

Computer Vision

What is an image?

Slides Credit : James Tompkin

What is an image?

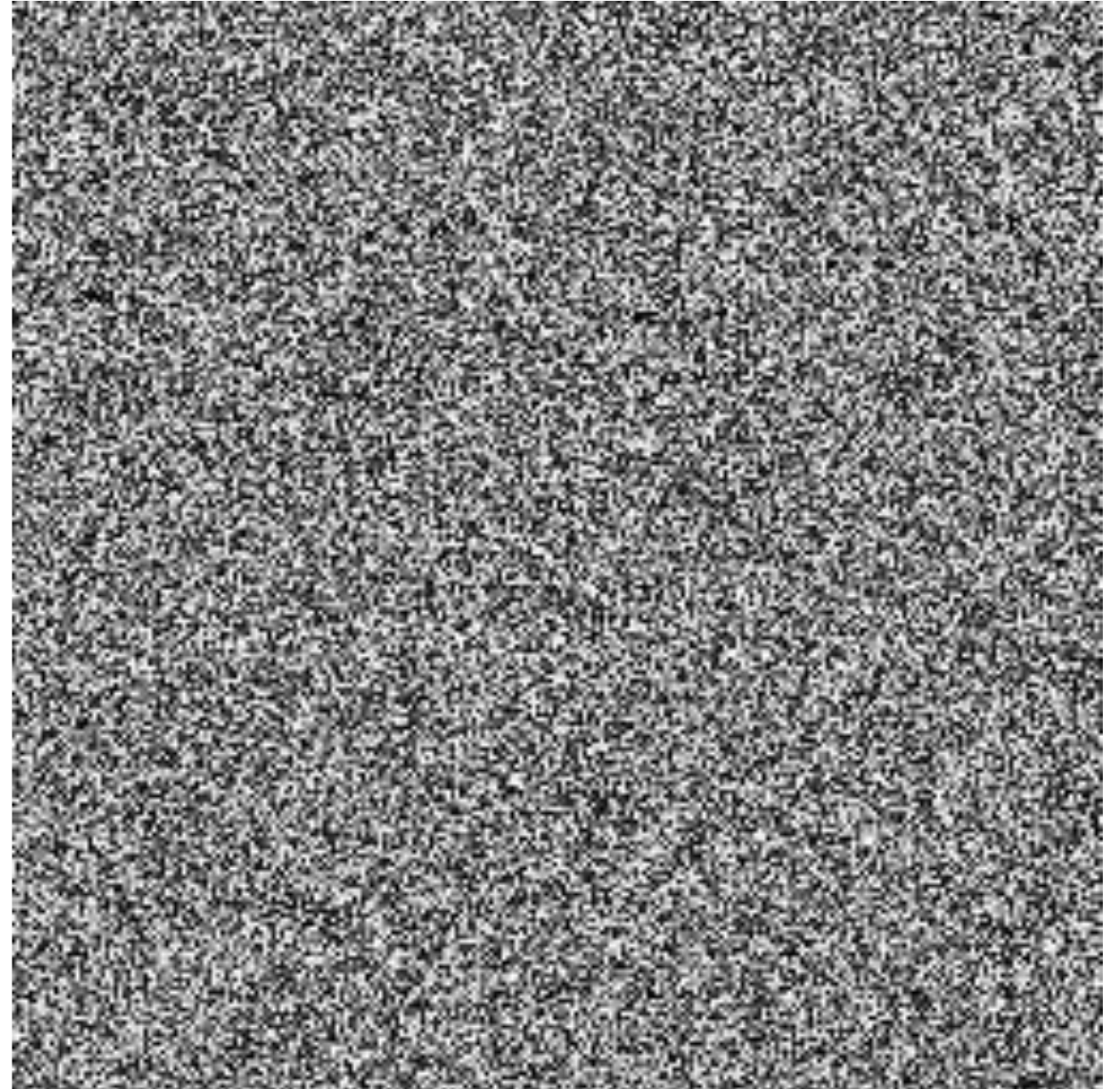
- Image
- Sampling
- Digital image representation
- Pixels and pixel manipulation
- Images in practice



What is an image?

```
>>> from numpy import random as r  
>>> I = r.rand(256,256)
```

- What is this? What does it look like?
- Which values does it have?
- How many are there?



Dimensionality of an image

- 256x256 pixels @ 8bit = 256 value ⁶⁵⁵³⁶
 - There is absolutely no way to fit this into any memory
- Computer vision -> making sense of an extremely high-dimensional space.
 - Subspace of 'natural' images.
 - Deriving low-dimensional, explainable model.

