CV 2026 – Week 1 – HC1a

# Images
and
# Interpolation

Dimitris Tzionas

d.tzionas@uva.nl, CC: e.a.veltmeijer@uva.n
(please CC your TA)

# Images

## How do we represent images?

https://rvdboomgaard.github.io/ComputerVision_LectureNotes/LectureNotes/IP/Images/index.html

Functions

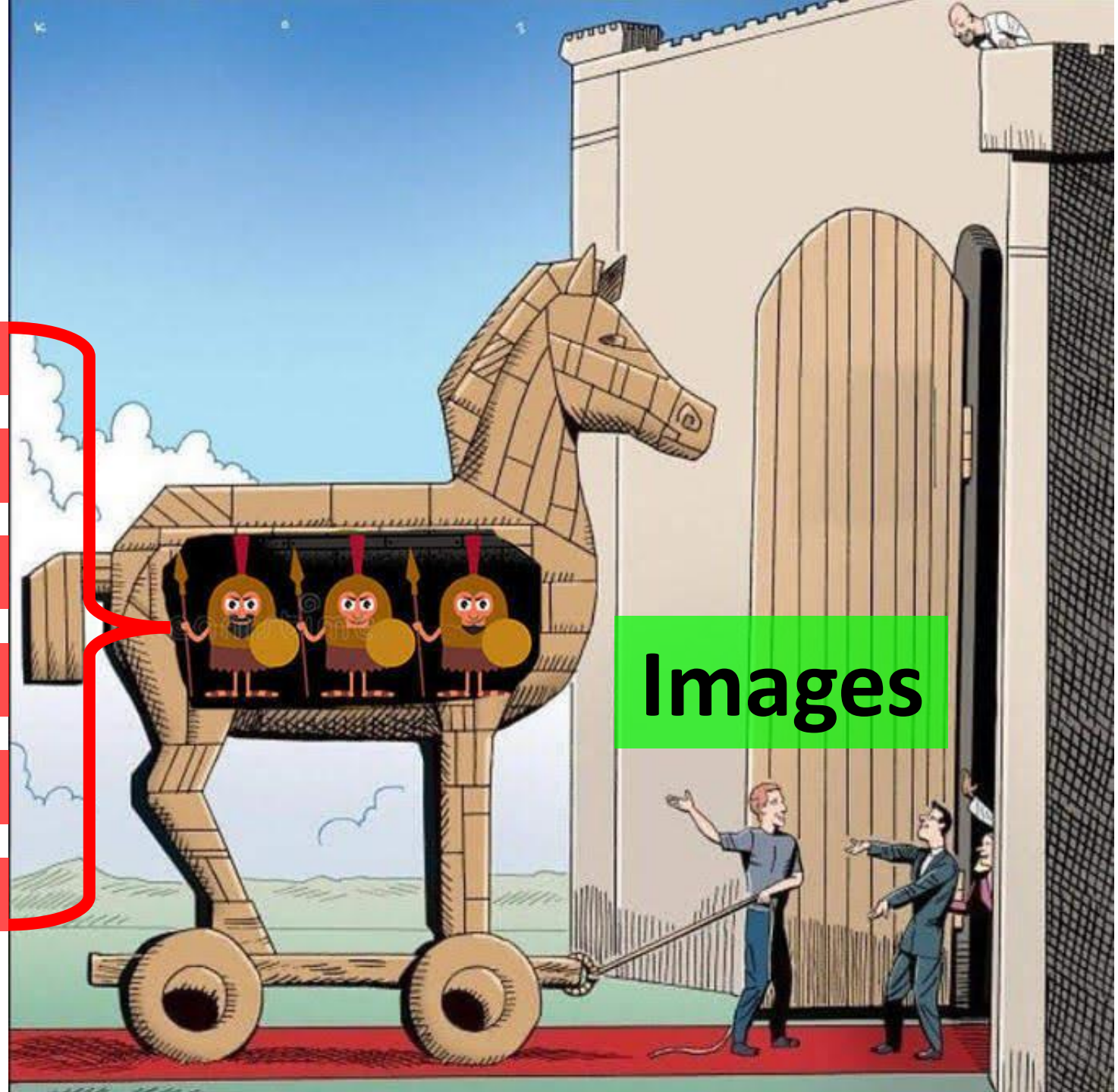
Cont./Disc. Space

Sampling Grid

Quantization

Interpolation

Histograms



# Images

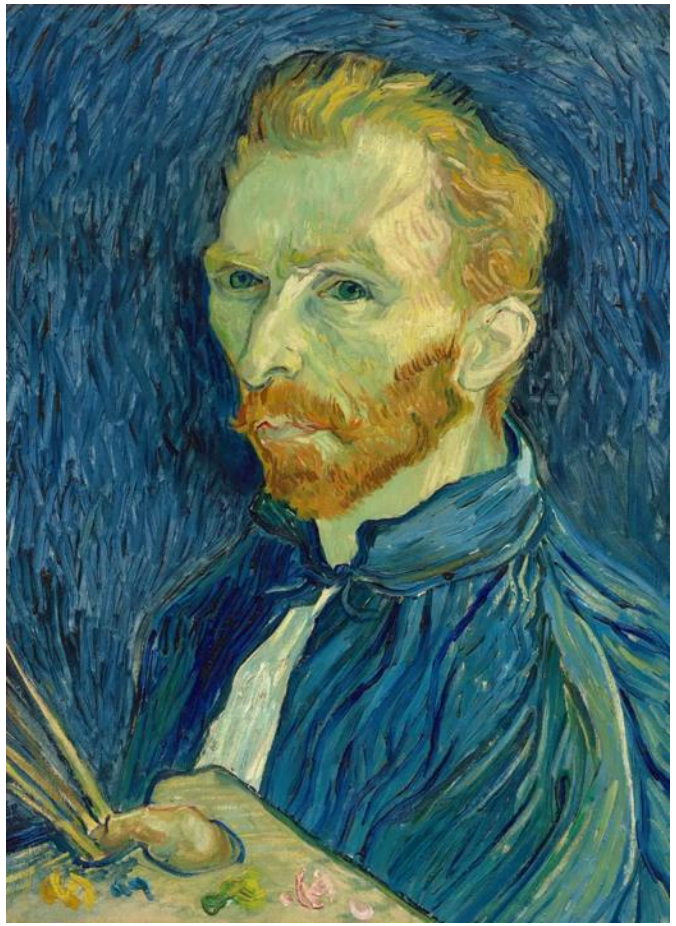
## Standard ‘concept’

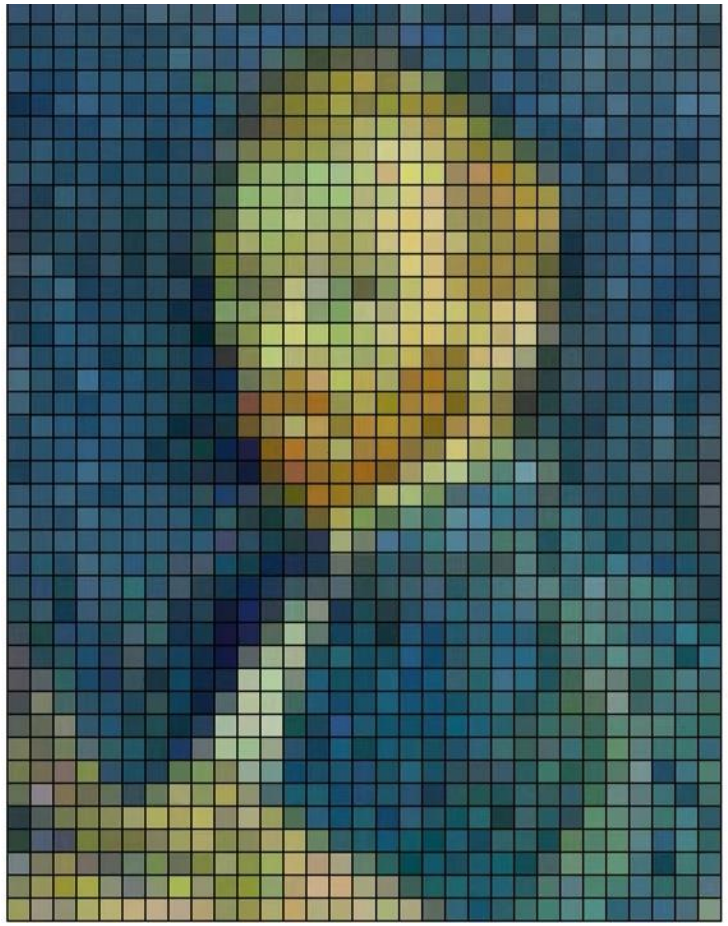
***Pixels***

‘Atoms’ of an image

**Pict**ure **el**ement**s**

Each pixel → ***Color triplet***





*e.g. 800 x 600 pixels*

# Images

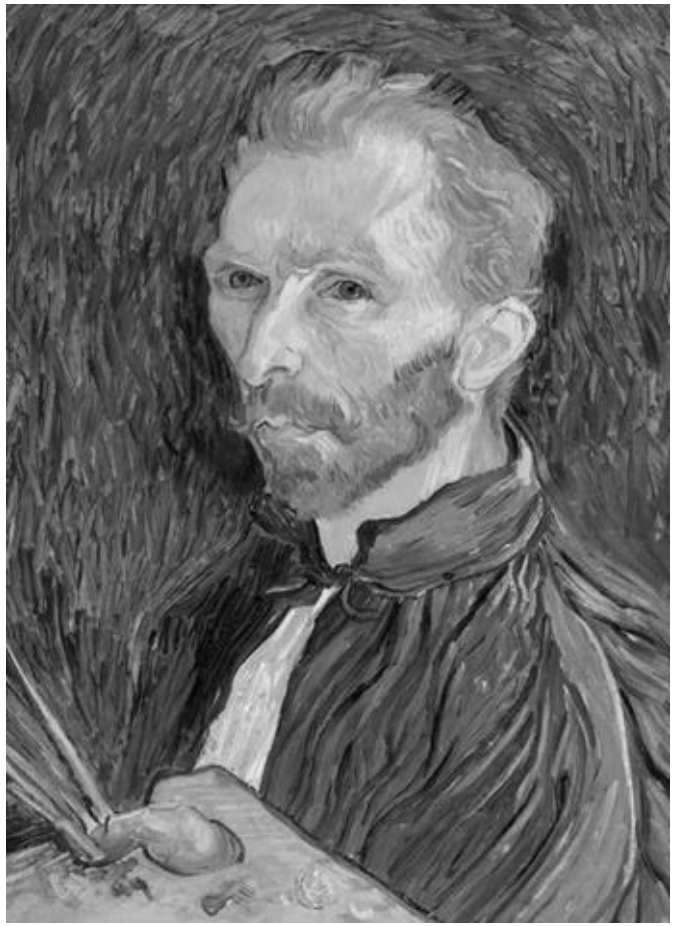
## Standard ‘concept’

