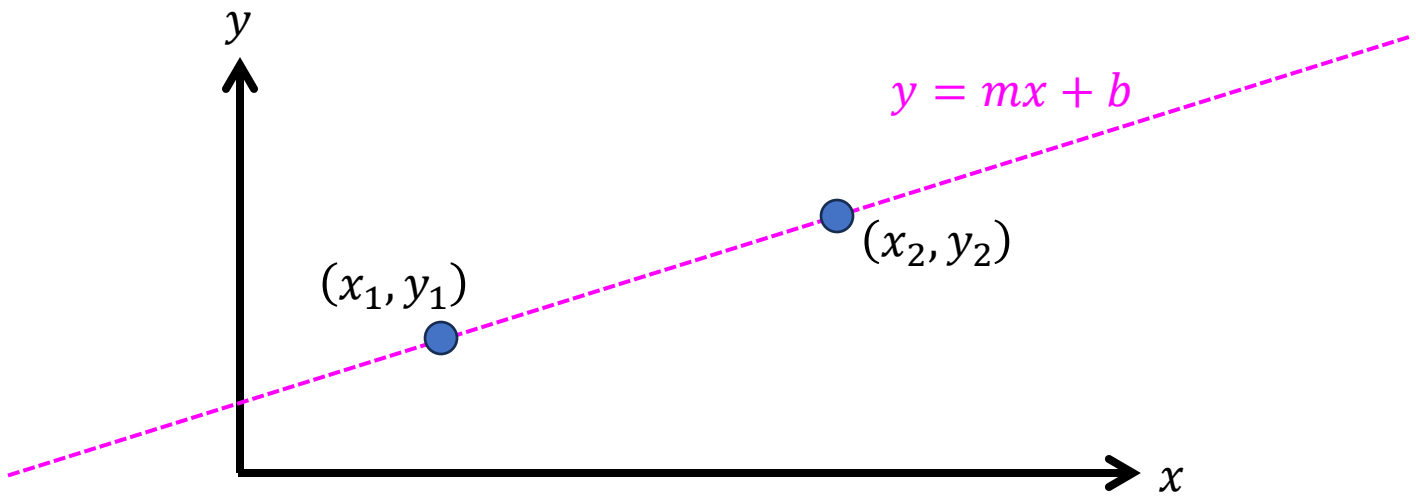
$$y = \frac{(y_2 - y_1)}{(x_2 - x_1)} x - \frac{(y_2 - y_1)}{(x_2 - x_1)} x_1 + y_1$$

Therefore

$$m = \frac{(y_2 - y_1)}{(x_2 - x_1)}$$

$$b = -\frac{(y_2 - y_1)}{(x_2 - x_1)} x_1 + y_1$$

2D Line Fitting



2D Line Fitting

A line between (x_1, y_1) and (x_2, y_2) is given by:

Matrix form

Re-write

$$x_1 m + b = y_1$$

$$x_2 m + b = y_2$$

as

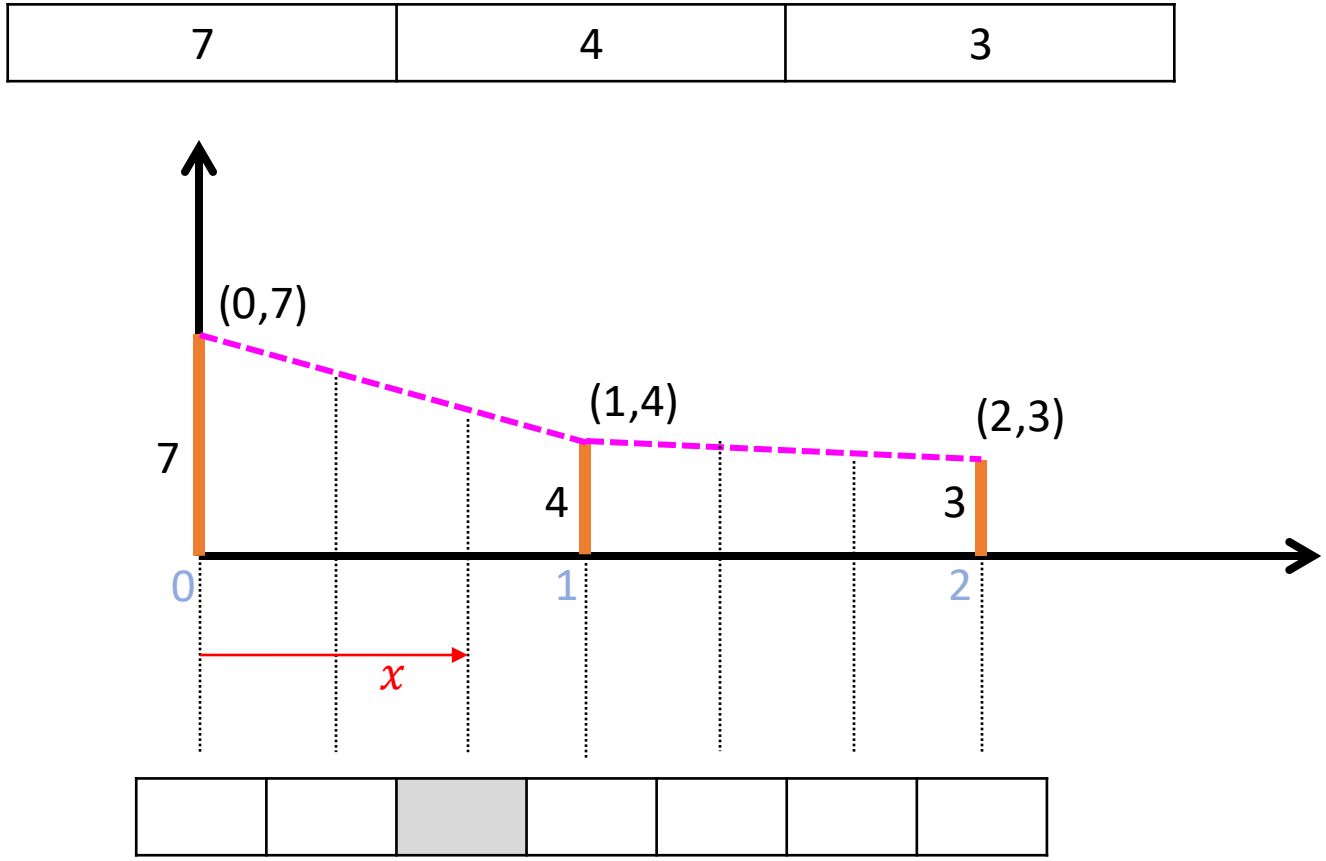
$$\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

$$\begin{bmatrix} m \\ b \end{bmatrix} = \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \end{bmatrix}^{-1} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

where

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

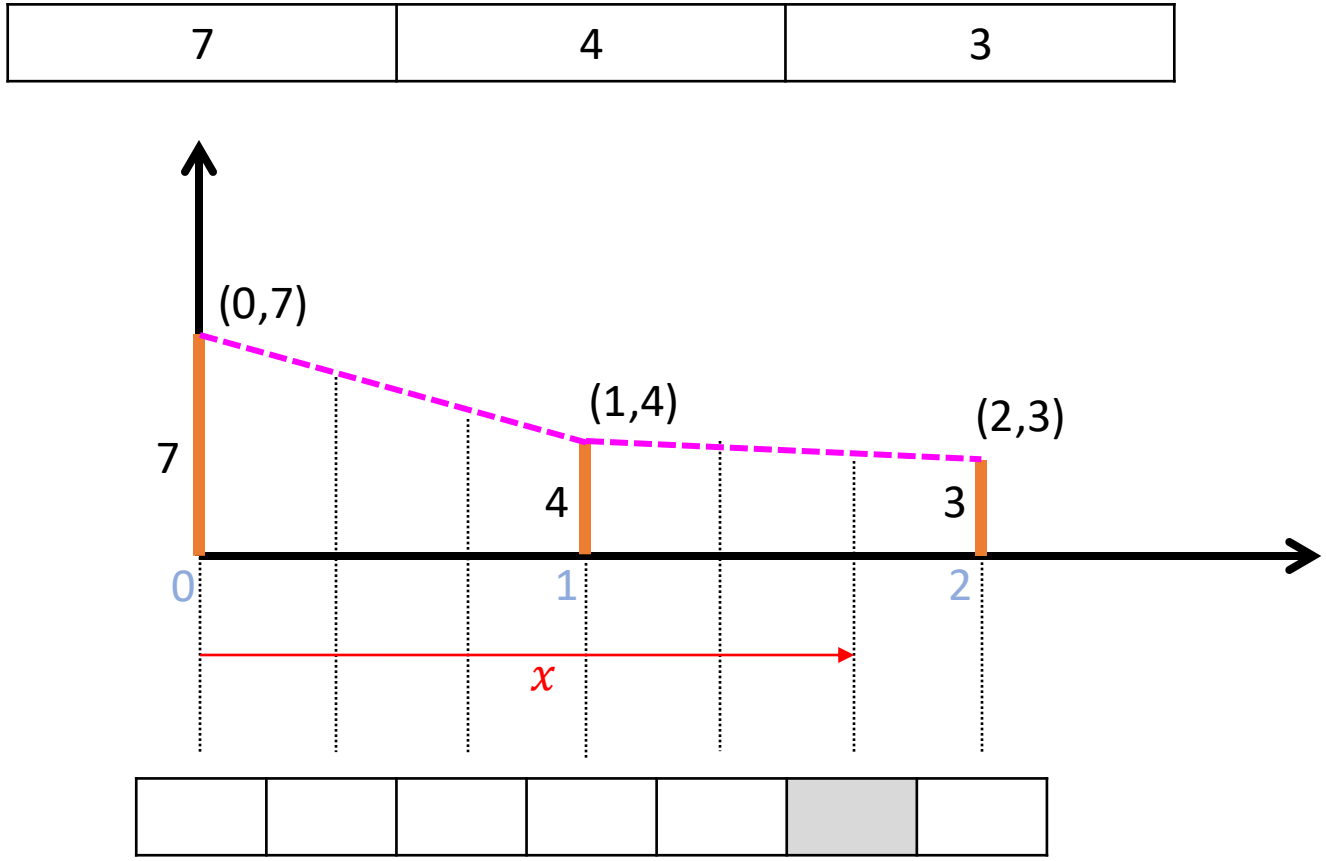
Linear Interpolation



Compute the value for shaded pixel?

Setup a line between (0,7) and (1,4) and evaluate it at x value corresponding to the shaded location

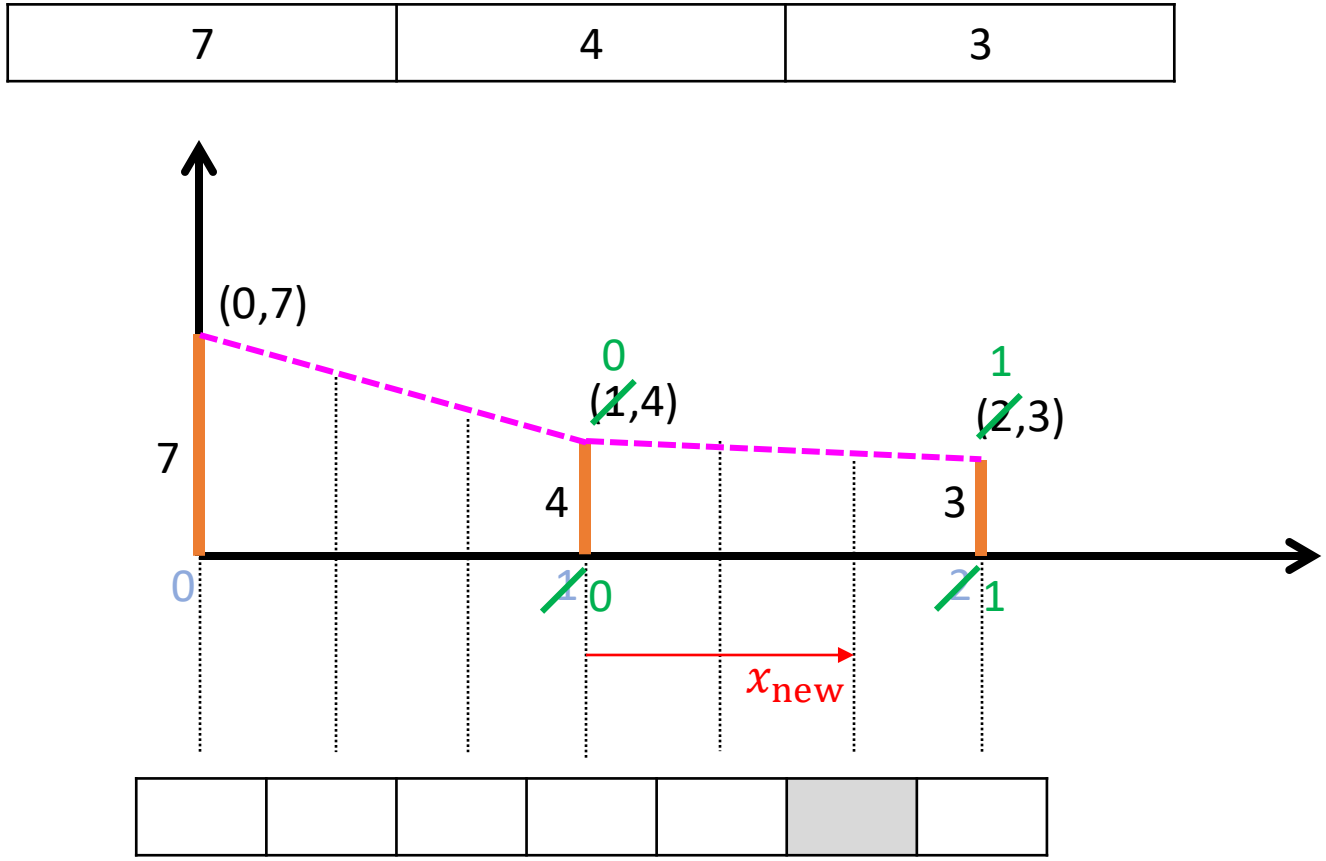
Linear Interpolation



Compute the value for shaded pixel?