Elements of a Digital Image

Pixel: picture element

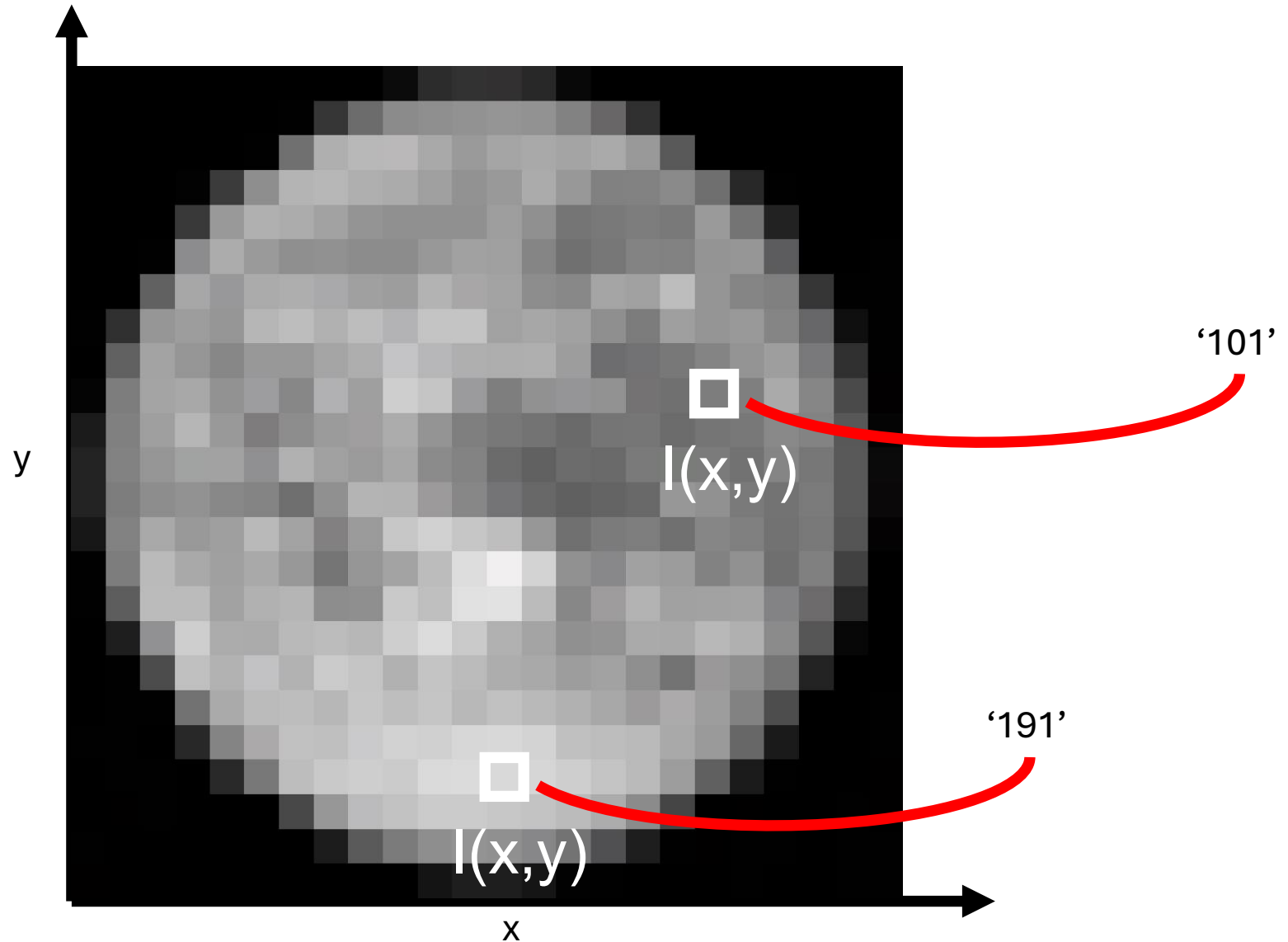


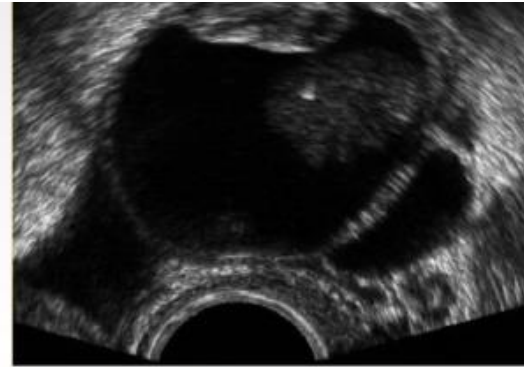
Image as a 2D sampling of signal

- Signal: function depending on some variable with physical meaning.
- Image: sampling of that function.
 - 2 variables: x, y coordinates.
 - 3 variables: x, y + time (video)
 - 'Brightness' is the value of the function for visible light.
- Other physical values too: temperature, pressure, depth,...

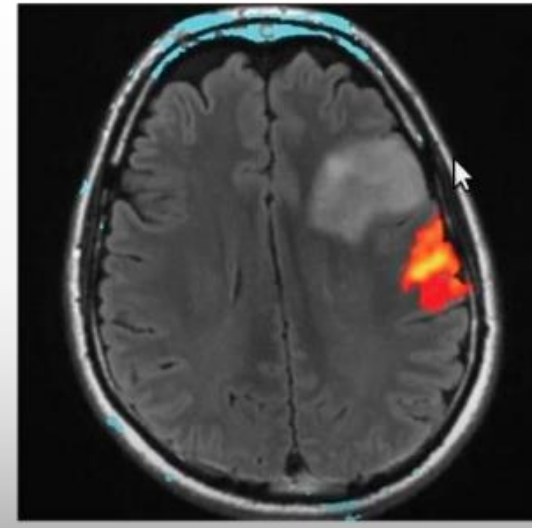
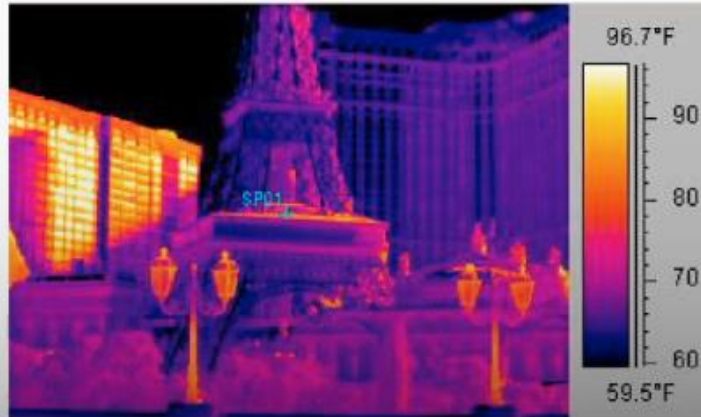
Examples of 2D images