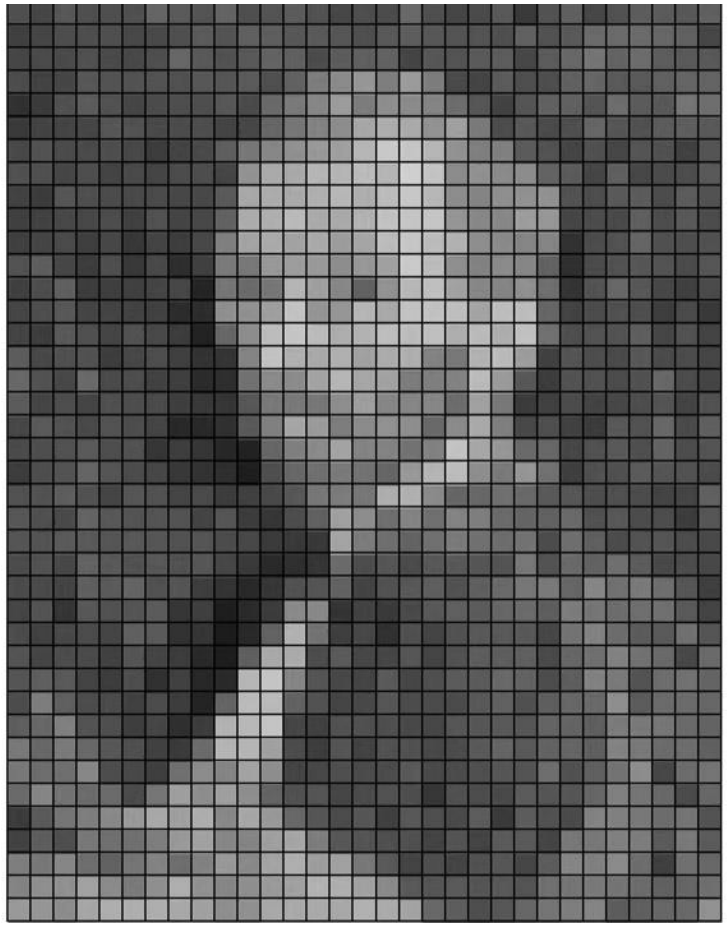
***Pixels***

‘Atoms’ of an image

**Pict**ure **el**ement**s**

Each pixel → ***Gray ‘luminance’* value**





*e.g. 800 x 600 pixels*

# Images

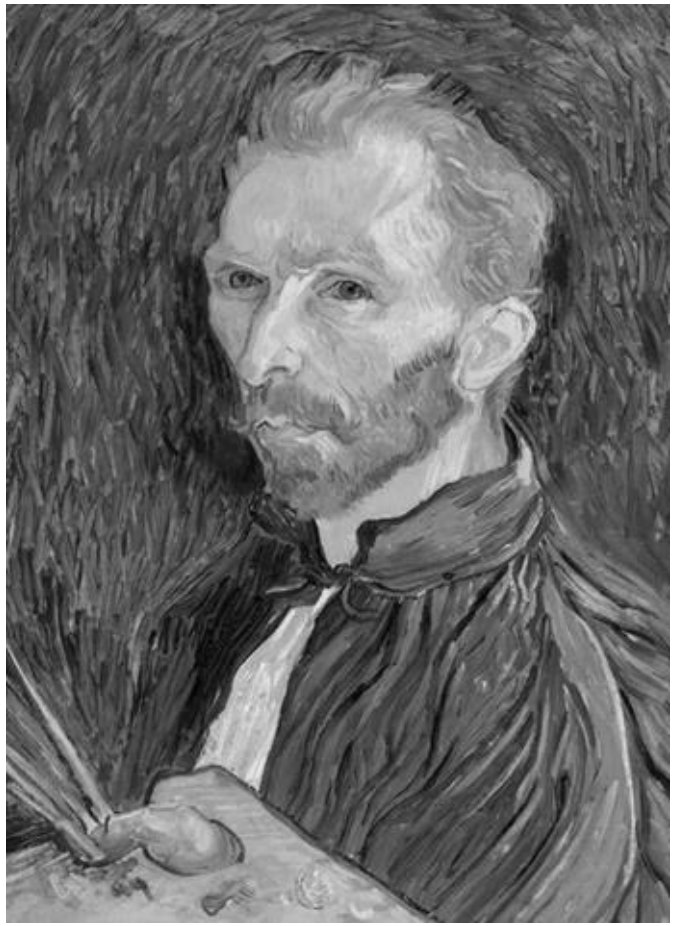
## Standard ‘concept’

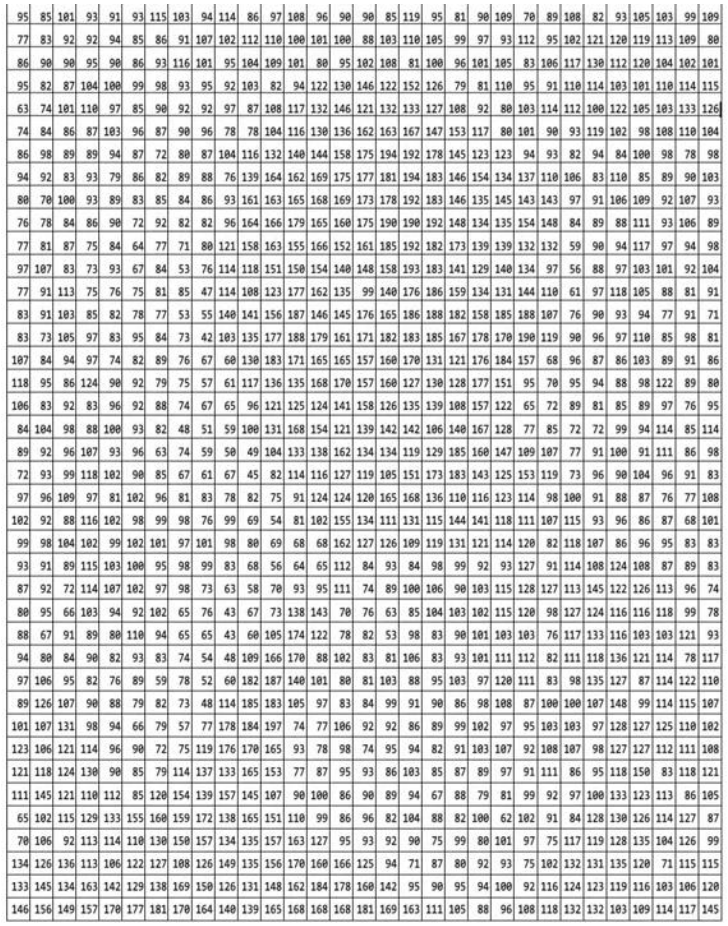
***Pixels***

‘Atoms’ of an image

**Picture elements**

Each pixel → ***Gray ‘luminance’* value**





*e.g. 800 x 600 pixels*

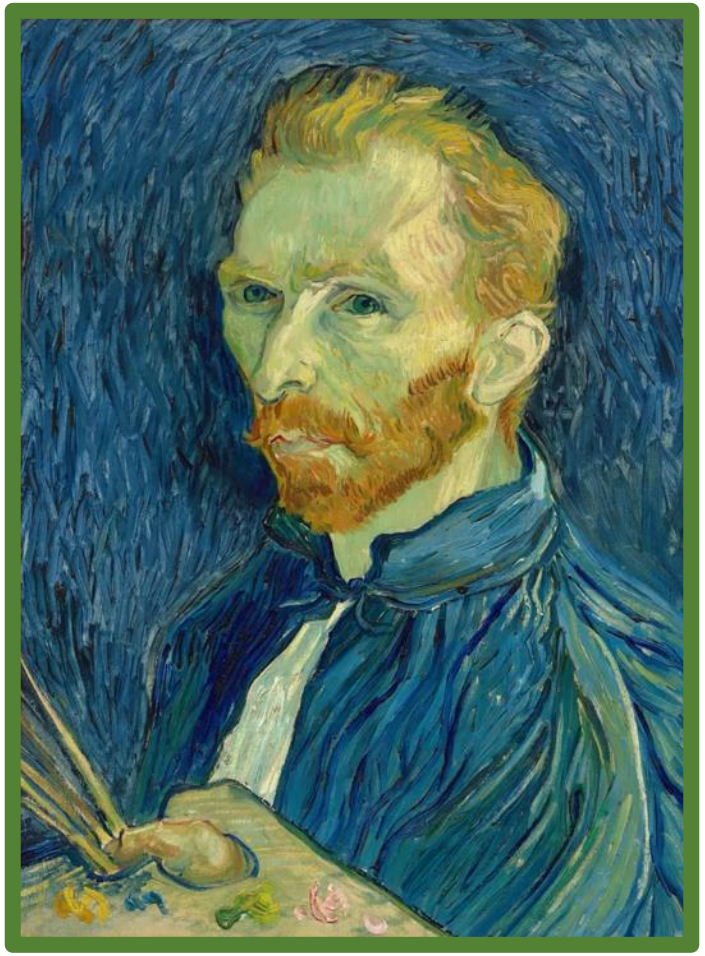
# Images

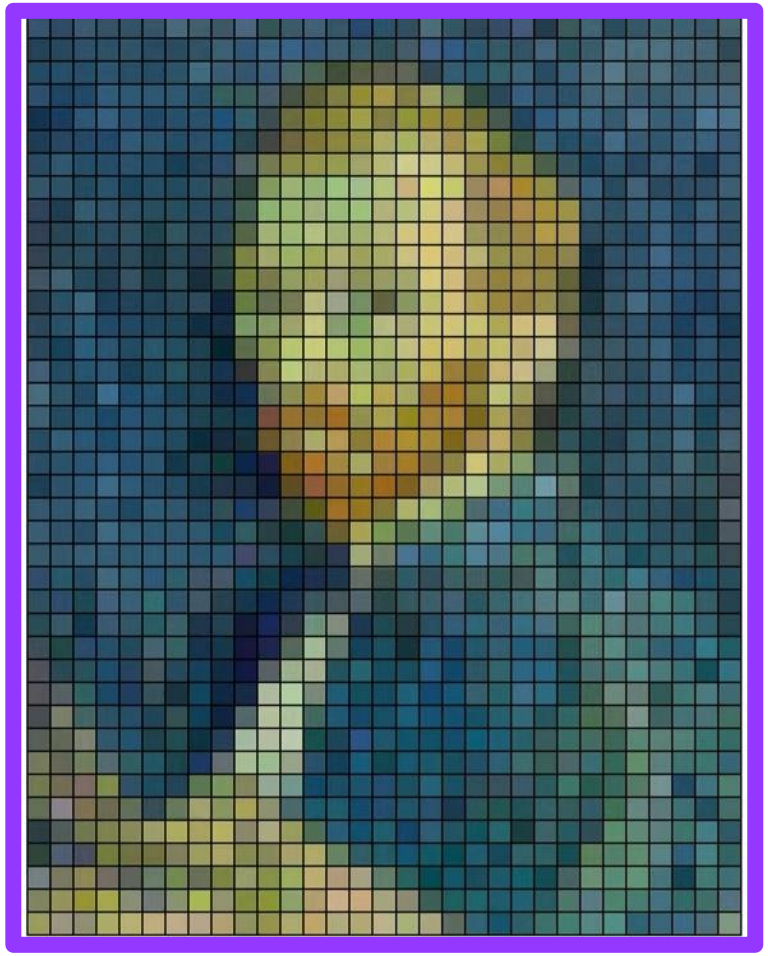
## Standard ‘concept’

How can it be that:

A *natural-looking image*

Turns into an
*artificial-looking ‘puzzle’*?





*e.g. 800 x 600 pixels*

# Images

## Measuring Light

Physical observation == Light hits objects & bounces to the observer (eye / camera)

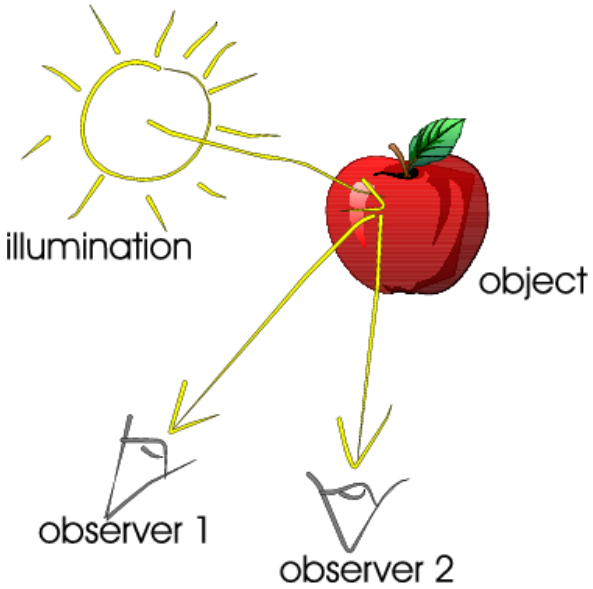
Mathematical model

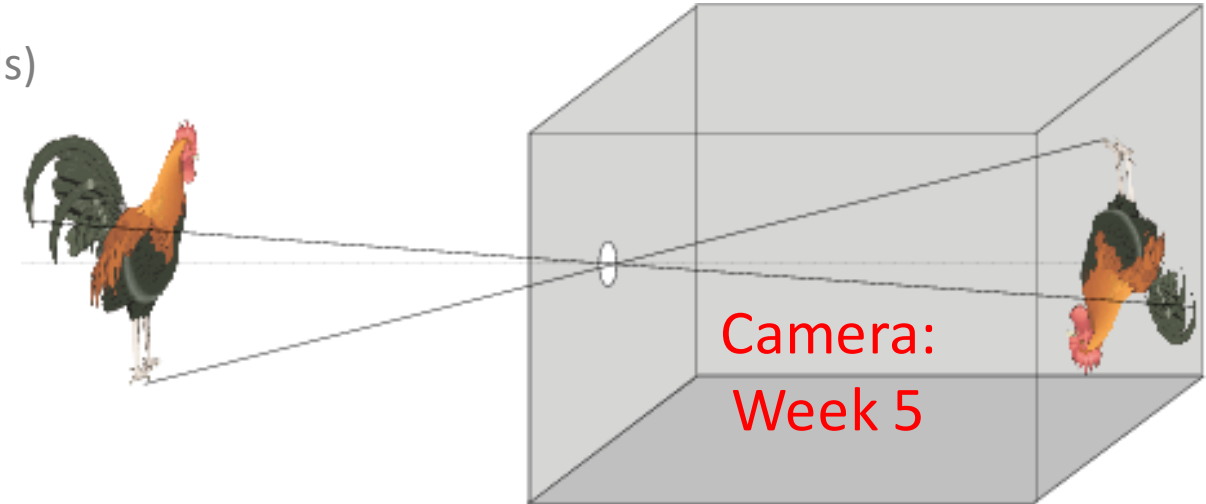
Image

Measure
electromagnetic energy
@ every point of

- eye's retina
  (light-sensitive inner-eye cells)
- camera sensor

Function over a
*spatial domain*
(position on sensor)



# Images

## Measuring Light

Physical observation

Mathematical model

Image

==

Light hits objects & bounces to the observer (eye / camera)

