

CV 2026 – Week 1 – HC1a

Images and Interpolation

Dimitris Tzionas

d.tzionas@uva.nl, CC: e.a.veltmeijer@uva.nl

(please CC your TA)

Images

How do we represent images?

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



Functions

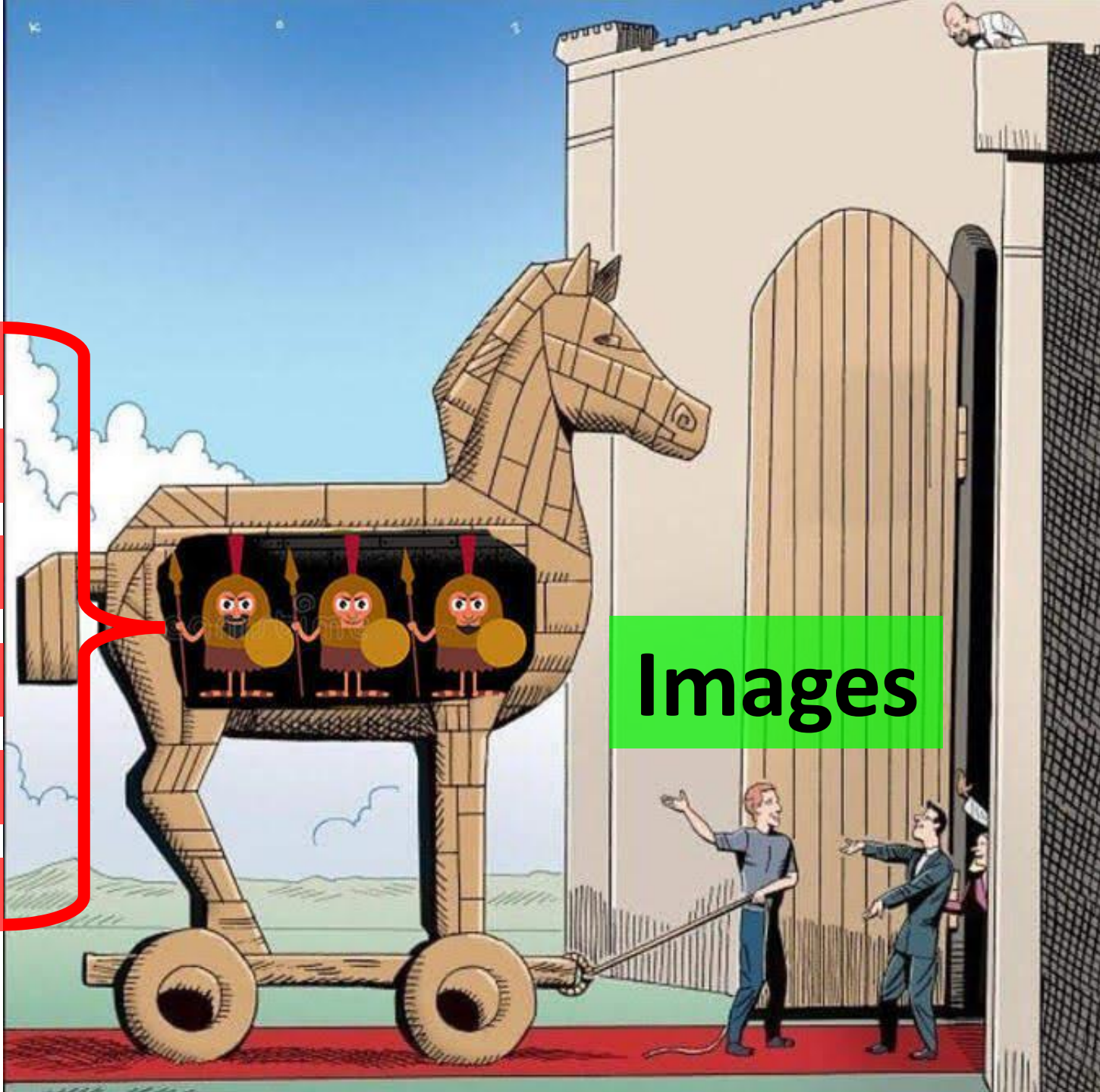
Cont./Disc. Space

Sampling Grid

Quantization

Interpolation

Histograms



Images

Images

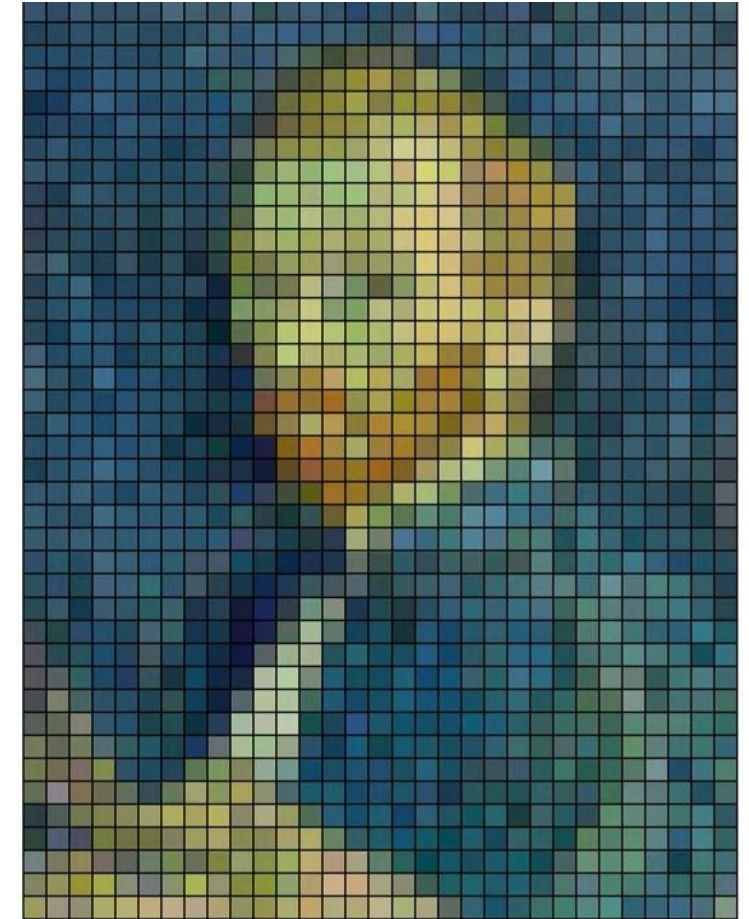
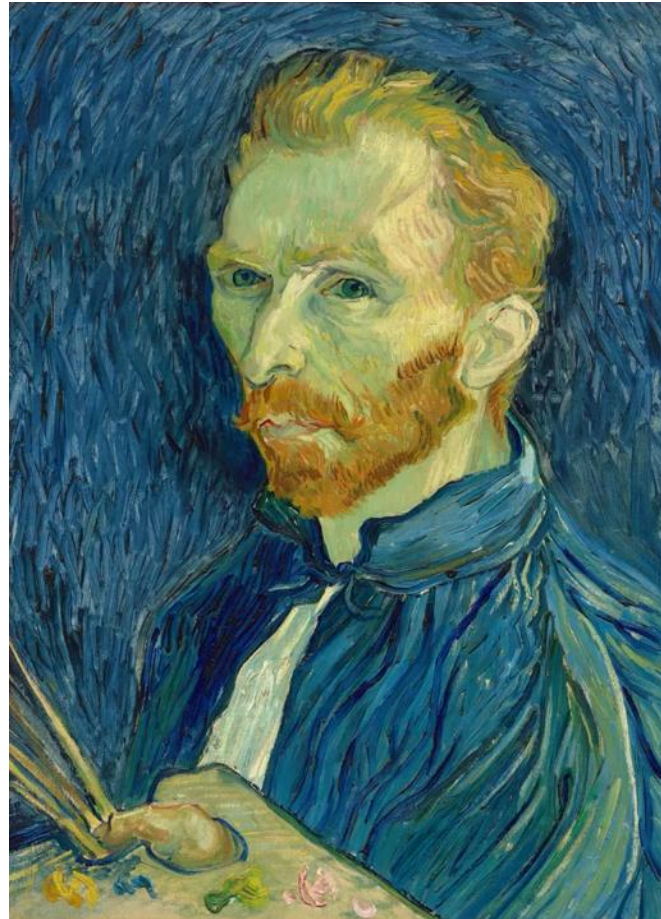
Standard 'concept'

Pixels

'Atoms' of an image

Picture **e**lements

Each pixel → **Color triplet**



e.g. 800 x 600 pixels

Images

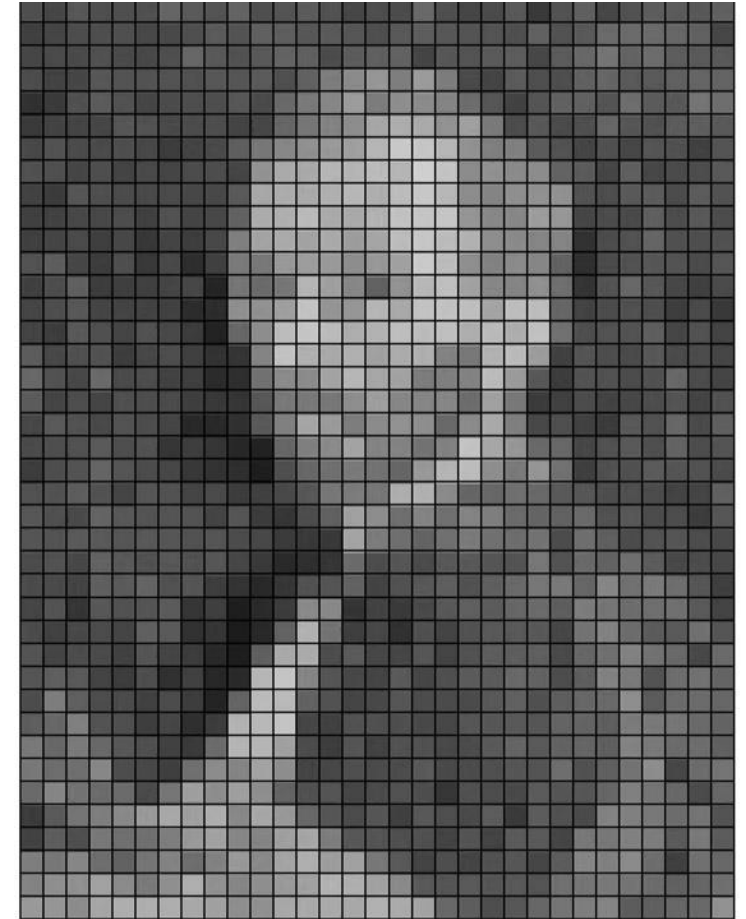
Standard 'concept'

Pixels

'Atoms' of an image

Picture elements

Each pixel → **Gray**
'luminance'
value



e.g. 800 x 600 pixels

Images

Standard 'concept'

