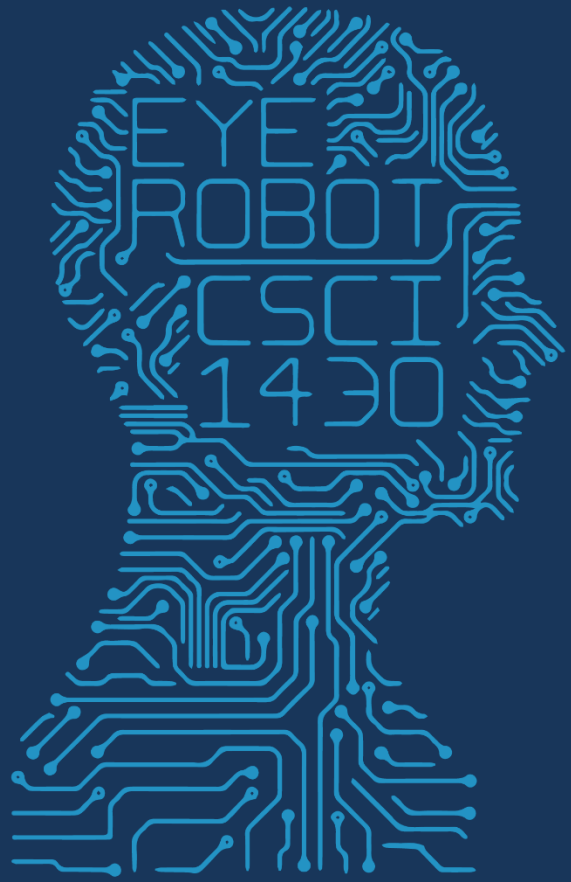
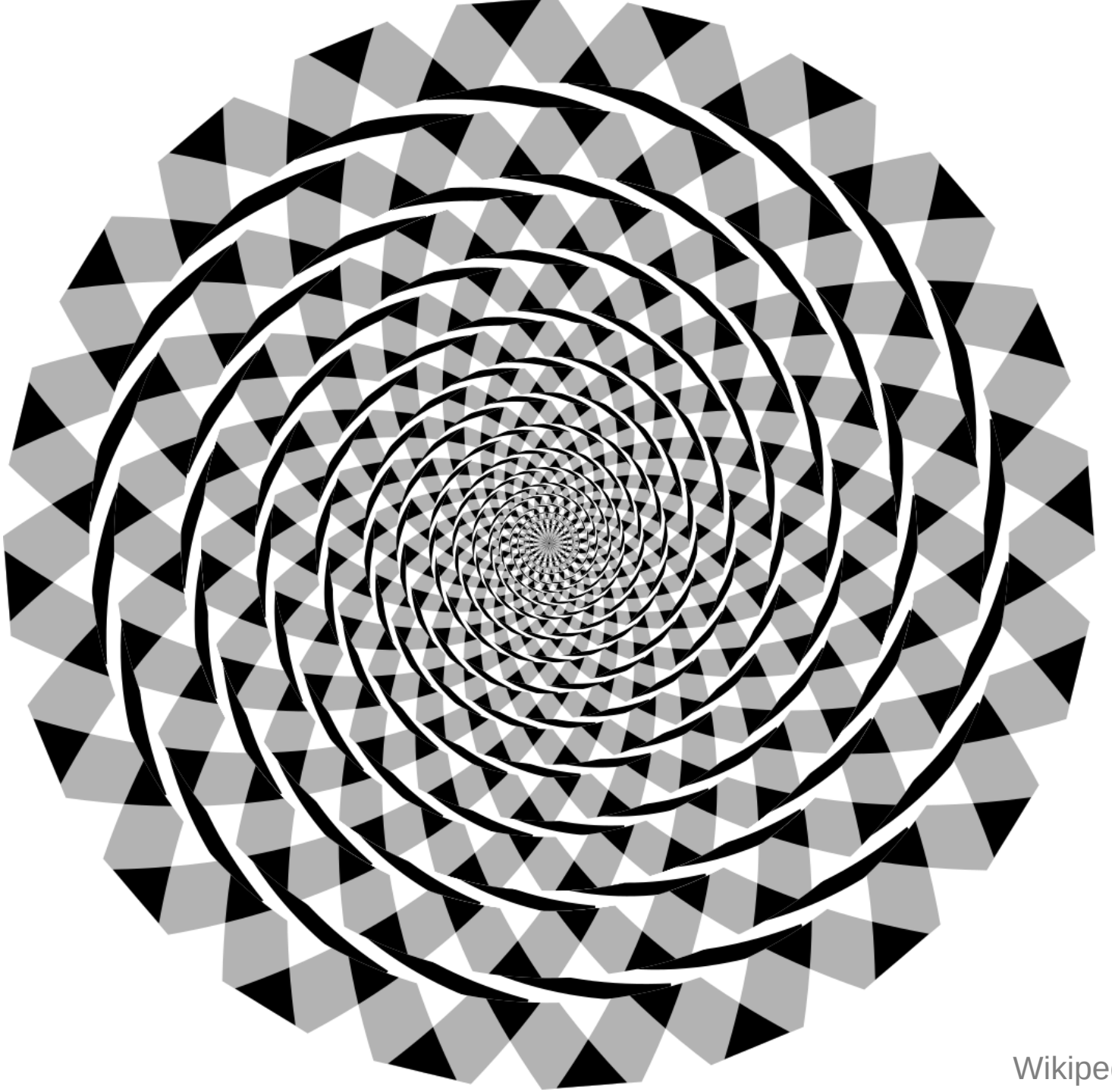
1950