Measure electromagnetic energy @ every point of

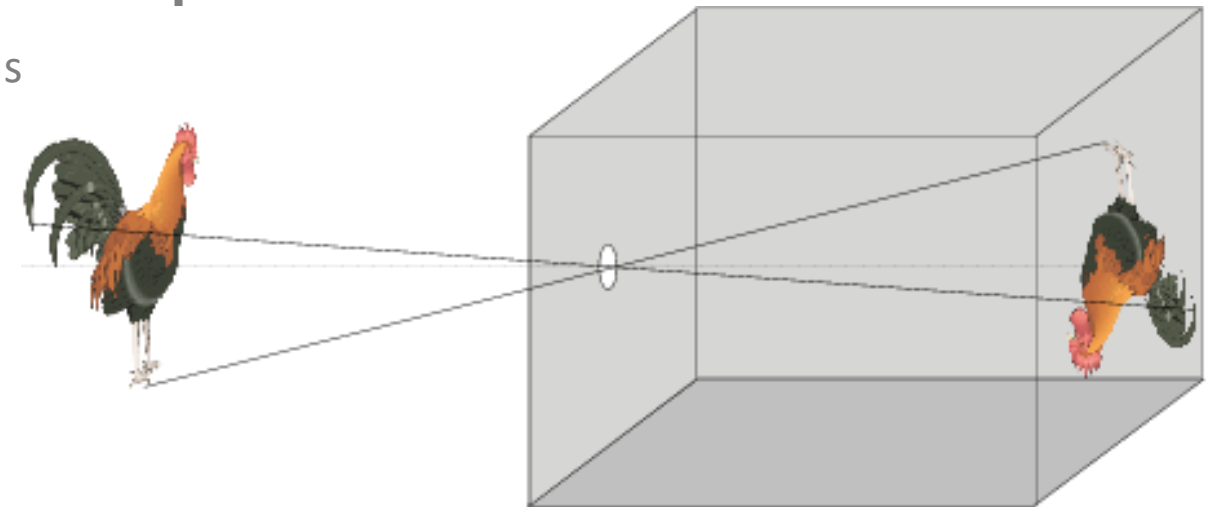
- eye's retina (light-sensitive inner-eye cells
- camera sensor

Function over a *spatial domain* (position on sensor)

Sensor as 2D plane $E \in \mathbb{R}^2$

Points on 2D plane: $x \in E$

Image → $f(x)$



# Images

## Measuring Light

Physical observation

Mathematical model

Image

==

Light hits objects & bounces to the observer (eye / camera)

Measure electromagnetic energy @ every point of

- eye's retina (light-sensitive inner-eye cells)
- camera sensor

Function over a *spatial domain* (position on sensor)

Sensor as 2D plane $E \in \mathbb{R}^2$

Points on 2D plane: $x \in E$

Image → $f(x)$

Value $f$ proportional to measured energy at position $x$

Defined over a continuous spatial domain

# Images

## Measuring Light

How to measure light over a continuous domain?

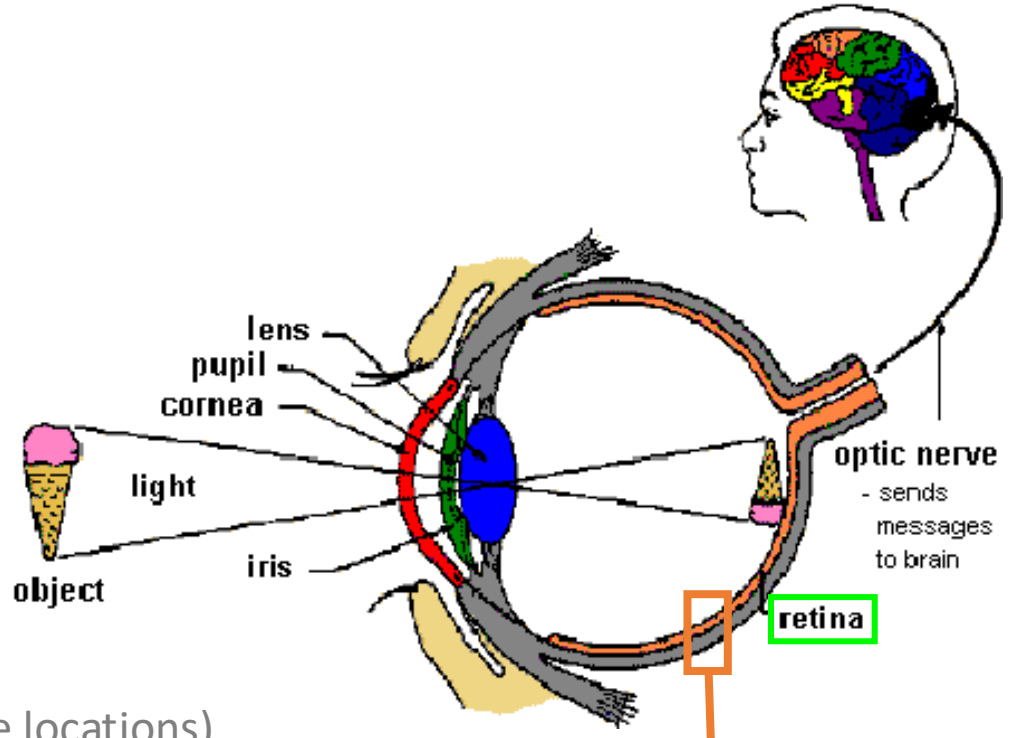
In theory → Infinite sample points $x \in E$

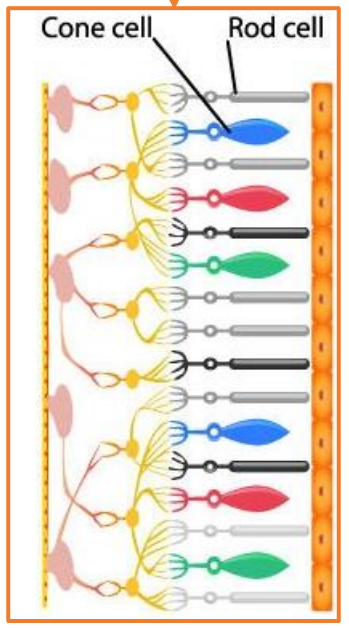
In practice → Sampling probes of ***finite***: ***Number*** (sample locations)

***Size*** (sampling area)

***Temporal sensing***

***Integrate*** sensed energy → over ***finite area*** & for ***finite time***



# Images

## Measuring Light

How to measure light over a continuous domain?

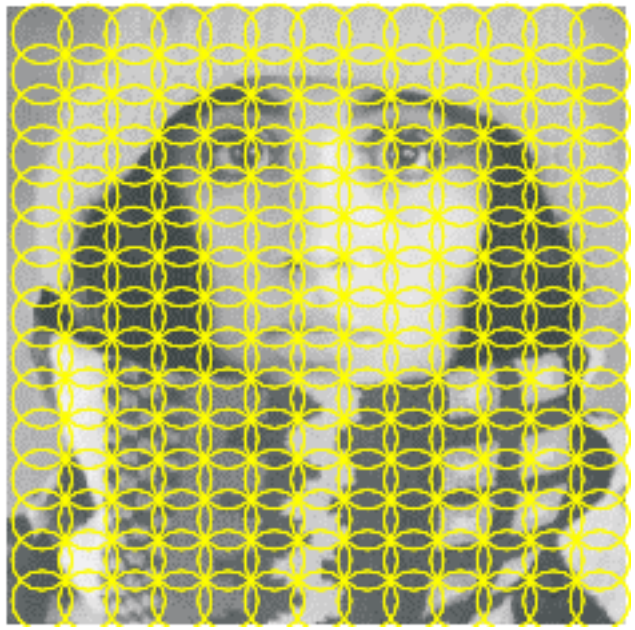
In theory → Infinite sample points $x \in E$

Compute 'average' within each 'circle' →

In practice → Sampling probes of ***finite***: ***Number*** (sample locations)