Pixels

'Atoms' of an image

Picture elements

Each pixel → *Gray 'luminance' value*



95	85	101	93	91	93	115	103	94	114	86	97	108	96	90	90	85	119	95	81	90	109	70	89	108	82	93	105	103	99	109
77	83	92	92	94	85	86	91	107	102	112	110	100	101	100	88	103	110	105	99	97	93	112	95	102	121	120	119	113	109	80
86	90	90	95	90	86	93	116	101	95	104	109	101	80	95	102	108	81	100	96	101	105	83	106	117	130	112	120	104	102	101
95	82	87	104	100	99	98	93	95	92	103	82	94	122	130	146	122	152	126	79	81	110	95	91	110	114	103	101	110	114	115
63	74	101	110	97	85	90	92	92	97	87	108	117	132	146	121	132	133	127	108	92	80	103	114	112	100	122	105	103	133	126
74	84	86	87	103	96	87	90	96	78	78	104	116	130	136	162	163	167	147	153	117	80	101	90	93	119	102	98	108	110	104
86	98	89	89	94	87	72	80	87	104	116	132	140	144	158	175	194	192	178	145	123	123	94	93	82	94	84	100	98	78	98
94	92	83	93	79	86	82	89	88	76	139	164	162	169	175	177	181	194	183	146	154	134	137	110	106	83	110	85	89	90	103
80	70	100	93	89	83	85	84	86	93	161	163	165	168	169	173	178	192	183	146	135	145	143	143	97	91	106	109	92	107	93
76	78	84	86	90	72	92	82	82	96	164	166	179	165	160	175	190	190	192	148	134	135	154	148	84	89	88	111	93	106	89
77	81	87	75	84	64	77	71	80	121	158	163	155	166	152	161	185	192	182	173	139	139	132	132	59	90	94	117	97	94	98
97	107	83	73	93	67	84	53	76	114	118	151	150	154	140	148	158	193	183	141	129	140	134	97	56	88	97	103	101	92	104
77	91	113	75	75	81	85	47	114	108	123	177	162	135	99	140	176	186	159	134	131	144	110	61	97	118	105	88	81	91	
83	91	103	85	82	78	77	53	55	140	141	156	187	146	145	176	165	186	188	182	158	185	188	107	76	90	93	94	77	91	71
83	73	105	97	83	95	84	73	42	103	135	177	188	179	161	171	182	183	185	167	178	170	190	119	90	96	97	110	85	98	81
107	84	94	97	74	82	89	76	67	60	130	183	171	165	165	157	160	170	131	121	176	184	157	68	96	87	86	103	89	91	86
118	95	86	124	90	92	79	75	57	61	117	136	135	168	170	157	160	127	130	128	177	151	95	70	95	94	88	98	122	89	80
106	83	92	83	96	92	88	74	67	65	96	121	125	124	141	158	126	135	139	108	157	122	65	72	89	81	85	89	97	76	95
84	104	98	88	100	93	82	48	51	59	100	131	168	154	121	139	142	142	106	140	167	128	77	85	72	99	94	114	85	114	
89	92	96	107	93	96	63	74	59	50	49	104	133	138	162	134	134	119	129	185	160	147	109	107	77	91	100	91	111	86	98
72	93	99	118	102	98	85	67	61	67	45	82	114	116	127	119	105	151	173	183	143	125	153	119	73	96	90	104	96	91	83
97	96	109	97	81	102	96	81	83	78	82	75	91	124	124	120	165	168	136	110	116	123	114	98	100	91	88	87	76	77	108
102	92	88	116	102	98	99	98	76	99	69	54	81	102	155	134	111	131	115	144	141	118	111	107	115	93	96	86	87	68	101
99	98	104	102	99	102	101	97	101	98	80	69	68	68	162	127	126	109	119	131	121	114	120	82	118	107	86	96	95	83	83
93	91	89	115	103	100	95	98	99	83	68	56	64	65	112	84	93	84	98	99	92	93	127	91	114	108	124	108	87	89	83
87	92	72	114	107	102	97	98	73	63	58	70	93	95	111	74	89	100	106	90	103	115	128	127	113	145	122	126	113	96	74
80	95	66	103	94	92	102	65	76	43	67	73	138	143	70	76	63	85	104	103	102	115	120	98	127	124	116	116	118	99	78
88	67	91	89	80	110	94	65	65	43	60	105	174	122	78	82	53	98	83	90	101	103	103	76	117	133	116	103	103	121	93
94	80	84	90	82	93	83	74	54	48	109	166	170	88	102	83	81	106	83	93	101	111	112	82	111	118	136	121	114	78	117
97	106	95	82	76	89	59	78	52	60	182	187	140	101	80	81	103	88	95	103	97	120	111	83	98	135	127	87	114	122	110
89	126	107	90	88	79	82	73	48	114	185	183	105	97	83	84	99	91	90	86	98	108	87	100	100	107	148	99	114	115	107
101	107	131	98	94	66	79	57	77	178	184	197	74	77	106	92	92	86	89	99	102	97	95	103	103	97	128	127	125	110	102
123	106	121	114	96	90	72	75	119	176	170	165	93	78	98	74	95	94	82	91	103	107	92	108	107	98	127	127	112	111	108
121	118	124	130	90	85	79	114	137	133	165	153	77	87	95	93	86	103	85	87	89	97	91	111	86	95	118	150	83	118	121
111	145	121	116	112	85	120	154	139	157	145	107	90	100	86	90	89	94	67	88	79	81	99	92	97	100	133	123	113	86	105
65	102	115	129	133	155	160	159	172	138	165	151	110	99	86	96	82	104	88	82	100	62	102	91	84	128	130	126	114	127	87
70	106	92	113	114	110	130	150	157	134	135	157	163	127	95	93	92	90	75	99	80	101	97	75	117	119	128	135	104	126	99
134	126	136	113	106	122	127	108	126	149	135	156	170	160	166	125	94	71	87	80	92	93	75	102	132	131	135	120	71	115	115
133	145	134	163	142	129	138	169	150	126	131	148	162	184	178	160	142	95	90	95	94	100	92	116	124	123	119	116	103	106	120
146	156	149	157	170	177	181	170	164	140	139	165	168	168	168	181	169	163	111	105	88	96	108	118	132	132	103	109	114	117	145

e.g. 800 x 600 pixels

Images

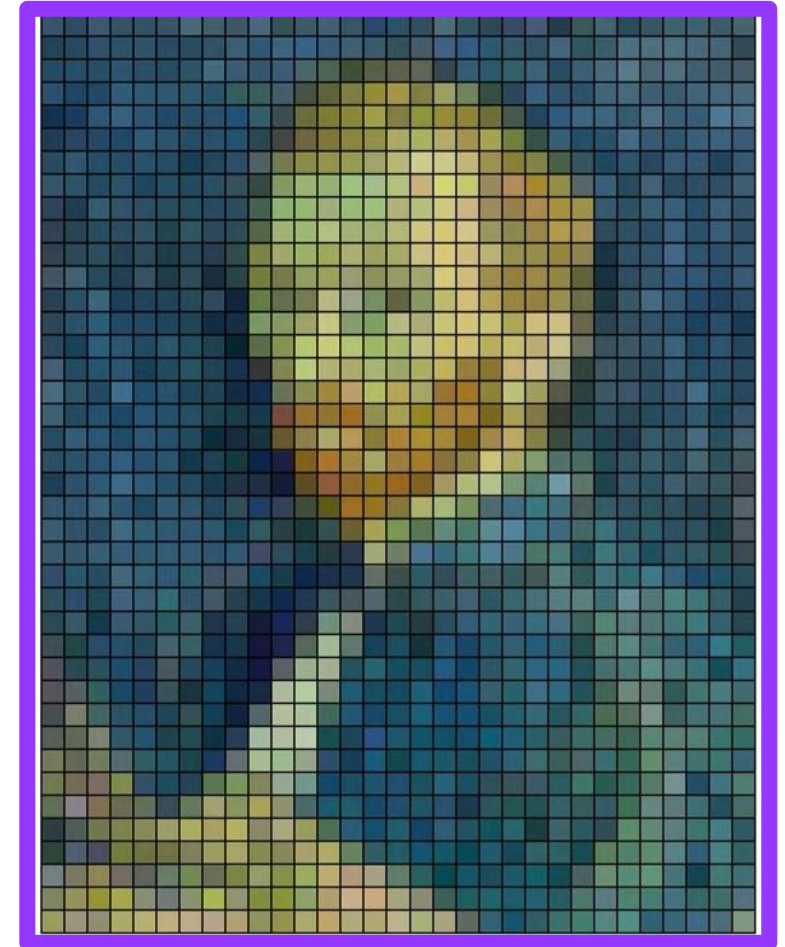
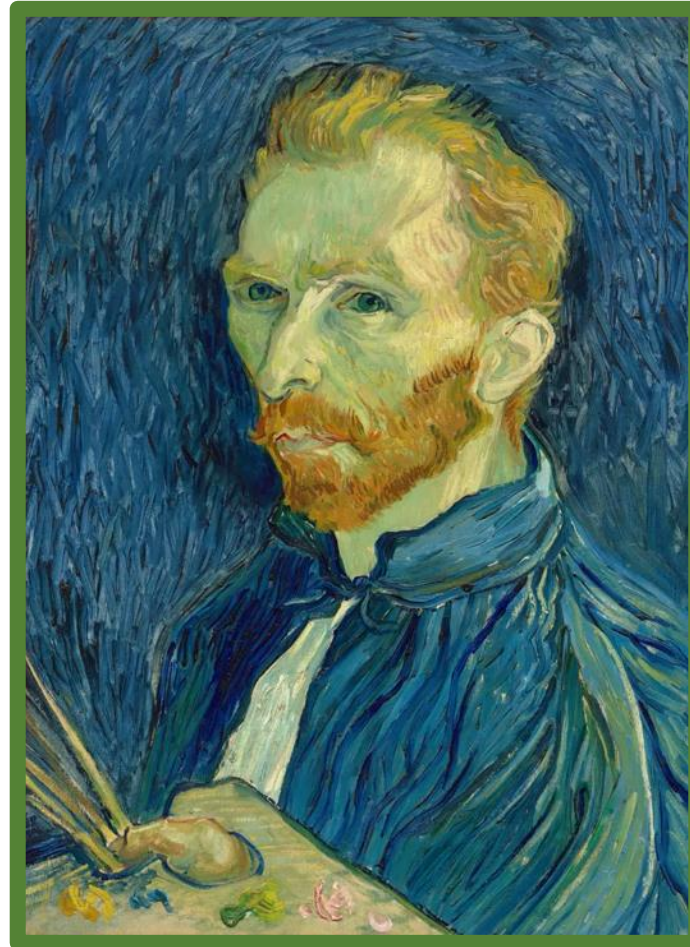
Standard 'concept'

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)

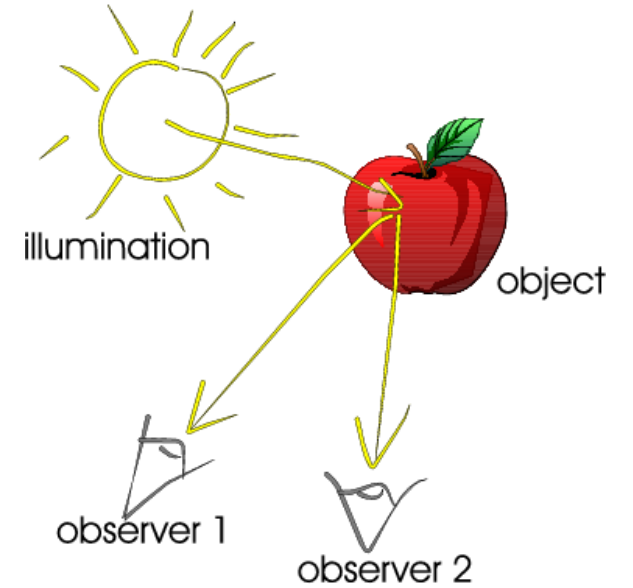
Measure

electromagnetic energy

@ every point of

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

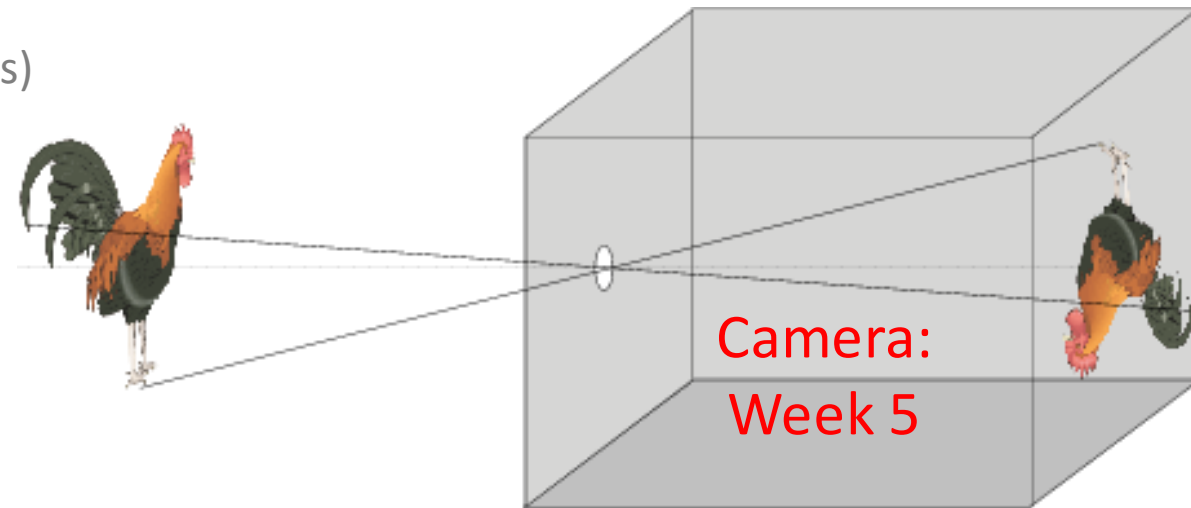
Function over a
spatial domain
(position on sensor)