Future Vision

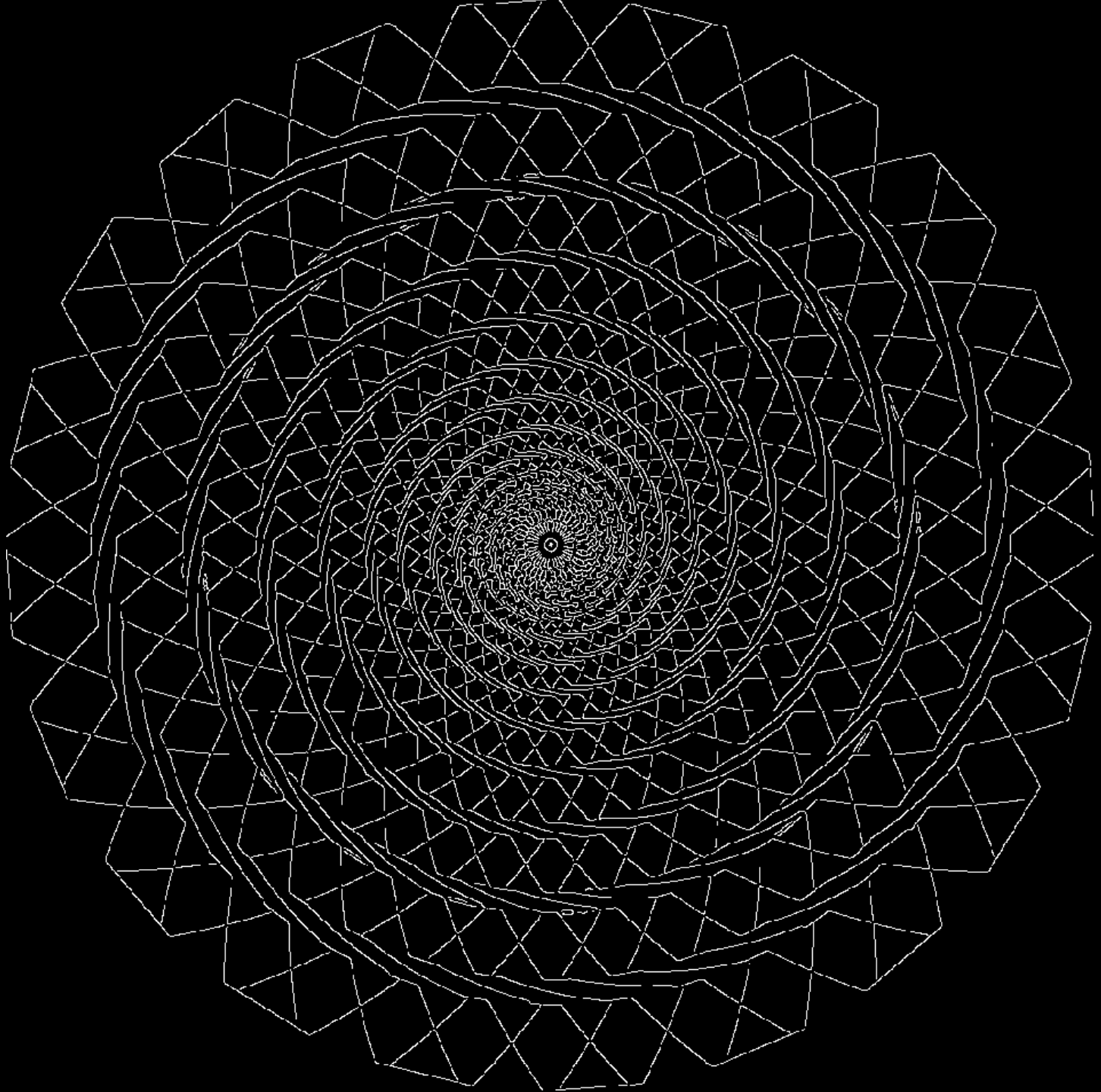


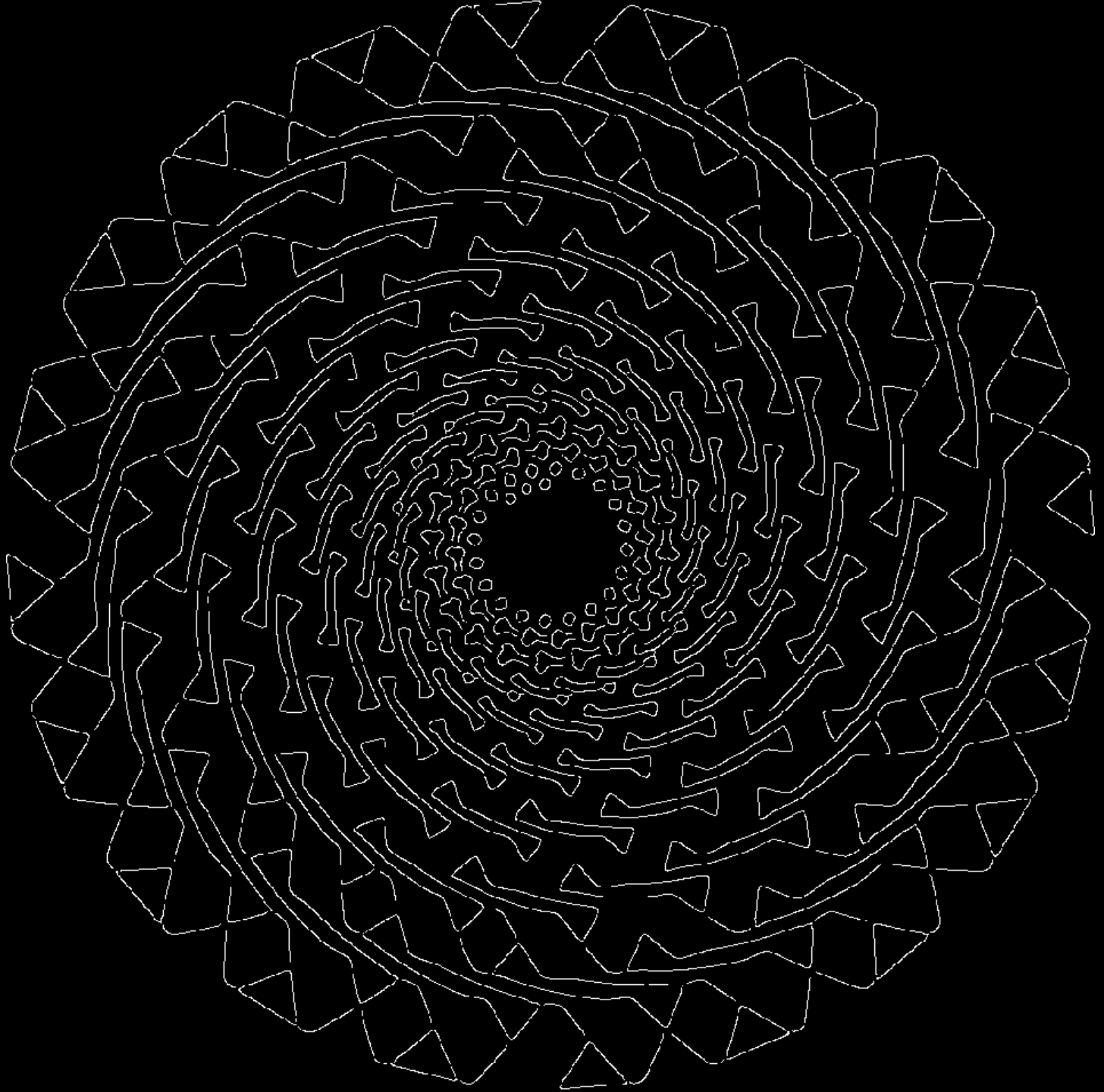
2020

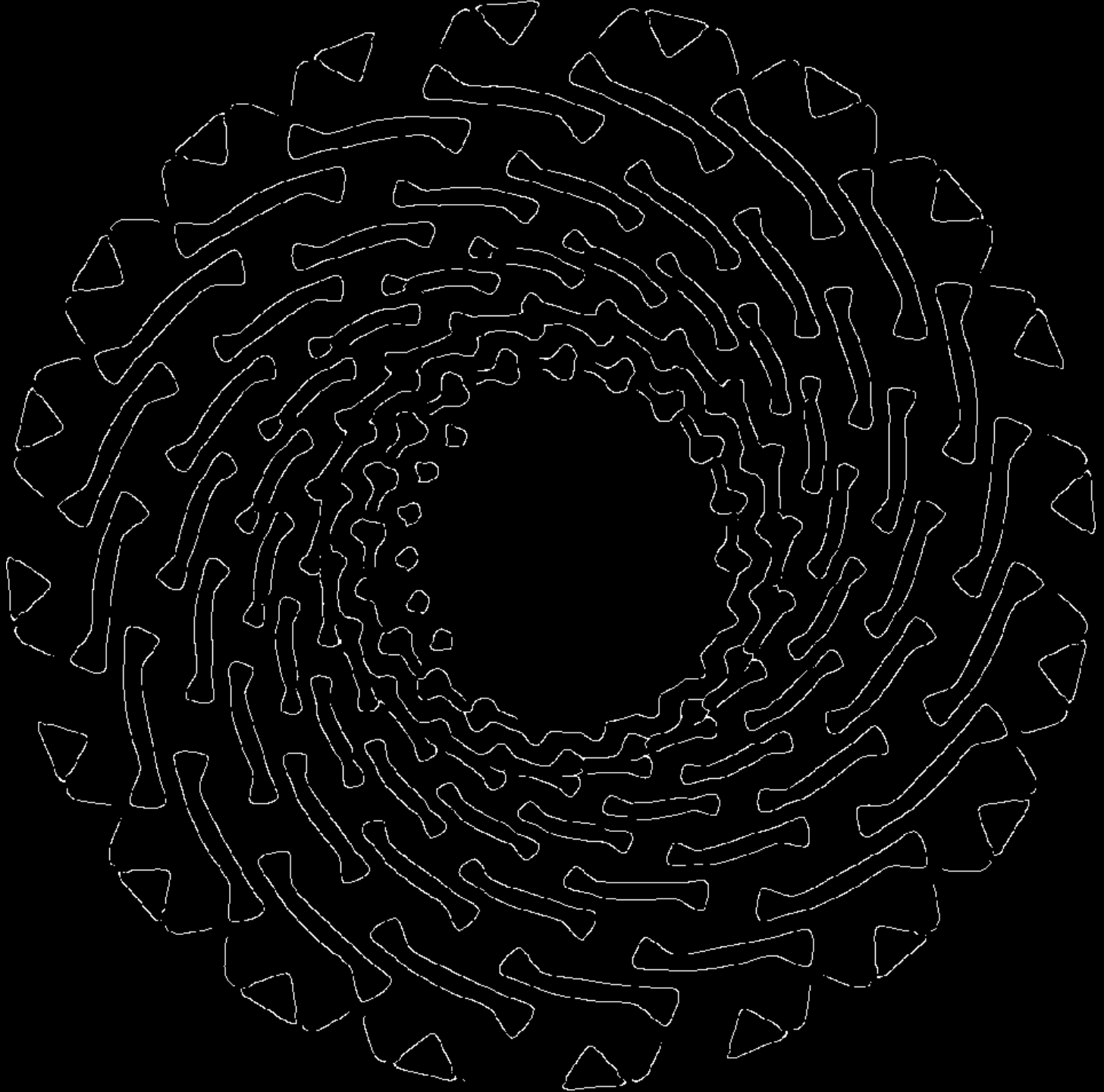
Computer Vision

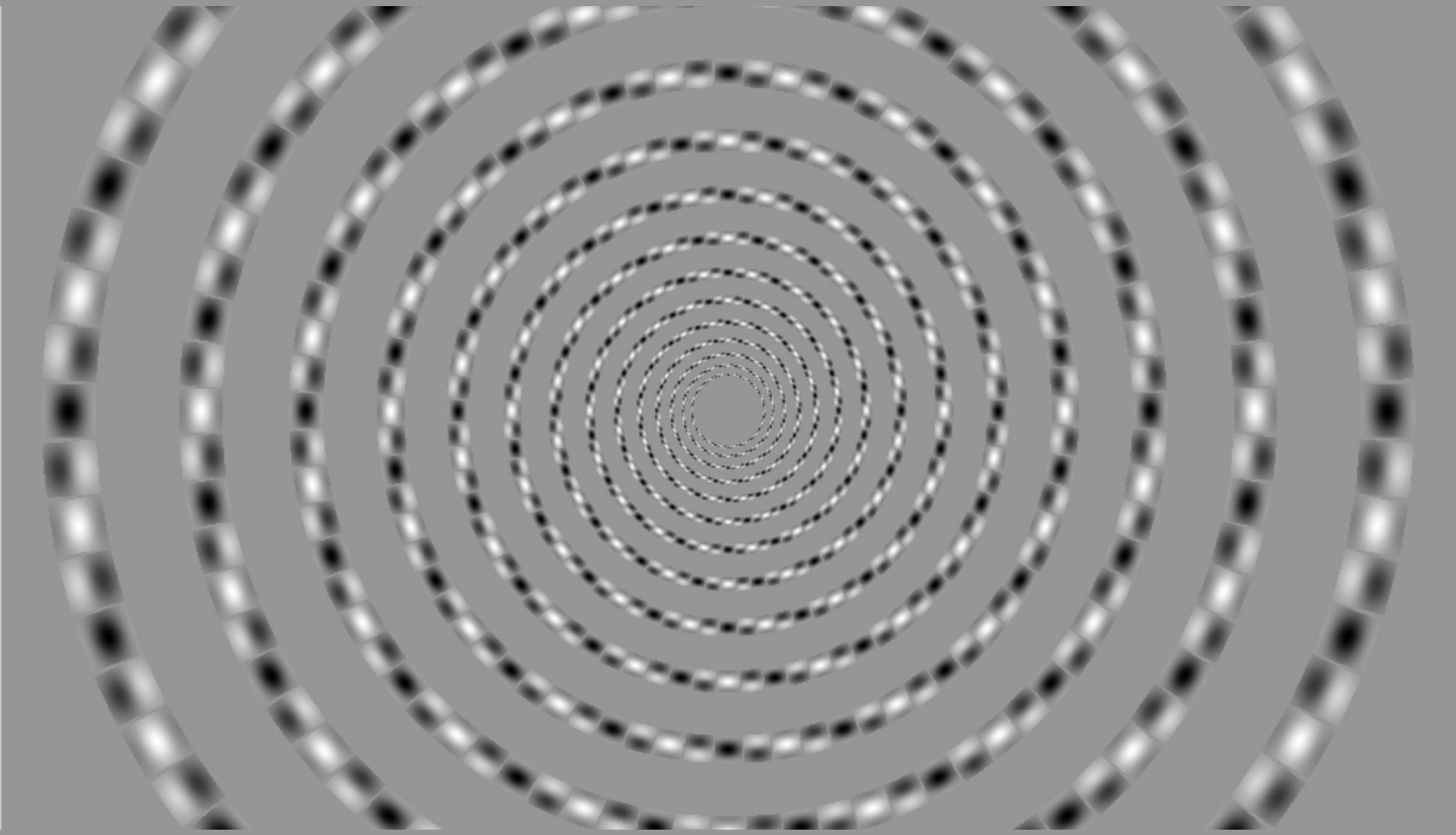


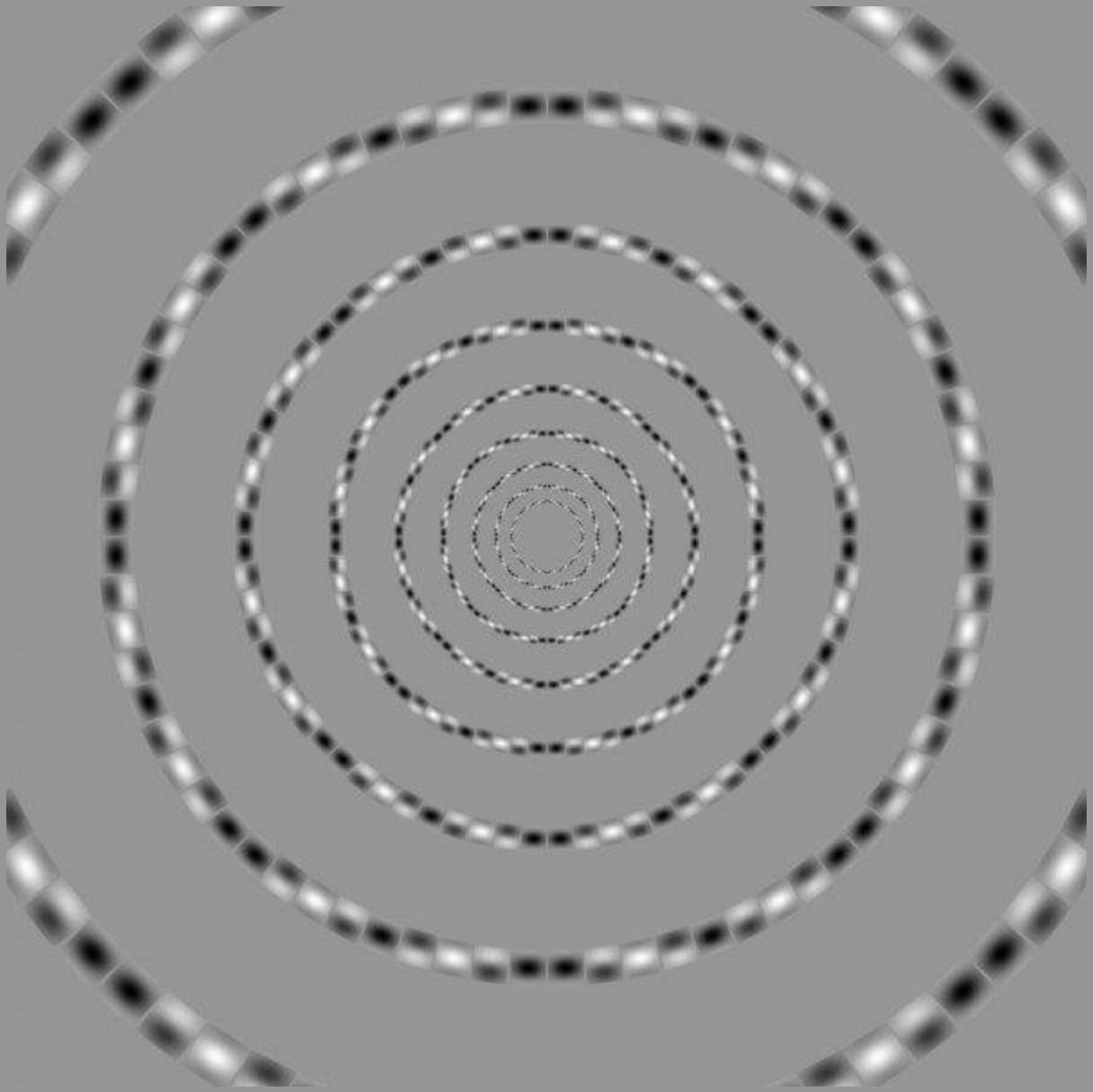
Wikipedia - Mysid



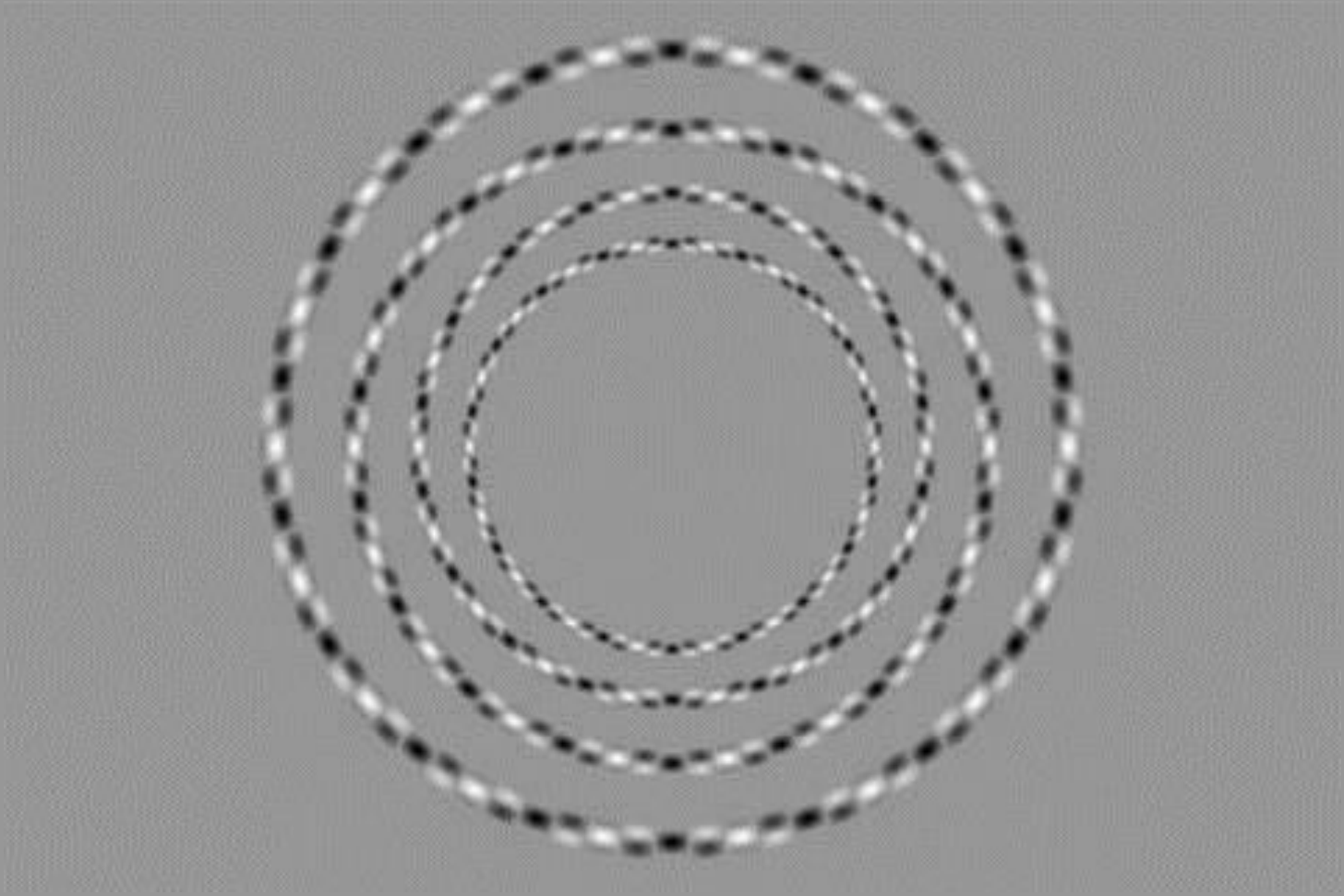


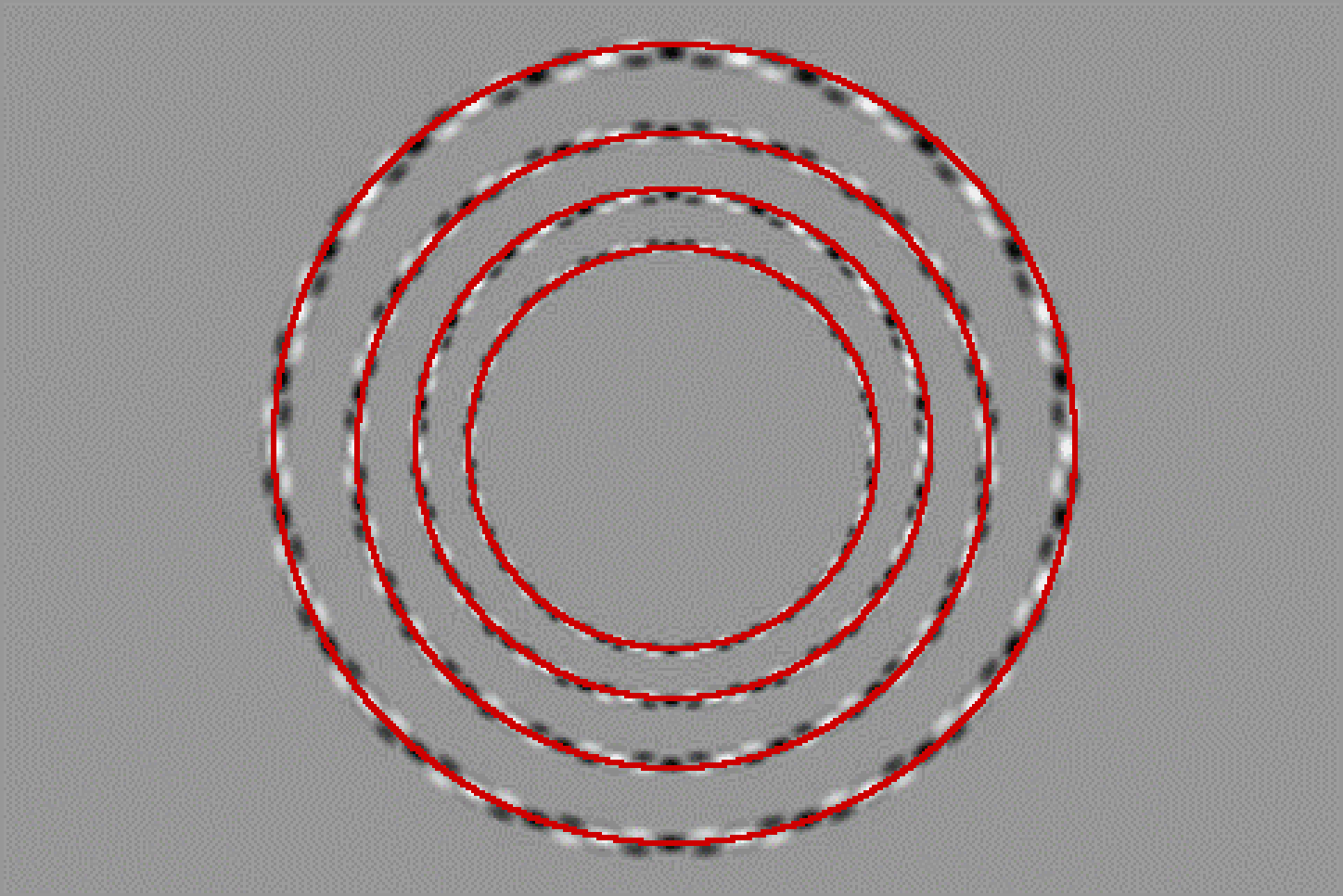


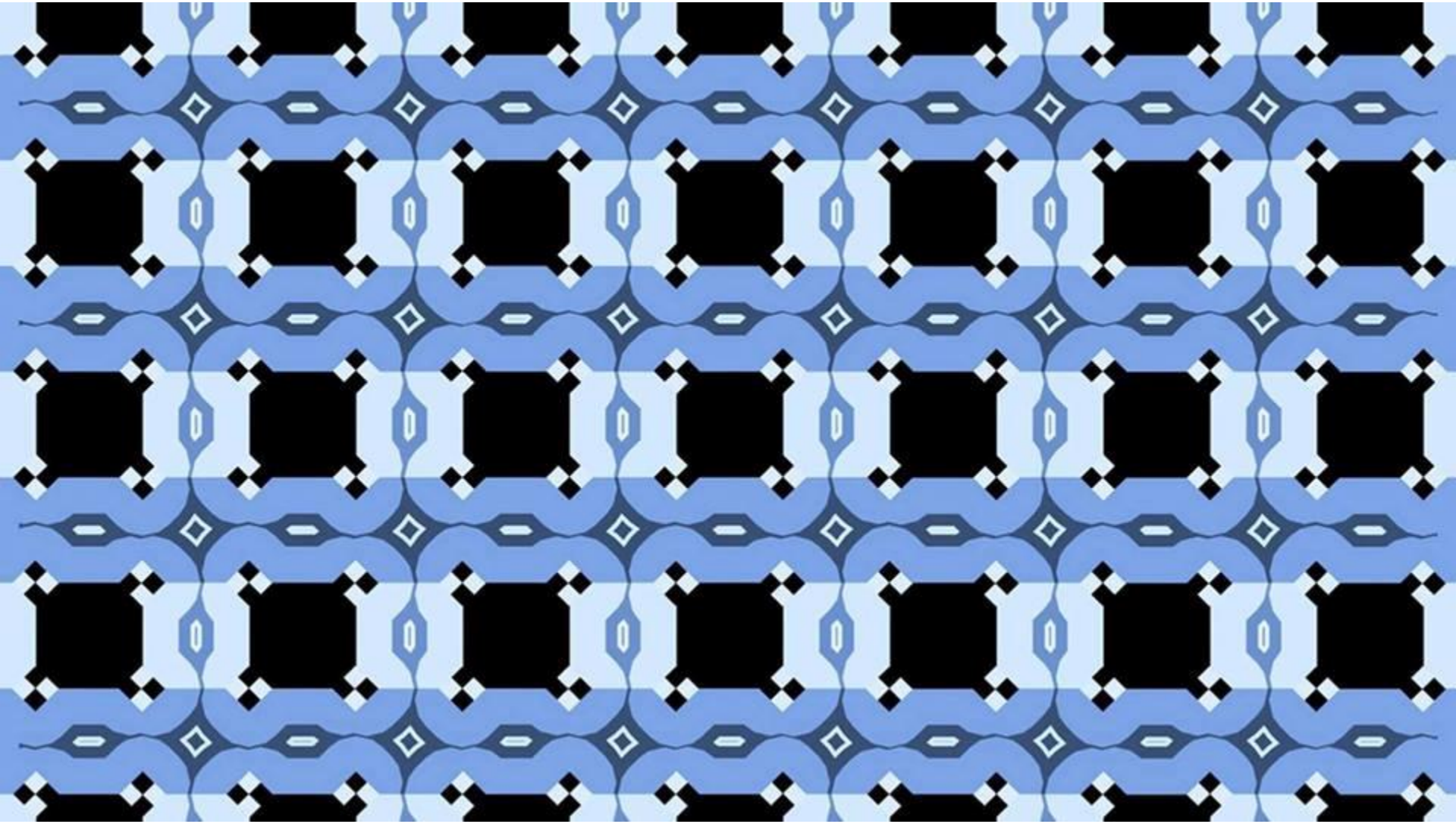




[Akiyoshi Kitaoka, Ritsumeikan University, Kyoto]







Erik Brynjolfsson, MIT

Project 1 Feedback

Thank you all.

- There are no late days on proj0
- Numpy gearup beyond proj0 tutorial – OK
- Short time frame [my fault]
- Axes3D - OK
- Improvements to code description – OK

Project 1 Convolution Speeds

	Convolution speed (seconds)
Jake Chanan	0.91
Alice Marbach	0.94
Andrew Cooke	1.07
Albert Webson	1.2
Reza Esfandiarpoor	1.2
Andrew Levy	1.22
James White	1.23
Da Huo	1.24
Troy Moo Penn	1.27
