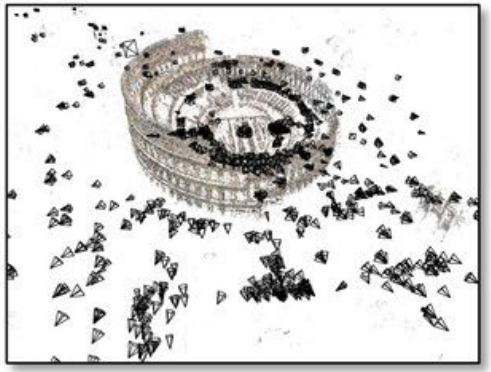
Image processing

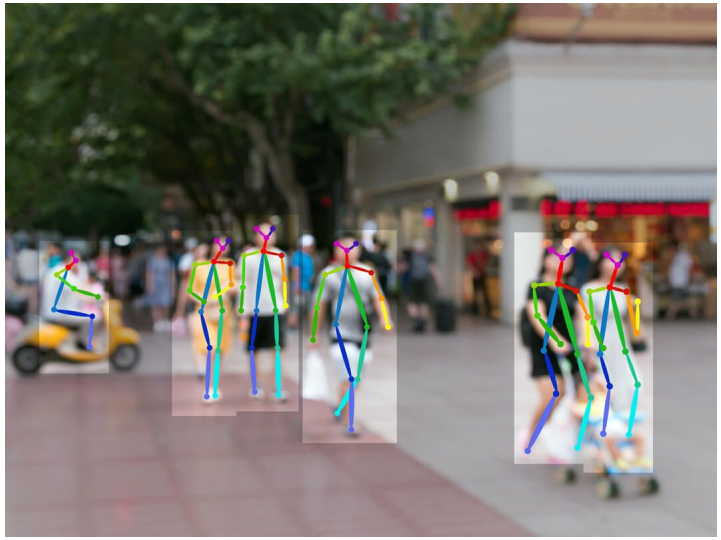
Rotem Shalev-Arkushin
rotems7@mail.tau.ac.il

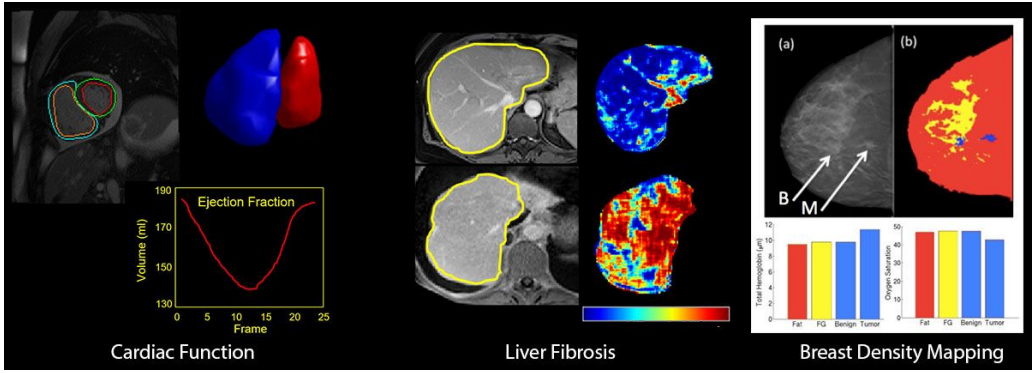
December 17 2025

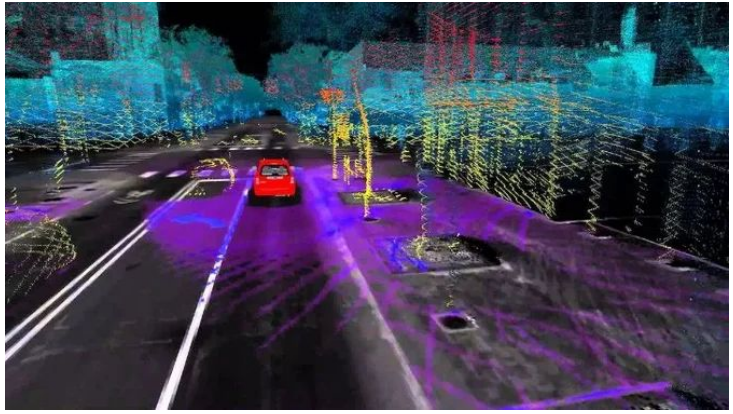
Applications

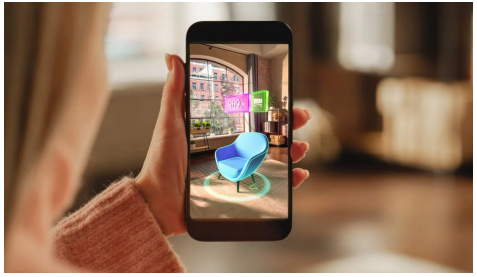
- **Image Enhancement** – deblurring, denoising, sharpening, etc.
- **Registration & Alignment** – panorama stitching, satellite image alignment, AR overlay.
- **Segmentation** – used for object detection and recognition, e.g. for medical imaging.
- **Measurement & Inspection** – detecting defects in manufacturing, counting cells, measuring distances or motion.
- **Tracking & Motion Analysis** – optical flow, gesture tracking, traffic monitoring, video stabilization.
- **3D Reconstruction** – structure-from-motion, depth estimation, multi-view geometry, robotics navigation.
- **Compression** – pyramids, transforms, and feature extraction used in JPEG, wavelets, video codecs.
- **Augmented Reality & Robotics** – feature matching, localization, SLAM, obstacle detection.

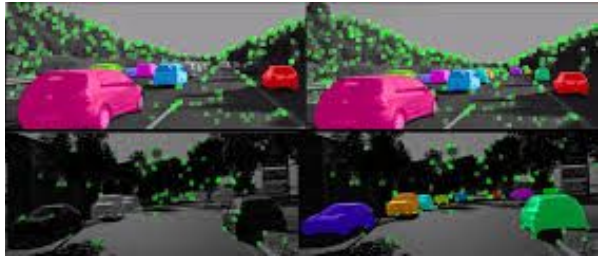





Ejection Fraction
Volume (ml)
Frame
(a)
(b)
B
M
Cardiac Function
Liver Fibrosis
Breast Density Mapping

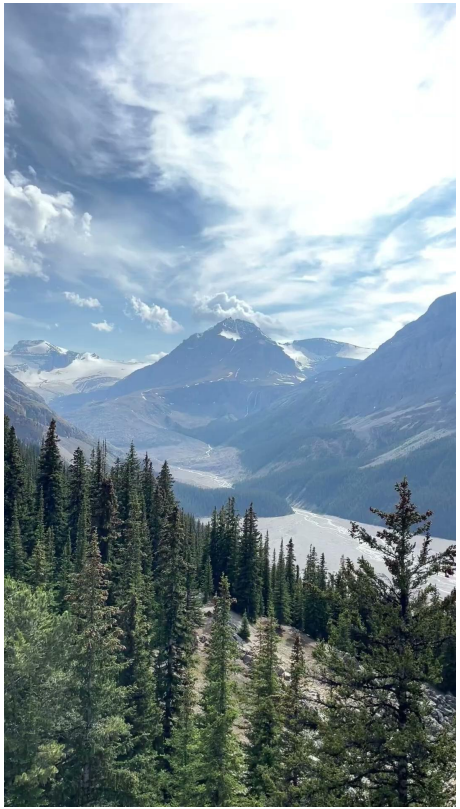




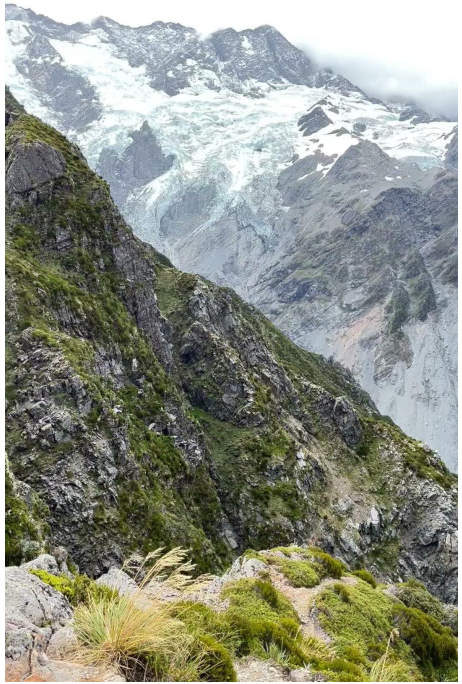


Panorama

Panorama exercise



Panorama exercise



Panorama exercise

What do we need to know to construct a panorama from video frames?