Mathematical model



Image



Images

Measuring Light

Physical observation

==

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

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

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

Mathematical model



Image

Measure

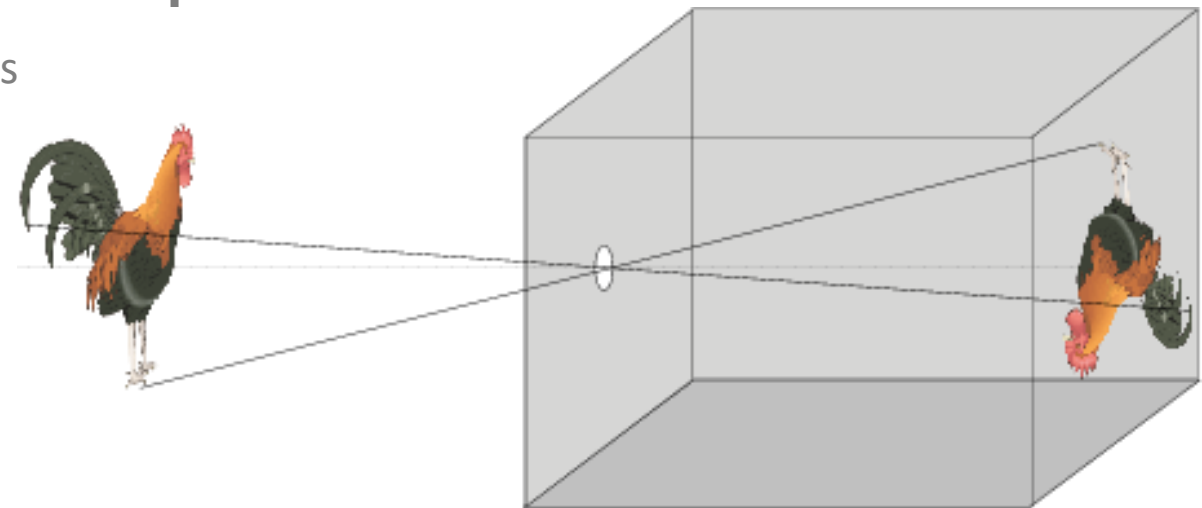
electromagnetic energy

@ every point of

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

Image $\rightarrow f(x)$

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



Images

Measuring Light

Physical observation

==

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 $\rightarrow f(x)$

Value f proportional to measured energy at position x

Defined over a
continuous spatial domain

Mathematical model



Image

Images

Measuring Light

How to measure light over a **continuous** domain?

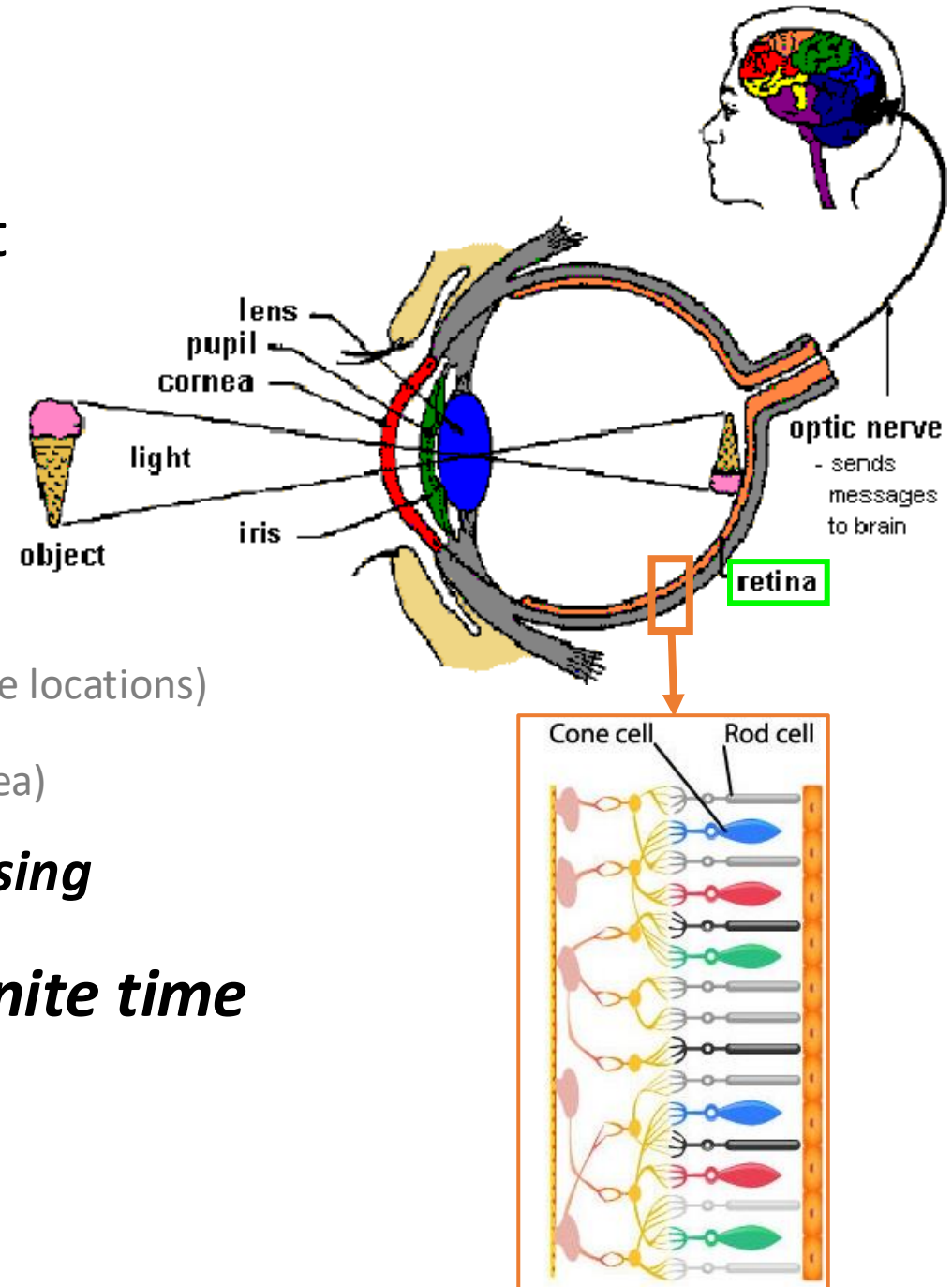
In theory \rightarrow Infinite sample points $x \in E$

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

Size (sampling area)

Temporal sensing

Integrate sensed energy \rightarrow 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)