***Size*** (sampling area)

***Temporal sensing***



***Integrate*** sensed energy → over ***finite area*** & for ***finite time***

Choose sample … { *-distance* / *-area size*

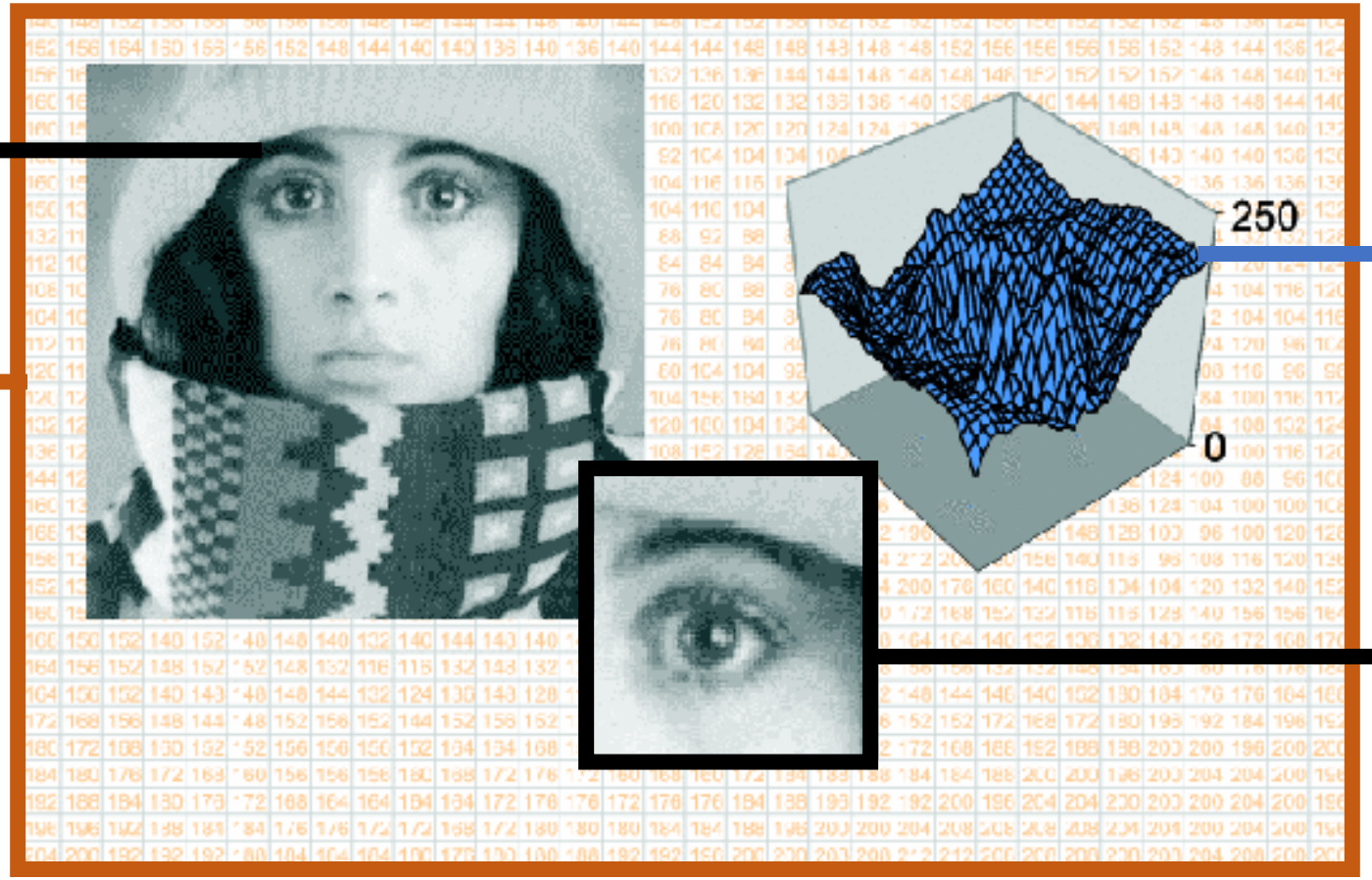
?? Pros / cons →

# Images

## Continuous Functions

What ***humans*** perceive as image

What ***computers*** perceive as image
*(discrete representation)*




The underlying ***mathematical model***

**2D function**
(height == luminance)
(continuous)

***same*** info!

**Image crop**

# Images

## Definition

Image $f$ **maps** ***from:*** a spatial domain $E$

***to:*** a range $V$

***from:*** element $x \in E$

***to:*** value $f(x) \in V$

$$f: E \rightarrow V$$

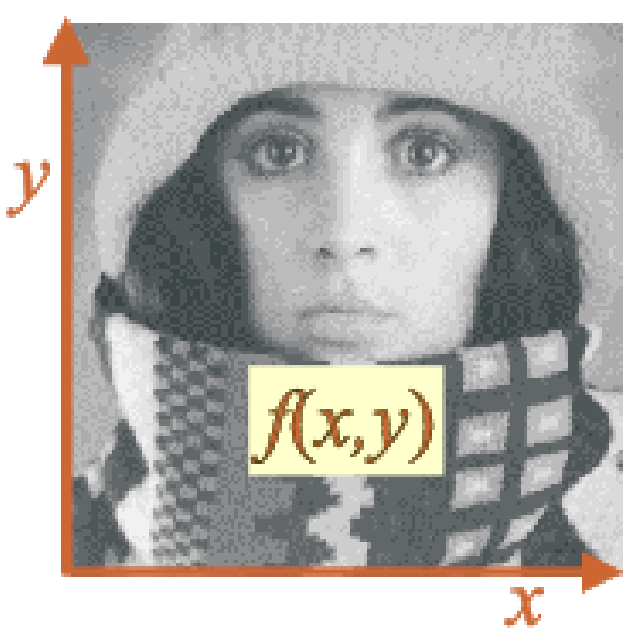
→ Euclidian Space $E \in \mathbb{R}^n$

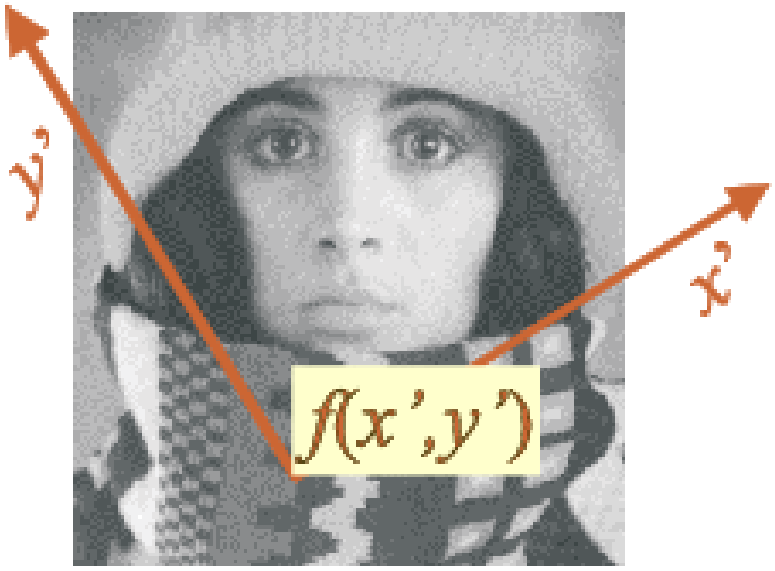
$E \in \mathbb{R}^2$ ✓

$E \in \mathbb{R}^3$

# Images

Definition - Domain $E \in \mathbb{R}^2$




The *basis* in domain space is *arbitrary*

The *coordinate frame* can *change*, but signal stays the *same*

# Images

## Definition - Range $V$

At each element $x \in E$ we measure…

**Grayscale Images**

Luminance
X-ray radiation

$$f(x) \in \mathbb{R}$$

**Color Images**

$$f(x) = \begin{bmatrix} b(x) \\ g(x) \\ r(x) \end{bmatrix} \in \mathbb{R}^3$$

**Indicator Images**

$$f(x) \in \{0,1\}$$

aka: **'Binary Mask'** or **'Binary Segmentation'**

$$f(x) = \begin{cases} 0 \rightarrow \text{background} \\ 1 \rightarrow \text{person} \end{cases}$$

$$f(x) = \begin{cases} 0 \rightarrow \text{background} \\ 1 \rightarrow \text{dress} \\ 2 \rightarrow \text{arm} \\ 3 \rightarrow \text{head} \\ 4 \rightarrow \text{hair} \end{cases}$$

aka: **'Semantic Segmentation'**

# Images

## Discretization

How can we *store* a continuous grayscale image?

$$f: \mathbb{R}^2 \rightarrow \mathbb{R}^1$$

Discretize
- Spatial domain → **Sampling**
- Image range → **Quantization**

# Images

## Discretization – Quantize Range

**Per pixel:** 1 byte / 8 bit / uint8

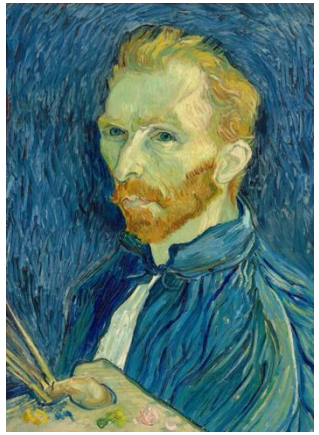
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|

**Range:** 0 … 255

We can store a total of $2^8 = 256$ values

Processing can yield values that **cannot** be encoded:

**Values > 255**
**Values < 0**
→ ***Bit Overflow***

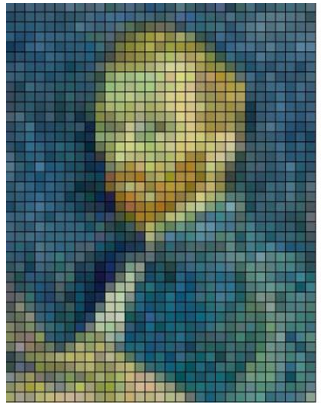


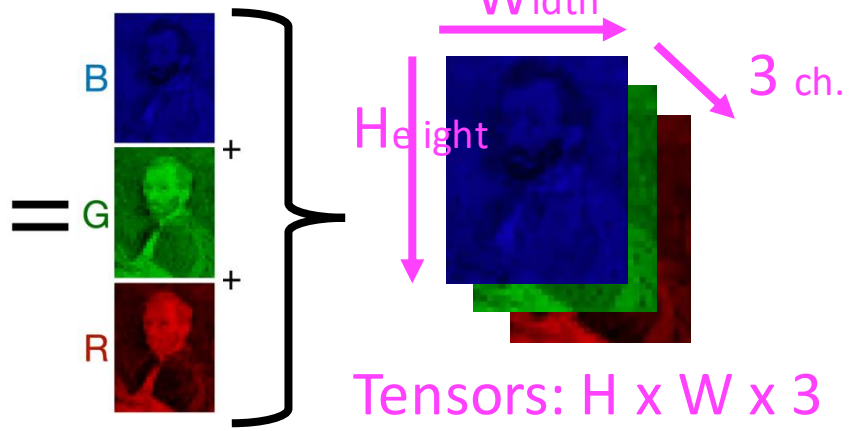
3 byte / 24 bit

| 8 bit Red | 8 bit Green | 8 bit Blue |
|---|---|---|

| 8 bit Blue | 8 bit Green | 8 bit Red |
|---|---|---|

order depends on framework!!!
OpenCV






Tensors: H x W x 3

# Images

## Discretization – Quantize Range

**Per pixel:** 1 byte / 8 bit / uint8

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|

**Range:** 0 … 255

We can store a total of $2^8 = 256$ values

If processing gives values:

- smaller than 0
- bigger than 255

*cannot* be represented → *'burnout'*
*'overexposure'*

For all *processing*
stick to float

Quantize to uint8 *only*
at the end to *visualize*

# Images

## Discretization – Quantize Range


Transform
uint8 → float


or


?

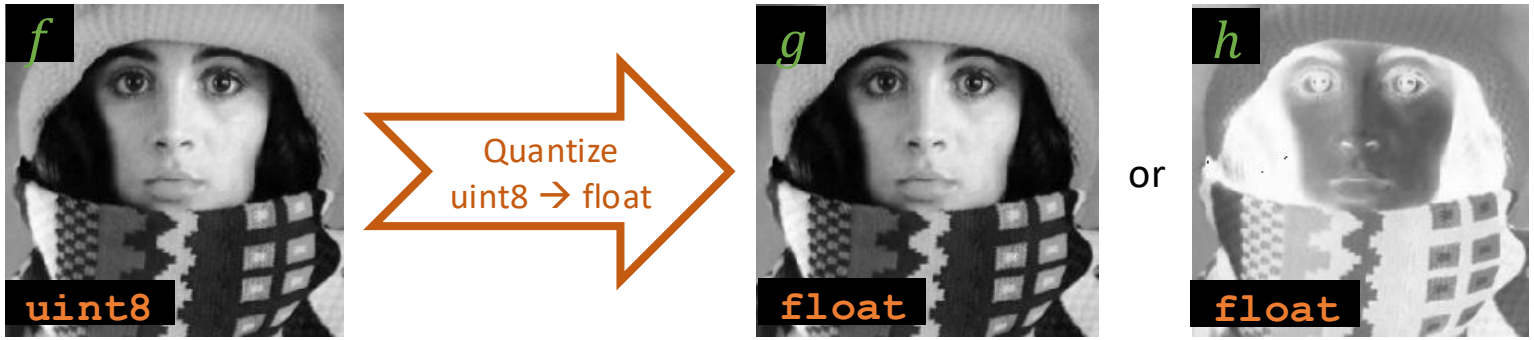
Python Code

```
g =       (f - f.min()) / (f.max()-f.min()) * 255

h = 255 * (f - f.min()) / (f.max()-f.min())
```

# Images

## Discretization – Quantize Range




Numerical example:
(for 1 pixel in $f$)

$f = 50$
$f_{min} = 20$
$f_{max} = 200$

```
In [30]: f = np.array([50], np.uint8)

In [31]: fmin = np.array([20], np.uint8)

In [32]: fmax = np.array([200], np.uint8)

In [33]: f
Out[33]: array([50], dtype=uint8)

In [34]: fmin
Out[34]: array([20], dtype=uint8)

In [35]: fmax
Out[35]: array([200], dtype=uint8)

In [36]: 255 * (f - fmin)
Out[36]: array([226], dtype=uint8)
```

Is still uint8
Can overflow!

This is **226** instead of **7650**

1 byte (uint8) cannot encode numbers <0 or >255 → **Overflow!**

Execution priority: *Left → Right*

**Essentially defines *implicit* parentheses!**

```
g =       (f - f.min()) / (f.max()-f.min()) * 255
```

```
h = 255 * (f - f.min()) / (f.max()-f.min())
```

Pixel(s) value in $f$

Maximum/Minimum value in $f$

**Mathematical equivalent:**

$$g = \left(\frac{f - f_{\min}}{f_{max} - f_{\min}}\right) * 255$$

$$h = \left(255(f - f_{\min})\right)\left(\frac{1}{f_{max} - f_{\min}}\right)$$

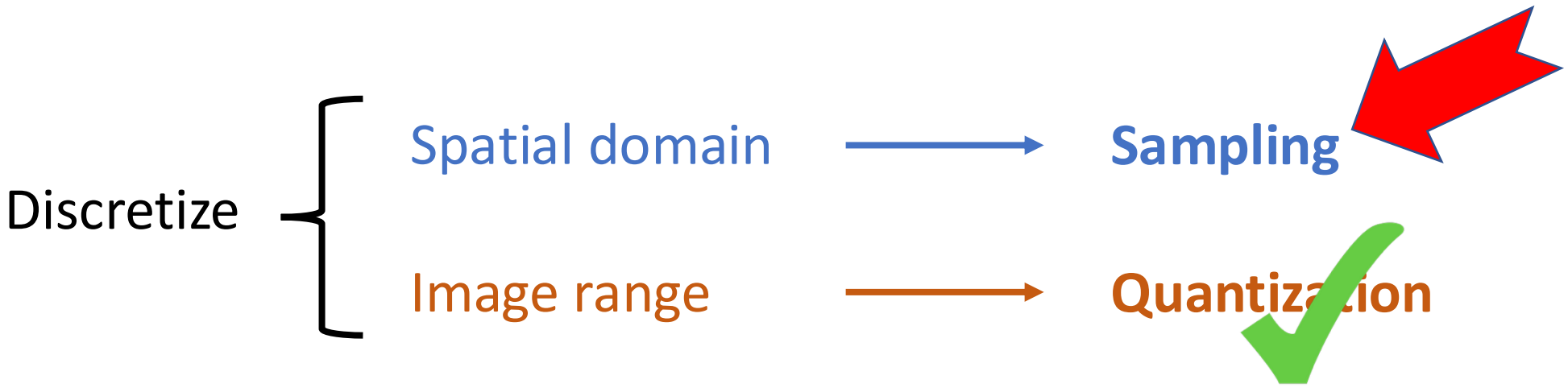
Kind note: In the e-notes you can copy the code, edit & run :)

# Images

## Discretization

How can we store a continuous grayscale image?

$$f: \mathbb{R}^2 \rightarrow \mathbb{R}^1$$



# Images

## Discretization – Sample Domain

Image pixels → apply rect. ***sampling grid***
over ***continuous*** function $f$

Each box filled with sample value

*Continuous* images → $f: \mathbb{R}^2 \rightarrow \mathbb{R}_+$

*sampling* ↓

*Discrete* images → $F: \mathbb{Z}^2 \rightarrow \mathbb{R}_+$

$$F(i,j) = f(i\Delta y, j\Delta x)$$

$i, j \in \mathbb{Z}$

Usually → $\Delta y = \Delta x = 1$

row column

$(i, j)$

Sample Coord.