Panorama exercise

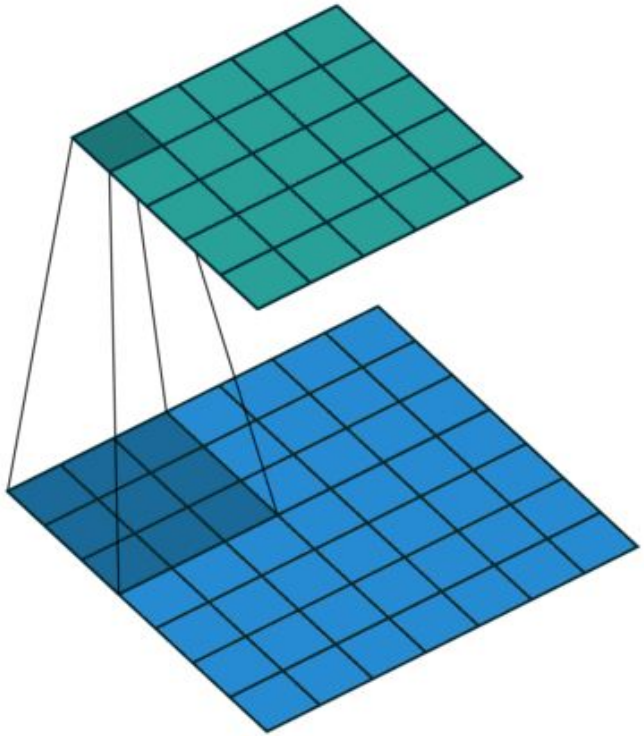
What do we need to know to construct a panorama?

- Image registration and alignment
  - Find unique features in each image (use feature detectors and descriptors)
  - Find matching features between images
  - Align images

- Stitch images together
  - Place aligned images on a shared canvas
  - Blend images together seamlessly

Reminder - 2D convolution

$$g(x,y) = \omega * f(x,y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} \omega(i,j) f(x-i, y-j)$$



* Illustration from Aaditya Prakash's blog

Image Blurring/smoothing

Convolve with a blurring kernel.

In Python: `convolve2d(koshka_im_gray, kernel, mode='same', boundary='symm')`

- Box blurring

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

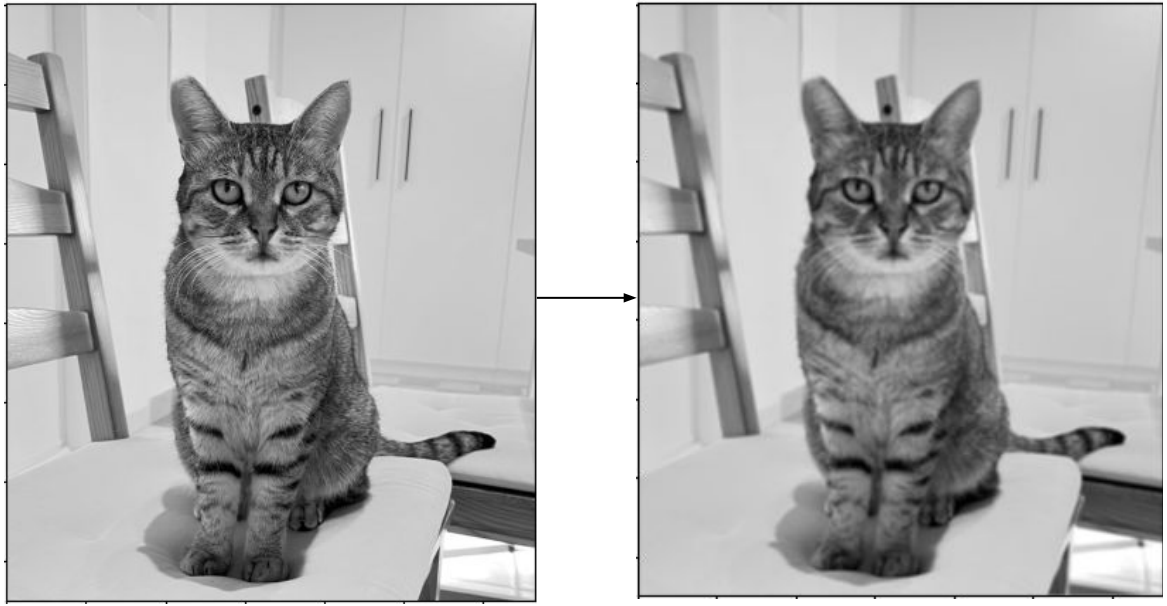
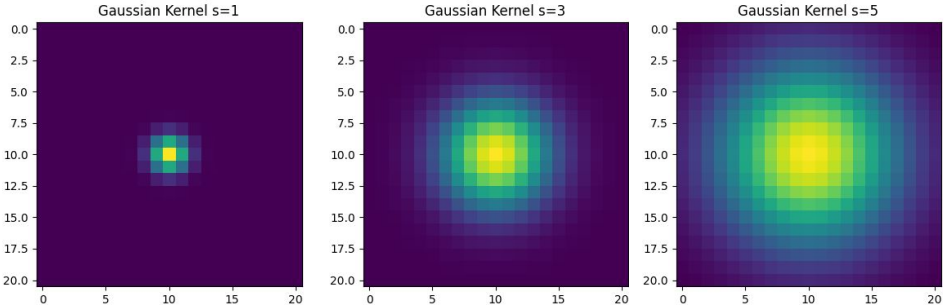
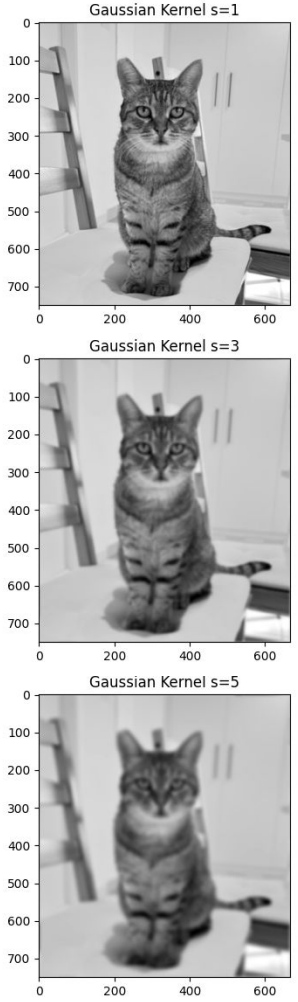


Image Blurring/smoothing

- Convolve with a Gaussian kernel

$$g(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$





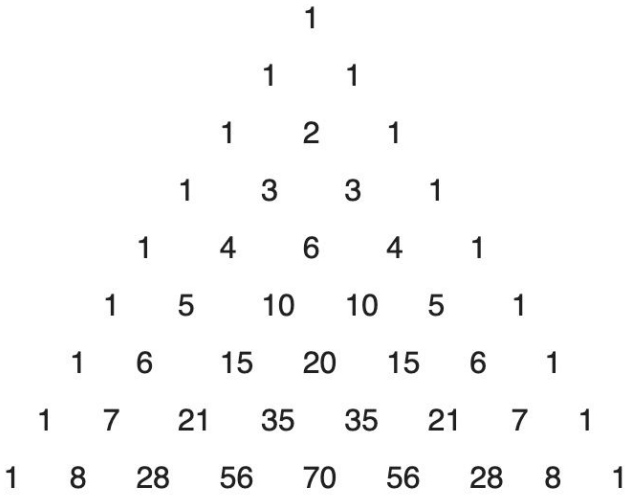
Pascal's Triangle

- An infinite triangular array of the binomial coefficients:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

- De Moivre–Laplace theorem:

$$\frac{1}{2^n}\binom{n}{k} \lim_{n \to \infty} \frac{1}{2\pi\sigma^2} e^{-\frac{((k-n)/2)^2}{2\sigma^2}}$$

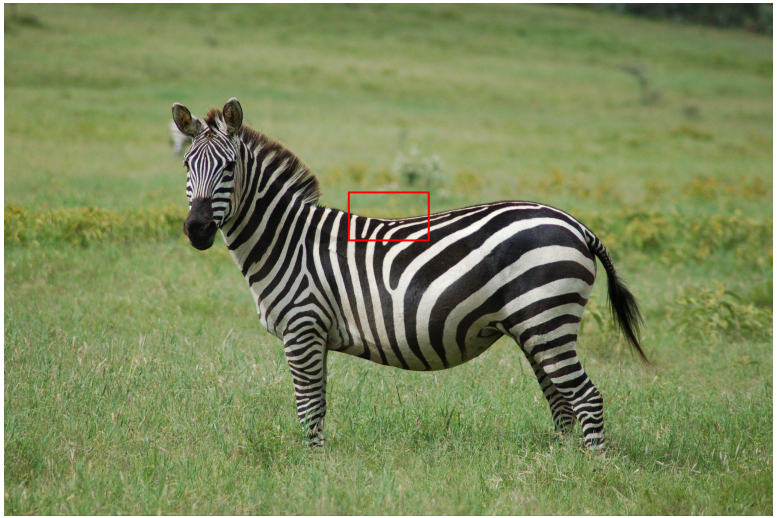


- Repeated convolution with [1 1] generates a sequence of kernels whose shape approaches a Gaussian => binomial filters act as discrete Gaussian approximations.

$$[1\ 1] * [1\ 1] = [1\ 2\ 1]$$
$$[1\ 2\ 1] * [1\ 2\ 1] = [1\ 4\ 6\ 4\ 1]$$

Edge detection

- Edge in an image = a line of high intensity change



Edge detection

- Edge in an image = a line of high intensity change
- Why is it interesting?
  segmentation, feature detection, image alignment, recognizing shapes or structures in scenes, controlled generation, and more.
- Gradient (vector of the derivatives) = direction of most rapid intensity change.

$$\frac{d}{dx}f(i,j) = \lim_{\epsilon->0} \frac{f(i,j)-f(i-\epsilon,j)}{\epsilon} \simeq f(i,j) - f(i-1,j)$$