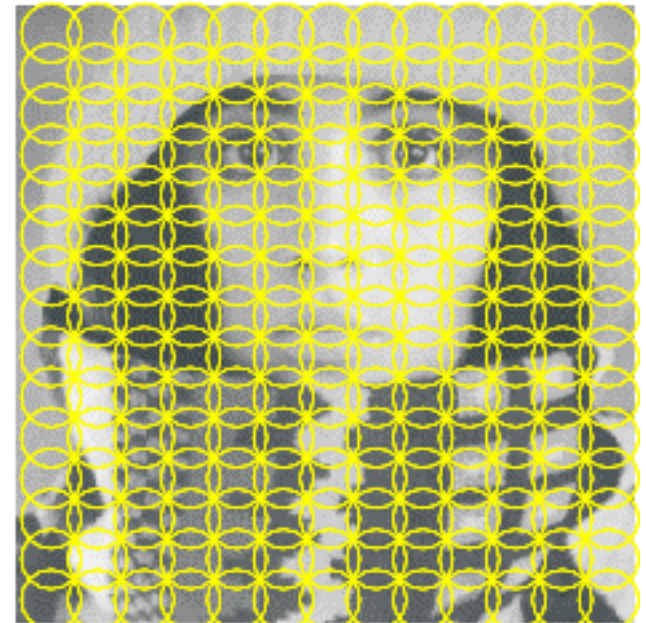
Temporal sensing

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



Choose sample ... {
-distance
-area size

?? Pros / cons →

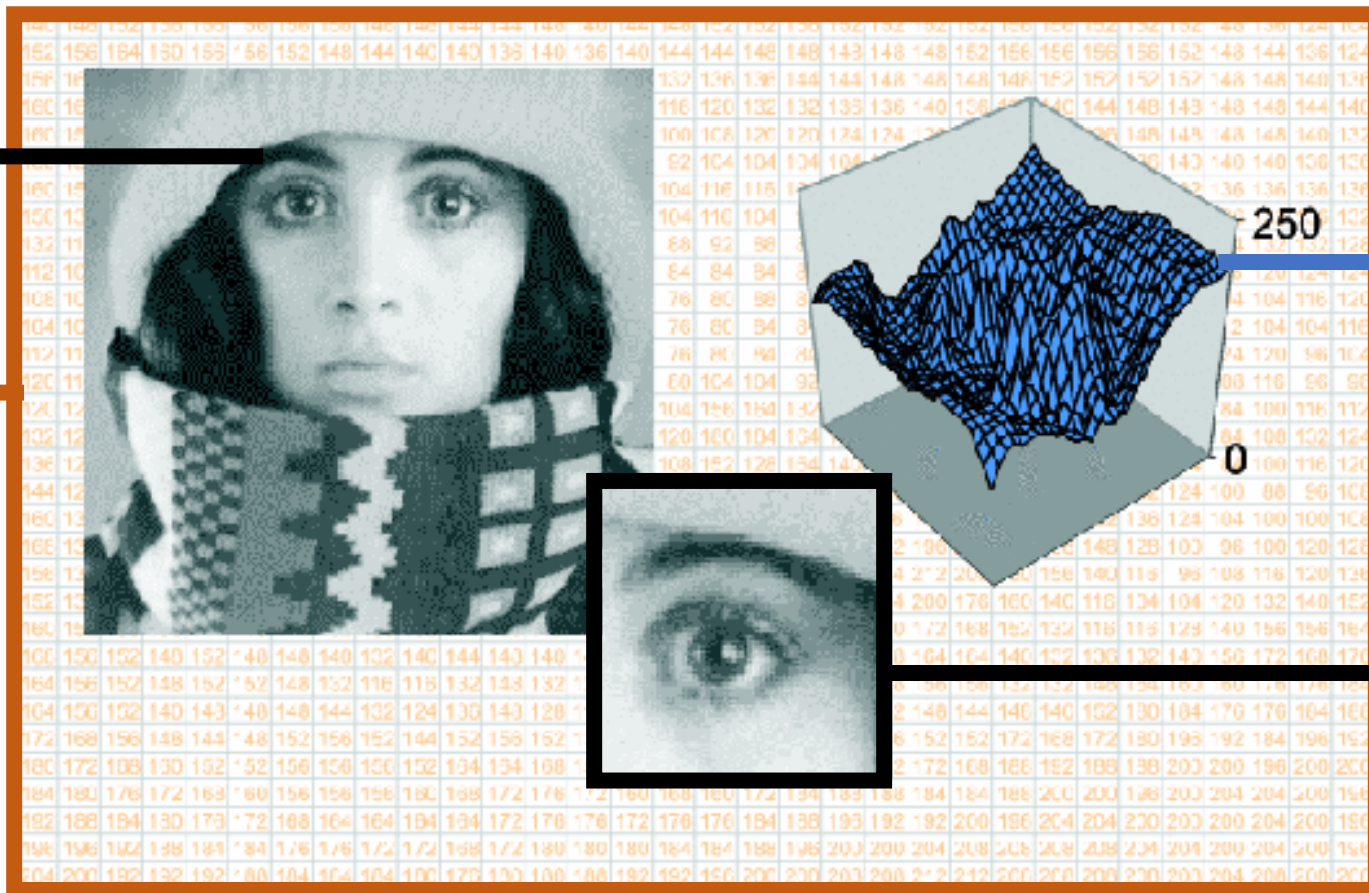


Images

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



$E \in \mathbb{R}^2$ ✓

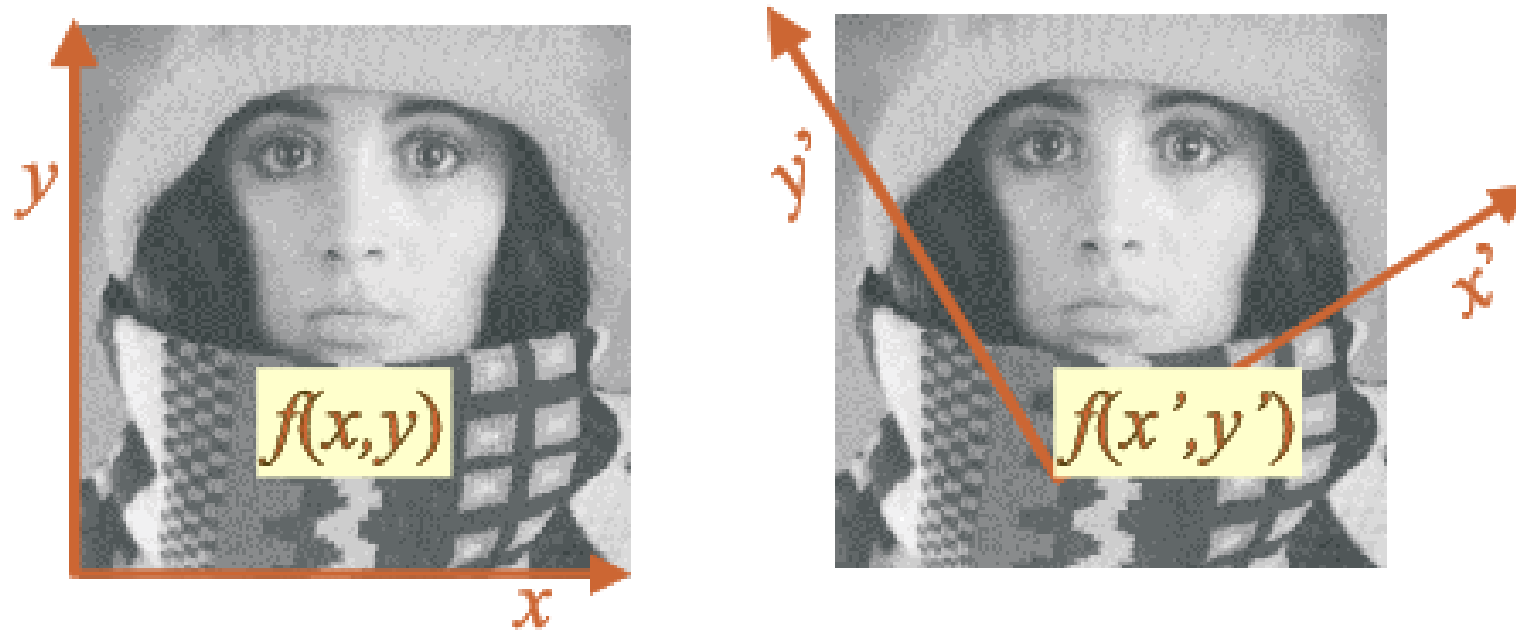


$E \in \mathbb{R}^3$



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'

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



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

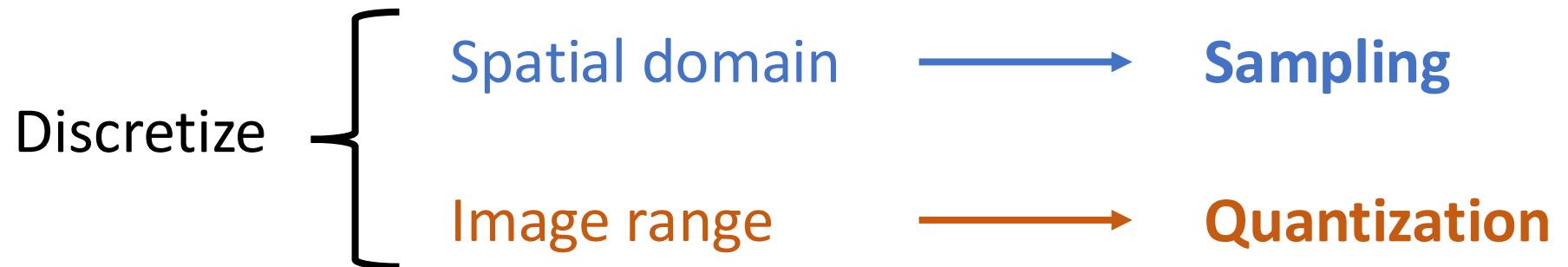
aka: 'Semantic Segmentation'

Images

Discretization

How can we *store* a **continuous** **grayscale** image?

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



Images

Discretization – Quantize Range



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

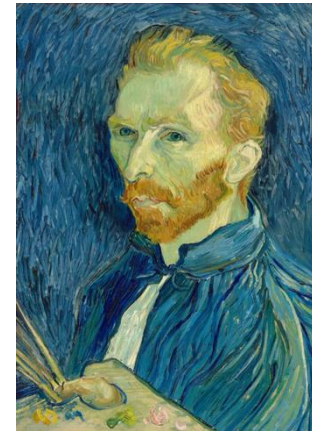


Range: 0 ... 255

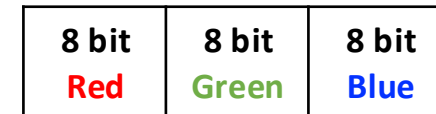
We can store a total of $2^8 = 256$ values

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

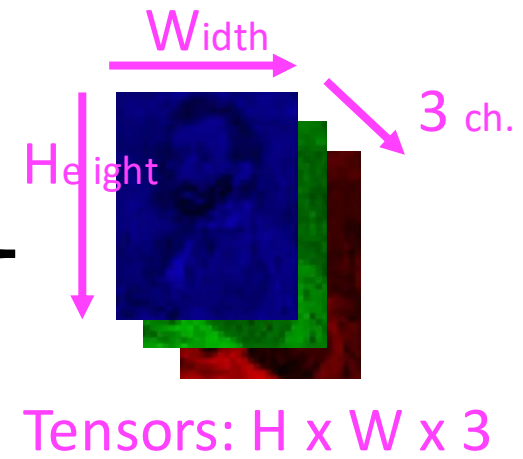
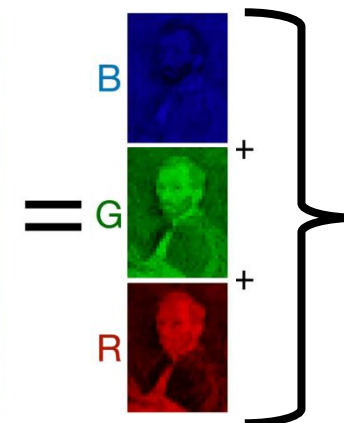
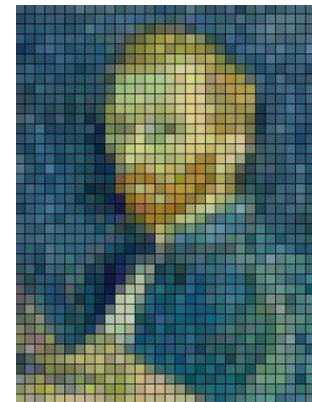
Values > 255
Values < 0 → **Bit Overflow**



3 byte / 24 bit



order depends on framework!!!
OpenCV



Images

Discretization – Quantize Range



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



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'

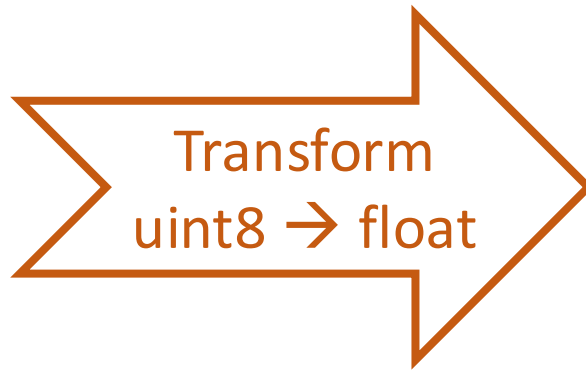
For all *processing*
stick to **float**

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



Images

Discretization – Quantize **Range**



or



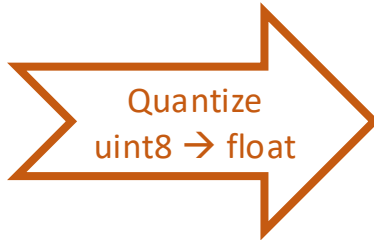
Python
Code



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

Images

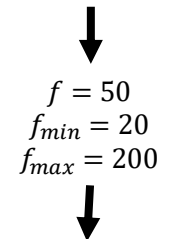
Discretization – Quantize Range



or



Numerical example:
(for 1 pixel in f)



```
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)
```



Execution priority: *Left* → *Right*
Essentially defines *implicit* parentheses!

$$g = (f - f.min()) / (f.max() - f.min()) * 255 \longrightarrow$$

$$h = 255 * (f - f.min()) / (f.max() - f.min()) \longrightarrow$$

Pixel(s) value in f

Maximum/Minimum value in f

Mathematical equivalent:

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

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

Is still uint8
Can overflow!

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

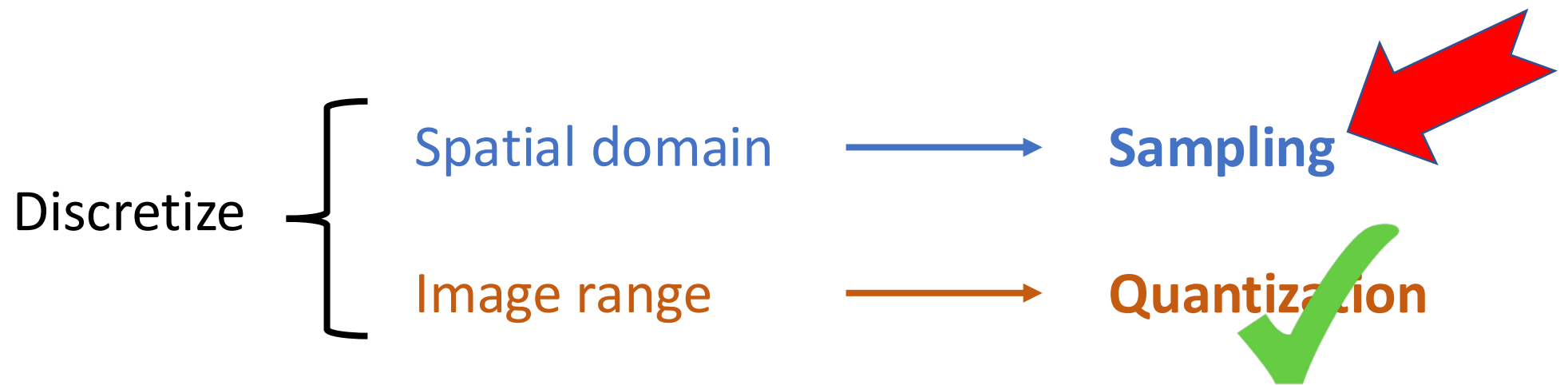
This is **226** instead of **7650**
1 byte (uint8) cannot encode numbers <0 or >255 → **Overflow!**

Images

Discretization

How can we store a **continuous grayscale image**?

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



Images

Discretization – Sample Domain

Image pixels \rightarrow apply rect. *sampling grid*
over *continuous* function f
Each box filled with *sample value*

Continuous images $\rightarrow f: \mathbb{R}^2 \rightarrow \mathbb{R}_+$
sampling \downarrow

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

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

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

Usually $\rightarrow \Delta y = \Delta x = 1$

} Sample Coord.

row
column
 (i, j)