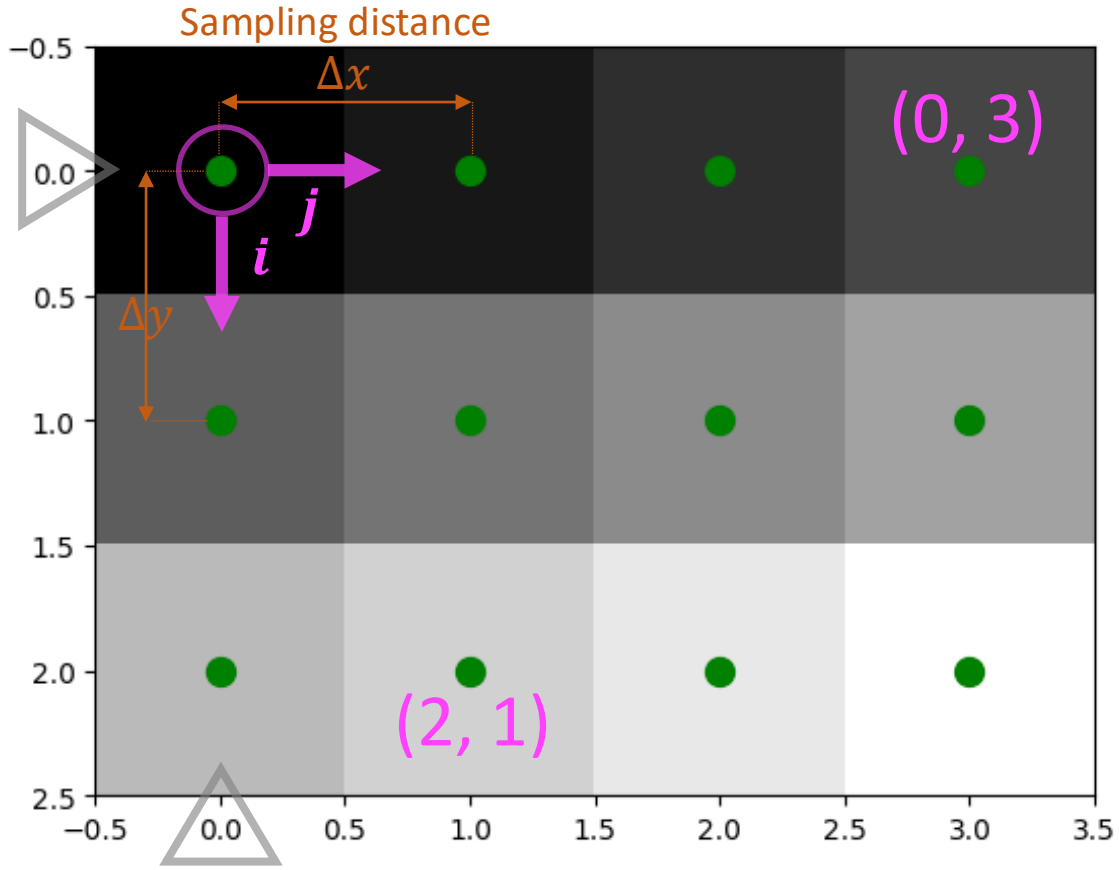
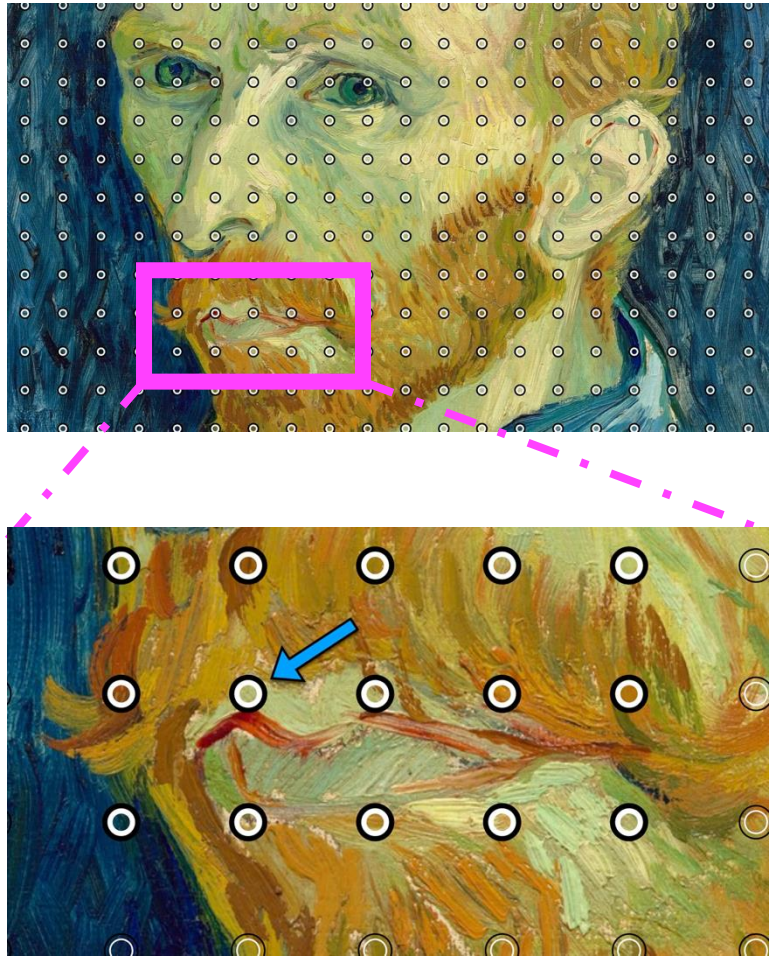



Image size → 3 x 4
rows cols

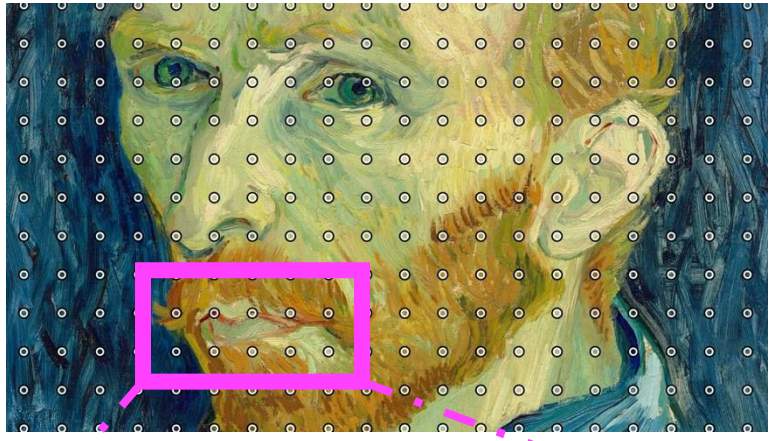
# Images

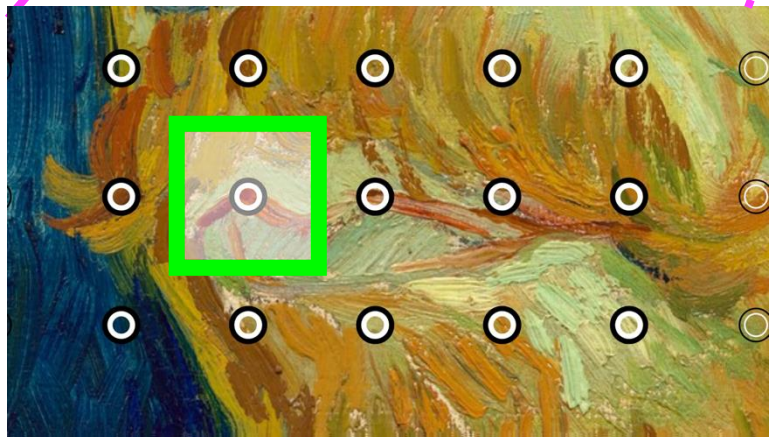
## Discretization – Sample Domain



# Images

## Discretization – Sample Domain





‘Sense’ a bigger area

Week 3
(Local Operators)

# Images

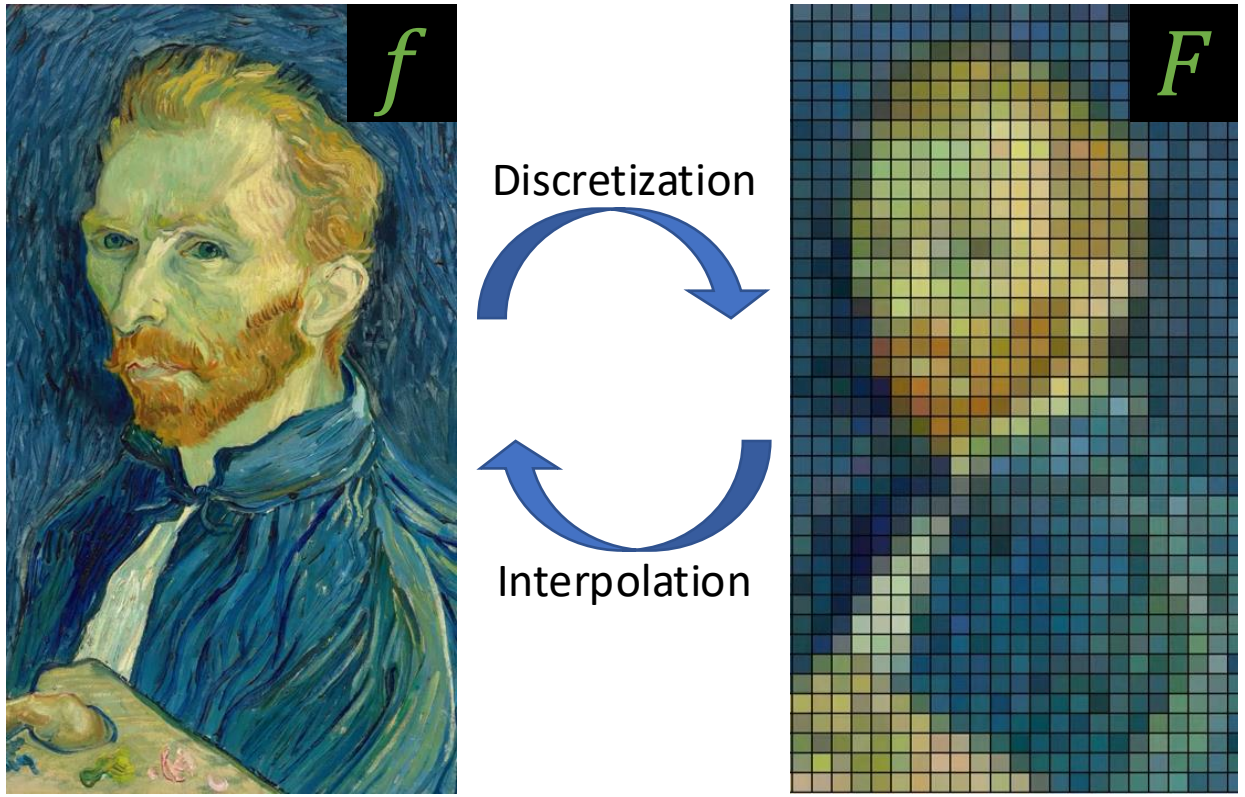
## Interpolation

Given samples $F$ of continuous $f$

Interpolation → Calculate value $f(x, y)$ *between* samples $F(i, j)$

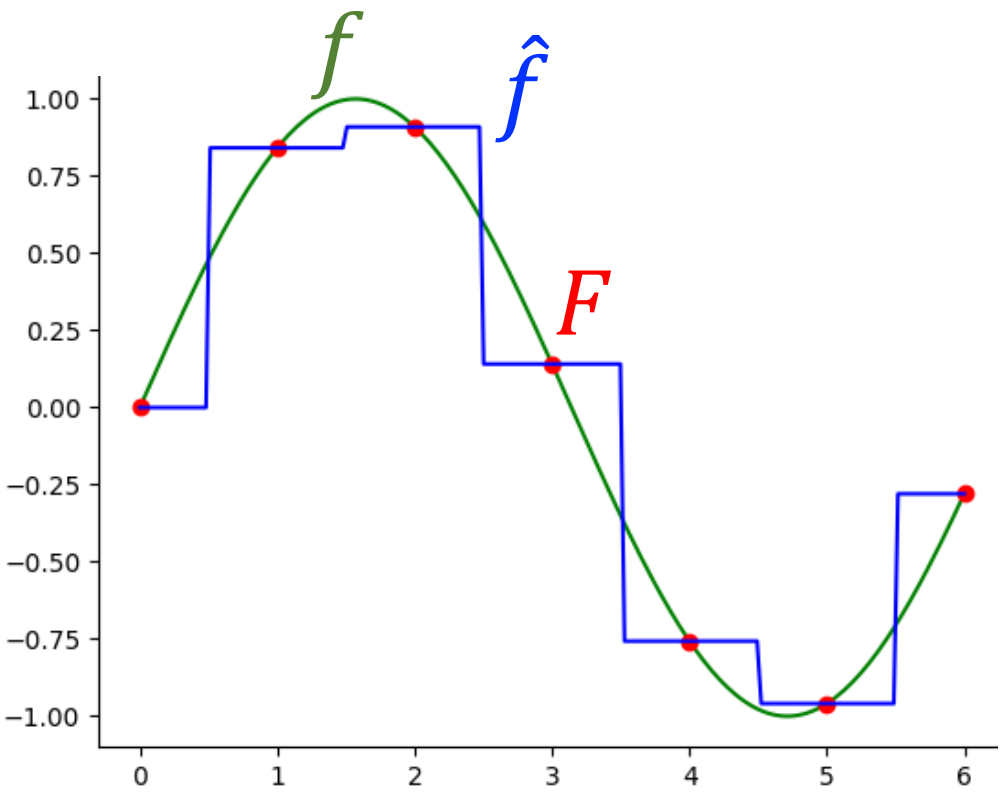
Assuming: Well-sampled image

(if curious – Information Theory: Nyquist-Shannon sampling theorem)