Michael T Lincoln	1.29

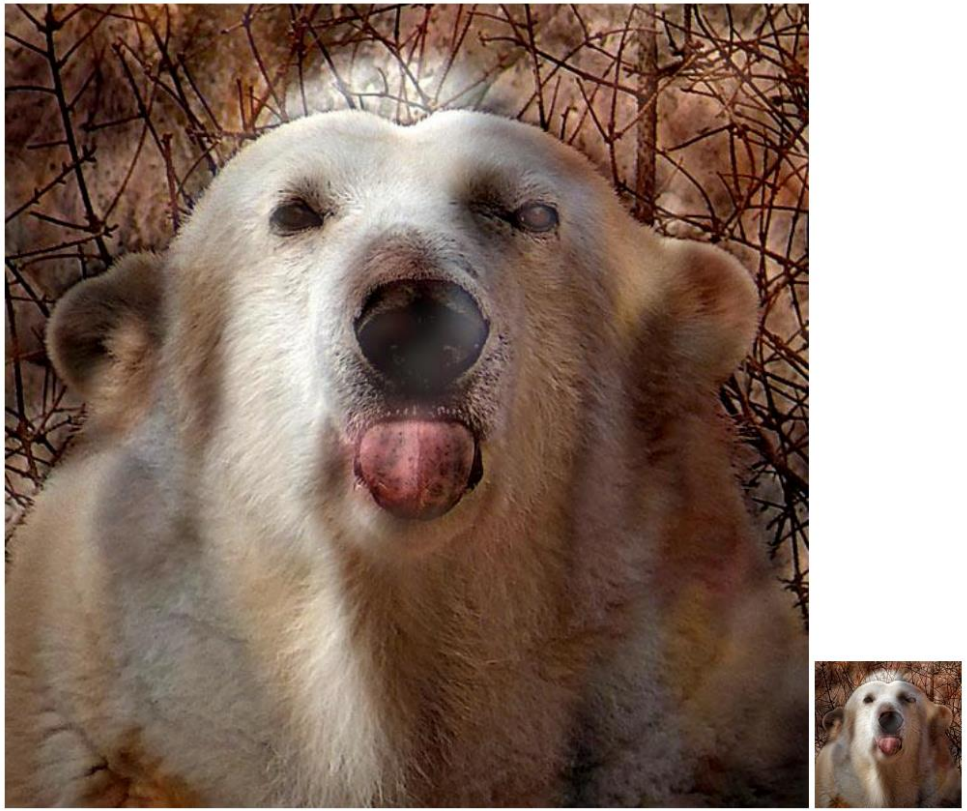
	FFT convolution (seconds)
James Wang	0.45
Matthew Kovoor	0.75
Isaiah Liu	0.85

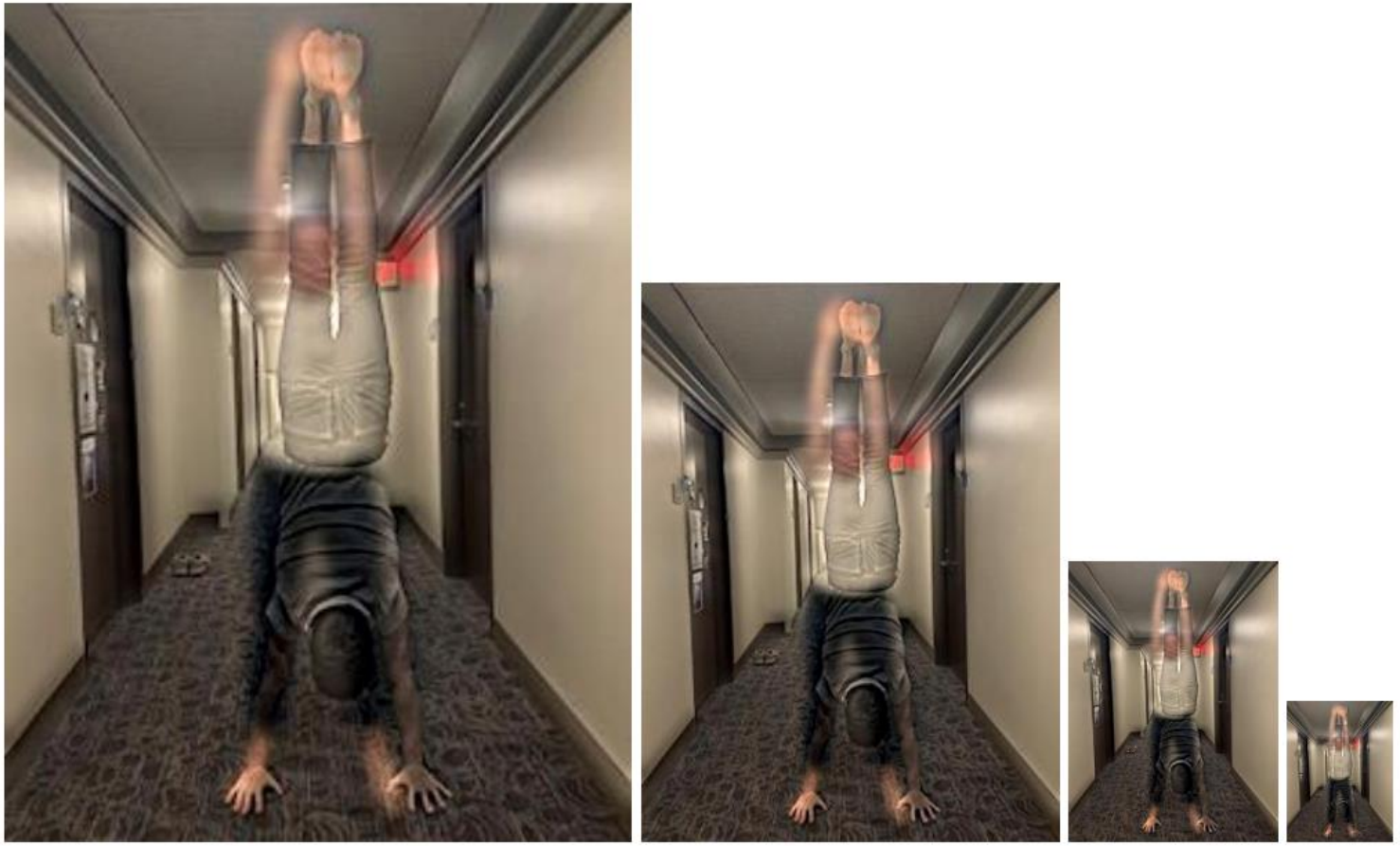
Hybrid images

African wild dog on left, lion profile on right

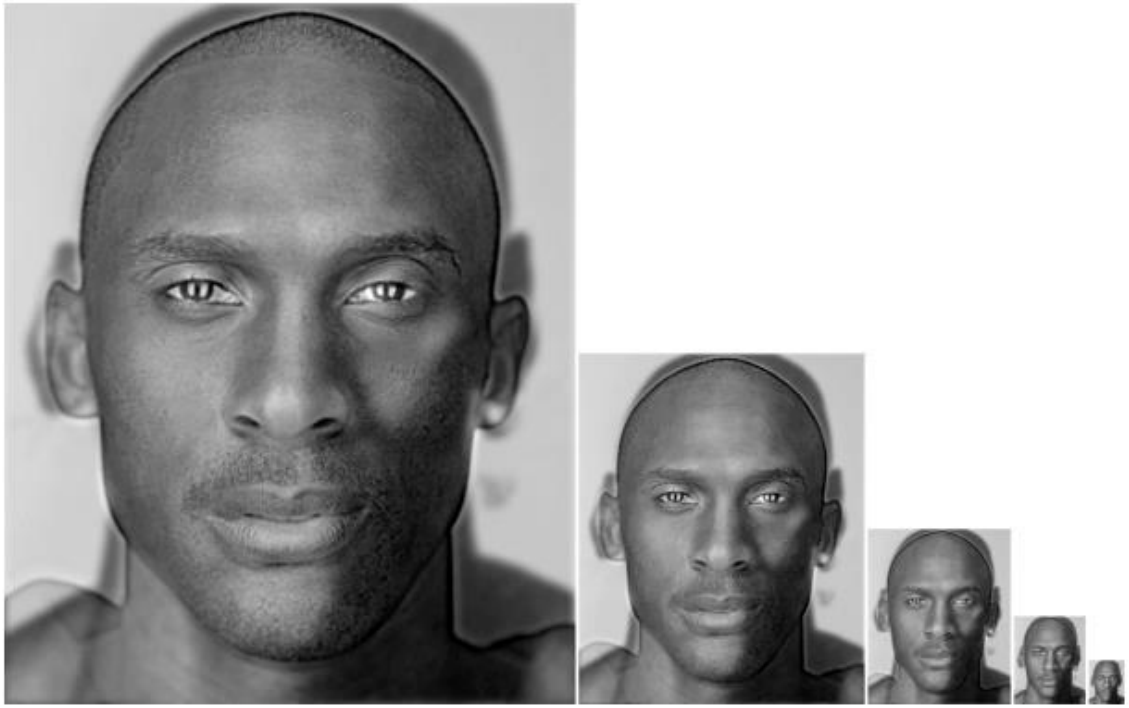
Figure 4: *Left:* Dog in high frequency *Right:* Lion in low frequency

Figure 5: Hybrid Image scales of Lion and African Dog





Thus, I took these two images, merged them, and got this result:



Hybrid Image at Different Scales

Figure 2: *Left:* My Friend, Larry, *Right:* My Other Friend, Jeffrey

1. My own hybrid image, titled **Seeing vs. Believing**

Figure 7: Hybrid image for Starbucks and Wendy's. (cut-off frequency: 4)

And here's the hybrid scales:







Filtering → Edges → Corners

Feature points

Also called interest points, key points, etc.
Often described as ‘local’ features.