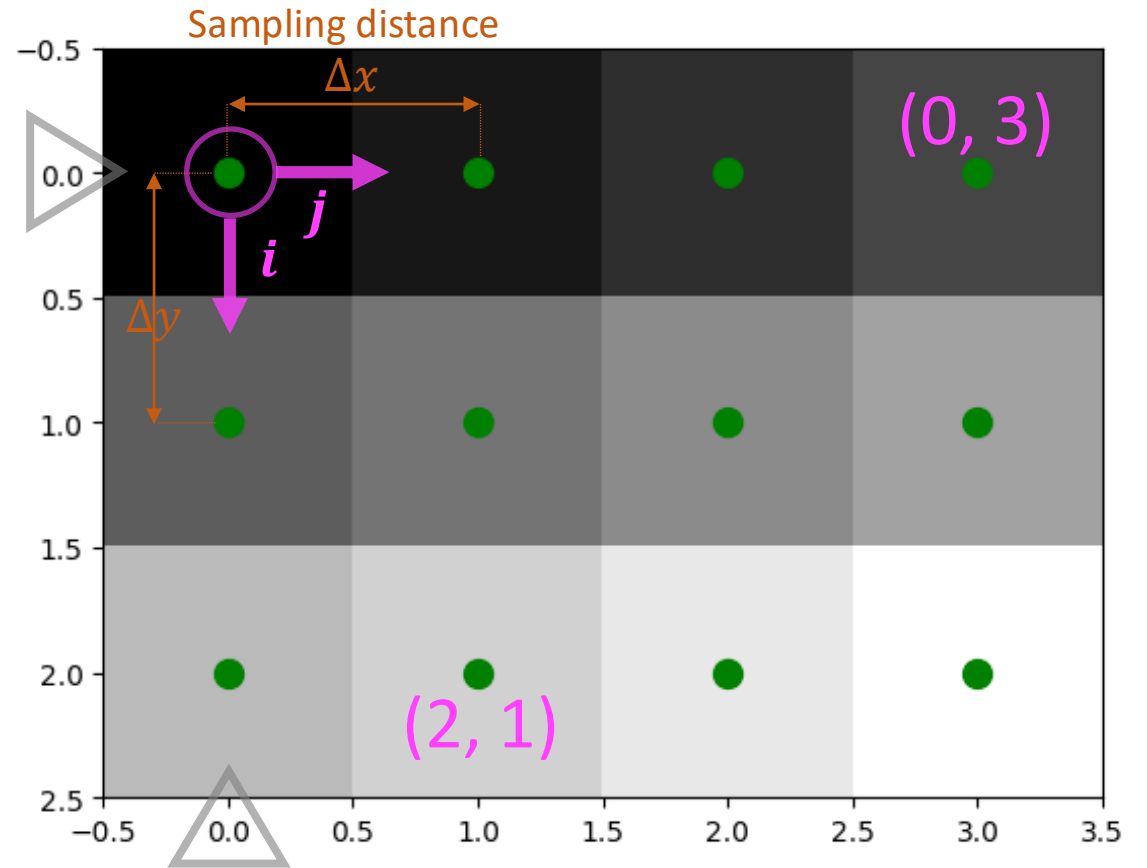


Image size \rightarrow 3×4
rows *cols*

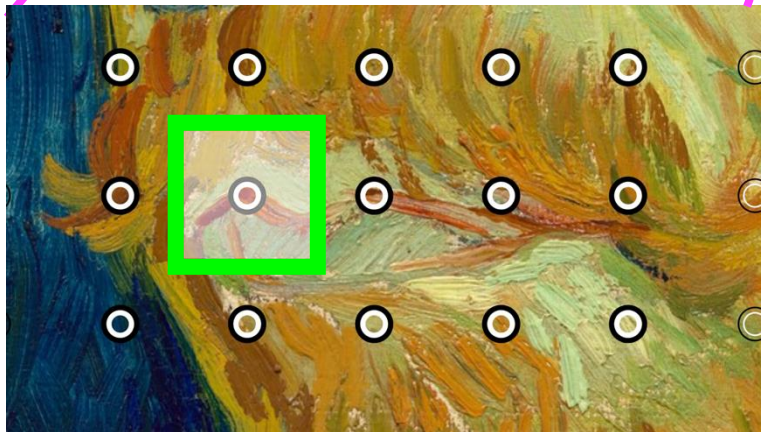
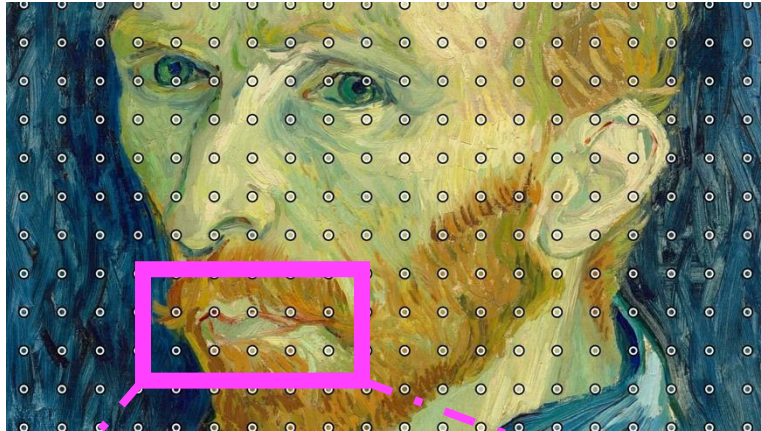
Images

Discretization – Sample Domain



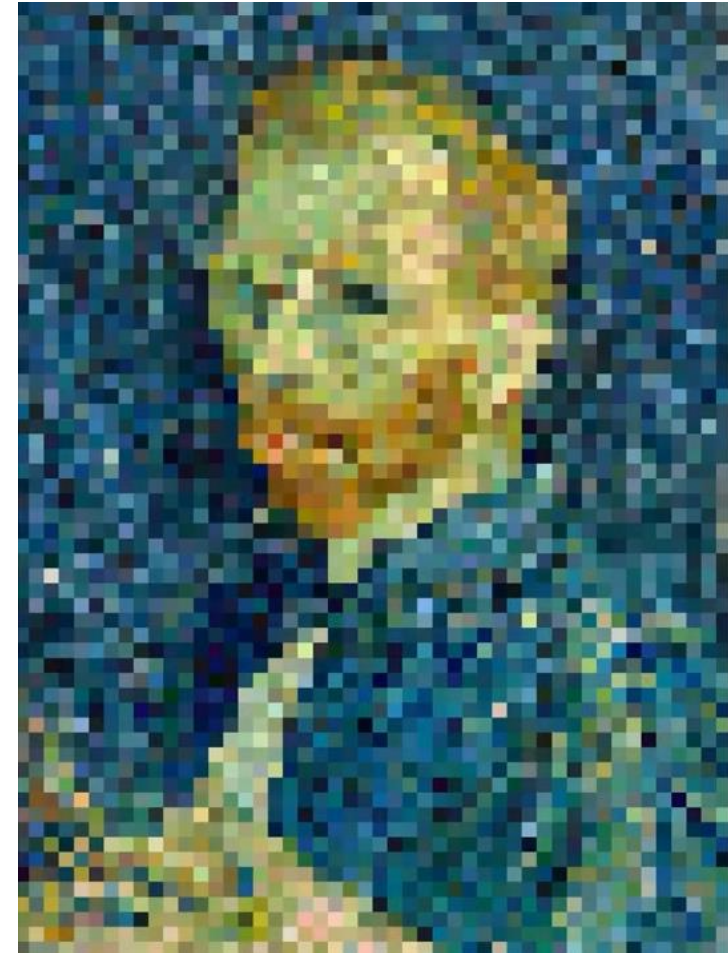
Images

Discretization – Sample Domain



'Sense' a
bigger area

Week 3
(Local Operators)



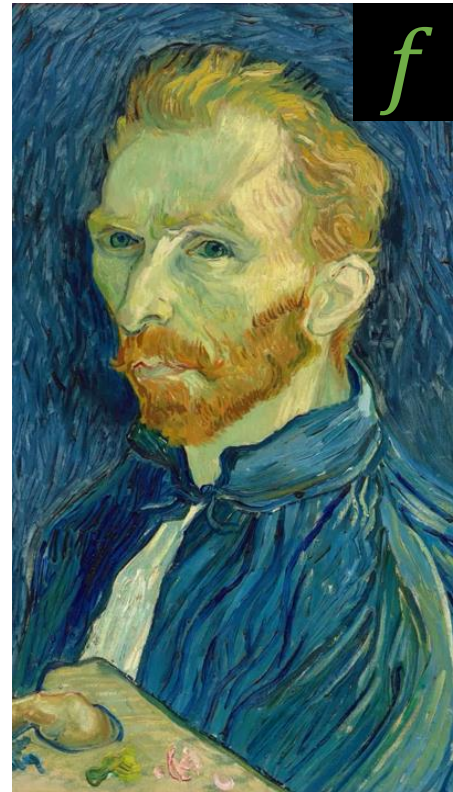
Images

Interpolation

Given samples F of continuous f

Interpolation \rightarrow Calculate value $f(x, y)$
between samples $F(i, j)$

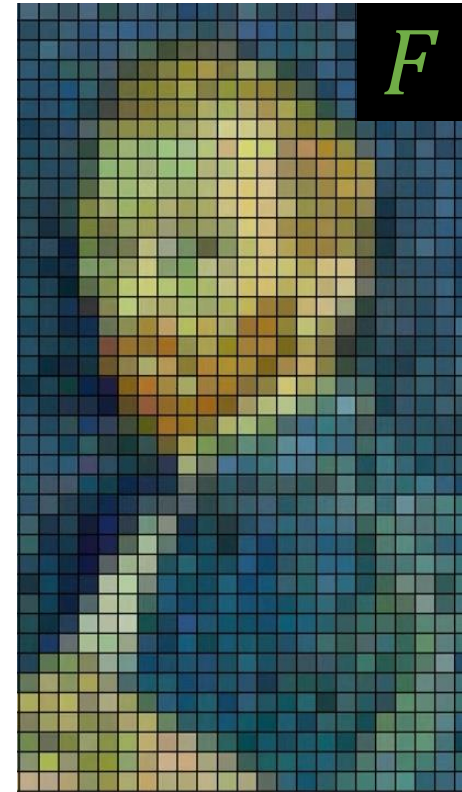
Assuming: Well-sampled image
(if curious – Information Theory:
Nyquist-Shannon sampling theorem)