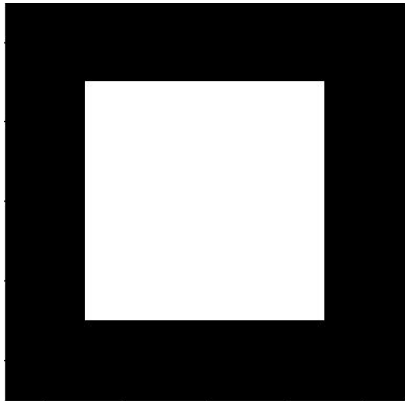
Image Gradient

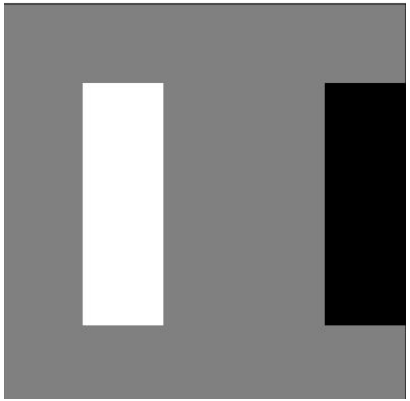
0	0	0	0	0
0	255	255	255	0
0	255	255	255	0
0	255	255	255	0
0	0	0	0	0

* [1, -1] =

0	0	0	0	0
0	255	0	0	-255
0	255	0	0	-255
0	255	0	0	-255
0	0	0	0	0

→ clip(0,255)





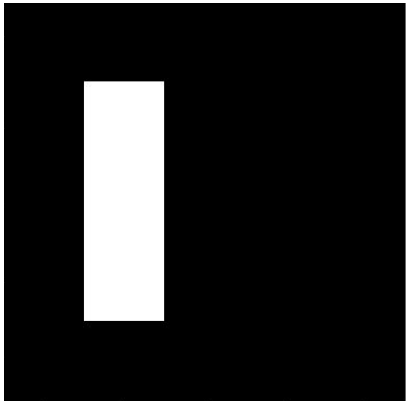
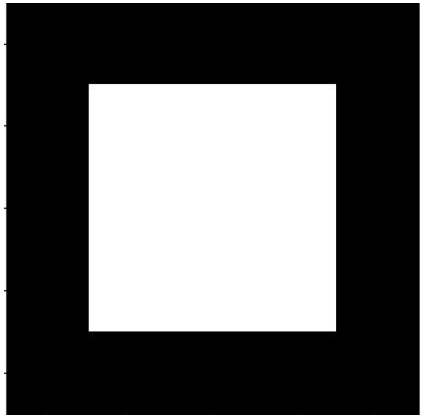


Image Gradient

0	0	0	0	0
0	255	255	255	0
0	255	255	255	0
0	255	255	255	0
0	0	0	0	0

* [1, -1]^T =

0	0	0	0	0
0	255	255	255	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0



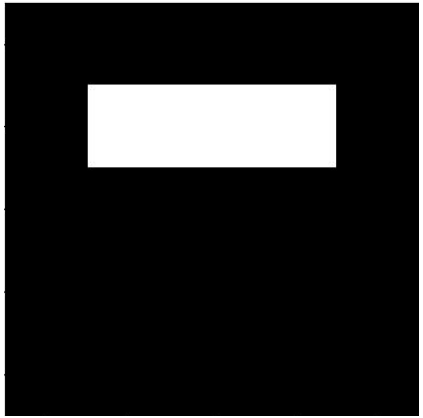
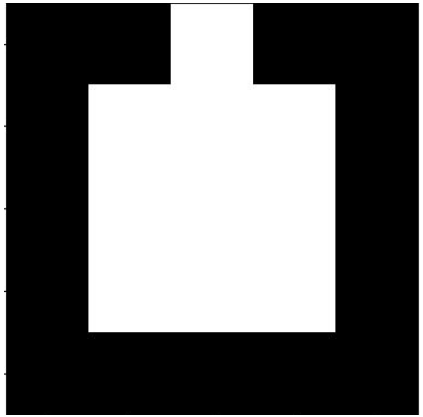


Image Gradient

0	0	**255**	0	0
0	255	255	255	0
0	255	255	255	0
0	255	255	255	0
0	0	0	0	0

* [1, -1]^T =



Boundaries?

- Zero-padding
- Mirror last
- Cyclic? **No!**

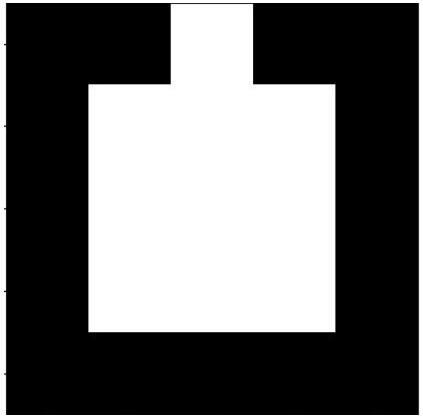
Image Gradient

0	0	255	0	0
0	255	255	255	0
0	255	255	255	0
0	255	255	255	0
0	0	0	0	0

* [1, -1]^T =

Symmetric / mirror

0	0	0	0	0
0	255	0	255	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0



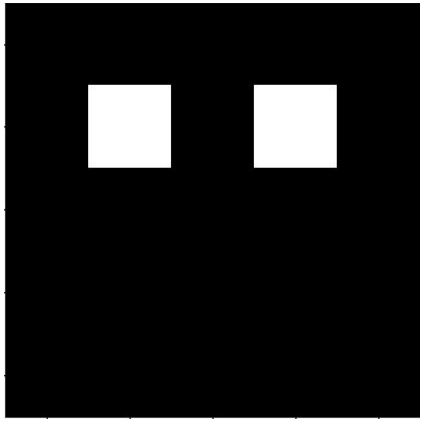
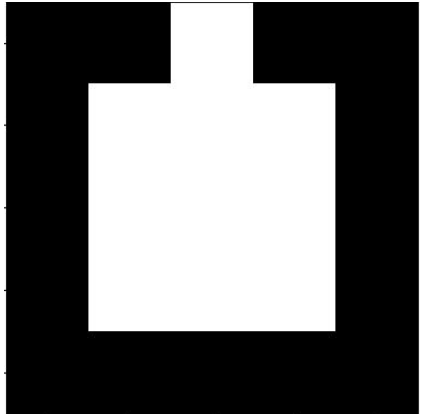


Image Gradient

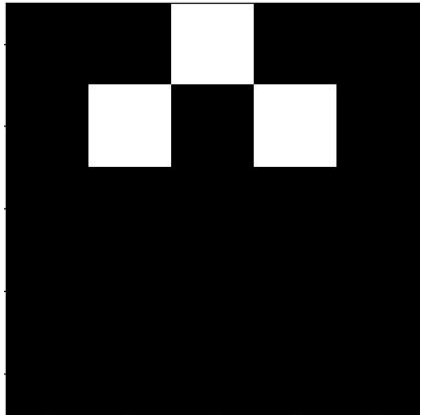
0	0	255	0	0
0	255	255	255	0
0	255	255	255	0
0	255	255	255	0
0	0	0	0	0



* [1, -1]^T =

zero-pad / cyclic (wrap)

0	0	255	0	0
0	255	0	255	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

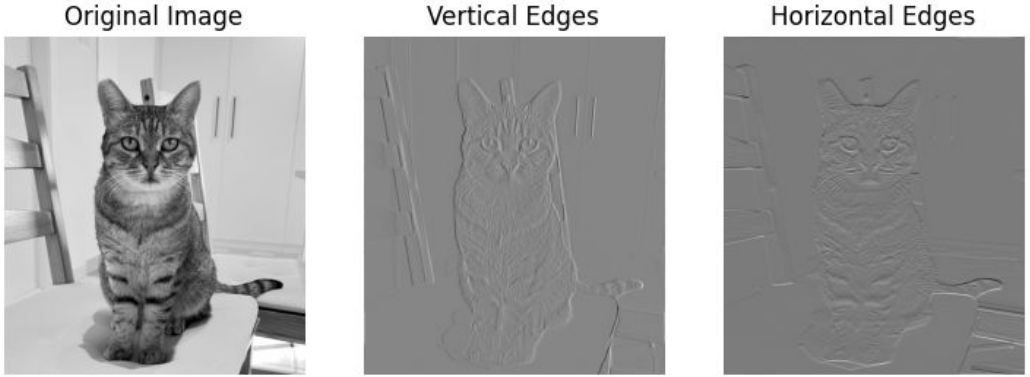


Derivative Approximation

$$f'(x) \approx \frac{f(x+1) - f(x-1)}{2}$$

With Convolution:

- use kernel of [1 0 -1] and detect vertical edges in an image.
- use kernel of [1 0 -1]^T and detect horizontal edges in an image.



Sobel

- To avoid noise – smooth the image and then apply derivative.

$$(Im * \frac{1}{4}[1\ 2\ 1]) * [-\ 1\ 0\ 1]$$

- Sobel – apply them both together with one convolution!