(a) Natural Image

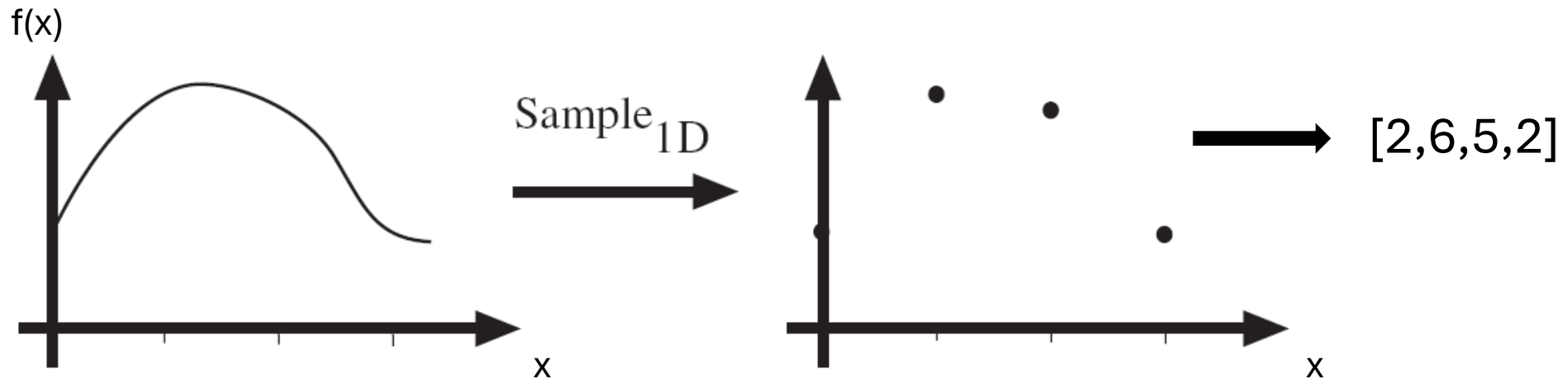


(c) Ultra Sound Image

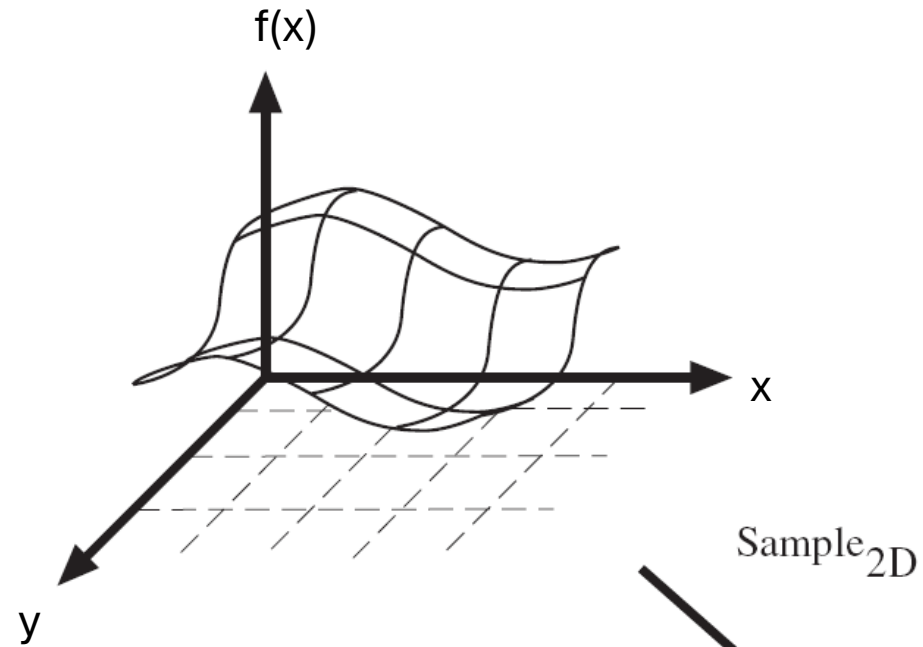


Sampling

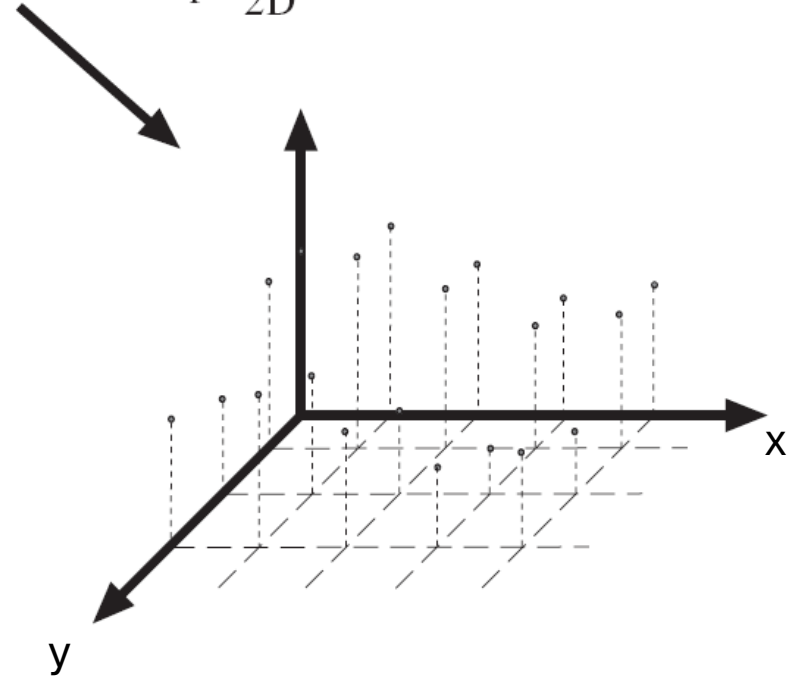
Sampling in 1D takes a function and returns a vector whose elements are values of that function at the sample points.



