# Image processing

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

December 2025

Many of the slides are adapted from slides by Kris Kitani, Shmuel Peleg

# Last week

**RANSAC** - A powerful, iterative algorithm used to estimate parameters for a mathematical model from noisy data containing outliers.

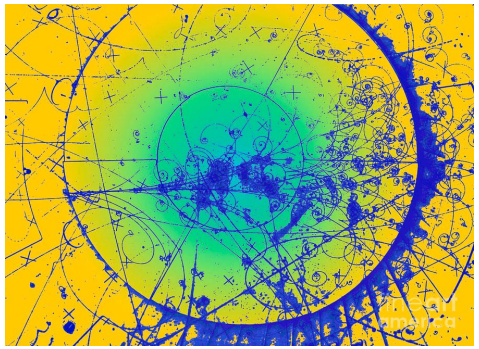
- **Sample** random minimal data subsets repeatedly
- **Fit** the mathematical model
- **Count** how many other points support that model (inliers)
- **Select** the most-supported model

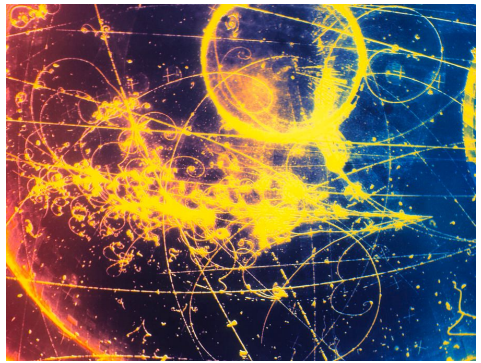
# Last week

- RANSAC was an example of a voting-based fitting algorithm.
- Each hypothesis gets “voted” by each data point, the most scored hypothesis wins.
- There are more types of voting-based algorithm..

# Hough Transform

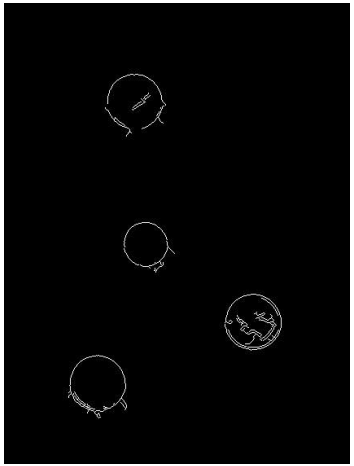
Originally suggested by Hough for bubble chamber photographs analysis in 1959.

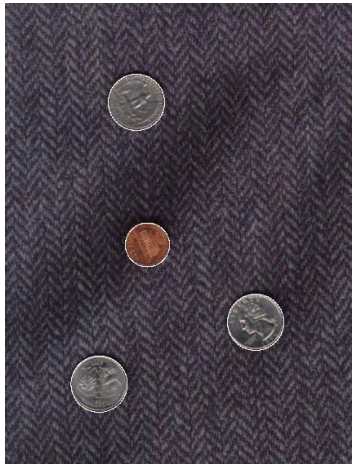




# Hough Transform

- Originally – used for detecting lines.
  Could also be noisy, incomplete, occluded lines.
- Later – generalized for detecting arbitrary shapes: lines, circles, etc.





# Hough Transform

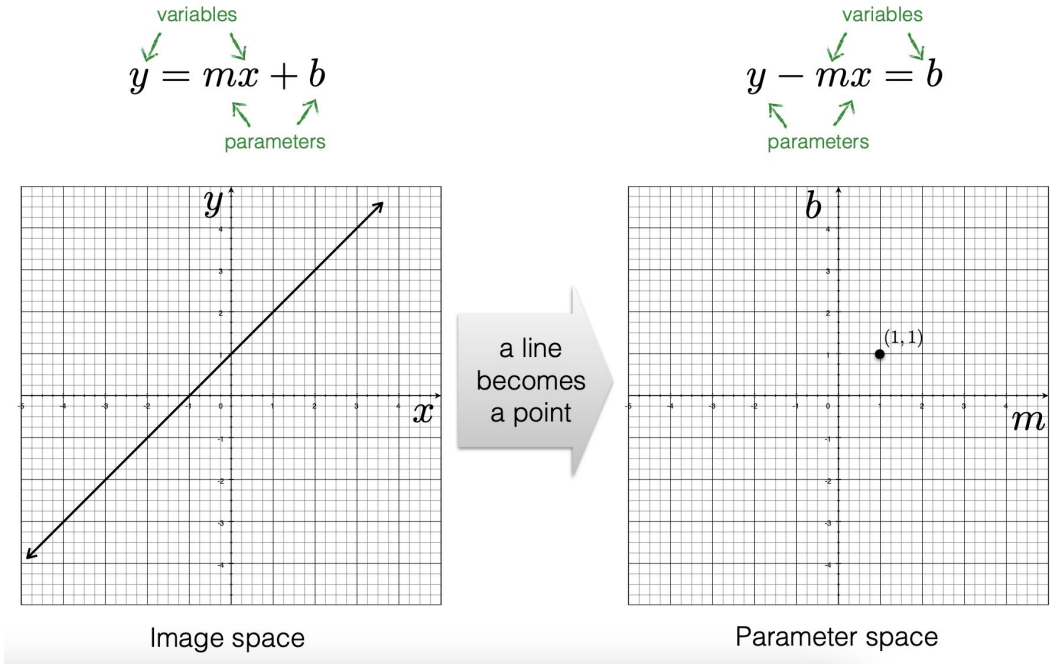
**Idea:**

- Map points from the image space into a parameter space where lines are represented as points (peaks), and vice versa.

- Edges vote for possible lines, and eventually the most voted line is taken.

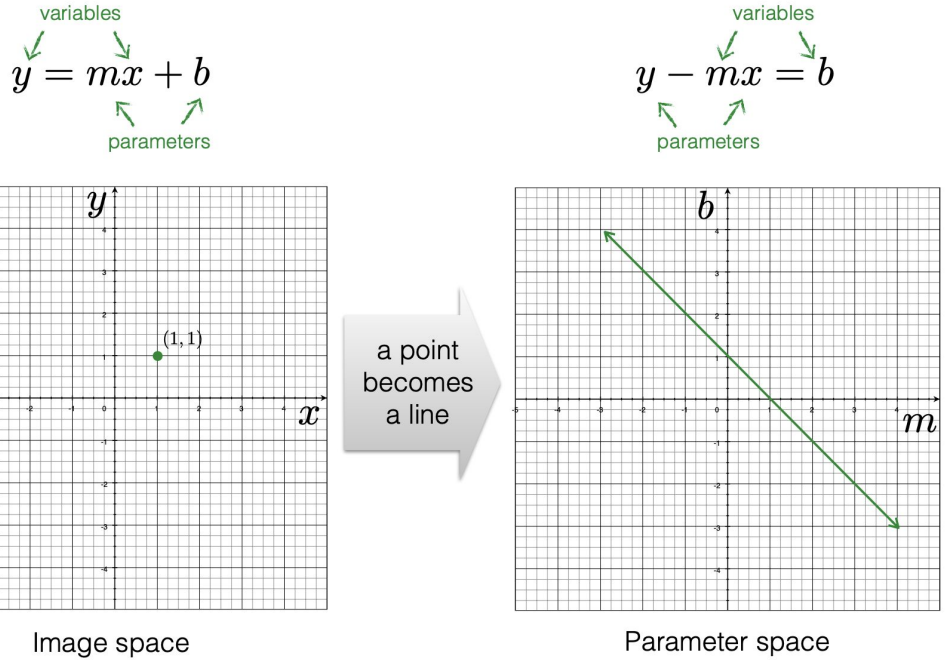
# Representation shift

- A line in Cartesian Representation: y = mx + b.
- Each image point (x,y) could be mapped to the (slope, intercept) parameter space.



# Representation shift

- A line in Cartesian Representation: y = mx + b.
- Similarly, each point becomes a line.



# Representation shift

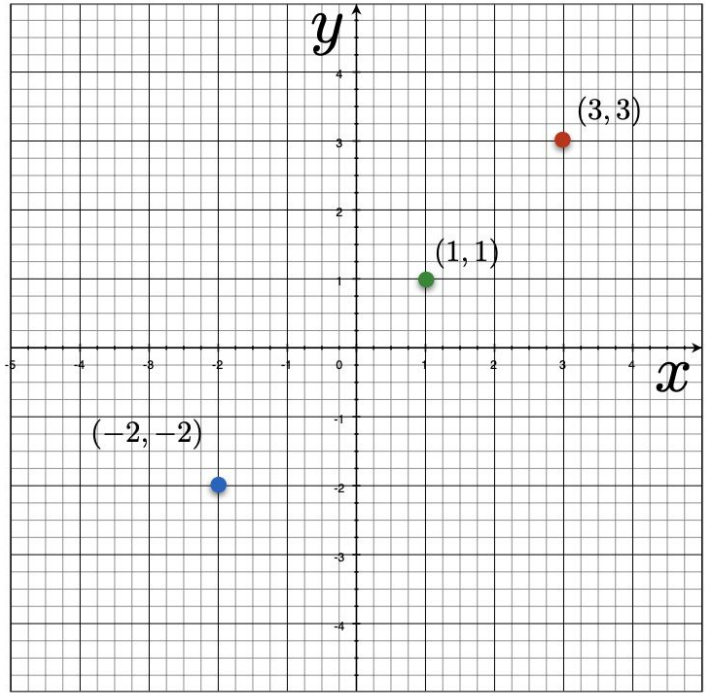
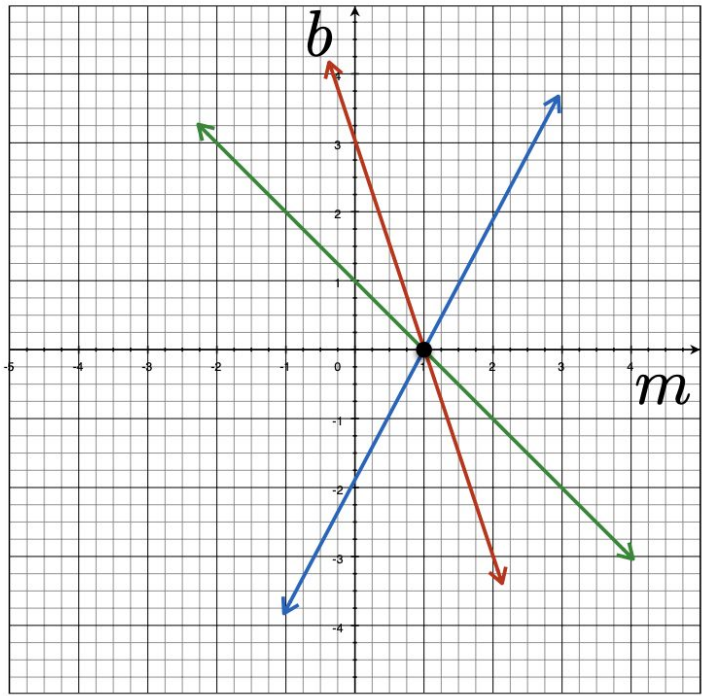



Image space

three points become ?




Parameter space

# Representation shift

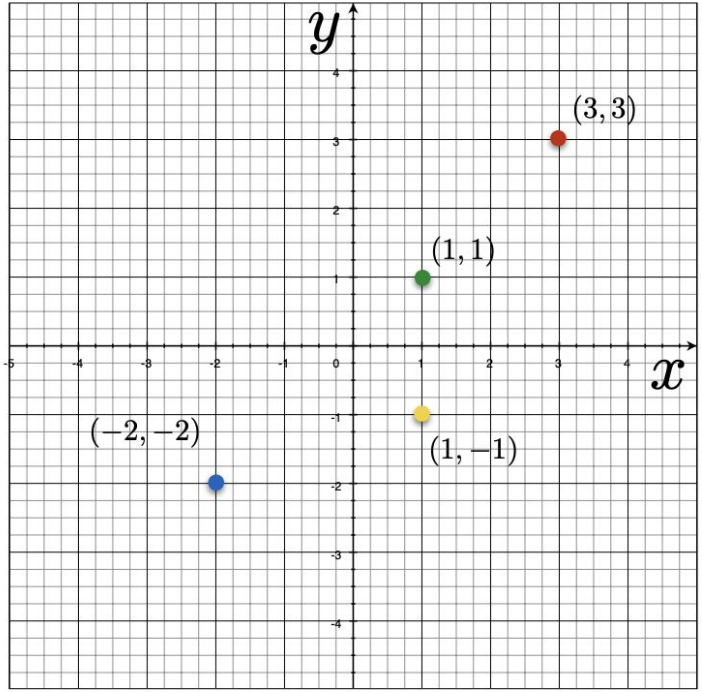
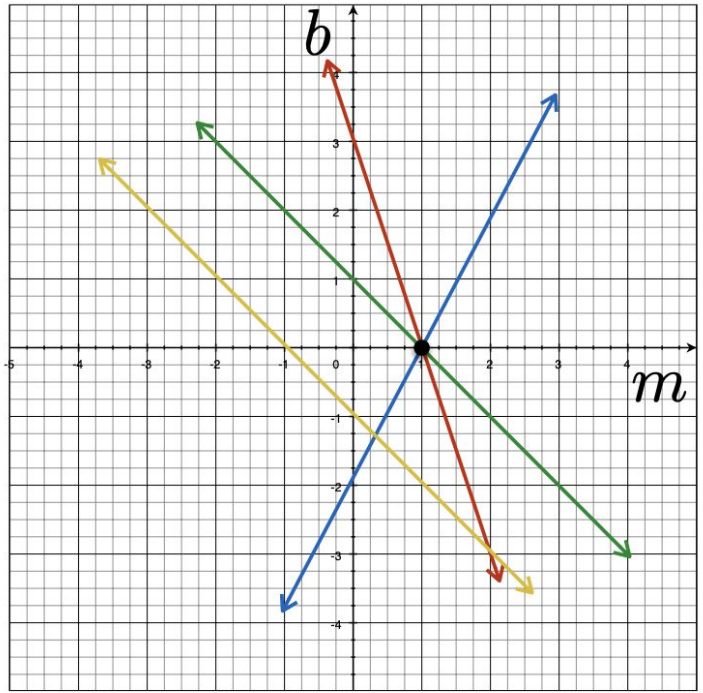



Image space

four points become ?




Parameter space

# Representation shift

If this is our data – how can we fit a line to it? vote!

Choose the (m,b) that is the intersection between the largest number of lines.

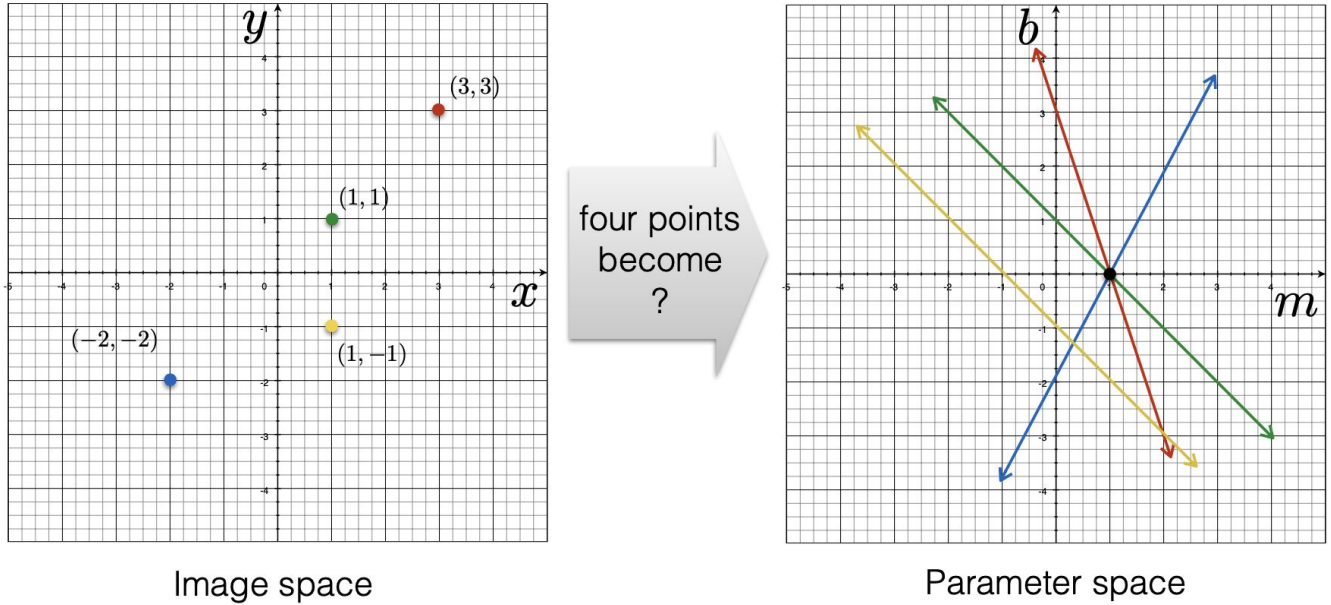



Image space

Parameter space

# Hough Transform

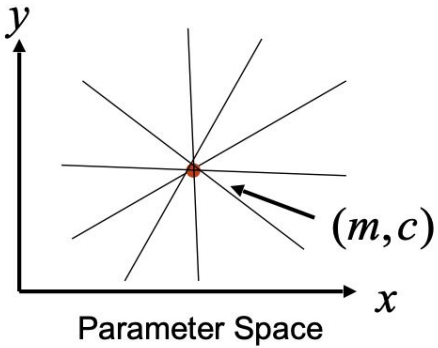
**Algorithm #1:**

1. Create an accumulator grid A for the (m,c) space with zeros everywhere.

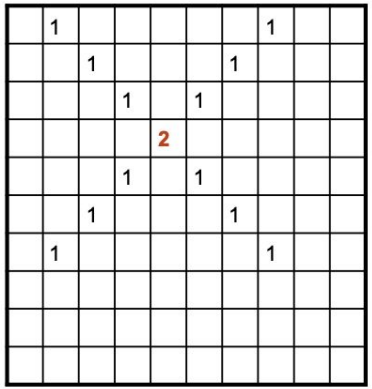
2. Run an edge detector (e.g. canny) on the image.

3. For each pixel of an image edge (x, y):
   - For each entry in A: if (m,c) lies on the line c = -xm+y => A[m,c] += 1

4. Find local maxima in A.




$A(m,c)$



# Hough Transform

**Problems?**

What's the size of A?

There are too many (infinite) m, c possibilities!