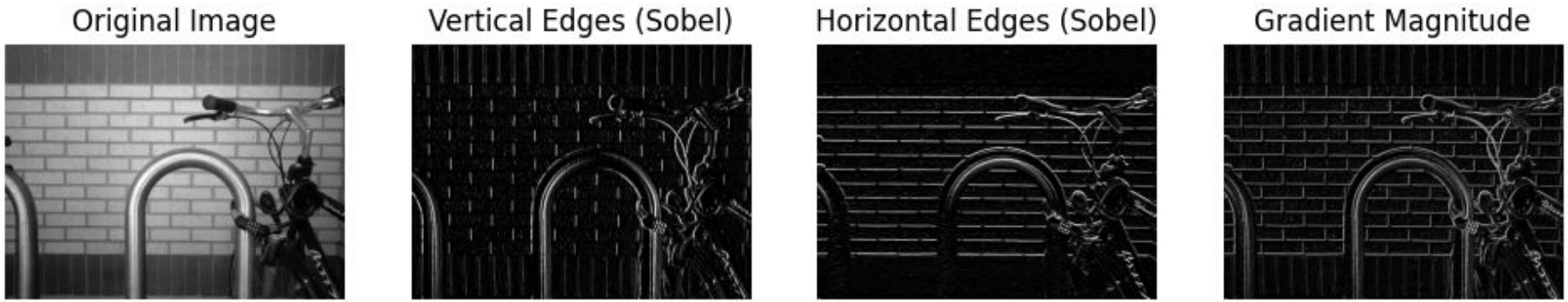
$$S_x = \begin{bmatrix}1\\2\\1\end{bmatrix}\begin{bmatrix}-1 & 0 & 1\end{bmatrix} = \begin{bmatrix}-1 & 0 & 1\\-2 & 0 & 2\\-1 & 0 & 1\end{bmatrix}$$

$$S_y = \begin{bmatrix}-1\\0\\1\end{bmatrix}\begin{bmatrix}1 & 2 & 1\end{bmatrix} = \begin{bmatrix}-1 & -2 & -1\\0 & 0 & 0\\1 & 2 & 1\end{bmatrix}$$

Sobel

Gradient magnitude and direction:

$$|\nabla f| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2} \qquad \alpha = \tan^{-1}\left(\left(\frac{\partial f}{\partial y}\right) \Big/ \left(\frac{\partial f}{\partial x}\right)\right)$$



Canny Edge Detector

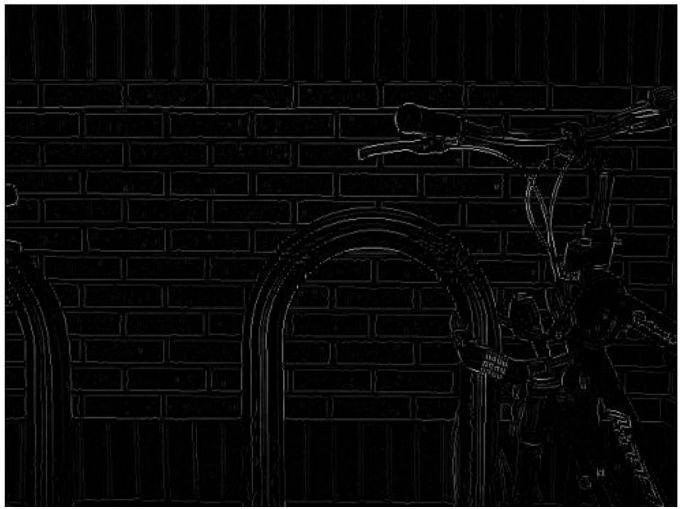
Algorithm:

1. Smooth the image with a Gaussian filter to remove noise.
2. Find the intensity gradients of the image with a simple kernel (e.g. ½ [1, 0, -1]): Gx, Gy.
3. Calculate the gradient direction: $\tan(\alpha) = \frac{G_x}{G_y}$ and magnitude: $\sqrt{G_x^2 + G_y^2}$
4. **Non-maximum suppression**: only keep local magnitude maxima in each edge direction.

Non-maximum suppression

Before NMS

After NMS



Non-maximum suppression

Angles

18	96	103	100	0.08
15	95	100	97	12
0.72	102	125	124	26
0.4	102	125	125	70
18	96	103	101	0.08

Magnitudes

26	45	78	18	18
29	43	80	42	13
30	36	78	47	8
32	30	76	51	4
35	25	75	54	2

Non-maximum suppression

Angles

18	96	103	100	0.08
15	95	100	97	12
0.72	102	125	124	26
0.4	102	125	125	70
18	96	103	101	0.08

Magnitudes

0	45	78	18	18
0	43	80	42	0
0	36	78	47	8
0	30	76	51	4
35	25	75	54	2

0 <= a <= 22

Non-maximum suppression

Angles

18	96	103	100	0.08
15	95	100	97	12
0.72	102	125	124	26
0.4	102	125	125	70
18	96	103	101	0.08

Magnitudes

0	0	78	0	18
0	0	80	0	0
0	36	78	47	8
0	30	76	51	4
35	0	75	0	2

67.5 <= a <= 112.5

Non-maximum suppression

Angles

18	96	103	100	0.08
15	95	100	97	12
0.72	102	125	124	26
0.4	102	125	125	70
18	96	103	101	0.08

Magnitudes

0	0	78	0	18
0	0	80	0	0
0	36	78	47	8
0	30	76	0	4
35	0	75	0	2

$112.5 <= a <= 157.5$

Non-maximum suppression

Before NMS

26	45	78	18	18
29	43	80	42	13
30	36	78	47	8
32	30	76	51	4
35	25	75	54	2

After NMS

0	0	**78**	0	**18**
0	0	**80**	0	0
0	**36**	**78**	**47**	**8**
0	**30**	**76**	0	**4**
35	0	**75**	0	2

Canny Edge Detector

Algorithm:

1. Smooth the image with a Gaussian filter to remove noise.
2. Find the intensity gradients of the image with a simple kernel (e.g. ½ [1, 0, -1]): Gx, Gy.
3. Calculate the gradient direction: $\tan(\alpha) = \frac{G_x}{G_y}$ and magnitude: $\sqrt{G_x^2 + G_y^2}$
4. **Non-maximum suppression**: only keep local magnitude maxima in each edge direction.
5. **Hysteresis with double threshold (t1>t2)**: keep edges > t1.
   Link t1 > edges > t2 – keep only edges neighboring an edge > t1.

Hysteresis

Before Hysteresis

0	0	**78**	0	**18**
0	0	**80**	0	0
0	**36**	**78**	**47**	**8**
0	**30**	**76**	0	**4**
35	0	**75**	0	2

After Hysteresis

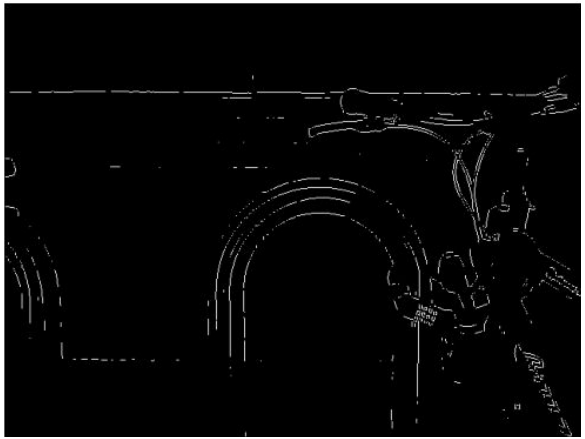
0	0	**78**	0	0
0	0	**80**	0	0
0	0	**78**	**47**	0
0	0	**76**	0	0
0	0	**75**	0	0