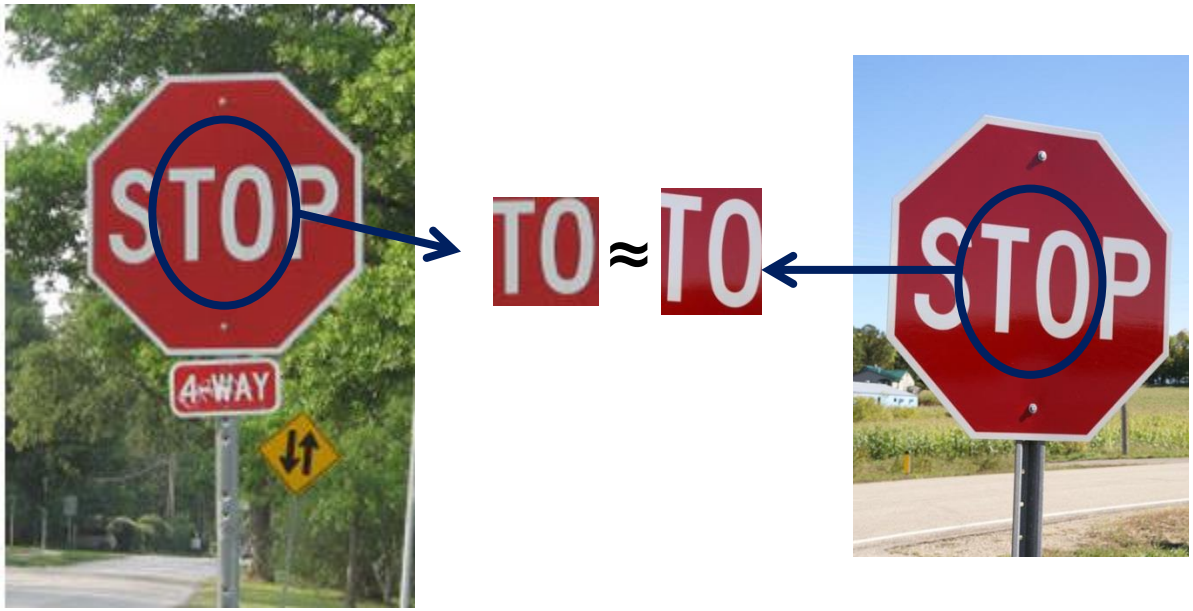
Szeliski 4.1

Correspondence across views

Matching points, patches, edges,
or regions across images.

Sparse or local correspondence vs.

dense correspondence (at every pixel).



Fundamental to Applications

- Image alignment
- 3D reconstruction
- Motion tracking (robots, drones, AR)
- Indexing and database retrieval
- Object recognition

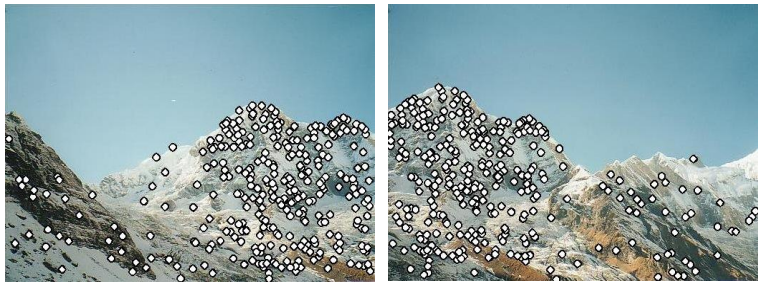
Hays

Example application: Panorama stitching

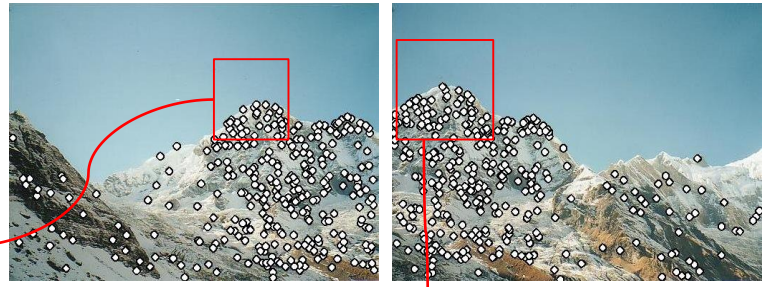
We have two images –
how do we estimate how to overlay them?

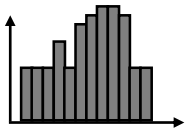
Local features: main components

1) Detection:
Find a set of distinctive key points.

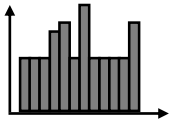


2) Description:
Extract feature descriptor around each interest point as vector.



x_1 

$$\mathbf{x}_1 = [x_1^{(1)}, \ldots, x_d^{(1)}]$$

x_2 

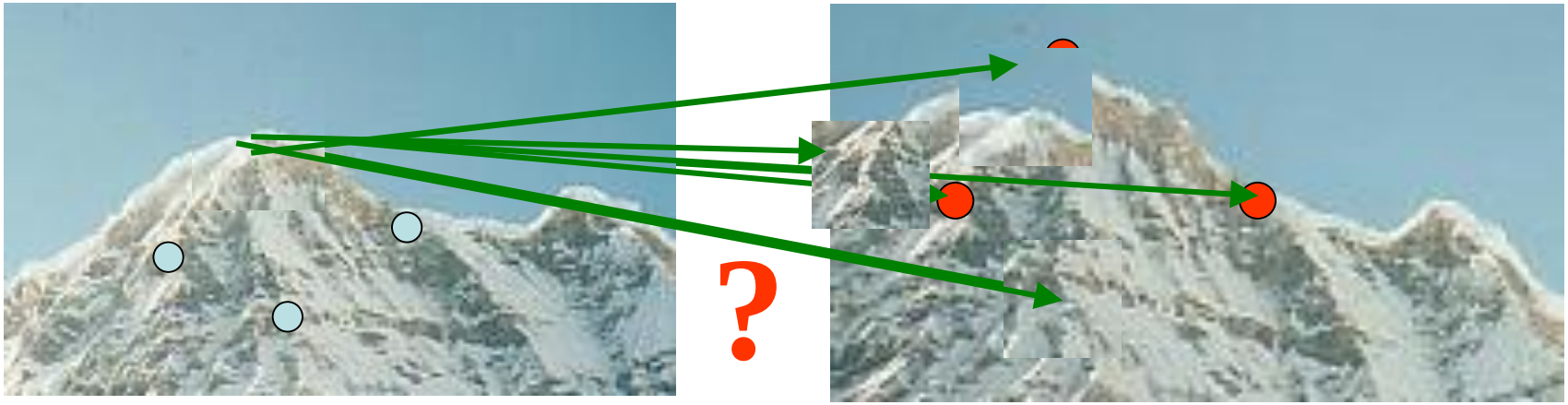
$$\mathbf{x}_2 = [x_1^{(2)}, \ldots, x_d^{(2)}]$$

3) Matching:
Compute distance between feature vectors to find correspondence.

$$d(\mathrm{x}_1, \mathrm{x}_2) < T$$

Goal: Distinctiveness

We want to be able to reliably determine which point goes with which.



May be difficult in structured environments with repeated elements

Goal: Repeatability

We want to detect (at least some of) the same points in both images.

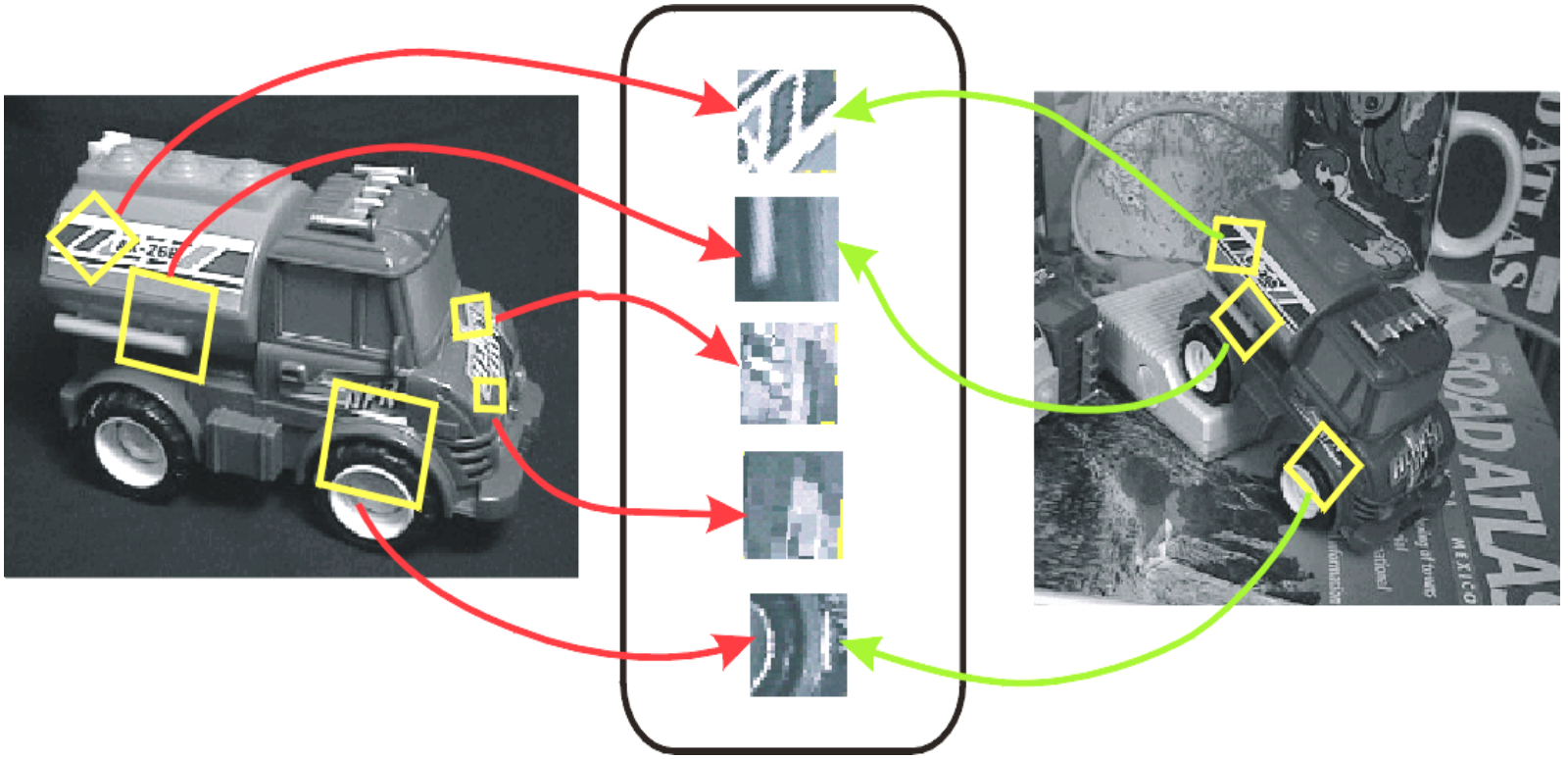
With these points, there's no chance to find true matches!

Under geometric and photometric variations.

Example: Object Detection

Finding *distinctive* and *repeatable* feature points can be difficult when we want our features to be invariant to large transformations:

- geometric variation (translation, rotation, scale, perspective)
- appearance variation (reflectance, illumination)



Keypoint Descriptors

Goal: Compactness and Efficiency

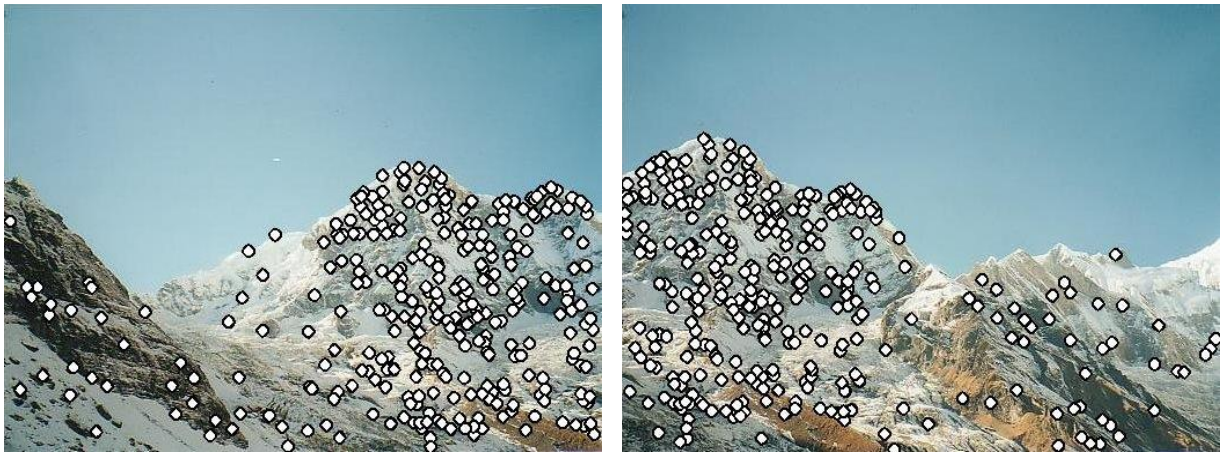
We want the representation to be as small and as fast as possible

- Much smaller than a whole image

We'd like to be able to run the detection procedure *independently* per image

- Match just the compact descriptors for speed.
- *Difficult!* We don't get to see 'the other image' at match time, e.g., object detection.

Characteristics of good features



Distinctiveness

Each feature can be uniquely identified

Repeatability

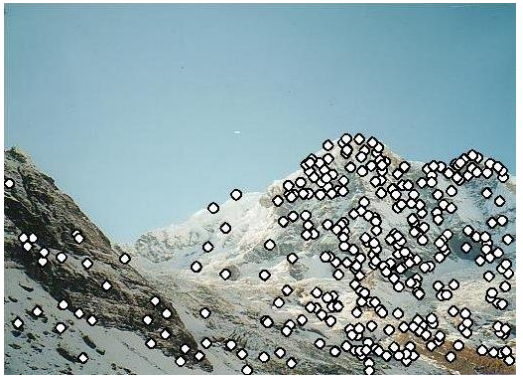
The same feature can be found in several images despite differences:
- geometrically (translation, rotation, scale, perspective)
- photometrically (reflectance, illumination)

Compactness and efficiency

Many fewer features than image pixels; run independently per image

Local features: main components

1) Detection:
   Find a set of distinctive key points.



2) Description:
   Extract feature descriptor around each interest point as vector.

3) Matching:
   Compute distance between feature vectors to find correspondence.

Detection: Basic Idea

We do not know which other image locations the feature will end up being matched against.

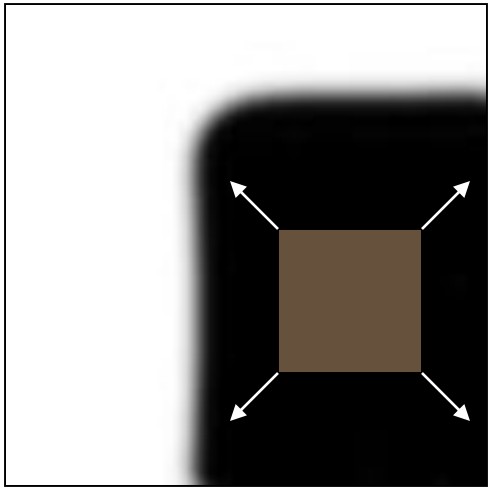
But we can compute how stable a location is in appearance with respect to small variations in its position.

Strategy: Compare image patch against local neighbors.