Sampling

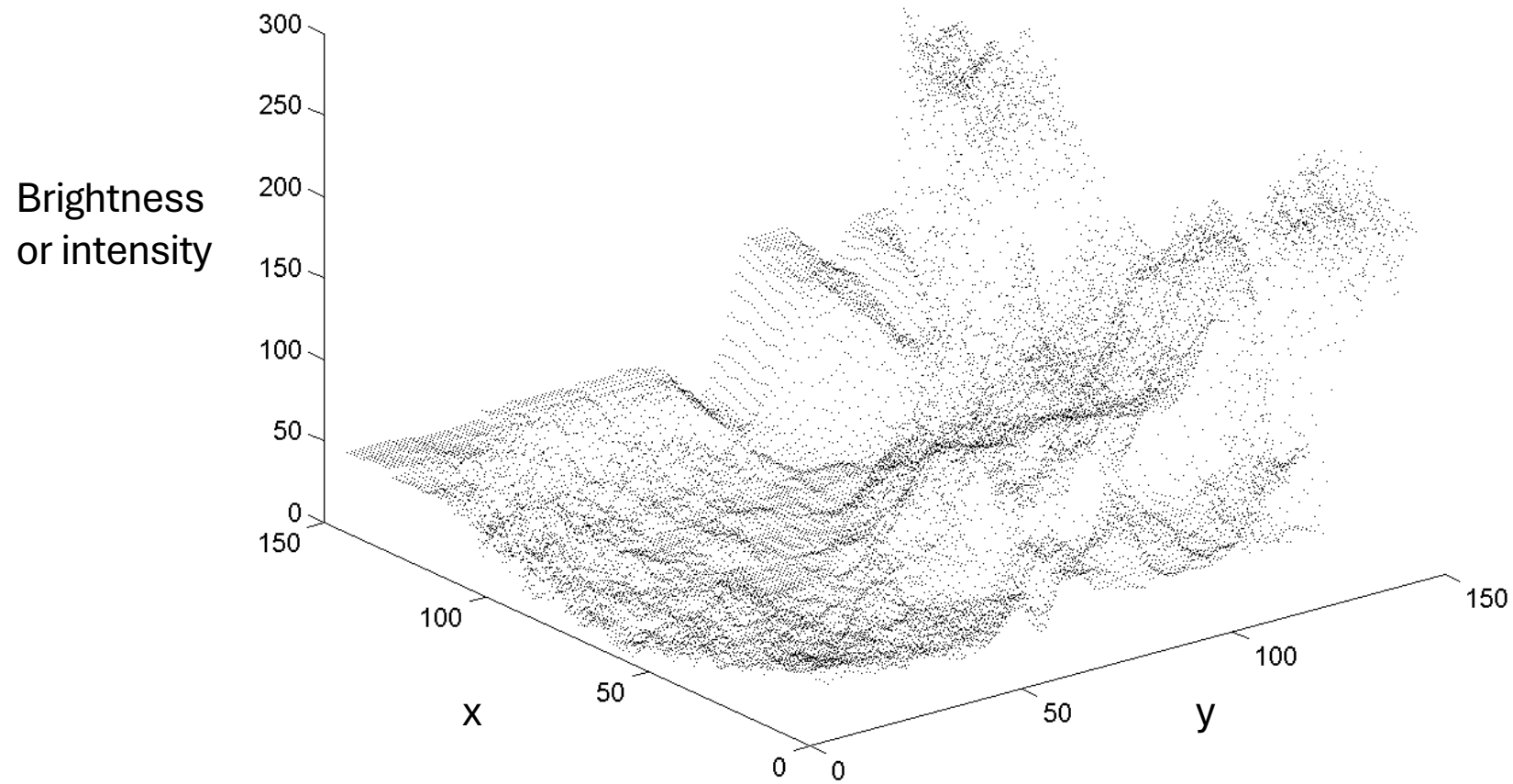


Sample_{2D}

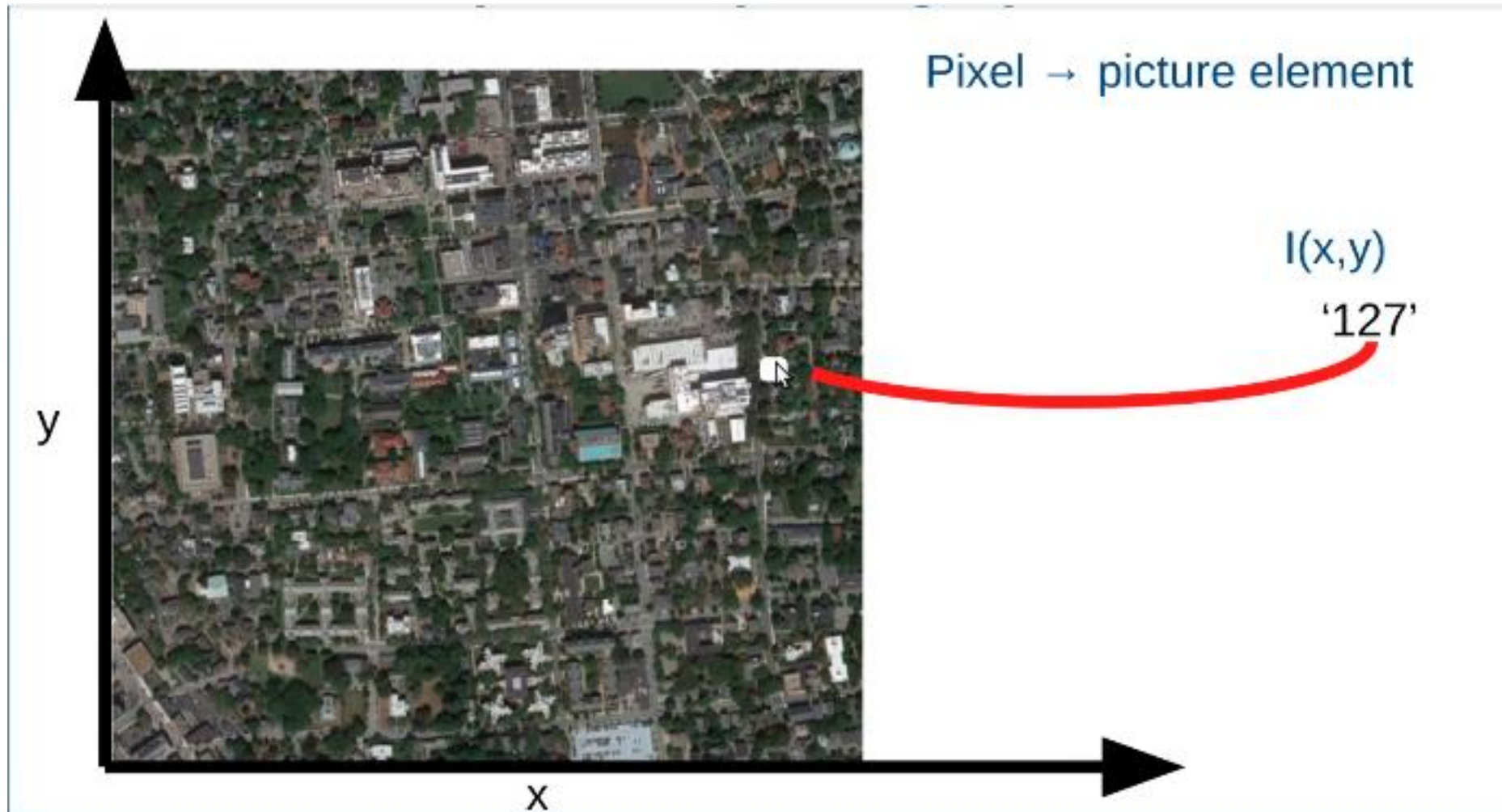


Sampling in 2D takes a function and returns a matrix.

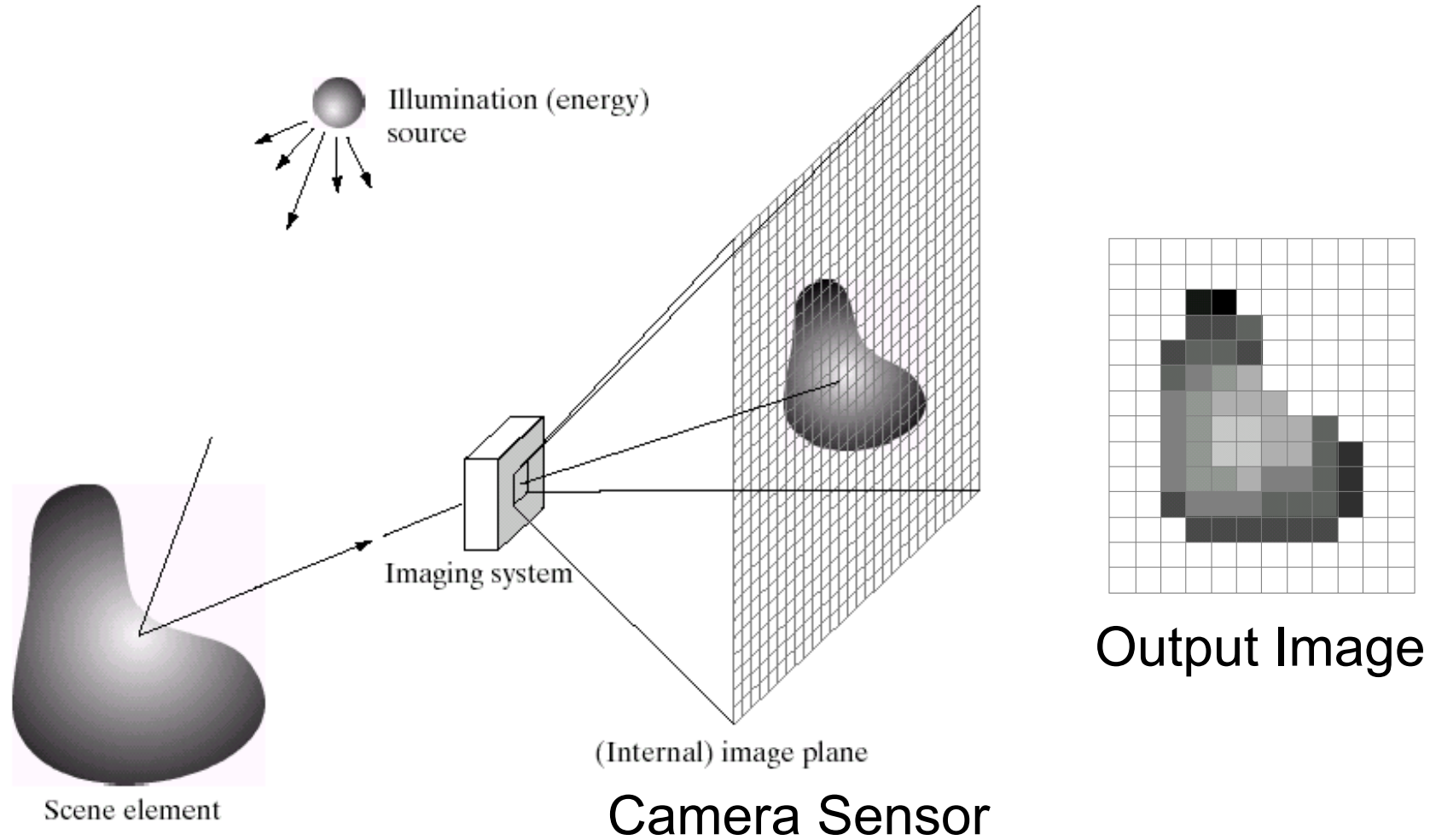
Digital Image as 'Height Map'