t1 = 77, t2 = 40

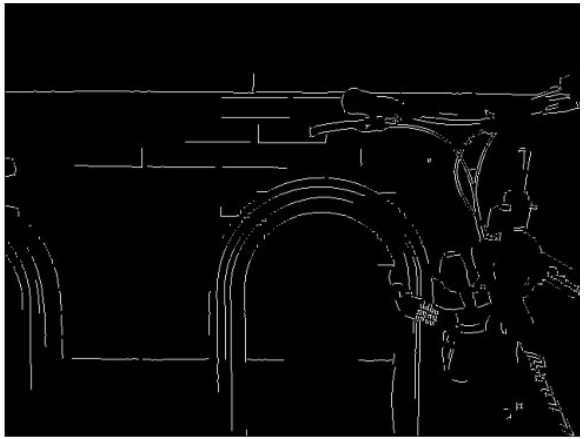
Canny Edge Detector

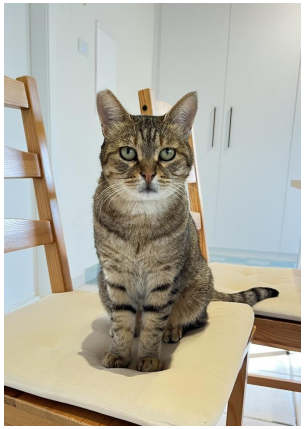
Initial magnitude

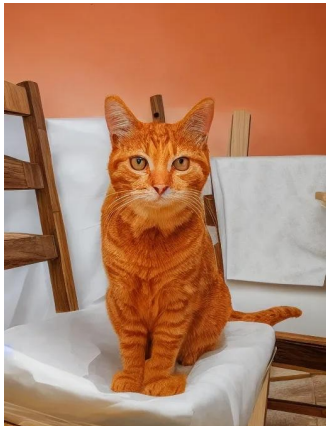
> t1

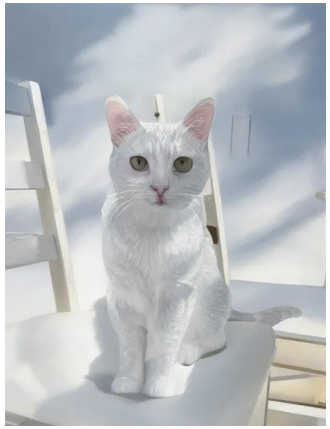


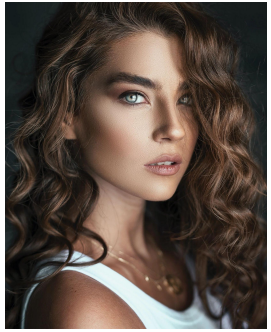
After hysteresis

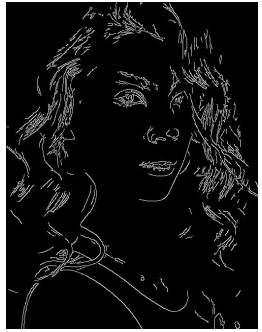


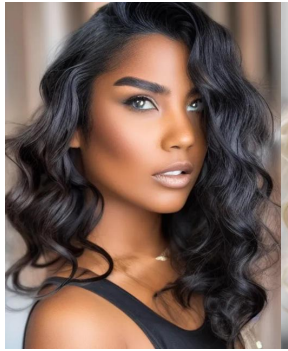


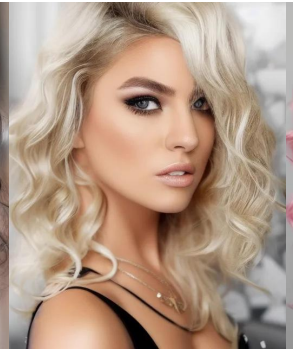


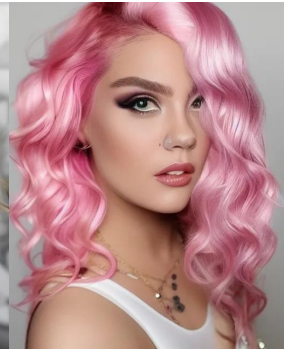












ControlNet generations with Canny edges

Binary images

- Images that are only black and white (pixel values are either 0 or 255).
- Used for segmentation, masking for inpainting, controlled generation, etc.

Pixel Neighborhoods and Connectivity

- **Connected Component:** A set of pixels is a Connected Component if for every two pixels in the set, there is a path between them such that every two successive pixels in the path are in the set, and they are neighbors (could be with 4 or 8 connectivity).

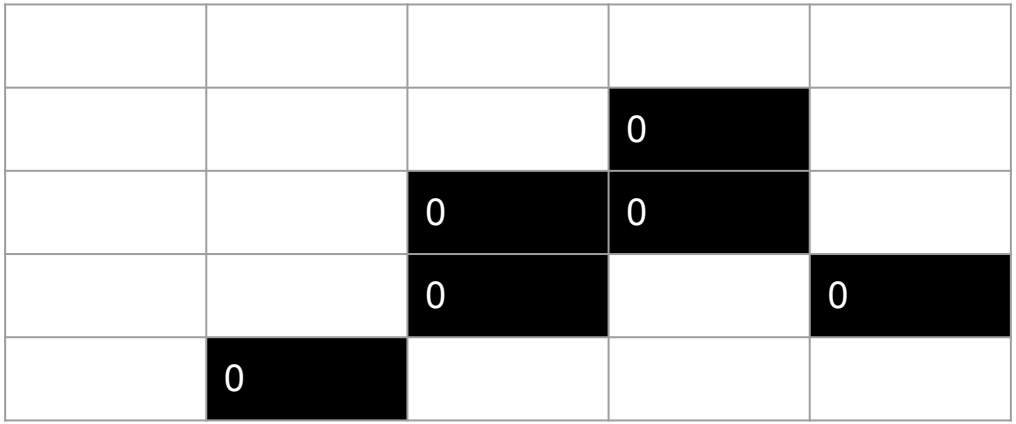
4-neighbors

0	255	0
255	255	255
0	255	0

8-neighbors

255	255	255
255	255	255
255	255	255

Pixel Neighborhoods and Connectivity

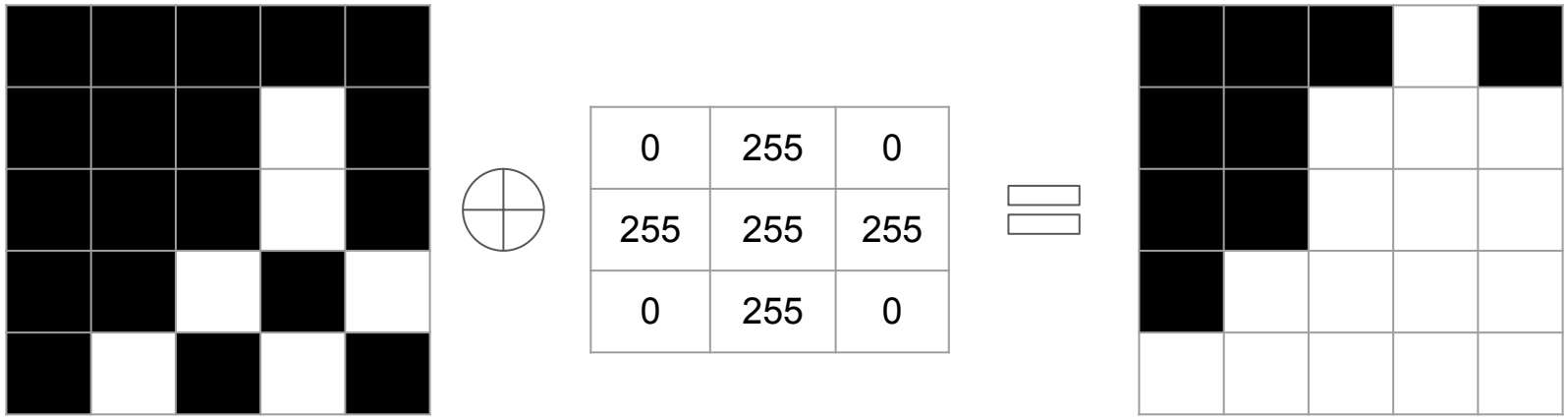


How many connected components?

Morphological Operations

- **Dilation** - expansion of the white connected-component by a filter.

$$A \oplus B = \{ x \mid (B)_x \cap A \neq \emptyset \}$$

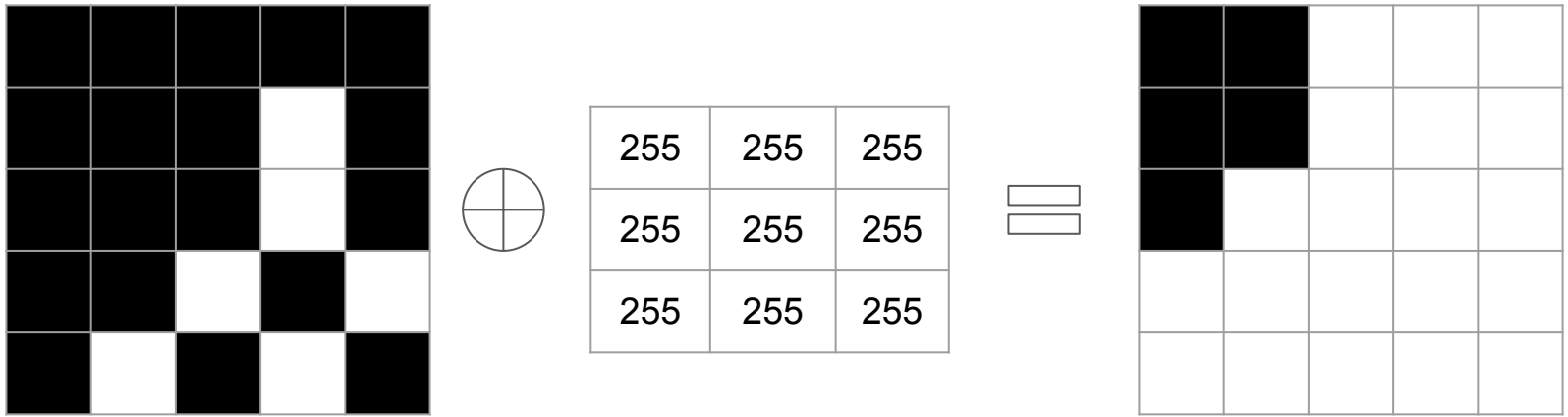


0	255	0
255	255	255
0	255	0

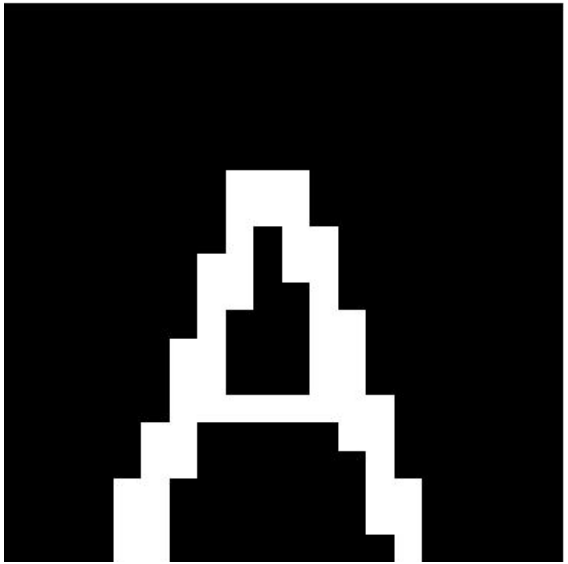
Morphological Operations

- **Dilation** - expansion of the white connected-component by a filter.