Setup a line between (1,4) and (2,3) and evaluate it at x value corresponding to the shaded location

Linear Interpolation



Compute the value for shaded pixel?

Setup a line between (1,4) and (2,3) and evaluate it at x value corresponding to the shaded location

Trick: A common trick is to re-index the original pixel locations to 0 and 1. This simplifies the calculations.

Don't forget to update the x value. In this example $x_{\text{new}} = x - 1$.

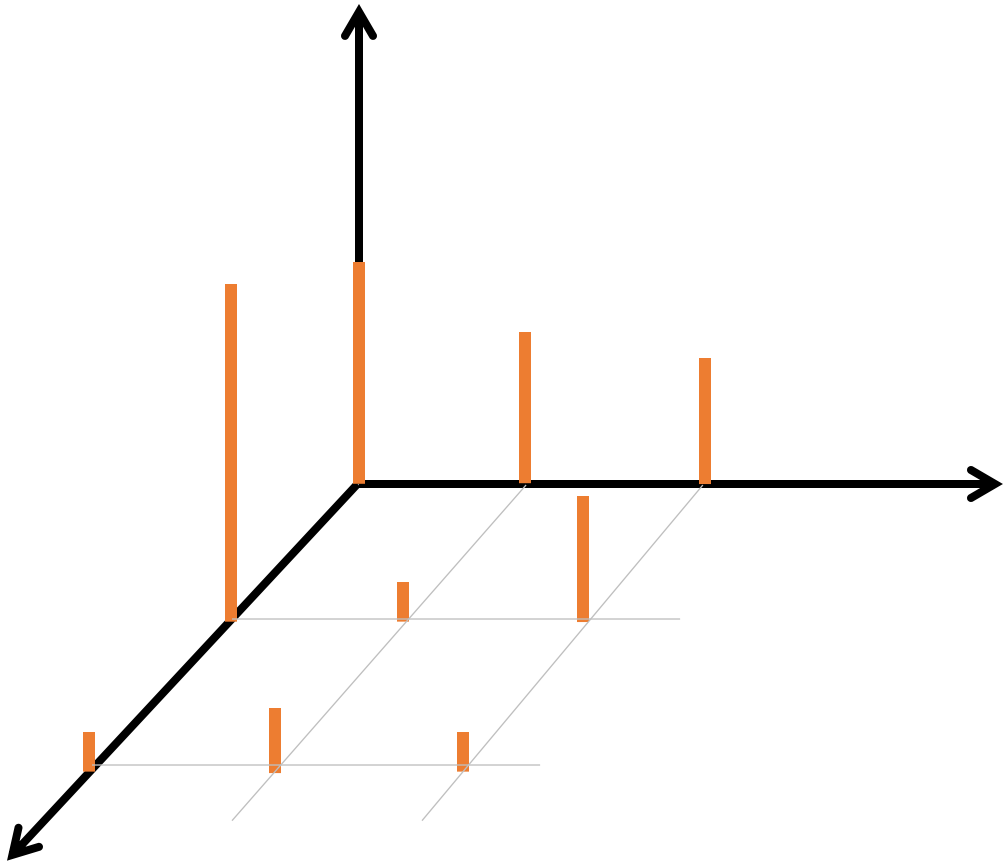
Now we setup a line between (0,4) and (1,3) and evaluate it at x_{new} value.

Images as surfaces

Images are not just 1D. How do we deal with a 2D image?

Consider the following 3x3 2D image.

7	4	3
9	1	3
1	2	1

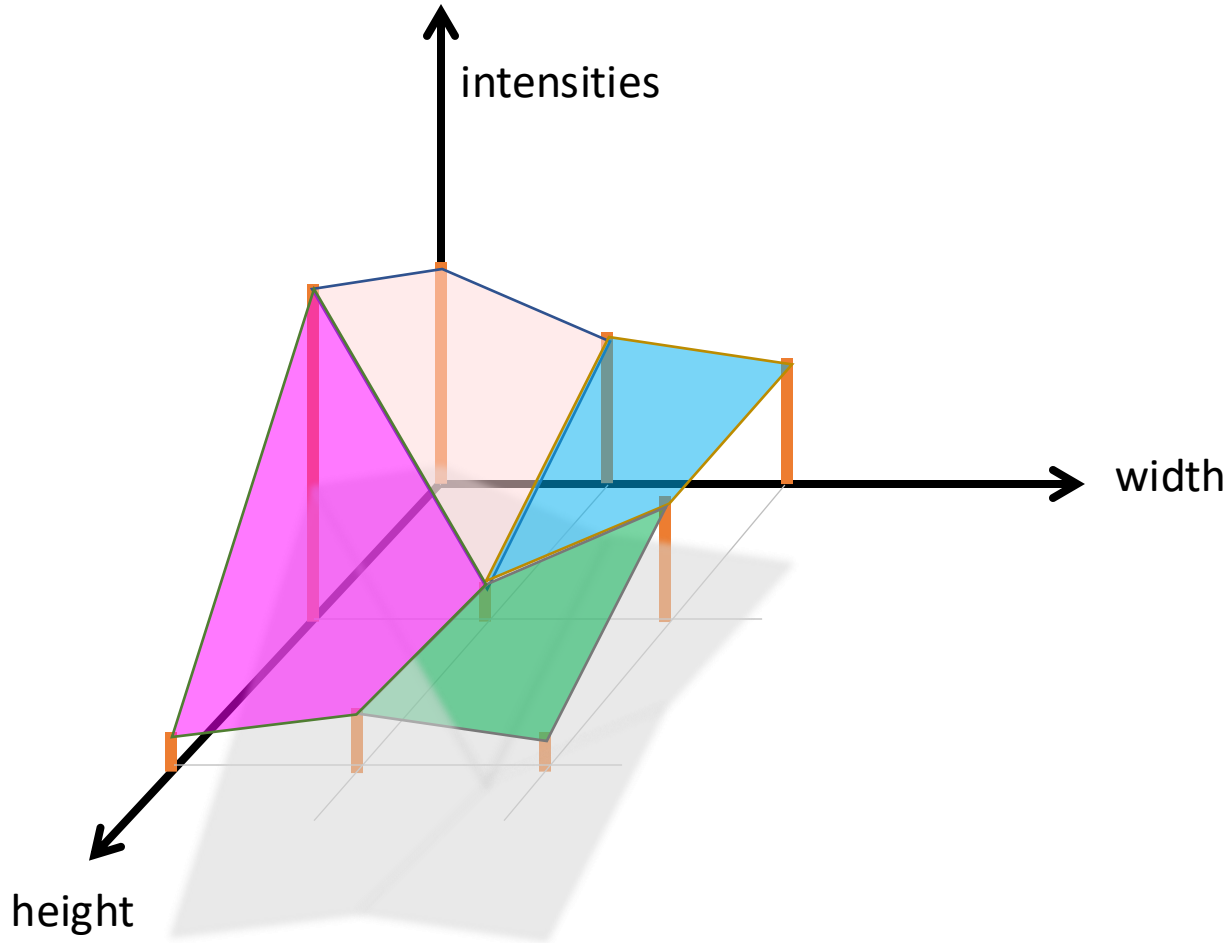


Images as surfaces

Images are not just 1D. How do we deal with a 2D image?

Consider the following 3x3 2D image.

7	4	3
9	1	3
1	2	1

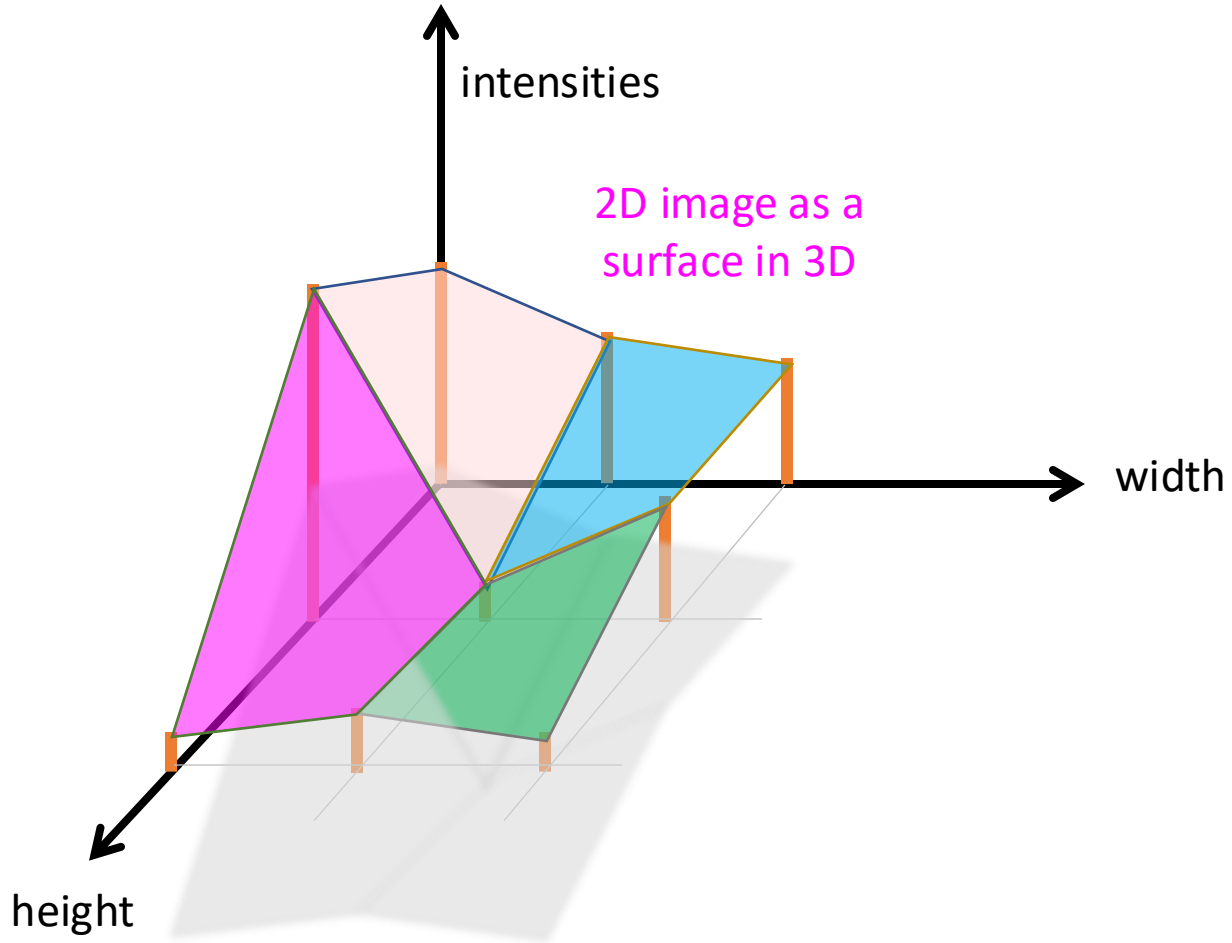


Images as surfaces

Images are not just 1D. How do we deal with a 2D image?

Consider the following 3x3 2D image.