# A better representation – Polar coordinates

- Use normal form: $\rho = x\cos(\theta) + y\sin(\theta)$
- ρ = perpendicular distance from origin to line
- θ = angle of the line's normal vector
- Finite array size:

  $0 <= \theta < \pi$

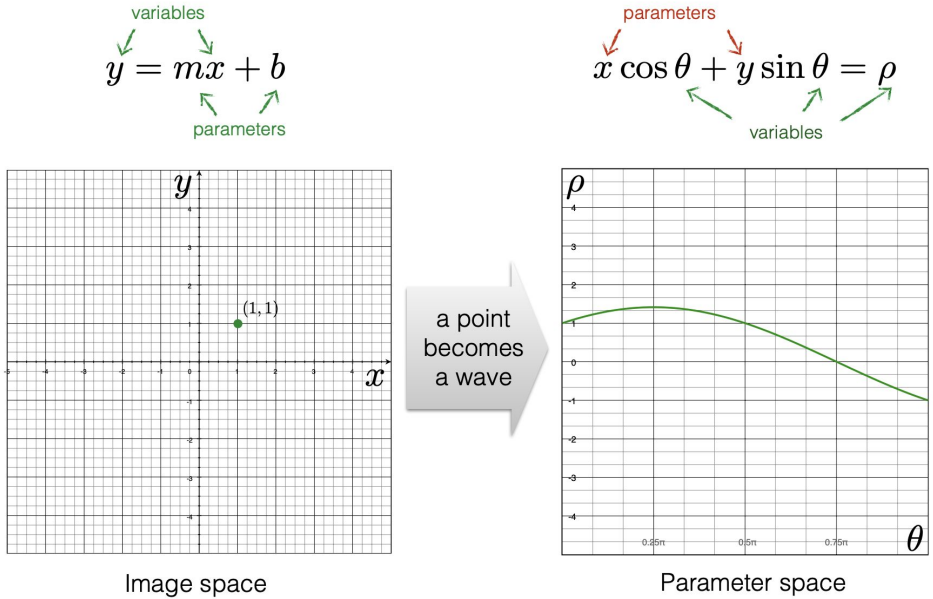
  $0 <= \rho <= \rho_{max}$

  ($\rho$ is limited by image diagonal)

- Up to $\pi$ because:

  $\sin(\theta) = -\sin(\theta+\pi)$

  $\cos(\theta) = -\cos(\theta+\pi)$



# A better representation

- Use normal form: $\rho = x\cos(\theta) + y\sin(\theta)$
- ρ = perpendicular distance from origin to line
- θ = angle of the line's normal vector

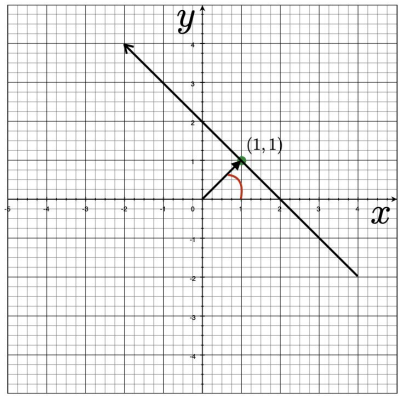
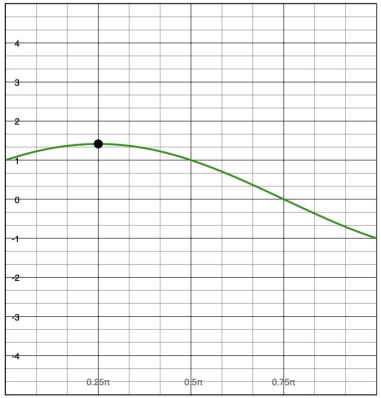



Image space

a line becomes a point



Parameter space

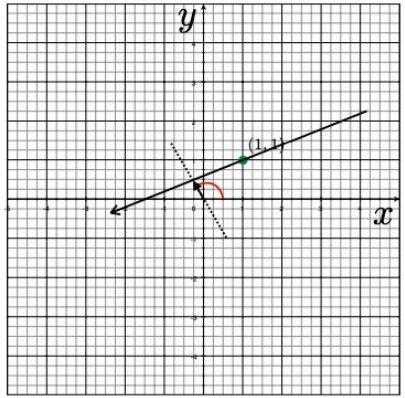
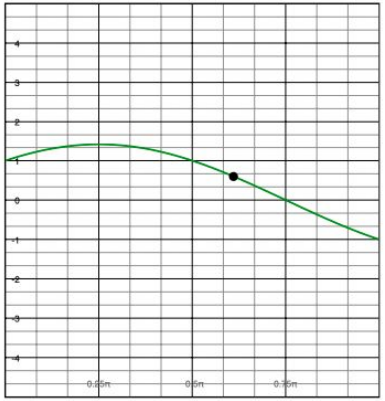



Image space

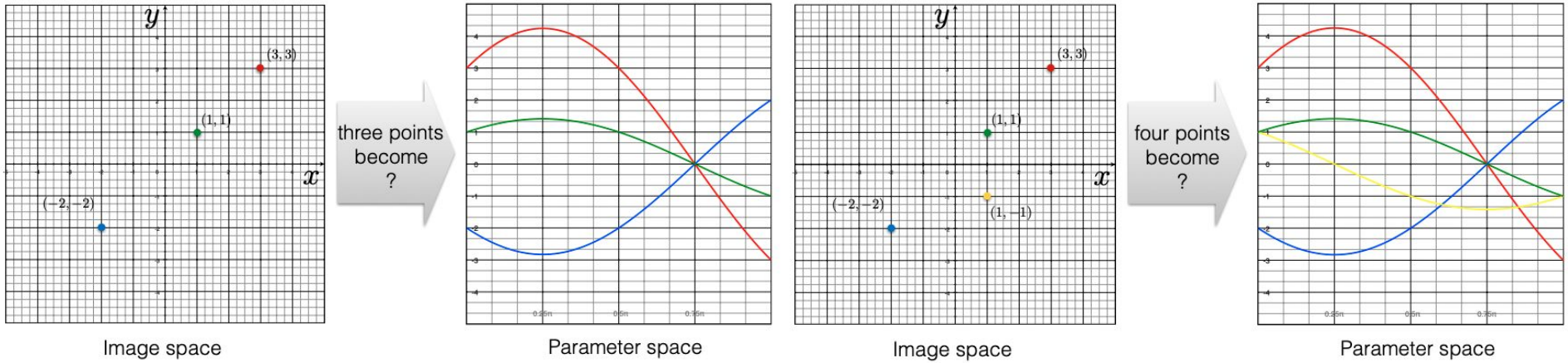
a line becomes a point



Parameter space

# A better representation

- Use normal form: $\rho = x\cos(\theta) + y\sin(\theta)$
- ρ = perpendicular distance from origin to line
- θ = angle of the line's normal vector



# Hough Transform

**Algorithm:**

1. Create an accumulator grid A for the (ρ,θ) space with zeros everywhere (need to perform quantization – define angle bins).

2. Run an edge detector (e.g. canny) on the image.

3. For each pixel of an image edge (x, y):
   - For θ in [0,180): $\rho = x \cos\theta + y \sin\theta \Rightarrow A[\rho,\theta] \mathrel{+}= 1$

4. Find local maxima in A.

5. The detected line in the image is given by: $\rho = x \cos\theta + y \sin\theta$

# Hough Transform

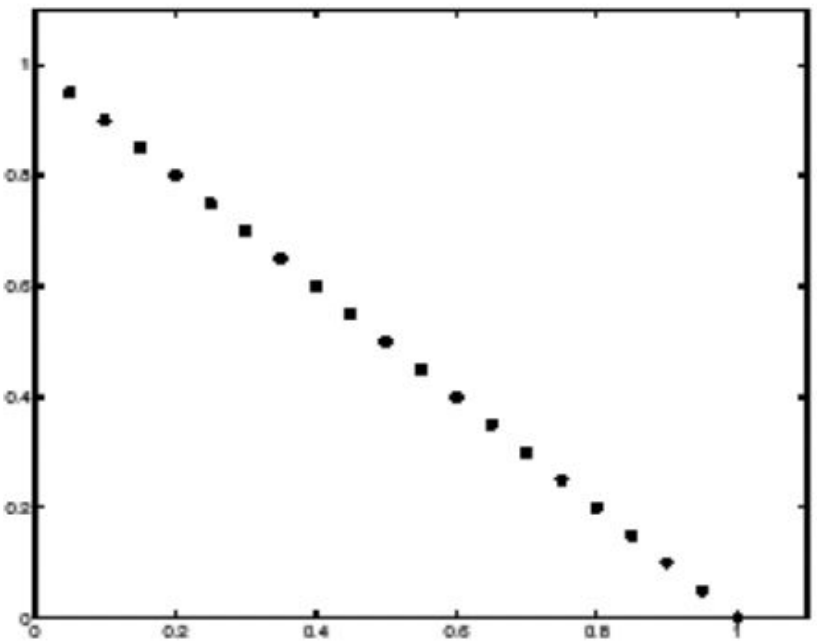
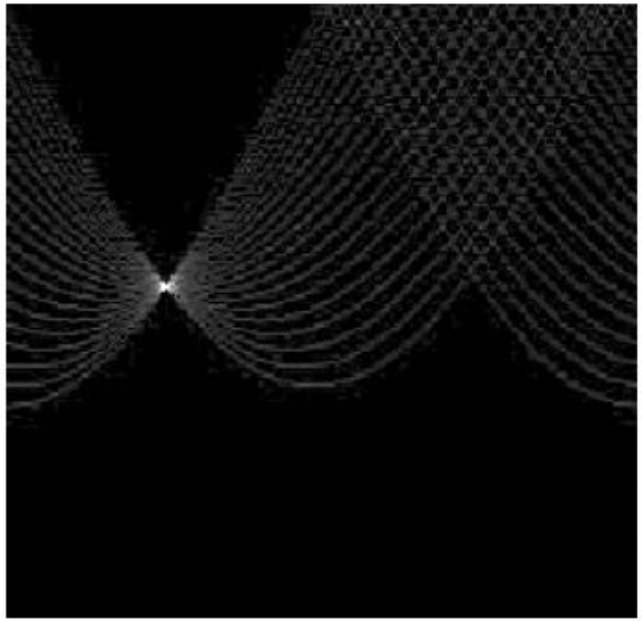



Image space



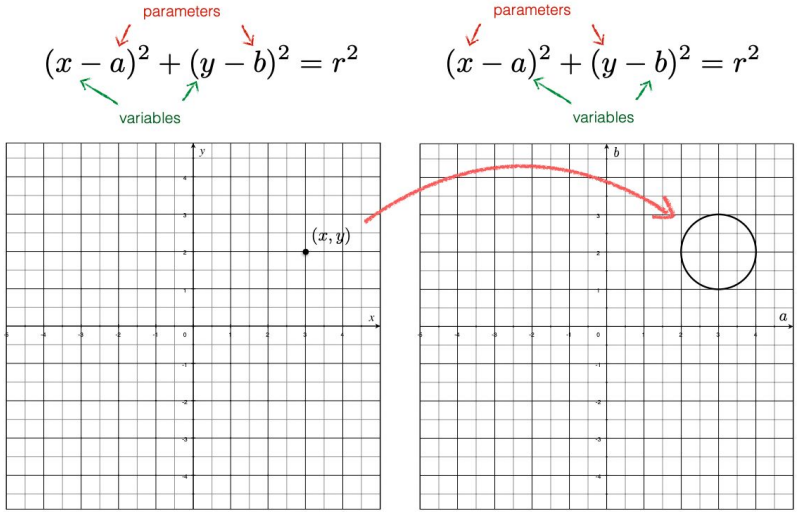
Votes

# Hough Transform

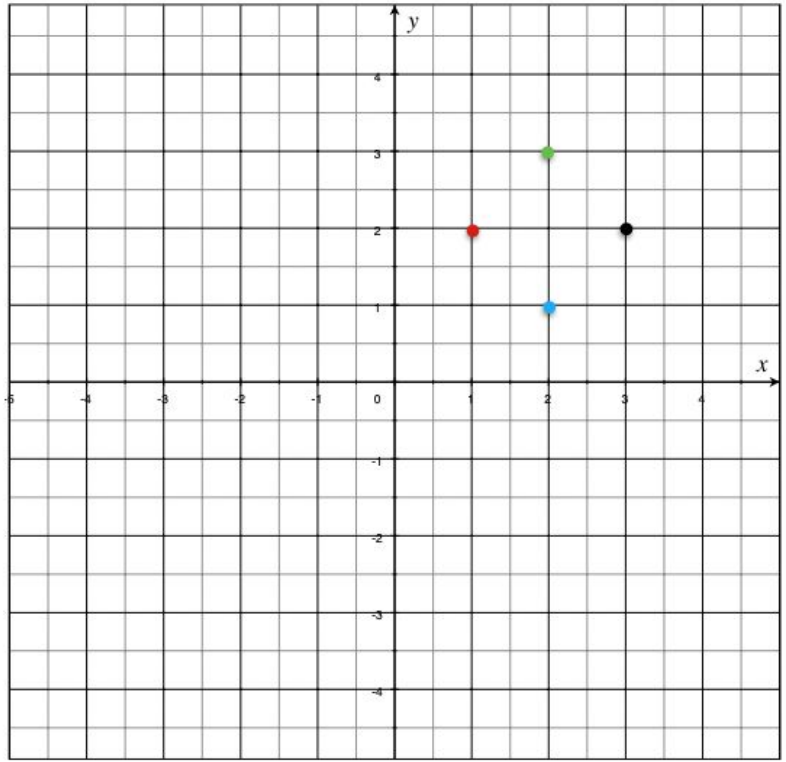
How to generalize to circles?

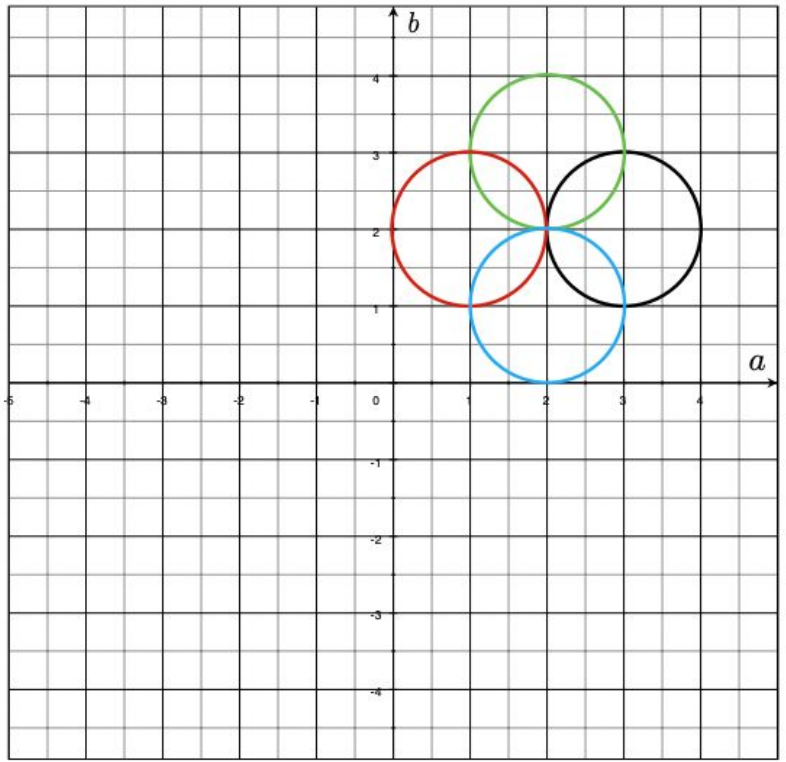
Use circle equation: $(x-a)^2 + (y-b)^2 = r$

Difference from lines – 3 parameters. If radius is known – same as lines.



# Hough Transform

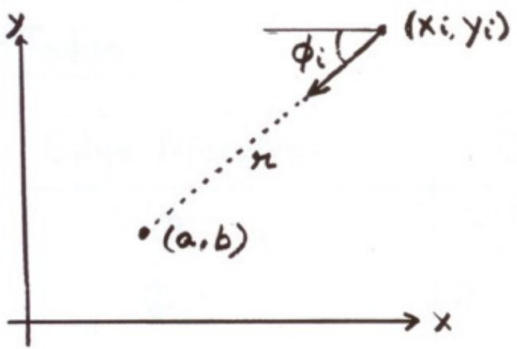


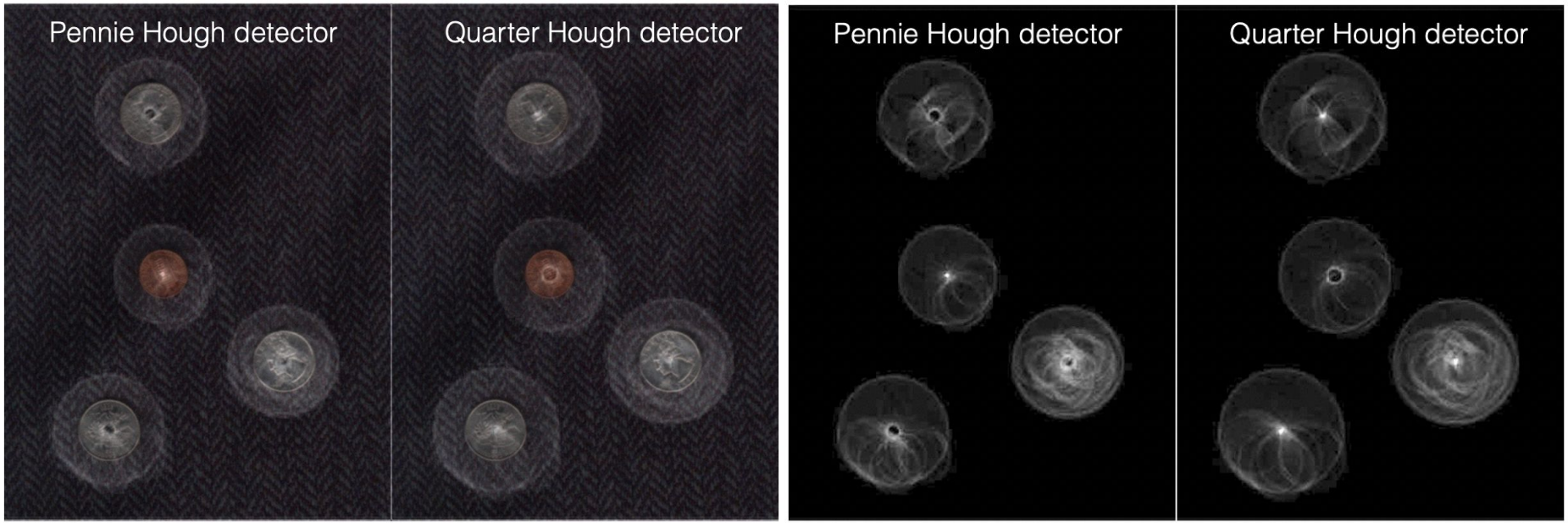
# Hough Transform

If radius is not known - 3D hough transform, use an accumulator array with (a,b,r).

- Can use gradient information to save computation:
- For edge location (x,y) and edge direction θ, circle center (a,b): $a = x - r\cos(\theta)$, $b = y - r\cos(\theta)$
- For each radius, need to increment only one point in accumulator – the center in the gradient direction.



# Hough Transform



# Hough Transform

Properties:

- Robust to noise, occlusions
- Can detect multiple instances
- Slow – Bad computational complexity

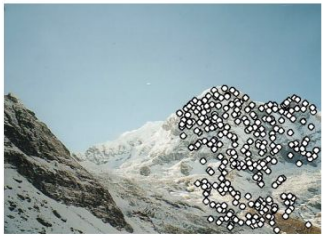
# Panorama exercise

(a) Image 1

(b) Image 2

(c) SIFT matches 1

(d) SIFT matches 2



(e) RANSAC inliers 1

(f) RANSAC inliers 2

(g) Images aligned according to a homography

Credit: Brown and Lowe, IJCV 2007

# 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

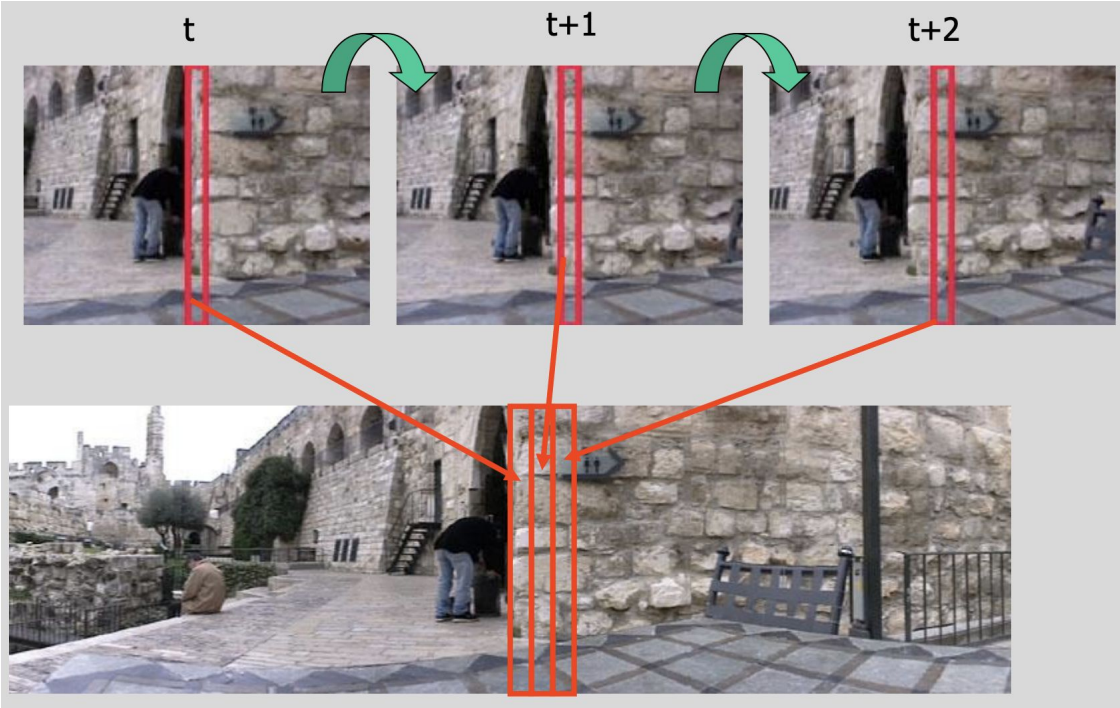
# Panorama Stitching

**Problems:**

- Illumination changes between images
- Ghosting (due to moving objects)
- Blurriness (due to mis-registration)

# Panorama Stitching

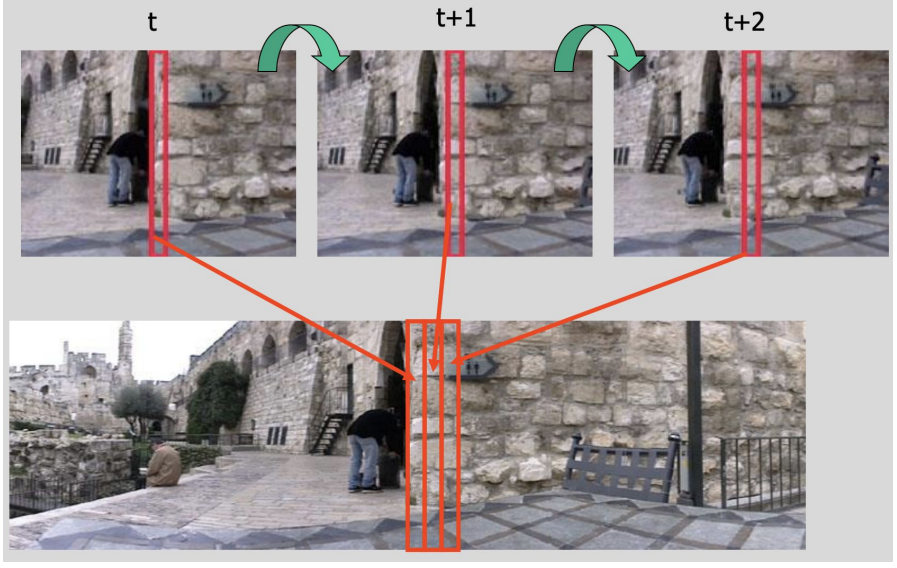
Use Center Strips from each warped frame, stitch together.