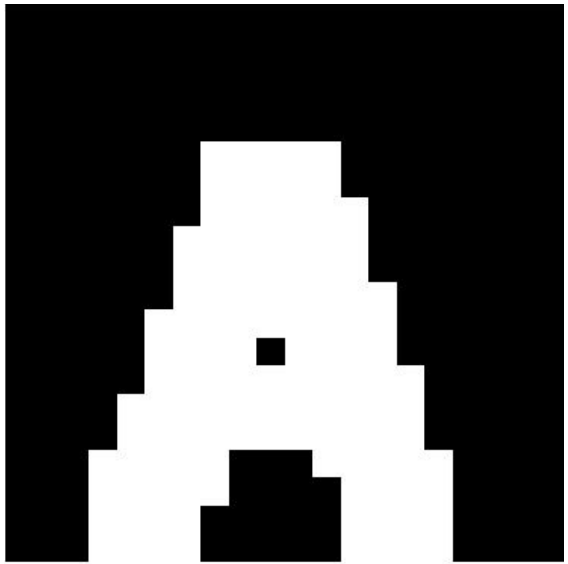
$$A \oplus B = \{ x \mid (B)_x \cap A \neq \emptyset \}$$



255	255	255
255	255	255
255	255	255



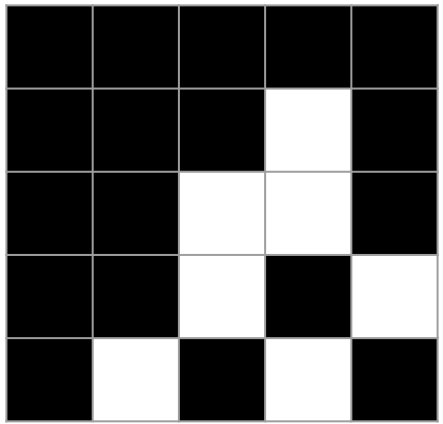
255	255	255
255	255	255
255	255	255



Morphological Operations

- **Erosion** - shrinking of the white connected-component by a filter.

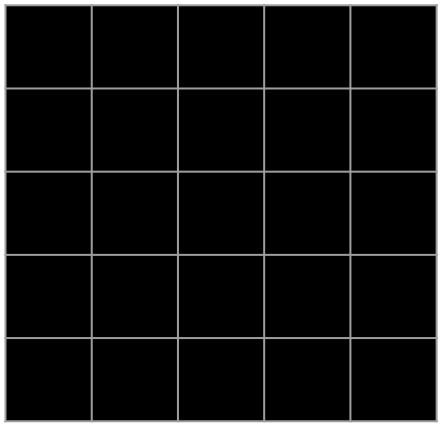
$$A \ominus B = \{ x \mid (B)_x \subseteq A \}$$

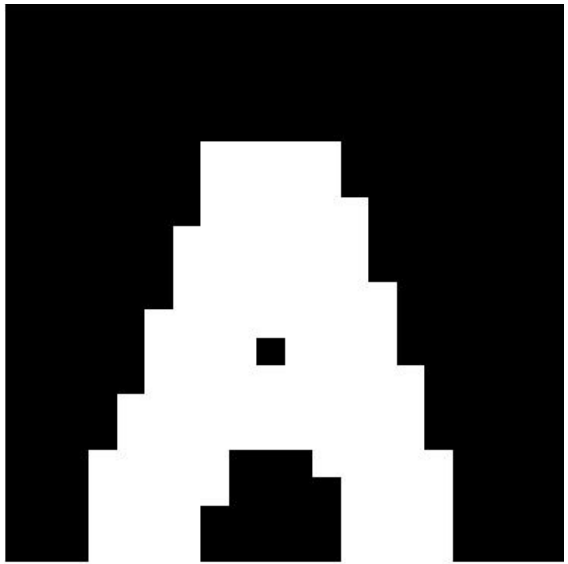


$\ominus$

255	255	255
255	255	255
255	255	255

$=$

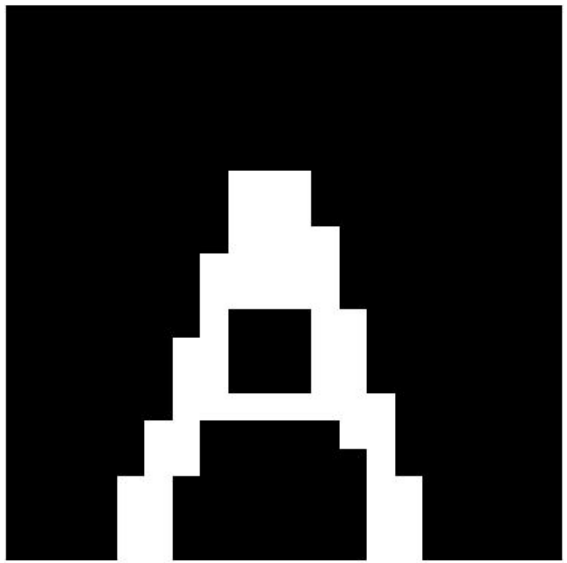




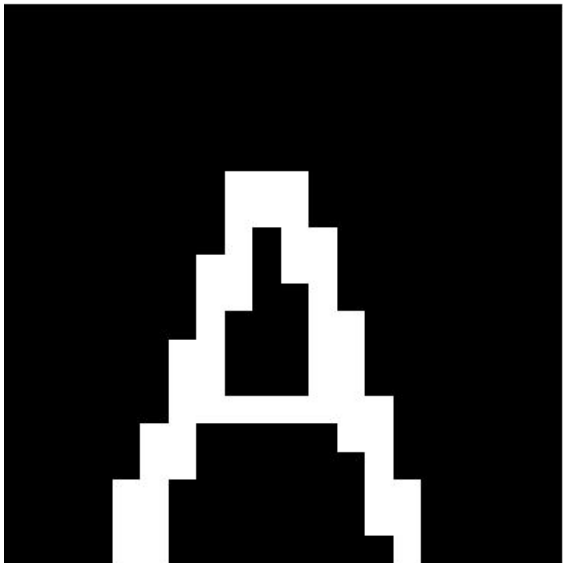
$\ominus$

255	255	255
255	255	255
255	255	255

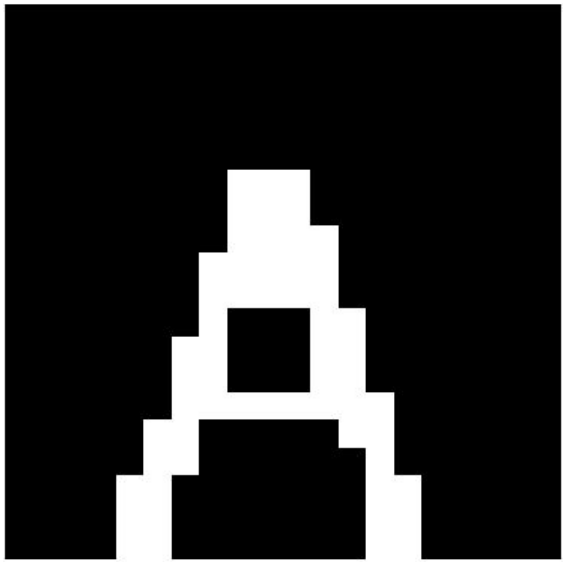
$=$



Before dilation



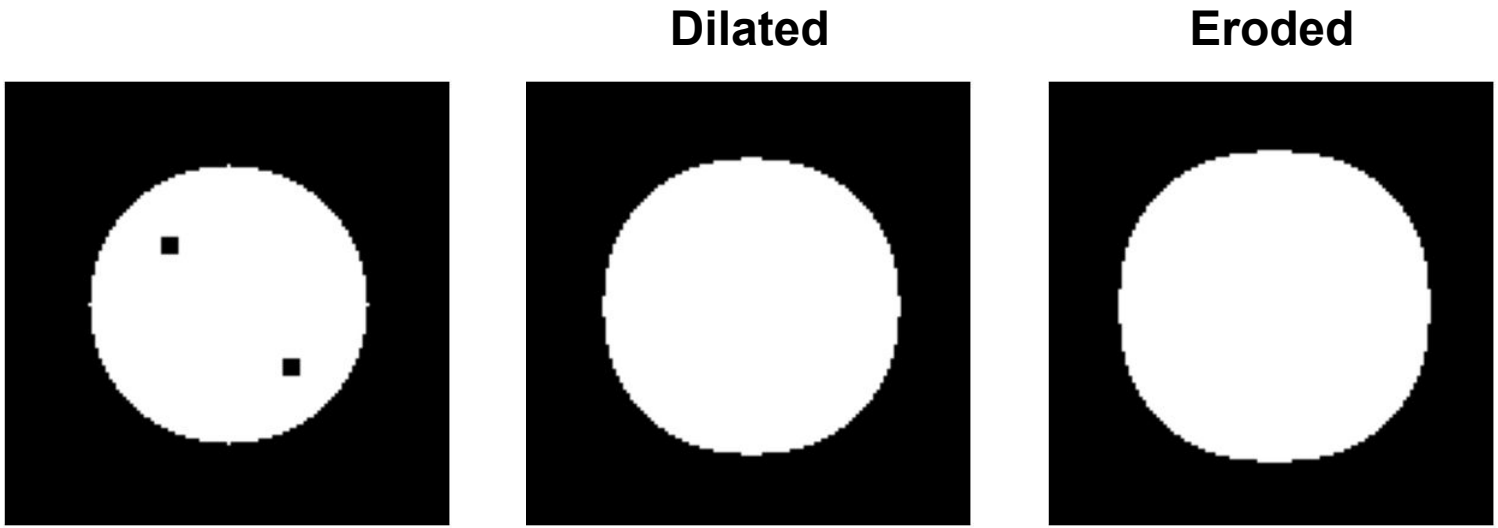
After erosion(dilation(im))