7	4	3
9	1	3
1	2	1

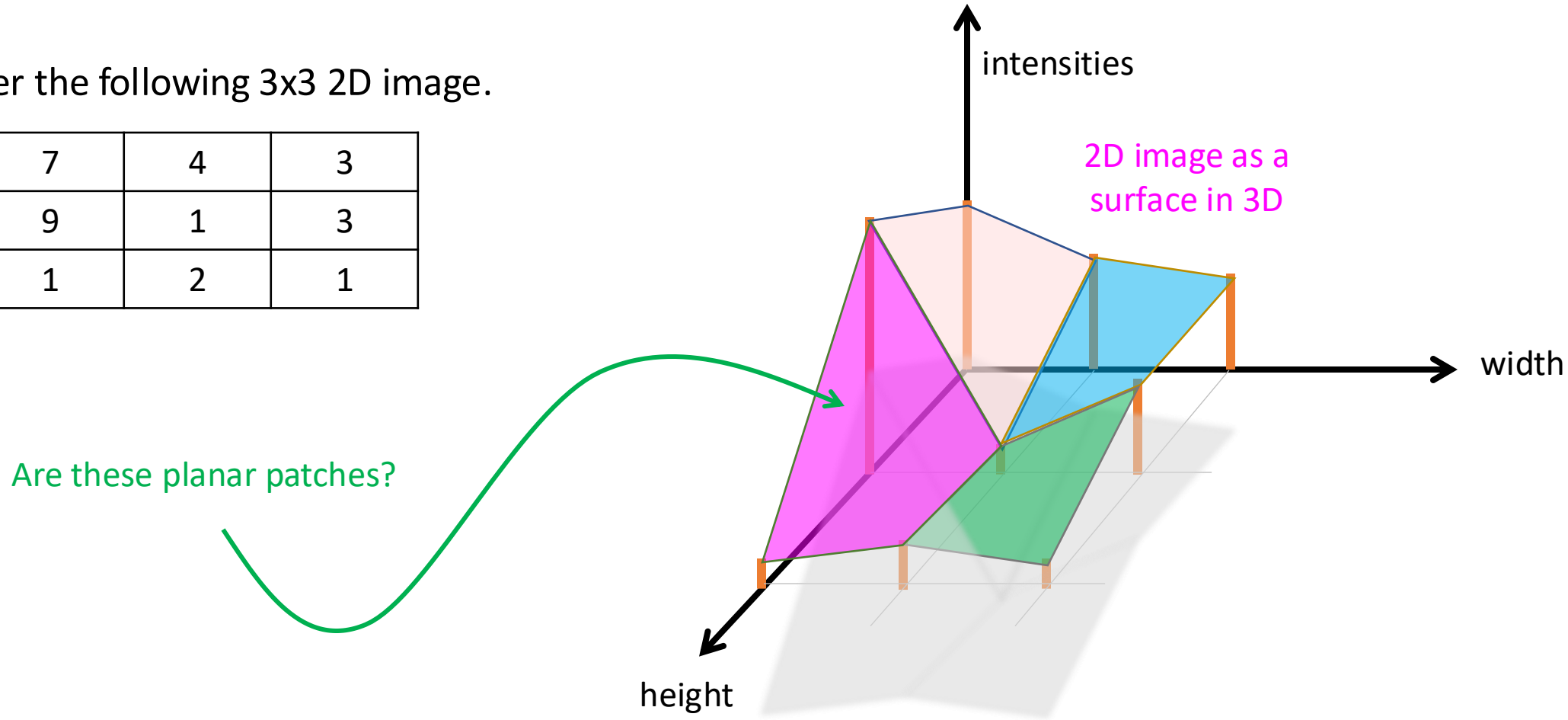


Images as surfaces

Images are not just 1D. How do we deal with a 2D image?

Consider the following 3x3 2D image.

7	4	3
9	1	3
1	2	1



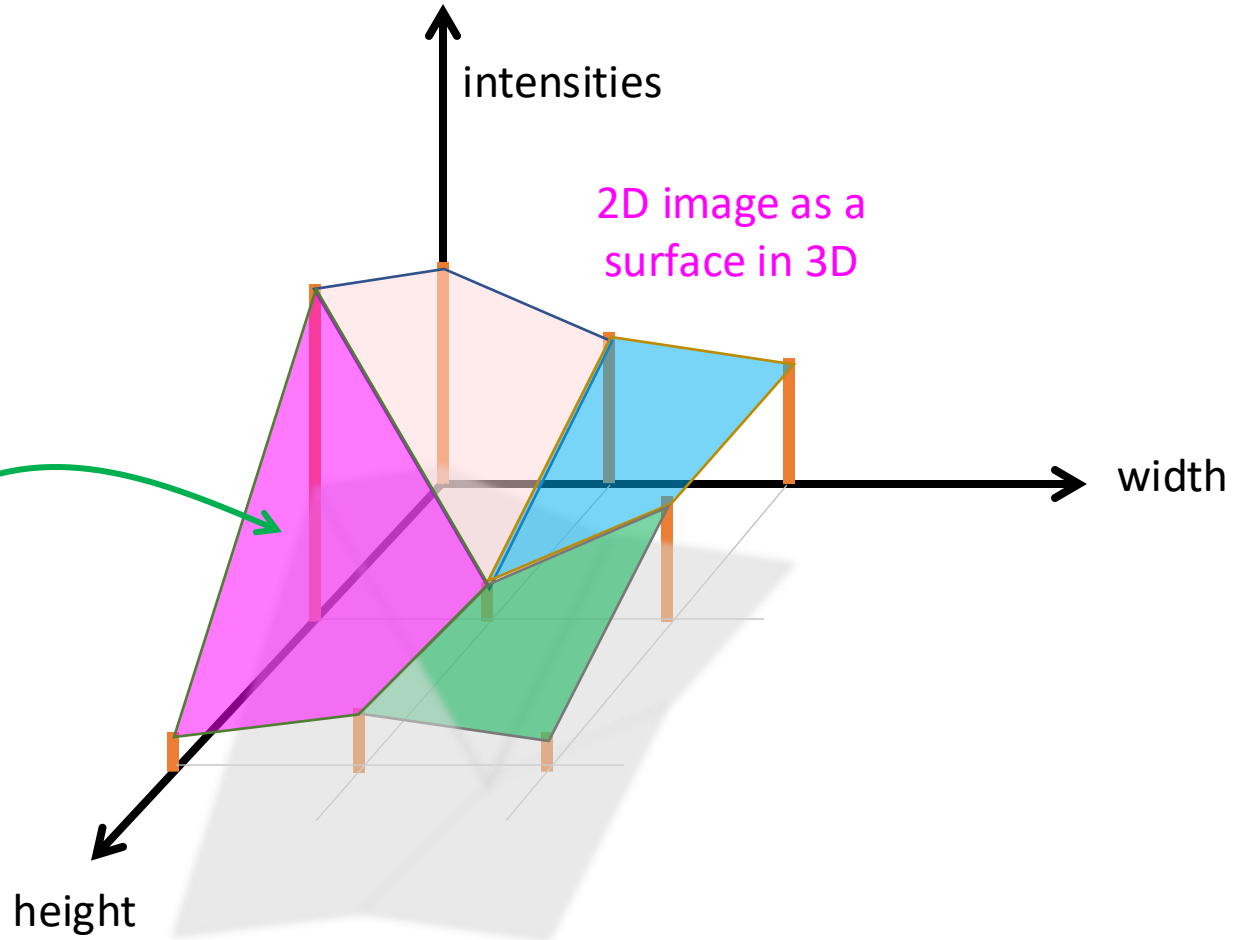
Images as surfaces

Images are not just 1D. How do we deal with a 2D image?

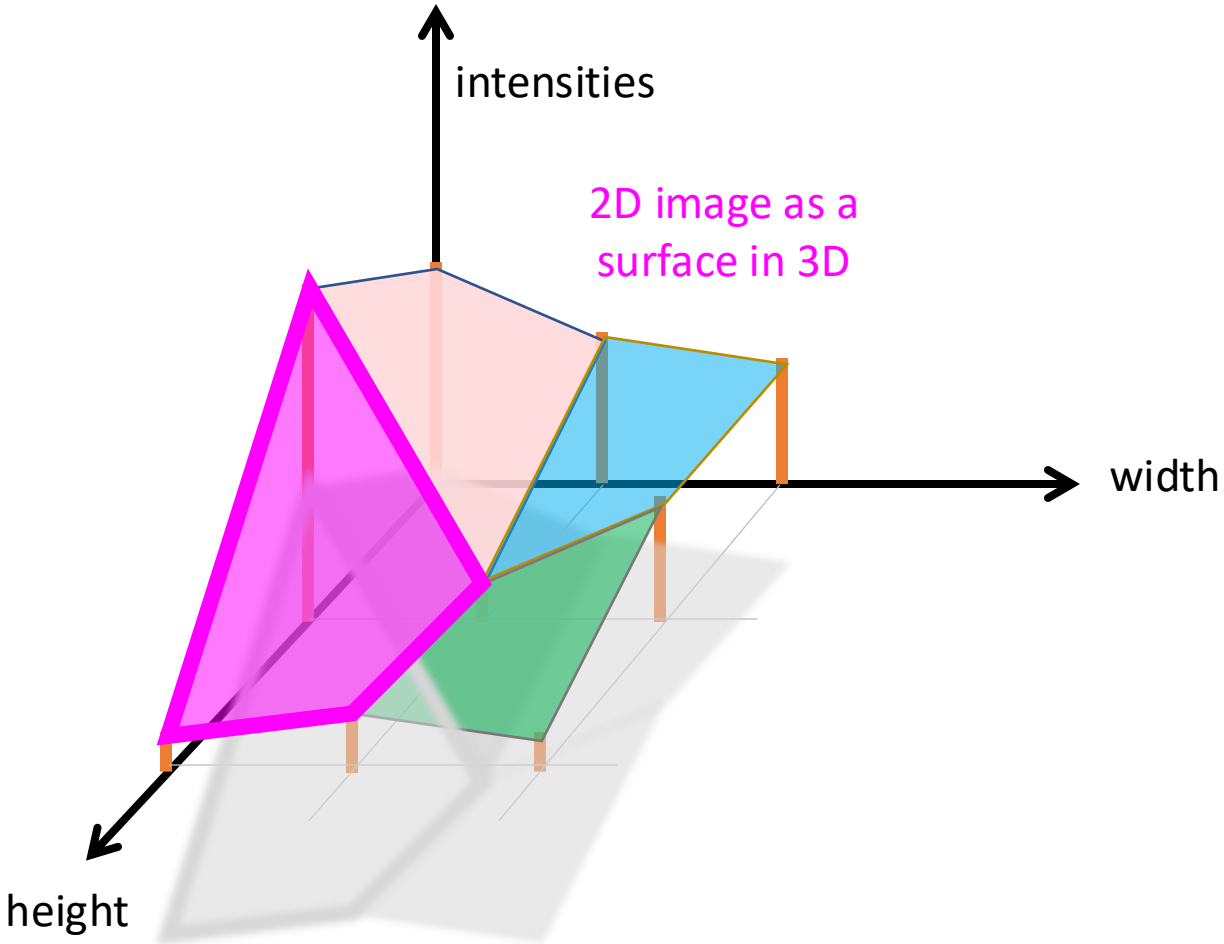
Consider the following 3x3 2D image.