# Panorama Stitching

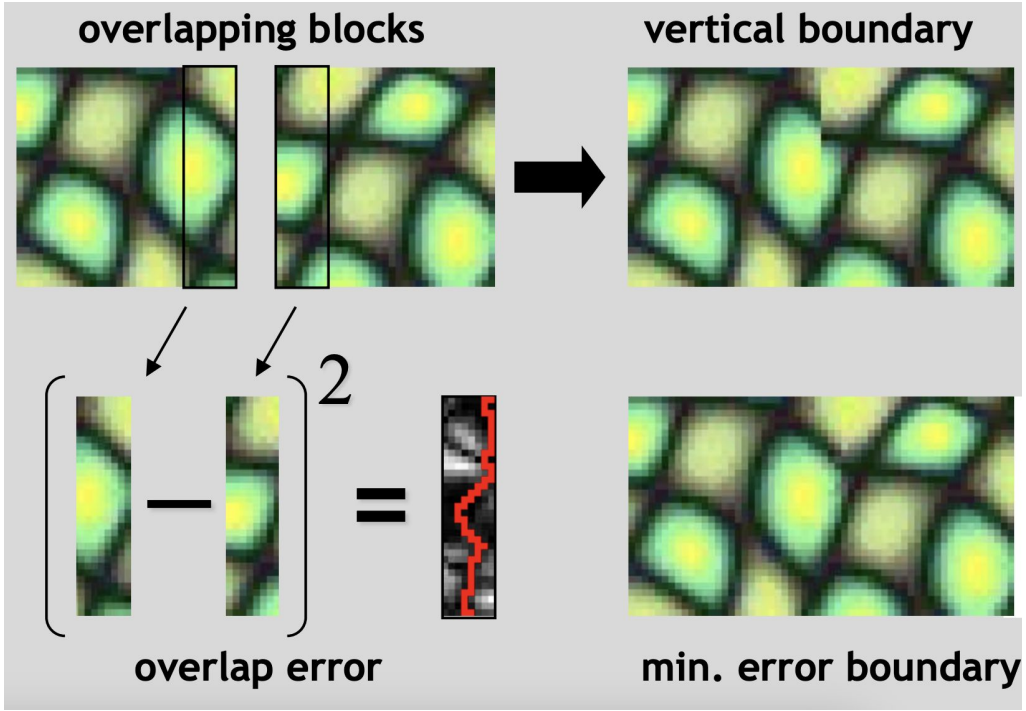
**Algorithm:**

1. Choose a reference image (usually the middle).
2. Find homographies and warp all images toward the reference image
3. Take a center strip for each image
4. Stitch together by blending (e.g. using pyramid blending)



# Panorama Stitching

Better: use graph-cut dynamic programming to find best seam!




Credit: Alexei Efros

# Panorama Stitching

Better: use graph-cut dynamic programming to find best seam!

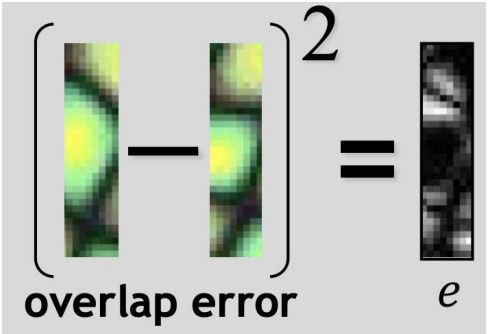
Scan the image row by row and compute a cumulative minimum squared difference E for all paths:

$$E[i,j] = e(i,j) + min(E[i,j-1], E[i-1,j], E[i-1,j-1])$$

$$s.t.\ e = (I_1 - I_2)^2$$

(For overlapping areas between images)

The optimal path can be obtained by tracing back the minimal cost from bottom to top.




Credit: Alexei Efros

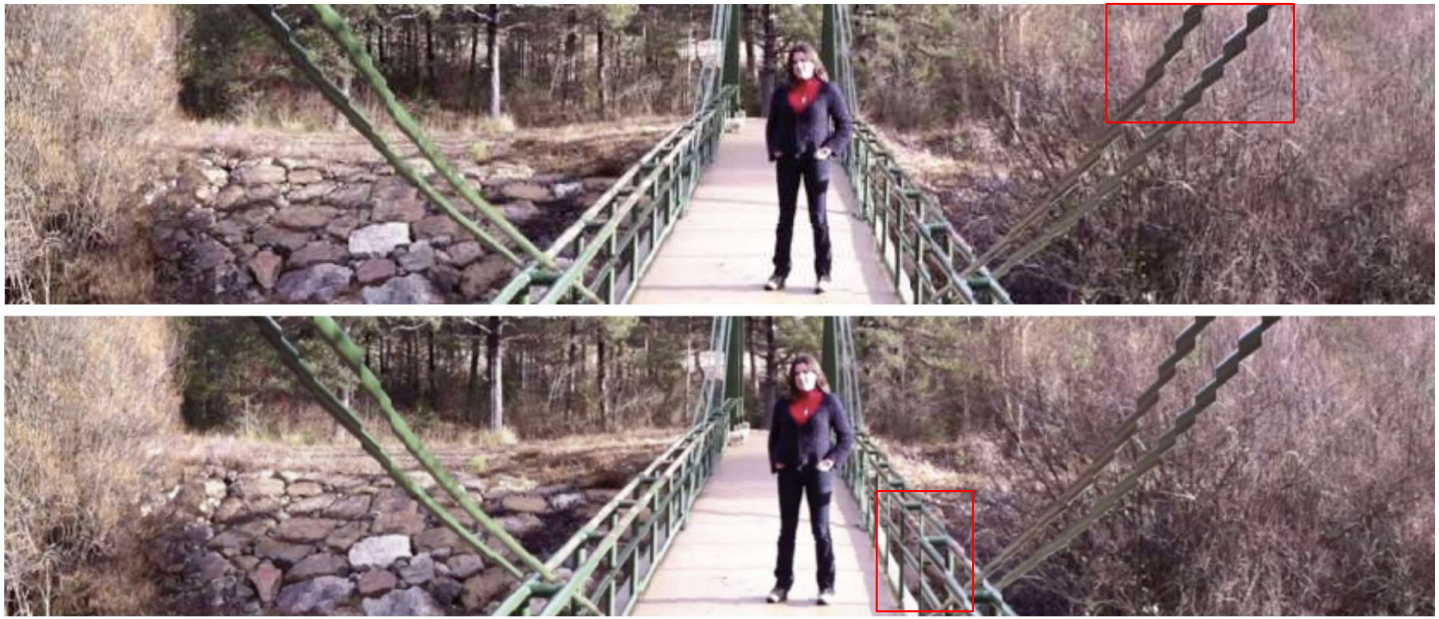
# Panorama Stitching

Simple vs. Min-Cut Stitching

# Barcode Blending

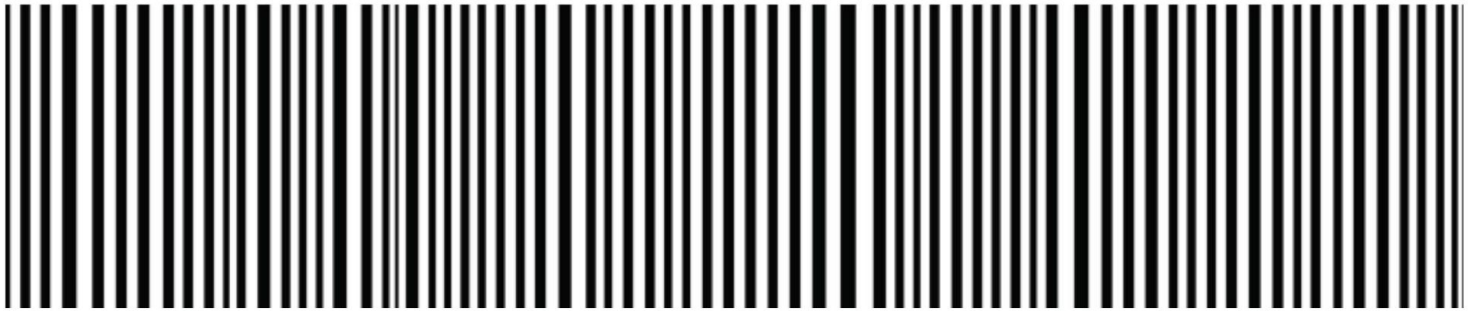
Create 2 intermediate mosaics where strips are with double width:

(i) from Odd frames (ii) from Even frames



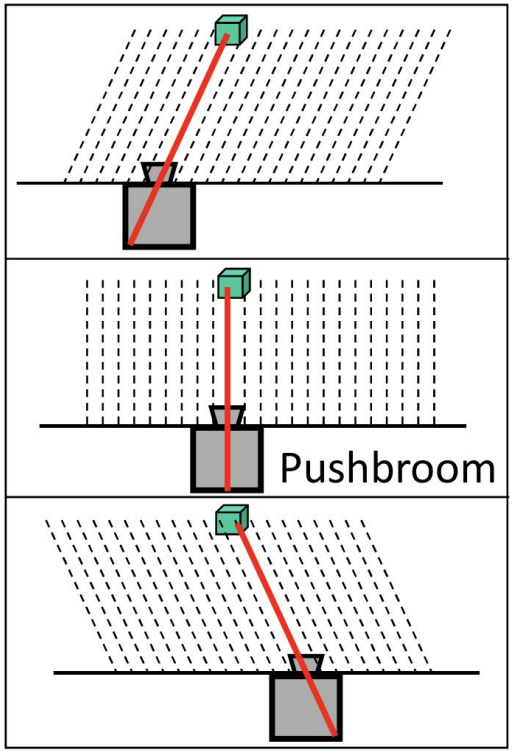
# Barcode Blending

Blend the 2 intermediate mosaics with a barcode mask.



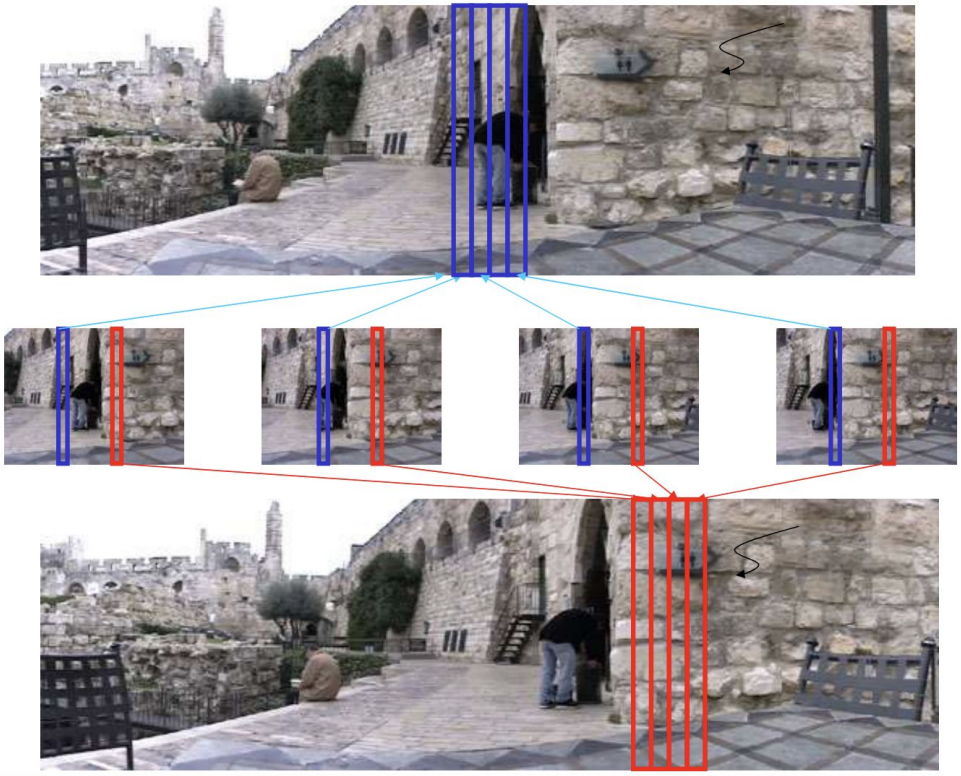
# Multiple Viewing Directions

Different ray angles yield different panorama viewing directions.




# Multiple Viewing Directions

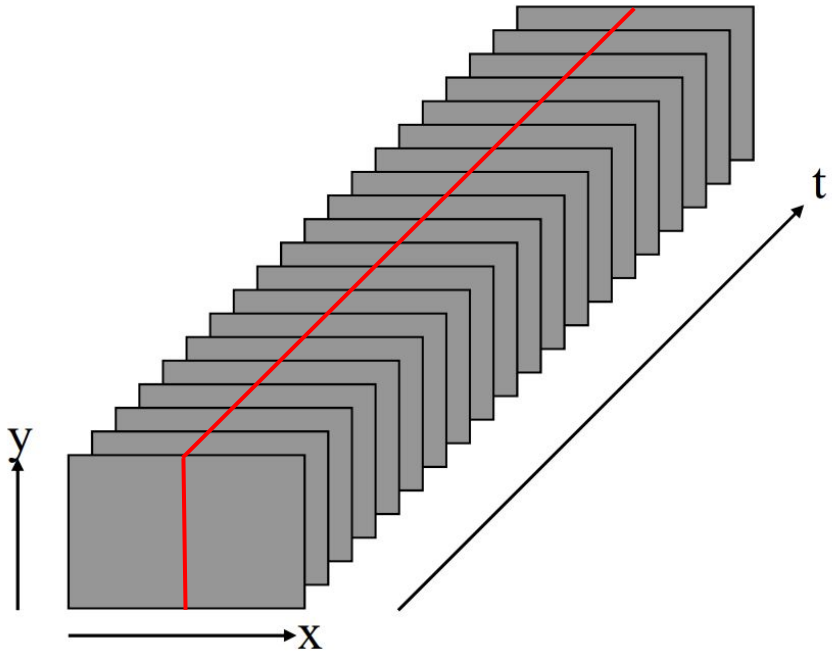
Different ray angles yield different panorama viewing directions.



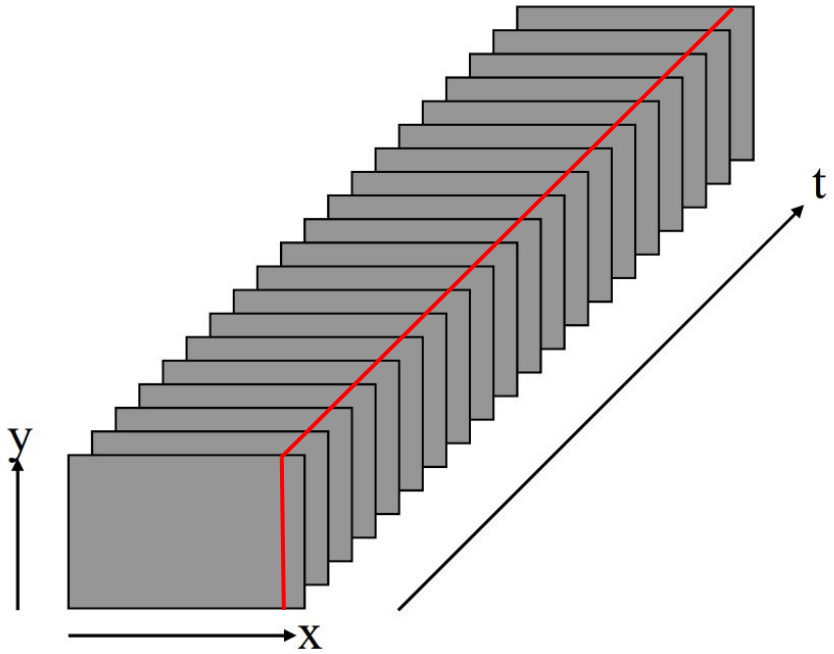
# Multiple Viewing Directions

Different ray angles yield different panorama viewing directions.

Center strips

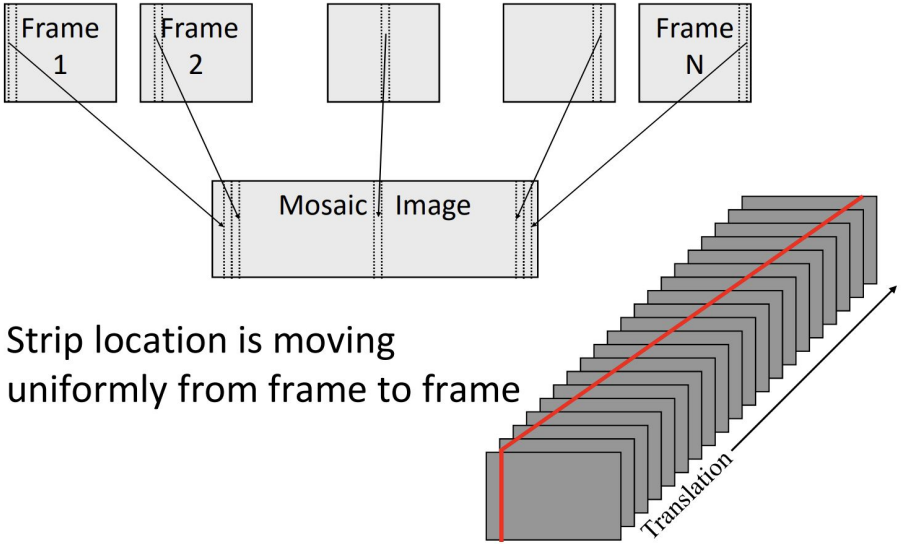



Right strips = left-eye view

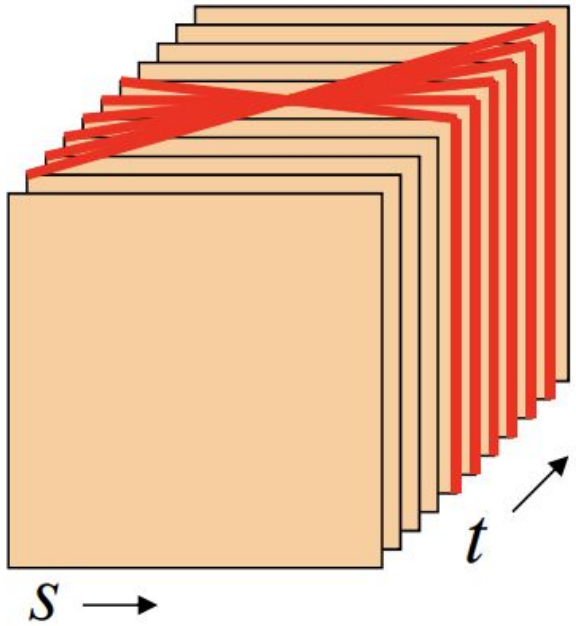


# X-Slits Videos

Taking strips from different locations in different images creates a motion feeling.