Discretization

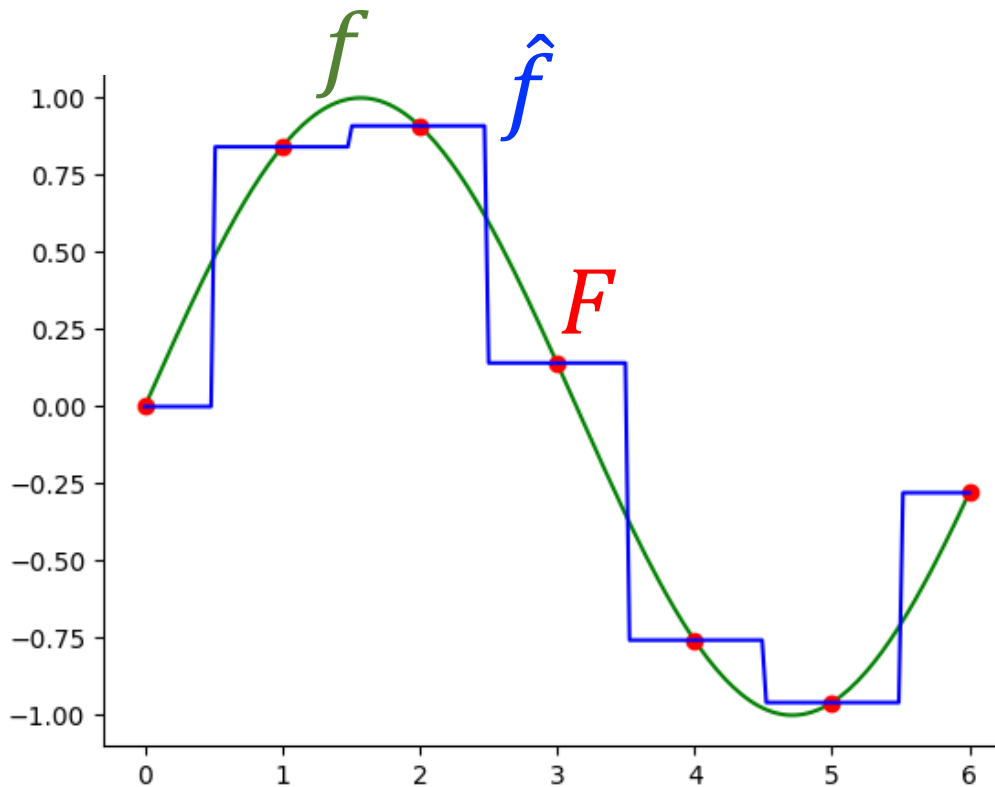


Interpolation



Images

1D Interpolation - Goal



$F(x), x \in \mathbb{Z}$

→ Input: Samples of f

$f(x) = \sin(x), x \in \mathbb{R}$

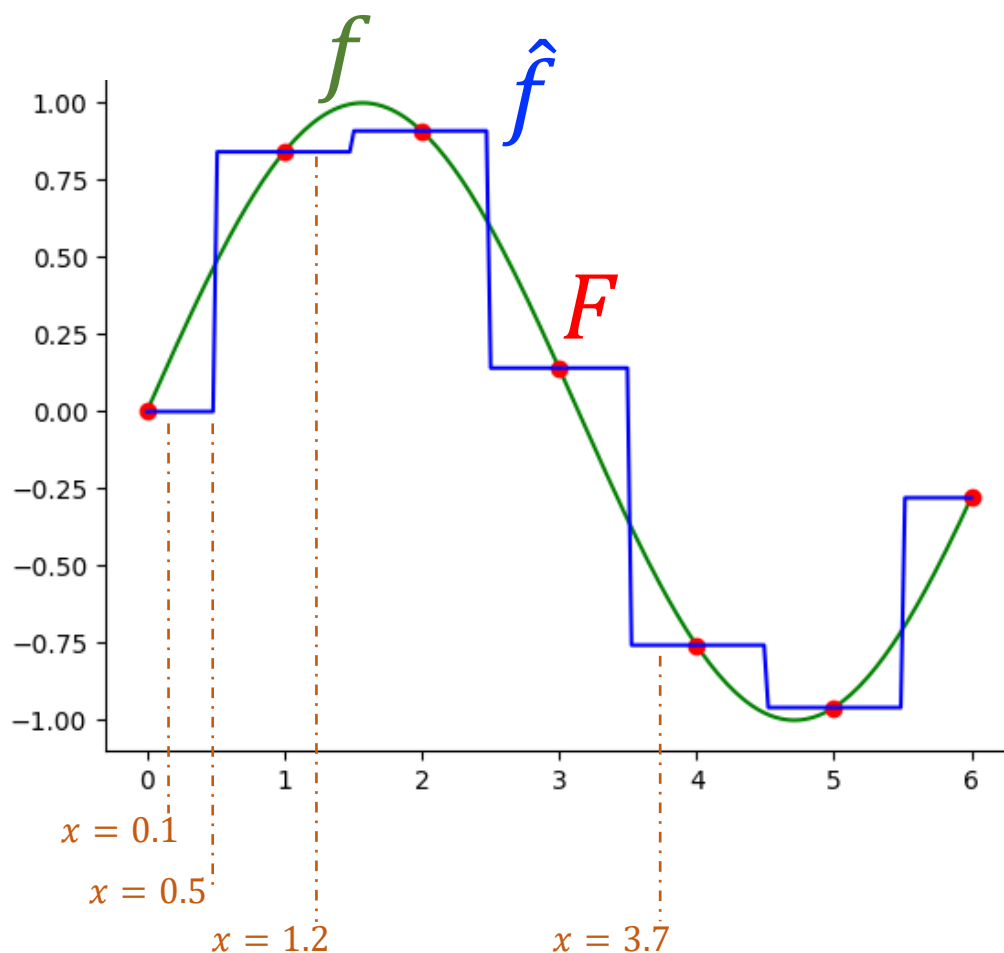
→ Continuous 'target' function that we want to **guesstimate**

Goal:

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

Images

1D Interpolation - Nearest Neighbor



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

programmer's view: $\text{round}(x)$

'Floor' function
 $\lfloor x \rfloor \rightarrow$ largest int that is $\leq x$

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

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

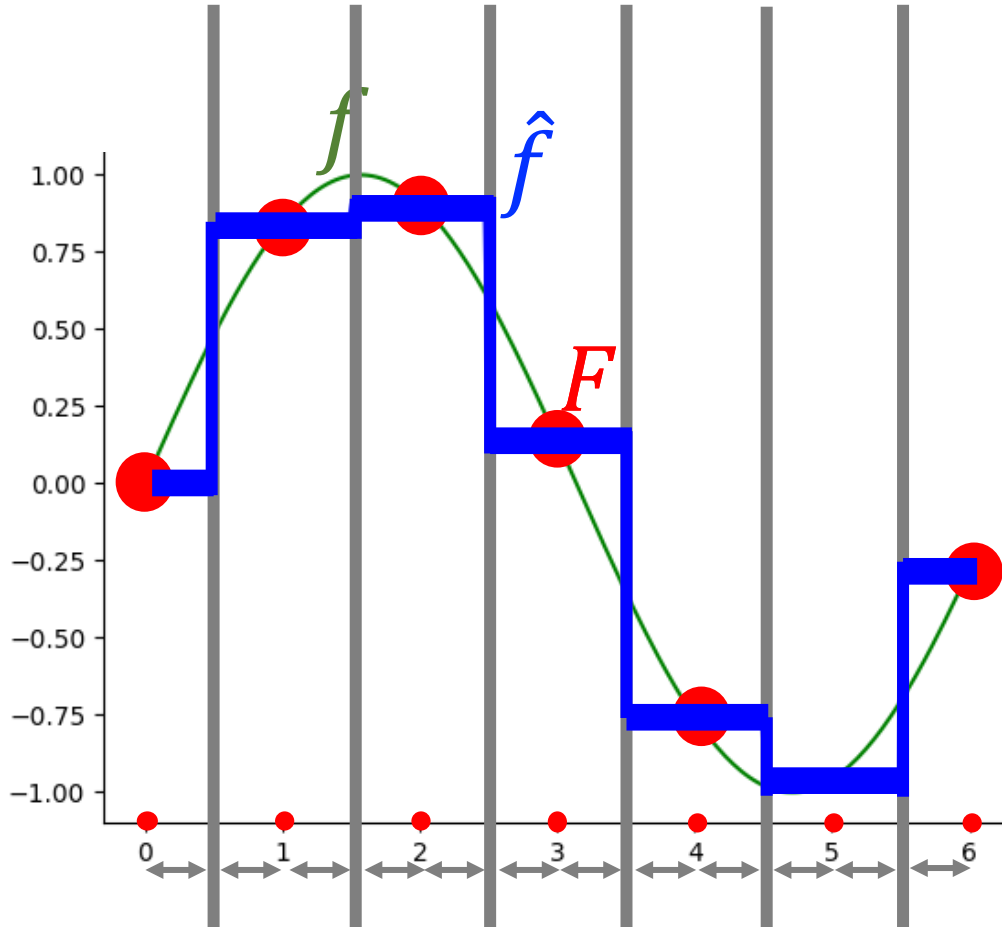
$$\begin{aligned} \hat{f}(1.2) &= F(\lfloor 1.2 + 0.5 \rfloor) \\ &= F(\lfloor 1.7 \rfloor) \\ &= F(1) \end{aligned}$$

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

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

Images

1D Interpolation - Nearest Neighbor

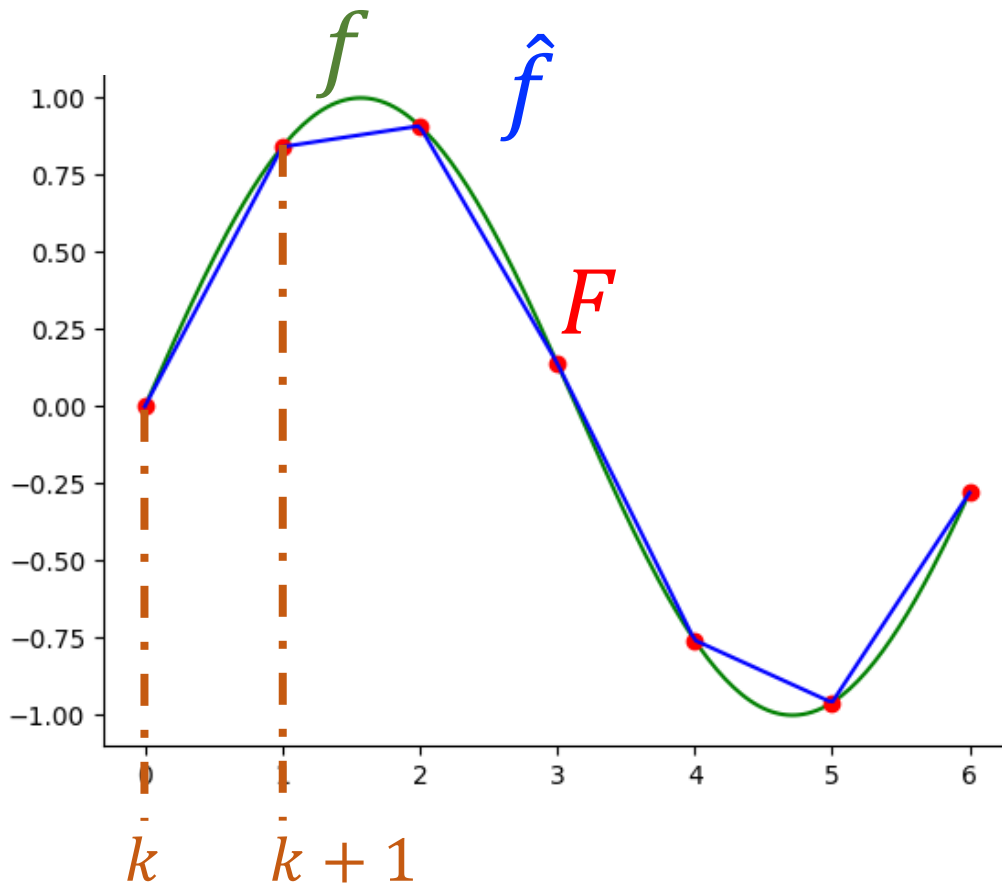


$$\hat{f}(x) = F(\lfloor x + 0.5 \rfloor)$$

$\hat{f} \rightarrow$ 'Staircase' Continuous Differentiable

Images

1D Interpolation - Linear



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

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

$$\hat{f}(x) = \underbrace{(1 - (x - k))}_{\text{steering / mixing weights}} \underbrace{F(k)}_{\text{samples}} + \underbrace{(x - k)}_{\text{steering / mixing weights}} \underbrace{F(k + 1)}_{\text{samples}}$$

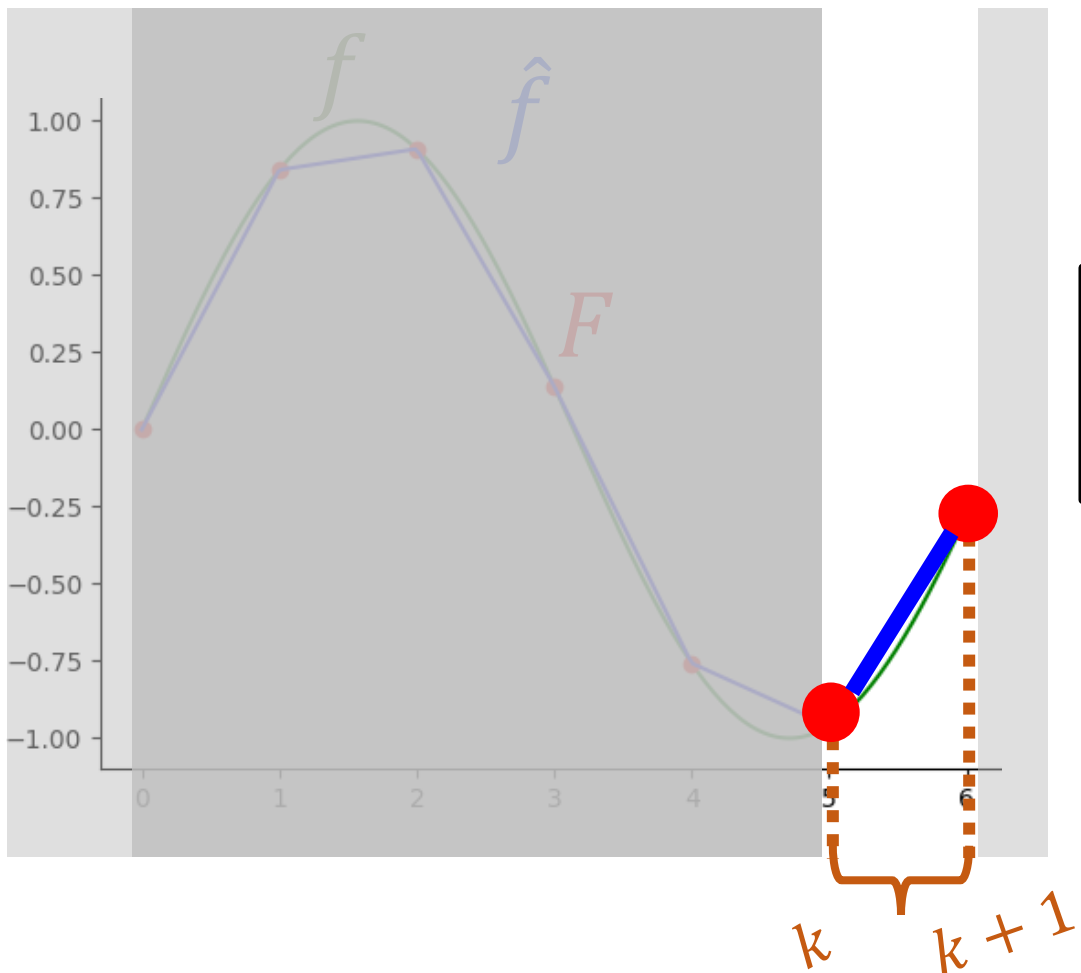
steering / mixing
weights

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: $F(k)$ & $F(k + 1)$

$$\hat{f}(x) = \underbrace{(1 - (x - k))}_{\text{steering / mixing weights}} \underbrace{F(k)}_{\text{samples}} + \underbrace{(x - k)}_{\text{steering / mixing weights}} \underbrace{F(k + 1)}_{\text{samples}}$$