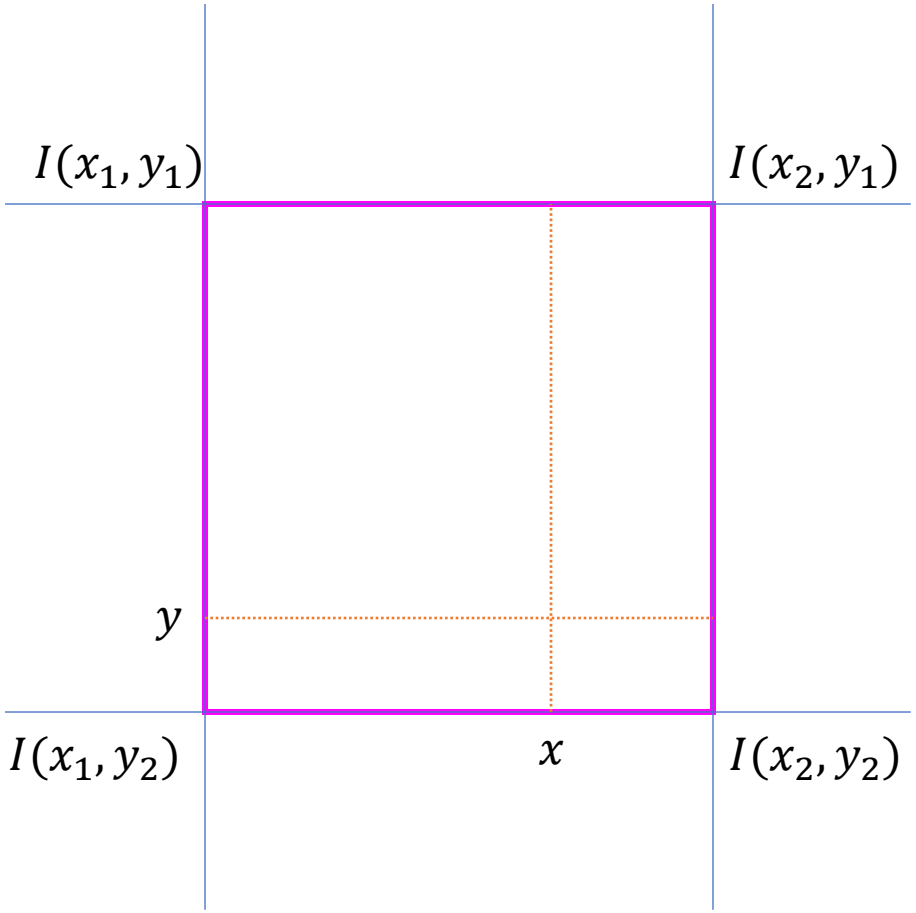
7	4	3
9	1	3
1	2	1

Are these planar patches? **No**

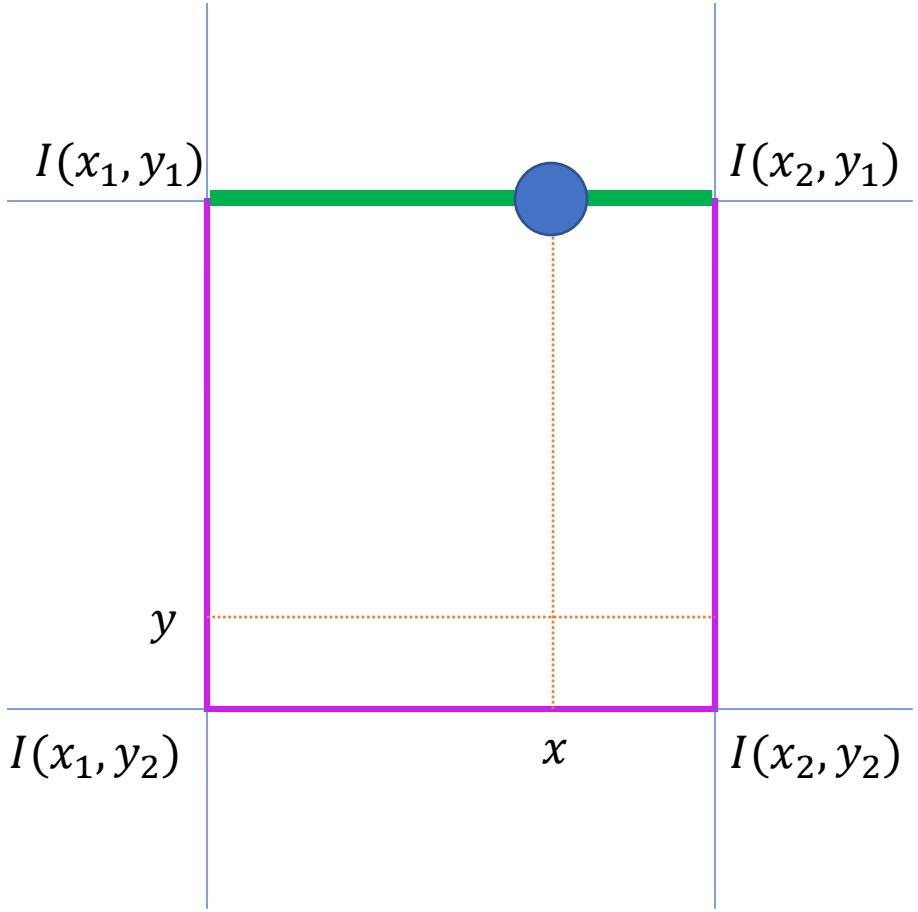


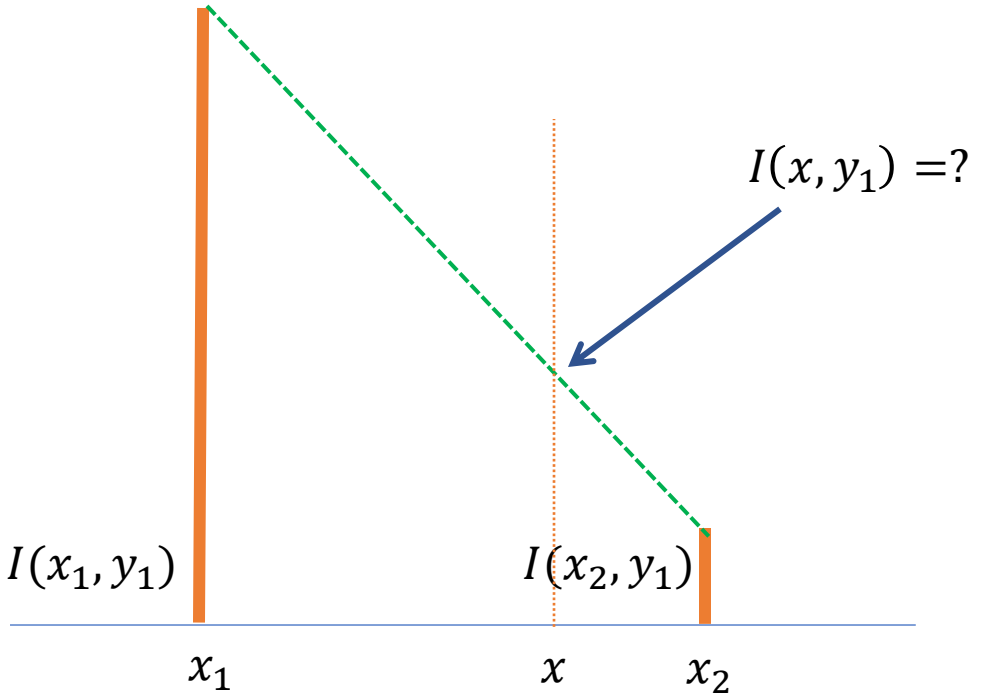
Images as surfaces



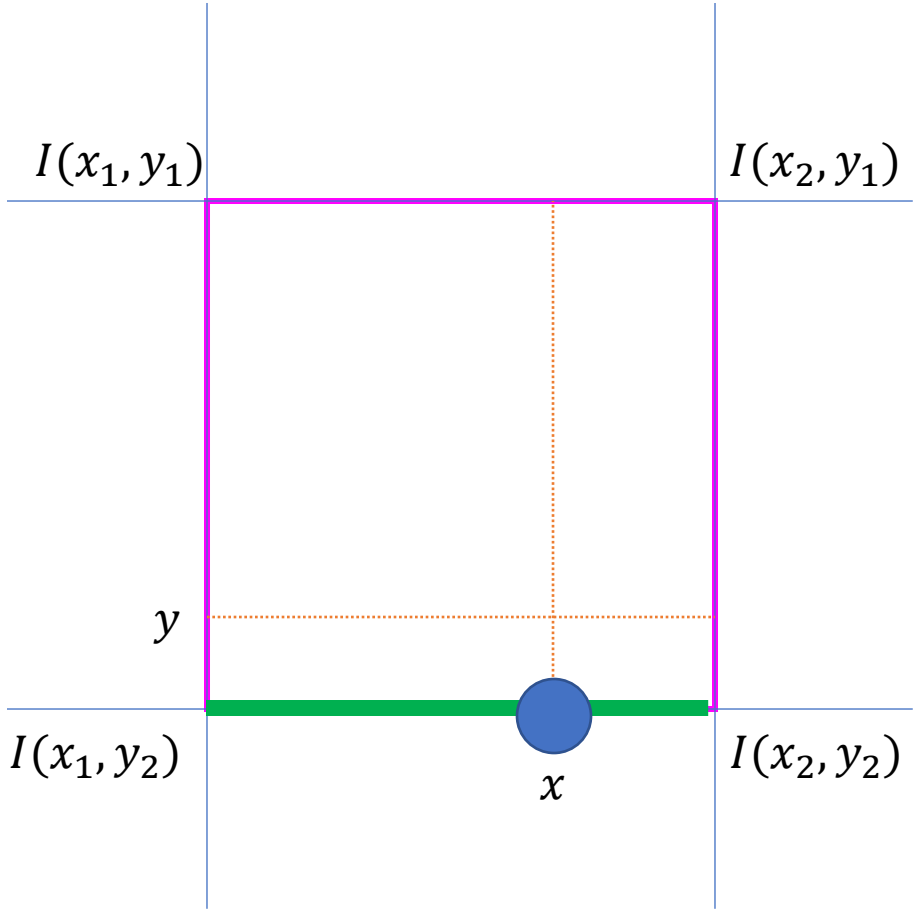


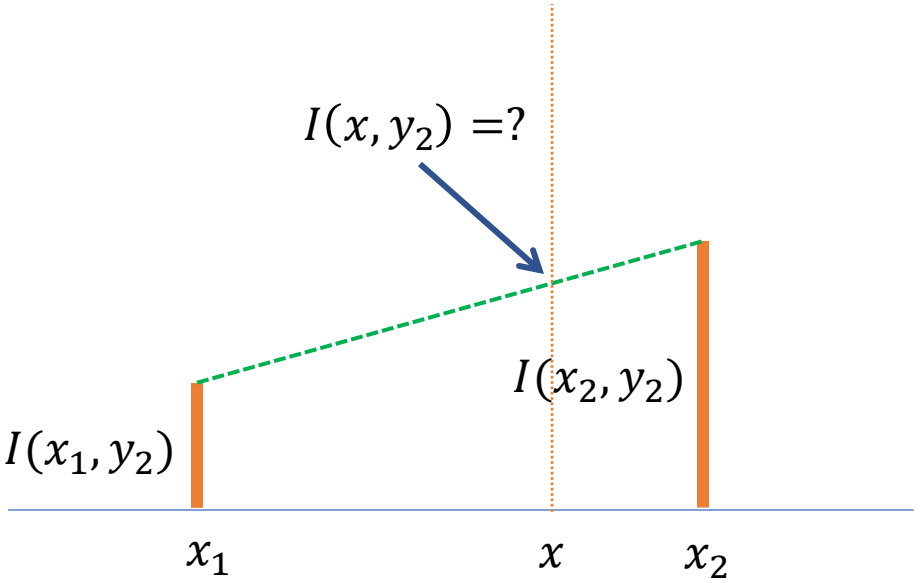
Bi-linear interpolation



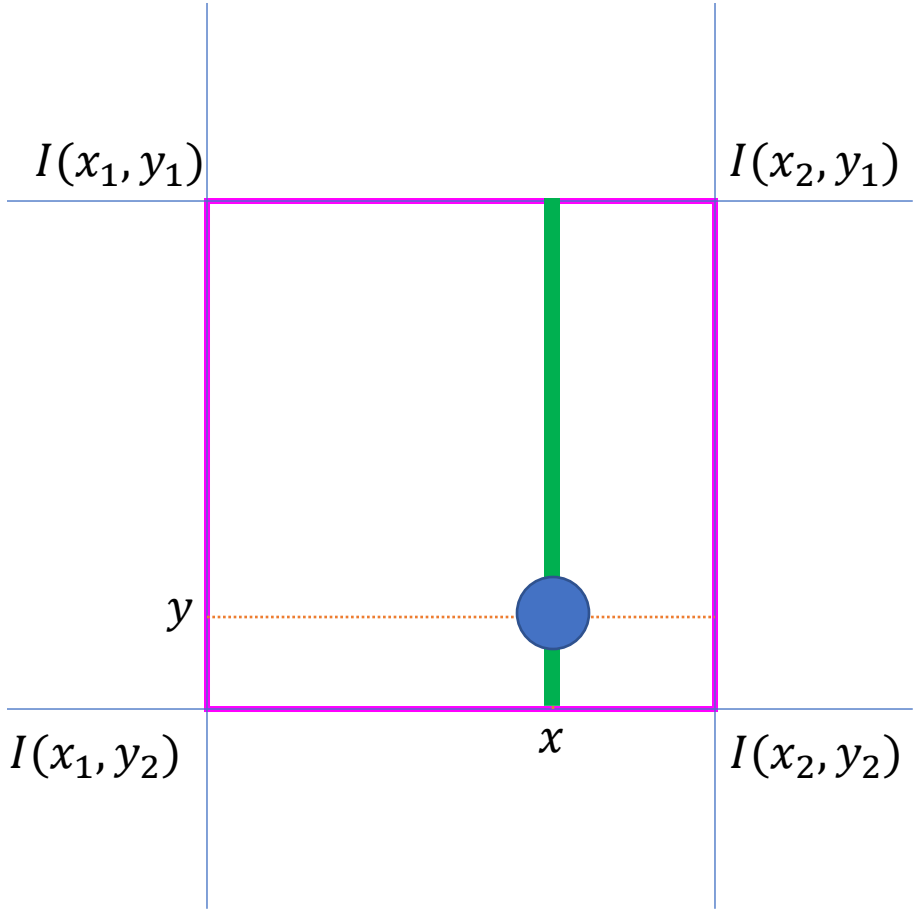


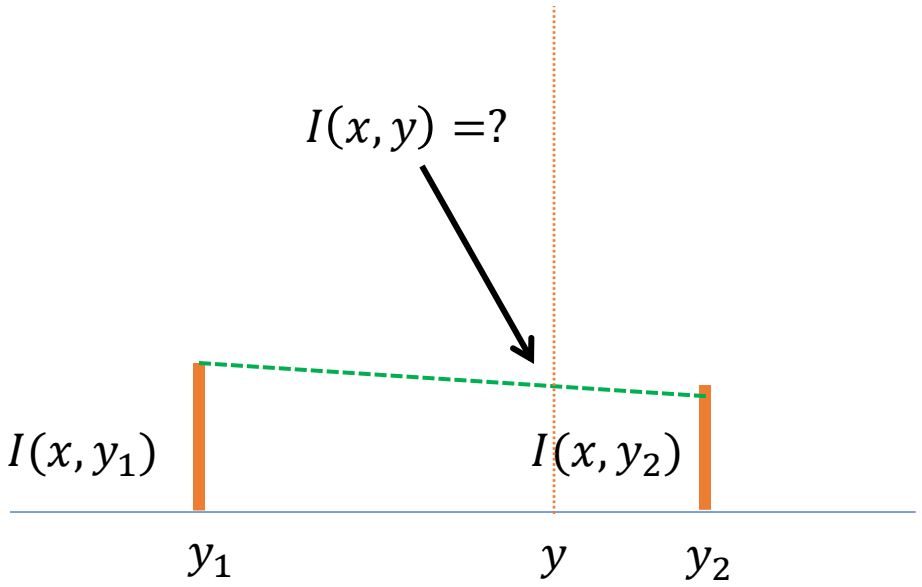
Bi-linear interpolation



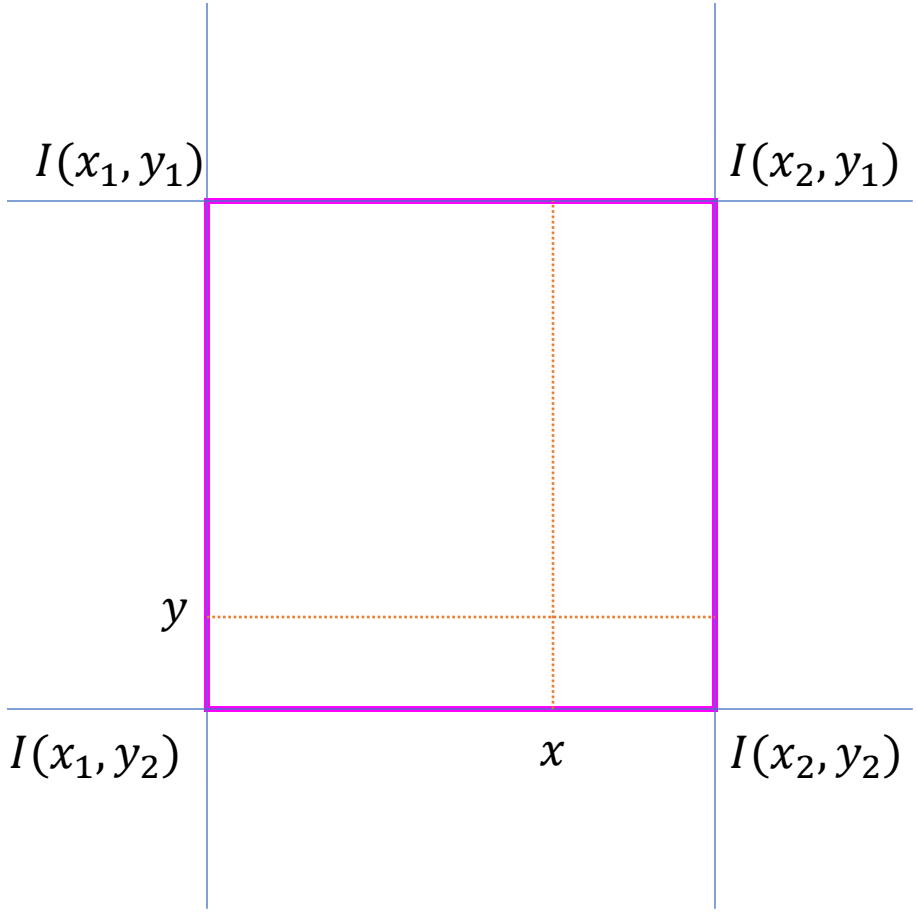


Bi-linear interpolation





Bi-Linear interpolation



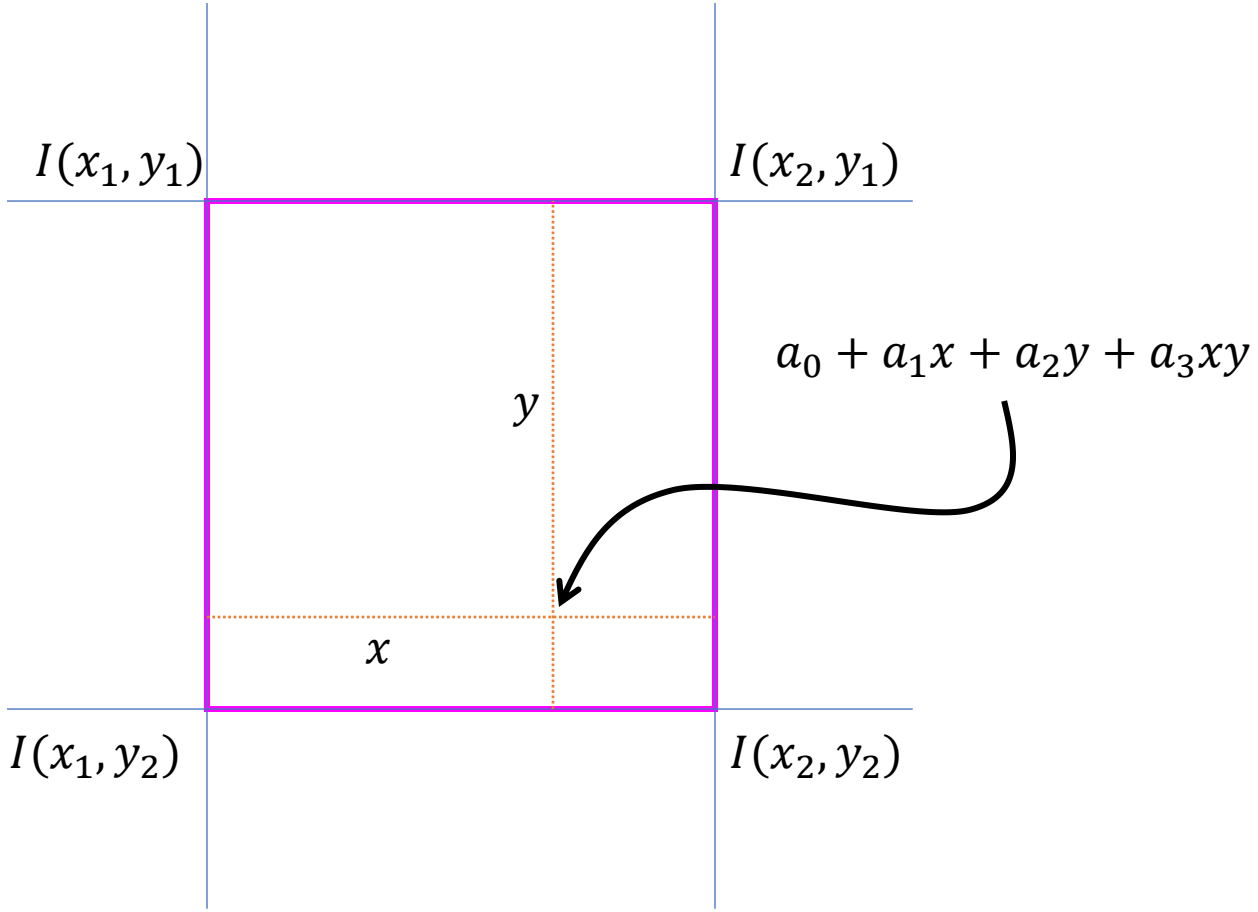
Multi-linear polynomial

$$f(x, y) = a_0 + a_1 x + a_2 y + a_3 xy$$

Then for $i, j \in [1,2]$

$$I(x_i, y_i) = a_0 + a_1 x_i + a_2 y_j + a_3 x_i\, y_j$$

Bi-Linear interpolation



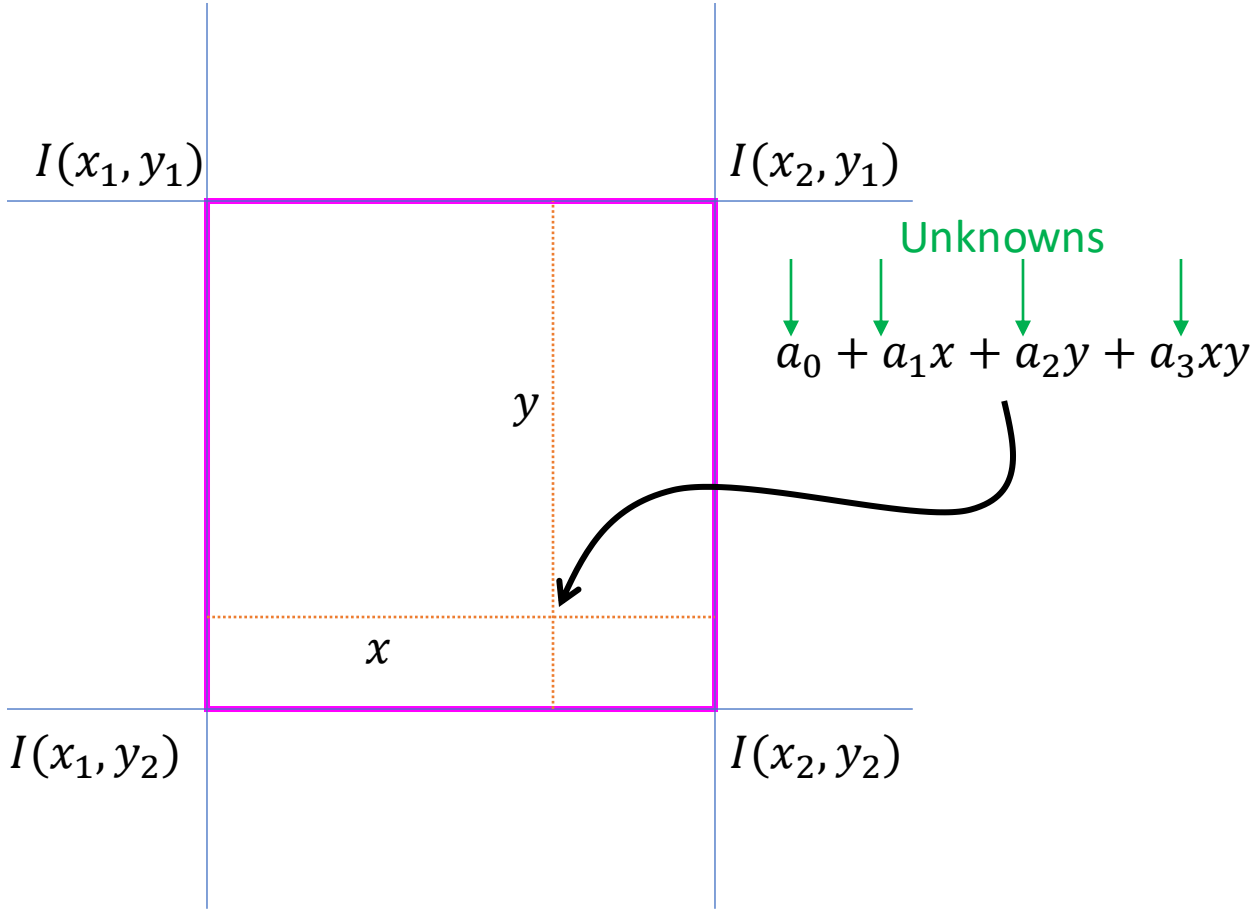
Multi-linear polynomial

$$f(x, y) = a_0 + a_1x + a_2y + a_3xy$$

Then for $i, j \in [1,2]$

$$I(x_i, y_i) = a_0 + a_1x_i + a_2y_j + a_3x_i\, y_j$$

Bi-Linear interpolation



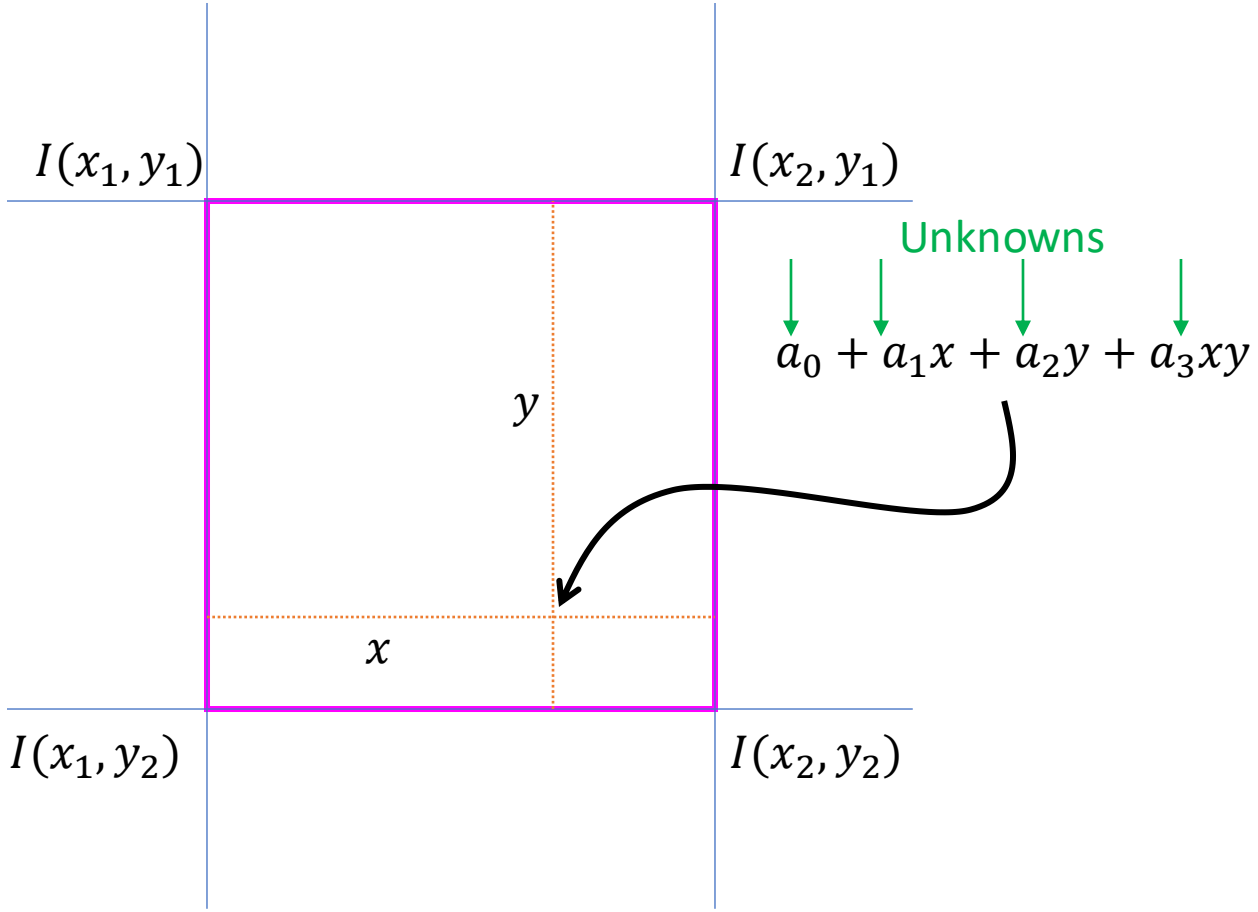
Multi-linear polynomial

$$f(x, y) = a_0 + a_1x + a_2y + a_3xy$$

Then for $i, j \in [1,2]$

$$I(x_i, y_i) = a_0 + a_1x_i + a_2y_j + a_3x_i\, y_j$$

Bi-Linear interpolation



Multi-linear polynomial

$$f(x, y) = a_0 + a_1x + a_2y + a_3xy$$

Then for $i, j \in [1,2]$

$$I(x_i, y_i) = a_0 + a_1x_i + a_2y_j + a_3x_i\, y_j$$

Solve for the unknowns using the following system of equations

$$\begin{bmatrix} 1 & x_1 & y_1 & x_1y_1 \\ 1 & x_2 & y_1 & x_2y_1 \\ 1 & x_1 & y_2 & x_1y_2 \\ 1 & x_2 & y_2 & x_2y_2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} I(x_1, y_1) \\ I(x_2, y_1) \\ I(x_1, y_2) \\ I(x_2, y_2) \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 1 & x_1 & y_1 & x_1y_1 \\ 1 & x_2 & y_1 & x_2y_1 \\ 1 & x_1 & y_2 & x_1y_2 \\ 1 & x_2 & y_2 & x_2y_2 \end{bmatrix}^{-1} \begin{bmatrix} I(x_1, y_1) \\ I(x_2, y_1) \\ I(x_1, y_2) \\ I(x_2, y_2) \end{bmatrix}$$

Bilinear interpolation: Pros

- Smoothing Effect, which helps reduce jagged edges and pixelation.