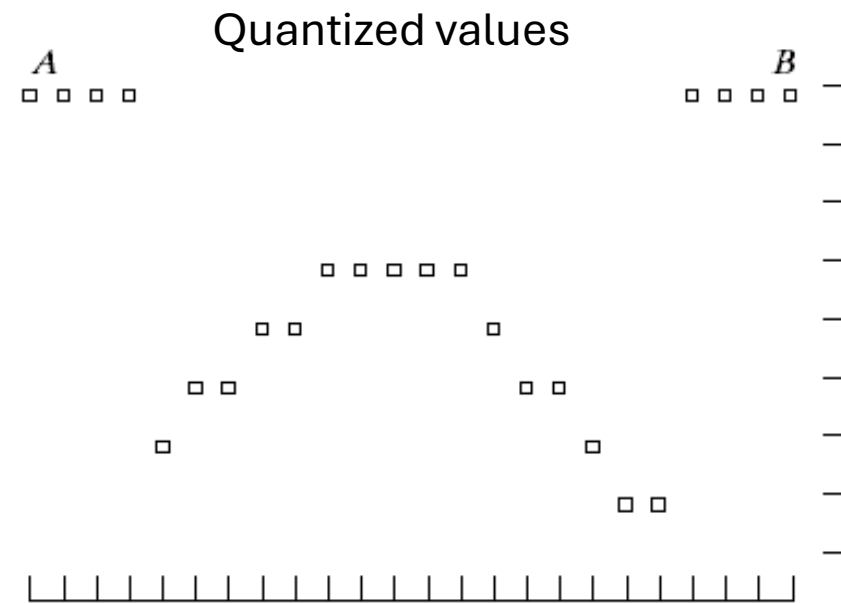
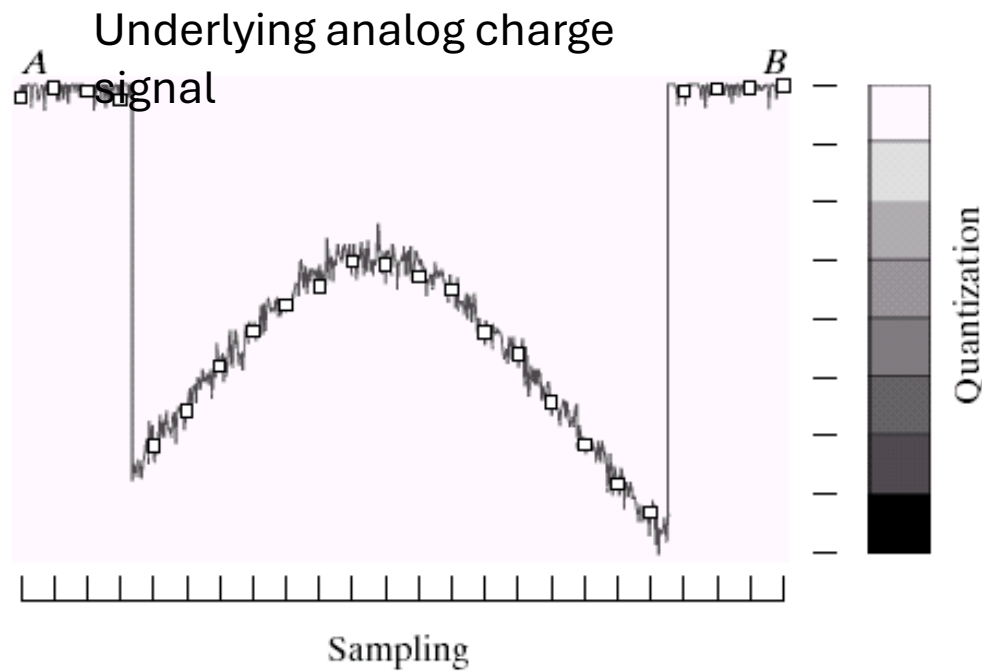
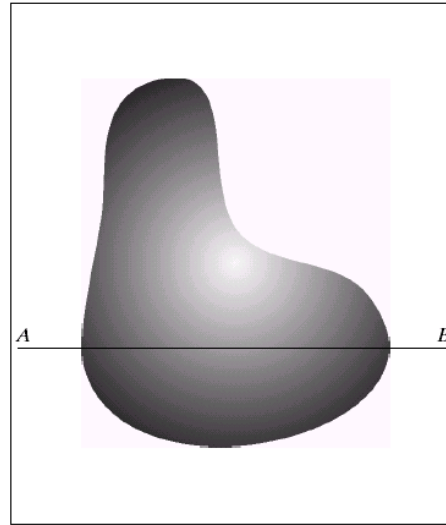
What is each part of a ptotograph?



Light Integration Over the *Frustum*



Quantization



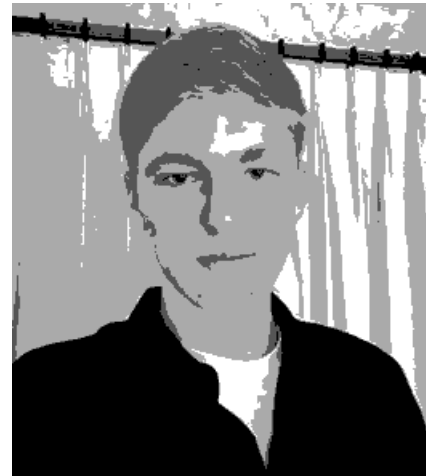
Quantization Effects – Radiometric Resolution