Note – these are not opposites! erosion(dilation(im)) != im

Morphological Operations

- **Closing** - Dilation → Erosion
  Good for filling holes



Morphological Operations

- **Opening** - Erosion → Dilation
  Good for removing noise

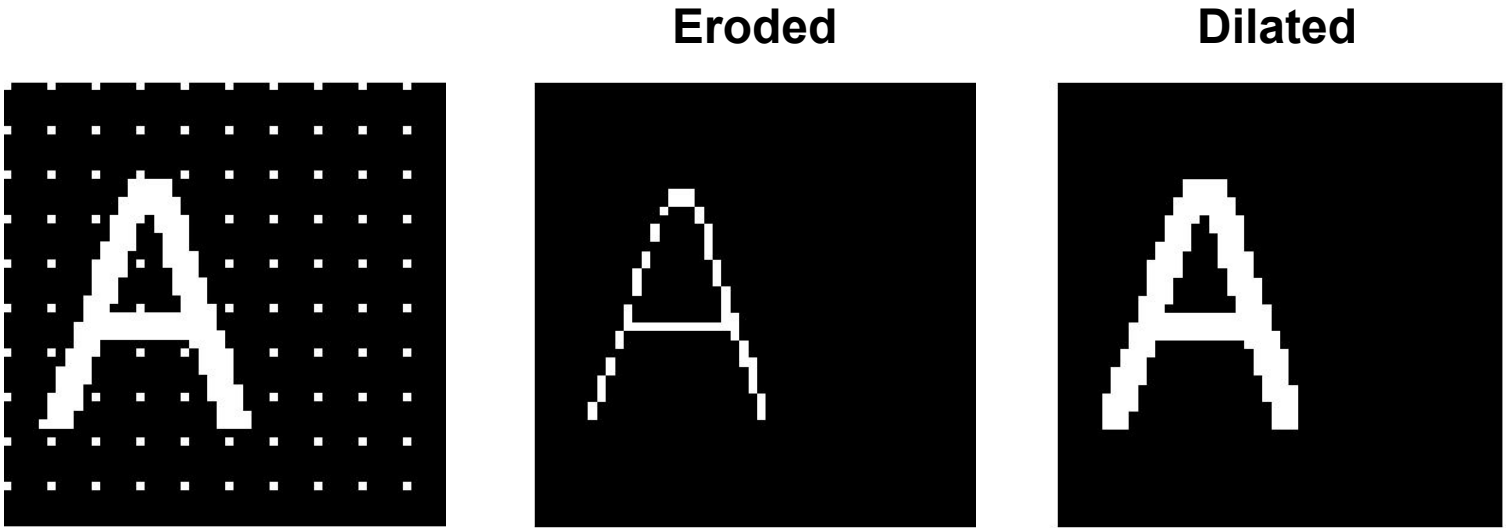


Image Resizing

Original image

im/4

im/8

Why does this happen?

Image Resizing

Solution – blur before sub-sampling!

Original image

G(im)/4

G(im)/8

Note – Filter size should double for each ½ size reduction. Why?

Image Resizing

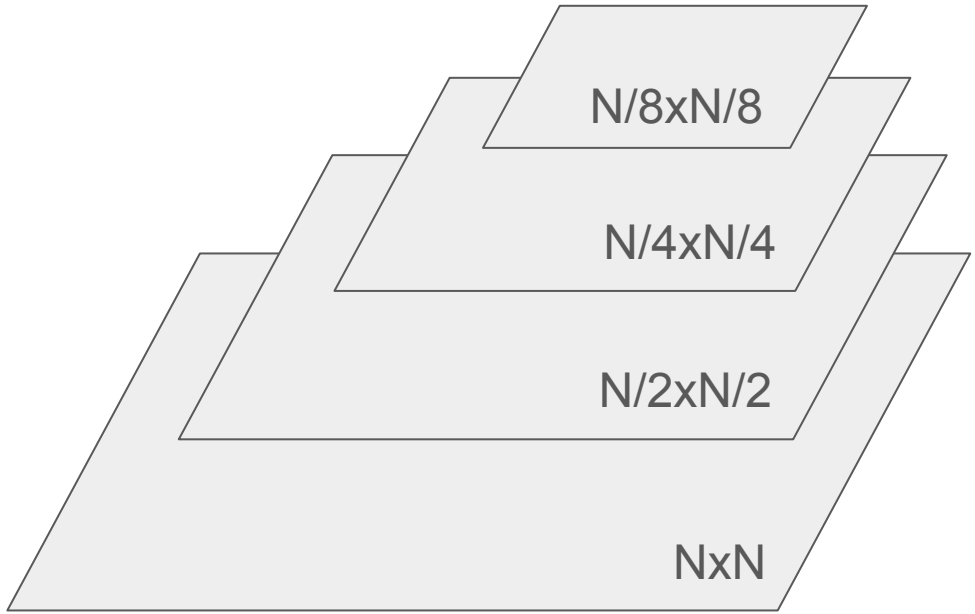
- **Reduce:**
  - Blur
  - Sub-sample: select every n-th pixel in every n-th row.

- **Expand:**
  - Zero padding of every other pixel and row (e.g. for n=2: 1, 0, 2, 0, 3, 0,...)
  - Blur

Aliasing

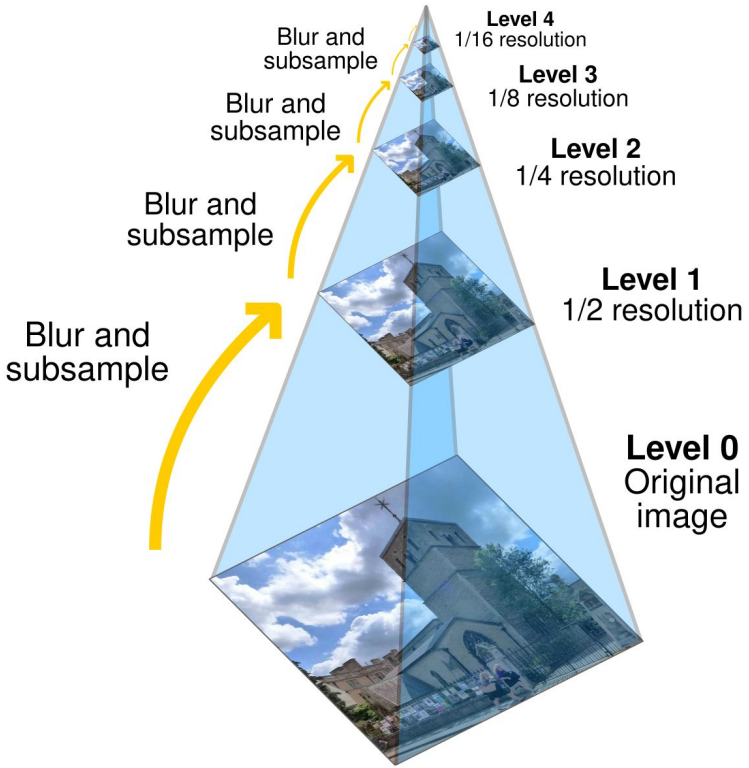
Image Pyramids

- **Multi-scale template matching / object / feature detection:** Searching for patterns at different scales efficiently.
- **Image blending / seamless stitching:** combine images smoothly with Laplacian pyramids.
- **Efficient optical flow / motion estimation:** Coarse-to-fine pyramids let you estimate large motions at low resolution, then refine at higher resolution.
- **Compression:** Gaussian pyramids are used in wavelet-based or pyramid-based image compression.



Gaussian Pyramid

Each level is **reduced** from the previous –
gaussian blur –> subsample.