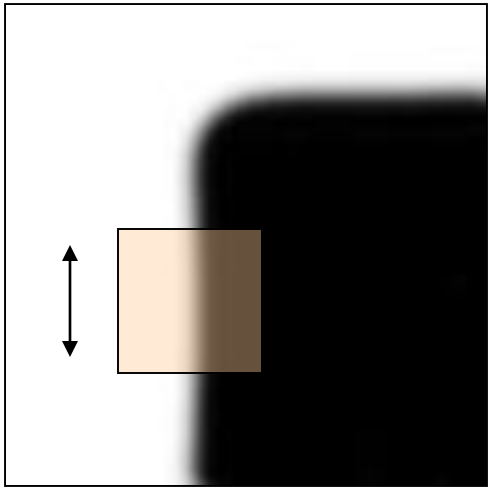
Corner Detection: Basic Idea

Recognize corners by looking at small window.

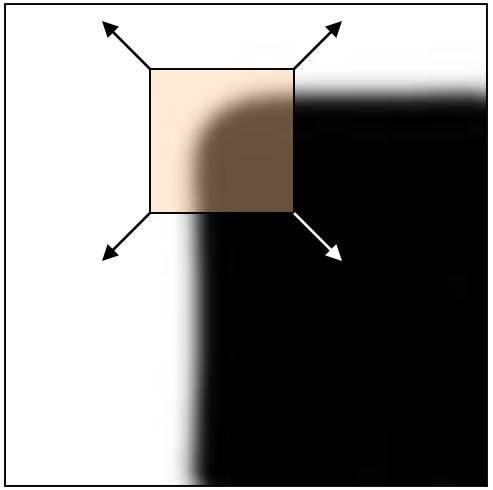
We want a window shift in *any direction* to give *a large change* in intensity.



"Flat" region:
no change in
all directions



"Edge":
no change
along the edge
direction



"Corner":
significant
change in all
directions

Corner Detection by Auto-correlation

Change in appearance of window $w(x,y)$ for shift $[u,v]$:

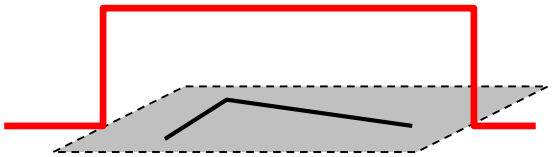
$$E(u,v) = \sum_{x,y} w(x,y)\left[I(x+u,y+v) - I(x,y)\right]^2$$

Window function

Shifted intensity

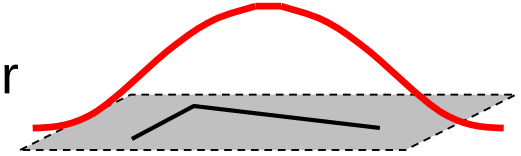
Intensity

Window function $w(x,y)$ =



1 in window, 0 outside

or

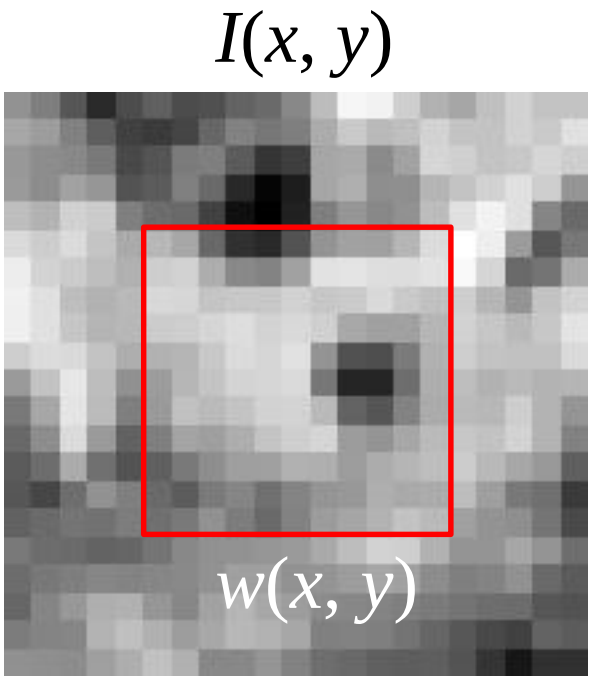


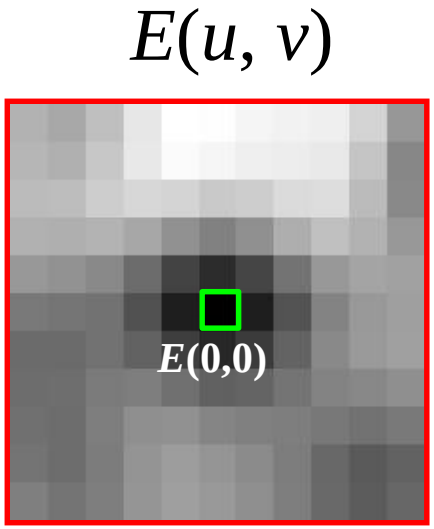
Gaussian

Corner Detection by Auto-correlation

Change in appearance of window $w(x,y)$ for shift $[u,v]$:

$$E(u,v) = \sum_{x,y} w(x,y)\left[I(x+u, y+v) - I(x,y)\right]^2$$

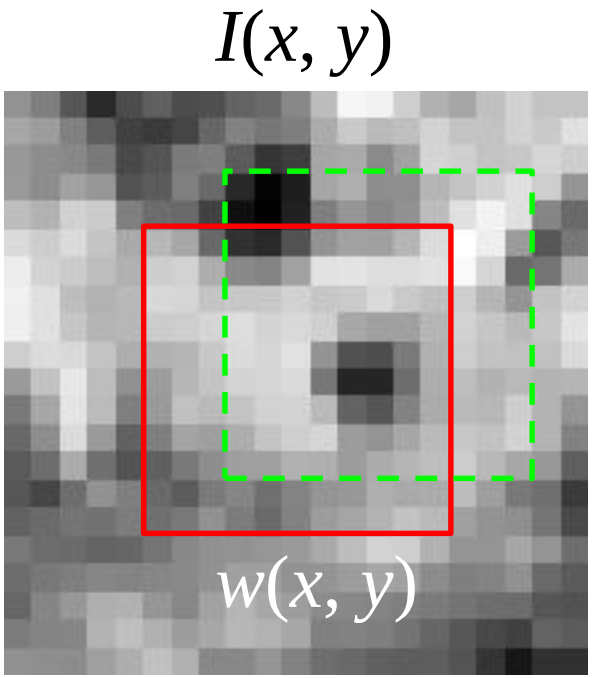


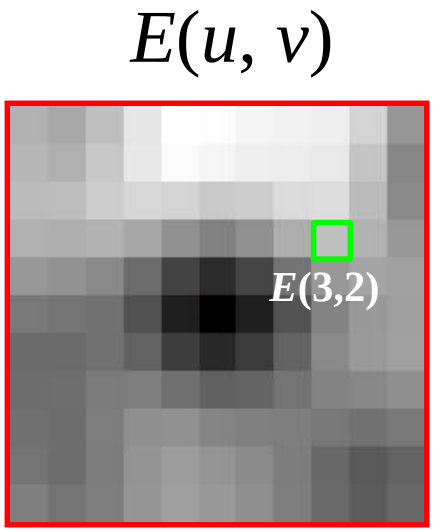


Corner Detection by Auto-correlation

Change in appearance of window *w*(*x*,*y*) for shift [*u*,*v*]:

$$E(u,v) = \sum_{x,y} w(x,y)\left[I(x+u,y+v) - I(x,y)\right]^2$$



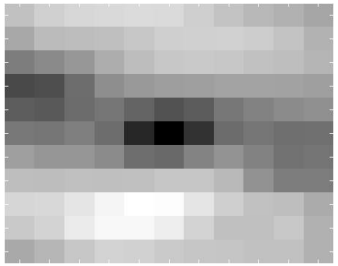


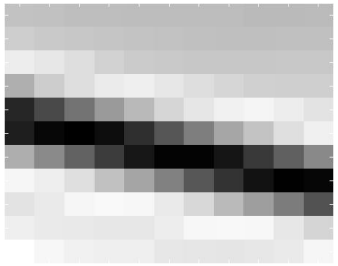
$$E(u,v) = \sum_{x,y} w(x,y)\left[I(x+u, y+v) - I(x,y)\right]^2$$

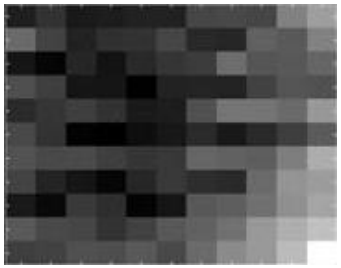
Think-Pair-Share:

Correspond the three red crosses to (b,c,d).

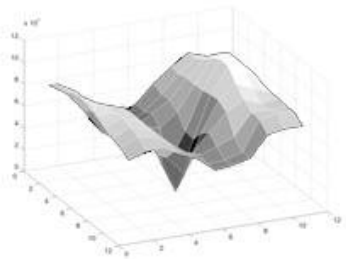
$E(u,v)$

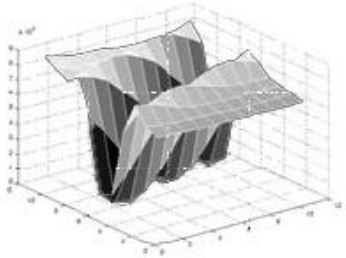


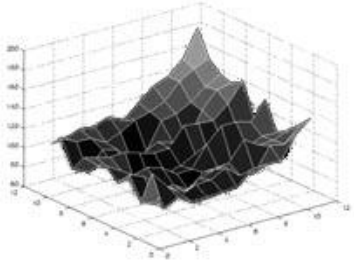




$E(u,v)$







As a surface

Corner Detection by Auto-correlation

Change in appearance of window $w(x,y)$ for shift $[u,v]$:

$$E(u,v) = \sum_{x,y} w(x,y)\left[I(x+u,y+v) - I(x,y)\right]^2$$

We want to discover how E behaves for small shifts

But this is very slow to compute naively.
O(window_width^2 * shift_range^2 * image_width^2)

O(11^2 * 11^2 * 600^2) = 5.2 billion of these
14.6k ops per image pixel

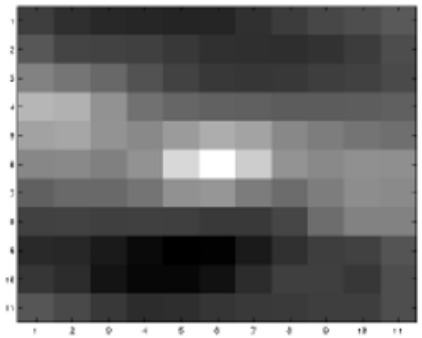
Corner Detection by Auto-correlation

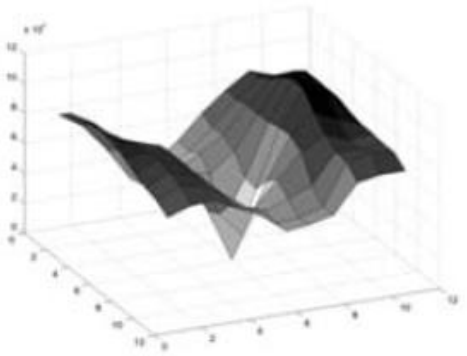
Change in appearance of window $w(x,y)$ for shift $[u,v]$:

$$E(u,v) = \sum_{x,y} w(x,y)\left[I(x+u,y+v) - I(x,y)\right]^2$$

We want to discover how E behaves for small shifts

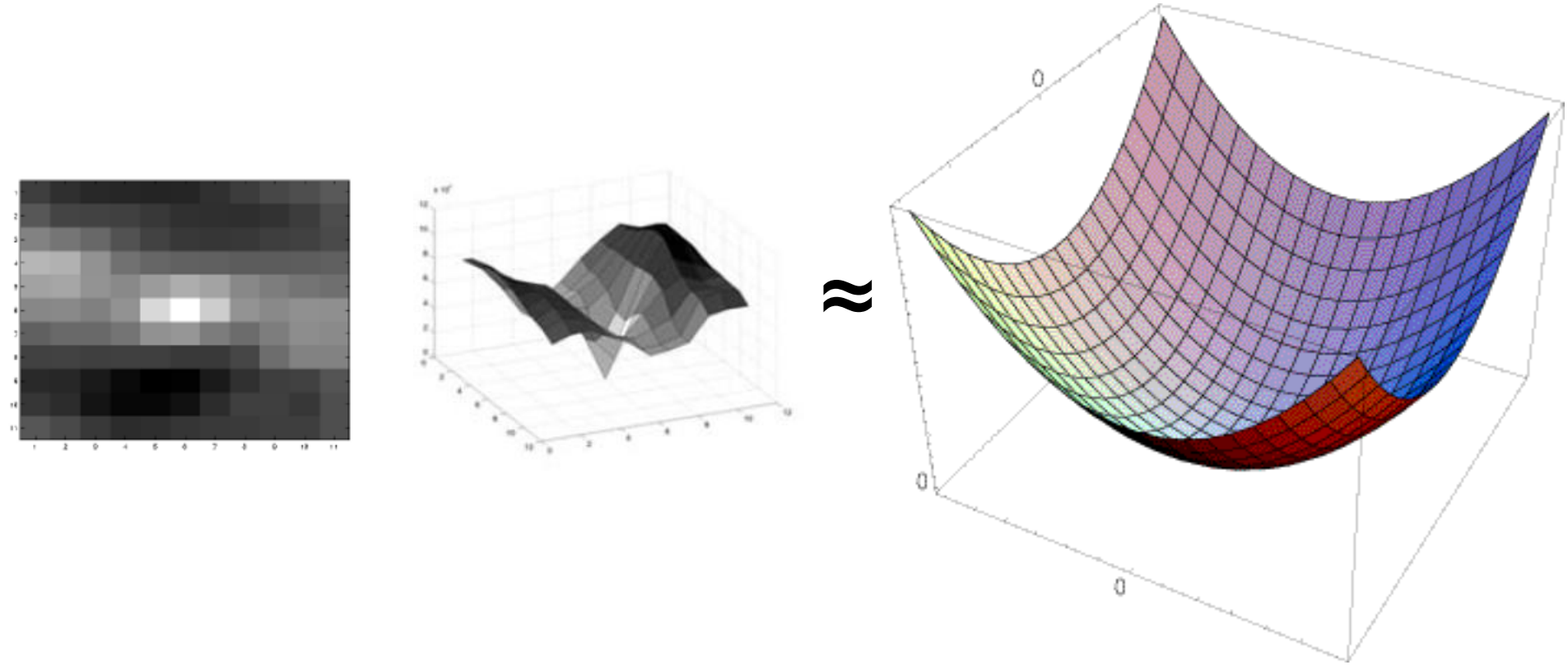
But we know the response in *E* that we are looking for – strong peak.





Strategy:

Approximate $E(u,v)$ locally by a quadratic surface, and look for that instead.