8 bit – 256 levels



4 bit – 16 levels



2 bit – 4 levels

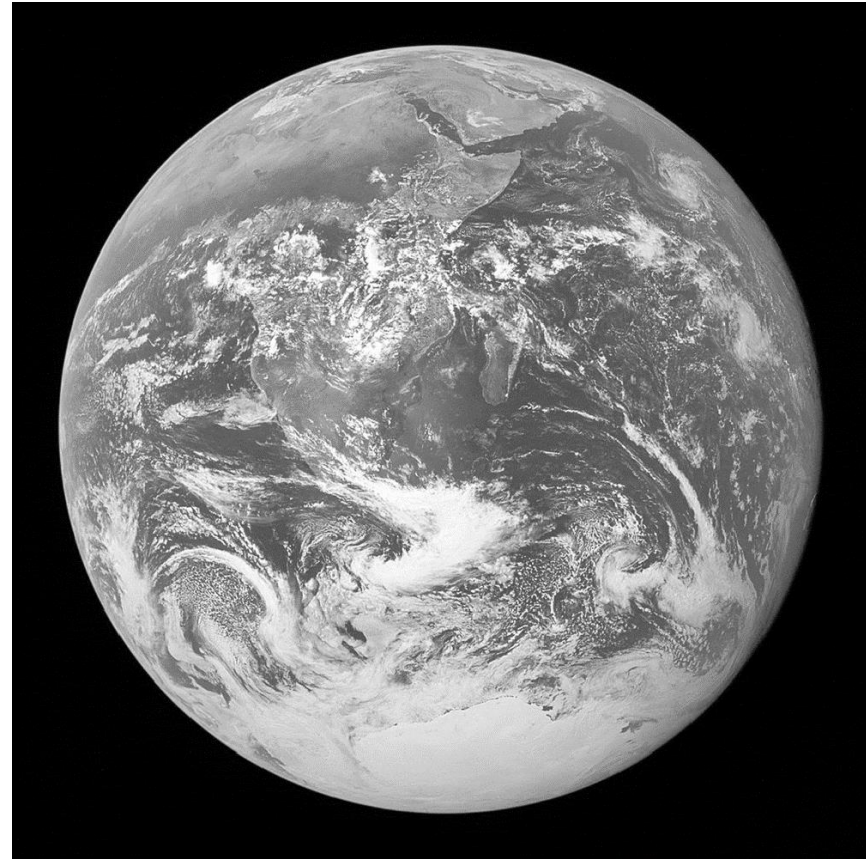
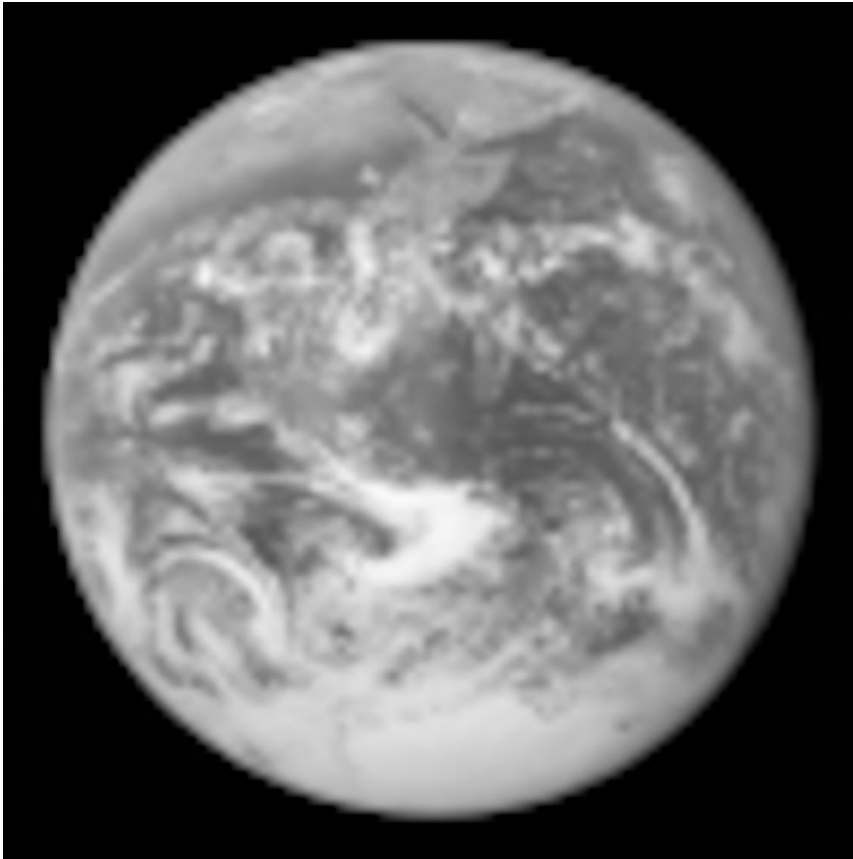


1 bit – 2 levels

We often call this *bit depth*.
For photography, this is also related to *dynamic range*.

Resolution: geometric vs. spatial

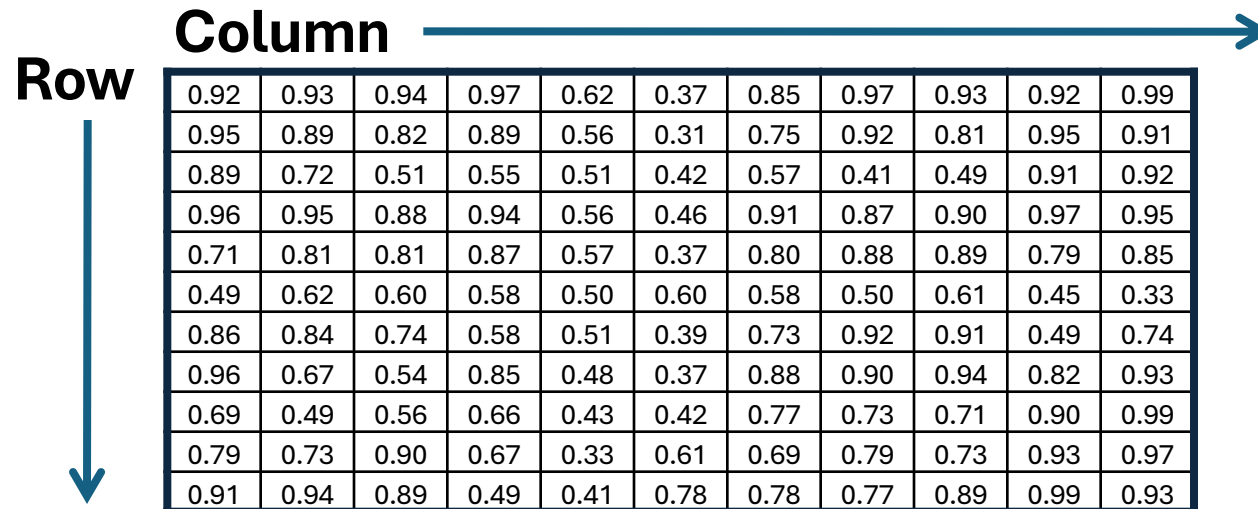
Both images are 1000x1000 pixels



Images in Python (import numpy)

$N \times M$ grayscale image “im”