- Simple to Implement, requires fewer calculation and computational inexpensive as compared to other methods
- Maintains linearity between the known data points, which can be desirable in certain applications, such as computer graphics.

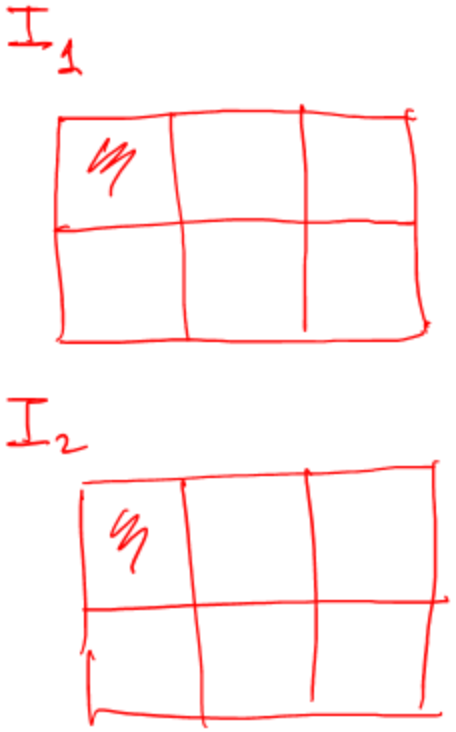
Bilinear interpolation: Cons

- Loss of sharpness and fine details
- Color artifacts
- No consideration for high-frequency components
  - Not suitable for images with intricate patterns or textures
- Not ideal for large scaling
- Limited accuracy and it may not be suitable for photometric applications

Comparing two images

- Mean Squared Error

$$\mathrm{MSE} = \frac{1}{HW}\sum_{x=1}^{W}\sum_{y=1}^{H}\left(I_1(x,y) - I_2(x,y)\right)^2$$



Comparing two images

- Peak Signal-to-Noise Ratio

$$\mathrm{PSNR} = 10 \log_{10} \frac{\mathrm{MAX}^2}{\mathrm{MSE}}$$

- MAX refers to the maximum value of a pixel, so for an 8-bit image, $\mathrm{MAX} = 2^8 = 255$
- Units are in decibels (db)
- MSE is averaged squared error, therefore, signal power needs to expressed in square as well. Generally, power is the square of amplitude.

Peak Signal to Noise Ratio($\mathrm{PSNR} = 10 \log_{10} \frac{\mathrm{MAX}^2}{\mathrm{MSE}}$)

signal / noise

- Compression of range
  - Signal-to-noise ratios can span orders of magnitude (from 1 to 10,000+). The log compresses this into a manageable scale (e.g. 0–60 dB).
- Additivity
  - Using log turns multiplicative ratios into additive values (e.g. a $10 \times$ improvement = $+10$ dB).
- Perception
  - Human senses (hearing, vision, brightness perception) roughly follow a logarithmic response — so the log scale correlates better with perceived differences.

The log in PSNR makes huge ratios easier to interpret, aligns with human perception, and is consistent with the dB convention for measuring power ratios.

How good is your interpolation algorithm?

- Start with a high-quality, high-resolution reference image: I_{HR}