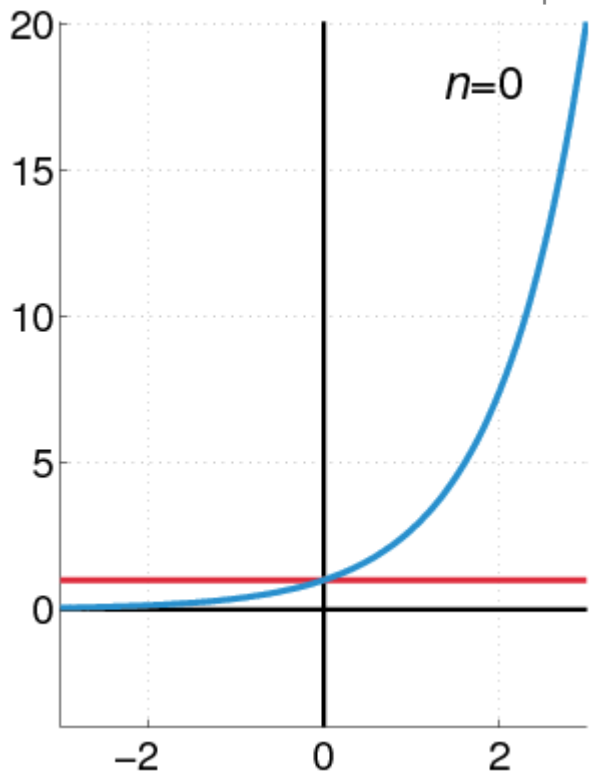
Recall: Taylor series expansion

A function f can be represented by an infinite series of its derivatives at a single point *a*:

$$f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \cdots.$$

As we care about window centered, we set *a = 0* (MacLaurin series)

Wikipedia



Approximation of
$f(x) = e^x$
centered at f(0)

Local quadratic approximation of $E(u,v)$ in the neighborhood of (0,0) is given by the *second-order Taylor expansion*:

$$E(u,v) \approx E(0,0) + \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} E_u(0,0) \\ E_v(0,0) \end{bmatrix} + \frac{1}{2} \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} E_{uu}(0,0) & E_{uv}(0,0) \\ E_{uv}(0,0) & E_{vv}(0,0) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

Notation: partial derivative

Local quadratic approximation of $E(u,v)$ in the neighborhood of (0,0) is given by the *second-order Taylor expansion*:

$$E(u,v) \approx E(0,0) + [u \;\; v]\begin{bmatrix} E_u(0,0) \\ E_v(0,0) \end{bmatrix} + \frac{1}{2}[u \;\; v]\begin{bmatrix} E_{uu}(0,0) & E_{uv}(0,0) \\ E_{uv}(0,0) & E_{vv}(0,0) \end{bmatrix}\begin{bmatrix} u \\ v \end{bmatrix}$$

Ignore function value; set to 0

Ignore first derivative, set to 0

Just look at shape of second derivative (2D quadratic surface)

Corner Detection: Mathematics

The quadratic approximation simplifies to

$$E(u,v) \approx [u \;\; v] \begin{bmatrix} E_{uu}(0,0) & E_{uv}(0,0) \\ E_{uv}(0,0) & E_{vv}(0,0) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

$$E(u,v) \approx [u \;\; v] \; M \begin{bmatrix} u \\ v \end{bmatrix}$$

where *M* is a *second moment matrix* computed from image derivatives:

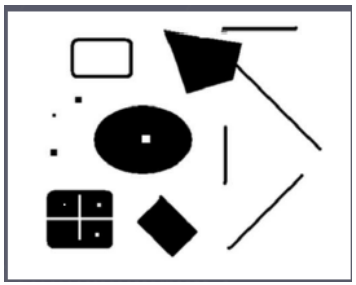
$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

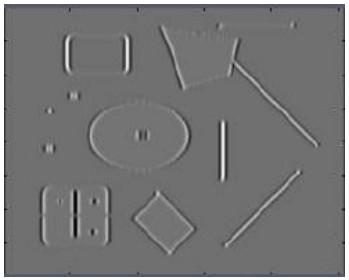
$$M = \begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} = \sum \begin{bmatrix} I_x \\ I_y \end{bmatrix} [I_x \; I_y] = \sum \nabla I (\nabla I)^T$$

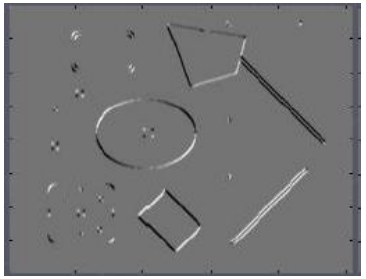
Corners as distinctive interest points

$$M = \sum w(x,y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix}$$

2 x 2 matrix of image derivatives
(averaged in neighborhood of a point)







Notation:

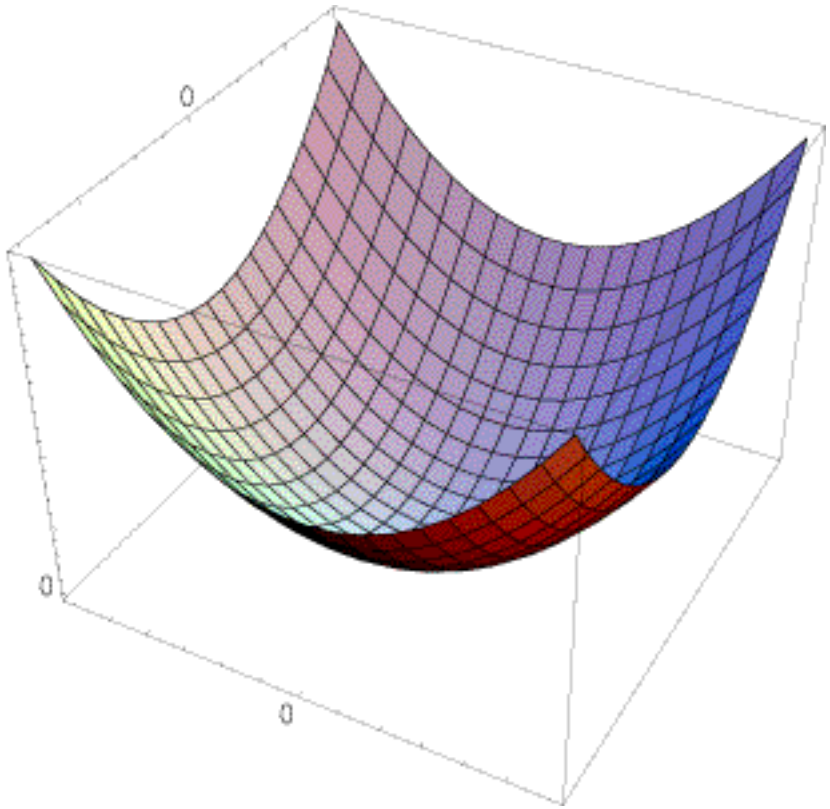
$$I_x \Leftrightarrow \frac{\partial I}{\partial x} \qquad I_y \Leftrightarrow \frac{\partial I}{\partial y} \qquad I_x I_y \Leftrightarrow \frac{\partial I}{\partial x}\frac{\partial I}{\partial y}$$

Interpreting the second moment matrix

$E(u,v)$ is locally approximated by a quadratic surface. Let's try to understand how its shape relates to M.

$$E(u,v) \approx [u \;\; v] \; M \begin{bmatrix} u \\ v \end{bmatrix}$$

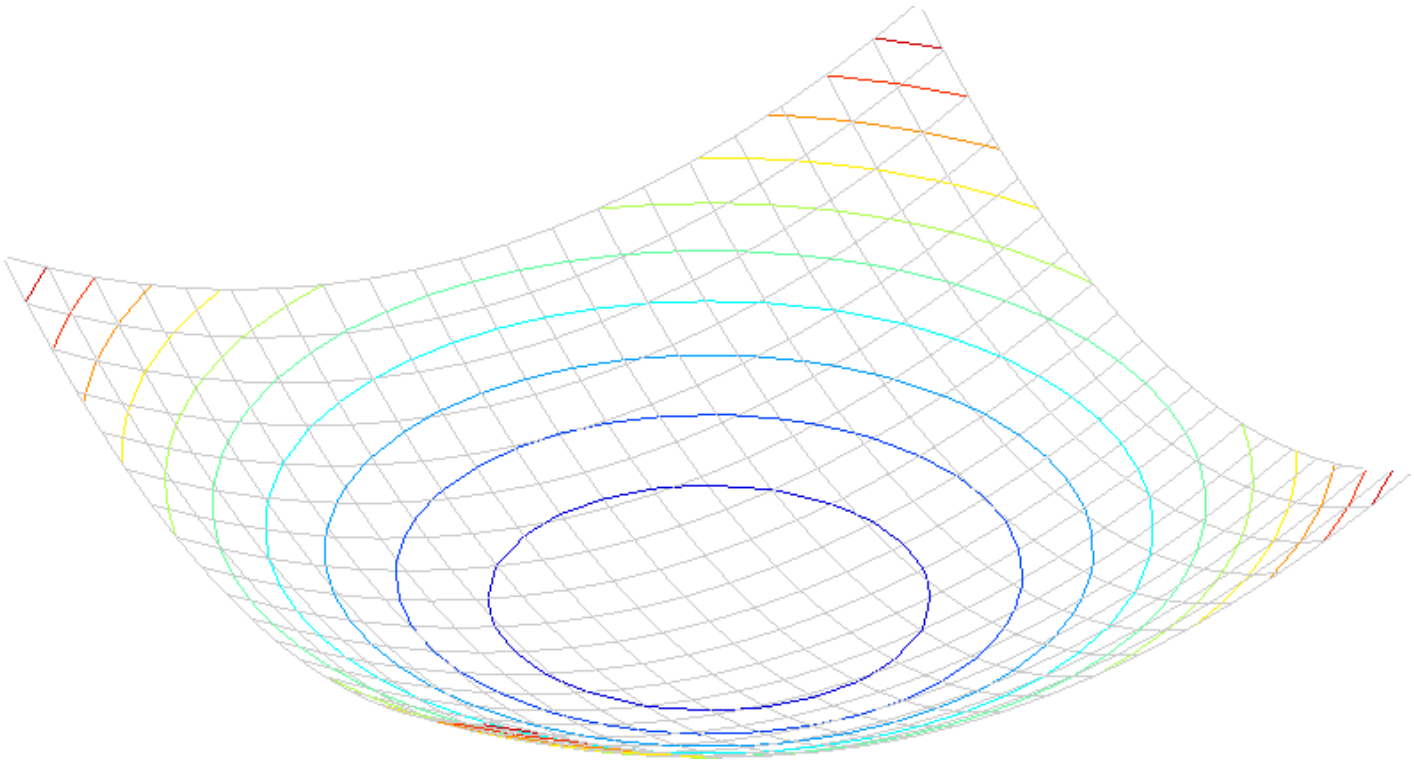
$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$



Interpreting the second moment matrix

Let's take horizontal "slices" of our approximation of $E(u, v)$:

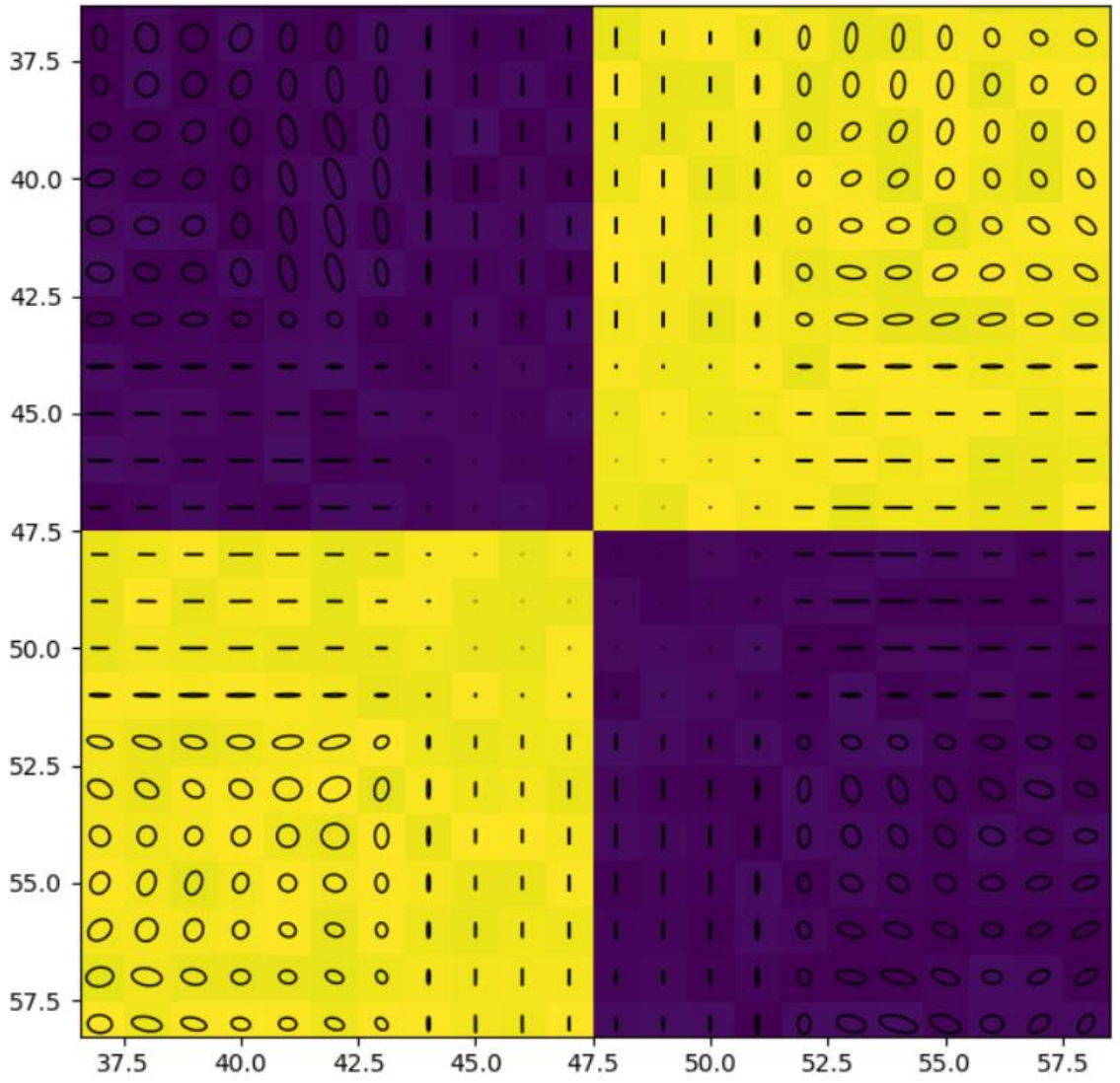
Each coloured line in the diagram below is where $\begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix} = \text{constant}$



Visualization of second moment matrix

Simple image:
Checkerboard.