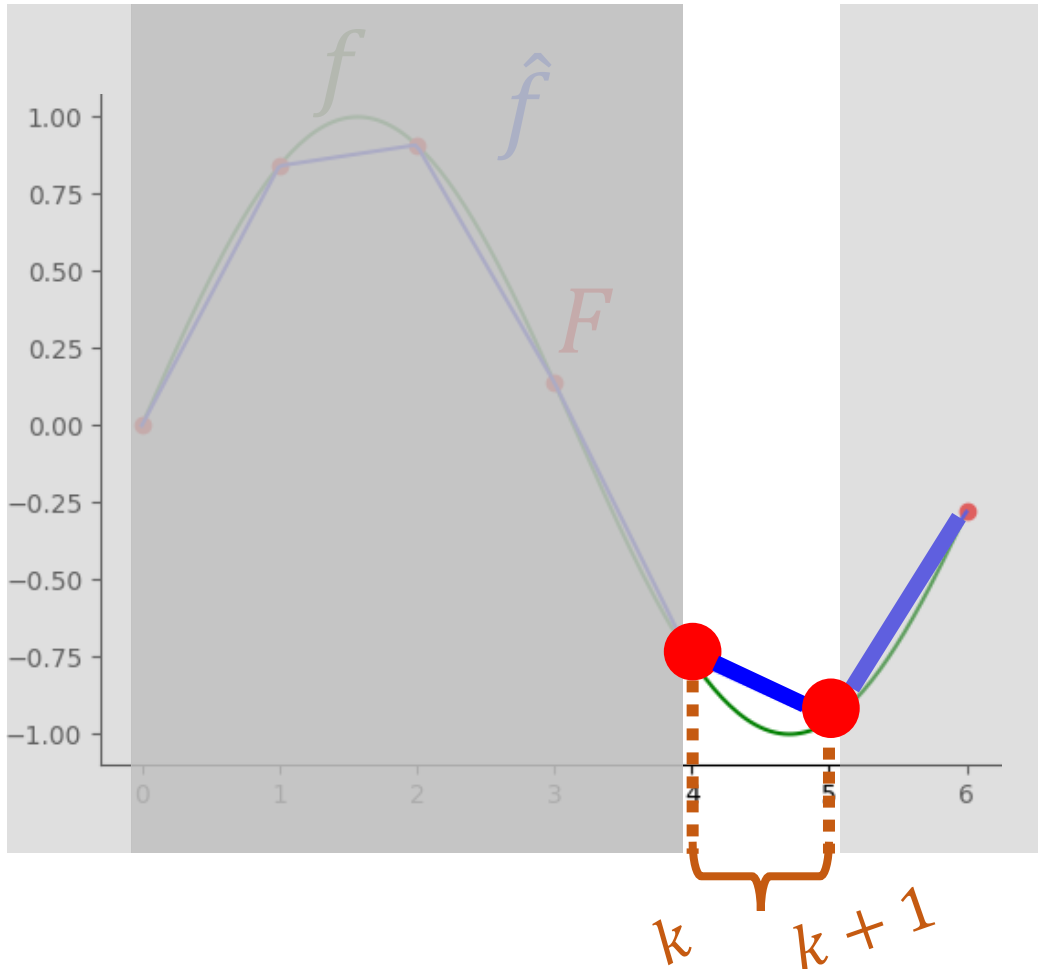
steering / mixing
weights

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: $F(k)$ & $F(k + 1)$

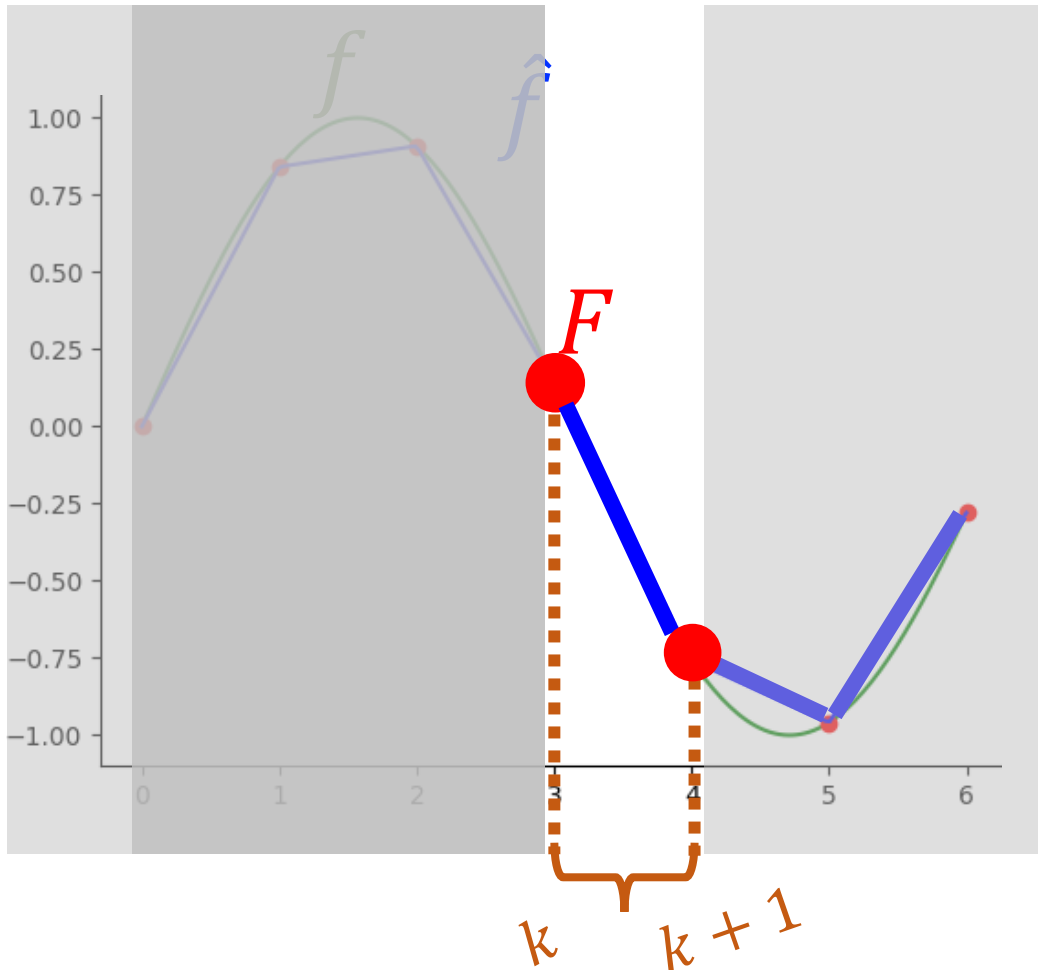
$$\hat{f}(x) = \underbrace{(1 - (x - k))}_{\text{steering / mixing weights}} \underbrace{F(k)}_{\text{samples}} + \underbrace{(x - k)}_{\text{steering / mixing weights}} \underbrace{F(k + 1)}_{\text{samples}}$$

steering / mixing
weights

samples

Images

1D Interpolation - Linear



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

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

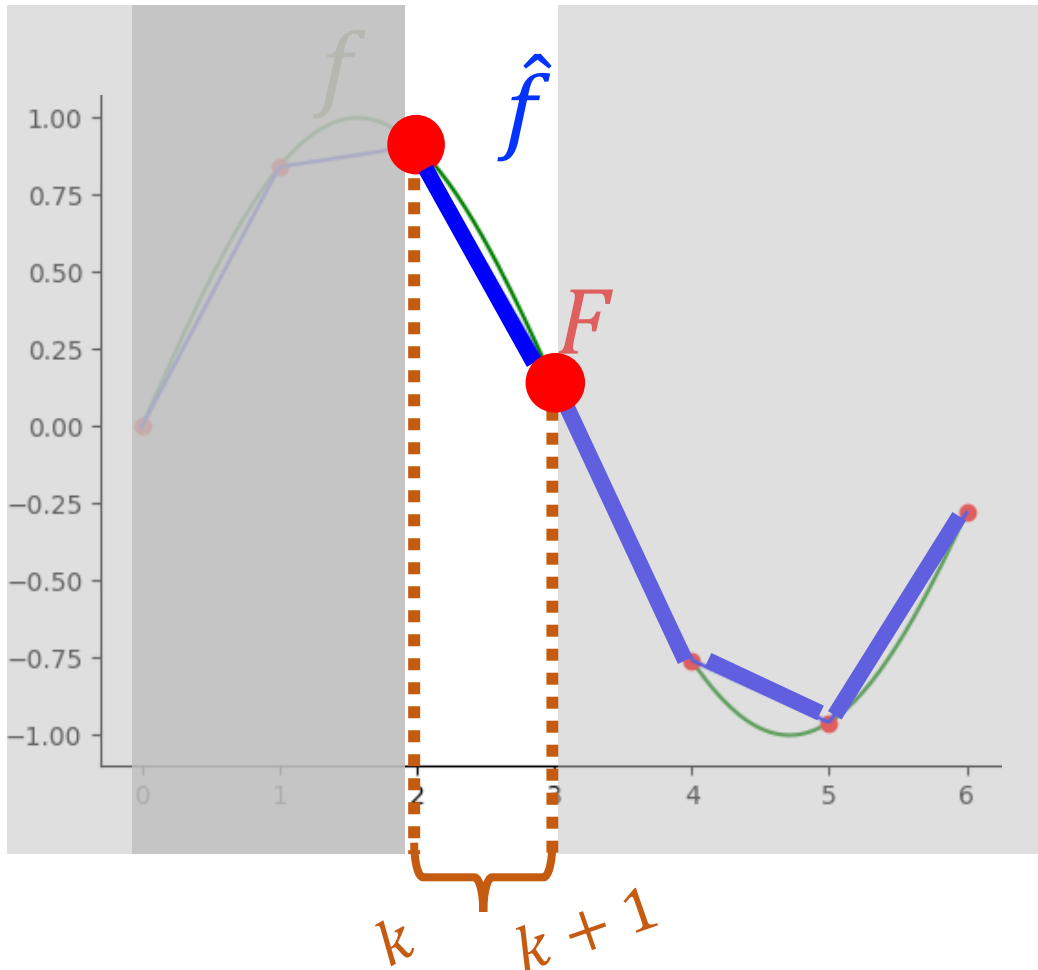
$$\hat{f}(x) = \underbrace{(1 - (x - k))}_{\text{steering / mixing weights}} \underbrace{F(k)}_{\text{samples}} + \underbrace{(x - k)}_{\text{steering / mixing weights}} \underbrace{F(k + 1)}_{\text{samples}}$$

steering / mixing
weights

samples

Images

1D Interpolation - Linear



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

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

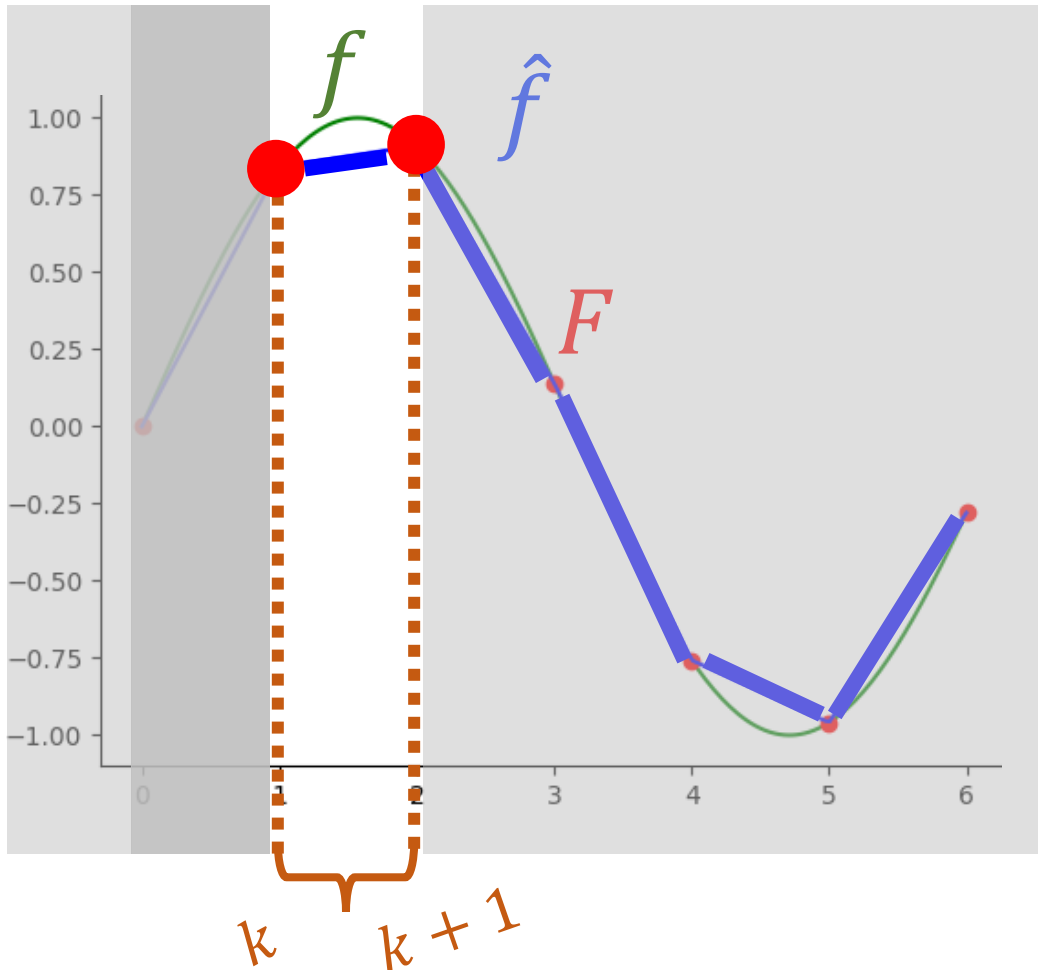
$$\hat{f}(x) = \underbrace{(1 - (x - k))}_{\text{steering / mixing weights}} \underbrace{F(k)}_{\text{samples}} + \underbrace{(x - k)}_{\text{steering / mixing weights}} \underbrace{F(k + 1)}_{\text{samples}}$$