# Images

## 1D Interpolation - Goal




$F(x), x \in \mathbb{Z}$ → **Input:** Samples of $f$

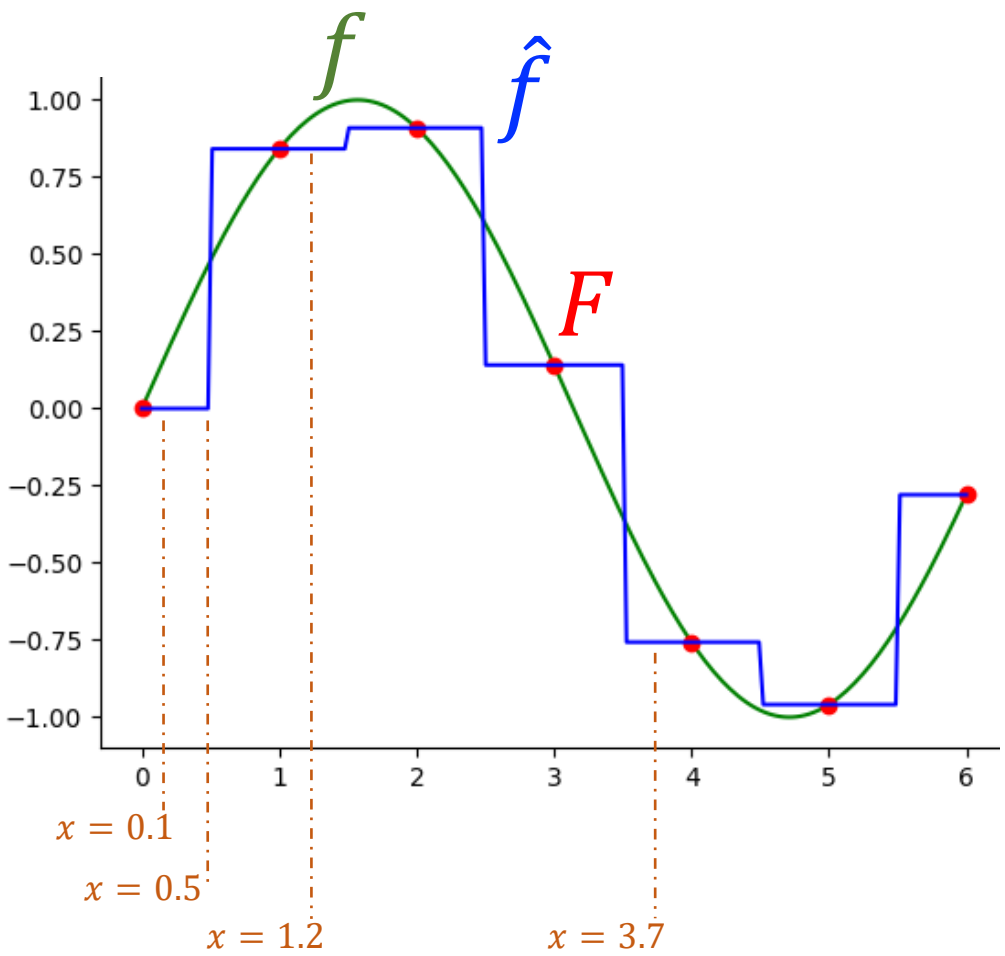
$f(x) = \sin(x),\ x \in \mathbb{R}$ → Continuous '*target*' function that we want to **guesstimate**

**Goal:**

**Guesstimate** $\hat{f}$ that **approximates** $f$ given (**input**) samples $F$

# Images

## 1D Interpolation - Nearest Neighbor




$$\hat{f}(x) = F(\lfloor x + 0.5 \rfloor)$$ → Copy nearest sample

*programmer's view:* $round(x)$

'Floor' function

$\lfloor x \rfloor$ → largest int that is $\leq x$

$$\hat{f}(0.1) = F(\lfloor 0.1 + 0.5 \rfloor) = F(\lfloor 0.6 \rfloor) = F(0)$$

$$\hat{f}(0.5) = F(\lfloor 0.5 + 0.5 \rfloor) = F(\lfloor 1.0 \rfloor) = F(1)$$

$$\hat{f}(1.2) = F(\lfloor 1.2 + 0.5 \rfloor) = F(\lfloor 1.7 \rfloor) = F(1)$$

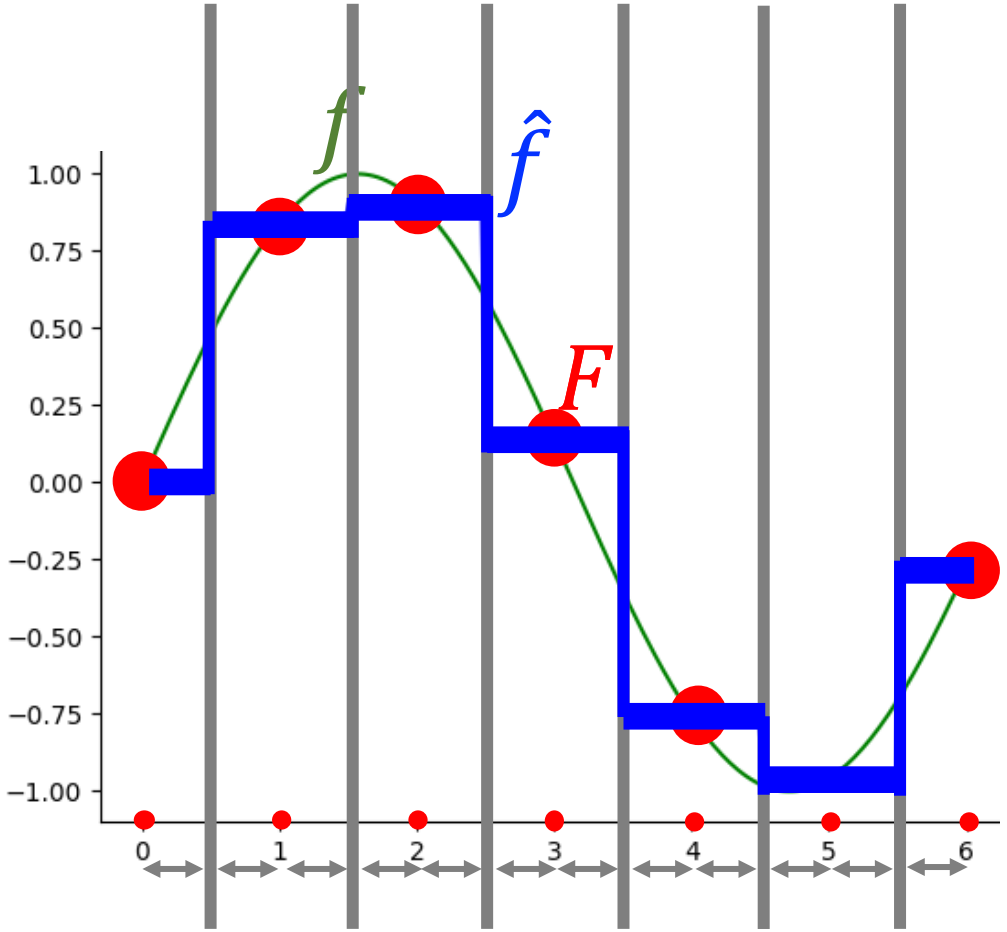
$$\hat{f}(3.7) = F(\lfloor 3.7 + 0.5 \rfloor) = F(\lfloor 4.2 \rfloor) = F(4)$$

?

*$\hat{f}$ could be be computed for each point $x$ independently! Can be heavily parallelized!*

# Images

## 1D Interpolation - Nearest Neighbor

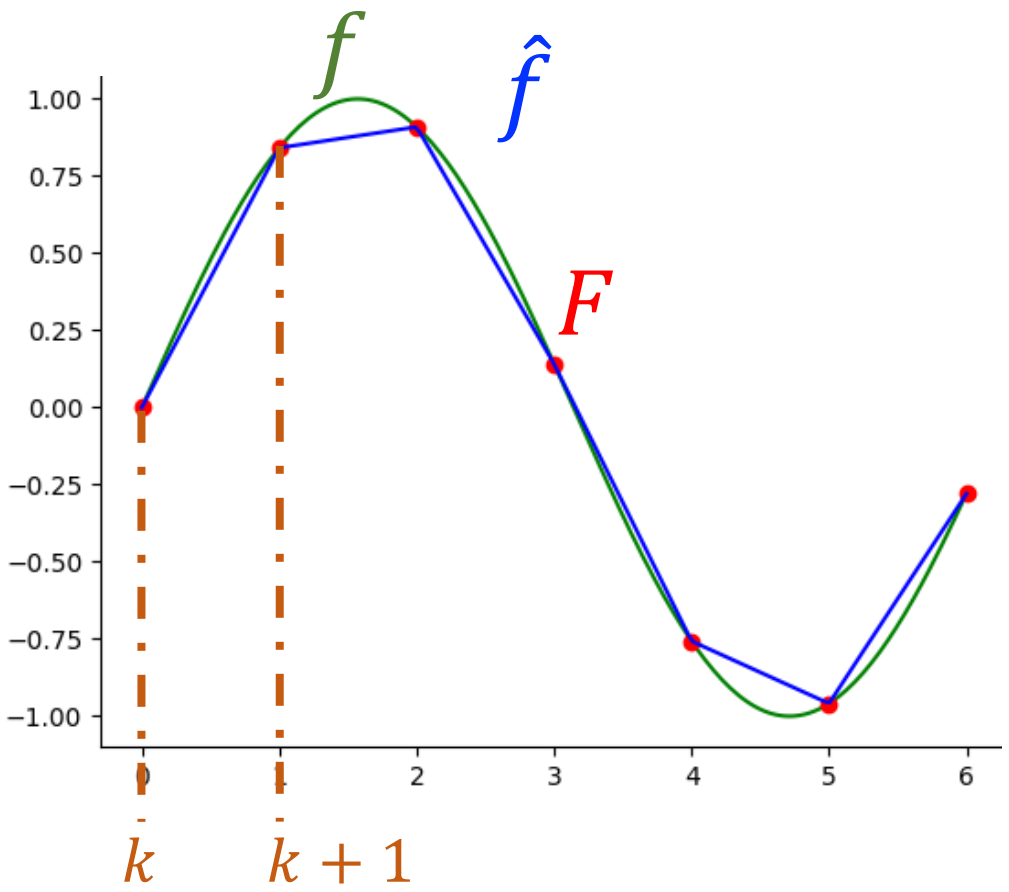



$$\hat{f}(x) = F(\lfloor x + 0.5 \rfloor)$$

$\hat{f}$ → ‘Staircase’ ~~Continuous~~ ~~Differentiable~~

# Images

## 1D Interpolation - Linear




Segment ***between*** points: $x \in [k, k+1]$

$\hat{f}$ linearly 'mixes'/connects: $F(k)$ & $F(k+1)$

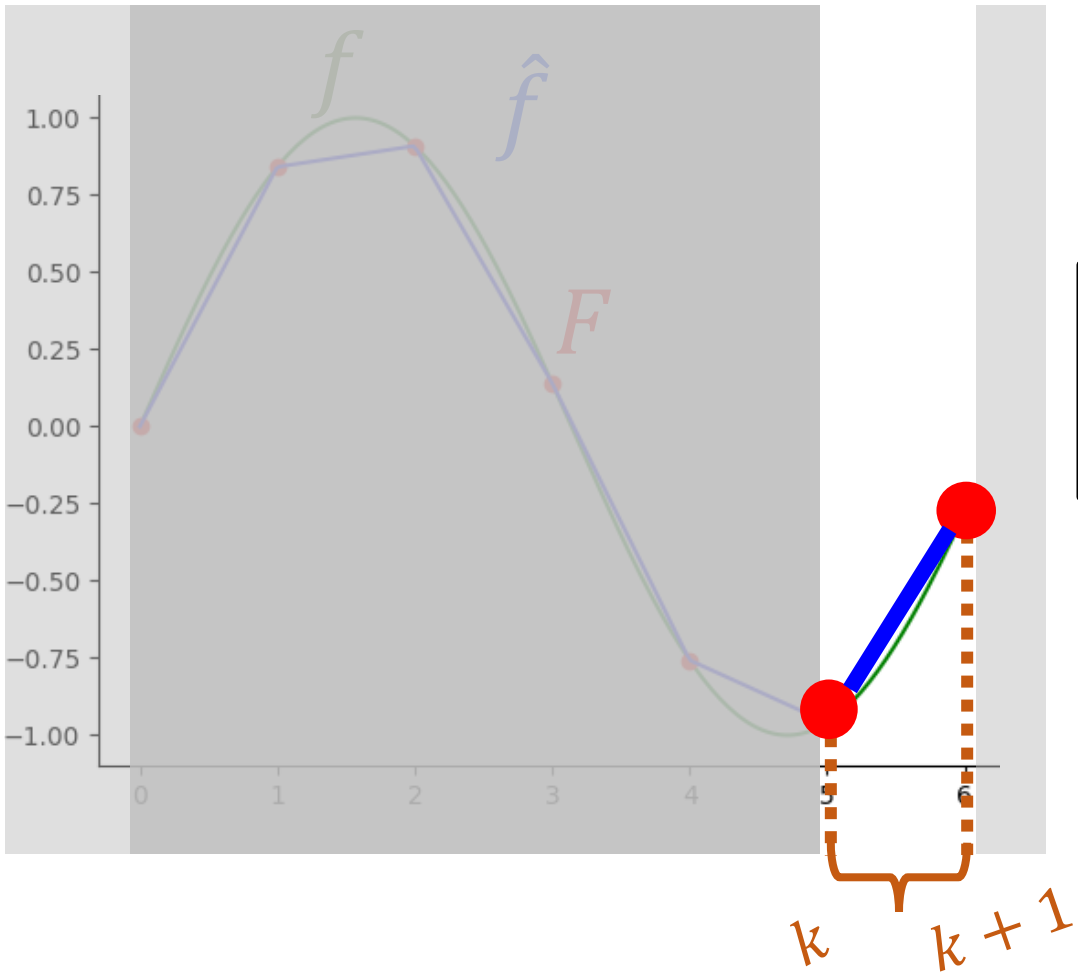
$$\hat{f}(x) = (1-(x-k))F(k) + (x-k)\,F(k+1)$$