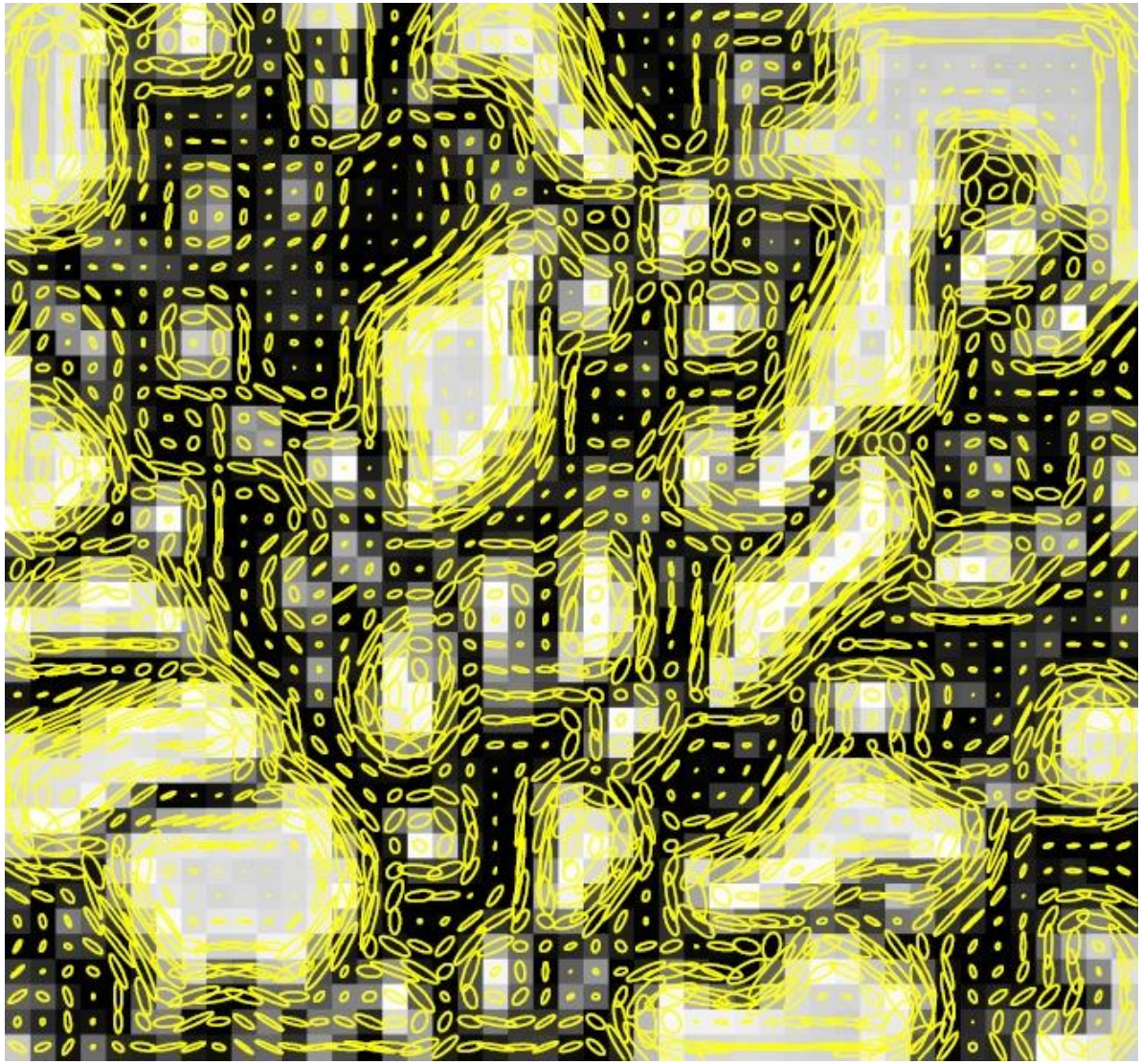
1 corner
4 edges
4 flat regions



Note: Edges show 'wide' response because image derivatives were blurred by Gaussian $\sigma = 1$ before visualizing.

Visualization of second moment matrices

Visualization of second moment matrices



For cornerness, we only care about the ‘steepness’, not the rotation. Can we ignore this somehow?

Linear algebra review

M is symmetric. Symmetric matrices have orthogonal eigenvectors (i.e., a basis).

M is square. Square matrices are diagonalizable if some matrix P exists s.t. $M = P^{-1}AP$, where A has only diagonal entries and P represents a change of basis (in 2D, a rotation).

Interpreting the second moment matrix

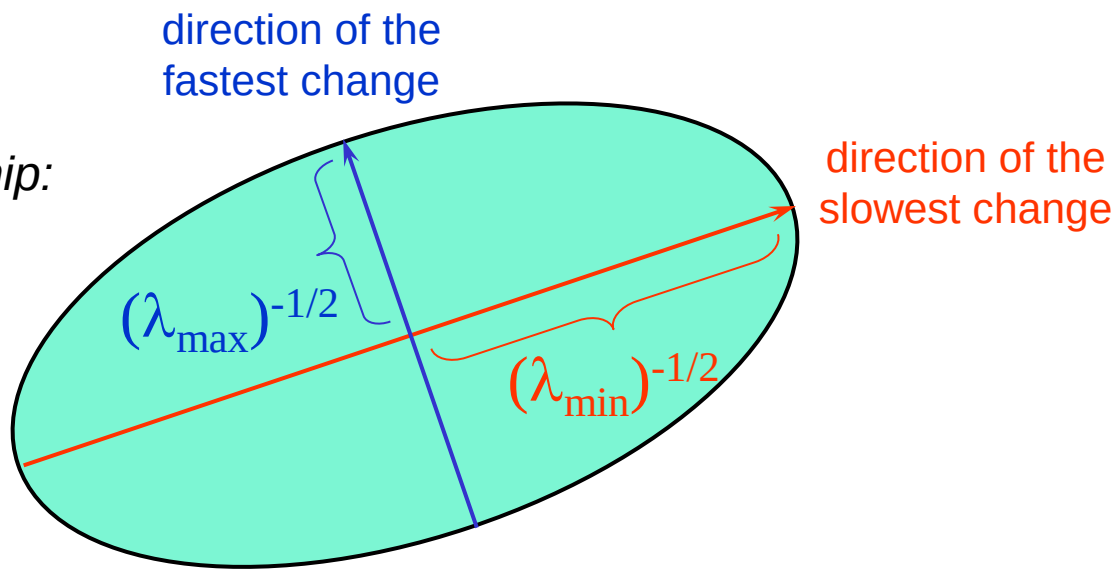
Consider a horizontal "slice" of $E(u, v)$: $\begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix} = \text{const}$

This defines an ellipse.

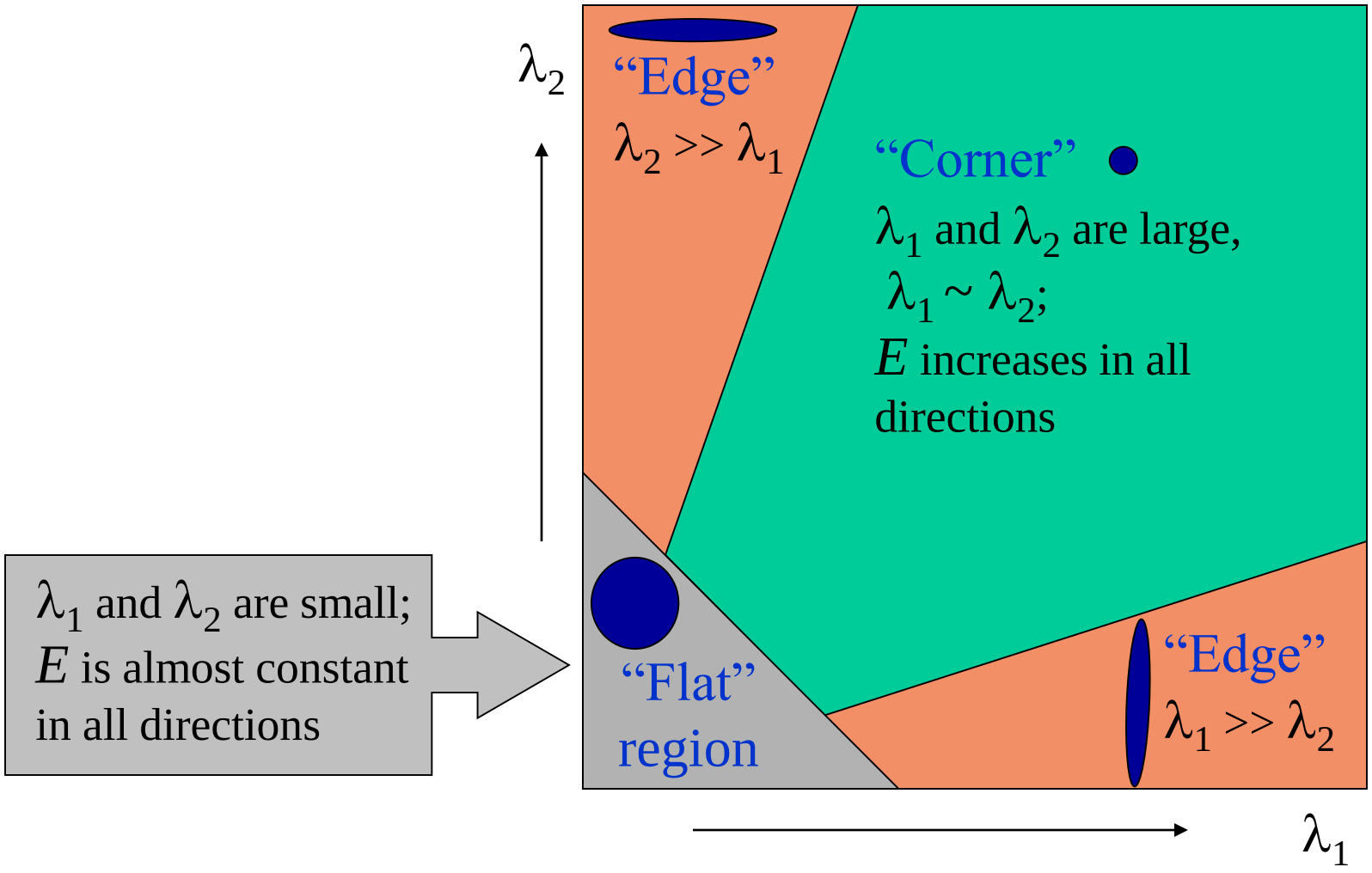
Diagonalization of M: $M = R^{-1} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R$ (A)

The axis lengths of the ellipse are determined by the eigenvalues, and the orientation is determined by a rotation matrix R.

Note inverse relationship: larger eigenvalue = steeper slope; smaller ellipse in visualization



Classification of image points using eigenvalues of *M*

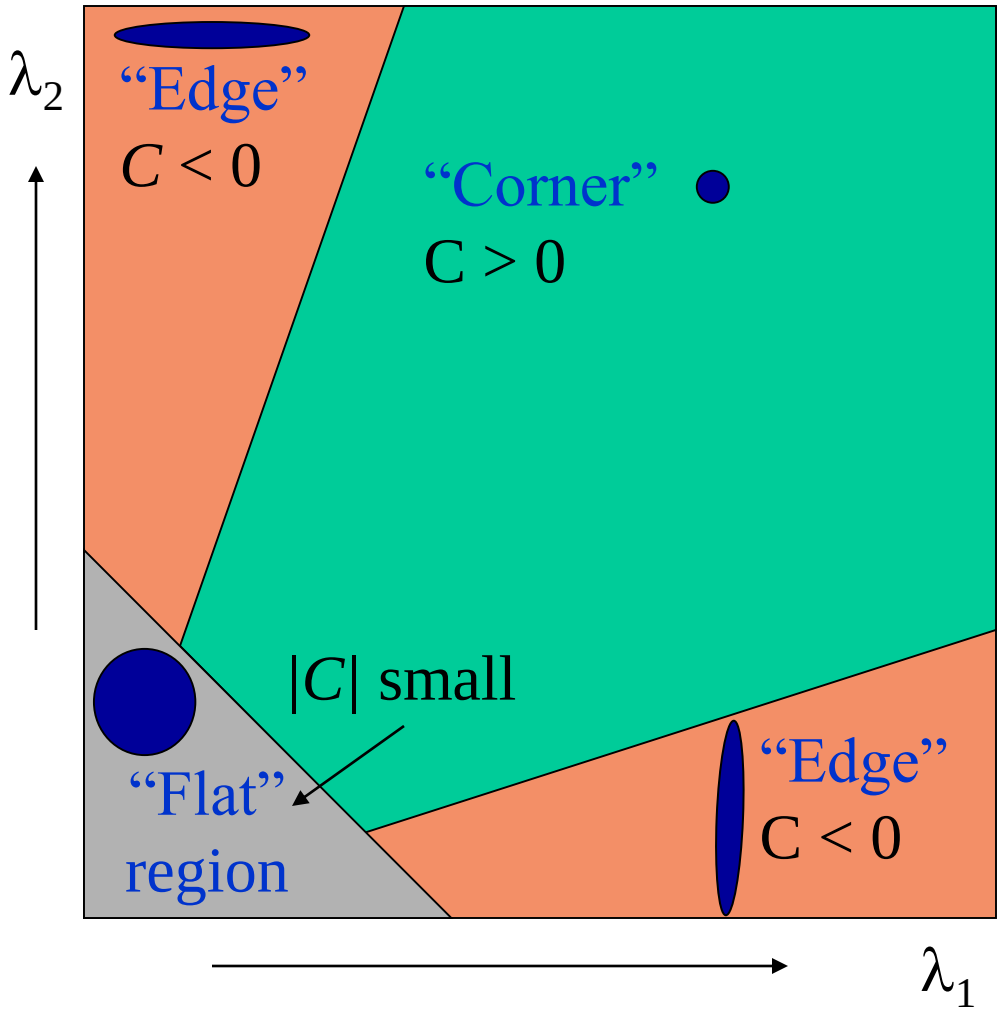


Classification of image points using eigenvalues of *M*

Cornerness score:

$$C = \lambda_1\lambda_2 - \alpha(\lambda_1 + \lambda_2)^2$$

α: some constant (~0.04 to 0.06)



Linear algebra review

Determinant of a diagonal matrix is the *product* of all eigenvalues. A is diagonal.

Trace of a square matrix is the *sum* of its diagonal entries; and is the sum of its eigenvalues.

Classification of image points using eigenvalues of *M*

Cornerness score:

$$C = \lambda_1\lambda_2 - \alpha(\lambda_1 + \lambda_2)^2$$

α: some constant (~0.04 to 0.06)

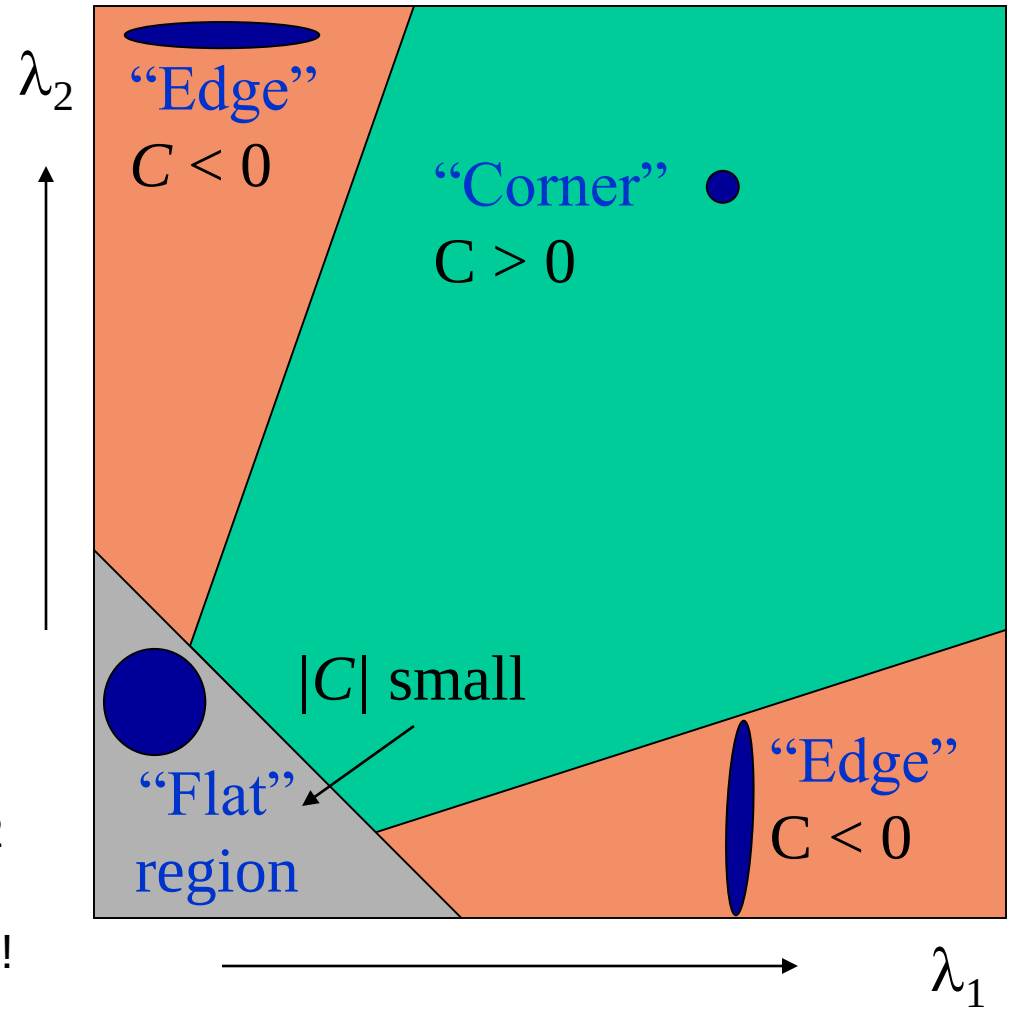
Remember your linear algebra:

Determinant: $\det(A) = \prod_{i=1}^{n} \lambda_i = \lambda_1\lambda_2 \cdots \lambda_n$.
(diagonal matrices)

Trace: $\mathrm{tr}(A) = \sum_i \lambda_i$.

$$C = \det(M) - \alpha\,\mathrm{trace}(M)^2$$

Avoids explicit eigenvalue computation!