steering / mixing
weights

samples

Images

1D Interpolation - Linear



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

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

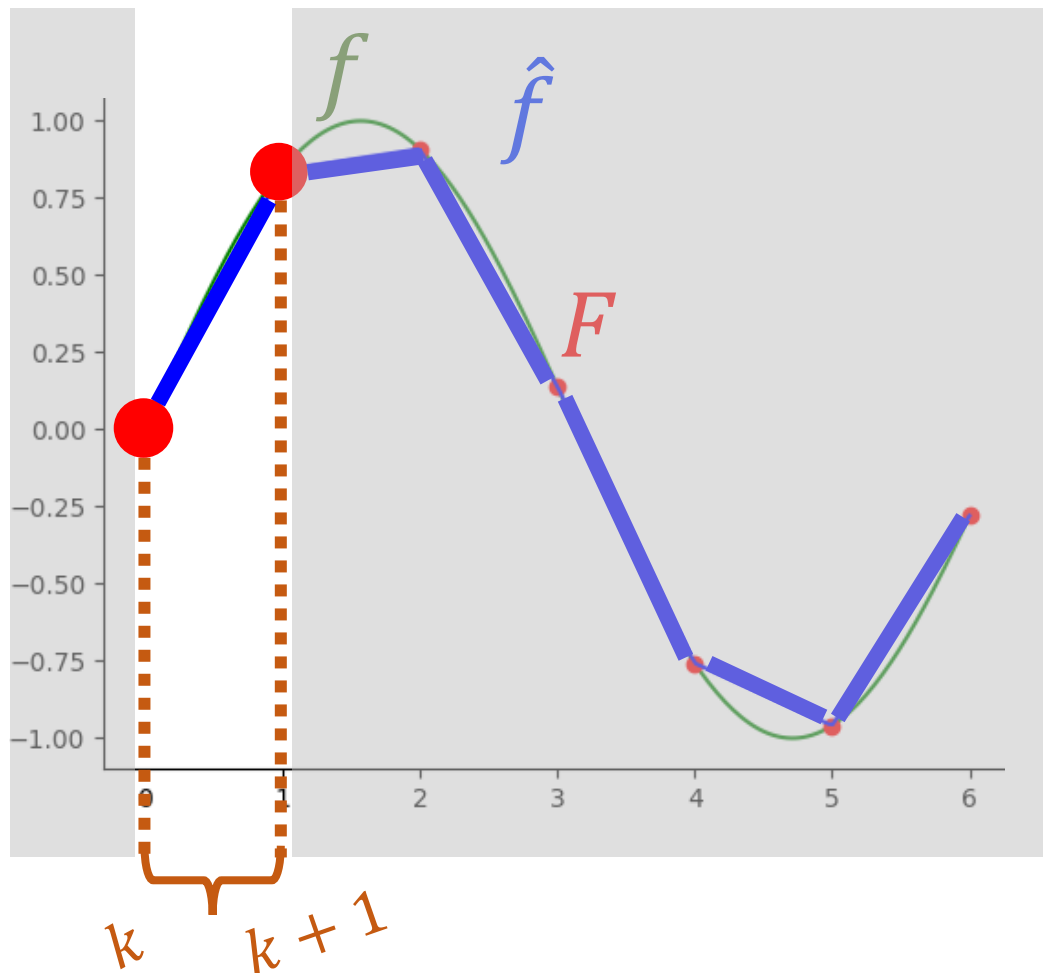
$$\hat{f}(x) = \underbrace{(1 - (x - k))}_{\text{steering / mixing weights}} \underbrace{F(k)}_{\text{samples}} + \underbrace{(x - k)}_{\text{steering / mixing weights}} \underbrace{F(k + 1)}_{\text{samples}}$$

steering / mixing
weights

samples

Images

1D Interpolation - Linear



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

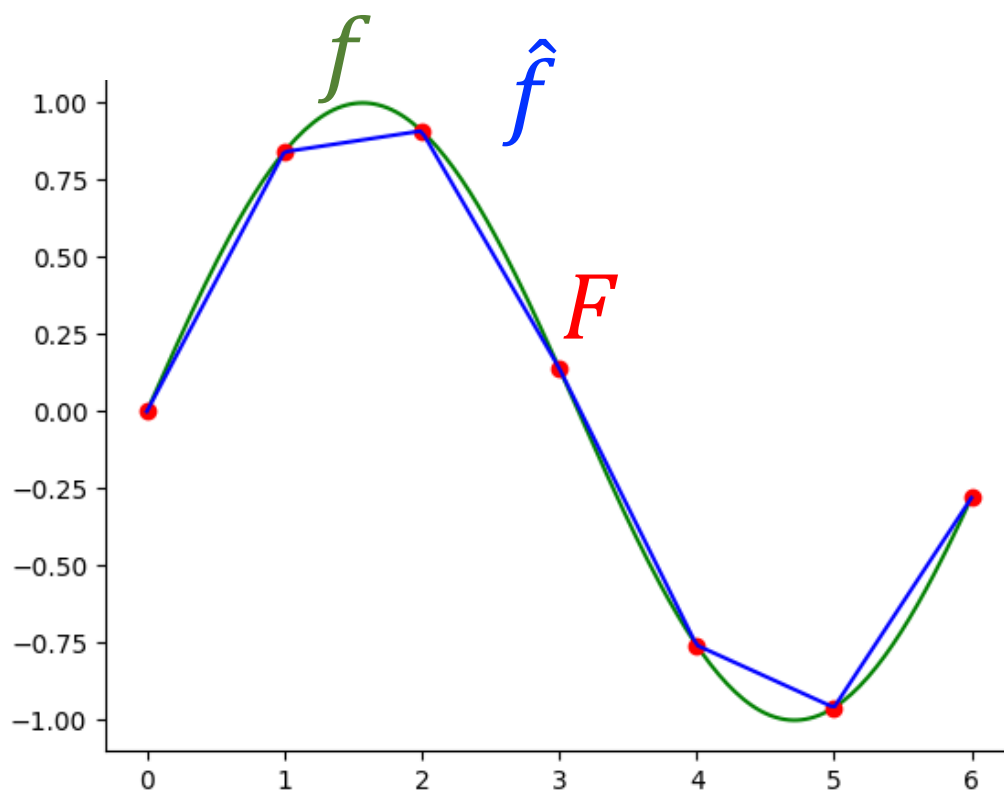
\hat{f} linearly 'mixes'/connects: $F(k)$ & $F(k + 1)$

$$\hat{f}(x) = \underbrace{(1 - (x - k))}_{\text{steering / mixing weights}} \underbrace{F(k)}_{\text{samples}} + \underbrace{(x - k)}_{\text{steering / mixing weights}} \underbrace{F(k + 1)}_{\text{samples}}$$

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

Images

1D Interpolation - Linear



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

$\hat{f} \rightarrow$

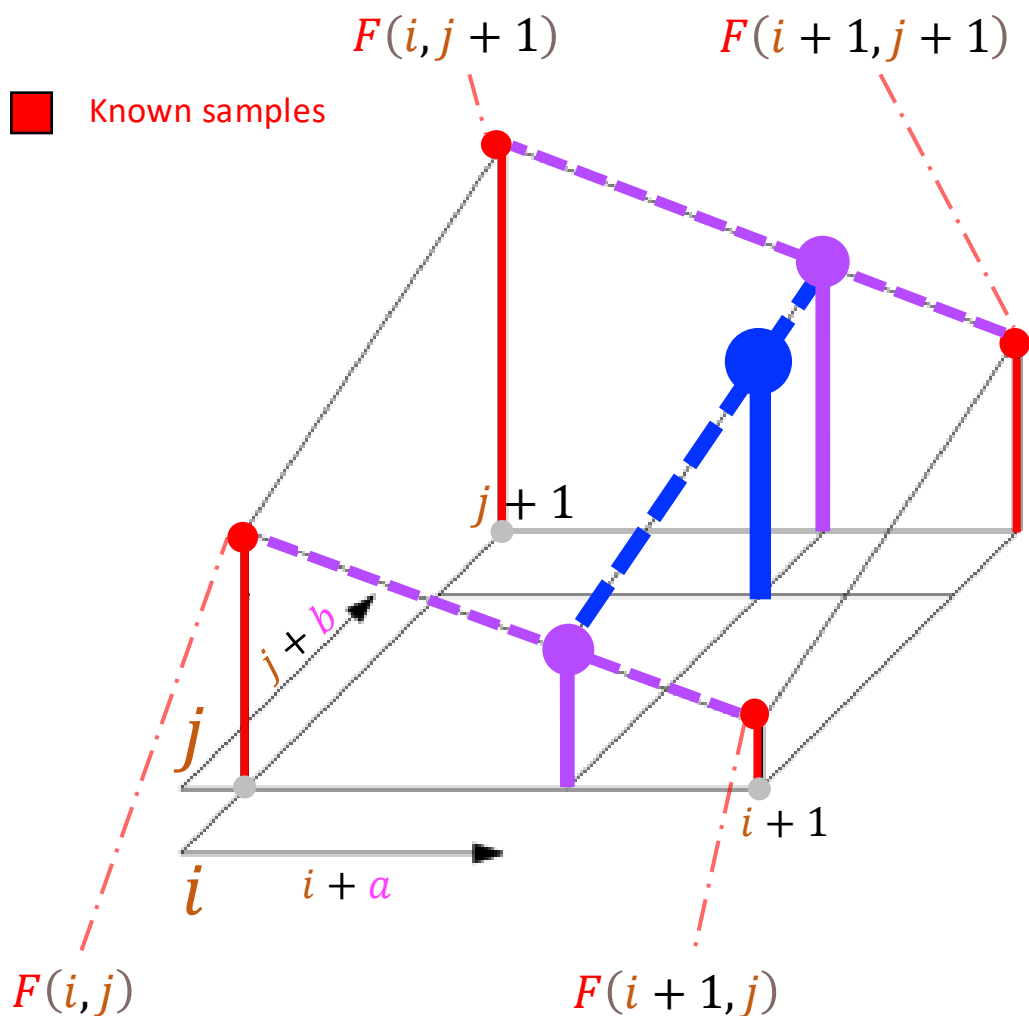
Piece-wise
linear

Continuous

- not differentiable @ sample point locations
- high-order deriv. = 0

Images

2D Interpolation - Bilinear



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

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

2-step process:

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

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

$l = \text{fixed}$

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

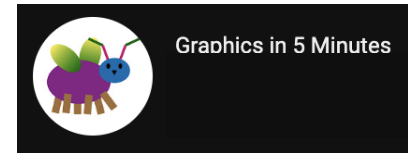
$$f(x, y) = f(i + a, j + b) = (1 - a)(1 - b) F(i, j) + (1 - a) b F(i, j + 1) + a (1 - b) F(i + 1, j) + a b \underbrace{F(i + 1, j + 1)}_{\text{Known samples!}}$$

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 ←