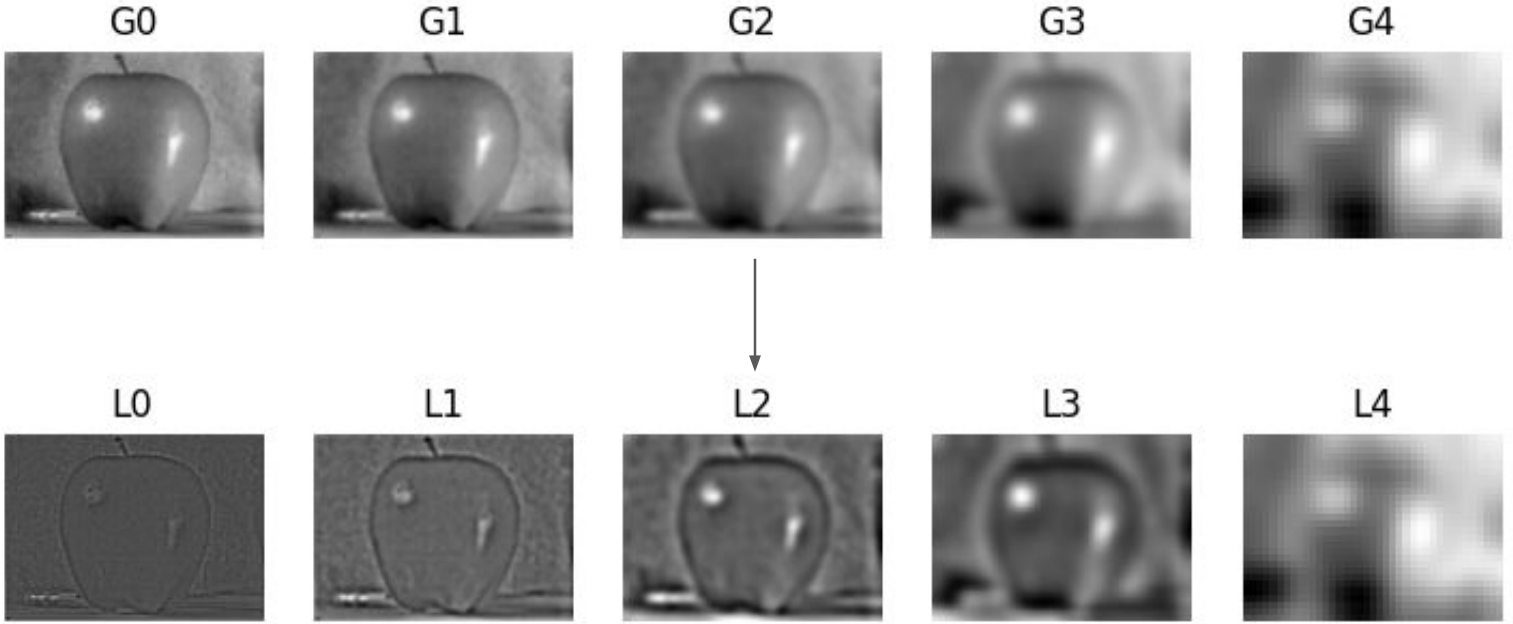


- Pyramid image from wikipedia

Laplacian Pyramid

$$L_n = G_n$$

$$L_{n-1} = G_{n-1} - Expand(G_n)$$



Laplacian Pyramid

$$L_n = G_n$$
$$L_{n-1} = G_{n-1} - Expand(G_n)$$
$$Expand(L_n) + L_{n-1} = Expand(G_n) + G_{n-1} - Expand(G_n) = G_{n-1}$$
$$\sum_{i=0}^{n} L_i = G_0$$

- **Gaussian** Pyramid in the Fourier domain – Multiplication with a Gaussian kernel.
- **Laplacian** Pyramid in the Fourier domain – difference between two powers of Gaussian kernels.
- Sum of all laplacian layers = the original image.

Image Blending with Laplacian Pyramids

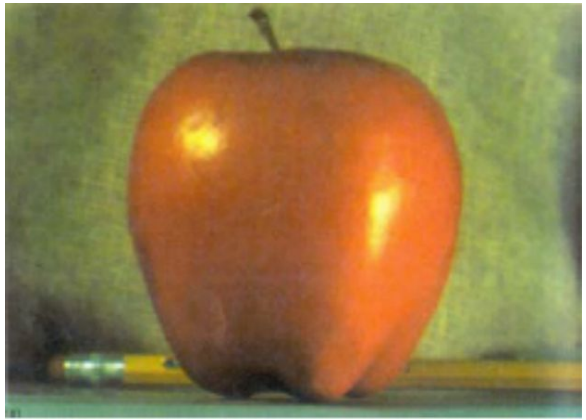
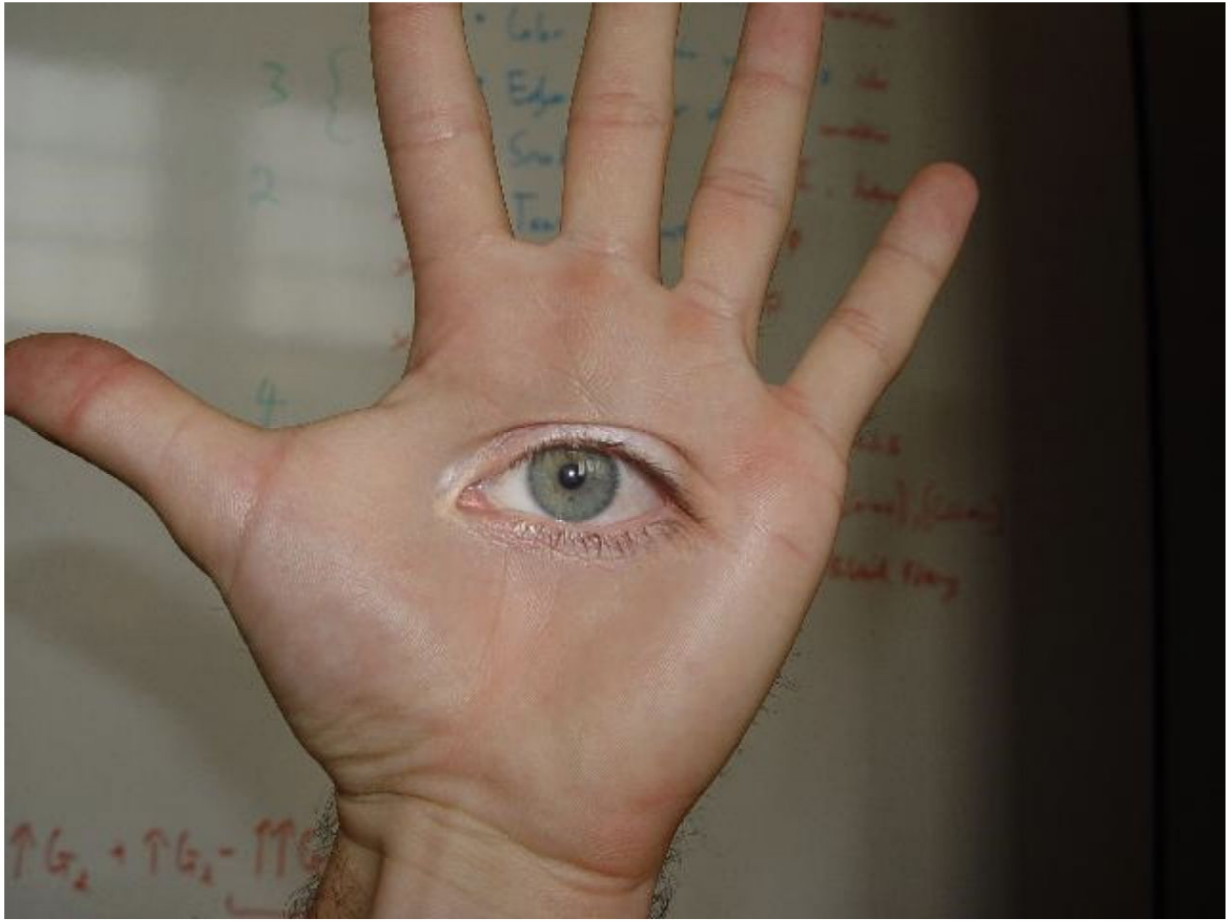


Image Blending with Laplacian Pyramids

- **Input:** 2 images (im1, im2) and a binary mask (M)
- **Output:** a blended image based on the mask

- **Algorithm:**
  - Construct laplacian pyramids for each image: L1, L2.
  - Construct a gaussian pyramid for the mask: Gm
  - Construct another laplacian pyramid where for each level k:
    L3[k] = Gm[k]*L1[k]+(1-Gm[k])*L2[k]
  - Sum all levels in L3 to get the blended image



© prof. dmartin

Newer blending methods

GP-GAN (Wu et al., ACMMM 2019)




Deep Image Blending (Zhang et al., WACV 2020)






Blended Latent Diffusion (Avrahami et al., SIGGRAPH 2023)

Input Image

+

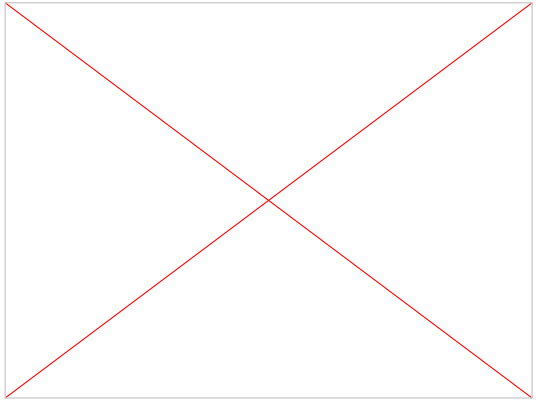
Input Mask

+ “A man with a yellow sweater”

Input Prompt

Result

Split-then-Merge (Kara et al., Nov’ 2025)



GenCompositor (Yang et al., Sept’ 2025)