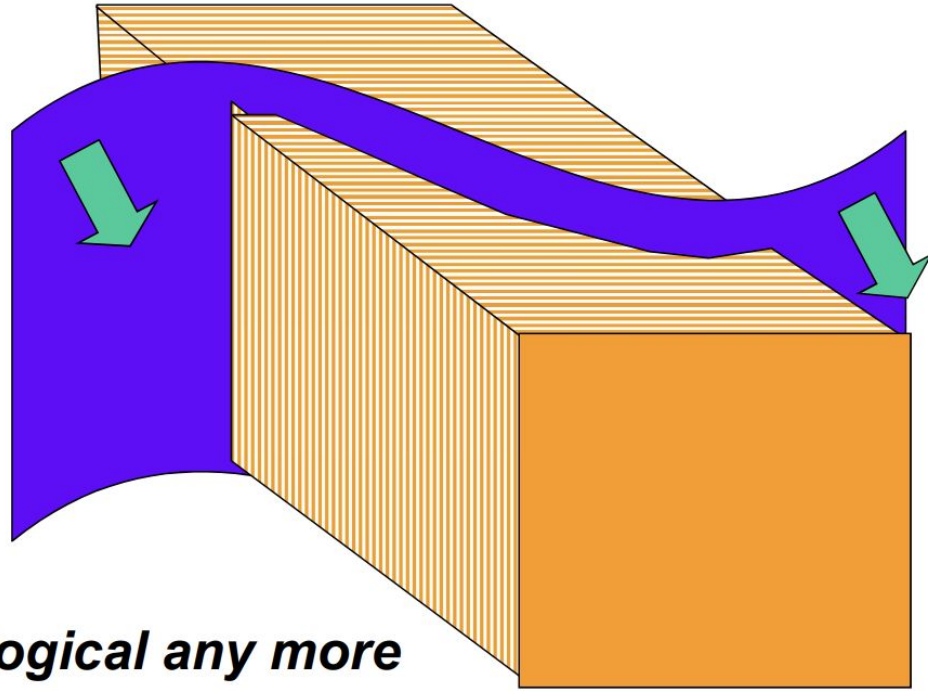

Strip location is moving uniformly from frame to frame



# X-Slits Videos

# Dynamic Mosaics

Evolving Time Fronts (Rav-Acha et al., SIGGRAPH 2005)

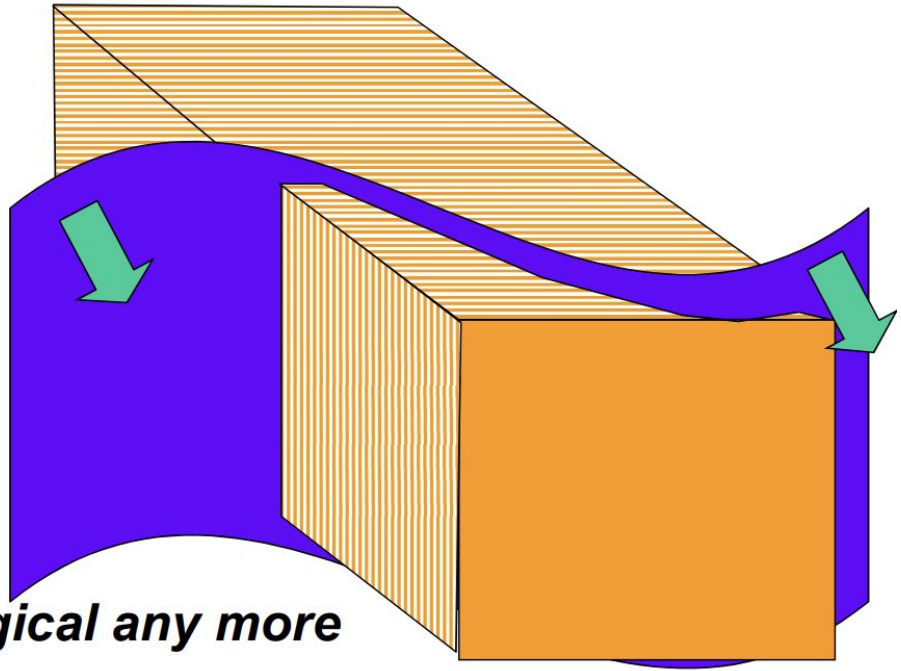


***Interpolation***

***Time is not chronological any more***

# Dynamic Mosaics

Evolving Time Fronts (Rav-Acha et al., SIGGRAPH 2005)

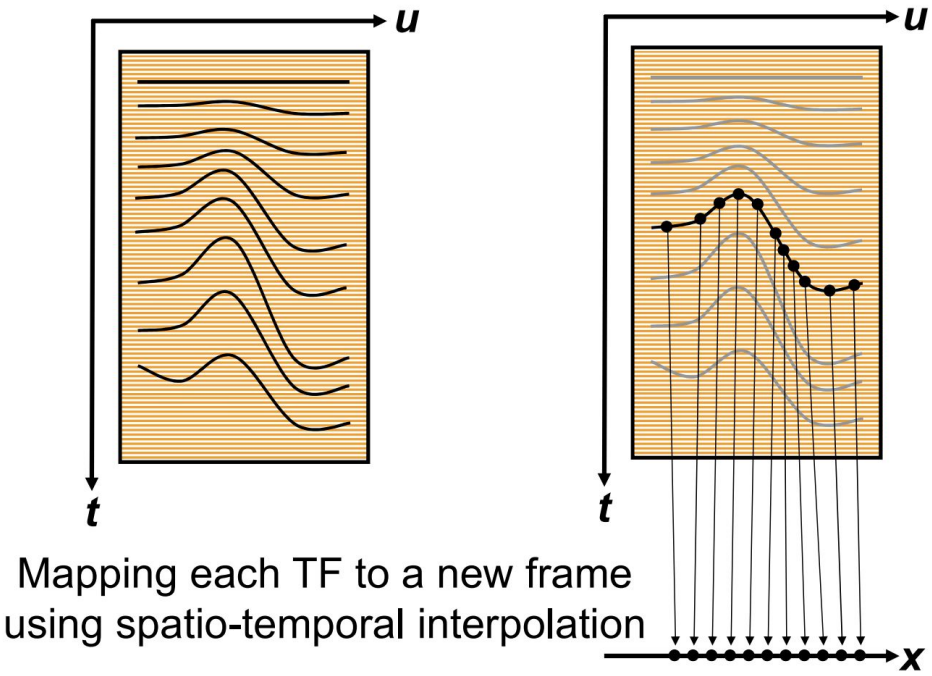


***Interpolation***

***Time is not chronological any more***

# Dynamic Mosaics

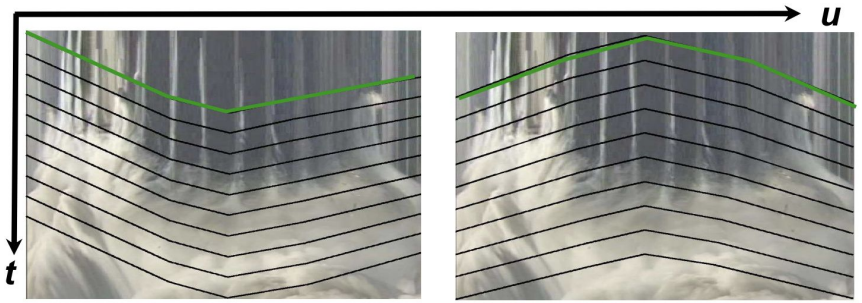
Evolving Time Fronts (Rav-Acha et al., SIGGRAPH 2005)




Mapping each TF to a new frame
using spatio-temporal interpolation

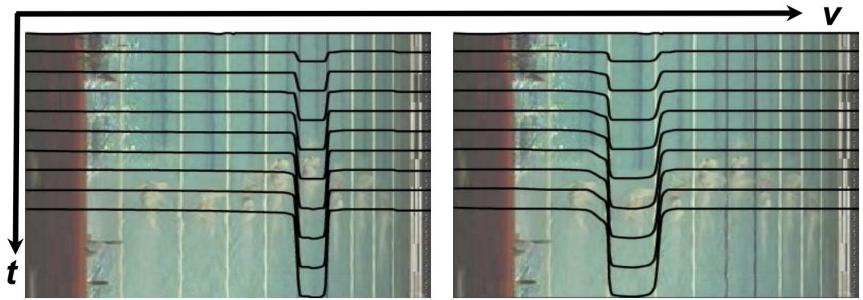
# Dynamic Mosaics

Evolving Time Fronts (Rav-Acha et al., SIGGRAPH 2005)

# Dynamic Mosaics

Evolving Time Fronts (Rav-Acha et al., SIGGRAPH 2005)

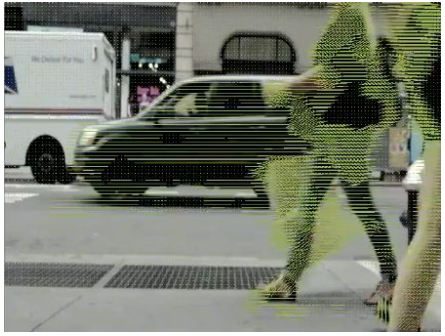



# Dynamic Mosaics

Evolving Time Fronts (Rav-Acha et al., SIGGRAPH 2005)

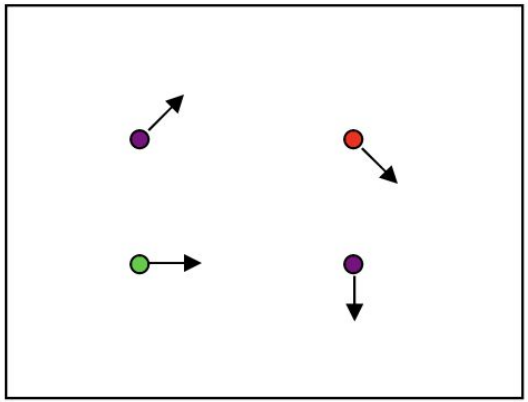
# Optical Flow

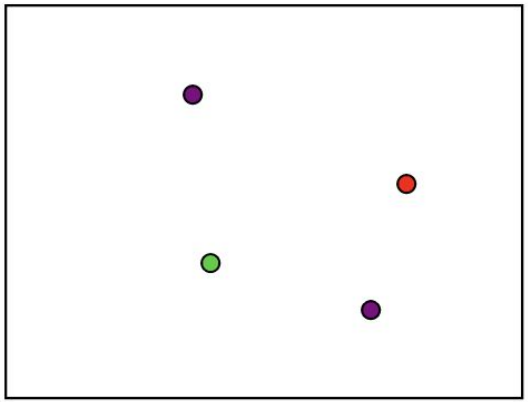
- The apparent 2D motion field of image points between consecutive frames, caused by object motion, camera motion, or both.
- Applications:
  - Motion detection and estimation
  - Object tracking
  - Action recognition
  - Robotics and navigation
  - Video Stabilization
  - Video generation motion smoothness evaluation and supervision

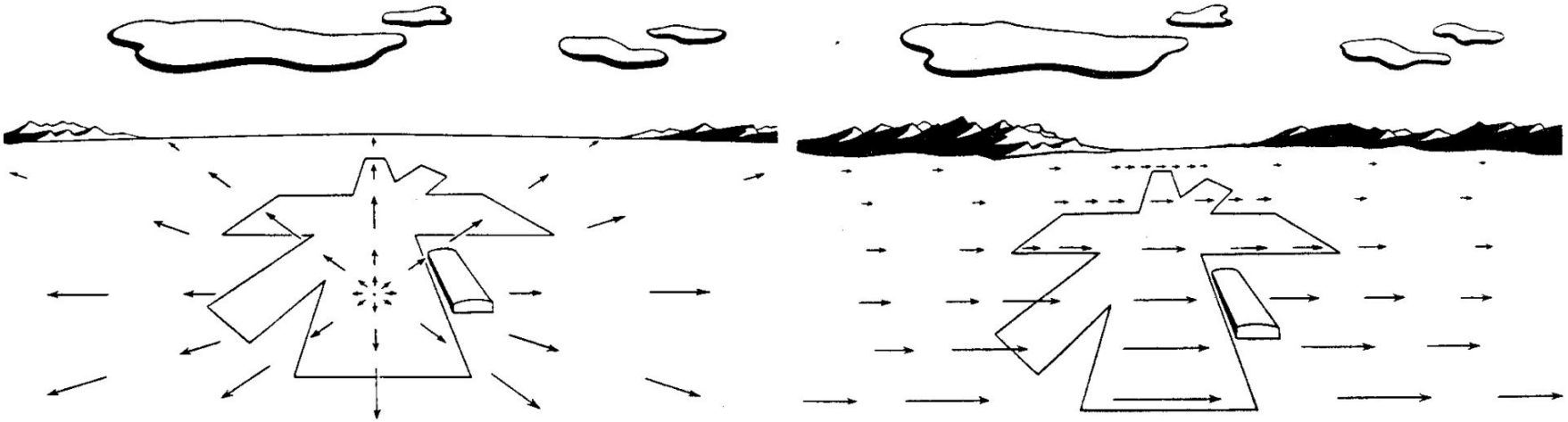
# Optical Flow

Estimate motion of objects from frame1 to frame2.





# Optical Flow

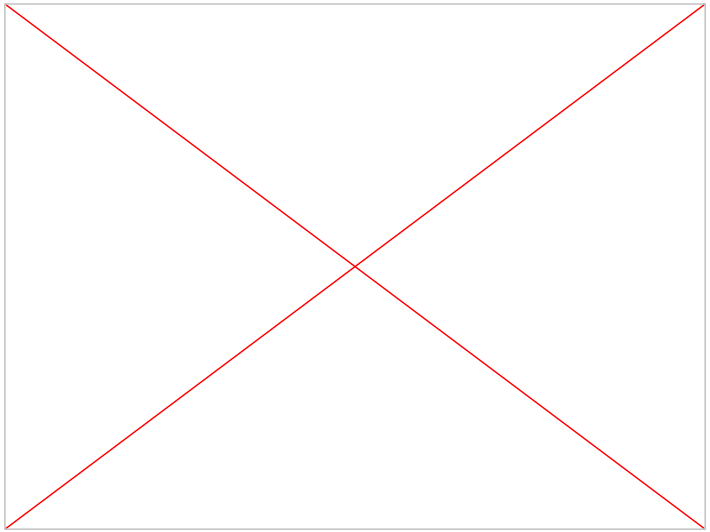


What motion generated this optical flow?

# Optical Flow

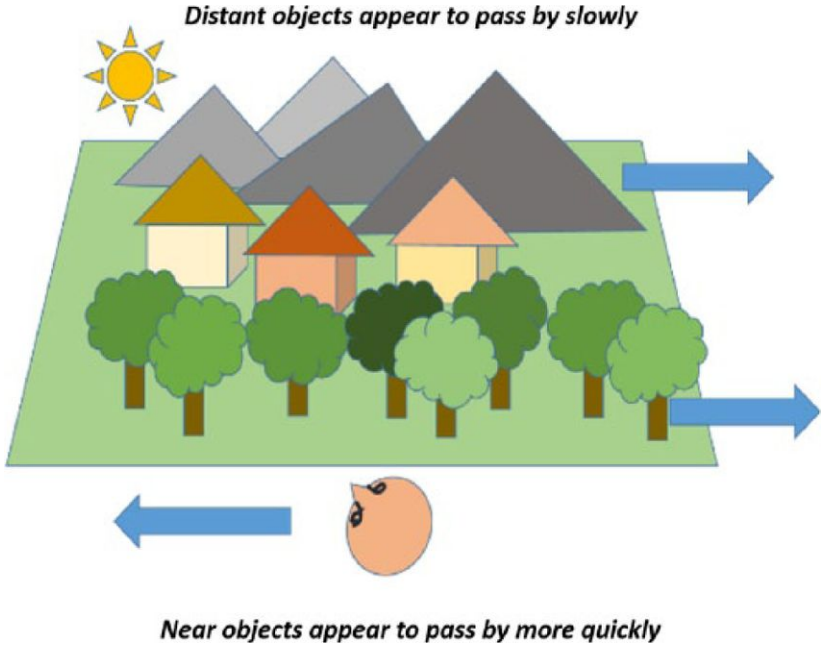
Why not just use matches and estimate homography between each pair of consecutive frames?

- Independent object movements, e.g. cars, people..
- Motion parallax



# Motion parallax

A monocular depth cue that causes objects that are **closer** to appear to move **faster** than objects that are **further** away.




# Lucas-Kanade

Estimate optical flow between frame1 and frame2.

Assumptions:

- Constant brightness
- Small motion between frames
- Constant motion within a small windows

# Lucas-Kanade

Idea: For a small neighborhood (e.g. a 5x5 window), assume all pixels share the same motion ($u$,$v$), and minimize the error function:

$$E(u,v) = \sum_{x,y} w(x,y)[I_2(x+u, y+v) - I_1(x,y)]^2$$

Using Taylor approximation:

$$= \sum w(x,y)[I_2(x,y) + \frac{\partial I_2}{\partial x}u + \frac{\partial I_2}{\partial y}v - I_1(x,y)]^2$$

# Lucas-Kanade

$$= \sum w(x,y)[I_2(x,y) + \frac{\partial I_2}{\partial x}u + \frac{\partial I_2}{\partial y}v - I_1(x,y)]^2$$

Mark:

$I_x$ : The *x* derivative of image $I_2$

$I_y$ : The *y* derivative of image $I_2$

$I_t$ : The image difference $I_2 - I_1$

$$= \sum_{x,y} w(x,y)[u + I_y v + I_t]^2$$

# Lucas-Kanade

Find the (u,v) that minimizes the error function E by setting the derivatives to zero:

$$\frac{\partial E}{\partial u} = \sum_{x,y} I_x \cdot (I_x \cdot u + I_y \cdot v + I_t) = 0$$

$$\frac{\partial E}{\partial v} = \sum_{x,y} I_y \cdot (I_x \cdot u + I_y \cdot v + I_t) = 0$$

$$\begin{bmatrix} \sum_{x,y} I_x \cdot I_x & \sum_{x,y} I_x \cdot I_y \\ \sum_{x,y} I_y \cdot I_x & \sum_{x,y} I_y \cdot I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum_{x,y} I_x \cdot I_t \\ \sum_{x,y} I_y \cdot I_t \end{bmatrix}$$

# Iterative Lucas-Kanade

For larger (u,v) motions:

- Compute image derivatives $I_x$, $I_y$. Set *u,v* to 0.
- Compute once $A = \begin{bmatrix} \sum I_x \cdot I_x & \sum I_x \cdot I_y \\ \sum I_y \cdot I_x & \sum I_y \cdot I_y \end{bmatrix}$
- Iterate until convergence *($I_t \approx 0$):*
  - compute $b = \begin{bmatrix} \sum I_x \cdot I_t \\ \sum I_y \cdot I_t \end{bmatrix}, I_t(x,y) = I_2(x,y) - I_1(x+u, y+v)$
  - Solve equations to compute residual motion

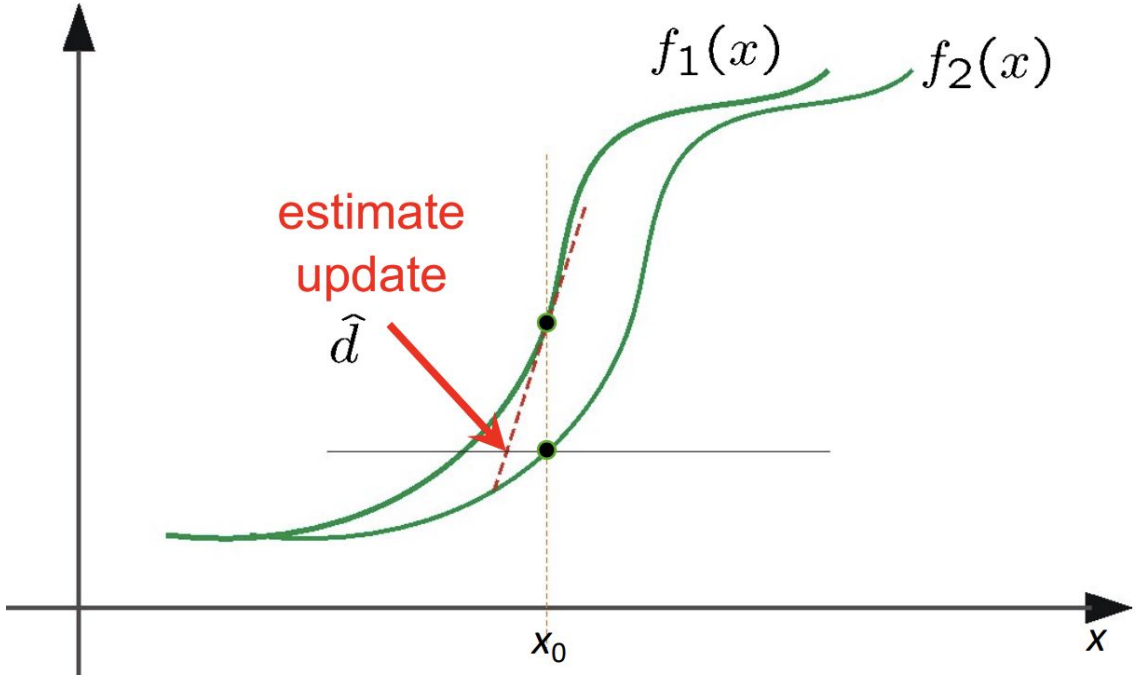
$$A \cdot \begin{bmatrix} du \\ dv \end{bmatrix} = -b$$

  - Update total motion with residual motion: *u+=du*, *v+=dv*
  - **Warp** $I_2$ towards $I_1$ with total motion *(u,v)*.