- Construct a truth pair by down sampling I_{HR} to construct I_{LR} using a gold-standard antialias (e.g., sinc/Lanczos with strong low-pass)
  - A good starting point is to perform Gaussian blur and then down-sample

Case 1: Down sampling

- Use your algorithm to downsample I_{HR} and compare the result with I_{LR} using MSE and PSNR

Case 2: Up sampling

- Use your algorithm to upsample I_{LR} and compare the result with I_{HR} using MSE and PSNR

This technique will allow you to compare your interpolation method against each other. This is sometimes referred to as reference-based protocol.

Other considerations

- **Downscaling specifics (antialias matters most)**
  - A good downscale = low-pass then sample.
  - Check spectra: energy above target Nyquist should be minimal.
  - Compare kernels (bicubic/Lanczos/B-spline) for stopband attenuation vs. ringing tradeoffs.

- **Upscaling specifics**
  - Evaluate edge preservation (thin lines, diagonals), texture fidelity, and halo control.
  - If you prefer “crisp”, quantify it: combine gradient magnitude stats with a ringing penalty near edges.

- **Efficiency & robustness**
  - Runtime & memory (per megapixel) on CPU/GPU; batch behavior.
  - Stability across scale factors (e.g., $0.5\times, 0.7\times, 1.5\times, 3\times$) and aspect-ratio changes.
  - Determinism (same bits in = same output) when you need reproducibility.

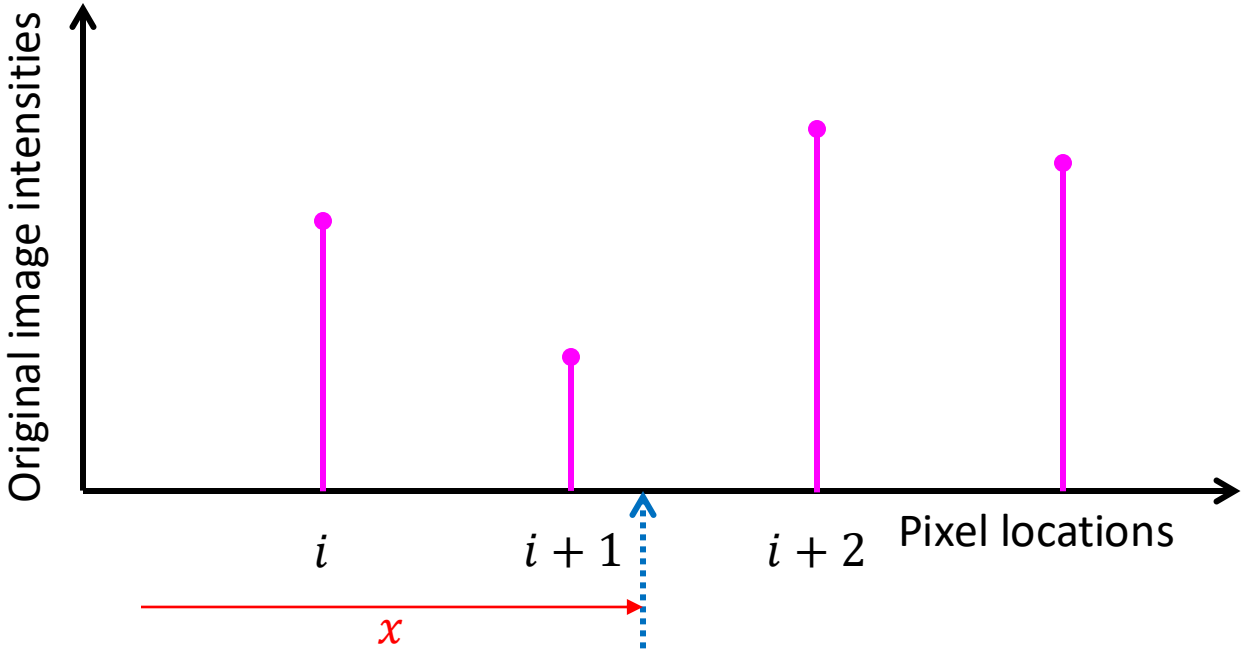
Image quality

- Fidelity
  - Looks like the original at the intended scale
- Sharpness vs. ringing
  - Edges crisp but no halos
- Alias suppression
  - No moiré/zippering on patterns
- Color faithfulness
  - No hue shifts; correct gamma handling
- Temporal consistency
  - If resizing video or bursts

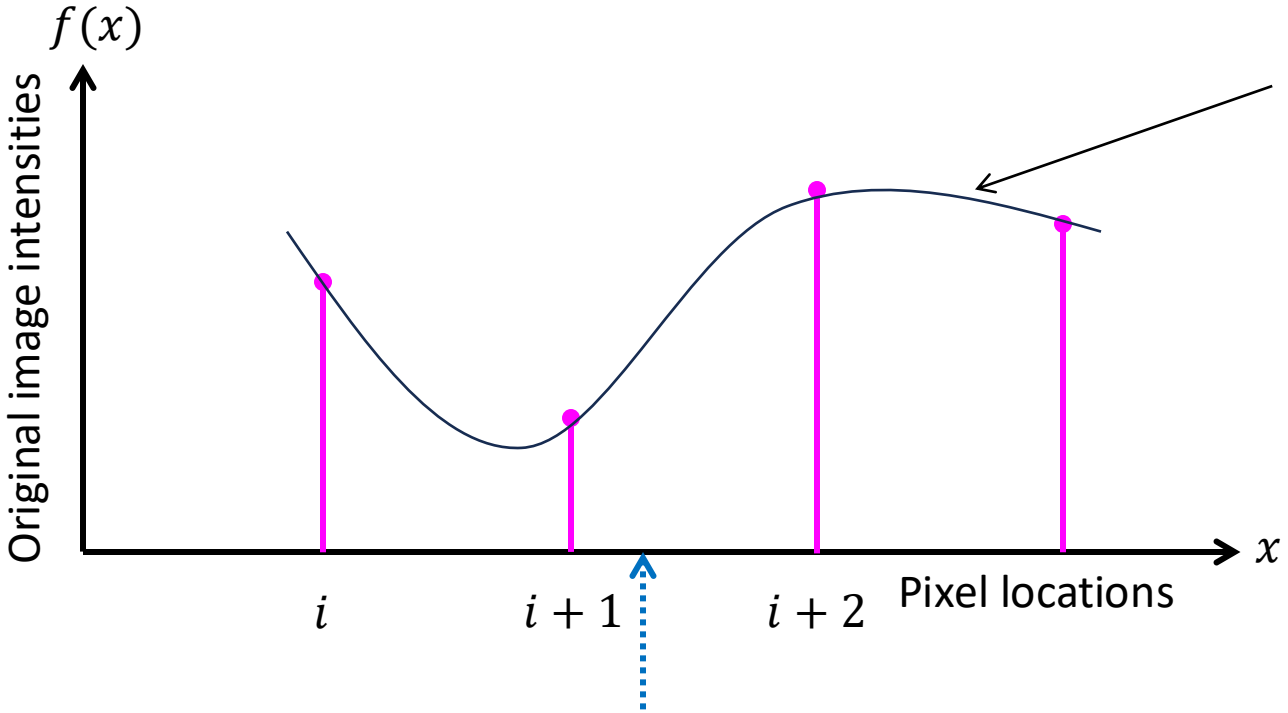
Bicubic interpolation

- Bicubic interpolation is a method for image resizing that calculates new pixel values using the nearest 16 pixels (a 4x4 grid).
- It produces smoother and higher-quality results compared to simpler methods like nearest-neighbor and bilinear interpolation.