This is the Harris corner detector!

1) Compute *M* matrix for each window to recover a *cornerness* score C.

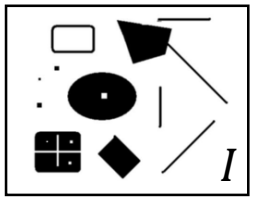
   Note: We can find *M* purely from the per-pixel image derivatives!

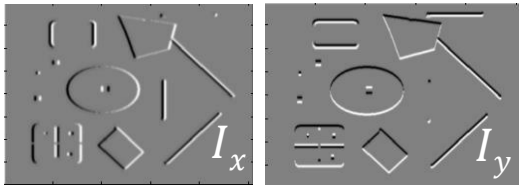
2) Threshold to find pixels which give large corner response (C > threshold).

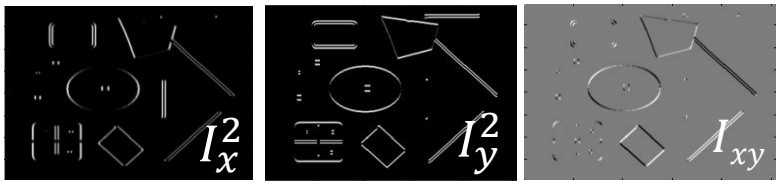
3) Find the local maxima pixels,
   i.e., non-maximal suppression.

C.Harris and M.Stephens. "A Combined Corner and Edge Detector."
Proceedings of the 4th Alvey Vision Conference: pages 147—151, 1988.

Harris Corner Detector [Harris88]







0. Input image
   We want to compute M at each pixel.

1. Compute image derivatives (optionally, blur first).

2. Compute M components as squares of derivatives.

3. Gaussian filter *g()* with width σ

$$= g(I_x^2),\ g\left(I_y^2\right),\ g\left(I_x \circ I_y\right)$$

Reminder: $a \circ b$ is Hadamard product (element-wise multiplication)

4. Compute cornerness

$$C = \det(M) - \alpha\ \mathrm{trace}(M)^2$$
$$= g(I_x^2) \circ g\left(I_y^2\right) - g\left(I_x \circ I_y\right)^2 - \alpha\left[g(I_x^2) + g\left(I_y^2\right)\right]^2$$

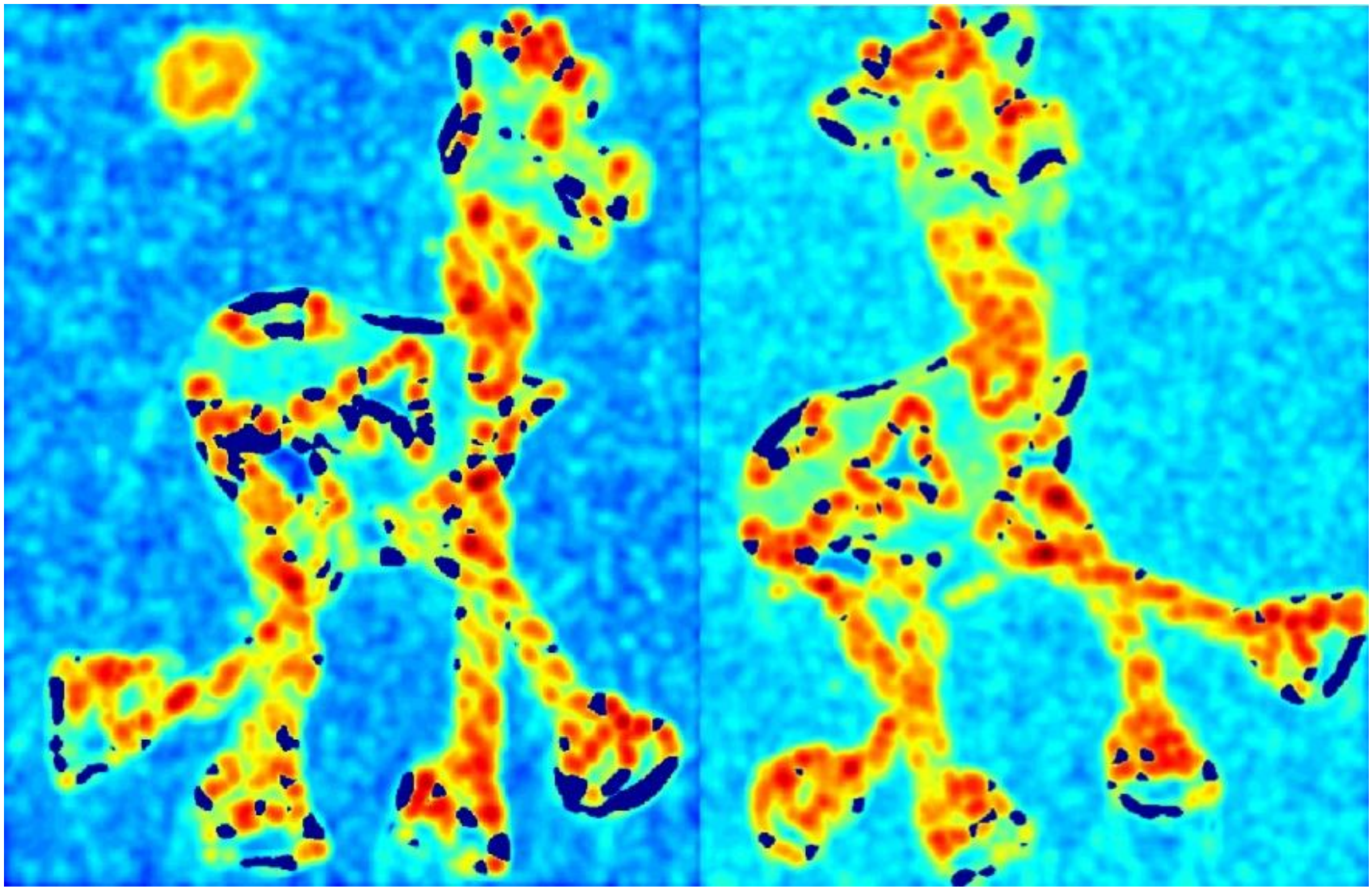
5. Threshold on C to pick high cornerness

6. Non-maximal suppression to pick peaks.

Harris Detector: Steps

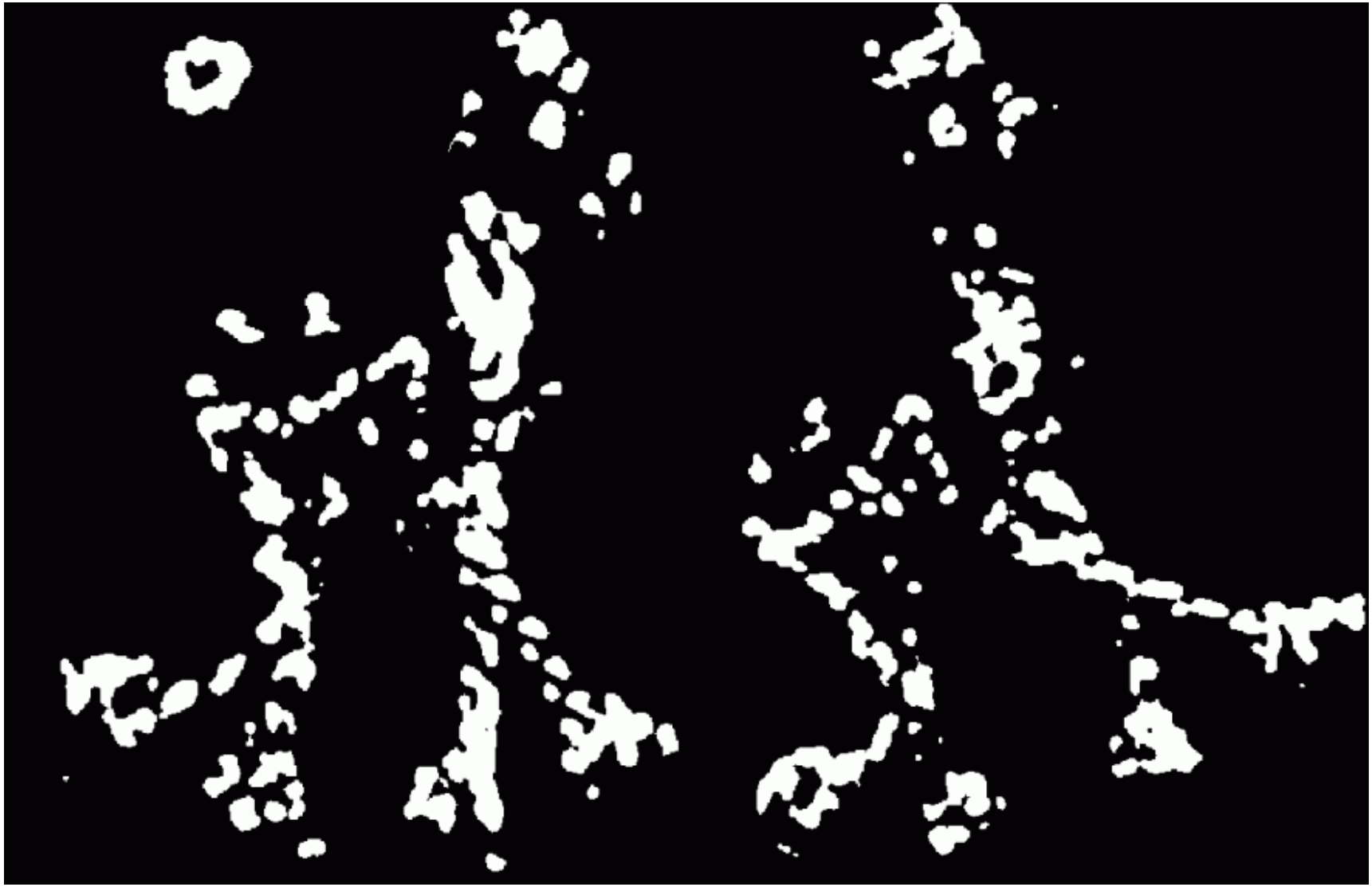
Harris Detector: Steps

Compute corner response C



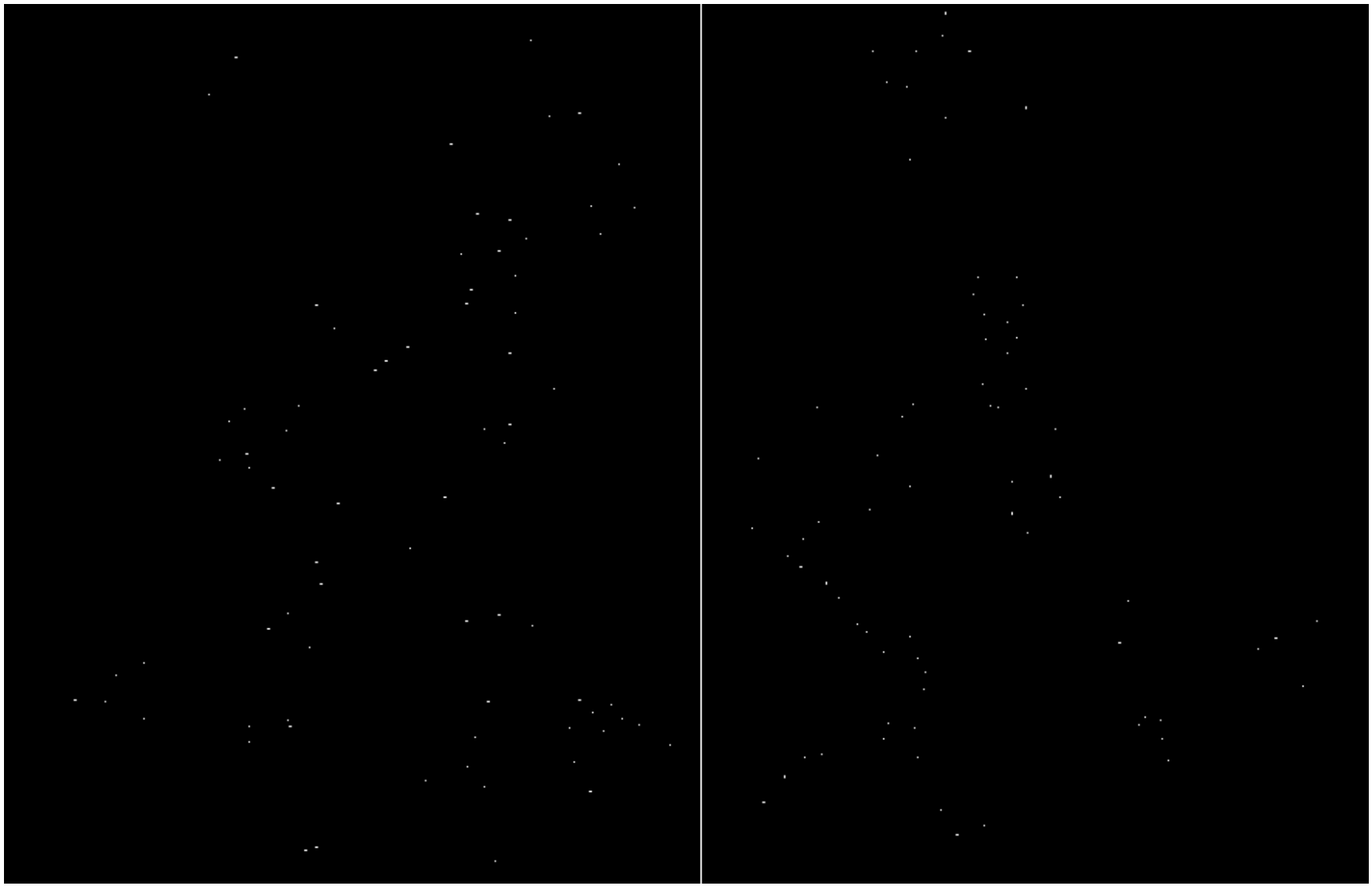
Harris Detector: Steps

Find points with large corner response: C > threshold



Harris Detector: Steps

Take only the points of local maxima of C



Harris Detector: Steps

Live Harris Demo