- Note: Warping of one image towards the other is done from **original image** using **total motion**, and not from previous image using residual motion. (Repetitive warping blurs!)

# Iterative Lucas-Kanade

Example:




Initial: $u_0=0$

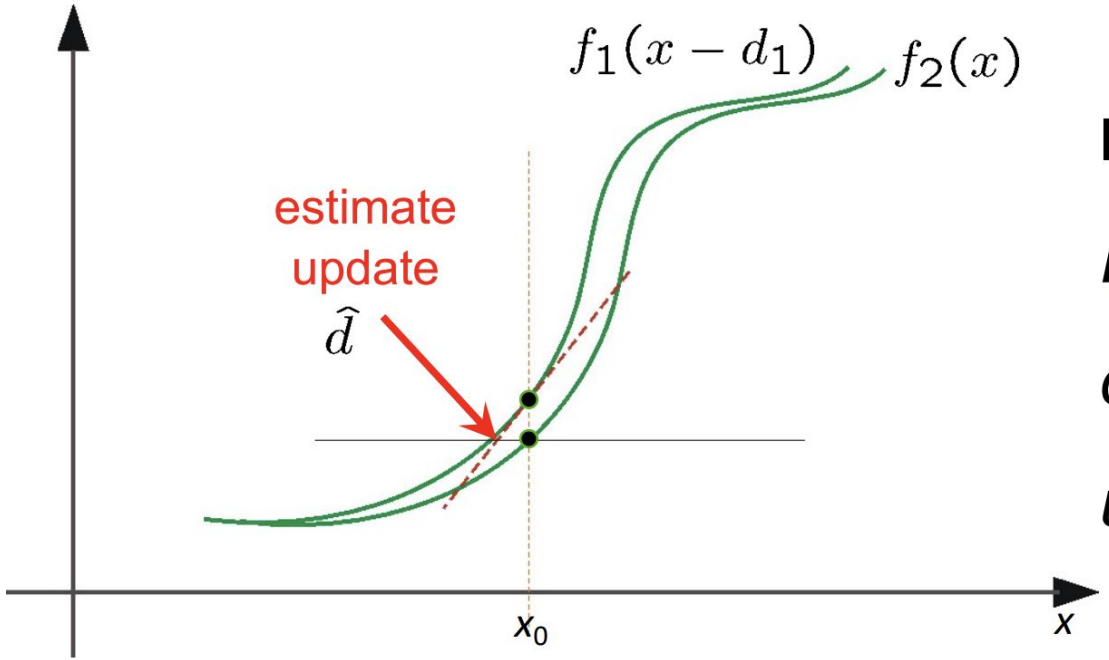
$I_t=f_1(x_0)-f_2(x_0)$

$du=I_t / f'_1(x_0)$

$u_1=u_0+du$

# Iterative Lucas-Kanade

Example:




Initial: $u_1$

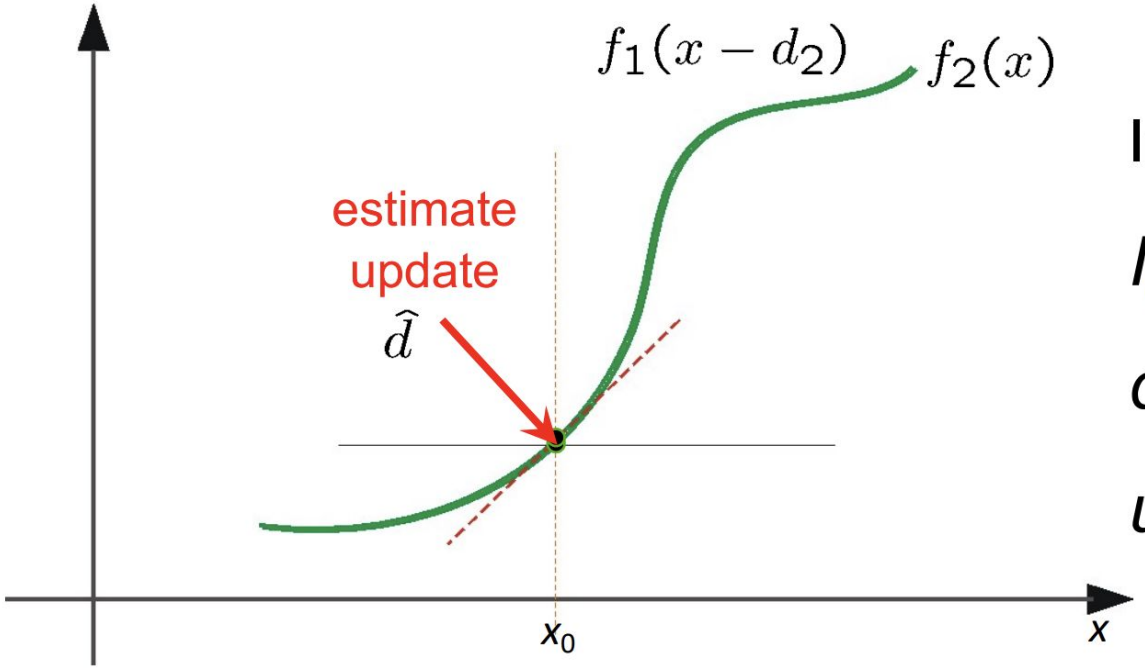
$I_t = f_1(x_0 - u_1) - f_2(x_0)$

$du = I_t / f'_1(x_0 - u_1)$

$u_2 = u_1 + du$

# Iterative Lucas-Kanade

Example:




Initial: $u_2$

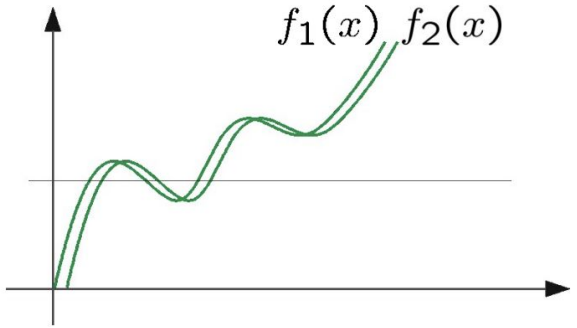
$I_t=f_1(x_0-u_2)-f_2(x_0)$

$du=I_t / f'_1(x_0-u_2)$

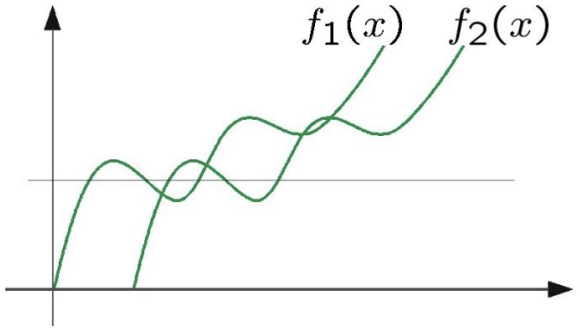
$u_3=u_2+du$

# Lucas-Kanade

Lucas-Kanade assumes that corresponding pixels in the two images have same derivative. It works OK even if derivatives are similar. But this is incorrect for very large motions.



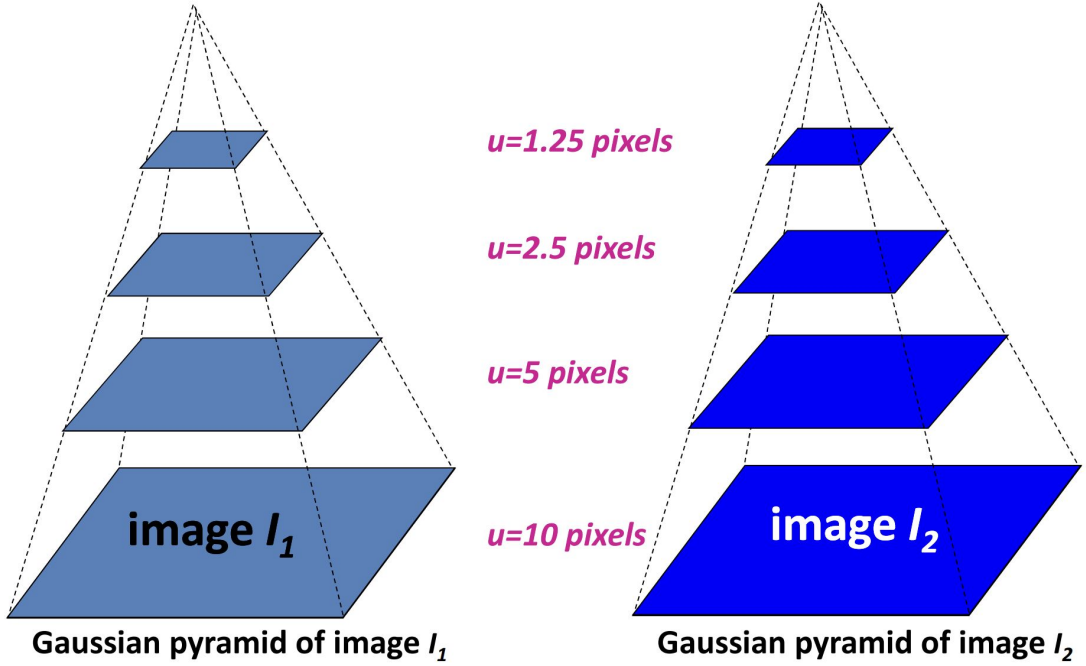

*Most areas have similar derivatives*




*Uncorrelated derivatives (opposite signs)*

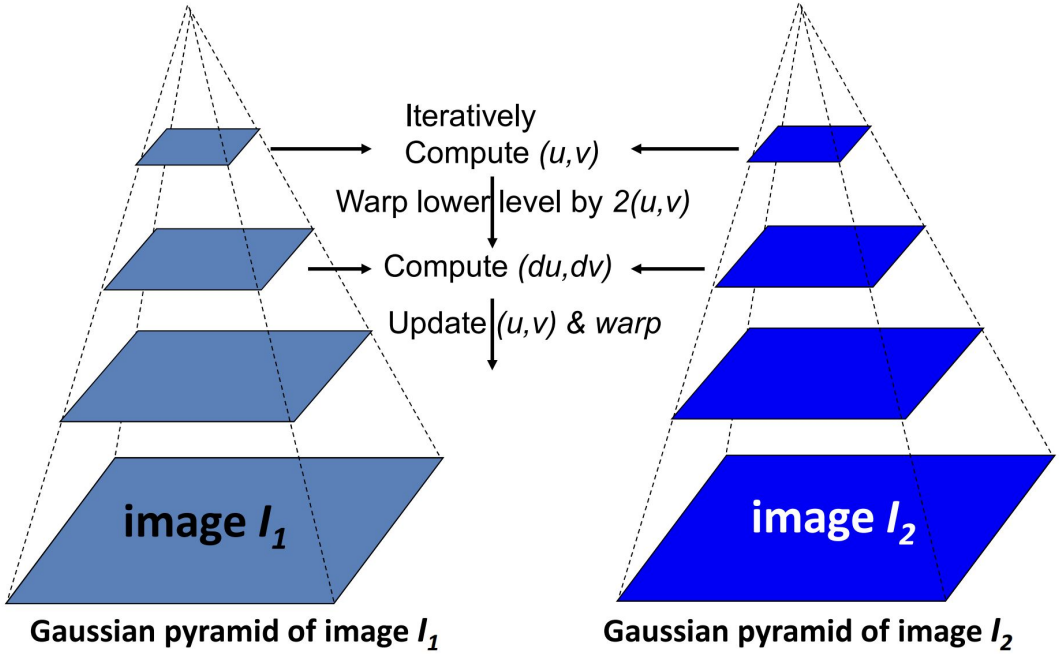
# Lucas-Kanade

**Solution:** use pyramid coarse-to-fine estimation.
estimate motion first at low resolution, then refine at higher resolutions.



# Lucas-Kanade

**Solution:** use pyramid coarse-to-fine estimation.
estimate motion first at low resolution, then refine at higher resolutions.



# Lucas-Kanade

**Pyramid algorithm:**

1. Build a Gaussian pyramid for both frames.
2. Start at the coarsest level (Images are small → motion appears small) and initialize flow $(u,v)$ = (0,0).
3. Estimate flow using Lucas–Kanade at this level.
4. Upsample the flow to the next finer level (scale by 2).
5. Warp the second image using the current flow estimate.
6. Refine the flow by running Lucas–Kanade on the residual motion.
7. Repeat until the original resolution is reached.

# Lucas-Kanade

Why it works:

- Large displacements become small at coarse scales
- Warping removes most motion, so LK’s small-motion assumption holds

Failure modes:

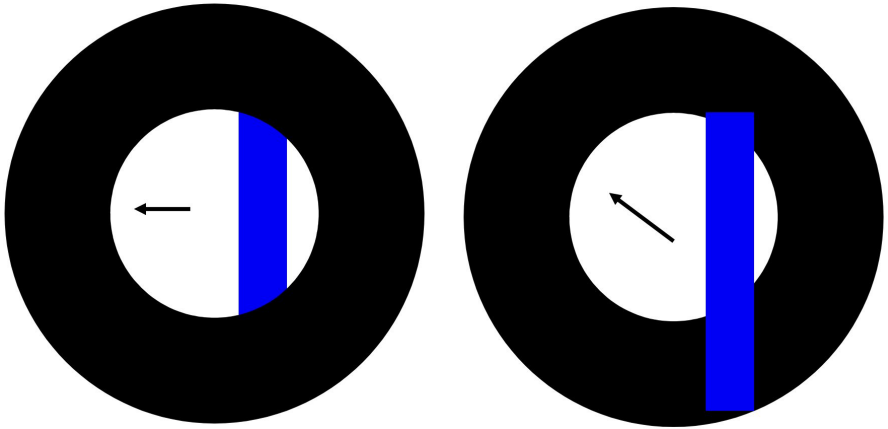
- Fast motion + motion blur
- Occlusions
- Illumination changes
- Motion parallax (depending on window size)
- Edges or flat regions → aperture problem

# The Aperture Problem

Motion perception becomes ambiguous when viewing a moving object through a small window (an "aperture").
We only see a small part of its edge, making its true velocity unclear.

- Motion along the edge direction cannot be determined
- Only motion perpendicular to the edge is observable

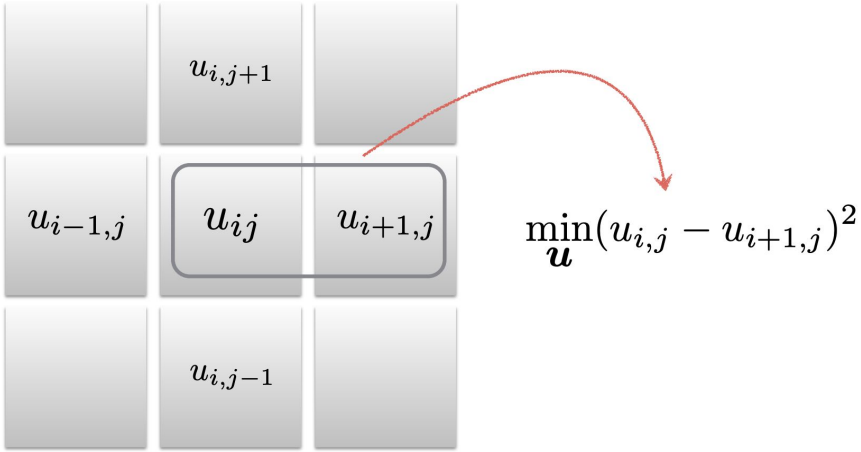
Examples: https://elvers.us/perception/aperture/



# Horn–Schunck

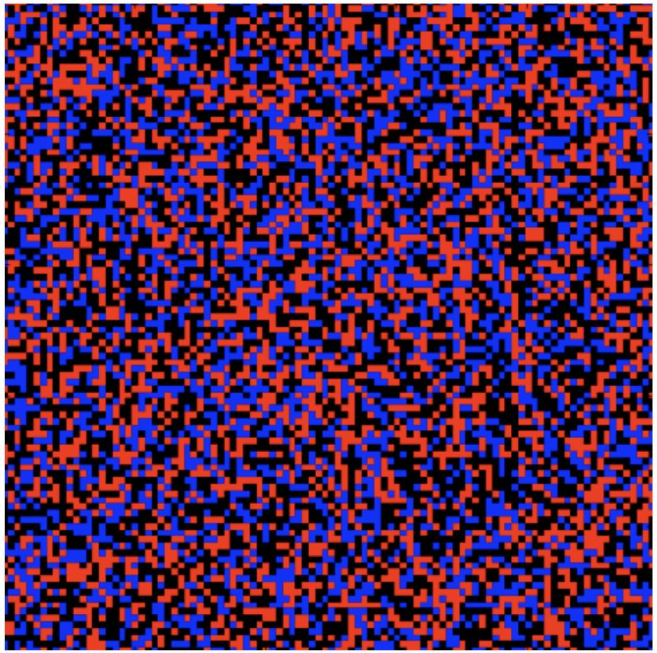
introduce a global smoothness constraint to solve the aperture problem.

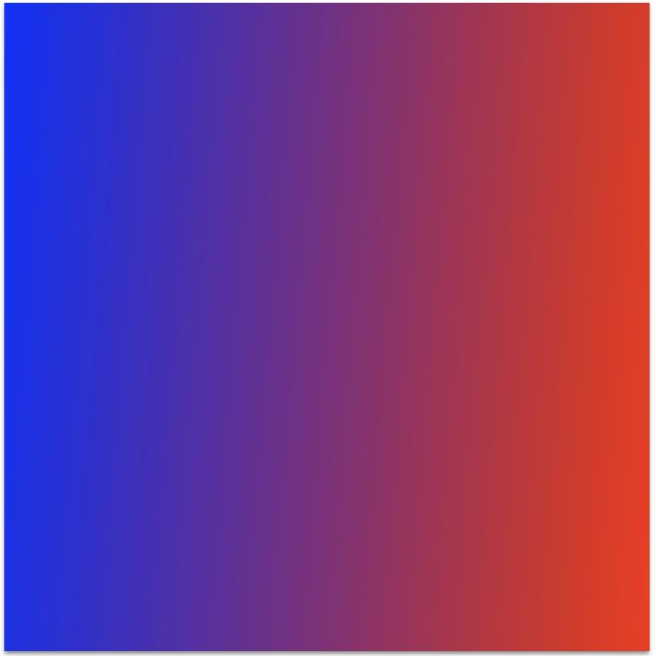
**Key idea:** assume the true motion field varies smoothly across the image as most natural motions, so ambiguous local motion can be inferred from neighboring pixels.