*steering / mixing weights* — *samples*

Fit a **line** equation (y=ax+b) ***separately***
for each *pair* of *consecutive samples* $\{k, k+1\}$

# Images

## 1D Interpolation - Linear




Segment ***between*** points: $x \in [k, k+1]$

$\hat{f}$ linearly 'mixes'/connects: $F(k)$ & $F(k+1)$

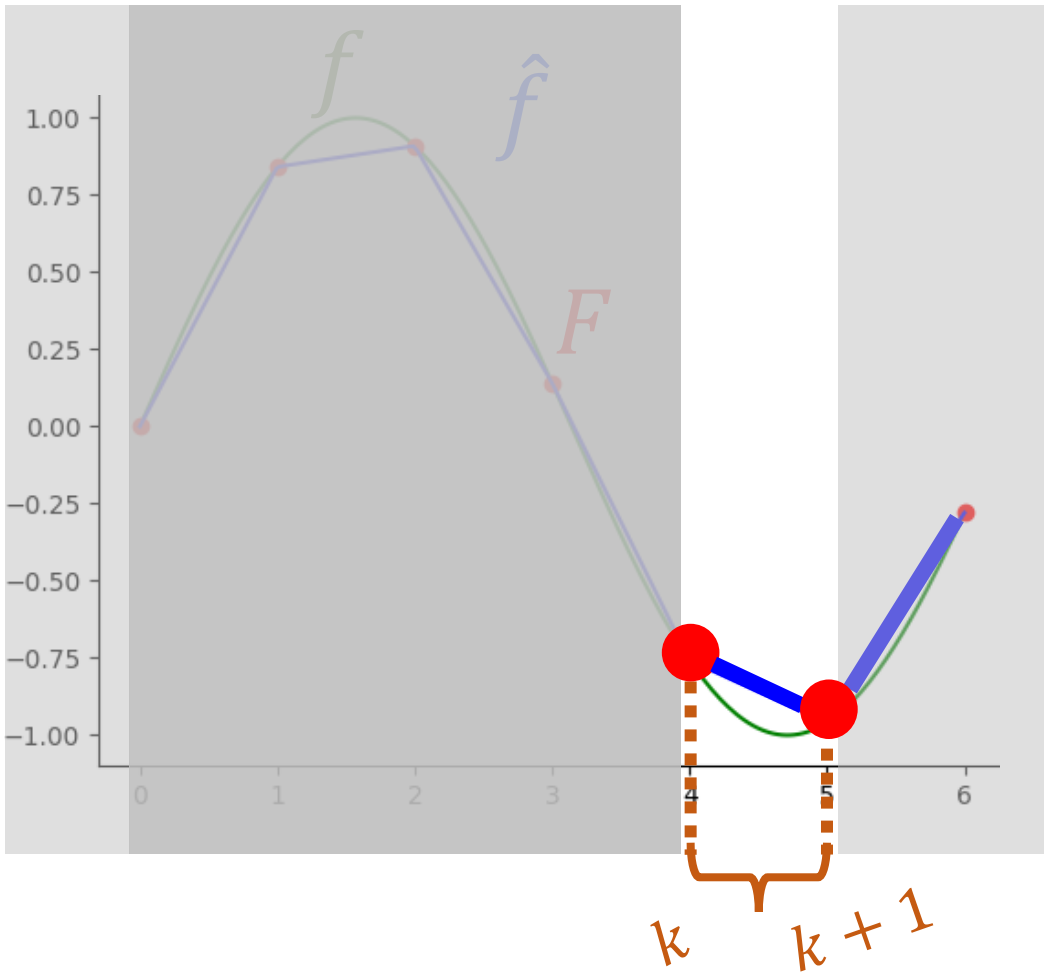
$$\hat{f}(x) = (1-(x-k))F(k) + (x-k)\,F(k+1)$$

*steering / mixing weights* — *samples*

Fit a **line** equation (y=ax+b) ***separately***
for each *pair* of *consecutive samples* $\{k, k+1\}$

# Images

## 1D Interpolation - Linear




Segment ***between*** points: $x \in [k, k+1]$

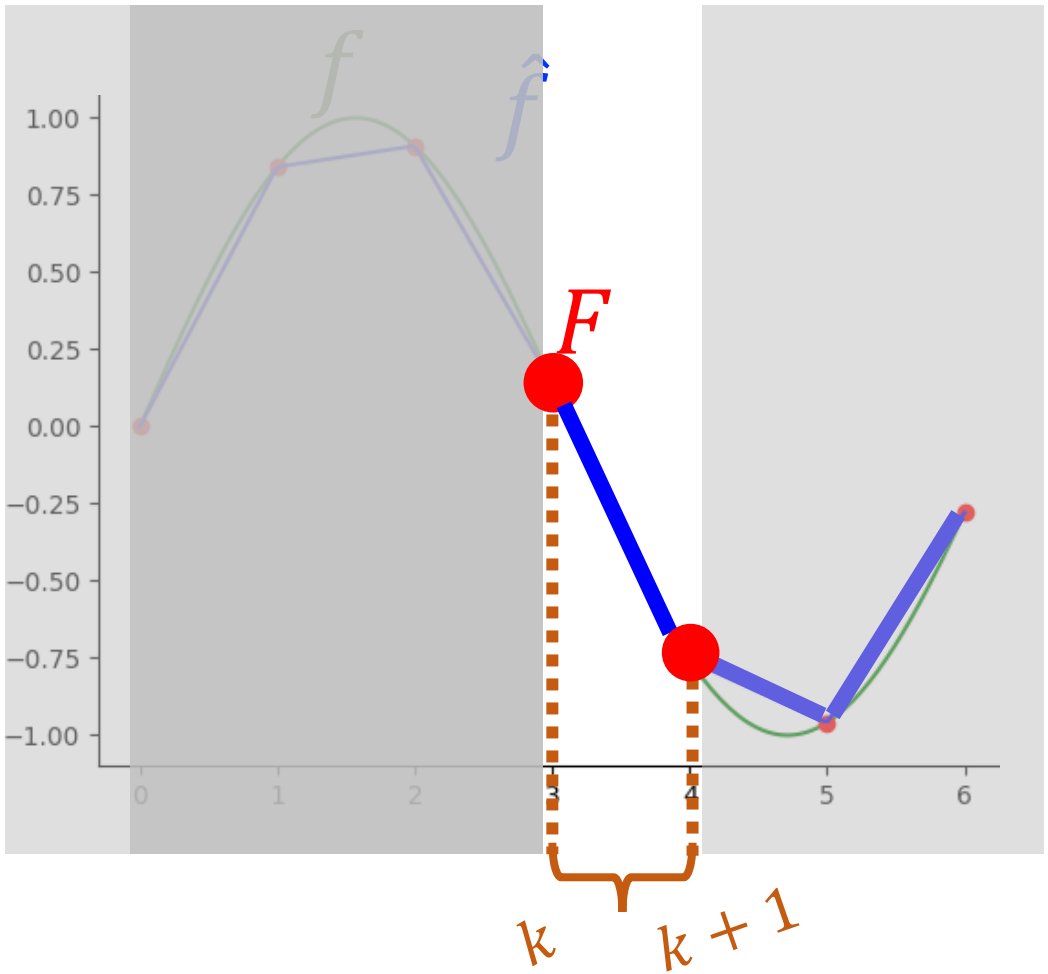
$\hat{f}$ linearly 'mixes'/connects: $F(k)$ & $F(k+1)$

$$\hat{f}(x) = (1 - (x - k))F(k) + (x - k)\,F(k+1)$$

*steering / mixing weights* *samples*

# Images

## 1D Interpolation - Linear




Segment ***between*** points: $x \in [k, k+1]$

$\hat{f}$ linearly 'mixes'/connects: $F(k)$ & $F(k+1)$

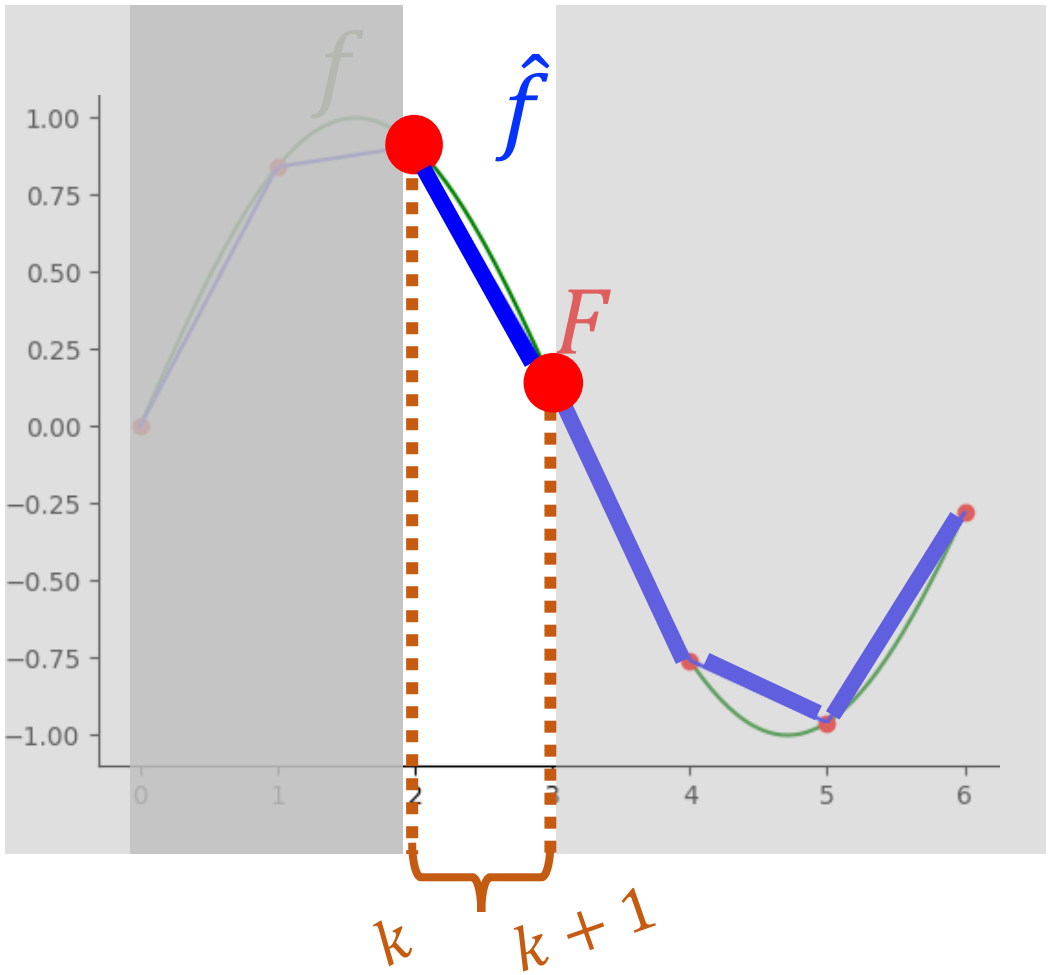
$$\hat{f}(x) = (1 - (x - k))F(k) + (x - k)\,F(k+1)$$

*steering / mixing weights*

*samples*

# Images

## 1D Interpolation - Linear




Segment ***between*** points: $x \in [k, k+1]$

$\hat{f}$ linearly 'mixes'/connects: $F(k)$ & $F(k+1)$

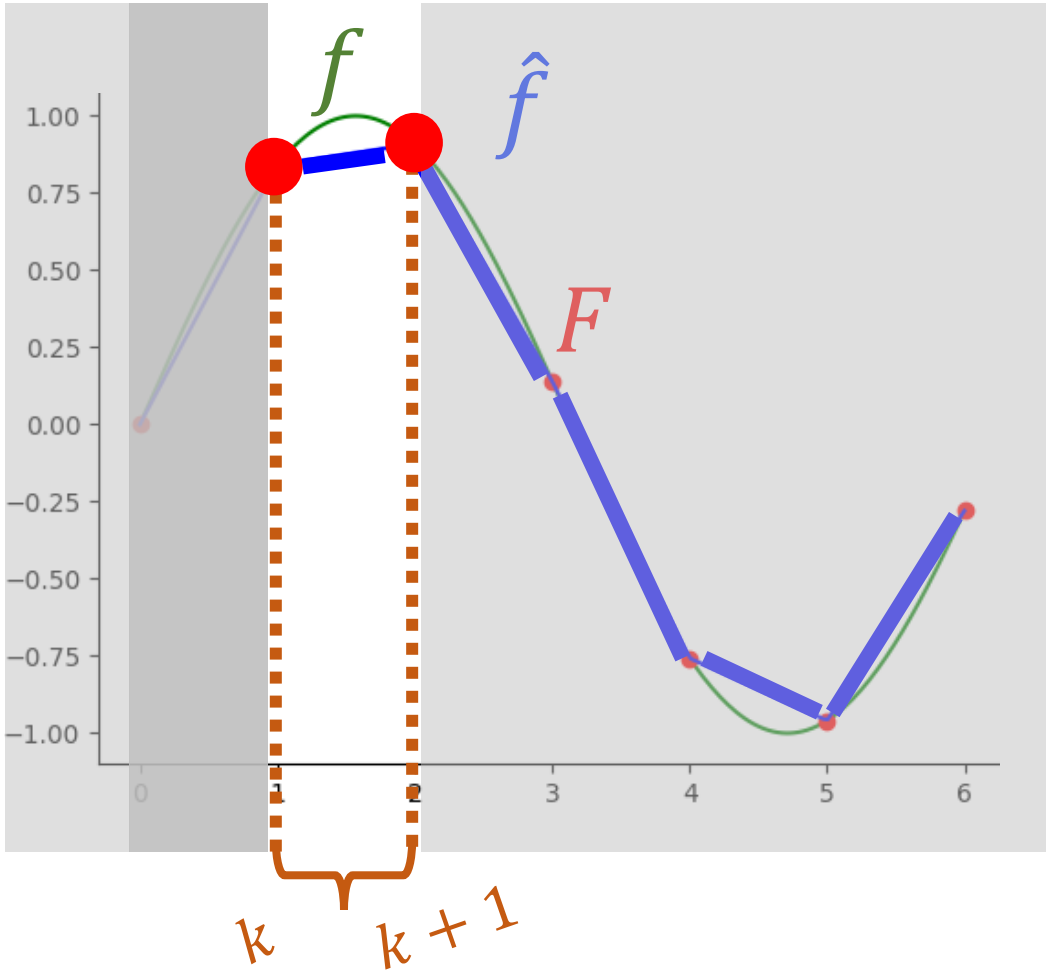
$$\hat{f}(x) = (1-(x-k))F(k) + (x-k)\,F(k+1)$$