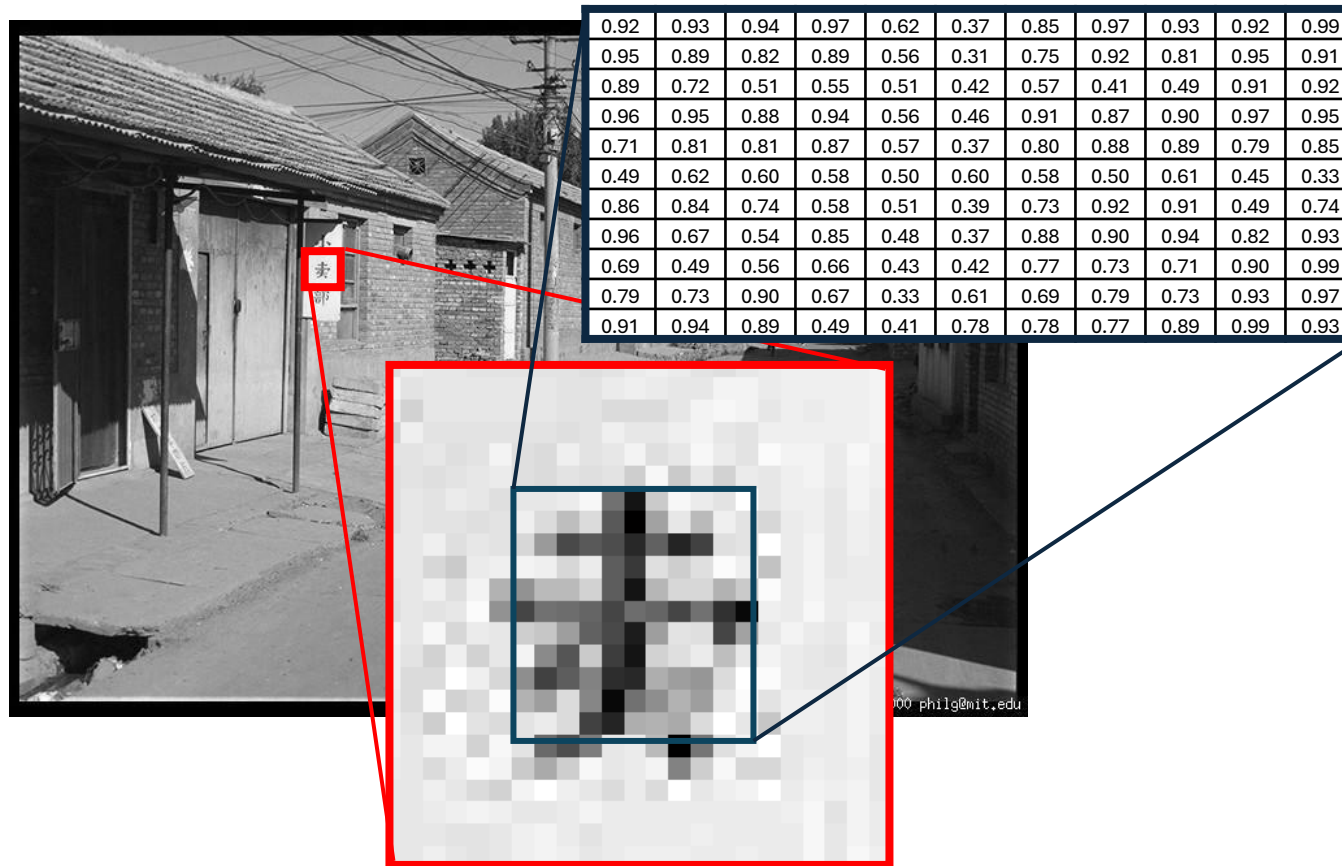
- `im[0,0]` = top-left pixel value
- `im[y,x]` = y pixels down, x pixels to right
- `im[N-1,M-1]` = bottom-right pixel



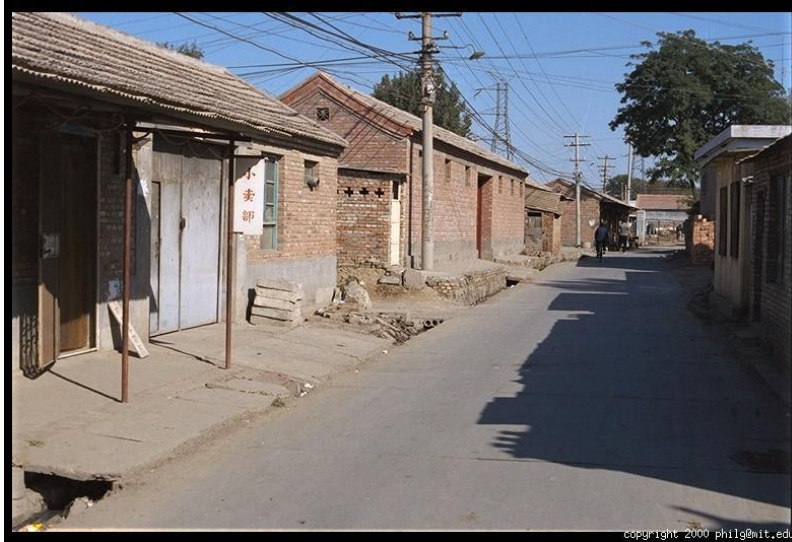
The diagram shows a 10x10 matrix representing a grayscale image. A vertical blue arrow on the left points downwards, labeled "Row". A horizontal blue arrow on the top points to the right, labeled "Column". The matrix contains numerical values for each pixel.

0.92	0.93	0.94	0.97	0.62	0.37	0.85	0.97	0.93	0.92	0.99
0.95	0.89	0.82	0.89	0.56	0.31	0.75	0.92	0.81	0.95	0.91
0.89	0.72	0.51	0.55	0.51	0.42	0.57	0.41	0.49	0.91	0.92
0.96	0.95	0.88	0.94	0.56	0.46	0.91	0.87	0.90	0.97	0.95
0.71	0.81	0.81	0.87	0.57	0.37	0.80	0.88	0.89	0.79	0.85
0.49	0.62	0.60	0.58	0.50	0.60	0.58	0.50	0.61	0.45	0.33
0.86	0.84	0.74	0.58	0.51	0.39	0.73	0.92	0.91	0.49	0.74
0.96	0.67	0.54	0.85	0.48	0.37	0.88	0.90	0.94	0.82	0.93
0.69	0.49	0.56	0.66	0.43	0.42	0.77	0.73	0.71	0.90	0.99
0.79	0.73	0.90	0.67	0.33	0.61	0.69	0.79	0.73	0.93	0.97
0.91	0.94	0.89	0.49	0.41	0.78	0.78	0.77	0.89	0.99	0.93

Grayscale Intensity



Color



Red intensity



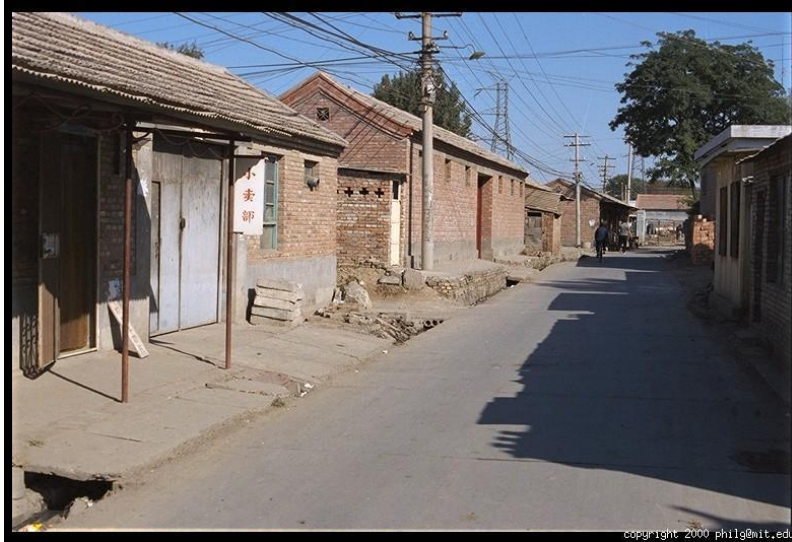
Green



Blue



Color



Red intensity



Green



Blue



Images in Python (import numpy)

N x M grayscale image “im”

- `im[0,0,0]` = top-left pixel value, **red channel**
- `im[y,x,1]` = y pixels down, x pixels to right, **green channel**
- `im[N-1,M-1,2]` = bottom-right pixel, **blue channel**

Diagram illustrating the structure of a 3D image array (N x M x 3) representing the Red (R), Green (G), and Blue (B) channels. The array is shown as a stack of 3D slices, with the first slice labeled 'R' (Red), the second 'G' (Green), and the third 'B' (Blue). The 'Row' and 'Column' axes are indicated by arrows.