# Horn–Schunck

*Which flow field optimizes the objective?* $\min_{\boldsymbol{u}}(u_{i,j} - u_{i+1,j})^2$





# Horn–Schunck

$$\min_{\boldsymbol{u},\boldsymbol{v}} \sum_{i,j} \left\{ \overbrace{E_s(i,j)}^{\text{smoothness}} + \lambda \overbrace{E_d(i,j)}^{\text{brightness constancy}} \right\}$$

weight

$$\sum_{ij} \left\{ \frac{1}{4} \left[ (u_{ij} - u_{i+1,j})^2 + (u_{ij} - u_{i,j+1})^2 + (v_{ij} - v_{i+1,j})^2 + (v_{ij} - v_{i,j+1})^2 \right] + \lambda \left[ I_x u_{ij} + I_y v_{ij} + I_t \right]^2 \right\}$$

smoothness term

brightness constancy

New update rule:

`Initialize flow field`

$$\boldsymbol{u} = \mathbf{0}$$
$$\boldsymbol{v} = \mathbf{0}$$

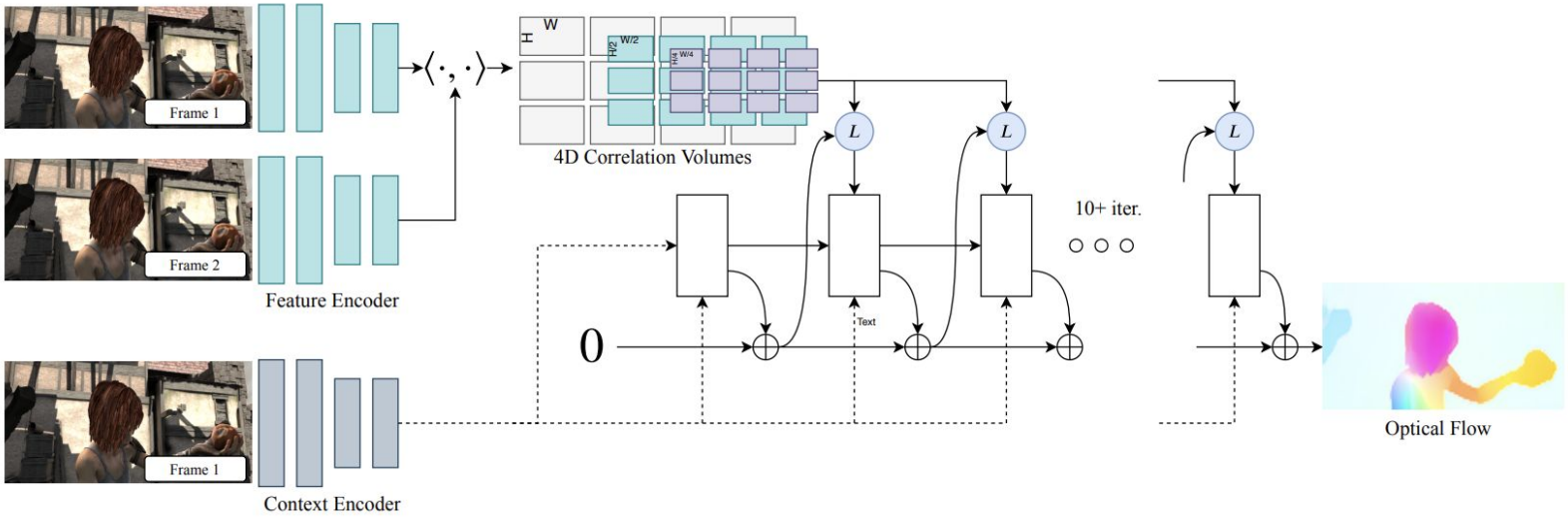
`While not converged`

`Compute flow field updates for each pixel:`

$$\hat{u}_{kl} = \bar{u}_{kl} - \frac{I_x \bar{u}_{kl} + I_y \bar{v}_{kl} + I_t}{\lambda^{-1} + I_x^2 + I_y^2} I_x \qquad \hat{v}_{kl} = \bar{v}_{kl} - \frac{I_x \bar{u}_{kl} + I_y \bar{v}_{kl} + I_t}{\lambda^{-1} + I_x^2 + I_y^2} I_y$$

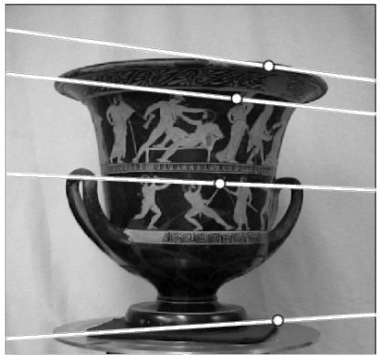
# Today – learning-based Optical Flow

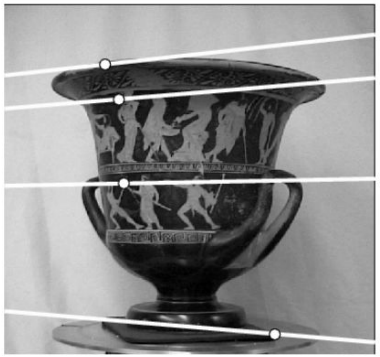
RAFT (Recurrent All-Pairs Field Transforms) – Teed & Deng, ECCV 2020.

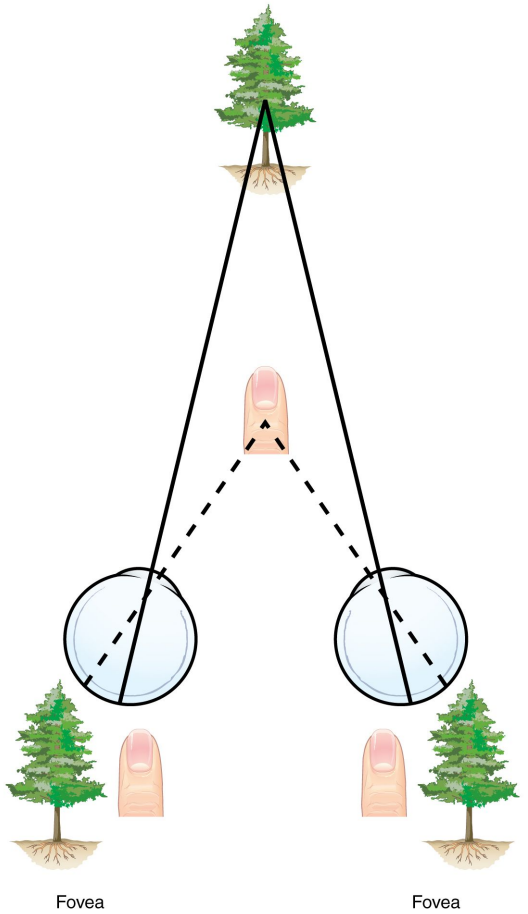


# Stereo Vision

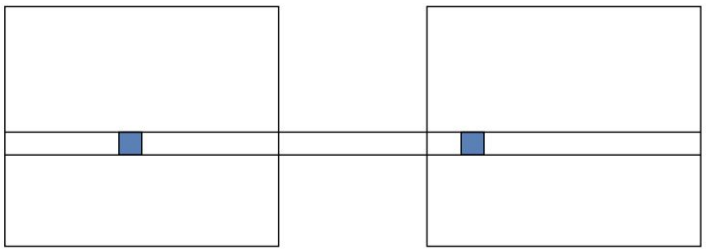
Stereo vision mimics human binocular vision, using two cameras to capture slightly different views of a scene, to calculate depth and create 3D perception.





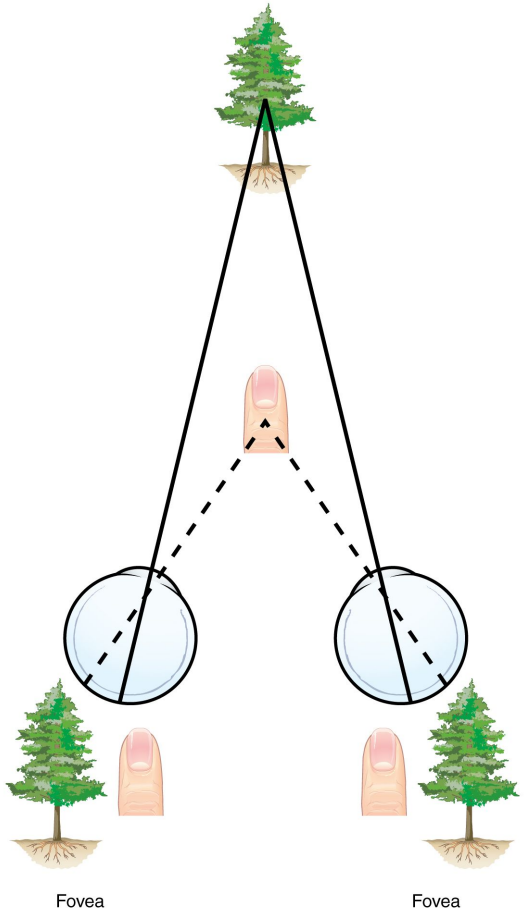


# Disparity




**disparity:** the difference in image location of the same 3D point when viewed by two different cameras.

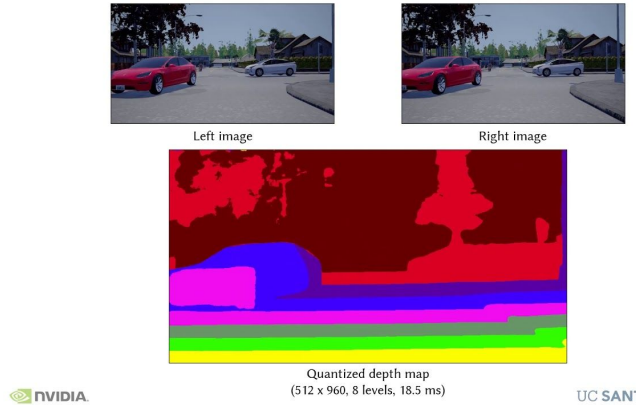
$$d = x_l - x_r$$

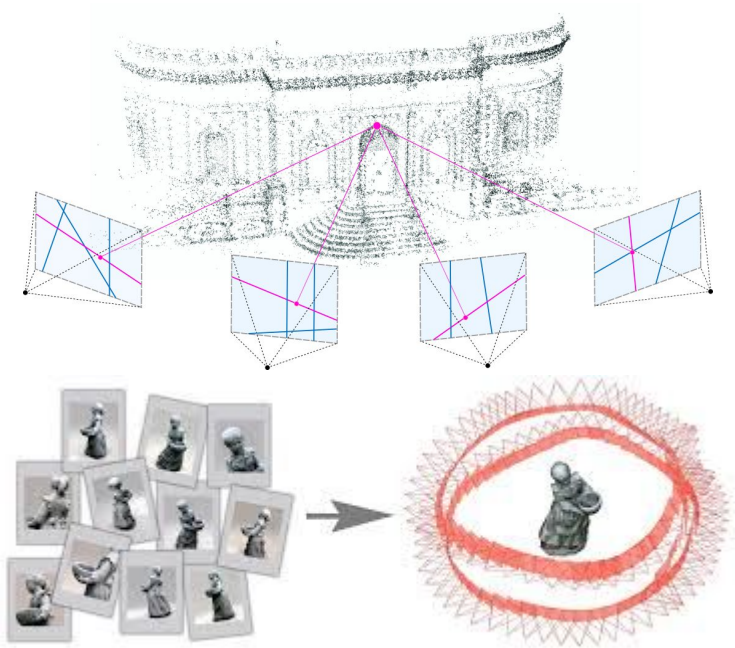


# Stereo Vision

**Applications:**

- Depth estimation
- 3D reconstruction (SfM)
- Autonomous driving
- Robotics & manipulation
- AR / VR
- …



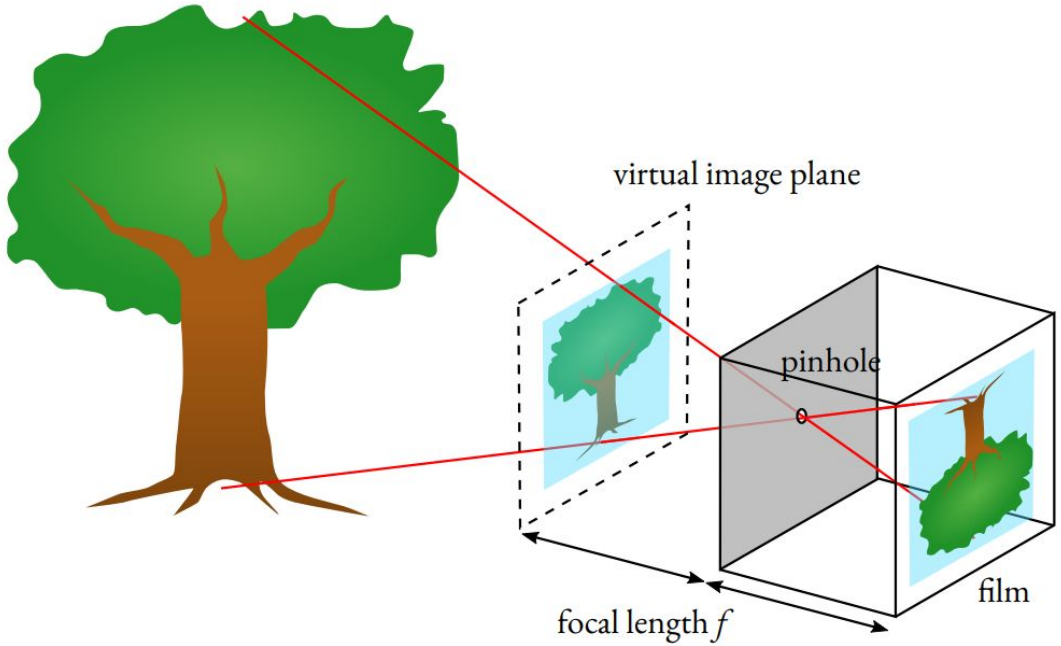
# Stereo Vision

- Viewing angles and object distance from the camera changes the way we perceive the objects in an image.

- A single view is an illusion!

# The pinhole camera model

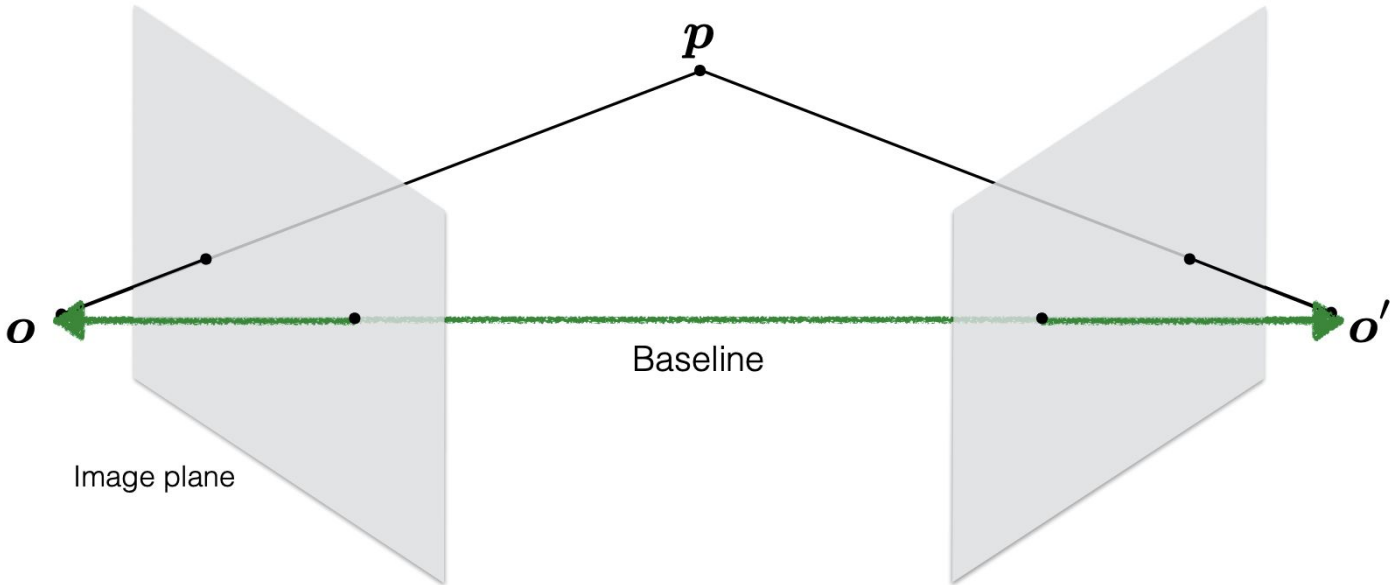
- All rays pass through a single point - the Center of Projection (COP) which is the pinhole.
- The image is formed on the image plane, and the focal length $f$ is the distance from the COP to the image plane.




Credit: Hoàng-Ân Lê

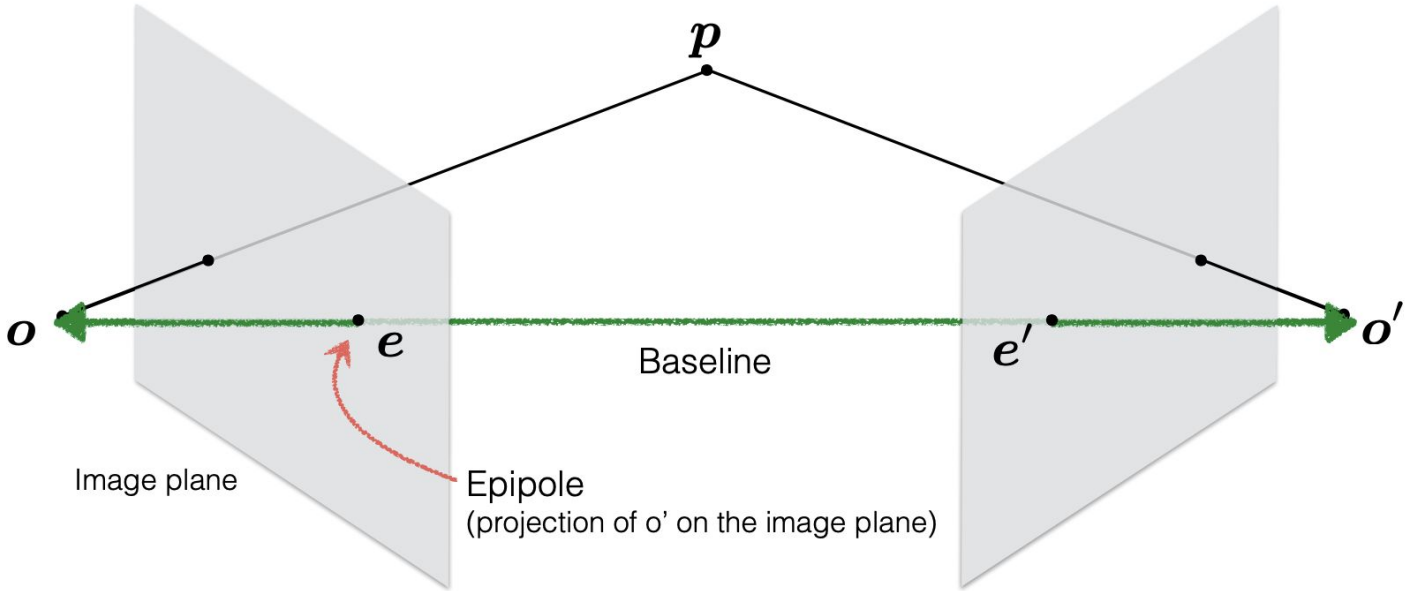
# Epipolar Geometry

Epipolar geometry describes the geometric relationship between two camera views of the same 3D scene.