Cubic Interpolation (in 1D)



Cubic Interpolation (in 1D)

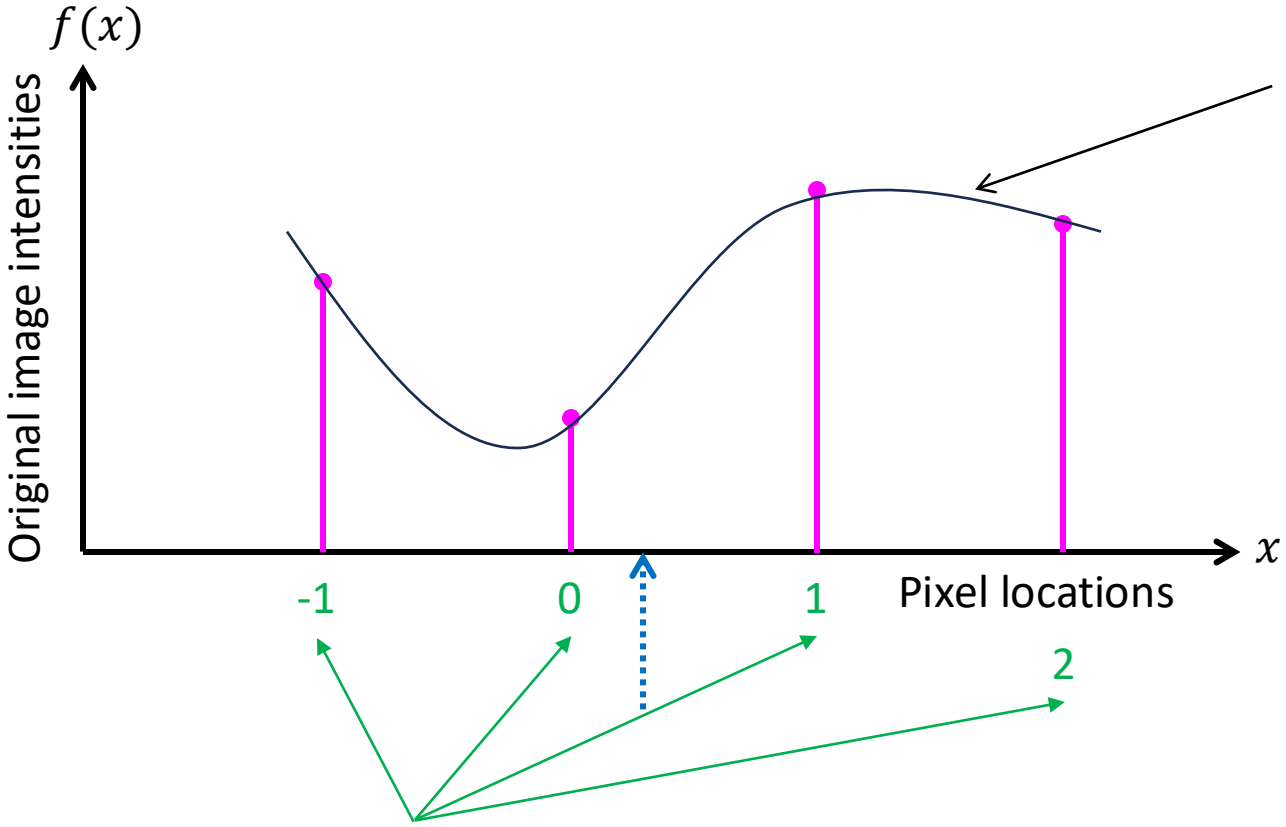


Approximate local structure using a cubic polynomial

$$f(x) = ax^3 + bx^2 + cx + d$$

This equation has four unknowns, so we need at least four points to fit this model (to the available image intensities)

Cubic Interpolation (in 1D)



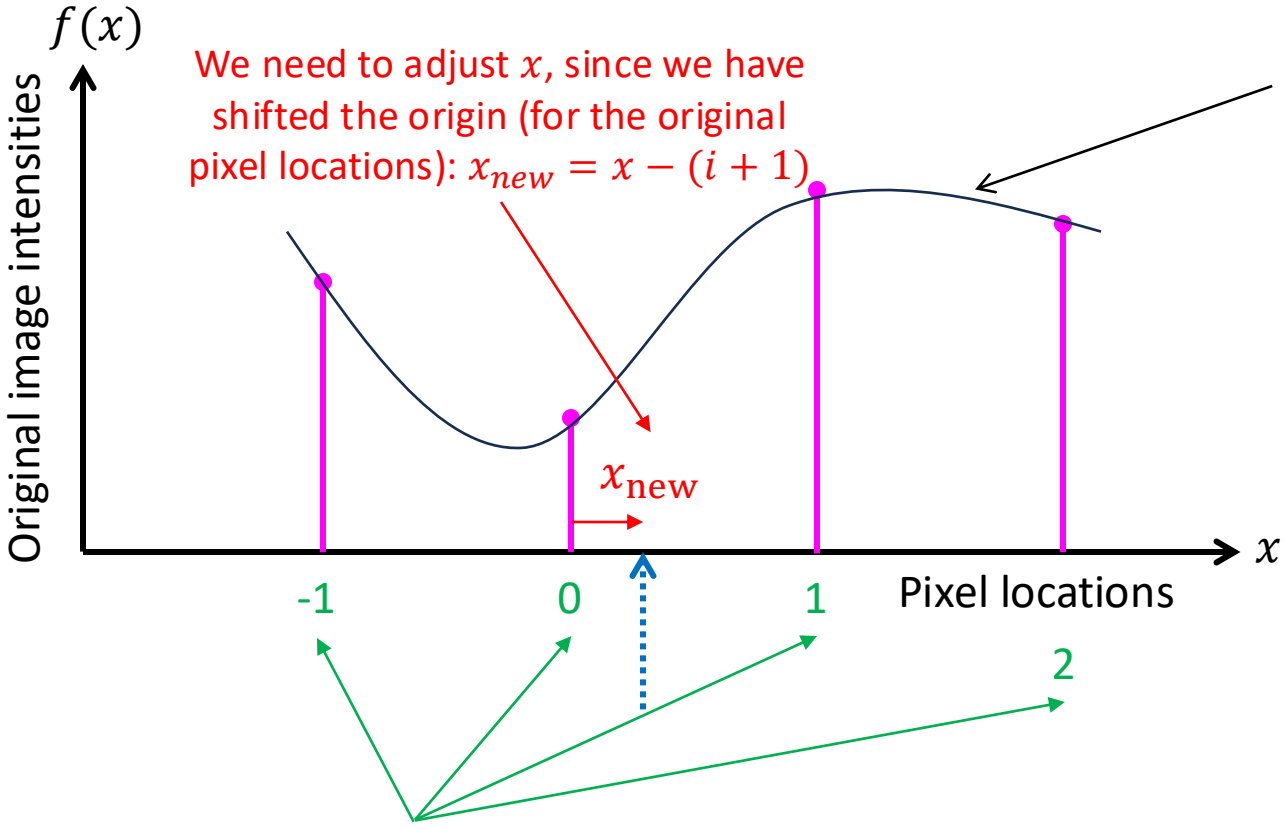
Approximate local structure using a cubic polynomial

$$f(x) = ax^3 + bx^2 + cx + d$$

This equation has four unknowns, so we need at least four points to fit this model (to the available image intensities)

To keep things simple, we re-index pixel locations from -1 to +2, such that the interpolation location lies between 0 and 1

Cubic Interpolation (in 1D)



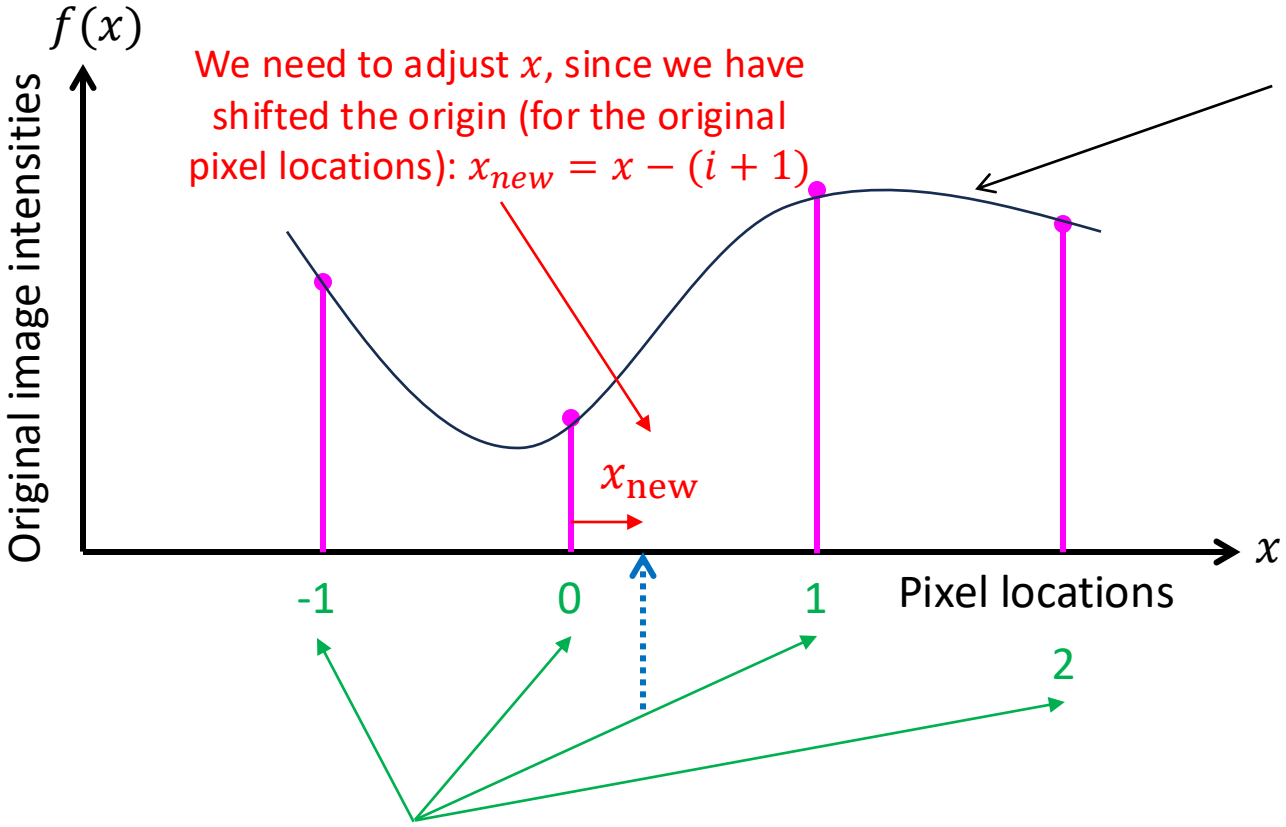
To keep things simple, we re-index pixel locations from -1 to +2, such that the interpolation location lies between 0 and 1

Approximate local structure using a cubic polynomial

$$f(x) = ax^3 + bx^2 + cx + d$$

This equation has four unknowns, so we need at least four points to fit this model (to the available image intensities)