*steering / mixing weights*

*samples*

# Images

## 1D Interpolation - Linear




Segment ***between*** points: $x \in [k, k+1]$

$\hat{f}$ linearly 'mixes'/connects: $F(k)$ & $F(k+1)$

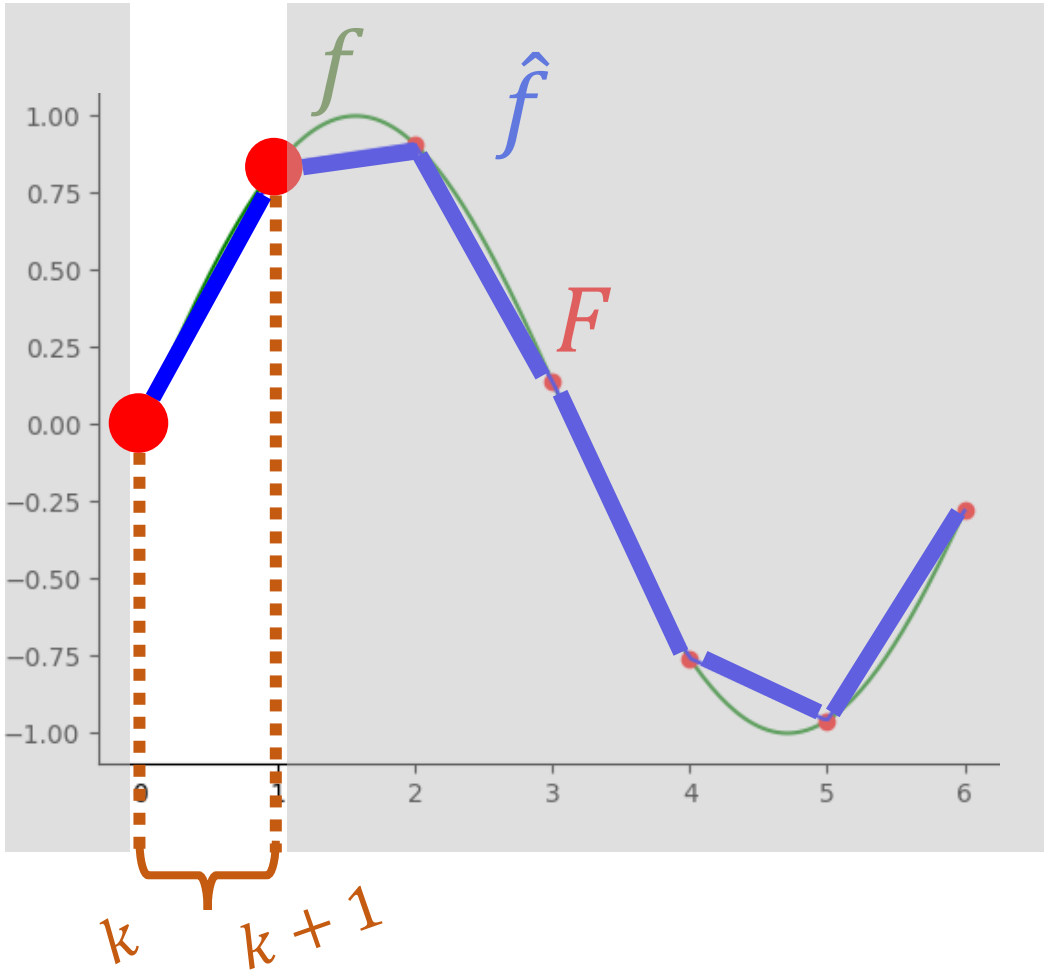
$$\hat{f}(x) = (1-(x-k))F(k) + (x-k)\,F(k+1)$$

*steering / mixing weights*

*samples*

# Images

## 1D Interpolation - Linear




Segment ***between*** points: $x \in [k, k+1]$

$\hat{f}$ linearly 'mixes'/connects: $F(k)$ & $F(k+1)$

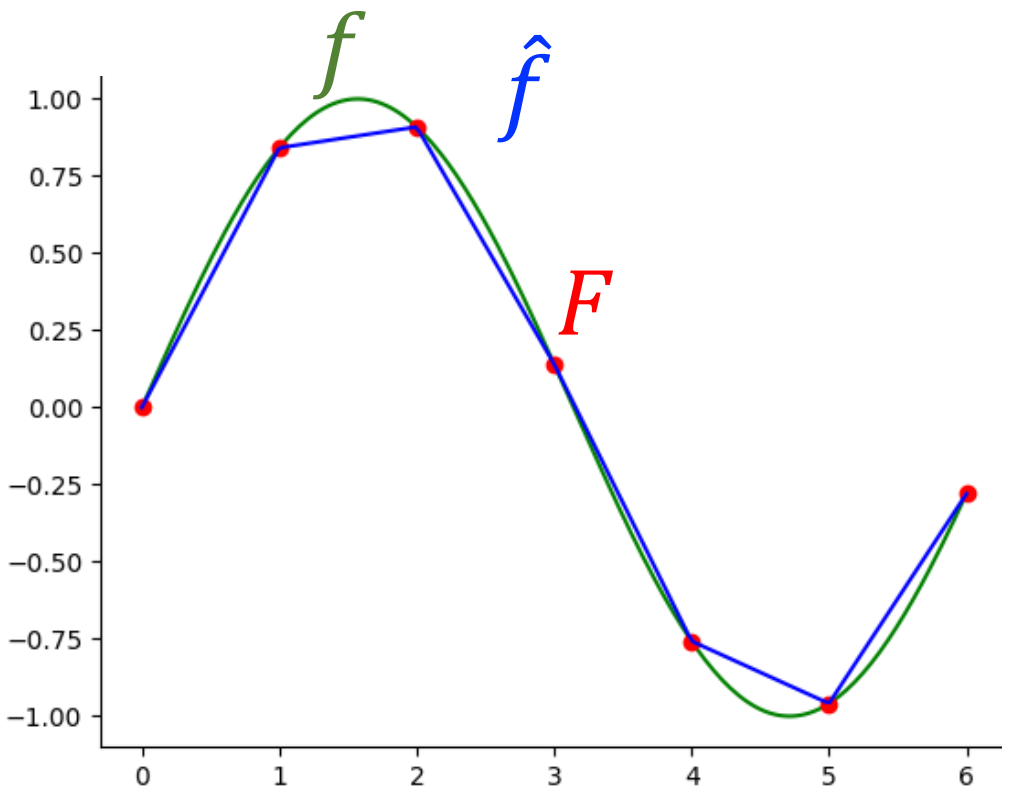
$$\hat{f}(x) = (1-(x-k))F(k) + (x-k)\,F(k+1)$$

*steering / mixing weights* — *samples*

*Each line segment of $\hat{f}$ could be computed independently from other segments – in parallel!*

# Images

## 1D Interpolation - Linear




$$\hat{f}(x) = (1 - (x - k))F(k) + (x - k)\,F(k+1)$$

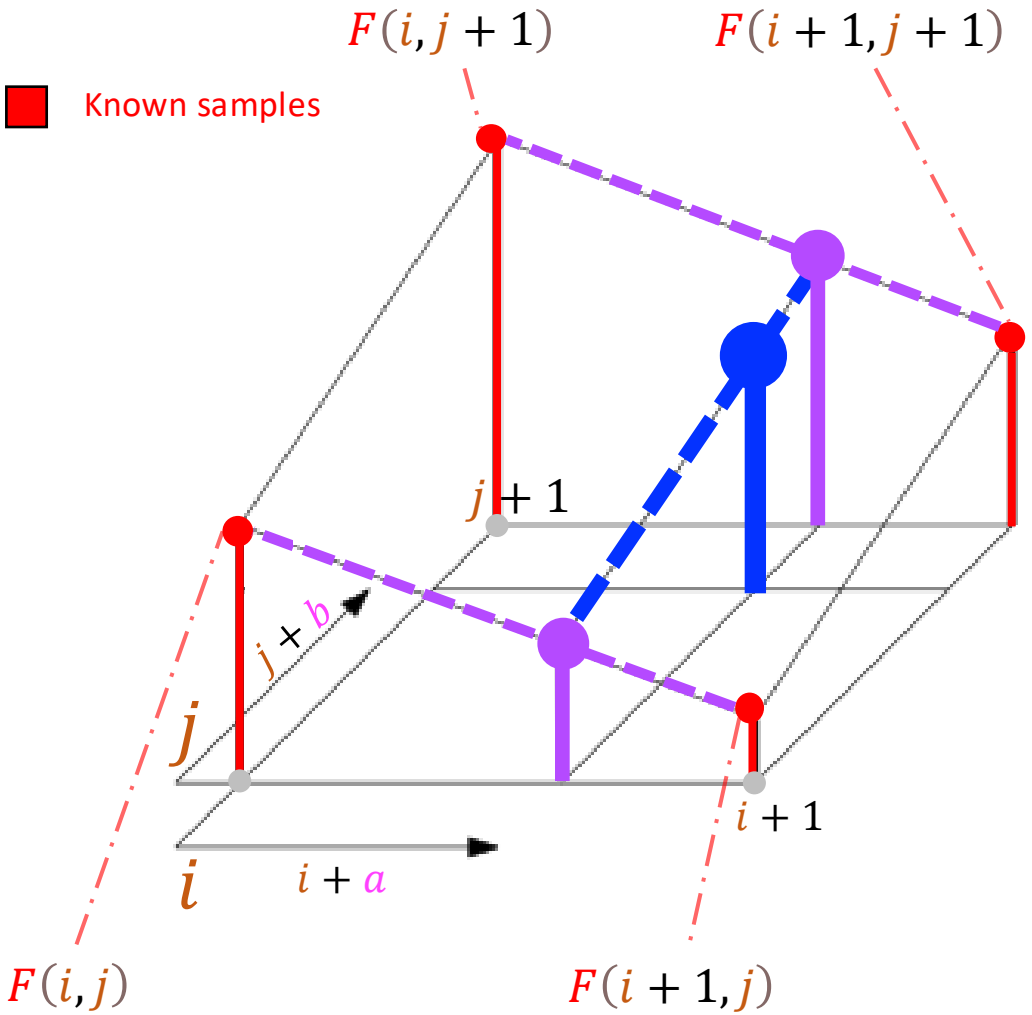
$\hat{f}$ → **Piece-wise linear** **Continuous**

**- not differentiable @ sample point locations**

**- high-order deriv. = 0**

# Images

## 2D Interpolation - Bilinear




Estimate $\hat{f}(x, y)$ for $i \le x \le i+1$, $j \le y \le j+1$

$x = i + a$
$y = j + b$
$a, b \in [0,1]$

2-step process:

- First, 1D interpolation ***in x-direction***

$$\hat{f}(x, l)_{\,l=\text{fixed}} = (1-(x-k))F(k, l) + (x-k)\,F(k+1, l)$$

- Then 1D interpolation ***in y-direction***

**Final 2D interpolation** $\hat{f}(x, y)$

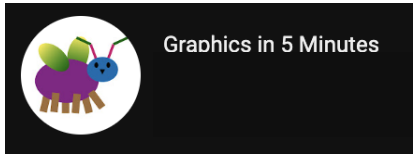
$$\begin{aligned} f(x, y) = f(i+a, j+b) = &(1-a)(1-b)\,F(i, j) \\ &+(1-a)\quad b\quad F(i, j+1) \\ &+\quad a\quad (1-b)\,F(i+1, j) \\ &+\quad a\quad\quad b\quad F(i+1, j+1) \end{aligned}$$

Known samples!

# Images

## Resources for these slides

https://rvdboomgaard.github.io/ComputerVision_LectureNotes/LectureNotes/IP/Images/index.html




https://youtu.be/xUzhKqf22mY

Duration: 4:37 min

→ If you have any suggestions, please post on Canvas ←