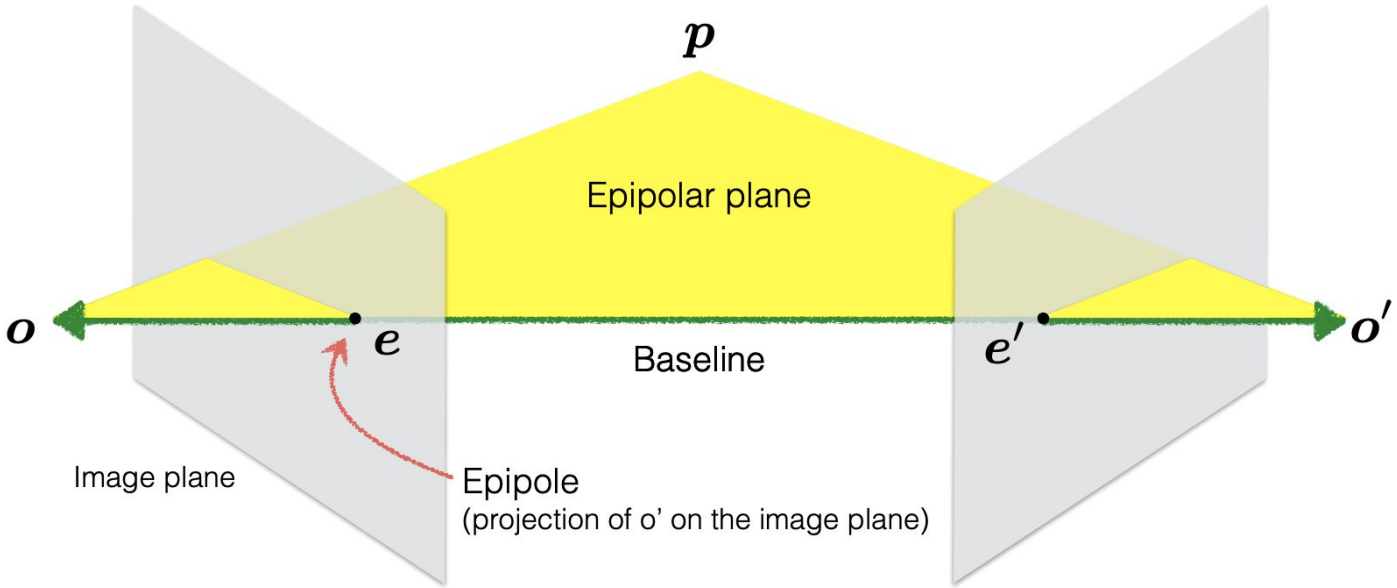
# Epipolar Geometry

Epipole = projection of each camera center (cop) onto the other image.
The epipoles lie on the baseline connecting the 2 cops.



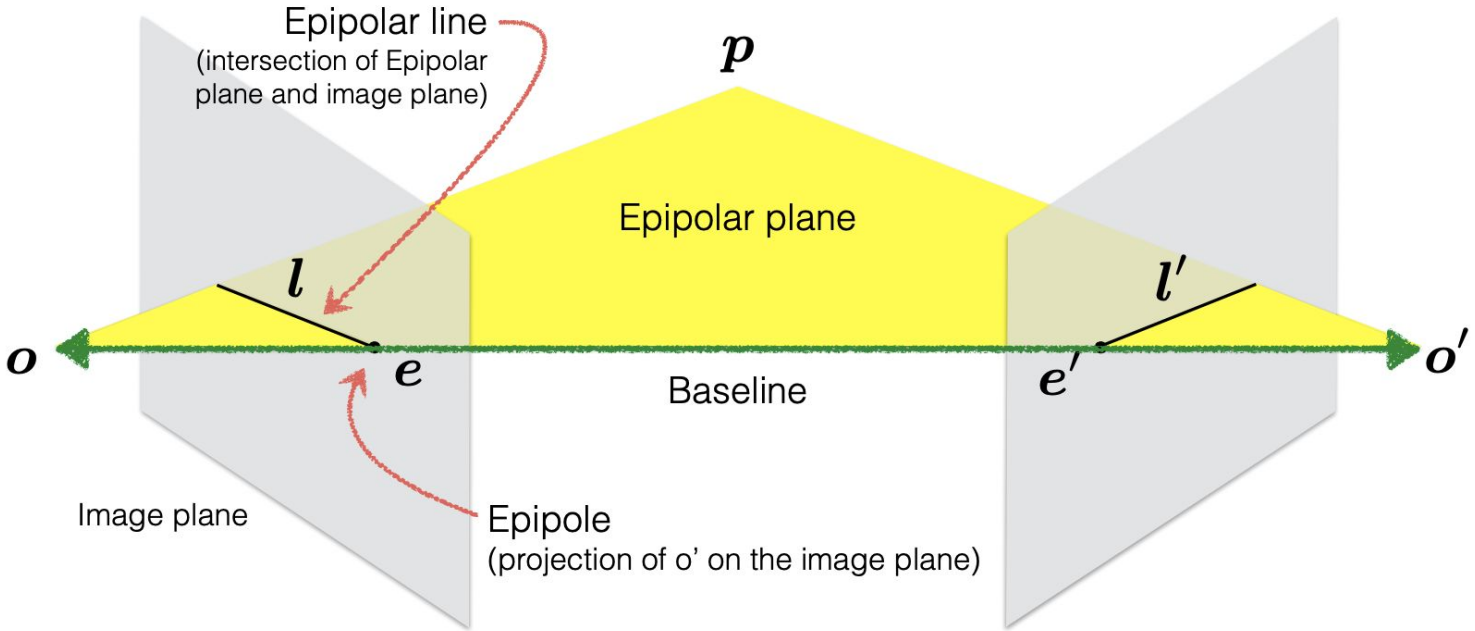
# Epipolar Geometry

The epipolar plane is the 3D plane defined by a 3D point and the 2 cops.
This plane contains all possible rays for that 3D point in both views.



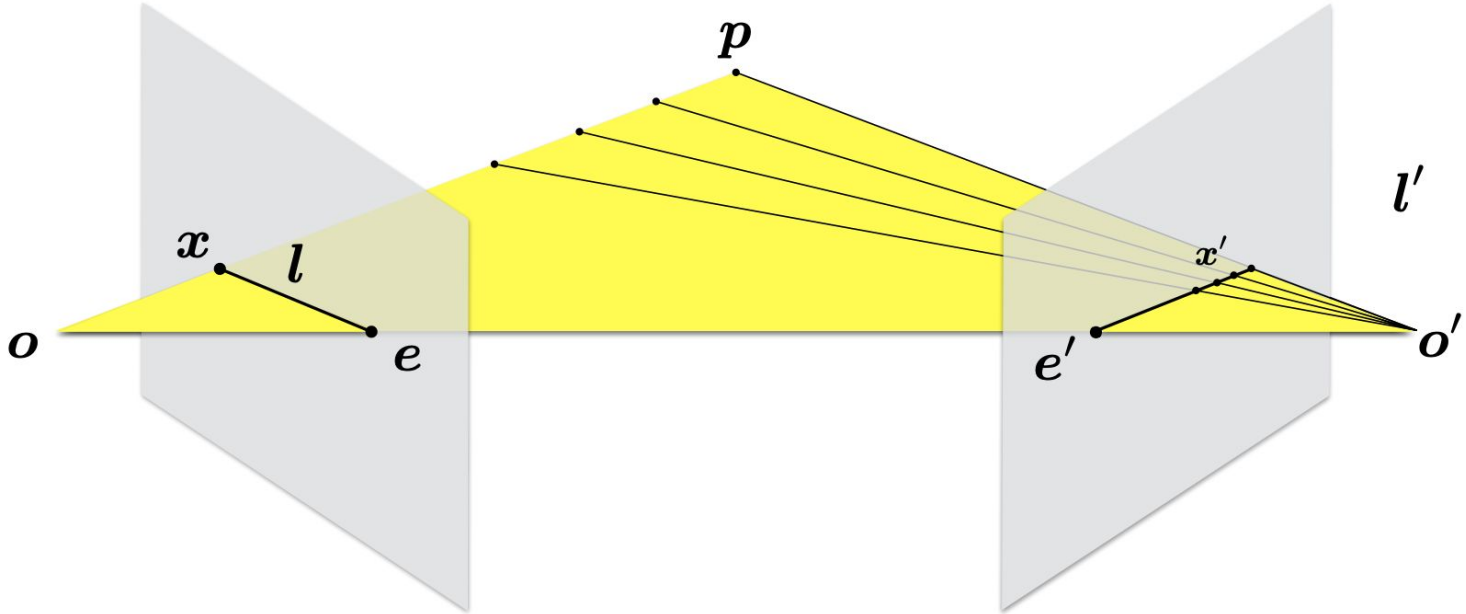
# Epipolar Geometry

The epipolar plane intersects each image plane in an epipolar line.

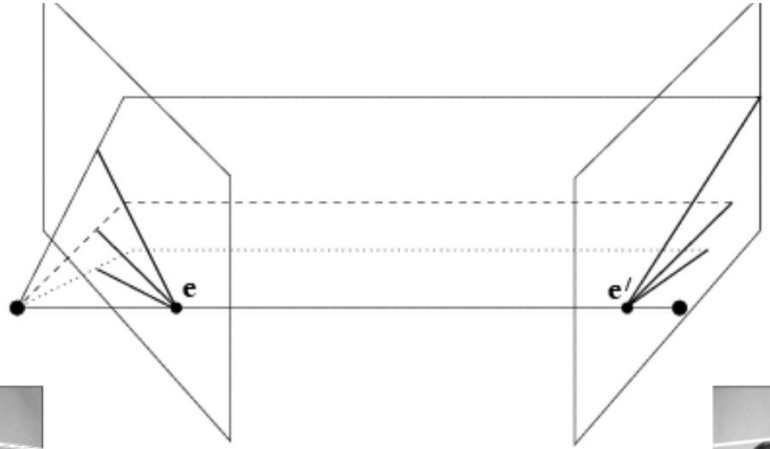


# Epipolar Geometry

x could correspond to each of the points on the ray between p and x.
The projection of these points would be on the other epipolar line $\ell$'.

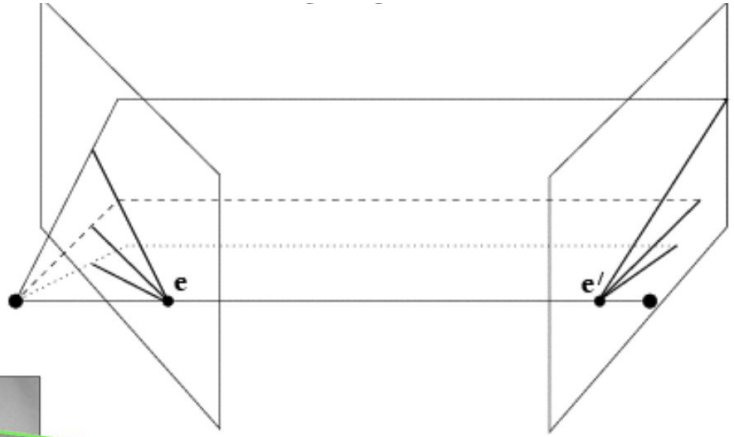


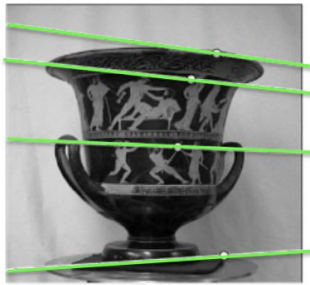
# Epipolar Geometry




*Where is the epipole in this image?*

# Epipolar Geometry



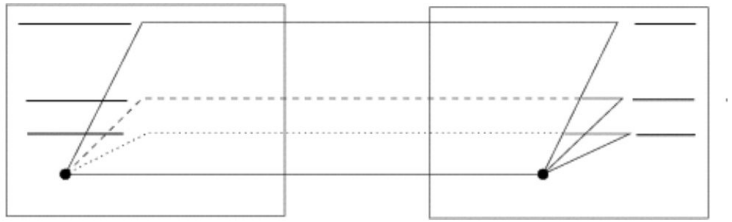
here!

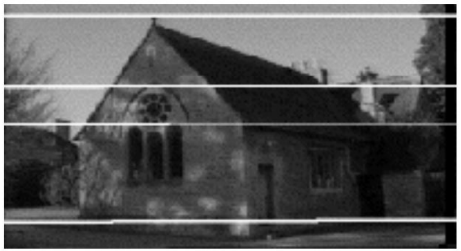
*Where is the epipole in this image?*

It's not always in the image

# Epipolar Geometry

Parallel cameras

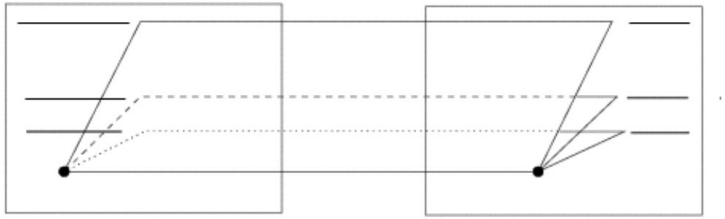


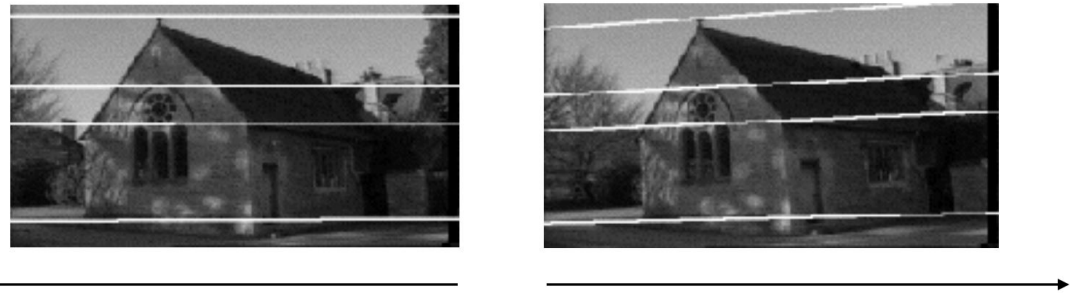


*Where is the epipole?*

# Epipolar Geometry

Parallel cameras





epipole at infinity

# Epipolar Geometry

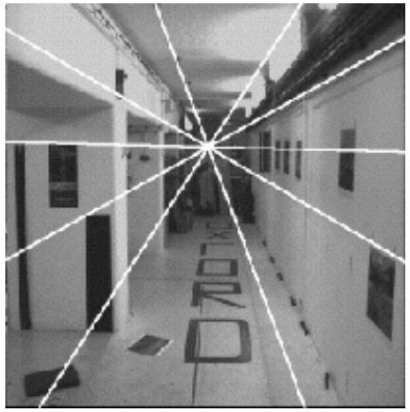
Forward moving camera

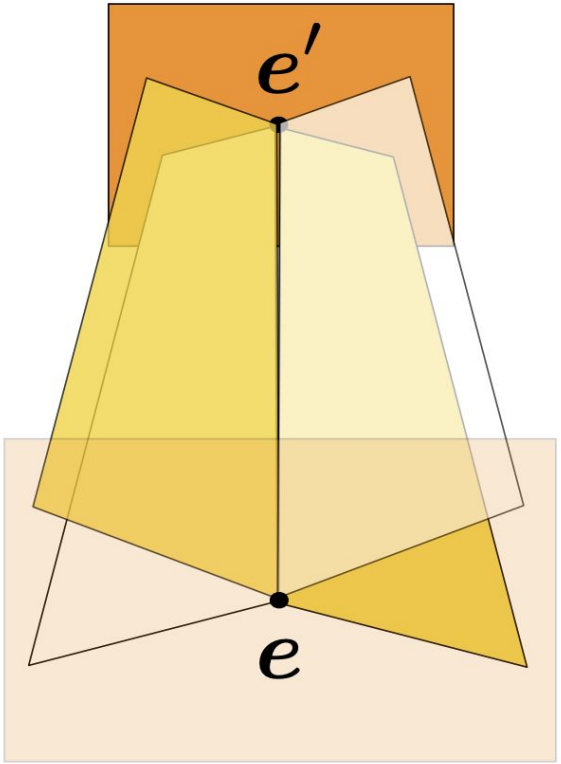


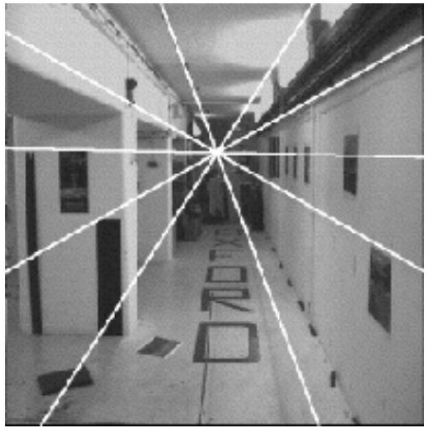
*Where is the epipole?*

# Epipolar Geometry

Forward moving camera

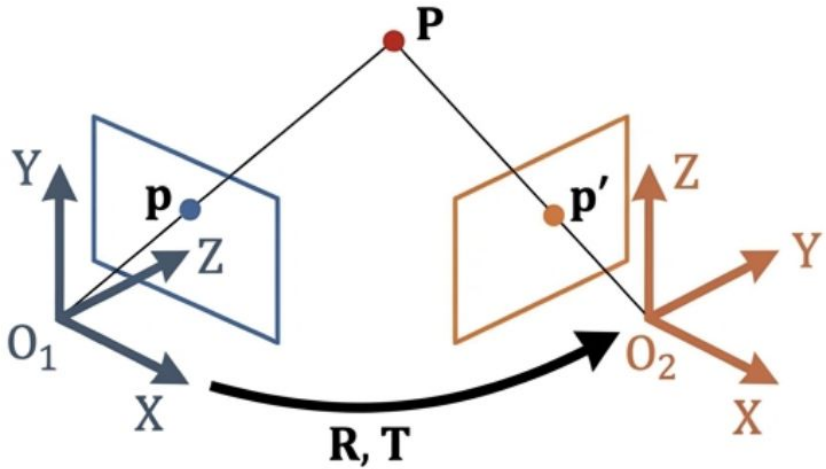




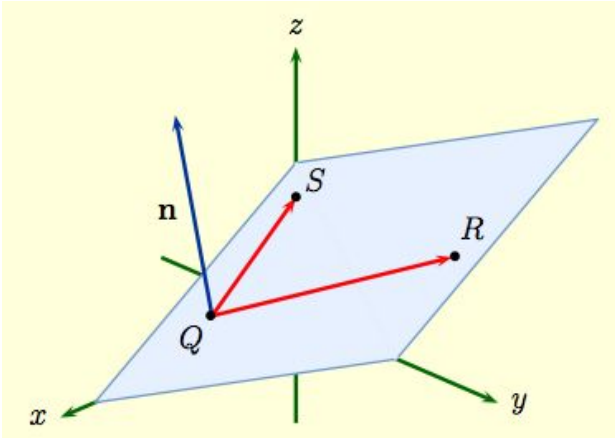
# The Essential Matrix

- The Essential Matrix is a 3 x 3 matrix that encodes epipolar geometry.
- Given a point p in one image, multiplying by the essential matrix yields the epipolar line in the second view.
- It is defined by the translation and rotation between the cameras (E = [t]xR).



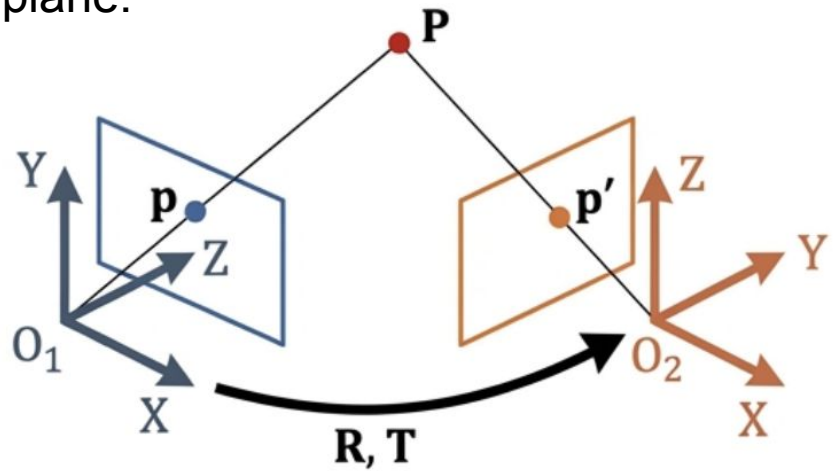
# Reminders

- Given two non-collinear vectors *QS*, *QR* that span a 3D plane, a normal vector to that plane is given by n = *QS* x *QR*.
- n is perpendicular to both vectors *QS*, *QR*.
- Any vector in the plane is a linear combination of *QS* and *QR*. Therefore, n is perpendicular to any other vector in the plane.