Cubic Interpolation (in 1D)



To keep things simple, we re-index pixel locations from -1 to +2, such that the interpolation location lies between 0 and 1

Approximate local structure using a cubic polynomial

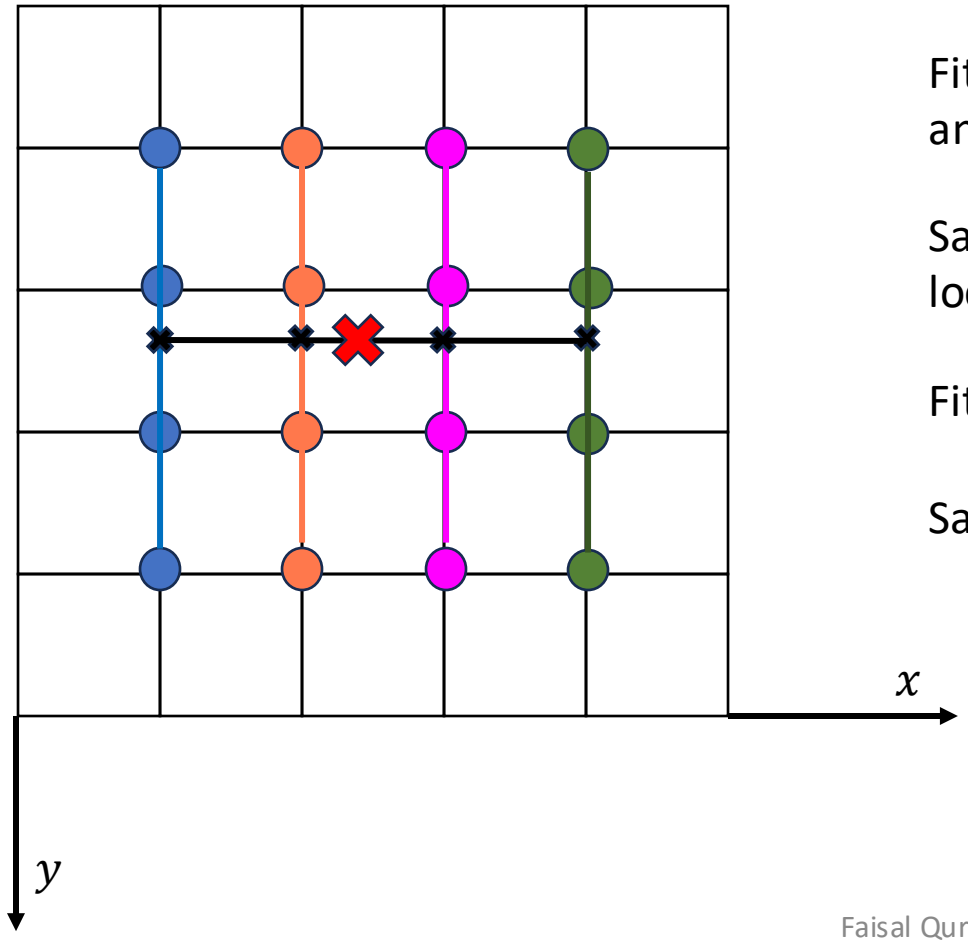
$$f(x) = ax^3 + bx^2 + cx + d$$

Solve for a, b, c, and d using

$$f(-1) = -a + b - c + d$$
$$f(0) = d$$
$$f(1) = a + b + c + d$$
$$f(2) = 8a + 4b + 2c + d$$

Evaluate at the fitted cubic polynomial at x_{new}

Bicubic Interpolation



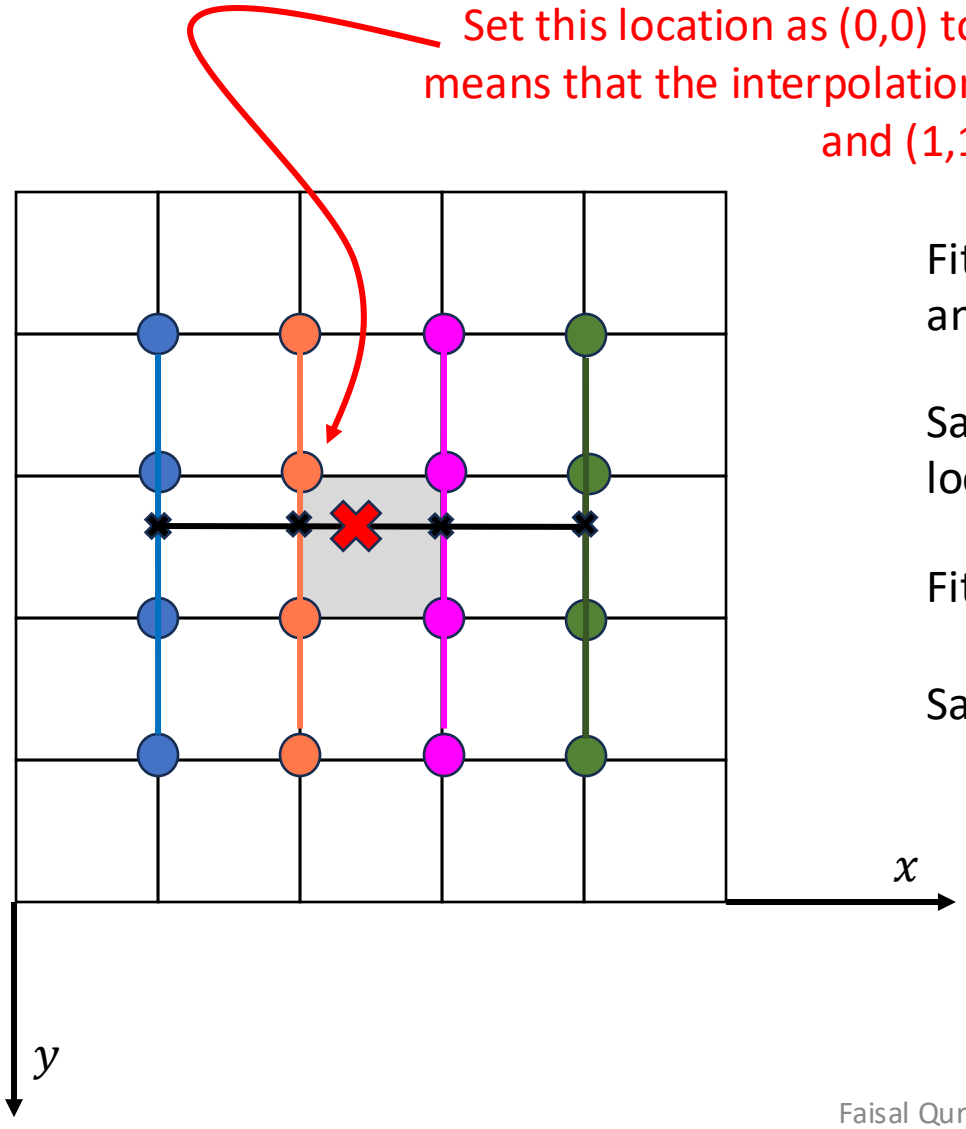
Fit 4 cubic polynomials along y using the blue, orange, pink and green values, respectively

Sample cubic polynomials using y value of the desired location

Fit a cubic polynomial (black) using the 4 sampled values

Sample this cubic polynomial using the desired x location

Bicubic Interpolation



Fit 4 cubic polynomials along y using the blue, orange, pink and green values, respectively

Sample cubic polynomials using y value of the desired location

Fit a cubic polynomial (black) using the 4 sampled values

Sample this cubic polynomial using the desired x location

Bicubic interpolation: Pros

- Better Image Quality:
  - Reduces jagged edges and pixelation, handling edges and gradients effectively.
- Improved Detail Preservation:
  - Retains fine details, ideal for upscaling images.
- Smooth Transitions:
  - Minimizes artifacts like sudden intensity changes, providing a natural look.
- Widely Used:
  - Implemented in many image processing tools, making it a well-established method.

Bicubic interpolation: Cons

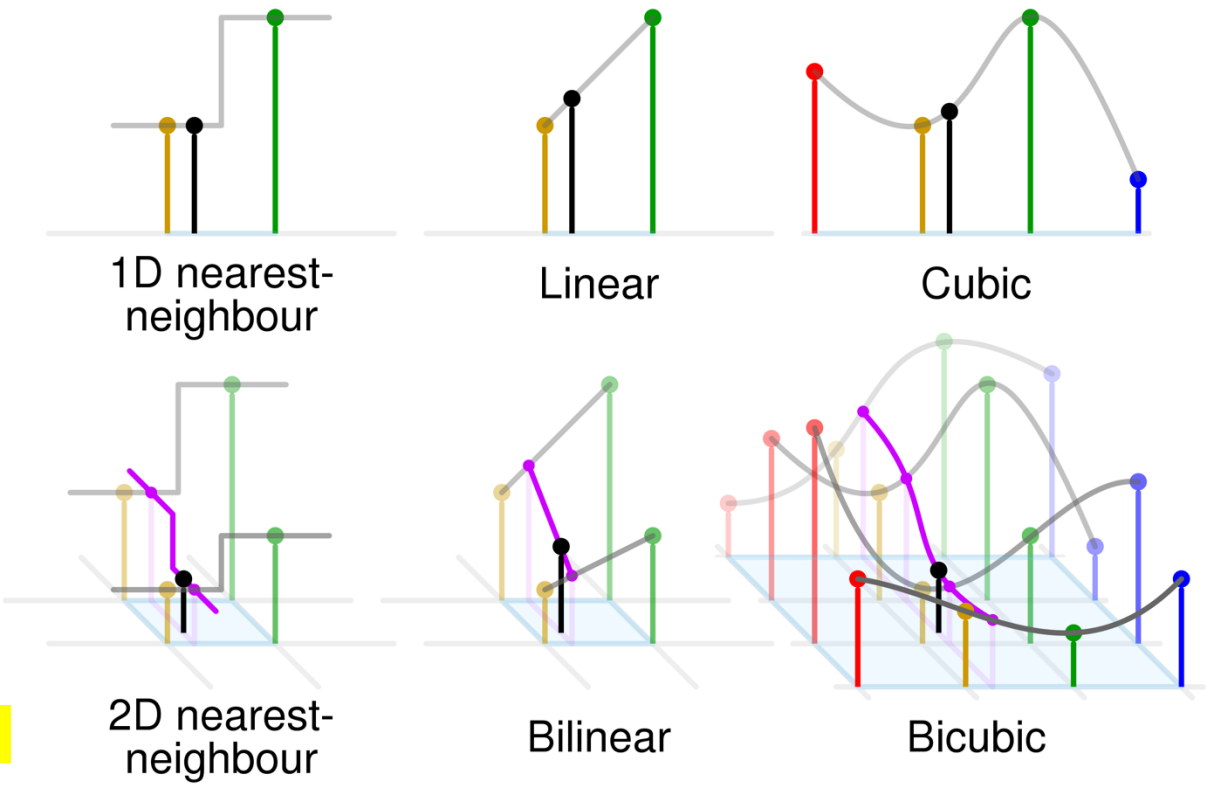
- Slower Performance:
  - More computationally expensive than simpler methods, making it slower on large images.
- Blurring:
  - Can introduce blurriness, especially when scaling down.
- Halo Artifacts:
  - Sometimes causes halo effects around edges in high-contrast areas.
- Over-Smoothing:
  - May smooth out fine details too much during upscaling, leading to a soft image.

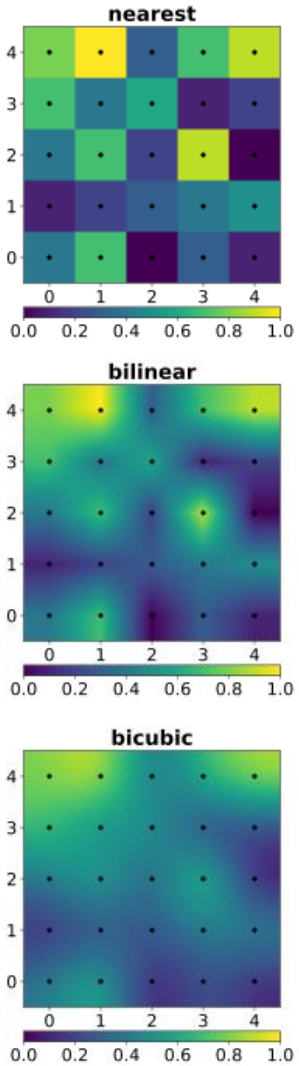
Bicubic interpolation: Best use cases

- Moderate upscaling where image sharpness is not the highest priority but smoothness is.
- General-purpose resizing for photographs and images with a balance of speed and quality.

Summary

- Image interpolation methods
- Nearest neighbor interpolation
- Bilinear interpolation





Black dot denotes the sampled pixel value

(CMG Le. Wikipedia)

On image interpolation

https://www.menti.com/bltyg9abucso