Row	Column	R	G	B
0	0	0.92	0.92	0.92
0	1	0.93	0.99	0.99
0	2	0.94	0.91	0.91
0	3	0.97	0.92	0.95
0	4	0.62	0.95	0.97
0	5	0.37	0.91	0.85
0	6	0.85	0.97	0.89
0	7	0.97	0.90	0.79
0	8	0.93	0.97	0.85
0	9	0.92	0.95	0.85
0	10	0.99	0.95	0.85
1	0	0.95	0.91	0.91
1	1	0.89	0.92	0.95
1	2	0.82	0.95	0.97
1	3	0.89	0.91	0.95
1	4	0.56	0.97	0.95
1	5	0.31	0.95	0.95
1	6	0.75	0.91	0.95
1	7	0.92	0.97	0.95
1	8	0.81	0.90	0.95
1	9	0.95	0.97	0.95
1	10	0.91	0.95	0.95
2	0	0.89	0.91	0.95
2	1	0.72	0.92	0.95
2	2	0.51	0.95	0.95
2	3	0.55	0.97	0.95
2	4	0.51	0.91	0.95
2	5	0.42	0.97	0.95
2	6	0.57	0.90	0.95
2	7	0.41	0.97	0.95
2	8	0.49	0.95	0.95
2	9	0.91	0.97	0.95
2	10	0.92	0.95	0.95
3	0	0.96	0.95	0.95
3	1	0.95	0.92	0.95
3	2	0.88	0.95	0.95
3	3	0.94	0.97	0.95
3	4	0.56	0.95	0.95
3	5	0.46	0.91	0.95
3	6	0.91	0.97	0.95
3	7	0.87	0.90	0.95
3	8	0.90	0.97	0.95
3	9	0.97	0.95	0.95
3	10	0.95	0.95	0.95
4	0	0.71	0.91	0.95
4	1	0.81	0.92	0.95
4	2	0.81	0.95	0.95
4	3	0.87	0.97	0.95
4	4	0.57	0.95	0.95
4	5	0.37	0.91	0.95
4	6	0.80	0.97	0.95
4	7	0.88	0.90	0.95
4	8	0.89	0.97	0.95
4	9	0.79	0.95	0.95
4	10	0.85	0.95	0.95
5	0	0.49	0.91	0.95
5	1	0.62	0.92	0.95
5	2	0.60	0.95	0.95
5	3	0.58	0.97	0.95
5	4	0.50	0.95	0.95
5	5	0.60	0.91	0.95
5	6	0.58	0.97	0.95
5	7	0.50	0.90	0.95
5	8	0.61	0.97	0.95
5	9	0.45	0.95	0.95
5	10	0.33	0.95	0.95
6	0	0.86	0.91	0.95
6	1	0.84	0.92	0.95
6	2	0.74	0.95	0.95
6	3	0.58	0.97	0.95
6	4	0.51	0.95	0.95
6	5	0.39	0.91	0.95
6	6	0.73	0.97	0.95
6	7	0.92	0.90	0.95
6	8	0.91	0.97	0.95
6	9	0.49	0.95	0.95
6	10	0.74	0.95	0.95
7	0	0.96	0.91	0.95
7	1	0.67	0.92	0.95
7	2	0.54	0.95	0.95
7	3	0.85	0.97	0.95
7	4	0.48	0.95	0.95
7	5	0.37	0.91	0.95
7	6	0.88	0.97	0.95
7	7	0.90	0.90	0.95
7	8	0.94	0.97	0.95
7	9	0.82	0.95	0.95
7	10	0.93	0.95	0.95
8	0	0.69	0.91	0.95
8	1	0.49	0.92	0.95
8	2	0.56	0.95	0.95
8	3	0.66	0.97	0.95
8	4	0.43	0.95	0.95
8	5	0.42	0.91	0.95
8	6	0.77	0.97	0.95
8	7	0.73	0.90	0.95
8	8	0.71	0.97	0.95
8	9	0.90	0.95	0.95
8	10	0.99	0.95	0.95
9	0	0.79	0.91	0.95
9	1	0.73	0.92	0.95
9	2	0.90	0.95	0.95
9	3	0.67	0.97	0.95
9	4	0.33	0.95	0.95
9	5	0.61	0.91	0.95
9	6	0.69	0.97	0.95
9	7	0.79	0.90	0.95
9	8	0.73	0.97	0.95
9	9	0.93	0.95	0.95
9	10	0.97	0.95	0.95
10	0	0.91	0.91	0.95
10	1	0.94	0.92	0.95
10	2	0.89	0.95	0.95
10	3	0.49	0.97	0.95
10	4	0.41	0.95	0.95
10	5	0.78	0.91	0.95
10	6	0.78	0.97	0.95
10	7	0.77	0.90	0.95
10	8	0.89	0.97	0.95
10	9	0.99	0.95	0.95
10	10	0.93	0.95	0.95

Images in Python (import numpy, scikit)

Take care of types!

- uint8 (values 0 to 255) – `io.imread("file.jpg")`
- float32 (values 0 to 255) – `io.imread("file.jpg").astype(np.float32)`
- float32 (values 0 to 1) – `img_as_float32(io.imread("file.jpg"))`

Row ↓ **Column** →

0.92	0.93	0.94	0.97	0.62	0.37	0.85	0.97	0.93	0.92	0.99
0.95	0.89	0.82	0.89	0.56	0.31	0.75	0.92	0.81	0.95	0.91
0.89	0.72	0.51	0.55	0.51	0.42	0.57	0.41	0.49	0.91	0.92
0.96	0.95	0.88	0.94	0.56	0.46	0.91	0.87	0.90	0.97	0.95
0.71	0.81	0.81	0.87	0.57	0.37	0.80	0.88	0.89	0.79	0.85
0.49	0.62	0.60	0.58	0.50	0.60	0.58	0.50	0.61	0.45	0.33
0.86	0.84	0.74	0.58	0.51	0.39	0.73	0.92	0.91	0.49	0.74
0.96	0.67	0.54	0.85	0.48	0.37	0.88	0.90	0.94	0.82	0.93
0.69	0.49	0.56	0.66	0.43	0.42	0.77	0.73	0.71	0.90	0.99
0.79	0.73	0.90	0.67	0.33	0.61	0.69	0.79	0.73	0.93	0.97
0.91	0.94	0.89	0.49	0.41	0.78	0.78	0.77	0.89	0.99	0.93

Thanks

- Next lecture: image Filtering