# The Essential Matrix

- The 3D point P, the two camera centers, and the image rays between them, all lie in the same epipolar plane.
- The translation vector t from O1→O2, and the viewing ray Rp (direction of P as seen from O2, expressed in O1's frame) both lie in this plane and span it.
- => n = t x Rp is the normal to the epipolar plane.
- Since the ray from O2 to p' lies on the same plane, it is orthogonal to n.
- => p' (t x Rp) = p'Ep = 0



# Essential Matrix vs Homography

*What's the difference between the essential matrix and a homography?*

They are both 3 x 3 matrices but …

$$\boldsymbol{l}' = \mathbf{E}\boldsymbol{x}$$

Essential matrix maps a **point** to a **line**

$$\boldsymbol{x}' = \mathbf{H}\boldsymbol{x}$$

Homography maps a **point** to a **point**

# The Fundamental Matrix

- What if cameras are uncalibrated?
- The Essential matrix actually operates on image points expressed in normalized coordinates (points have been aligned (normalized) to camera coordinates)
- Rewriting the epipolar constraint again in terms of image coordinates:

$$\boldsymbol{x}'^{\top}\mathbf{K}'^{-\top}\mathbf{E}\mathbf{K}^{-1}\boldsymbol{x} = 0$$
$$\boldsymbol{x}'^{\top}(\mathbf{K}'^{-\top}\mathbf{E}\mathbf{K}^{-1})\boldsymbol{x} = 0$$
$$\boldsymbol{x}'^{\top}\mathbf{F}\boldsymbol{x} = 0$$

# The Fundamental Matrix

- The Fundamental matrix works for image coordinates $\boldsymbol{x}'^{\top}\mathbf{F}\boldsymbol{x} = 0$ and maps pixels to epipolar lines.

- It depends on both intrinsic and extrinsic parameters.

# The Fundamental Matrix

**Fundamental Matrix Estimation:**

Given a set of M image points correspondences: $\{\boldsymbol{x}_m, \boldsymbol{x}'_m\}$

Each point pair should satisfy: $\boldsymbol{x}'^{\top}_m \mathbf{F} \boldsymbol{x}_m = 0$

$$\begin{bmatrix} x'_m & y'_m & 1 \end{bmatrix} \begin{bmatrix} f_1 & f_2 & f_3 \\ f_4 & f_5 & f_6 \\ f_7 & f_8 & f_9 \end{bmatrix} \begin{bmatrix} x_m \\ y_m \\ 1 \end{bmatrix} = 0$$

Each correspondence gives **one** equation: (not two as in homographies!)

$$x_m x'_m f_1 + x_m y'_m f_2 + x_m f_3 + y_m x'_m f_4 + y_m y'_m f_5 + y_m f_6 + x_m f_7 + y'_m f_8 + f_9 = 0$$

# The Fundamental Matrix

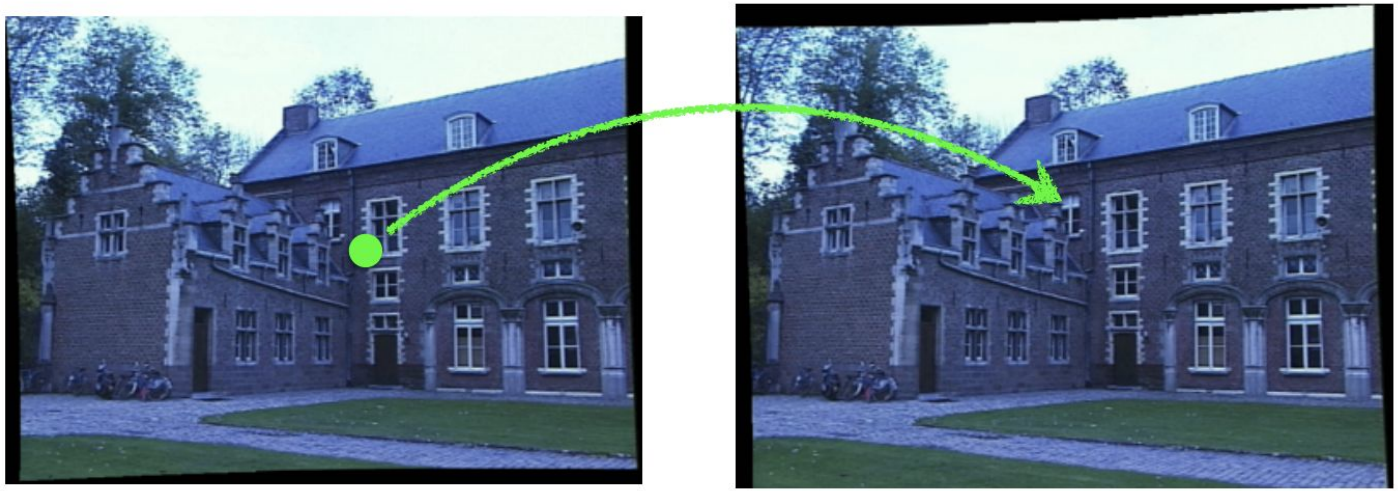
**Fundamental Matrix Estimation:** The 8-point algorithm.

0. (Normalize points)
1. Construct the M x 9 matrix **A**
2. Find the SVD of $\mathbf{A}^T\mathbf{A}$
3. Entries of **F** are the elements of column of **V** corresponding to the least singular value
4. (Enforce rank 2 constraint on F)
5. (Un-normalize F)

**+ RANSAC**

# Stereo Vision

**Task:** Match point in left image to point in right image



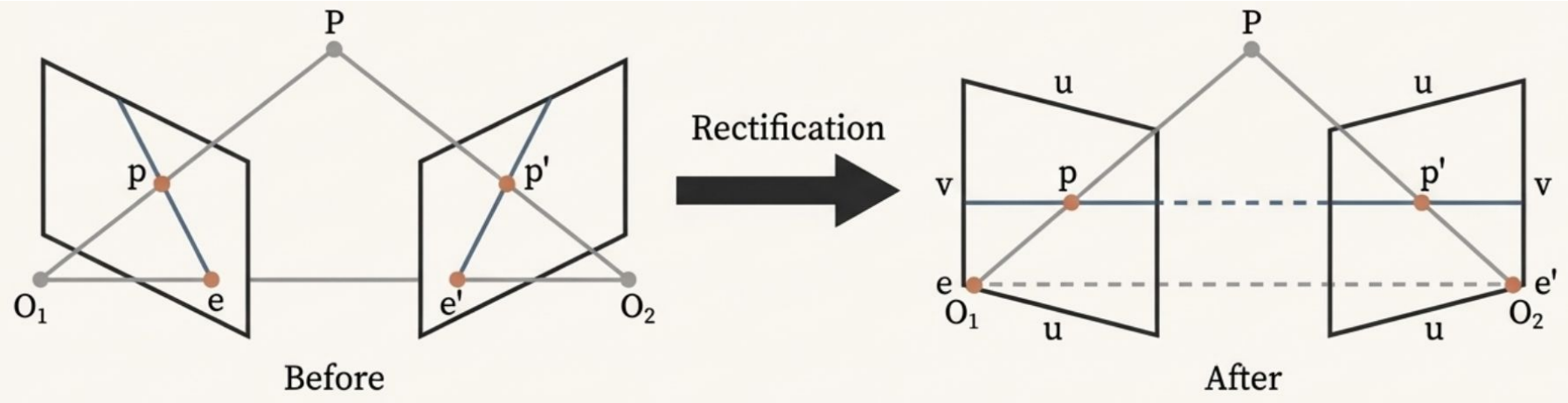
Left image

Right image

Want to avoid searching the whole image.. What can we do?

All matches lie on the epipolar line - only look there!

# Image Rectification




Using the fundamental matrix F, we can simplify stereo matching.

In a rectified image pair:

- All epipolar lines are horizontal
- Corresponding points lie on the same horizontal scanline

# Image Rectification

**Why not just use Homographies?**

Stereo vision estimates depth by observing the same 3D scene from two different viewpoints. Different depths produce different image displacements (disparity).

A Homography assumes that all points lie on a single plane, or camera motion is pure rotation.

Under those conditions, every point undergoes the **same** 2D transformation and one homography maps all points correctly. This isn't the case with **motion parallax**.

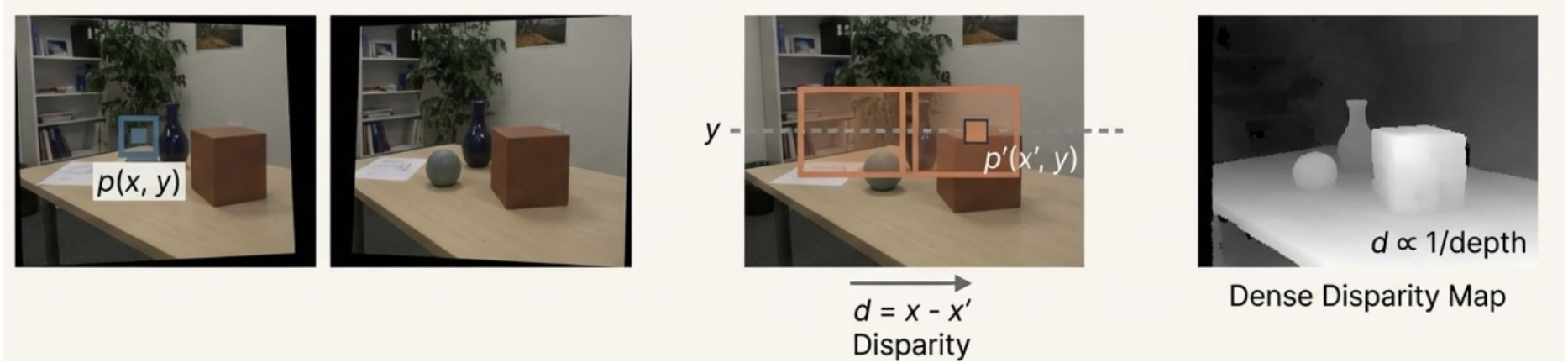
# Image Rectification

**Why stereo works?**

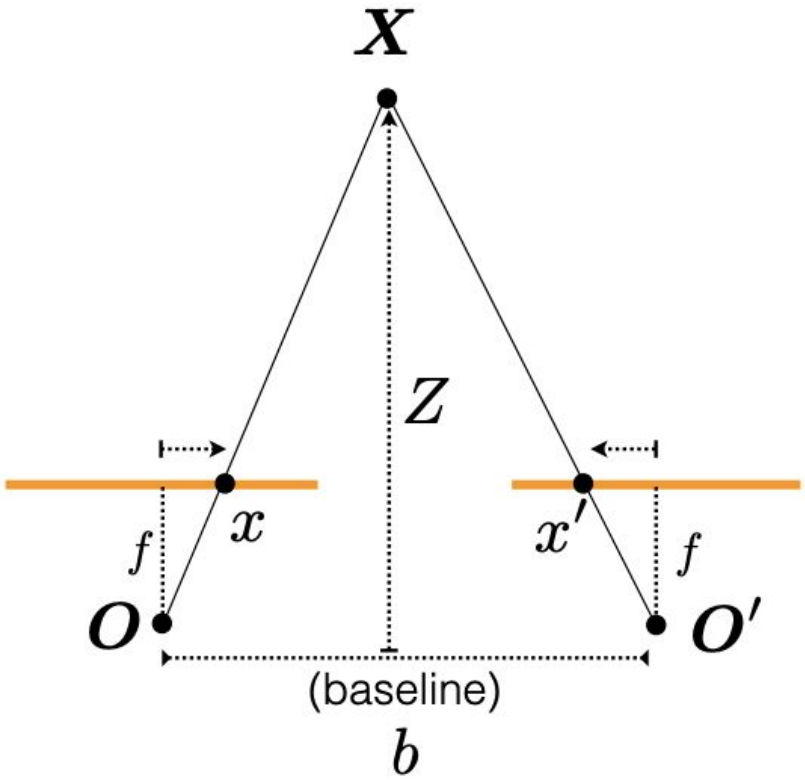
- The Fundamental Matrix preserves parallax instead of removing it.
- Two cameras with translation observe a 3D scene → points at different depths shift differently between views (parallax).
- The Fundamental Matrix does not align images; it encodes a constraint: a point in image 1 must match somewhere along the specific epipolar line in image 2.
- By measuring where along the epipolar line the match lies (disparity), we infer **depth**.

# From rectified images to depth map

- Once images are rectified, we can find a correspondence for every pixel.
- The disparity is the horizontal shift between pixels on the rectified images.
- It is inversely proportional to depth. Small disparity = object is far, large disparity = object is close.

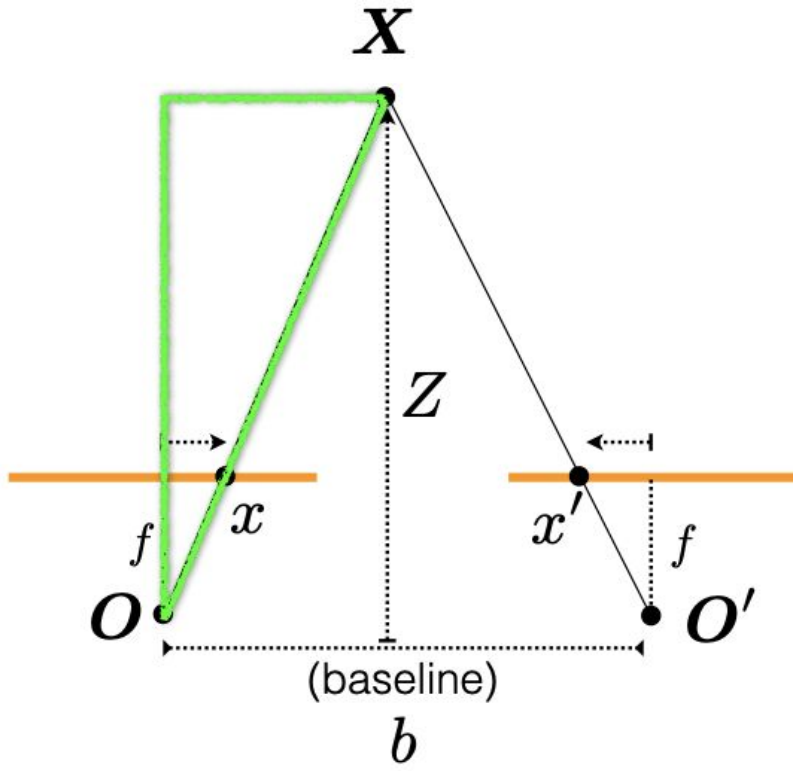


# From rectified images to depth map



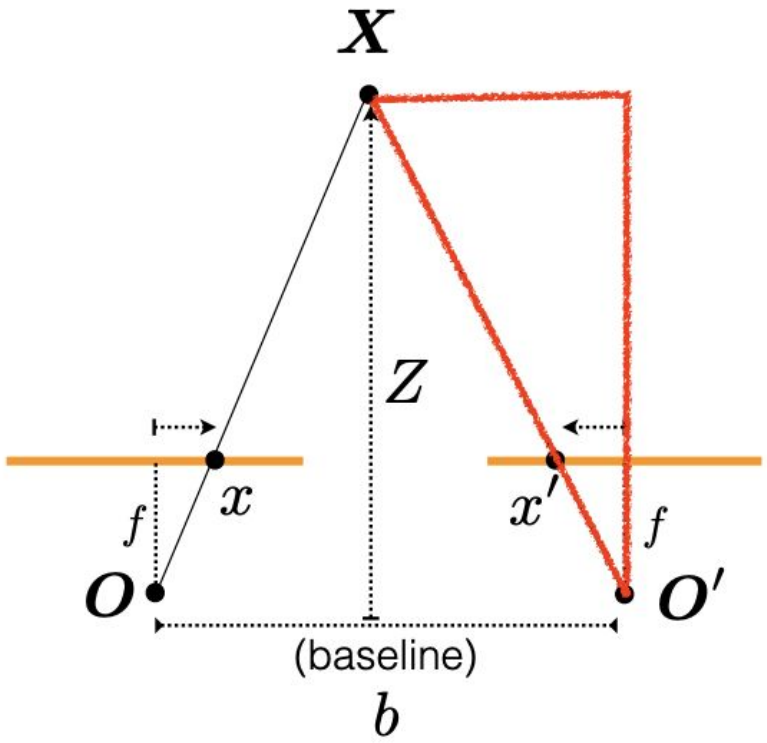
# From rectified images to depth map

$$\frac{X}{Z} = \frac{x}{f}$$



# From rectified images to depth map

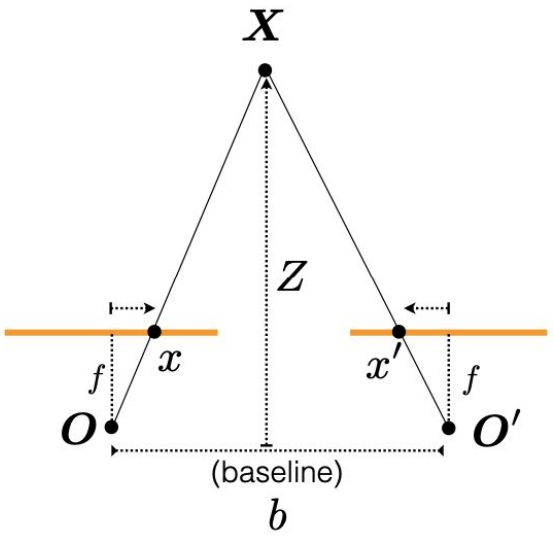
$$\frac{X}{Z} = \frac{x}{f}$$




$$\frac{b - X}{Z} = \frac{x'}{f}$$

# From rectified images to depth map

$$\frac{X}{Z} = \frac{x}{f}$$




$$\frac{b - X}{Z} = \frac{x'}{f}$$

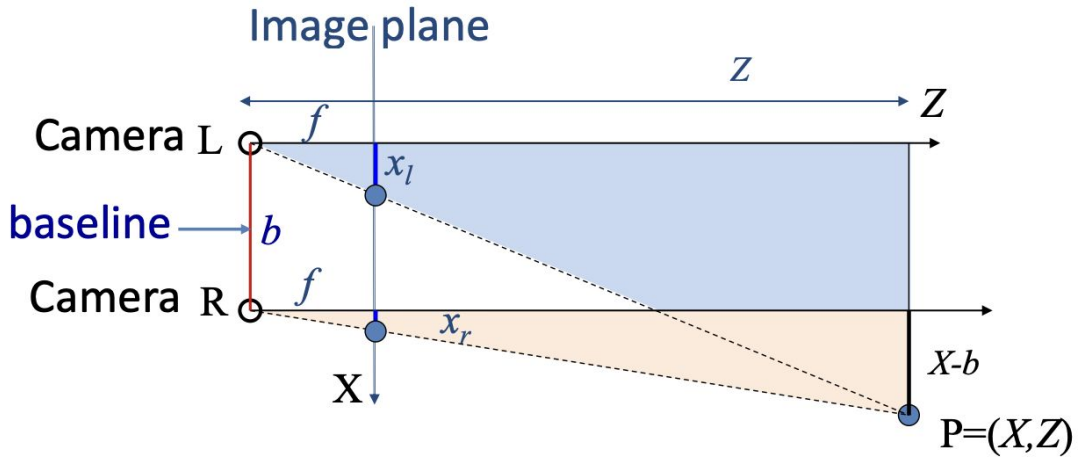
**Disparity**

$$d = x - x'$$

$$= \frac{bf}{Z}$$

inversely proportional to depth

# Depth from stereo




Similar triangles:

$$\frac{X}{Z} = \frac{x_l}{f} \qquad \frac{X-b}{Z} = \frac{x_r}{f} \qquad \Rightarrow \quad z = \frac{fb}{x_l - x_r} = \frac{fb}{d}$$

$$X = z\frac{x_l}{f}$$

# From rectified images to depth map

**Algorithm:**

- Find Sparse matches (e.g. with SIFT)
- From sparse matches estimate F (8-point + RANSAC)
- Compute rectifying homographies from F
- Rectify images (make epipolar lines horizontal)
- Perform dense stereo matching efficiently. For each pixel:
  - Find epipolar line
  - Scan line for best match
  - Compute depth from disparity: $Z = \frac{bf}{d}$